



# An Alternative to Certificate Authorities using Blockchain based Decentralized PKI



W.K.B.A.K Fernando

MASTER OF INFORMATION SECURITY  
UNIVERSITY OF COLOMBO SCHOOL OF COMPUTING

2020

---

## Declaration

I hereby declare that the thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis. This thesis has also not been submitted for any degree in any university previously.

Student Name: W.K.B.A.K Fernando

Registration Number: 2018MIS016

Index Number: 18770161

---

Signature of the Student & Date

This is to certify that this thesis is based on the work of Ms.W.K.B.A.K Fernando under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by,

Supervisor Name: Dr. Kasun De Zoysa

---

Signature of the Supervisor & Date

# Acknowledgments

First, I would like to convey my sincere gratitude to my project supervisor Dr.Kasun De Zoysa for giving me constant guidance and suggestions for this project and being available for discussions even when the university is closed down due to the pandemic situation.

Secondly, I would like to extend my gratitude to all the academic and administrative staff of University of Colombo School of Computing for the great assistance they have given to me in numerous ways.

I would also like to thank Alexandra Elbakyan for breaking down barriers in accessing research articles and making knowledge accessible to everyone in every corner of the world. Her website made it possible to access many resources that helped me with this project.

I am very much grateful to my family for their constant encouragement, support and patience through out this time.

Finally I extend my thanks to the “Hacker Who Hacked Me”. Without you, I would have never followed a Masters in Information Security hence thank you for paving this path for me.

# Abstract

A digital certificate is an electronic document which can be used to prove the ownership of a public key, which is a crucial aspect in web communication. These digital certificates are issued by a central governing body named Certificate Authorities. Activities of these certification authorities are not transparent and not audited as well.

The problem with certification authorities is that the internet community need to trust these certification authorities completely and currently we have placed immense trust in them. Due to the importance of the certification authorities they are tempting targets of criminals. There are many real-life examples for certification authorities being hacked or misbehaving. If the certification authority is compromised, it is possible to create fake digital certificates which can be used for malicious activities.

We have identified the reason for this issue is the fact that certification authorities can be considered as a single point of failure in public key infrastructure. Hence in this project we propose to eliminate this central figure and try to find out whether it is possible to decentralize the functionalities of a certification authorities by incorporating the blockchain technology.

In this project, we have followed an experimental research approach. Our prototype implementation “TLSCchain” is based on Ethereum blockchain network and integrated with novel on-chain domain verification. The test cases carried out revealed that the blockchain technology is a well-suited platform to build a Public Key Infrastructure and possible to eliminate Certificate Authorities. After evaluating the monetary side of “TLSCchain”, it was evident the design should address the scalability aspect due to the volatile nature of the cypto currency market.

# Contents

<b>Acknowledgments</b>	<b>ii</b>
<b>Abstract</b>	<b>iii</b>
<b>Contents</b>	<b>iv</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xiii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Statement of the problem . . . . .	4
1.3 Research Aims and Objectives . . . . .	4
1.3.1 Aims . . . . .	4
1.3.2 Objectives . . . . .	5
1.3.3 Scope . . . . .	6
1.3.4 Structure of the Thesis . . . . .	6
<b>2 Literature Review</b>	<b>8</b>
2.1 Distributed Ledger Technology . . . . .	8
2.2 Blockchain . . . . .	8
2.2.1 Blockchain Characteristics . . . . .	9
2.2.2 Blockchain Categorization . . . . .	10
2.2.3 Bitcoin and Ethereum . . . . .	13
2.2.3.1 Bitcoin . . . . .	13

## CONTENTS

---

2.2.3.2	Ethereum . . . . .	13
2.2.4	Smart Contracts . . . . .	14
2.2.5	Dapps (Decentralized application) . . . . .	15
2.2.5.1	Characteristics of Dapps . . . . .	15
2.2.5.2	Benefits of Dapp Development . . . . .	15
2.2.6	Blockchain Oracles (Blockchain Middleware) . . . . .	16
2.2.6.1	The need of Oracles . . . . .	16
2.2.6.2	Chainlink . . . . .	16
2.3	Public Key Infrastructure . . . . .	17
2.3.1	PKI supported functions . . . . .	18
2.3.2	PKI Trust Models . . . . .	18
2.3.2.1	Certificate Authorities . . . . .	19
2.3.2.2	Web of Trust . . . . .	19
2.3.2.3	Log based PKI . . . . .	19
2.3.3	Certificate Authority Functionality . . . . .	20
2.3.4	Certificate Authority Certificate Issuing Steps . . . . .	20
2.3.4.1	Certificate Issuance Process . . . . .	20
2.3.4.2	Revoking Certificates . . . . .	22
2.3.5	X.509 Certificates . . . . .	23
2.4	Current Applications . . . . .	24
2.4.1	Decentralized Public Key Infrastructure . . . . .	24
2.4.1.1	Web of Trust . . . . .	25
2.4.1.2	Log Based PKI . . . . .	25
2.4.2	ACME Protocol . . . . .	25
2.4.3	Let's Encrypt . . . . .	26
2.4.3.1	Certificate Management Agent Software . . . . .	26
2.4.3.2	Domain Validation . . . . .	26
2.4.3.3	Certification Issuance and Revocation . . . . .	27
2.4.4	Trust CA . . . . .	28
2.4.5	DeTRACT . . . . .	29
2.4.6	NameCoin . . . . .	29
2.4.7	Blockstack . . . . .	30
2.4.8	Certcoin . . . . .	30

2.4.9	SCPki -Smart Contract-based PKI and Identity system . . . . .	31
2.4.9.1	SCPki Improvements . . . . .	31
2.4.10	Analysis of the current applications . . . . .	31
2.4.11	Conclusion of the Review . . . . .	32
2.4.11.1	Gap . . . . .	32
2.4.11.2	Viable Direction . . . . .	33
<b>3</b>	<b>Research Methodology</b>	<b>34</b>
3.1	Knowledge gathered from previous researches . . . . .	35
3.2	Solving the identified problems . . . . .	36
3.3	Selecting the Blockchain Technology . . . . .	37
3.4	Selecting the Blockchain Network . . . . .	39
3.5	Our Approach . . . . .	40
<b>4</b>	<b>Research Design - TLSChain</b>	<b>42</b>
4.1	TLSChain Overview . . . . .	43
4.2	Main Components . . . . .	43
4.2.1	Domain Owner . . . . .	43
4.2.2	TLSChain CLI . . . . .	44
4.2.3	Chain Link . . . . .	44
4.2.4	TLSChain Smart Contract . . . . .	44
4.2.5	TLSChain Web Extension . . . . .	44
4.3	TLSChain CLI Functionalities . . . . .	45
4.3.1	Main functionalities of the TLSChain CLI . . . . .	45
4.4	Smart Contract Functionalities . . . . .	45
4.4.1	Domain public key registration . . . . .	45
4.4.1.1	Validation points . . . . .	45
4.4.2	Domain Public Key Revoke . . . . .	47
4.4.3	Domain Public Key Renewal . . . . .	47
4.4.4	Retrieve Validity . . . . .	47
<b>5</b>	<b>Implementation</b>	<b>49</b>
5.1	TLSChain Smart Contract . . . . .	49
5.1.1	Storing Domain Public Key Mapping . . . . .	49

## CONTENTS

---

5.1.1.1	Information Storage . . . . .	49
5.1.1.2	Struct Certificate . . . . .	50
5.1.1.3	Solidity Code . . . . .	50
5.1.2	Domain Registration Function . . . . .	50
5.1.2.1	requestRegister Function . . . . .	50
5.1.2.2	requestRegister Validations . . . . .	51
5.1.2.3	requestRegister Solidity Code . . . . .	52
5.1.3	Renew Domain Public Key Validity Function . . . . .	53
5.1.3.1	requestRenewDomainValidity Solidity Code . . . . .	53
5.1.4	Update Domain Public Key Function . . . . .	54
5.1.4.1	requestRenewDomainPubKey Solidity Code . . . . .	55
5.1.5	Revoke Domain Public Key Registration Function . . . . .	56
5.1.5.1	revoke Solidity Code . . . . .	56
5.1.6	Retrieve Domain Public Key Information Function . . . . .	56
5.1.6.1	retrieve Solidity Code . . . . .	57
5.2	TLSCChain CLI . . . . .	57
5.2.1	TLSCChain CLI Available Commands . . . . .	57
5.2.2	getPubHash Usage . . . . .	58
5.2.2.1	getPubHash scope . . . . .	59
5.2.2.2	getPubHash Parameters . . . . .	59
5.2.2.3	getPubHash Python Code . . . . .	59
5.2.3	registerDomain Usage . . . . .	59
5.2.3.1	registerDomain Scope . . . . .	59
5.2.3.2	registerDomain Pre-Configurations . . . . .	60
5.2.3.3	registerDomain Parameters . . . . .	60
5.2.3.4	registerDomain Python Code . . . . .	61
5.2.4	retriveValidityCer Usage . . . . .	61
5.2.4.1	retriveValidityCer scope . . . . .	61
5.2.4.2	retriveValidityCer Parameters . . . . .	62
5.2.4.3	retriveValidityCer Python Code . . . . .	62
5.2.5	retriveValidityHash Usage . . . . .	62
5.2.5.1	retriveValidityHash Scope . . . . .	62
5.2.5.2	retriveValidityHash Parameters . . . . .	62



5.2.5.3	retriveValidityHash Python Code . . . . .	63
5.2.6	revokeDomain Usage . . . . .	63
5.2.6.1	revokeDomain Scope . . . . .	63
5.2.6.2	revokeDomain Parameters . . . . .	63
5.2.6.3	revokeDomain Python Code . . . . .	63
5.2.7	renewDomainValidity Usage . . . . .	64
5.2.7.1	renewDomainValidity Scope . . . . .	64
5.2.7.2	renewDomainValidity Parameters . . . . .	64
5.2.7.3	renewDomainValidity Python Code . . . . .	65
5.2.8	requestRenewDomainPubKe Usage . . . . .	65
5.2.8.1	requestRenewDomainPubKe Scope . . . . .	65
5.2.8.2	requestRenewDomainPubKey Parameters . . . . .	65
5.2.8.3	requestRenewDomainPubKey Python Code . . . . .	66
5.3	TLSSChain Chainlink Usage . . . . .	66
5.3.1	TLSSChain ChainLink Job Spcification . . . . .	67
5.3.1.1	Job Specification Json . . . . .	67
5.3.1.2	Chainlink Job Reading Server Value . . . . .	68
5.4	TLSSChain Extension . . . . .	68
5.4.1	TLSSChain Extension Scope . . . . .	70
5.4.2	TLSSChain Java Script Code . . . . .	70
<b>6</b>	<b>TLSSChain - A Use Case</b>	<b>72</b>
6.1	TLSSChain Functional Flow . . . . .	72
6.1.1	Step 1 - Creaking a Self-Sign Certificate . . . . .	72
6.1.2	Step 2- Use TLSSChainCLI to get the Public Key Hash . . . . .	72
6.1.3	Step 3- Configure the Web Server . . . . .	74
6.1.4	Step 4- Register a Domain with TLSSChain . . . . .	76
6.1.5	Step 5- Verification . . . . .	77
6.1.5.1	retriveValidityCer . . . . .	77
6.1.5.2	retriveValidityHash . . . . .	79
<b>7</b>	<b>Evaluation</b>	<b>80</b>
7.1	Evaluation Setup . . . . .	80

7.2	Evaluation Process . . . . .	81
7.2.1	TLSCChain Achieving Certificate Issuing Process . . . . .	81
7.2.1.1	Test Case 1 - Possibility of registering a public key for a domain using TLSCChain . . . . .	81
7.2.1.2	Test Case 2 - An attacker trying to register a new public key for an already registered domain should not be successful . . . . .	82
7.2.1.3	Test Case 3 - Initial owner updating the registered public key information should be successful . . . . .	82
7.2.2	TLSCChain for Storage of Certificate Information . . . . .	85
7.2.2.1	Test Case 4 - Successful storage of public key information . . . . .	85
7.2.2.2	Test Case 5 - Successful retrieval of public key information . . . . .	85
7.2.3	Blockchain based PKI for certificate revocation . . . . .	86
7.2.3.1	Test Case 6 - Initial Owner is able to revoke the registered information . . . . .	86
7.2.3.2	Test Case 7 - Attacker should not able be able to revoke a registered information . . . . .	87
7.2.4	TLSCChain Achieve Certificate Verification Process . . . . .	87
7.2.4.1	Test Case 8 - When accessing a domain registered with TLSCChain, if the correct public key sent from the server it will be indicated . . . . .	87
7.2.4.2	Test Case 9 - When accessing a domain registered with TLSCChain, an incorrect public key is sent and it will be indicated . . . . .	88
7.2.4.3	Test Case 10 - When accessing a domain not registered with TLSCChain it will be indicated . . . . .	89
7.2.5	TLSCChain Efficiency . . . . .	91
7.2.5.1	Test Case 11 - Time taken for registering a public key for a domain . . . . .	91
7.2.5.2	Test Case 12 - Time taken for the validation in the client side . . . . .	92

7.2.5.3	Test Case 13 - Registration cost in TLSChain compared to the existing systems . . . . .	92
7.3	Security Analysis . . . . .	95
7.3.1	Test Case 14 - Preventing Man in the Middle Attack . . .	95
7.3.2	Test Case 15 - Preventing Replay Attacks . . . . .	96
7.3.3	Test Case 16 - Preventing Intruder trying to register a do- main that he does not own . . . . .	96
7.3.4	Test Case 17 -Preventing DNS Poisoning attacks . . . . .	97
<b>8</b>	<b>Conclusion and Future Work</b>	<b>98</b>
8.1	Revisiting Aims and Objectives . . . . .	98
8.1.1	Aims and Objectives . . . . .	98
8.1.2	Addressing Identified Problem . . . . .	98
8.2	Conclusion . . . . .	100
8.3	Future Work . . . . .	101
8.3.1	Interoperability Support . . . . .	101
8.3.2	Domain Validation . . . . .	101
8.3.3	Ethereum Price . . . . .	102
8.3.4	Validation to the browser and TLSChain support to meta- mask . . . . .	102
8.4	Contribution and Novelty . . . . .	103
<b>Appendix A : TLSChain Source Code</b>		<b>104</b>
<b>Bibliography</b>		<b>118</b>

# List of Figures

1.1	Evolution of Secure Communication . . . . .	2
1.2	CA Hierarchy of Trust . . . . .	3
2.1	Blockchain Overview . . . . .	9
2.2	Chainlink Overview . . . . .	17
2.3	Certificate Issuing Process . . . . .	21
2.4	Structure of a X.509 V3 Certificate . . . . .	24
2.5	Let'sEncrypt Domain Verification . . . . .	27
4.1	TLSCChain Overview . . . . .	42
4.2	Domain Public Key Registration . . . . .	48
5.1	TLSCChainCLI Help Option . . . . .	58
5.2	Published Chainlink Job Specification . . . . .	68
5.3	Chainlink Job Reading Server Value . . . . .	69
5.4	Certificate Details Provided by the Browser . . . . .	69
5.5	TLSCChain Extension . . . . .	71
6.1	Self-Sign SSL Certificate Creation . . . . .	73
6.2	Generating public key hash using TLSCChain . . . . .	73
6.3	Creating Configuration Snippet . . . . .	74
6.4	Adjust the Nginx Configuration to Use SSL . . . . .	75
6.5	Public Key Hash in the public folder . . . . .	75
6.6	Public Key Hash in the public folder . . . . .	75
6.7	Domain registration with TLSCChain . . . . .	76
6.8	TLSCChain Extension . . . . .	77

## LIST OF FIGURES

---

6.9	Step 5 - TLSChain Extension . . . . .	78
6.10	Step 5 - TLSChain Extension . . . . .	78
6.11	Validate the Public Key Hash with TLSChainCLI . . . . .	79
7.1	Test Case 1 - Anne Successfully Register Domain Certificate with TLSChain . . . . .	82
7.2	Test Case 2 - Eve's unsuccessful attempt to register annefernando.com with her certificate . . . . .	83
7.3	Test Case 3a - Anne successfully extending the validity . . . . .	83
7.4	Test Case 3b - Eve's unsuccessful attempt to extend the validity . . . . .	84
7.5	Test Case 3c - Anne successfully changing the certificate for the same domain . . . . .	84
7.6	Test Case 4 - Eve's unsuccessful attempt to renew the domain public key validity . . . . .	85
7.7	Test Case 5 - Retrieving Certificate Domain Details . . . . .	86
7.8	Test Case 6 - Anne successfully revoking . . . . .	87
7.9	Test Case 7 - Eve's unsuccessful attempt to revoke . . . . .	88
7.10	Test Case 8 - TLSChain Extension Verification . . . . .	89
7.11	Test Case 9a - TLSChain Extension Verification Incorrect Public Key Received from the Server . . . . .	90
7.12	Test Case 9 - TLSChain Extension Verification Incorrect Public Key Received from the Serve . . . . .	90
7.13	Test Case 10 - TLSChain Extension Verification . . . . .	91
7.14	Test Case 14 - TLSChain Smart Contract Registration . . . . .	93
7.15	Test Case 14 - Chainlink Oracle . . . . .	93
7.16	Test Case 14 - TLSChain Smart Contract Revocation . . . . .	94
7.17	Test Case 14 - ETH to USD fluctuating . . . . .	95

# List of Tables

7.1	Legit User Anne's Details . . . . .	81
7.2	Attacker Eve's Details . . . . .	81
7.3	DV Certificate Providers . . . . .	91
7.4	TLSChain Costs Analysis . . . . .	93
7.5	TLSChain Costs Analysis . . . . .	94
7.6	TLSChain Costs Analysis . . . . .	94
7.7	TLSChain Costs Analysis . . . . .	94

# Chapter 1

## Introduction

### 1.1 Motivation

Originally, symmetric encryption scheme with the involvement of a secret key, is the cryptographic solution which is used to share a secret between two parties. The sender encrypts the message using the secret key and the receiver decrypts the cipher text using the same secret key to obtain the original message. In this scheme it is necessary to share the secret key between message exchanging parties. Secure exchange of this secret key is difficult and can be considered as the main drawback of the symmetric encryption scheme.

To overcome the above-mentioned secret key sharing problem, public key cryptography which is based on two keys was introduced. The public key is used for encrypting the message and the private key is used for decrypting the cipher text to obtain the original message. The private key is not disclosed and securely kept with the owner, while public key can be disclosed to any party interested in sending an encrypted message.

Public key cryptography solves the secret key sharing problem in symmetric encryption scheme but introduces a new problem of distributing trusted authentic public keys. Trusted 3<sup>rd</sup> parties named Certification Authorities(CA) provides a solution for this by issuing digital certificates which includes metadata such as organization's name, email address, country, entity that issued the certificate, along with the public key. Any party that wishes to communicate securely can

---

obtain this digital certificate and extract the public key. CAs are trusted by the owner of the digital certificate and the party relying on the verification. Refer Figure 1.1

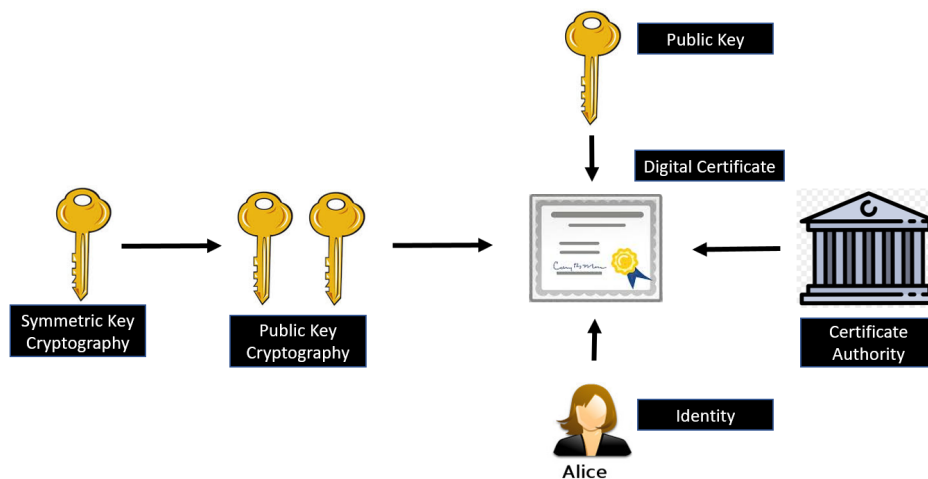


Figure 1.1: Evolution of Secure Communication

Public key infrastructure(PKI) guarantees secure exchange of identities over the internet by providing policies and procedures to issue, manage, validate and distribute digital certificates. To do this, PKI implements a centralized trust model with a hierarchy of trusted certificate authorities.

CA is one of the most important elements in the PKI. Certification authority is a centralized solution and it can be considered as the single point of failure in the PKI which can lead to Denial of Service attacks. Recently there have been several incidents of hackers taking advantage of this nature of the certification authorities.

For instance, due to a security breach in CA DigiNotar, by using the company infrastructure (Wolff 2016) many different attackers were able to generate a large number of rogue digital certificates for high profile domains. This breach infected domains pertaining to CIA, Mossad, Google, Microsoft & Twitter. There were roughly 500 such fake certificates discovered hence web browser vendors were forced to revoke certificates issued by the CA DigiNotar. During the same month



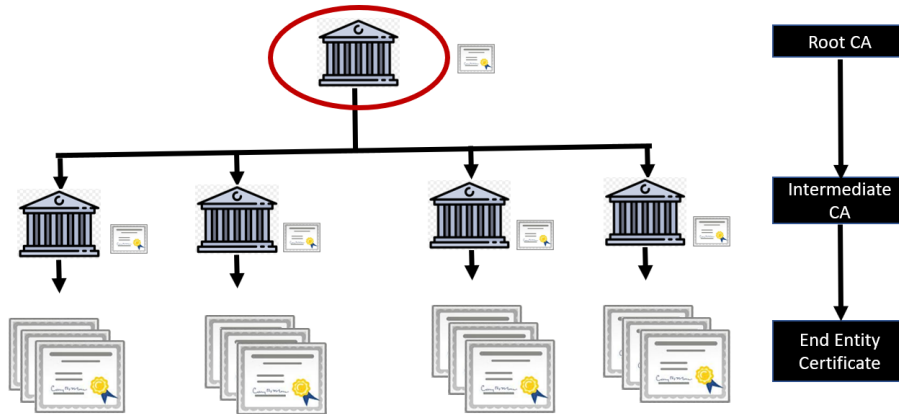


Figure 1.2: CA Hierarchy of Trust

DigiNotar had to declare bankruptcy (Fisher 2012).

In a different case, Malaysian CA DigiCert Sdn. Bhd had mistakenly issued weak SSL certificates. These certificates have been used to impersonate websites and due to this issue, major browsers had to revoke all the certificates issued by this CA (Fisher 2012).

Even Though root certificates are trustworthy there could be mistakes along the chain by intermediate certification authorities. Such an incident occurred for TrustWave, a large U.S based certificate authority. This intermediate root certificate authority allowed their holders to create certificates to any domain on the internet. TrustWave then revoked the certificate and terminated issuing intermediate certificates to customers (Constantin 2012) Refer Figure 1.2.

Apart from above situations, researchers have challenged the trustworthiness of certification authorities. CAs continued to use obsolete cryptographic technologies, signed certificates without verifying their content, and signed certificates that browsers parsed incorrectly, putting users at risk of undetectable attacks (Schoen 2017).

---

## 1.2 Statement of the problem

Is it possible to design a novel Public Key Infrastructure by eliminating the centralized Certification Authorities by incorporating blockchain middleware to achieve on-chain domain verification?

## 1.3 Research Aims and Objectives

### 1.3.1 Aims

From the incidents that we have examined we can identify following issues related to Certification Authorities

1. CA compromised. Hackers issuing rouge certificates using the private key of the CA.
2. CA accidentally issuing erroneous certificates to customers enabling customers to act as CA themselves.
3. Trusted root CAs are hardcoded to browsers and operating systems. In order to remove a trusted CA a security updates should be pushed.
4. Certificate signing cost. A single certificate can cost between 100\$ to 1000\$ depending on the specific CA and the required certificate.
5. Slow certificate signing process

As there are a growing number of adversary attacks and security concerns related to CAs, the aim of this project is to completely eliminate the single point of failures in the PKI which is the Certification Authority. Current blockchain based PKI systems uses blockchain as a storage medium and uses some-form attestors ( existing Certification Authorities or 3rd party attestors). This project we will consider whether it is possible to decentralize the functionalities of the certification authorities by tightly coupling the on-chain domain validation without using any form of attestors and using blockchain technology more than a storage medium to overcome the above-mentioned drawbacks of certification authorities in the secure web communication domain.

---

### 1.3.2 Objectives

The iconic paper published by Satoshi Nakamoto “Bitcoin: A peer-to-peer electronic cash system” introduced a novel way to transfer digital money by solving the long-worried problem of digital cash “double spending” (Nakamoto 2008). This is exactly the similar situation that we are trying to solve in this project. We are looking for a mechanism that controls and verifies the authenticity of the public key (Two distinct entities cannot have the same public key). With this inspiration we are proceeding this project by incorporating blockchain technology.

By design blockchain technology provides many characteristics which are usable when finding a solution to the CAs based PKI problems. Blockchain is a peer to peer distributed ledger technology that provides a shared, immutable and transparent history of all the transactions in a network. It is decentralized!

We are expecting to evaluate different properties of blockchain technologies. Such identified properties are Permission Type, Blockchain Type, Storage Type, Smart Contracts, Privacy concerns. Also, there are PKI related properties that we are expecting to consider. Such PKI properties are Trust Model, Revocation, Data Structure, Incentives, Updatable Keys.

To address the inefficiencies tied to current blockchain based PKI systems, we are hoping to incorporate next generation blockchain technology named smart contracts. With the usage of smart contracts and blockchain middleware, we are hoping to incorporate verification functionalities with the storage of identities which is a major drawback of the current approaches of blockchain based PKI where CAs are still in use to issue certificates and blockchain is used as a storage mechanism.

Following the experimental research approach, we will try to design a blockchain based PKI system to replace the functionalities provided by the CAs. Before doing this, we will evaluate different blockchain platforms and trust models to come up with a novel approach for a blockchain based PKI without storing X.509 certificates.

A proof of concept will be developed by using available blockchain technologies and hoping to use publicly available resources. Knowledge of theory extracted from the literature survey will be used when determining resources which will be

---

effectively lead to find a solution for the identified problem.

### **1.3.3 Scope**

There are different type of digital certificates which are used in different domain.

#### **Types of digital certificates**

- Root Certificate
- Personal Certificate
- Software Publisher Certificate
- Content Signing Certificate
- Server Certificate(SSL Certificate)

#### **Different domains that digital certificates are used**

- Digital Signatures
- Encryption for Email
- Smart Cards
- Web Communication

In this project we will be only considering server certificate(SSL Certificates) in the web communication domain. The main focus is towards establishing a trust model which can be replace the hierarchical trust model that certificate authorities have established.

### **1.3.4 Structure of the Thesis**

Chapter 2 of this thesis will contain a detailed literature review. It will comprise of an in-depth study on the blockchain technology and public key infrastructure. Chapter 3 will contain the methodology and the Chapter 4 will describe the design

---

of the proposing PKI. Chapter 5 will discuss the implementation, Chapter 6 will be a sample case study and Chapter 7 is focused on evaluation and Chapter 8 will provide the conclusions.

# Chapter 2

## Literature Review

### 2.1 Distributed Ledger Technology

A distributed ledger is a record of consensus with a cryptographic audit trail which is maintained and validated by several separate nodes (Rutland 2018). Synchronized ledger is spread across multiple nodes and it is cryptographically secured. Distributed ledgers could be either centralized or decentralized. In the decentralized approach it gives the equal rights within the protocol to all the participants and in the centralized, only a designated user has particular rights.

### 2.2 Blockchain

Blockchain is a decentralized, append-only database of signed transaction or operations that yield a new globally consistent state which can be considered as a combination of distributed ledger technology and blocked transactions.

Blockchain is one way of implementing a distributed ledger (but not all distributed ledgers necessarily employ blockchains). As per distributed ledger technology, blockchain contains chain of digitally signed, unchangeable data packages and in the blockchain domain it is called “blocks”. A block is a bundle of transaction data. After certain predetermined criteria is met, blockchain is capable of producing a new “block” where distributed ledger is only verifies a transaction one it is submitted (Rutland 2018).

---

## 2.2.1 Blockchain Characteristics

- Un-editable record of all the transactions made

Blockchain creates the chaining effect by, current block referring to the signature of the previous block in the chain, and that chain can be traced all the way back to the genesis block. Data in a specific block cannot be altered without changing subsequent blocks, which require the network consensus. As a result of feature blockchain contains un-tampered records of all the transactions made. Refer Figure 2.1.

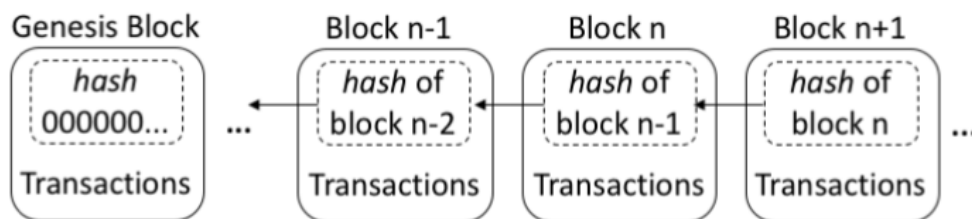


Figure 2.1: Blockchain Overview

- Transparent and Distributed

A non-refutable and unbreakable record of data Blockchain is inherently distributed (Many parties hold the copied of the ledger as it is replicated across several nodes), hence there is no single point of failure.

- Consensus algorithms

Consensus algorithms determines the state of the ledger. The ledger's version validity is established though the consensus among the participating nodes, called miners. Even though there are several mechanisms like Proof Work, Proof of Stake, Proof of Authority but ultimately all serves to validate information from inputs to the network.

Miners creates new blocks by validating transactions through Proof of Work exercise (Or any other consensus algorithm). Anyone can create a valid block if they can expand the required computing power (in PoW calculating hashes). This resource intensive process also creates a financial barrier to prevent malicious attacks.

- 
- Proof of Work(PoW)

Consensus mechanisms that requires miners to go through an intense race of trial and error to find the nonce for a block. Only blocks with a valid nonce can be added to the chain.(Buterin et al. 2014).

- Proof of Stake(PoS)

Consensus mechanisms that work by selecting validators in proportion to their quantity of holdings in the associated cryptocurrency. Unlike a proof of work protocol, PoS systems do not incentivize extreme amounts of energy consumption.

- Proof of Authority(PoA)

Consensus mechanism based on identity as a stake.

## 2.2.2 Blockchain Categorization

There are numerous variations of blockchain technologies exist in the conceptual and the implementation level. We are exploring and distinguishing following properties which are relevant to this project (Brunner, Knirsch, Unterweger & Engel 2020).

- Permission Type

- Public

Public (Permission-less\Decentralized) Anyone can perform transactions on the blockchain. All the data is accessible to the participants in blockchains. As this type of blockchains are open to public, mechanisms to combat vulnerabilities should be included to the design. These mechanisms will prevent people from corrupting the system(Eg :- Proof of work in Bitcoin).

- Private

Private (Permissioned\Centralized) These type of blockchains are consist of parties whose identities are known. As there are only credible and repudiated participants can post to the blockchain systems are



---

considered to be valid and since the identities are known, the transactions can be audited. Data is only selectively accessible at participant level.

- Blockchain Type

- Using Established Network

Implementation can be done on established and well investigated technologies. For this purpose existing blockchain networks can be used. Bitcoin, Ethereum or a fork of an existing blockchain code base to setup a new (Eg:- Namecoin is a fork of Bitcoin). Advantage of using an established blockchain is that it has been used and tested by the community around the world.

- Starting from scratch

Developers should write code to implement a new blockchain network. Security and end user acceptance can be negatively impacted in these situations.

- Storage Type

- On-chain

Data relevant for the use case is directly stored directly on the blockchain. Within blocks and/or transactions. Full on-chain storage means all the data is stored on the blockchain.

- Off - chain

Data is not stored in the blockchain. It can be stored in an external storage either privately or publicly.

- \* Public - Stored in publicly accessible web servers. Everyone has access to the data without restrictions.

- \* Private - Data is kept under the control of a limited number of entities and allows only limited access.

- 
- \* DHT (Distributed Hash Table) - Special form of off-chain storage, where the stored data is distributed among multiple participants. If a cryptographically secure hash function is used within the DHT to address the stored data then the data can be timestamped, integrity and tamper-proof protected.
  - \* IPFS (Inter Planetary File System) - This is a popular storage layer for decentralized applications. It is a peer-to-peer data distribution protocol where nodes in the IPFS network form a distributed file system. Cryptographic Hashes are used when addressing data in IPFS and the link always stays the same irrespective of which node serves the data. Storing data in blockchain costs money hence economically is not practical to store large amount of data on the Ethereum blockchain. IPFS is ideal for blockchain application, as it makes it possible to address large amount of data from truncation in the blockchain using permanent and immutable IPFS links.

- Privacy

- Storing a reference to the data not the actual data. Using hashed or salted hashes.
- Permissioned blockchain - Effectively private
- PET (Privacy Enhancing Technologies) - Can be used in permissionless blockchain

- Evaluation

Operation, transactions and storage on blockchains can be evaluated in terms of time, space, complexity and cost

- Complexity - How much computing power or memory is needed with a changing number of users, objects or per any dependent variable.
- Cost - Different blockchain implementations charge different fees for the transaction and the data volume contains. Hence it is needed to calculate cost per user, per object and dependent variable.

---

### **2.2.3 Bitcoin and Ethereum**

Currently Bitcoin and Ethereum are the most popular public blockchains that we have today. They have many similarities in terms of blockchain concepts, but the key difference is that Bitcoin is merely a cryptocurrency but Ethereum's capabilities goes beyond a cryptocurrency. Ethereum is intended as a platform to facilitate immutable, programmatic contracts and applications via its own currency.

#### **2.2.3.1 Bitcoin**

With the idea of eliminating financial institutions from the electronic cash based online transactions, Santoshi Nakamoto proposed a new electronic cash system named "Bitcoin" (Nakamoto 2008). This revolutionary idea of Bitcoin was not the first attempt at an online digital currency, but it has become the most successful cryptocurrency which has developed in past decade.

Even though the name Blockchain was not specifically mentioned in this paper, it is the underline technology used where series of data blocks are cryptographically chained together. With Bitcoin it was possible to transfer digital money without going through trusted third-party or intermediary bank.

Over these years Bitcoin has managed to co-exist with the financial systems, gained acceptance among regulators and government bodies despite of being regularly scrutinized and debated.

#### **2.2.3.2 Ethereum**

Vitalik Buterin published the Ethereum white paper after being inspired by the success of Bitcoin. Ethereum is more than a cryptocurrency which enables the deployment of smart contracts and decentralized applications (Dapps). Presently Ethereum is the largest and most well-established, open ended decentralized software platform(Buterin et al. 2014).

Ethereum's cryptocurrency is called Ether. Ethers servers as a meant to incentivize participants to engage in the protocol. Miners work is compensated by giving them transaction fees that are expressed in unit called gas and calculated based on the complexity of the code they execute.

---

## 2.2.4 Smart Contracts

Contracts are a well-established concept in the legal arena. It is a legally binding document between that defines and governs the rights and duties of a parties to an agreement. In Ethereum this concept has been incorporated and evolved.

Smart contract is a self-executing(automated) contract with the terms of the agreement between buyer and seller being directly written into lines of code. The code and the agreements contained therein exist across a distributed, decentralized blockchain network. Execution is controlled by the code and the created transactions are traceable and irreversible, no party can block or otherwise tamper with.

Smart contracts allow parties involved in an agreement to conclude with certainty that they in consensus all the times as the existence, nature and evolution of the facts shared among them, which are governed by the program. Also, smart contracts are useful to satisfy common contractual conditions, lower transaction cost and rusk and eliminate the need for trusted intermediaries.

Ethereum smart contracts are accessible and transparent like open APIs. In Ethereum smart contract is a program that runs on Ethereum blockchain. Smart contract has a specific address and it contains collection of code (functions) and data (state) that resides at a specific address on the Ethereum blockchain. The basic order of transactions maintains the consistency smart contract execution by peers. (Zhao, Lin, Huang, Zhang & Xiang 2020).

Ethereum supports Solidity programming language, which is a high-level, complex, Turing complete language and then it is compiled-down to Ethereum Virtual Machine. To prevents miners end up in a never ending loop (due to Turing completeness), transactions and message calls specify an upper bound on the amount of gas that they can consume. Contracts become part of Etheteum's global state by wrapping this initialization code in a transaction, signing it and broadcasting it to the network.

The code and the state of the smart contracts are publicly accessible hence it can be trusted for correctness. Also it is widely popular due to the large open source community contributes to an efficient development process.

---

## 2.2.5 Dapps (Decentralized application)

In General, Decentralized applications are digital application or programs that exist and run on peer to peer network of computers instead of a single computer and are outside of the scope and the control of a single authority. While the backend code running on a decentralized peer to peer network, the frontend and the user interface can be written in any language that can make calls to its backend. In Ethereum with smart contracts building Dapps has become much easier. Dapps back end consist of smart contracts (ethereum.org 2020).

### 2.2.5.1 Characteristics of Dapps

- Decentralized - No Central authority to control. Take independent decisions.
- Deterministic - Irrespective of the environment that they are executed, all the nodes perform the same function.
- Turing Compatible - Given the required resources, Dapps can perform any action.
- Isolated - Smart contracts are executed inside Ethereum virtual machines. Because of this approach if a smart contract contains a bug it won't hamper the normal functioning of the Ethereum blockchain network.

### 2.2.5.2 Benefits of Dapp Development

- Zero Downtime - smart contract which is the core of the Dapp deployed in the Ethereum blockchain, the network as a whole will be able to serve clients looking to interact with the contract. Hence when it comes to individual dapps, it is not possible to launch denial of service attacks.
- Privacy - It is not needed to have a real-world identity to deploy or interact with Dapps.
- Complete Data Integrity - Malicious actors cannot forge transaction or the data that has already been made public as Data stored in blockchain is immutable and indisputable.

- 
- **Transparent and Verifiable Behavior** - Without a need of a central authority it is possible to verify deployed smart contracts and they are executing in predictable ways. When compared the same with the traditional model, we need to trust a financial institution or a central authority with data and transaction misuse.

## **2.2.6 Blockchain Oracles (Blockchain Middleware)**

Blockchain Oracle is any device or an entity that connects a deterministic blockchain with off-chain real world data. Since blockchain oracles creates a bridge between two world it is also known as blockchain middleware (chain.link 2020).

### **2.2.6.1 The need of Oracles**

Due to the blockchain's distributed nature, each node in the network should come up with the same value with the given same input (Deterministic property). Otherwise when a node tries to validate a transaction that another node created, it would end up in different results. Then none of the nodes would be able to agree upon what the actual state of the blockchain is. Due to this reason the architecture has been intentionally designed to be deterministic. Deterministic blockchain means if a transaction is replayed it should end up in the correct state. If we have included an API call or another non-deterministic sources into the infrastructure of the blockchain then there is a possibility that the source will be deprecated, hacked, or broken. In situations like this it is difficult to come to a consensus.

### **2.2.6.2 Chainlink**

The deterministic nature of the blockchain does not allow to access the outside data. If the suggested oracle is a centralized solution, then it will nullify the advantages of smart contracts.

Chainlink is a decentralized oracle which will solve above mentioned problems. For blockchain oracles Chainlink introduces the same decentralized infrastructure concept that blockchain has. This is a framework to choose to independent network nodes to connect the real world's data to the block chain to enable smart

---

contracts to reach their true potential and security guarantees. The decentralized network will carry on if the nodes and sources are hacked, deprecated or deleted. Figure 2.2 how blockchain networks access outside data with the usage of Chainlink.

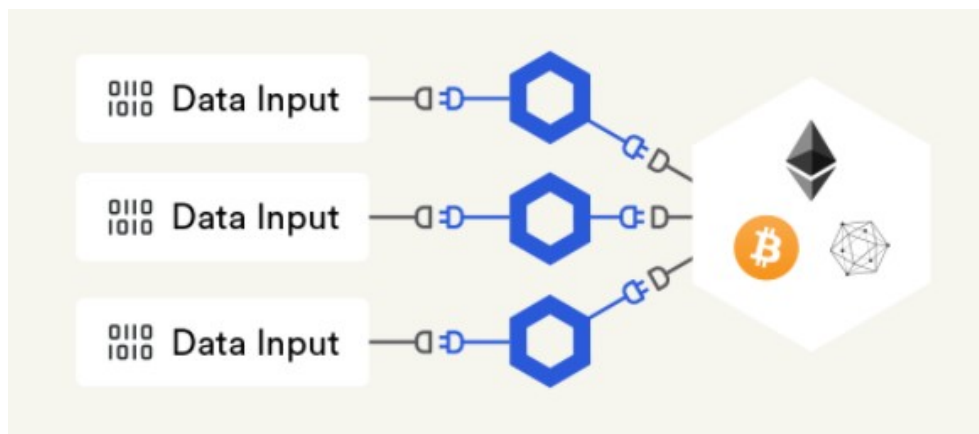


Figure 2.2: Chainlink Overview

Multiple chain-link to evaluate the same data before it becomes a trigger.

## 2.3 Public Key Infrastructure

To secure and to encrypt the communication among multiple parties, Public Key Infrastructure(PKI) is heavily used over the internet. Identity of a user is authenticated and managed by PKI entities. PKI binds the physical identity of a user with certificates.

PKIs are protocols of binding

- Public key to a name, email address, identity of an individual for authentication
- Establishing secure communication channel
- Verifying the creator of signatures

---

PKI is a set of entities, policies and procedures where the public key name pairs are issued managed and revoked. This is based on public key cryptography and require entities to have a public key and a private key (secret key). With the usage of a digital certificate, PKI provides the link between a public key (which has a corresponding private key) and its owner.

### 2.3.1 PKI supported functions

- Registering an identity with a corresponding public key
- Updating the public key corresponding to a previously-registered identity
- Looking up a public key corresponding to a given identity
- Verifying a public key corresponds to a given identity (step more than performing lookup)
- Revocation (coping with key compromises) or backup

fundamental building block of many applications that rely on secure and reliable authentication.

Digital Certificates ensure that a certain entity is bound to its public key. This relies on the trusted servers maintained by certificate authorities. These authorities issue a certificate for a person or a domain that publicly and verifiable binding this entity to a certain key. The most common format of a digital certificate is x.509.

### 2.3.2 PKI Trust Models

The most two common approaches to the public key infrastructure falls into two categories.

- Certificate Authorities
- Web of Trust



---

### 2.3.2.1 Certificate Authorities

Certificate authorities are trusted 3rd party entities, who certify the ownership of a public key by the said entity by providing a signed certificate. This is the most common choice in practice. Hierarchical structured trust model CA issues certificates to participants and other CAs. Root CAs issues certificates to other CAs to build the chain of trust. Recent incident shows that there is too much trust placed in CAs. Eg:-Symantec, GeoTrust, Comodo, DigiCert, Thawte, GoDaddy

Certification Authority based PKI consists of the following entities (Wikipedia contributors 2020)

1. A certificate authority (CA) - that stores, issues and signs the digital certificates
2. A registration authority (RA) - which verifies the identity of entities requesting their digital certificates to be stored at the CA.
3. A central directory - a secure location in which keys are stored and indexed.
4. A certificate management system - managing things like the access to stored certificates or the delivery of the certificates to be issued.
5. A certificate policy - stating the PKI's requirements concerning its procedures. Its purpose is to allow outsiders to analyze the PKI's trustworthiness.

### 2.3.2.2 Web of Trust

Web of Trust establish the trust by verifying that a party is trusted by at least one already trusted entity (Certificate is signed by an entity whom the verifier has previously established trust). In web of trust all the participants are equal to issue certificate to confirm each other's public keys. Eg:- PGP Network of trust (Caronni 2000)

### 2.3.2.3 Log based PKI

The Log based PKI's public log allows to audit all the CA activities but does not provide a fully decentralized approach. Log based PKI are an extension to

---

both hierarchical and Web of trust-based PKIs. This has a publicly append only database and needs to register before they are considered valid. As the certificates are publicized any misbehavior can detect quickly and denounce publicly.

Eg:- Certificate Transparency Project by Google

### **2.3.3 Certificate Authority Functionality**

CA provide signed certificates to an entity on request, Certifying ownership of a public key by the said entity. Relying parties check the validity of the signature using the corresponding CA's public key stored in a file called "TrustStore".

There are two types of certificate authorities. Root CA and Intermediate CA. Intermediate CA's get a signed certificate from root CA which allows intermediate CA to sign certificates on behalf of the root CA. There could be many intermediate certificates with different trust levels, issuing different kind of certificates. If they are acting behalf of one single CA then only the certificate of the root certification authority needed to be stored in the trust store.

Usually Root CA kept offline and if there is a need to revoke the trust in the intermediary CA, Root CA comes online and revoke the trust in the intermediary CA.

### **2.3.4 Certificate Authority Certificate Issuing Steps**

1. Acceptance of certificate signing requests
2. Verification of entities' identity
3. Signature of digital certificates
4. Revocation of certificates

#### **2.3.4.1 Certificate Issuance Process**

- Certificate Signing Request

When an entity requests for a certificate, they should provide the certificate signing request which will be signed by the CA. This CSR will include following information

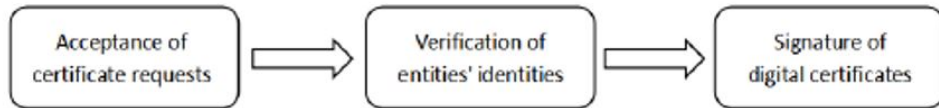


Figure 2.3: Certificate Issuing Process

- Common Name (CN)
  - Email address
  - Company Name
  - Department
  - Address
  - Public Key
- Registration Authority

Registration Authority is a part of the certificate validation process where it will help to apply for, approve, reject and revoke certificates. Once the RA validates the information provided in the CSR, it will contact the CA and then CA will issue a date of expiry and sign the certificate with its private key and provide it to the requesting party.
  - Types of Certificates
    - Domain Validation Certificate (Class 1, Class 2) - To obtain this kind of certificate client needs to prove the Registration Authority that they have the control over the specified domain in the CSR. During the verification process domain owner will receive a verification request via e-mail and to prove the ownership of the domain he has to reply it. With Automatic Certificate Management Environment Protocol (ACME Protocol) this process is automated.
    - Organization Validation Certificates (OV) and Extended Validation Certificates (EV) (Class 3) - Registration Authority will be more involved, and also more scrutiny. In this process it will check that the

---

company is registered under the specified country in CSR. During EV, RA will confirm that the CSR was originated from the authorized company. In this process's RA could make a phone call or request paperwork or other out of band communication.

#### 2.3.4.2 Revoking Certificates

CA will revoke the certificates due to following reasons.

- Request from the domain owner (Authorized person)
- Private key of the certificate has leaked
- The company that certificate was issued for gone out of business
- Certificate was revoked by an accident

When a certificate is invalidated, it should be notifying to the client that the certificate is no longer trusted. This is a service directly provided by the CA or Validation authority is authorized inform about the status of the revoked certificates behalf of the CA.

Certificate revocation Two approaches of certificate revocation.

- CRL(Certificate Revocation List) - This contains a list of revoked certificates signed by a CA. CRLs are uploaded to public repositories like a public FTP. Any client needs to verify a validity of a certificate should obtain the recent CRL and validate it against the serial number of the certificate. The drawback of this is approach is that CRLs could be very lengthy as the size of the CRL is proportional to the number of revoked certificates.
- OSCP(Online Certificate Status Protocol) - This protocol is used to check for the revocation status of a certificate. OSCP responses for an inquiry from a client consist of a time stamped data structure signed by the CA, which reveals the revocation status of a certificate at a given time. OSCP responder server is responsible for this task.

According to OSCP stapling protocol, to speed establishing secure connection, the OCSP responses can be sent during the TLS handshake.

---

If the OCSP responder becomes unresponsive then it is not possible to check the status of the certificate. This kind of situation can be occurred due to a man in the middle attack (dropping the response from the OCSP responder). Some client might ignore a failure to check the revocation status of a certificate instead of terminating connection which can be exploited by an adversary. This can be solved by using OCSP stapling and a certificate can enforce OCSP stapling through a X.509 v3 extension.

If an OCSP responder become compromised by an adversary then there isn't a mechanism to revoke the trust in an OCSP responder. This should be manually configured in the clients relying on the OCSP responder.

### 2.3.5 X.509 Certificates

The most common type for digital certificate is X.509. Figure 2.4 depicts fields of a X.509 Version 3 digital certificate.

- Certificate Serial - Number Uniquely identifies a certificate issued by the CA.
- Signature Algorithm - Identifier Contains the name and parameters of the signature algorithm used by the CA to construct the CA signature.
- Issuer - The X.500 name of the certificate authority who has signed the certificate.
- Validity - period Contains a start and expiry date which defines the period where the certificate should be considered valid.
- Subject - The distinguished name (DN) of the entity who owns the private key corresponding to the public key in the certificate.
- Subject Public Key Info - information Contains the public key of the subject together with the algorithm and parameters used to construct the key.
- Extensions - Added in X.509 version 3 and contains a list of certificate extensions.

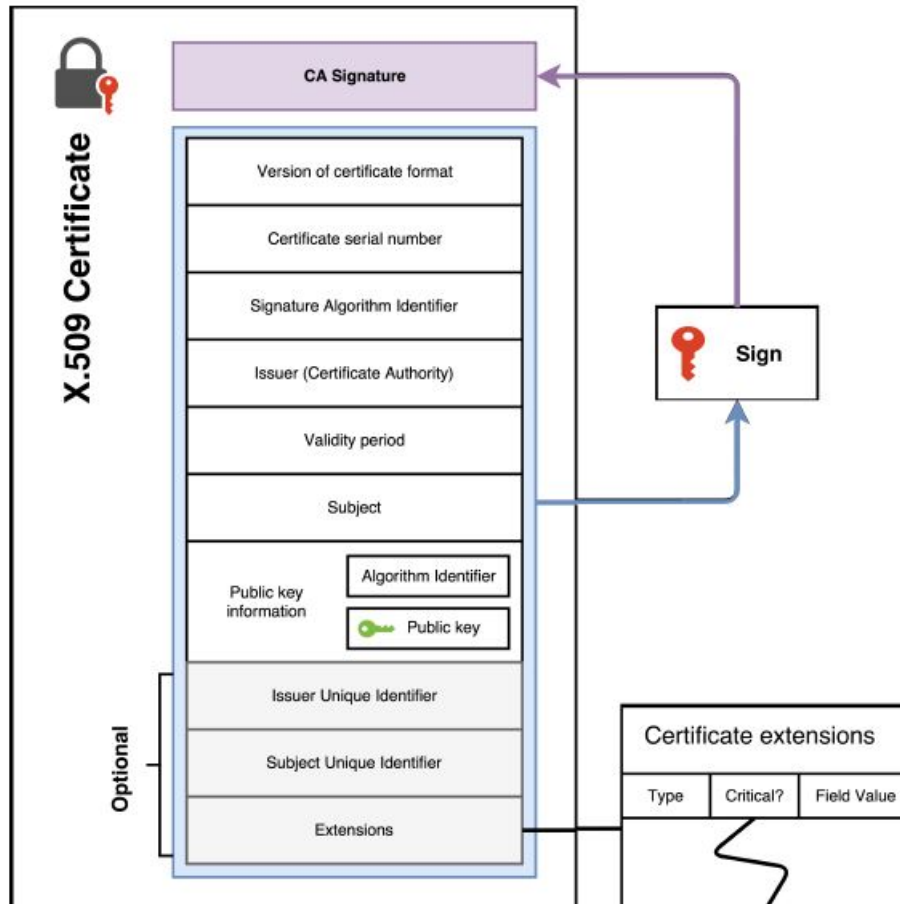


Figure 2.4: Structure of a X.509 V3 Certificate

## 2.4 Current Applications

### 2.4.1 Decentralized Public Key Infrastructure

Decentralized PKI to return control of online digital certificate to the entities they belong to.

---

#### **2.4.1.1 Web of Trust**

This is an entirely decentralized approach, which eliminates the central point of failure. Users label each other as trustworthy by signing their public key. A user will collect certificates which will contain his public keys and signatures from others who will find him trustworthy. Then a third party can verify this user by checking whether it contains the signature of someone he trusts.

Even though this is an efficient decentralized approach it is difficult for a new user to join the network. Also, it is not possible to deal with the key revocation. A user can choose only one other user to be the "designated revoker" and provide a grant to revoke the certificate if the private key is compromised (Yu & Ryan 2017).

#### **2.4.1.2 Log Based PKI**

In log-based PKI a public log is used to monitor and publish the certificates issued by CAs. These public logs are deployed in high availability servers and they provide transparency hence only publicly logged certificates are accepted. From this approach it is possible to identify misbehaving CAs. The most successful Log based PKI is Google's Certificate Transparency and it is available in Chrome and Firefox (Laurie, Langley & Kasper 2013). Even though there are many benefits in this scheme, certification revocation is still a challenge (Matsumoto & Reischuk 2017).

### **2.4.2 ACME Protocol**

In this project we are examining ACME protocol as it is an automatic certification issuing protocol which can be incorporated to blockchain technology when implementing a PKI (Kfoury, Khoury, AlSabeh, Gomez, Crichigno & Bou-Harb 2020).

Automatic Certificate Management Environment is a communication protocol which automates the interaction between the certificate authorities and the user's webserver. This was designed as a part of Let's Encrypt service which provides public key infrastructure for a lower cost. This protocol is based on passing JSON-formatted messages over HTTPs.

ACME is now published as an internet standard in RFC 8555.

---

### 2.4.3 Let's Encrypt

Let's encrypt is an open and automated certificate authority that uses the ACME protocol to provide free TLS/SSL certificates to any compatible clients. These are typical SSL certificates that can be used to encrypt communication between web server and the web client(LetsEncrypt n.d.).

Setting up a web server to use HTTPS communication with Let's encrypt is a two-step process.

- Certificate management agent deployed in the web server proves to the CA that web server controls a domain.
- Certificate management agent can request, renew, revoke certificates for that domain.

#### 2.4.3.1 Certificate Management Agent Software

To get a let's encrypted certificate an ACME client software should be used. These clients are offered by third parties. The most popular ACME client is Certbot. Certbot is capable of configuring TLS/SSL on both Apache and Nginx web servers in addition to verifying domain ownership and fetching certificates.

#### 2.4.3.2 Domain Validation

Let's encrypt offers domain validation certificates by identifying the sever administrator by the public key. These certificates make sure that the request come from a person who actually controls the domain. As the initial step agent software interacts with Let's Encrypt and generates a new key pair and proves to the Let's Encrypt CA that the server controls one or more domains. Refer Figure 2.5. For this Let's encrypt CA will look at the domain name being requested and issue one or more sets of challenges. There are two ways to that the agent can prove the control of the domain. Eg:- if proving the control of example.com

- Provisioning a DNS record under example.com
- Provisioning an HTTP resource under a well-known URI on <http://example.com>



Along with these challenges, Let's encrypt CA provide a nonce that the client agent must sign to prove that it controls the public and private and key pair. After completing above two tasks CA will verify that the challenges have been satisfied and the signature over the nonce is valid. The key pair that the agent used is marked as an "authorized key pair"

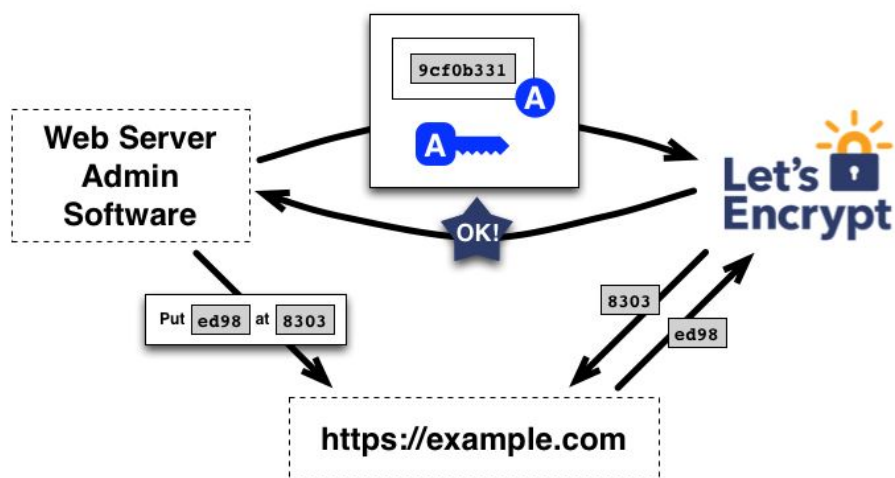


Figure 2.5: Let'sEncrypt Domain Verification

### 2.4.3.3 Certification Issuance and Revocation

Once key authorization step is done, by using certificate management messages certificates can be requested, renew and revoke. Certificate is obtained by providing ta PKCS#10 CSR. CSR includes a signature by the private key corresponding to the public key in the CSR. Agent also signs the CSR. Let's encrypt CA verifies the both signatures and if it is successfully verified a certificate is issued. To revoke a certificate, Certificate revocation request is made by signing with the authorized key pair. Let's encrypt CA verifies thing and publish the revocation information into the normal revocation channels.

---

#### 2.4.4 Trust CA

Authors of this project are using smart contract in Ethereum platform to build and independent entity named CA Proxy to manage life cycle of digital certificates also they are integrating the CA proxy with the current CAs through applying blockchain oracle services (Zhao et al. 2020).

The Certificate transparency is achieved through life cycle management of digital certificates in blockchain platform. They are considering the process of digital certificate issuance by CA, life cycle management of digital certificates, and integrating CA proxy with current CAs. This approach does not eliminate the Certification Authority central entity but introduced more security by making all the request that goes to certification authority more transparent. All the metadata related to certificate requests, certificate signing, status checking, verification requests will be stored in the blockchain. Issues related to certification authority is not eliminated but rather making the auditing and the information transparency more evident.

They have identified four weaknesses of current certificate authorities

1. Problem in Acceptance of certificate Requests - log of Certificate Signing Requests is not available publicly. If a hacker obtains CA's private key it is easy to forge the digital certificate. If there is a problem with the certificate, the browser or operating system can audit whether the CA has recorded the corresponding certificate request. If not, certificate is an invalid one.
2. Problem in Verification of Entities' Identities - CA does not have high credibility in identifying the entities. Domain verification implementation has some security holes when domain verification is requested via email.
3. Problems in signature of digital certificates - When the browser and the operating system vendors install a CA certificate in them, they trust the CA completely. At present there is no recognized root CA certificate list, each browser and OS as a different list of root CAs hence this makes it difficult for users to get a list of trusted root CAs. In the current architecture CAs cannot response quickly when their internal infrastructure is compromised and difficult to identify and track which certificates are forged and should be

---

revoked. In a situation like this, operating systems and browsers needed to push a security update, to remove the compromised certification authority from their trusted root CA repository.

4. Problems in revocation of Digital Certificates - There is no mechanism to auditing certificate authorities and the certificate transparency is not high enough and the security risks are high. CRL approach will over time becomes very large and causes a burden on CRL serves. These are central servers hence prone to denial of service attacks also. In the OSCP approach only provides the information of the requested certificate. Hence no impact to the bandwidth and also less complexity. But an attacker can interfere with the client's OSCP queries. If the client timeouts most client will ignore the OCSP.

### **2.4.5 DeTRACT**

This solution address the PKI single point of failure problem by proposing a decentralized, transparent, immutable and open PKI certificate revocation. Authors have mainly improved and focused on the certificate revocation aspect. Domain owner is responsible for creating a self-signed certificate and store it either of the Bitcoin or Etheruem blockchain networks. In this approach no central authority to manage the issuance of the keys and certificates. For identity verification uPort decentralized identification is used instead of CA that manages identity verification. Authors suggests by this way anyone can verify the identity of any entity of the network. This approach lacks a solid trust model and a domain verification. Instead it relies on the domain owners identity(Sermpinis, Vlahavas, Karasavvas & Vakali 2020).

### **2.4.6 NameCoin**

Namecoin is a distributed DNS based Bitcoin. It is the Bitcoin's first and the longest running fork and uses blockchain as a storage medium for digital identities. Namecoin is the very first pioneer project of blockchain based identity management area. Namecoin operate independent of ICANN and administer its

---

own .bit top domain. New names are registered in this namespace by posting a message to the Namecoin blockchain. This established Name-Value Ownership but it is not extended to handle other requirements related to the PKI like certificate issuance, signing, revocation. As Namecoin was a fork, it was possible to introduce new features, but it also leads to some other issues. 51% attacks from the miner pools were observed, software bugs caused issues and halted registration of new names, selfish mining nodes, discouraged miners due to consensus breaking software updates. The major drawback of this is that it forces clients to download and maintain an entire copy of blockchain to verify records. With the increment of number of registered records, computation and storage requirements scale linearly and limits the system applicability of storage-limited devices like smart phones(namecoin n.d.).

### **2.4.7 Blockstack**

PKI implementation supports a DNS and PKI implementation in the blockchain. Blockstack initially ran on top of the Namecoin blockchain, but now users Bitcoin. The challenges faced in the Namecoin lead Blockstack to use a mature blockchain like Bitcoin. But mature blockchains are slow and costly. Blockstack implemented a 4 layer architecture where they separated the control and the data planes. Blockchain can be used for control information like name registration or transfers while the data planes be ne used for storing data like DNS records or identity information. Hash of a "zonefile" is written on Atlas which is( Blockstack's distributed peer network.) When validating each zone file hash stored in the atlas is searched for a Bloackstack Name Service transaction that contains this hash.

### **2.4.8 Certcoin**

Certcoin is a completely decentralized PKI which has the main focus on stronger Identity retention( Do not effectively prevent one user from registering public key under another's already registered identity). It leverages the consistency offered by blockchain platform Namecoin (Fromknecht, Velicanu & Yakoubov 2014).

In this approach users need to own two key pairs. "Online" and "Offline" key

---

pairs. "Online" key pair is used to authenticate messages to and from the web server and "offline" is used to sign or revoke new keys in security incidents. When looking up the public key "pk" corresponding to an identity "id" is handled by traversing the blockchain and locating the latest value of the desired key. Later improved version uses Cryptography Accumulators for efficient lookups which reduces the time and space needed for verification from linear to logarithmic. Even though this approach has many good features in terms of lookups authors do not address any security model for the PKI implementation nor a proof that it provides the claimed service.

### **2.4.9 SCPKI -Smart Contract-based PKI and Identity system**

Proposed a smart contract-based PKI and identity system where users can add and sign identity attributes to themselves. Due to lack of authorization there is a possibility of users forging identity attributes(Patsonakis, Samari, Roussopoulos & Kiayias 2017).

#### **2.4.9.1 SCPKI Improvements**

first implementation of secure smart contract-based PKI on top of Ethereum. This construction incurs constant-sized storage at the expense of computational complexity. To address the trade-off between storage vs computational cost trade-off, they have built a Hash tree based universal accumulator as well as an RSA based accumulator. Their results should that the hash tree-based construction in the only smart contract-based PKI with constant fixed state that can be deployed on Ethereum's live chain(Patsonakis, Samari, Kiayias & Roussopoulos 2020).

### **2.4.10 Analysis of the current applications**

- Permission type - Majority of the applications are permission less. Permission less implementations are to be preferred for internet scale applications like domain holder verification and permission-ed approach are better for small scale use cases with few participants

- 
- Revocation - Revocation mechanism is built on almost all the approaches. This is one of the key security properties of PKIs and traditionally relies on trusted third parties. The use of blockchain spreads this trust over entities hence it is positive that all approaches implement this property.
  - Blockchain Types - Most implementations are based on Ethereum. Public blockchain is mostly used. Some have used custom blockchains like Namecoin which is based on blockchain. Some approaches only simulate the creation of blocks. Using an established public block chain is beneficial as their reliability, tamper proofness and availability. Due to the security and gathering support of participants in the long run issues, custom block chains are not used much.
  - Certificate Format - Due to interoperability reasons most of the time X.509 certificates used some rely on extensions as in X.509 Version 3. Implementation specific, custom formats are also in the literature and PGP format is also used.
  - PKI Type - Some blockchain based PKI implementations still uses the certificate generated by the CA( Not necessarily eliminating the CA) and store it in the blockchain to achieve greater security by making information and the process transparent.

Most of the implementations are hierarchical, but there are implementations done related to WOT. By having a hierarchical structure means it is still having a root CA and centralized behavior. Permission less blockchains minimize this as it has many decentralized properties. The trusted central nature can be eliminated by using WOT.

## **2.4.11 Conclusion of the Review**

### **2.4.11.1 Gap**

The current blockchain based PKI implementations still has centralized behavior and uses Certification Authorities. Blockchains are mainly used as a storage medium.

---

#### **2.4.11.2 Viable Direction**

Incorporating principles in the ACME protocol and the decentralized nature of the blockchain technology is a viable approach to eliminate the need of a Certification Authority in Public Key Infrastructure.

# Chapter 3

## Research Methodology

In this project we are hoping to follow the experimental research approach to find out whether it is possible to propose an alternative to Certificate Authorities using Blockchain based Decentralized PKI. According to the determined design goals an implementation will be carried out as a proof of concept to evaluate the effectiveness of the solution.

As we are following the experimental research approach and following steps are taken to solve the identified research objectives

1. Explore the current functionalities of a Certification Authority and different problems caused due to the Certification Authority being the central point of failure in the PKI.
2. Identifying design goals that needed to be achieved to have the full functionality of a PKI without the presence of the Certification Authority (Which will solve the identified problems). Use cases for evaluation are also defined in this stage.
3. Design the prototype of the proposed PKI considering the identified goals.
4. Identifying necessary tools and technologies to be use in the prototype implementation.
5. Carrying out the prototype implementation.



- 
6. Evaluating the implemented prototype based on the identified evaluation use cases. After the evaluation will measure the effectiveness of the proposed system and determine whether it is possible to have an alternative to Certification Authorities based on Blockchain based Decentralized PKI.

### **3.1 Knowledge gathered from previous researches**

Below properties gathered from the literature are incorporated as characteristics in this research. In the literature when referring Blockchain based PKI it was identified that the Blockchain technology was mostly used as an identity retention medium. In this project with the inspiration of the ACME protocol, on-chain domain validation is performed with the Blockchain technology tightly coupling the trust with identity retention. During on-chain domain validation process, the validation of the registering domain will be performed within the blockchain with the help of blockchain middleware and will not be using any external attestors. Following inspirations are gathered together to propose a novel blockchain based decentralized PKI.

1. Web of Trust - Decentralization
2. Log based public ledger - Keeping a publicly accessible log
3. Lets Encrypt - Automatic domain validations
4. Namecoin - First to file paradigm
5. TrustCA - Making things transparent
6. Chainlink - Blockchain middleware which makes smart contracts smarter
7. Tor Hidden Service Names - Self authenticating Onion names
8. X.509 Certificate - Using Subject Public Key Info attribute

---

## 3.2 Solving the identified problems

- Problem 1 - CA compromised. Hackers issuing rouge certificate using the private key of the CA.
  - Solution 1 - Alternative to CA

If we are to eliminate the CA, all the tasks performed by the CA should be achieved by in a better way. We will be incorporating the Ethereum smart contract principles and self-sovereign identity aspects to this. Identity retention will be achieved by using the smart contract and the key generation and certificate generation( for backwards compatibility) is perform by the owner of the domain (Domain owner - Owner of the domain and the server’s public private key pair).
  - Solution 2 - Identity Retention

Every domain will own a public key private key pair. Hash of this public key will be saved on-chain along with the domain name, transaction date and the validity as a Boolean. Domain owner should own an Ethereum wallet and this information will also be stored along with the previously mentioned information as it is needed during the revocation process. Also, the proposing model will have an identity renewal mechanism where domain owner can update the on-chain information if needed.
  - Solution 3 - Fist to File Paradigm

First registration of the domain public key mapping will succeed and the second attempt of using the same domain will fails unless the request originated from the domain owner.
- Problem 2 - CA accidentally issuing erroneous certificate to customers enabling certificate to act as CA themselves.
  - Solution 1 - Transparency

By using Ethereum smart contracts it is possible for anyone to save

---

and retrieve identities in the system. All the actions performed are transparent hence fraudulent behavior is detected. Because of this feature it is not necessary to rely on a central authority.

– Solution 2 - Backwards Compatibility

This project only considers the SSL certificate domain and to achieve the backwards compatibility, a self-signed X.509 certificate is generated to support current client server TLS communication protocols.

• Problem 3 - Slow certificate signing process

– Solution - On chain domain validation

Inspired by the domain validation mechanism in ACME protocol, on-chain domain validation is performed to tighten the security. Ownership of the domain and the ownership of the private key is verified, and identity retention process will occur only if the above validations are successful.

• Problem 4 - Trusted root CAs are hardcoded to browsers and Operating Systems. In-order to remove a trusted CA a security updates should be pushed.

– Solution - Fast and Reliable Verification

Due to the versatility of the identity retention and the backwards compatibility it is possible for incorporate any suitable approach in the verification stage. As the project scope is in the SSL domain, a browser extension is in place to verify the identity.

### 3.3 Selecting the Blockchain Technology

In the literature review we have come across different types of Blockchain technologies and PKI requirements. In our PKI design we preserve the PKI basic principles by carefully choosing the suitable Blockchain technologies to overcome the identified problems.

- 
- Permission type
    - Available choices - Public and Private
    - Selected - Public

Public Key Infrastructure should be publicly accessible by everybody, hence the suitable blockchain permission type is public as all the data is public. It is not possible to achieve this by selecting a private blockchain. In a private blockchain only selected participants can be involved.

- Blockchain Type
  - Available choices - Using Established Network, Starting from scratch
  - Selected - Using Established Network

Security of the implementing PKI is extremely important. As established networks like Bitcoin, Ethereum are well tested by the community around the world hence security the security of these networks is solid. Creating a blockchain network from scratch has its own advantages but, since the security aspect can have a negative impact and to have public acceptance 'Using Established Network' is selected.

- Storage Type
  - Available choices - On-chain, Off-chain
  - Selected - On-chain

By considering the available information to store, it is possible to directly store the data in the blockchain. When storing data off-chain always impose a security risk but there are distributed options (IPFS) are also available for this but since the data to be stored in this design is not increasing large storing data on-chain option is selected.

- Privacy
  - Available choices â Storing a reference, Permissioned (Effectively private), PET

---

– Selected - None

It is possible to incorporate privacy preserving technologies to the Blockchain based PKI solution, but Public key information should be publicly accessible hence privacy of the storing data is not considered in this solution.

As mentioned above, since we have selected to use publicly available blockchain networks next decision is to choose such network which will be suitable to design a blockchain based public key infrastructure.

Bitcoin and Ethereum are the most publicly used blockchain networks. We have carefully weighed in the pros and cons of using these networks and after careful examination selected Ethereum to proceed with.

### **3.4 Selecting the Blockchain Network**

Namecoin was a decentralized name registration application developed based on the Bitcoin network. In this project what they have done is building an independent network and a protocol on top of bitcoin network. For that they had to create a fork of the Bitcoin main network.

If we are to use this kind of approach the implementation and the testing will be difficult as it is needed to bootstrap an independent blockchain network which should be thoroughly tested and verified on the state transitioning and the network code. The main disadvantage of this kind of approach that it is that not possible to incorporate simple payment verification features as each protocol would have transactions related to the context of their own protocol and it will be needed to implement a simple payment verification protocol which will have to backtrack all the way back to the genesis of the Bitcoin blockchain and check the validity of a certain transaction.

The scripting allowed in bitcoin only allows a weak version of smart contracts. Limitation of the scripting language in Bitcoin are

- Lack of Turing completeness
- Value blindness

- 
- Lack of state
  - Block chain blindness

Due to restrictions of the UTXO model in Bitcoin when implementing decentralized applications, Ethereum uses an account-based model which developers can use to build different kind of consensus-based applications. As Ethereum has a built-in Turing complete programming language it has provided wider opportunities for developers to develop smart contract with decentralized applications with their own transactions format and transitions and rules of ownership. Along with the account-based model in Ethereum, in addition to the Turing complete scripting language, value awareness and blockchain awareness and state are added advantages.

Due to above reasons Ethereum was chosen as the blockchain network for this project.

### 3.5 Our Approach

The Domain Owner initiates a Domain identity registration request (Similar to Certificate Signing request in the standard PKI). This could be performed by a system administrator and he should have an Ethereum wallet with some Ether. He will initially generate a key pair for the web server(using a suitable cryptographic algorithm) and a self-sign X.509 certificate. Then he will setup the webserver with the newly created self-sign certificate. After creating the certificate, provided public key registration program of this project will be initiated and the relevant request to the smart contract will be invoked and the request will be signed by using the Ethereum wallet private key. Request will be sent to the smart contract reside in the Ethereum network and it will perform necessary validations and if it is fulfilled information will be stored on chain. This smart contract has the capability for the identity retention, revocation and domain validation. All the information is stored on-chain.

Suggested PKI system is decentralized which is operating in large P2P network hence it is not possible to hack into this infrastructure to perform fraudulent activities.

---

During the identity verification stage (accessing a web site using a web browser), standard SSL handshake will be proceed as the self-sign certificate is created to support the backward compatibility and web browser extension that we create will make a request to the smart contract and check the hash of the received public key and the hash resides in the blockchain network and determine whether the received public key is valid or not.

# Chapter 4

## Research Design - TLSChain

This chapter will include the design of the proposing decentralized PKI scheme named TLSChain.

The aim of TLSChain is to map a public key to each domain and place the public key hash and the domain name in the Ethereum blockchain. The corresponding public and private key pair will be used for the TLS communication. During verification public key hash of the domain will be retrieved from the blockchain and it will be compared with the received X.509 self-signed certificate.

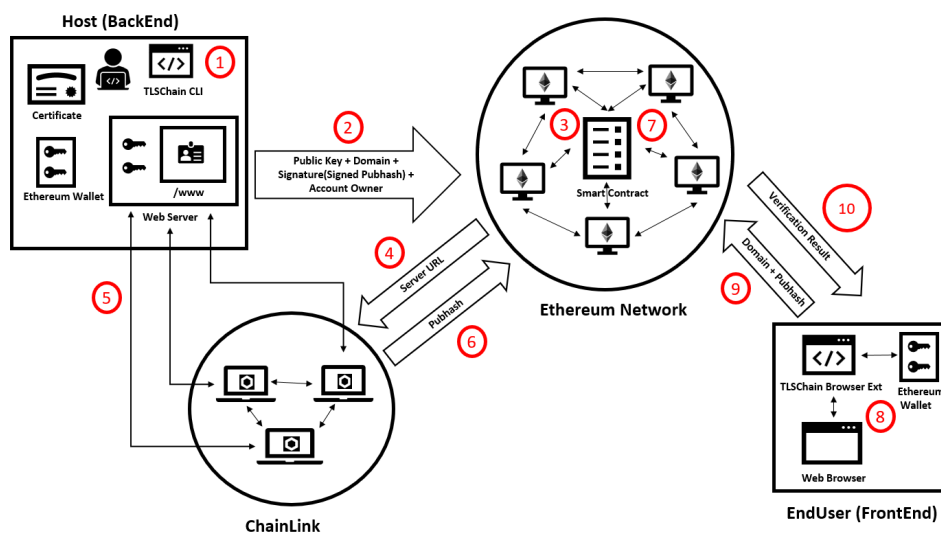


Figure 4.1: TLSChain Overview



---

## 4.1 TLSChain Overview

Following is a description of the steps mentioned in the Figure 4.1

1. **Step1** - Creating the X.509 Self-Sign Certificate, Placing the public key hash in the public folder
2. **Step2** - TLSChainCLI contact the TLSChain smart contract via Web3
3. **Step3** - TLSChain smart contract performing the validations
4. **Step4** - TLSChain smart contract contact ChainLink nodes
5. **Step5** - ChainLink Nodes will access the webserver and fetch the public key hash value
6. **Step6** - ChainLink Nodes will present the fetched value to the TLSChain smart contract
7. **Step7** - Smart contract will perform the identity retention
8. **Step8** - An end user will try to access a web page and he use TLSChain extension to check the validity
9. **Step9** - TLSChain extension contact the TLSChain smart contract via Web3
10. **Step10** - Smart contract will provide the requested details to TLSChain web extension

## 4.2 Main Components

### 4.2.1 Domain Owner

Domain owner is the person who will execute the TLSChain CLI. Domain Owner should own an Ethereum account to execute the TLSChain CLI. This same account should be use when performing this domain related activities.

---

### **4.2.2 TLSChain CLI**

This is a command line tool provided by the TLSChain which will perform the functionalities mentioned in Section 4.3. This tool will be written in Python programming language using Web3 libraries.

Domain owner should create an asymmetric key pair and a X.509 self-signed certificate for the web server TLS communication prior using the TLSChain CLI.

### **4.2.3 Chain Link**

Chain Link will act as a blockchain middle tier and it will support the smart contract in domain validation stage of the domain public key registration. Chain Link nodes will read the value exposed in the web server public folder and then come into a consensus and present the fetched value to the smart contract to proceed with the validation. A Chain Link job is configured to fetch information of from the Webserver.

### **4.2.4 TLSChain Smart Contract**

The smart contract of TLSChain will contain functionalities to Register a Domain, Update a public key for a Domain, Revoke a public key for a Domain. Smart contract will be written in the solidity programming language.

### **4.2.5 TLSChain Web Extension**

End user will try to communicate with a Domain which was registered with the TLSChain. Standard TLS communication will take place when End User starts communicating with the Webserver. Client browser will receive the X.509 self-sign certificate which will contain the public key information. TLSChain Web Extension will compare the received domain and the public key with the information in the blockchain and determine whether the received information is authentic.

---

## 4.3 TLSChain CLI Functionalities

### 4.3.1 Main functionalities of the TLSChain CLI

- Generate hash of the public key, to be placed it in the public folder of the Web Server.
- TLS Chain CLI will have the capability to invoke the functionality of the smart contract
  - Domain public key Registration - Register a domain with the public key
  - Domain Public Key Revoke - Revoke a domain public key registration
  - Domain Public Key Renewal - Renew domain public key registration and validity
  - Domain Public Key Validity - Check the validity given a X.509 certificate

## 4.4 Smart Contract Functionalities

### 4.4.1 Domain public key registration

Before the identity retention, the provided information goes through a validation process to make sure the legitimacy of the information. Refer [Figure 4.2](#)

#### 4.4.1.1 Validation points

- Whether this is an already registered domain - If this is an already registered domain, the domain registration process will be terminated. This validation is in place so that an attacker will not be able to re-register a domain with their public key. First to file paradigm is used.
- Signature validation - This validation is performed to check whether the domain owner actually has the private key corresponding to the public key that he wishes to register with TLSChain. Below information sent to the

---

smart contract.

$S\text{-Pub}, \text{Signature}(S\text{-Priv}(\text{Hash}(S\text{-Pub})))$

With the provided information signature is validated and prove that domain owner has the Private key related to the sent Public key.

- S-Pub - Sever Public Key
- S-Priv - Server Private Key
- Hash - Hash function
- Domain Validation - This validation is performed to verify that the domain owner is actually owns the domain and the domain is also accessible. To validate this we are taking advantage of the fact that only the domain owner is able to place the hash of the public key in the public folder in the configured web server (eg:- TLSChain.json).

Smart contract will try to read the value placed (Hash of the public key) in the public folder of the Webserver and compare it with the received public key. As Ethereum nodes are deterministic smart contracts are not able to access the web server public folder directly. For this Blockchain Oracle ChainLink is used.

Chain-link nodes will read the value exposed by the public folder of the web server and come into a consensus and then present it to the smart contract to validate. Smart contract will compare the value presented by the ChainLink and the received public key to make sure that the Domain Owner who requested the domain public key registration is the owner of the domain.

After success full validation of the received information below will be retained in the smart contract.

- Public key Hash - string
- Domain Name - string

- 
- Registration Date - uint256
  - Domain Owner - address
  - Validity - enum State{ Invalid,Valid }

#### **4.4.2 Domain Public Key Revoke**

Revocation can only be performed by the Domain Owner. Before proceeding the revocation request, the Ethereum wallet address of the revoke requesting transaction and the Domain owner information in the smart contract mapping is compared. If the request came from the original owner revoke request is carried out and the Domain and public key mapping will be invalidated. Domain Owner can use this revoke option in following situations

- If the private key of the Webserver is compromised, he can use the revoke option to remove the mapping.
- Domain is no longer functioning

#### **4.4.3 Domain Public Key Renewal**

Domain public key renewal can only be performed by the Domain owner. Domain will be valid for one year from the registration date. By using the Domain Public Key Renewal option, the registration date will be updated to the transaction date.

Also the domain owner has the the capability to change a public key mapped to a domain.

#### **4.4.4 Retrieve Validity**

This can be performed by anyone not necessarily the domain owner. If they receive a X.509 certificate they can use this function to verify the domain that the respective public key is mapped to.

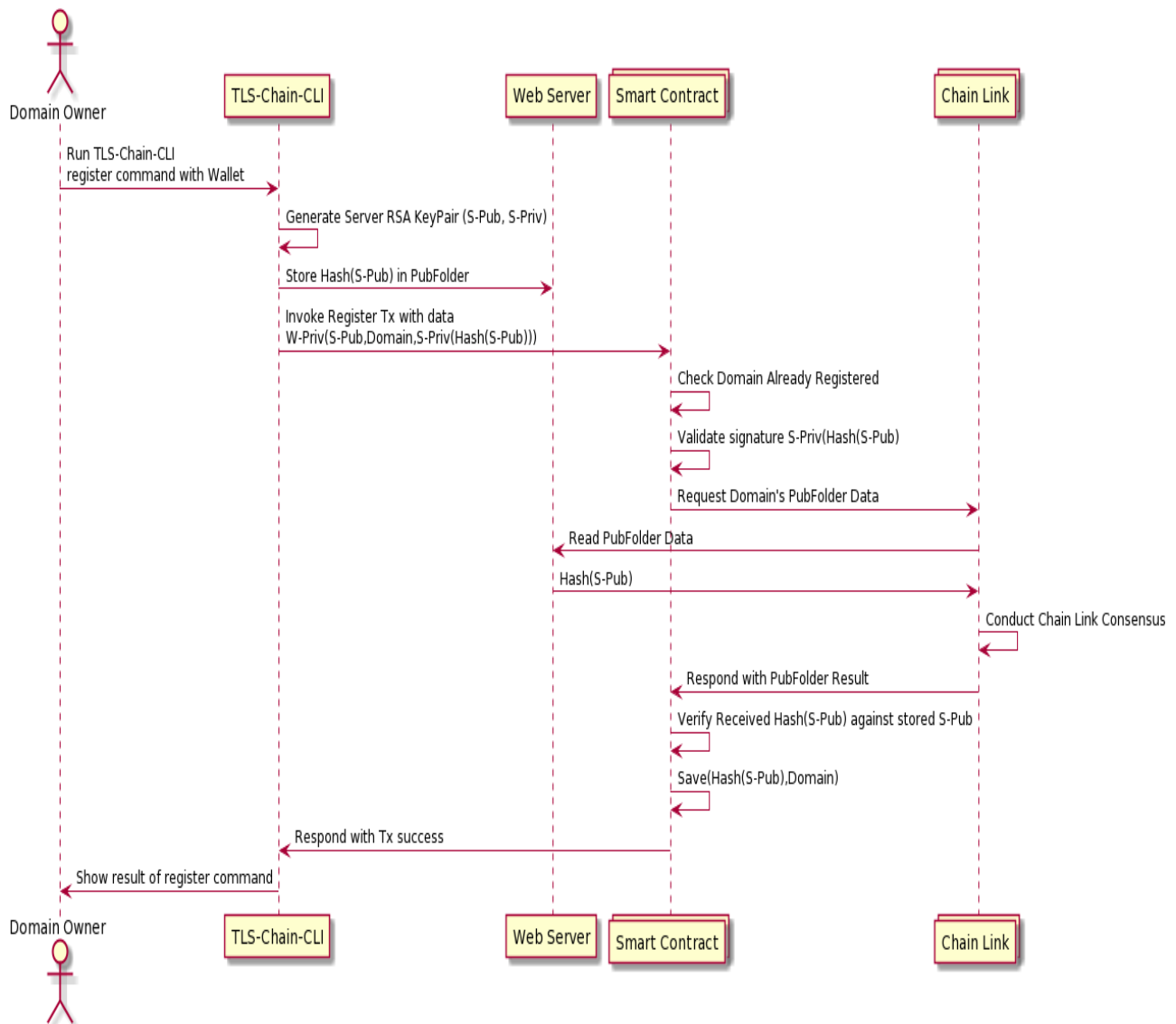


Figure 4.2: Domain Public Key Registration

# Chapter 5

## Implementation

Based on the research design we have implemented a proof of concept to verify whether it is possible to build Public Key Infrastructure based on blockchain technologies.

### 5.1 TLSChain Smart Contract

Smart contract was developed using the Solidity Programming language to communicate with the Ethereum blockchain network.

#### 5.1.1 Storing Domain Public Key Mapping

##### 5.1.1.1 Information Storage

We are storing the Domain and the Public key information as mappings in solidity as mappings in solidity gives greater flexibility. In TLSChain smart contract we are using two such mappings.

- **cerMapping** - Mapping between the public key hash and the domain
- **domainMapping** - Mapping between the domain and the public key hash

---

### 5.1.1.2 Struct Certificate

To keep the certificate information intact, we are using a struct names Certificate to store the necessary details.

- **domain** - Domain name
- **dateTime** - Date time of the transaction added to the blockchain
- **state** - Current state of the public key and the domain mapping
- **owner** - Owner of the domain. The initial Ethereum account registered the public key domain mapping

### 5.1.1.3 Solidity Code

```
enum State {Invalid,Valid,NotRegistered}
struct Certificate {
    string domain;
    uint256 dateTime;
    State state;
    address owner;
}
mapping(string => Certificate) cerMapping;
mapping(string => string) domainMapping;
```

## 5.1.2 Domain Registration Function

In the TLSChain smart contract we have a function names **requestRegister** which will register the initial domain public key mapping.

### 5.1.2.1 requestRegister Function

Below are the six parameters requested for the **requestRegister** function in the smart contract. This can be invoked by the TLSChain CLI very easily and then it is only necessary to provide the generated self-sign Certificate details, web



---

server private key details and the domain name. TLSChain will create the needed transaction and will invoke this smart contract method.

- **sPubHash** - Hash of the web server public key extracted from the certificate
- **domain** - Name of the domain to register
- **message** - Signature verification - Hex value of the hash of the public key extracted from the certificate
- **exponent** - Signature verification - Exponent of the RSA keys
- **modulus** - Signature verification - Modulus of the RSA keys
- **signature** - Signature verification - Generated signature by signing the web server public key hash using the web server private key.

This function should be only used, when the initial domain public key mapping registration. After the initial registration even the owner of the domain will not be able to execute this function again. Any changes after the initial registration should be using using **requestRenewDomainValidity** or **requestRenewDomainPubKey** functions and only the domain owner is able to perform such changes.

#### 5.1.2.2 requestRegister Validations

Several validations are performed before storing certificate details in the blockchain.

1. Domain and the public key is not used for registration before
2. Validate that the private key corresponding to the public key that the domain is going get registered is available to the domain owner
3. Information displayed in the server public folder is similar to the provided public key hash

---

### 5.1.2.3 requestRegister Solidity Code

```
function requestRegister(
    string memory sPubHash_,
    string memory domain_,
    bytes memory message_,
    bytes memory exponent_,
    bytes memory modulus_,
    bytes memory signature_
) public {
    uint valid_sig_ = retrieveResult(message_,signature_,exponent_,modulus_);
    require (valid_sig_ == 0, "Signature Validation Failed");
    bytes memory tempEmptyPubHash = bytes(domainMapping[domain_]);
    require (tempEmptyPubHash.length == 0, "This Domain has a Certificate Attached");
    bytes memory tempEmptyDomain = bytes(cerMapping[sPubHash_].domain);
    require (tempEmptyDomain.length == 0, "This Certificate has a Domain Attached");
    Chainlink.Request memory req = buildChainlinkRequest(
        stringToBytes32(JOBID),
        address(this),
        this.fulfillRequestRegister.selector
    );
    string memory URL= concat(concat("http://",domain_),"/TLSChain.json");
    req.add("get", URL);
    req.add("path", "TLSChain");
    RegInputDetails memory regInputDetails = RegInputDetails({
        sPubHash: sPubHash_,
        oldPubHash: "",
        domain: domain_,
        owner: msg.sender});
    bytes32 requestId = sendChainlinkRequestTo(ORACLE_ADDRESS, req, ORACLE_PAYMENT);
    InputRequests[requestId] = regInputDetails;
}

function fulfillRequestRegister(bytes32 _requestId, bytes32 _result)
    public recordChainlinkFulfillment(_requestId)
{
    bytes32 fromSever = _result;
    register(_requestId,fromSever);
}

function register(bytes32 requestId_, bytes32 fromSever_) public {
    string sPubHash_ = InputRequests[requestId_].sPubHash;
    string domain_ = InputRequests[requestId_].domain;
    address owner_ = InputRequests[requestId_].owner;
    require(fromSever_ == stringToBytes32(sPubHash_), "Server Value Not Matching");
    cerMapping[sPubHash_] = Certificate({
        domain: domain_,
        dateTime: now,
        state: State.Valid,
        owner: owner_
    });
}
```

---

```

    });
    domainMapping[domain_] = sPubHash_;
}

```

### 5.1.3 Renew Domain Public Key Validity Function

Domain public key registration will expire after 365 days hence, it is possible for the domain owner to extend the validity period. When the domain owner execute this function, the registration date will be updated to the current date.

Signature and exposed public key hash validations will be performed to make sure that the domain owner still have the access to the claimed domain.

Several validations are performed before renewing certificate details in the blockchain.

1. Transaction sender address is similar to the domain owner address
2. Validate that the private key corresponding to the public key that the domain is going get registered is available to the domain owner
3. Information displayed in the server public folder is similar to the provided public key hash

Parameters to invoke this function is similar to **requestRegister** function.

#### 5.1.3.1 requestRenewDomainValidity Solidity Code

```

function requestRenewDomainValidity(
    string memory sPubHash_,
    string memory domain_,
    bytes memory message_,
    bytes memory exponent_,
    bytes memory modulus_,
    bytes memory signature_
) public {
    uint valid_sig_ = retrieveResult(message_, signature_, exponent_, modulus_);
    require (valid_sig_ == 0, "Signature Not Matching");
    require (msg.sender == cerMapping[sPubHash_].owner, "Only Domain Owner
    Can Perform This Function");
    Chainlink.Request memory req = buildChainlinkRequest(
        stringToBytes32(JOBID),
        address(this),

```

---

```

        this.fulfillRequestRenewDomainValidity.selector
    );
    string memory URL = concat(concat("http://", domain_), "/TLSChain.json");
    req.add("get", URL);
    req.add("path", "TLSChain");
    RegInputDetails memory regInputDetails = RegInputDetails({
        sPubHash: sPubHash_,
        oldPubHash: "",
        domain: domain_,
        owner: msg.sender});
    bytes32 requestId = sendChainlinkRequestTo(ORACLE_ADDRESS, req, ORACLE_PAYMENT);
    InputRequests[requestId] = regInputDetails;
}

function fulfillRequestRenewDomainValidity(bytes32 _requestId, bytes32 _result)
    public recordChainlinkFulfillment(_requestId)
{
    bytes32 fromSever = _result;
    renewDomainValidity(_requestId, fromSever);
}

function renewDomainValidity(bytes32 requestId_, bytes32 fromSever_) public {
    string sPubHash_ = InputRequests[requestId_].sPubHash;
    string domain_ = InputRequests[requestId_].domain;
    address owner_ = InputRequests[requestId_].owner;
    require(fromSever_ == stringToBytes32(sPubHash_), "Server Value Not Matching");
    cerMapping[sPubHash_] = Certificate({
        domain: domain_,
        dateTime: now,
        state: State.Valid,
        owner: owner_
    });
}
}

```

### 5.1.4 Update Domain Public Key Function

If domain owner needs to change the web server certificate used to configure the webserver, he can change the already registered mapping with this function. Only the domain owner can preform this function and he should provide the previous web Certificate as well.

Several validations are performed before renewing certificate details in the blockchain.

1. Transaction sender address is similar to the domain owner address

- 
2. Domain name should be same as the previous mapping
  3. Validate that the private key corresponding to the public key that the domain is going to get registered is available to the domain owner
  4. Information displayed in the server public folder is similar to the provided public key hash

Parameters to invoke this function is similar to **requestRegister** function and as an addition the previous public key hash should be provided.

#### 5.1.4.1 requestRenewDomainPubKey Solidity Code

```
function requestRenewDomainPubKey(
    string memory sPubHash_,
    string memory oldSPubHash_,
    string memory domain_,
    bytes memory message_,
    bytes memory exponent_,
    bytes memory modulus_,
    bytes memory signature_
) public {
    uint valid_sig_ = retrieveResult(message_, signature_, exponent_, modulus_);
    require (valid_sig_ == 0, "Signature Not Matching");
    require (msg.sender == cerMapping[oldSPubHash_].owner, "Only Domain Owner
    Can Perform This Function");
    require ((keccak256(bytes(cerMapping[oldSPubHash_].domain))) ==
    (keccak256(bytes(domain_))));
    Chainlink.Request memory req = buildChainlinkRequest(
        stringToBytes32(JOBID),
        address(this),
        this.fulfillRequestRenewDomainPubKey.selector
    );
    string memory URL = concat(concat("http://", domain_), "/TLSChain.json");
    req.add("get", URL);
    req.add("path", "TLSChain");
    RegInputDetails memory regInputDetails = RegInputDetails({
        sPubHash: sPubHash_,
        oldPubHash: oldSPubHash_,
        domain: domain_,
        owner: msg.sender});
    bytes32 requestId = sendChainlinkRequestTo(ORACLE_ADDRESS, req, ORACLE_PAYMENT);
    InputRequests[requestId] = regInputDetails;
}

function fulfillRequestRenewDomainPubKey(bytes32 _requestId, bytes32 _result)
```

---

```

    public recordChainlinkFulfillment(_requestId)
{
    bytes32 fromSever = _result;
    renewDomainPubKey(_requestId,fromSever);
}

function renewDomainPubKey(bytes32 requestId_, bytes32 fromSever_) public {
    string sPubHash_ = InputRequests[requestId_].sPubHash;
    string oldSPubHash_ = InputRequests[requestId_].oldPubHash;
    string domain_ = InputRequests[requestId_].domain;
    address owner_ = InputRequests[requestId_].owner;
    require(fromSever_ == stringToBytes32(sPubHash_), "Server Value Not Matching");
    revoke(oldSPubHash_);
    cerMapping[sPubHash_] = Certificate({
        domain: domain_,
        dateTime: now,
        state: State.Valid,
        owner: owner_
    });
}

```

### 5.1.5 Revoke Domain Public Key Registration Function

If a domain owner wants to revoke the public key domain mapping that he already registered to the blockchain this function can be used. Only the domain owner is able to perform this function.

Only parameter is needed is the hash of the public key

#### 5.1.5.1 revoke Solidity Code

```

function revoke(string pubHash_){
    require (msg.sender == cerMapping[pubHash_].owner, "Only Domain Owner
Can Perform This Function");
    cerMapping[pubHash_].state = State.Invalid;
    cerMapping[pubHash_].dateTime = block.timestamp;
}

```

### 5.1.6 Retrieve Domain Public Key Information Function

This function can be invoked by anyone to check the validity of a domain and a public key.

Only parameter is needed is the hash of the public key

---

### 5.1.6.1 retrieve Solidity Code

```
function retrieve(string pubHash_)
    public
    view
    returns (string, State, uint256, address)
{
    if ((bytes(cerMapping[pubHash_].domain)).length == 0){
        return ('No Associated Domain for this Certificate', cerMapping[pubHash_].state, 0, 0);
    }else{
        return (cerMapping[pubHash_].domain, cerMapping[pubHash_].state, cerMapping[pubHash_].d
cerMapping[pubHash_].owner);
    }
}
```

## 5.2 TLSChain CLI

With TLSChainCLI domain owner can invoke all the smart contract function by giving simple parameters. TLSChainCLI is written using Python scripting language and use Web3 API to communicate with the Ethereum blockchain network. TLSChain CLI will convert the given parameter values to be compatible with the smart contract function, so that it will provide greater flexibility to the end users.

### 5.2.1 TLSChain CLI Available Commands

Following functions are available from the TLSChain CLI. Help option will provide a guidance which methods are available and what parameters should be provided.

Following functions are available in the TLSChain CLI

- getPubHash
- registerDomain
- retriveValidityCer
- retriveValidityHash

```
D:\TLSChain\ServerScript>TLSChainCLI.exe -h
usage: An Alternative to CA using Blockchain based Decentralized PKI

*****
* This is TLSChainCLI Decentralized PKI *
*****

positional arguments:
  Commands  'getPubHash -c','registerDomain -c -p -d -wa -wp','retriveValidityCer -c','retriveValidityHash -hp','revokeDomain -c -wa
            -wp','renewDomainValidity -c -p -d -wa -wp','requestRenewDomainPubKey -c -oc -p -d -wa -wp'

optional arguments:
  -h, --help  show this help message and exit
  -hp HP      Public Key Hash
  -c C        Certificate File Name
  -oc OC      Old Certificate File Name
  -p P        Private Key File
  -d D        Domain Name
  -wa WA      Wallet Address
  -wp WP      Wallet Private Key

Copyrights @ Anne Fernando
D:\TLSChain\ServerScript>
```

Figure 5.1: TLSChainCLI Help Option

- revokeDomain
- renewDomainValidity
- requestRenewDomainPubKey

## 5.2.2 getPubHash Usage

Domain owners can use this command to generate the hash of the public key, which then should be exposed by the webserver by placing it in the public folder.

The return value of this function is also can be used to invoke the method **retriveValidityHash** where you can check the public key domain mapping validity by giving the public key hash.

The hash generated is a string which is Base64 encoded SHA-256 hash of the DER-encoded public key info.

This function will provide the generated public key hash to the domain owner and internally the modulus and the exponent also will be extracted from the provided certificate for internal functionalities.



---

### 5.2.2.1 getPubHash scope

This function can be executed by anyone. Open to public. This function will not change the state of the smart contract hence it will not cost gas for the usage.

### 5.2.2.2 getPubHash Parameters

- c - Path to the the Self-Signed SSL Certificate Generated

### 5.2.2.3 getPubHash Python Code

```
# Get the public key hash to place in the public folder of the server
def getPubHash(CERT_FILE):
    file_path = os.path.join(os.getcwd(), CERT_FILE)
    f = open(file_path, "r")
    cert = f.read()

    pubkey = RSA.importKey(cert)
    modulus = "{0:#0{1}x}".format(pubkey.n, 256)
    exponent = "{0:#0{1}x}".format(pubkey.e, 256)

    pub_key_obj = crypto.load_certificate(crypto.FILETYPE_PEM, cert).get_pubkey()
    pub_key = crypto.dump_publickey(crypto.FILETYPE_ASN1, pub_key_obj)

    m = hashlib.sha256()
    m.update(pub_key)
    digest = m.digest()
    encoded = base64.b64encode(digest)
    print(f'\nPublic Key Hash of Certificate {CERT_FILE} : {encoded}')

    return modulus, exponent, encoded
```

## 5.2.3 registerDomain Usage

A domain owner can use this function to register a domain with a public key attached to the generated self-sign webserver certificate. This function can be used only once.

### 5.2.3.1 registerDomain Scope

This function can be executed by anyone. Open to public. This function will change the state of the smart contract hence it will cost gas for the usage.

---

### 5.2.3.2 registerDomain Pre-Configurations

For a successful execution of this function the domain owner should do some prior configurations in the webserver.

1. Should obtain Ethereum Ether and configure a Wallet. The wallet private address should secure.
2. Creating a RSA Key based X.509 self-signed certificate
3. Obtain the Pubic Key Hash of the Generated Certificate using TLSChain CLI command **getPubHash**
4. Secure the generated RSA private key
5. Place the public key hash in the public folder of the webserver The name of the file should be "TLSChain.Json" with the following mapping.

```
{ "TLSChain":<Public Key Hash > }
```

eg:-

```
{ "TLSChain": "66XtQ+REJXHBaFGVsKPrWe6n4FJ14MRxLhmXDq3QgwM=" }
```

6. Configure the Webserver to use HTTPs

For more details of the prior configurations please refer to Chapter6 case study chapter.

### 5.2.3.3 registerDomain Parameters

- c - Path to the Self-Signed SSL Certificate Generated
- p - Path to the Private key of the Self-Signed SSL Certificate Generated
- d - Domain name needed to be associate with the public key
- wa - Wallet address which will contain Ether

- 
- wp - Wallet Private key to sign the transaction

#### 5.2.3.4 registerDomain Python Code

```
#Register a public key with a domain
def registerDomain(CERT_FILE,KEY_FILE, domain_name ,wallet_address ,wallet_private_key):

    connected_to_ganache,web3_client,contract = runConfig();
    modulus,exponent,encoded = getPubHash(CERT_FILE)
    signature = getSignature(KEY_FILE,encoded)

    if (connected_to_ganache):
        print(f'\nRegister Domain Public Key in TLSChain: {domain_name}')
        txn_dict = contract.functions.requestRegister(encoded, domain_name,
            encoded.hex(),exponent,modulus,signature).buildTransaction({
            'from': wallet_address,
            'gas': 2000000,
            'gasPrice': web3_client.toWei('50', 'gwei'),
            'nonce': web3_client.eth.getTransactionCount(wallet_address)
        })
        try:
            signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
            result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
            tx_receipt = web3_client.eth.waitForTransactionReceipt(result)
        except ValueError as e:
            print(e.args[0]['message'])

    time.sleep(10)
    retrieveValidityHash(encoded)
```

#### 5.2.4 retrieveValidityCer Usage

After registering the domain and the public key with the TLSChain it is possible to check the status of the registration using this command. If the end user has the public key hash he can use **retrieveValidityHash** command also. Both gives the same result. TLSChain CLI gives the flexibility to choose a command according to the available information.

##### 5.2.4.1 retrieveValidityCer scope

This function can be executed by anyone. Open to public. This function will not change the state of the smart contract hence it will not cost gas for the usage.

---

#### 5.2.4.2 `retriveValidityCer` Parameters

- `c` - Path to the Self-Signed SSL Certificate Generated

#### 5.2.4.3 `retriveValidityCer` Python Code

```
# Retrieve the validity using the certificate
def retriveValidityCer(CERT_FILE):

    connected_to_ganache,web3_client,contract = runConfig();
    modulus,exponent,encoded = getPubHash(CERT_FILE)

    if (connected_to_ganache):
        # Retrieve the value to verify
        retrieved_domain_name, domain_is_valid, dateTime, address =
        contract.functions.retrieve(encoded).call()
        print(f'\nQuery from Ethereum:')
        print(f'Domain Name: {retrieved_domain_name}')
        print(f'Domain Is Valid: {domain_is_valid}')
        if dateTime != 0:
            dateTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(dateTime))
            print(f'Registered Date: {dateTime}')
            print(f'Owner: {address}')
```

### 5.2.5 `retriveValidityHash` Usage

After registering the domain and the public key with the TLSChain it is possible to check the status of the registration using this command.If the end user has the SSL Certificate he can use `retriveValidityCer` command also. Both gives the same result. TLSChain CLI gives the flexibility to choose a command according to the available information.

#### 5.2.5.1 `retriveValidityHash` Scope

This function can be executed by anyone. Open to public. This function will not change the state of the smart contract hence it will not cost gas for the usage.

#### 5.2.5.2 `retriveValidityHash` Parameters

- `ph` - Hash of the public key. Generated from the TLSChain CLI `getPub-Hash` command.

---

### 5.2.5.3 retrieveValidityHash Python Code

```
# Retrieve the validity using the public key hash
def retrieveValidityHash(pubHash):
    connected_to_ganache,web3_client,contract = runConfig();
    # Retrieve the value to verify
    retrieved_domain_name, domain_is_valid, dateTime, address =
    contract.functions.retrieve(pubHash).call()
    print(f'\nQuery from Ethereum:')
    print(f'Domain Name: {retrieved_domain_name}')
    print(f'Domain Is Valid: {domain_is_valid}')
    if dateTime != 0:
        dateTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(dateTime))
    print(f'Registered Date: {dateTime}')
    print(f'Owner: {address}')
```

### 5.2.6 revokeDomain Usage

Domain owner might need to revoke the validity of the registered domain and the public key mapping. For this he can use this command.

#### 5.2.6.1 revokeDomain Scope

This function is restricted and can be executed only by the domain owner (Determined by the wallet address). Not open to public. This function will change the state of the smart contract hence it will cost gas for the usage.

#### 5.2.6.2 revokeDomain Parameters

- c - Path to the Self-Signed SSL Certificate Generated
- wa - Wallet address which will contain Ether
- wp - Wallet private key to sign the transaction

#### 5.2.6.3 revokeDomain Python Code

```
def revokeDomain(CERT_FILE,wallet_address,wallet_private_key):

    connected_to_ganache,web3_client,contract = runConfig ();
    modulus,exponent,encoded = getPubHash(CERT_FILE)
```

---

```

if (connected_to_ganache):
    print(f'\n Revoke Domain Public Key Validiy in TLSChain')
    txn_dict = contract.functions.revoke(encoded).buildTransaction({
        'from': wallet_address,
        'gas': 2000000,
        'gasPrice': web3_client.toWei('50', 'gwei'),
        'nonce': web3_client.eth.getTransactionCount(wallet_address)
    })
    try:
        signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
        result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
        tx_receipt = web3_client.eth.waitForTransactionReceipt(result)
    except ValueError as e:
        print(e.args[0]['message'])

    time.sleep(10)
    # Retrieve the value to verify
    revokedDetails(encoded)

```

## 5.2.7 renewDomainValidity Usage

After a revocation or automatic invalidation after 365 days, the domain owner might need to renew the validity of the domain and the public key mapping. For this only domain owner can use this command.

### 5.2.7.1 renewDomainValidity Scope

This function is restricted and can be executed only by the domain owner (Determined by the wallet address). Not open to public. This function will change the state of the smart contract hence it will cost gas for the usage.

### 5.2.7.2 renewDomainValidity Parameters

- c - Path to the Self-Signed SSL Certificate Generated
- p - Path to the Private key of the Self-Signed SSL Certificate Generated
- d - Domain name needed to be associate with the public key
- wa - Wallet address which will contain Ether
- wp - Wallet Private key to sign the transaction

---

### 5.2.7.3 renewDomainValidity Python Code

```
#Renew a domain valifiy
def renewDomainValidity(CERT_FILE,KEY_FILE, domain_name ,wallet_address ,wallet_private_key):

    connected_to_ganache,web3_client,contract = runConfig();
    modulus,exponent,encoded = getPubHash(CERT_FILE)
    signature = getSignature(KEY_FILE,encoded)

    if (connected_to_ganache):
        print(f'\nRenew Domain Public Key Validiy in TLSChain: {domain_name}')
        txn_dict = contract.functions.requestRenewDomainValidity(encoded, domain_name,encoded.hex(),e
            'from': wallet_address,
            'gas': 2000000,
            'gasPrice': web3_client.toWei('50', 'gwei'),
            'nonce': web3_client.eth.getTransactionCount(wallet_address)
        })
        try:
            signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
            result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
            tx_receipt = web3_client.eth.waitForTransactionReceipt(result)
        except ValueError as e:
            print(e.args[0]['message'])

        time.sleep(10)
        retrieveValidityHash(encoded)
```

### 5.2.8 requestRenewDomainPubKe Usage

After a domain public key mapping registration, the domain owner may need to change the certificate that the web server is configured. To change the public key mapping of a particular domain, following method can be used.

#### 5.2.8.1 requestRenewDomainPubKe Scope

This function is restricted and can be executed only by the domain owner (Determined by the wallet address). Not open to public. This function will change the state of the smart contract hence it will cost gas for the usage.

#### 5.2.8.2 requestRenewDomainPubKey Parameters

- c - Path to the Self-Signed SSL Certificate Generated

- 
- p - Path to the Private key of the Self-Signed SSL Certificate Generated
  - d - Domain name needed to be associate with the public key
  - wa - Wallet address which will contain Ether
  - wp - Wallet Private key to sign the transaction

### 5.2.8.3 requestRenewDomainPubKey Python Code

```
#Renew a domain public key
def requestRenewDomainPubKey(CERT_FILE, OLD_CERT_FILE, KEY_FILE, domain_name, wallet_address,
                             wallet_private_key):

    connected_to_ganache, web3_client, contract = runConfig();
    modulus, exponent, encoded = getPubHash(CERT_FILE)
    signature = getSignature(KEY_FILE, encoded)

    omodulus, oexponent, oencoded = getPubHash(OLD_CERT_FILE)

    if (connected_to_ganache):
        print(f'\nRenew Public for a Domain in TLSChain: {domain_name}')
        txn_dict = contract.functions.requestRenewDomainPubKey(encoded,
            oencoded, domain_name, encoded.hex(), exponent, modulus, signature).buildTransaction({
            'from': wallet_address,
            'gas': 2000000,
            'gasPrice': web3_client.toWei('50', 'gwei'),
            'nonce': web3_client.eth.getTransactionCount(wallet_address)
        })
        try:
            signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
            result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
            tx_receipt = web3_client.eth.waitForTransactionReceipt(result)
        except ValueError as e:
            print(e.args[0]['message'])

    time.sleep(10)
    retrieveValidityHash(encoded)
```

## 5.3 TLSChain Chainlink Usage

For the domain validation process explained in the chapter 4(4.3.1.1) in detail, we needed to incorporate blockchain middle tier. We have selected Chainlink for this and a Chainlink node(s) is setup.



---

End users will not be interacting with this and this setup is only used for internal functionalities.

### 5.3.1 TLSChain ChainLink Job Specification

The following job dedicated to TLSChain published to the Chainlink. The purpose is to read the content exposed by the given URL and extract only the information related to the given tag 'TLSChain' and present it to the smart contract.

#### 5.3.1.1 Job Specification Json

Address '0xc612350d871cda87fbfdafb2a9553b680707aca8' is the Chainlink oracle smart contract address it will be constant through out the this deployment of TLSChain.

```
{
  "initiators": [
    {
      "type": "runlog",
      "params": {
        "address": "0xc612350d871cda87fbfdafb2a9553b680707aca8"
      }
    }
  ],
  "tasks": [
    {
      "type": "httpget",
      "confirmations": null,
      "params": {}
    },
    {
      "type": "jsonparse",
      "confirmations": null,
      "params": {}
    },
    {
      "type": "ethbytes32",
      "confirmations": null,
      "params": {}
    },
    {
      "type": "ethtx",
      "confirmations": null,
      "params": {}
    }
  ]
}
```

```

    }
  ],
  "startAt": null,
  "endAt": null
}

```

For this deployment of TLSChain the job specification ID is '3d583fa977f14919b01ee09f26f1bbab' and it will be constant through out this deployment.

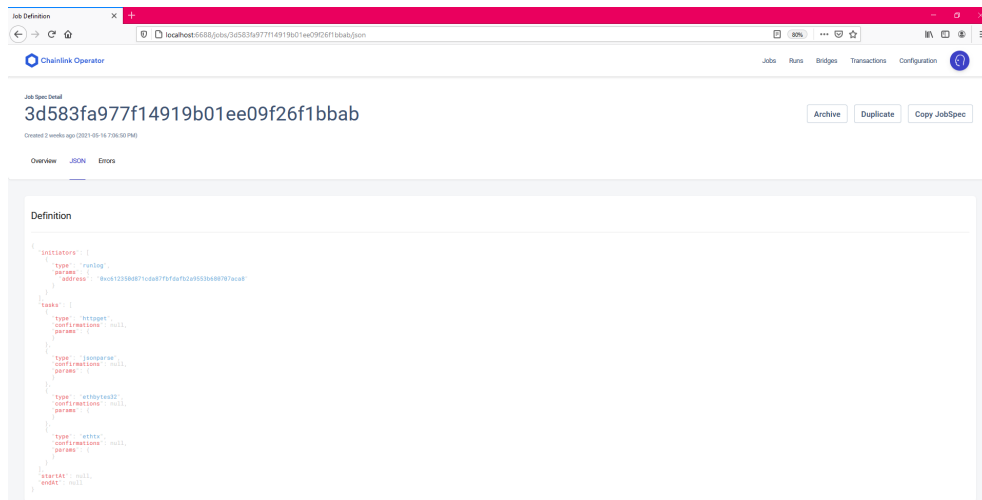


Figure 5.2: Published Chainlink Job Specification

### 5.3.1.2 Chainlink Job Reading Server Value

As per the Figure 5.3 chainlink job will read the public key hash value exposed in the sever and fetch the value and present it to the smart contract.

## 5.4 TLSChain Extension

TLSChain browser extension is developed only for the demonstration purposes to validate the concept that TLSChain is usable in browser side. Having a browser extension impose many security issues, hence as a future enhancement this should be an inbuilt browser function.

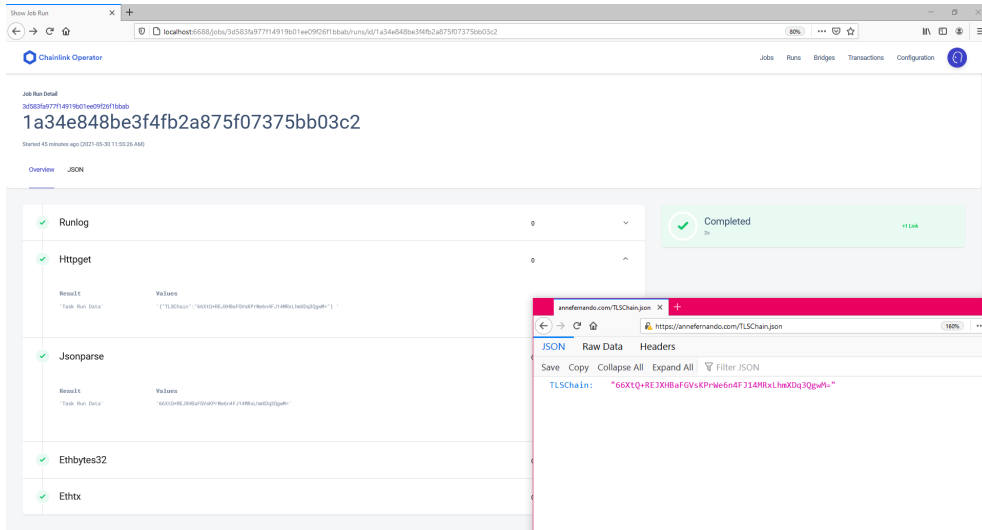


Figure 5.3: Chainlink Job Reading Server Value

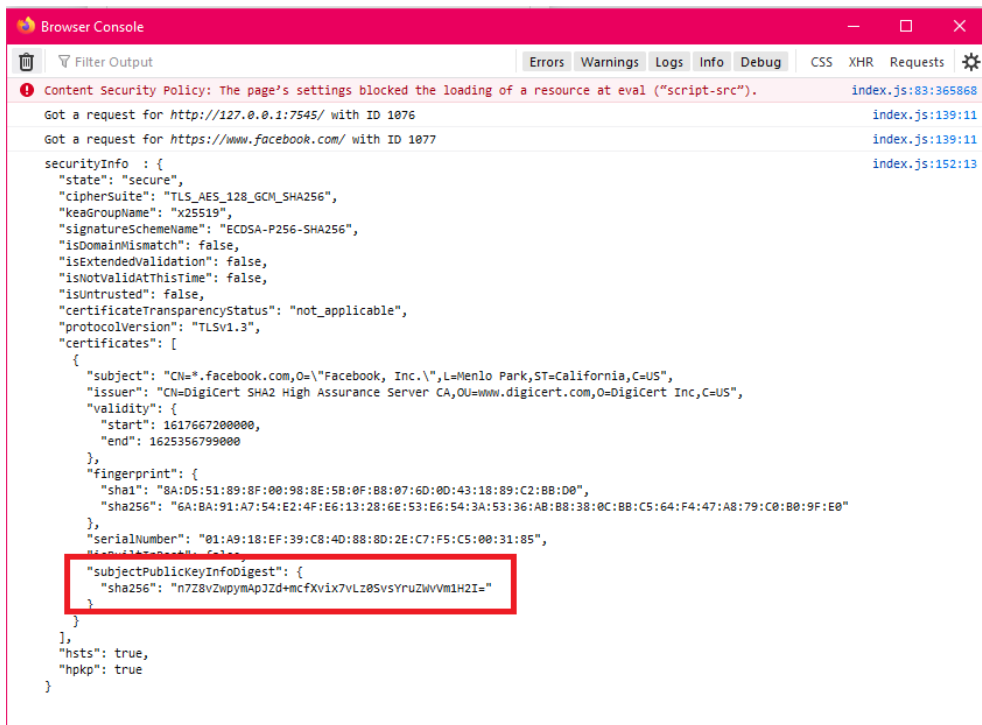


Figure 5.4: Certificate Details Provided by the Browser

---

During TLS Communication browser provided the public hash of the sever certificate, TLSChain take advantage of this and use it for the purpose of validating the received certificate against the TLSChain smart contract details.

### 5.4.1 TLSChain Extension Scope

This extension developed based on the TLSChain smart contract **retrieve** function. Hence this extension is publicly accessible by anyone and will not cost any gas.

### 5.4.2 TLSChain Java Script Code

Following is the code snippet for the retrieval of the validity. For the full code, please refer the Appendix.

```
if (securityInfo && securityInfo.certificates) {
  const certificateKeyHash = JSON.stringify(
    securityInfo.certificates[0].subjectPublicKeyInfoDigest.sha256,
    null,
    2
  );
  console.log(
    'securityInfo subjectPublicKeyInfoDigest: ${certificateKeyHash}'
  );
  console.log(web3.utils.toHex(certificateKeyHash));

  const KeyHash = certificateKeyHash.replace(/['"]+/g, "");

  const result = await smartContract.methods.retrieve(KeyHash).call();
  const {
    0: retrieved_domain_name,
    1: domain_is_valid,
    2: dateTime,
  } = result;

  console.log(result);
  console.log('retrieved_domain_name == ${retrieved_domain_name}');
  console.log('dateTime == ${dateTime}');
  console.log('domain_is_valid == ${domain_is_valid}');

  if (requestDetails.url === currentTabUrl) {
    let $statusIndicator = $("#status");
    $statusIndicator.removeClass("loader active");

    if (domain_is_valid === "1") {
```

```
    $statusIndicator.addClass("success animated pulse");
    $statusIndicator.text("Passed");

    console.log("Domain is valid");
} else {
    $statusIndicator.addClass("failure animated pulse");
    $statusIndicator.text("Failed");

    console.log("Domain is not valid");
}
}
}
}
```

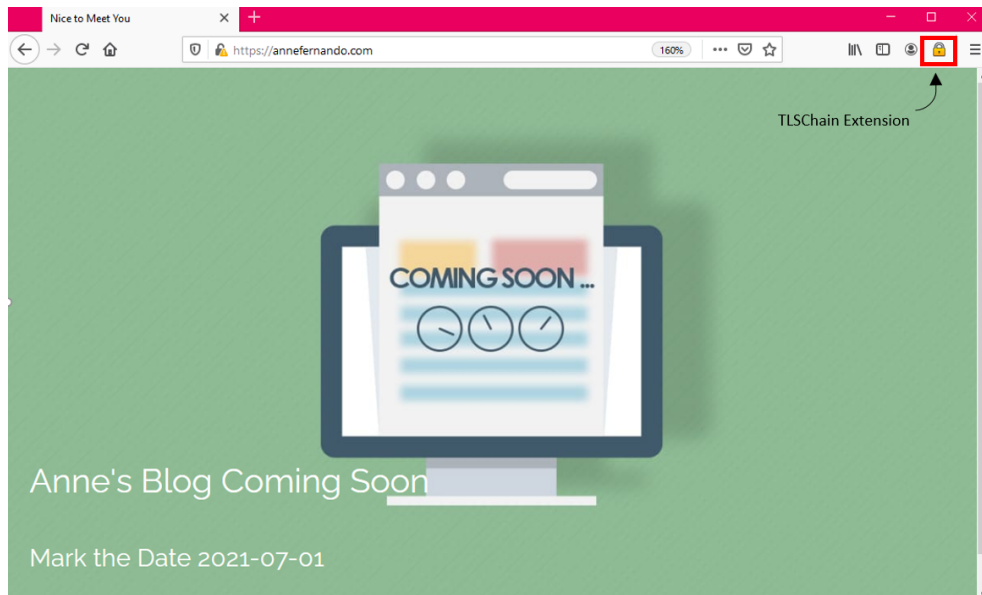


Figure 5.5: TLSChain Extension

# Chapter 6

## TLSCChain - A Use Case

This chapter will examine a use case where a domain owner configures his web server with TLSCChain to have HTTPS communication. It will be evident that a certification authority is not involved in this process.

### 6.1 TLSCChain Functional Flow

#### 6.1.1 Step 1 - Creating a Self-Sign Certificate

Domain owner can create a self-sign certificate by himself for the domain he owns. He can use Openssl commands for this purpose.

In this project we have created a X.509 certificate by using RSA algorithm for the domain **annefernando.com**. Certificate **annefernando.com.cer** and private key **annefernando.com.private.key** were created and extracted. Refer Figure [6.1](#).

#### 6.1.2 Step 2- Use TLSCChainCLI to get the Public Key Hash

After creating the certificate, TLSCChainCLI can be used to get the public key hash which should be placed in the public folder of the server. Path to the generated certificate should be given as a parameter. Refer Figure [6.2](#)

- Command to use - getPubHash

```
C:\Program Files\OpenSSL-Win64\bin\openssl.exe
OpenSSL> req -x509 -nodes -days 365 -newkey rsa:2048 -keyout annefernando.com.private.key -out annefernando.com.cer
Generating a RSA private key
...+++++
.....+++++
writing new private key to 'annefernando.com.private.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:LK
State or Province Name (full name) [Some-State]:
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Anne Fernando Pvt Ltd
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:annefernando.com
Email Address []:
OpenSSL>
```

Figure 6.1: Step 1 - Self-Sign SSL Certificate Creation

- Parameter needed - Generated self-sign certificate name

In this project we have given the previously created annefernando.com.cer as an argument

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.985]
(c) Microsoft Corporation. All rights reserved.

D:\TLSChain\ServerScript>TLSChainCLI.exe -h
usage: This is TLSChain CLI

*****
* This is TLSChain Decentralized PKI *
*****

positional arguments:
  Commands  'getPubHash -c','registerDomain -c -p -d -wa -wp','retriveValidityCer -c','retriveValidityHash
            -hp','revokeDomain -c -wa -wp','renewDomain -c -p -d -wa -wp'

optional arguments:
  -h, --help  show this help message and exit
  -hp HP      Public key hash
  -c C        Certificate file name
  -p P        Private key file
  -d D        Domain name
  -wa WA      Wallet address
  -wp WP      Wallet_private_key

Copyrights @ Anne Fernando

D:\TLSChain\ServerScript>TLSChainCLI.exe getPubHash -c annefernando.com.cer
b'66XtQ+REJXHbAFGVsKPrWe6n4FJ14MRxLhmXDq3QgwM='

D:\TLSChain\ServerScript>
```

Figure 6.2: Step 2 - Generating public key hash using TLSChain

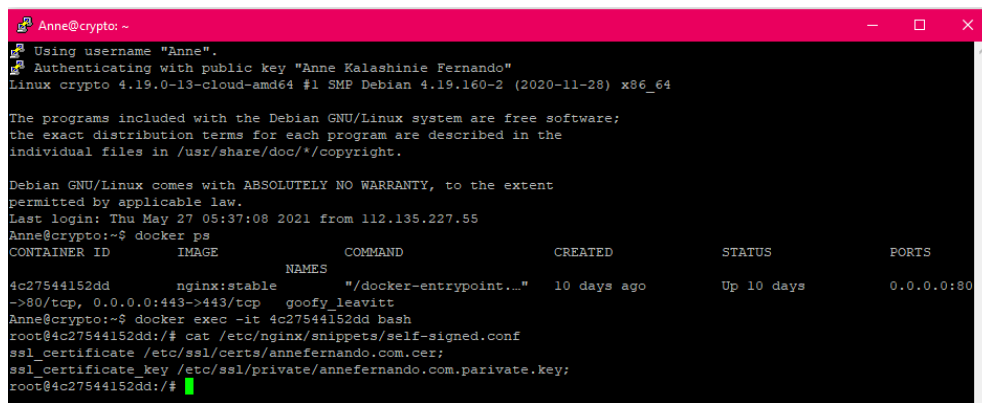
Retrieved public key hash is '66XtQ+REJXHbAFGVsKPrWe6n4FJ14MRxLhmXDq3QgwM='

---

### 6.1.3 Step 3- Configure the Web Server

Configure the web server with the newly created certificate and the private key to use HTTPS. Also place the generated public key hash in the public folder.

In this project we have used a linux hosting server and a Nginx web server. Nginx configuration snippet in the ‘/etc/nginx/snippets’ directory with the name ‘**self-signed.conf**’ is created to clearly distinguish the purpose. In this file we have specified ‘**ssl-certificate**’ directive to our certificate file and the ‘**ssl-certificate-key**’ to the associated private key. Refer figure 6.3



```
Anne@crypto: ~
└─$ Using username "Anne".
    Authenticating with public key "Anne Kalashinie Fernando"
Linux crypto 4.19.0-13-cloud-amd64 #1 SMP Debian 4.19.160-2 (2020-11-28) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu May 27 05:37:08 2021 from 112.135.227.55
Anne@crypto:~$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS
4c27544152dd   nginx:stable  "/docker-entrypoint..." 10 days ago   Up 10 days   0.0.0.0:80
->80/tcp, 0.0.0.0:443->443/tcp   goofy_leavitt
Anne@crypto:~$ docker exec -it 4c27544152dd bash
root@4c27544152dd:/# cat /etc/nginx/snippets/self-signed.conf
ssl_certificate /etc/ssl/certs/annefernando.com.cer;
ssl_certificate_key /etc/ssl/private/annefernando.com.priivate.key;
root@4c27544152dd:/#
```

Figure 6.3: Step 3a - Creating SSL Configuration Snippet

After creating the SSL configuration snippet we can add it to the Nginx Configurations to use SSL

In this project we have configured the Nginx sever to allow both HTTP and HTTPS traffic for the moment. After registering with TLSChain we will remove HTTP related configurations. For this ‘**deafult.conf**’ file is changed in the location ‘**etc/nginx/conf.d**’. Refer Figure 6.4

After that we will have to place the retrieved public key hash inside the public folder. For this we have created a file named ‘**TLSCchain.Json**’ and placed the following Json content. Refer Figures 6.5 6.6

```
{ "TLSCchain": "66XtQ+REJXHBaFGVsKPrWe6n4FJ14MRxLhmXDq3QgwM=" }
```



```
Anne@crypto: ~
root@4c27544152dd:/etc/nginx/conf.d# cat default.conf
server {
    listen 80 default_server;
    listen [::]:80 default_server;
    listen 443 ssl http2 default_server;
    listen [::]:443 ssl http2 default_server;

    server_name annefernando.com;
    include snippets/self-signed.conf;

    #charset koi8-r;
    #access_log /var/log/nginx/host.access.log main;

    location / {
        root /usr/share/nginx/html;
        index index.html index.htm;
    }

    #error_page 404 /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
```

Figure 6.4: Step 3b - Adjust the Nginx Configuration to Use SSL

```
Anne@crypto: ~
root@4c27544152dd:/usr/share/nginx/html# cat TLSChain.json
{"TLSChain": "66XtQ+REJXHBAFGVsKPrWe6n4FJ14MRxLhmXDq3QgwM="}
```

Figure 6.5: Step 3c - Placing Public Key Hash in the public folder

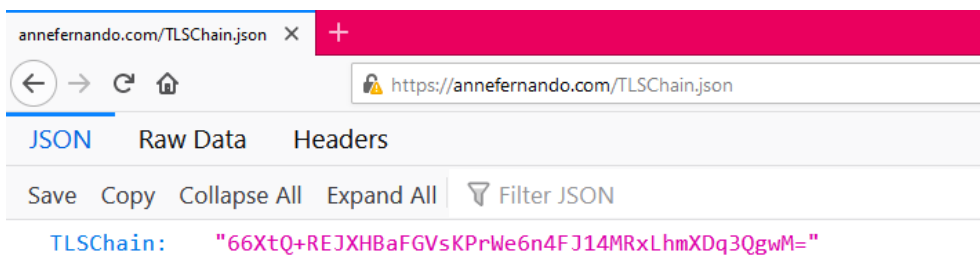


Figure 6.6: Step 3d - Public Key Hash in the public folder should be visible from the a browser

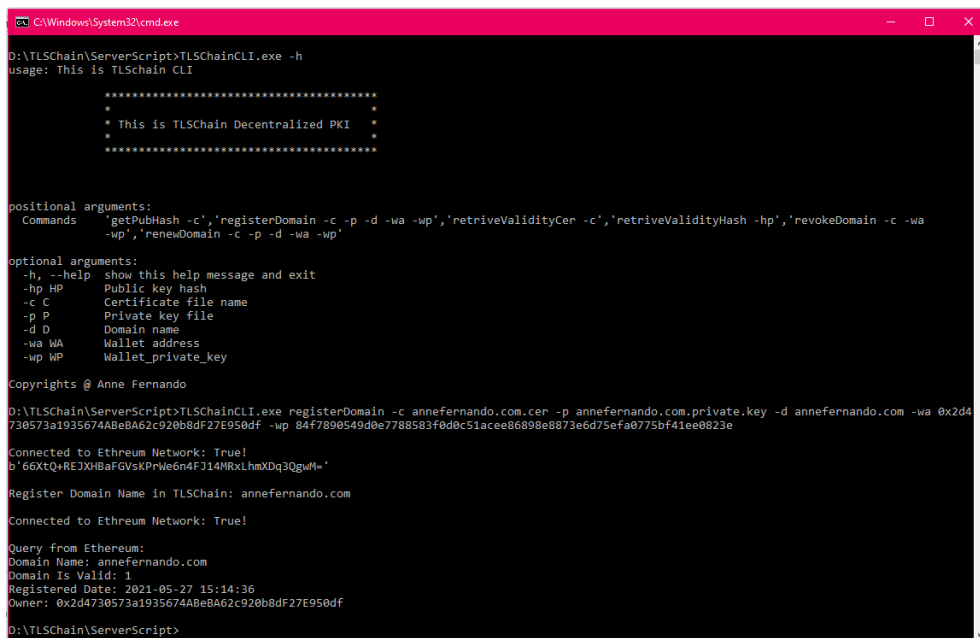
---

## 6.1.4 Step 4- Register a Domain with TLSChain

After configuring the web server, domain can be registered to the TLSChain. For this we can use the TLSChainCLI “registerDomain” command.

- Command to use - registerDomain
- Parameter needed - -c -p -d -wa -wp
  - c - Generated self-sign certificate name
  - p - Private key of the server certificate
  - d - Domain to register
  - wa - Wallet address
  - wp - Wallet private key

After a domain is registered, if the domain is successfully configured the registered date and the validity will be displayed. Refer Figure 6.7



```

C:\Windows\System32\cmd.exe
D:\TLSChain\ServerScript>TLSChainCLI.exe -h
usage: This is TLSChain CLI

*****
* This is TLSChain Decentralized PKI *
*****

positional arguments:
  Commands  'getPubHash -c','registerDomain -c -p -d -wa -wp','retriveValidityCer -c','retriveValidityHash -hp','revokeDomain -c -wa
            -wp','renewDomain -c -p -d -wa -wp'

optional arguments:
  -h, --help  show this help message and exit
  -hp HP      Public key hash
  -c C        Certificate file name
  -p P        Private key file
  -d D        Domain name
  -wa WA      Wallet address
  -wp WP      Wallet private key

Copyrights @ Anne Fernando

D:\TLSChain\ServerScript>TLSChainCLI.exe registerDomain -c annefernando.com.cer -p annefernando.com.private.key -d annefernando.com -wa 0x2d4
730573a1935674ABeBA62c920b8df27E950df -wp 84f7898549d0e7788583f0d0c51acee86898e8873e6d75efa0775bf41ee0823e

Connected to Ethereum Network: True!
b'66XtQ+REjXHbaFGVskPrWe6n4Fj14MRxLhmXDq30gwM+'

Register Domain Name in TLSChain: annefernando.com

Connected to Ethereum Network: True!

Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-27 15:14:36
Owner: 0x2d4730573a1935674ABeBA62c920b8df27E950df

D:\TLSChain\ServerScript>
```

Figure 6.7: Step 4 - Domain registration with TLSChain

---

### 6.1.5 Step 5- Verification

After performing the above steps, Domain Owner can add the TLSChain extension to the browser and verify the validity of the domain by pressing TLSChain icon. Green color text “Pass” will indicate the domain public key mapping is registered in the TLSChain. Red color text “Fail” will indicate that the domain public key mapping is not registered or the received public key is different to the public key registered with TLSChain. Refer figure 6.8 6.9

In this project we have developed the TLSChain extension for the Firefox browser.

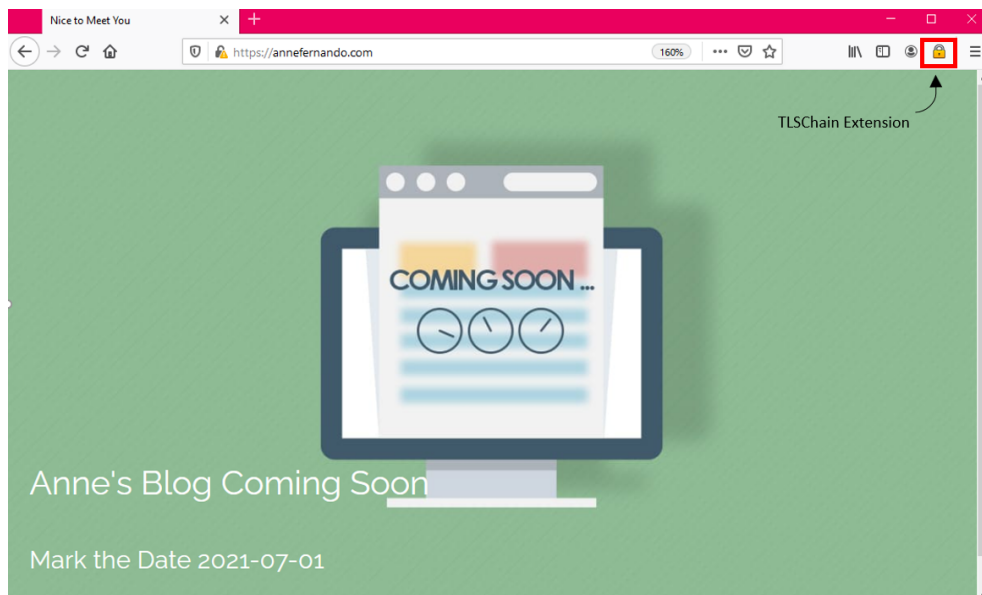


Figure 6.8: Step 5a - TLSChain Extension

Also the domain owner can verify the registration by using the public key hash or the certificate it self. He can you the TLSChainCLI for this. Refer figure 6.10 6.11

#### 6.1.5.1 retrieveValidityCer

- Command to use - retrieveValidityCer
- Parameter needed - X.509 certificate name

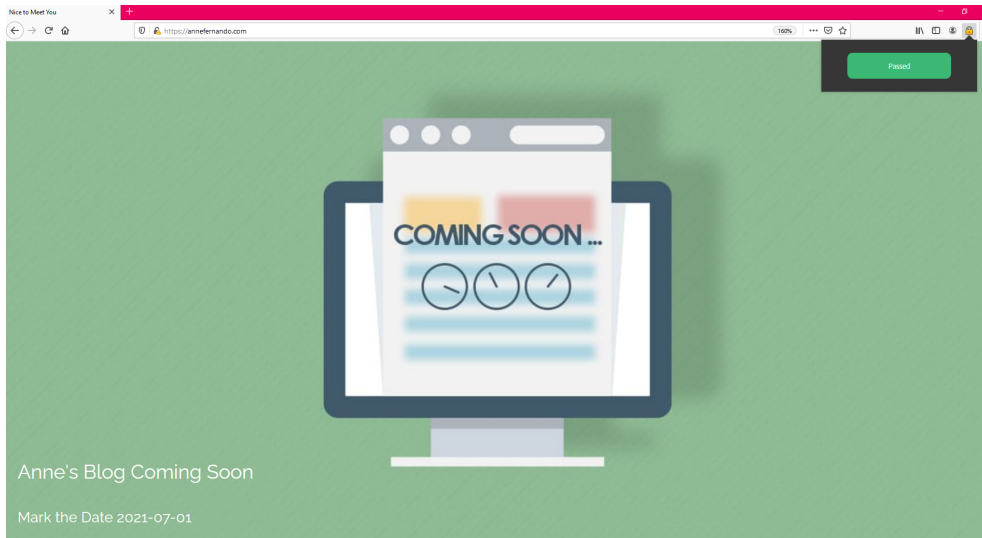


Figure 6.9: Step 5b - TLSChain Extension Successful Validation

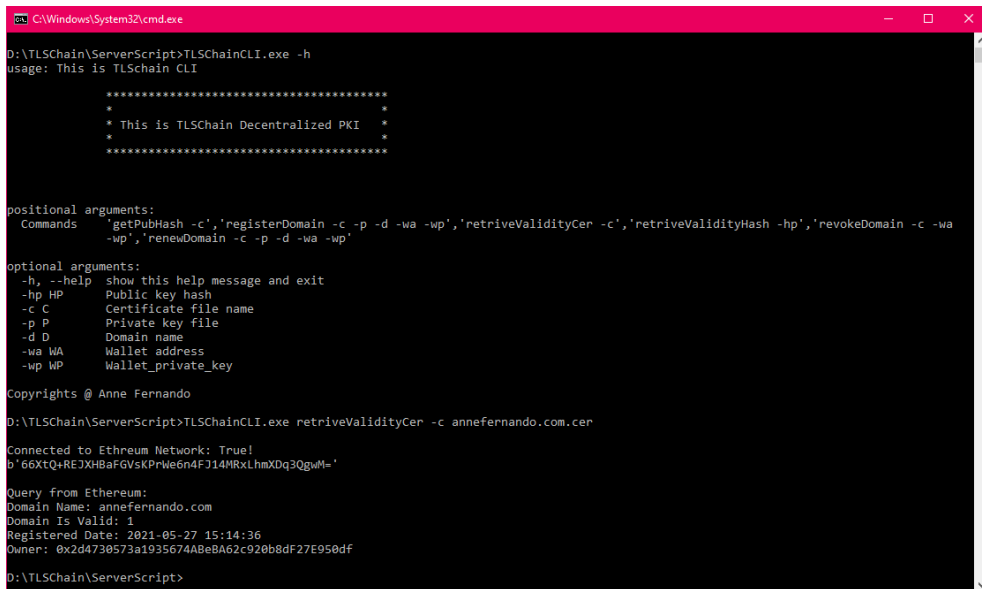
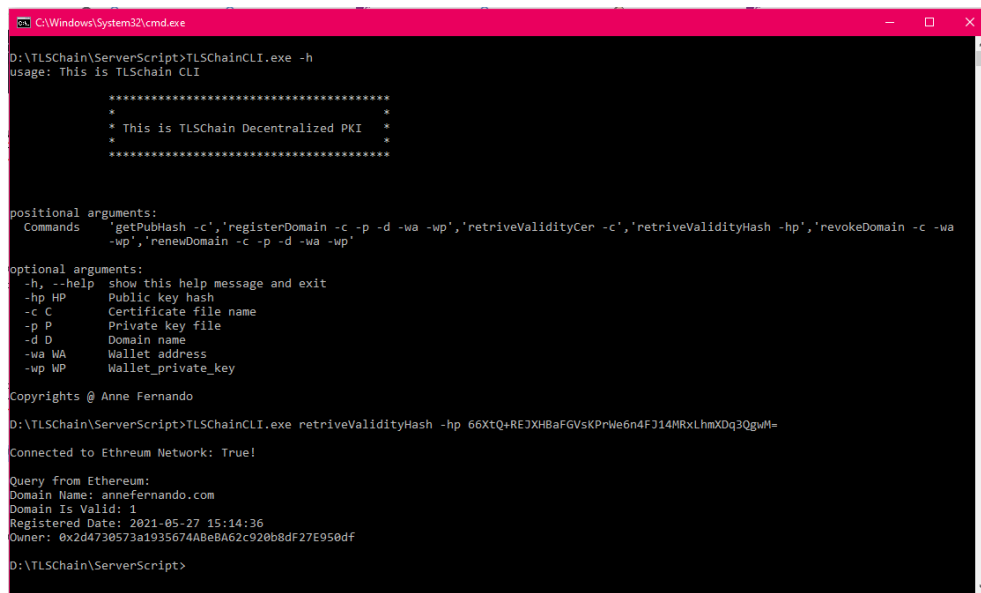


Figure 6.10: Step 5c - TLSChain CLI Successful Validation retrieveValidityCer

---

### 6.1.5.2 retrieveValidityHash

- Command to use - retrieveValidityHash
- Parameter needed - Public key hash



```
C:\Windows\System32\cmd.exe
D:\TLSChain\ServerScript>TLSChainCLI.exe -h
usage: This is TLSChain CLI

*****
* This is TLSChain Decentralized PKI *
*****

positional arguments:
  Commands      'getPubHash -c','registerDomain -c -p -d -wa -wp','retrieveValidityCer -c','retrieveValidityHash -hp','revokeDomain -c -wa
                 -wp','renewDomain -c -p -d -wa -wp'

optional arguments:
  -h, --help      show this help message and exit
  -hp HP          Public key hash
  -c C            Certificate file name
  -p P            Private key file
  -d D            Domain name
  -wa WA          Wallet address
  -wp WP          Wallet_private_key

Copyrights @ Anne Fernando

D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityHash -hp 66XtQ+REJXH8aFGVsKPrWe6n4FJ14NRxLhmXDq3QgwM=

Connected to Ethereum Network: True!

Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-27 15:14:36
Owner: 0x2d4738573a1935674ABeBA62c920b8dF27E950df

D:\TLSChain\ServerScript>
```

Figure 6.11: Step 5d -TLSChain CLI Successful Validation retrieveValidityHash

# Chapter 7

## Evaluation

Evaluation process will be carried out based on the implemented proof of concept **TLSCchain**. Current functionalities of certification authorities are compared against TLSCchain in this process.

We have identified 17 test cases that one case conclude whether is it possible to come up with a novel decentralized public key infrastructure based on blockchain technology without having a certification authority. Following test cases were carefully selected not only to evaluate the functionality of the PKI, but also to evaluate the efficiency and security of the proposing PKI.

### 7.1 Evaluation Setup

To have a consistency throughout the evaluation process, we will be using using two users, “**Anne**” and “**Eve**” continuously in this chapter. These two users actions will show how the TLSCchain would behave for a legitimate user and for an attacker.

**Anne** - The actual domain owner of the annefernando.com and a legitimate user of the TLSCchain.

**Eve** - An attacker, the main intention is to get the control of annefernando.com by associating Eve’s public key to this domain.

---

TLSChain User Anne	
Attribute	Value
Description	Domain owner of the annefernando.com
Certificate	annefernando.com.cer
Private Key	annefernando.com.private.key
Wallet Address	0x12e6EED8a7B2593649Aa3A75912d449521044769
Wallet Private Key	86924ffe8ff82494fbca5946ab9a56acb4447b6281c218c5c82afebbaac3e108

Table 7.1: Legit User Anne’s Details

TLSChain User Eve	
Attribute	Value
Description	Evil Attacker of the annefernando.com
Certificate	eve.cer
Private Key	eve.private.key
Wallet Address	0xa4B1146eDE50eEBD9fD2564e199E311f112B0Ae6
Wallet Private Key	a5aea50dfc6262a48577871f8476f2180bde219dae04b036b15605cbd0373d76

Table 7.2: Attacker Eve’s Details

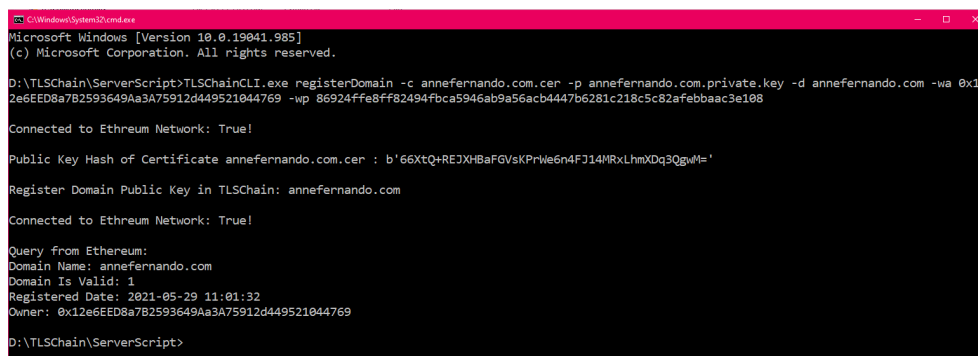
## 7.2 Evaluation Process

### 7.2.1 TLSChain Achieving Certificate Issuing Process

#### 7.2.1.1 Test Case 1 - Possibility of registering a public key for a domain using TLSChain

- Current CA Behavior - In the current CA based PKI, this is achieved by using a Certificate Signing Request (CSR). This CSR contain domain owner information and public key details.
- TLSChain Behavior - Generate a self-sign certificate and creates and Ethereum transaction which contains domain owner information and public key hash.
- Proof - User Anne is registering domain she owns annefernando.com with TLSChain and it’s successful. Anne is the first user to register this domain with a public key to TLSChain. Anne should have done all the configurations steps mentioned in the Chapter 6 before proceeding with this. Refer

figure 7.1



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.985]
(c) Microsoft Corporation. All rights reserved.

D:\TLSChain\ServerScript>TLSChainCLI.exe registerDomain -c annefernando.com.cer -p annefernando.com.private.key -d annefernando.com -wa 0x12e6EED8a7B2593649Aa3A75912d449521044769 -wp 86924ffe8ff82494fbc55946ab9a56acb4447b6281c218c5c82afebbaac3e108

Connected to Ethereum Network: True!

Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGvsKPrWe6n4FJ14MRxLhmXDq3QgW#='

Register Domain Public Key in TLSChain: annefernando.com

Connected to Ethereum Network: True!

Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:01:32
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769

D:\TLSChain\ServerScript>
```

Figure 7.1: Test Case1 -Anne Successfully Register Domain Certificate with TLSChain

### 7.2.1.2 Test Case 2 - An attacker trying to register a new public key for an already registered domain should not be successful

- Current CA Behavior - In the current CA based PKI this is achieved by the Registration Authority(RA) which is part of CA based PKI. RA verifies the identity in question, and it is not possible for a different entity to register a public key for an already registered domain and such request are rejected.
- TLSChain Behavior - If a domain is registered with a public key it is not possible for a different user to register a public key for the same domain. Necessary validations are performed by the smart contract.
- Proof - Eve is trying to register anefernando.com with her own public key. This is unsuccessful. Refer Figures 7.2

### 7.2.1.3 Test Case 3 - Initial owner updating the registered public key information should be successful

- Current CA Behavior - In the current CA based PKI this is achieved by following the renewal process of the CA.
- TLSChain Behavior - Only the initial owner can update the public key information



```

C:\Windows\System32\cmd.exe
D:\TLSChain\ServerScript>TLSChainCLI.exe registerDomain -c eve.cer -p eve.private.key -d annefernando.com -wa 0xa481146eDE50eEBD9FD2564e199E311f112B0Ae6 -wp a5aea50dfc6262a48577871f8476f2180bde219dae04b036b15605cbd0373d76

Connected to Ethereum Network: True!

Public Key Hash of Certificate eve.cer : b'52i0RXzWvO+4qcOqP6WXd39q/hpLhw3RTiLk/GoY4KU='

Register Domain Public Key in TLSChain: annefernando.com
VM Exception while processing transaction: revert This Domain has a Certificate Attached

Connected to Ethereum Network: True!

Query from Ethereum:
Domain Name: No Associated Domain For this Certificate
Domain Is Valid: 0
Registered Date: 0
Owner: 0x0000000000000000000000000000000000000000000000000000000000000000

D:\TLSChain\ServerScript>

```

Figure 7.2: Test Case2 - Eve’s unsuccessful attempt to register annefernando.com with her certificate

- Proof - Anne is trying to extend the validity this is successful. Anne is providing the wallet address and the wallet private key that she used register the domain initially. Eve is also trying to take control of the annefernando.com by using extending validity option but it is unsuccessful. Refer Figures 7.3 7.4 7.5

```

C:\Windows\System32\cmd.exe
D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityCer -c annefernando.com.cer

Connected to Ethereum Network: True!

Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGvsKPrWe6n4FJ14MRxLhmXDq3QgwM='

Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:01:32
Owner: 0x12e6EED8a782593649Aa3A75912d449521044769

D:\TLSChain\ServerScript>TLSChainCLI.exe renewDomainValidity -c annefernando.com.cer -p annefernando.com.private.key -d annefernando.com -wa 0x12e6EED8a782593649Aa3A75912d449521044769 -wp 86924ffe8ff82494fbc5946ab9a56ac64447b6281c218c5c82afebbaac3e188

Connected to Ethereum Network: True!

Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGvsKPrWe6n4FJ14MRxLhmXDq3QgwM='

Renew Domain Public Key Validiy in TLSChain: annefernando.com

Connected to Ethereum Network: True!

Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:46:57
Owner: 0x12e6EED8a782593649Aa3A75912d449521044769

D:\TLSChain\ServerScript>

```

Figure 7.3: Test Case 3a - Anne successfully extending the validity

```

D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityCer -c annefernando.com.cer
Connected to Ethereum Network: True!
Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGvsKPrWe6n4FJ14MRxLhmXDq3QgwM='
Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:46:57
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>TLSChainCLI.exe renewDomainValidity -c eve.cer -p eve.private.key -d annefernando.com -wa 0xa4B1146eDE50eEBD9FD2564e199E311f112B0Ae6 -wp a5aaa50dfc6262a48577871f8476f2180bde219dae04b036b15605cbd0373d76
Connected to Ethereum Network: True!
Public Key Hash of Certificate eve.cer : b'52i0RXzWvO+4qcOqP6WXd39q/hpLhw3RTlK/GoY4KU='
Renew Domain Public Key Validity in TLSChain: annefernando.com
VM Exception while processing transaction: revert Only Domain Owner Can Perform This Function
Connected to Ethereum Network: True!
Query from Ethereum:
Domain Name: No Associated Domain for this Certificate
Domain Is Valid: 0
Registered Date: 0
Owner: 0x0000000000000000000000000000000000000000000000000000000000000000
D:\TLSChain\ServerScript>

```

Figure 7.4: Test Case 3b - Eve's unsuccessful attempt to extend the validity

```

D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityCer -c selfsigned.cer
Connected to Ethereum Network: True!
Public Key Hash of Certificate selfsigned.cer : b'Tm08EfgZdyIhPw1lFcp2s6L8u0v6vkBvs3qa7884Ks='
Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-31 13:35:59
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>TLSChainCLI.exe getPubHash -c annefernando.com.cer
Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGvsKPrWe6n4FJ14MRxLhmXDq3QgwM='
D:\TLSChain\ServerScript>TLSChainCLI.exe requestRenewDomainPubKey -c annefernando.com.cer -oc selfsigned.cer -p annefernando.com.private.key -d annefernando.com -wa 0x12e6EE08a7B2593649Aa3A75912d449521044769 -wp 86924ffe8f82494fbca5946ab9a56acb4447b6281c218c5c82afebbaac3e188
Connected to Ethereum Network: True!
Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGvsKPrWe6n4FJ14MRxLhmXDq3QgwM='
Public Key Hash of Certificate selfsigned.cer : b'Tm08EfgZdyIhPw1lFcp2s6L8u0v6vkBvs3qa7884Ks='
Renew Public for a Domain in TLSChain: annefernando.com
Connected to Ethereum Network: True!
Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-31 13:37:37
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityCer -c selfsigned.cer
Connected to Ethereum Network: True!
Public Key Hash of Certificate selfsigned.cer : b'Tm08EfgZdyIhPw1lFcp2s6L8u0v6vkBvs3qa7884Ks='

```

Figure 7.5: Test Case 3c - Anne successfully changing the certificate for the same domain

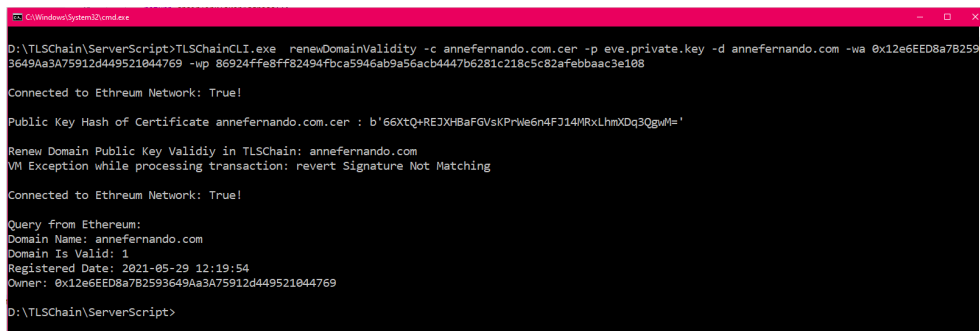
---

## 7.2.2 TLSChain for Storage of Certificate Information

### 7.2.2.1 Test Case 4 - Successful storage of public key information

- Current CA Behavior - In the current CA based PKI this is achieved by CA issuing a digital certificate which contains the public key information and storing public key information in a central directory.
- TLSChain Behavior - Self-sign certificate generated with the public key and the public key hash and domain information retained in the blockchain after successful verification. Else the transaction is revoked. Successful transactions are retained in the blockchain and unsuccessful transactions are ignored.
- Proof - Eve's unsuccessful attempt to renew domain with a different private key. Signature validation step detect this and the transaction is reverted. Retrieved information display that the domain is still belong to domain owner Anne.

Retrieved information displayed owner as Anne due to successful retention of the public key details. Refer figures 7.6 and 7.1



```

D:\TLSChain\ServerScript>TLSChainCLI.exe renewDomainValidity -c annefernando.com.cer -p eve.private.key -d annefernando.com -wa 0x12e6EED8a7B2593649Aa3A75912d449521044769 -wp 86924ffe8ff82494fbc5946ab9a56acb4447b6281c218c5c82afebbaac3e108

Connected to Ethereum Network: True!

Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJX#BaFGVskPriWe6n4FJ14/RxLhmX0q3Qw#t='

Renew Domain Public Key Validiy in TLSChain: annefernando.com
VM Exception while processing transaction: revert Signature Not Matching

Connected to Ethereum Network: True!

Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 12:19:54
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769

D:\TLSChain\ServerScript>
```

Figure 7.6: Test Case 4 - Eve's unsuccessful attempt to renew the domain public key validity

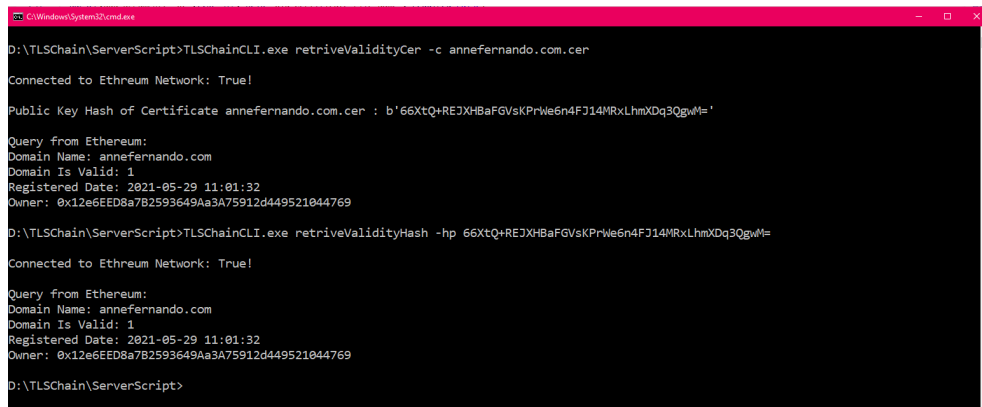
### 7.2.2.2 Test Case 5 - Successful retrieval of public key information

- Current CA Behavior - In the current CA based PKI this information is distributed using a Certification Revocation List (CRL) or provided when

---

requested via an Online Certificate Status Protocol (OCSP) request.

- TLSChain Behavior - This information can be retrieved from the smart contract by providing the hash of the public key or the certificate.
- Proof - If the Certificate is provided, it is possible to retrieve the status of the domain public key mapping. Refer figure 7.7



```
C:\Windows\System32\cmd.exe
D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityCer -c anefernando.com.cer
Connected to Ethereum Network: True!
Public Key Hash of Certificate anefernando.com.cer : b'66XtQ+REJXHBaFGvsKPrWe6n4FJ14MRxLhmXDq3QgwM='
Query from Ethereum:
Domain Name: anefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:01:32
Owner: 0x12e6EED8a782593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityHash -hp 66XtQ+REJXHBaFGvsKPrWe6n4FJ14MRxLhmXDq3QgwM=
Connected to Ethereum Network: True!
Query from Ethereum:
Domain Name: anefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:01:32
Owner: 0x12e6EED8a782593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>
```

Figure 7.7: Test Case 5 - Retrieving Certificate Domain Details

## 7.2.3 Blockchain based PKI for certificate revocation

### 7.2.3.1 Test Case 6 - Initial Owner is able to revoke the registered information

- Current CA Behavior - In the current CA based PKI a compromised entity will create a revocation request and present it to the CA.
- TLSChain Behavior - The initial owner can send an Ethereum transaction to the smart contract requesting to invalidate the entry for a particular public key.
- Proof - Anne successfully revoking the domain public key registration. Refer figure 7.8

```

C:\Windows\System32\cmd.exe
D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityCer -c annefernando.com.cer
Connected to Ethereum Network: True!
Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGVsKPrW6n4FJ14MRxLhmXDq3QgwM='
Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:46:57
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>TLSChainCLI.exe revokeDomain -c annefernando.com.cer -wa 0x12e6EED8a7B2593649Aa3A75912d449521044769 -wp 86924ffe8ff82494
Fbca5946ab9a56ac6447b6281c218c5c82afebbaac3e108
Connected to Ethereum Network: True!
Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXHbAFGVsKPrW6n4FJ14MRxLhmXDq3QgwM='
Revoke Domain Public Key Validity in TLSChain
Connected to Ethereum Network: True!
Query from Ethereum:
Domain Public Key Revoked
Revoked Date: 2021-05-29 11:57:38
Domain Name: annefernando.com
Domain Is Valid: 0
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>

```

Figure 7.8: Test Case 6 - Anne successfully revoking

### 7.2.3.2 Test Case 7 - Attacker should not be able to revoke a registered information

- Current CA Behavior - In the current CA based PKI a compromised entity will create a revocation request and present it to the CA.
- TLSChain Behavior - The initial owner can send an Ethereum transaction to the smart contract requesting to invalidate the entry for a particular public key. If the transaction is signed by a different user other than the initial owner, the transaction would fail and it not possible to revoke such mapping.
- Proof - Eve's unsuccessful attempt to revoke. Refer Figure 7.9

## 7.2.4 TLSChain Achieve Certificate Verification Process

### 7.2.4.1 Test Case 8 - When accessing a domain registered with TLSChain, if the correct public key sent from the server it will be indicated

- Current CA Behavior - Browser indicate this with a lock symbol in the URL field.

```

C:\Windows\System32\cmd.exe
D:\TLSChain\ServerScript>TLSChainCLI.exe retrieveValidityCer -c annefernando.com.cer
Connected to Ethereum Network: True!
Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXH8aFGVskPrWe6n4FJ14MRxLhmXDq3QgwM='
Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-29 11:46:57
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>TLSChainCLI.exe revokeDomain -c annefernando.com.cer -wa 0xa4B1146e0E50eEBD9FD2564e199E311f11280Ae6 -wp a5aea58dfc6262a48577871f8476f2180bde219dae04b036b15605cbd0373d7e
Connected to Ethereum Network: True!
Public Key Hash of Certificate annefernando.com.cer : b'66XtQ+REJXH8aFGVskPrWe6n4FJ14MRxLhmXDq3QgwM='
Revoke Domain Public Key Validity in TLSChain
VM Exception while processing transaction: revert Only Domain Owner Can Perform This Function
Connected to Ethereum Network: True!
Query from Ethereum:
Domain Public Key not Revoked
Registered Date: 2021-05-29 11:46:57
Domain Name: annefernando.com
Domain Is Valid: 1
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769
D:\TLSChain\ServerScript>

```

Figure 7.9: Test Case 7 - Eve's unsuccessful attempt to revoke

- TLSChain Behavior - The browser extension will fetch domain of the web page and retrieve the public key information and the domain and verify it against the information stored in the smart contract. If the domain and the relevant public key information stored and it is valid this will be indicated.
- Proof - annefernando.com is registered with the TLSChain hence extension will indicate the received public key from the server is indeed the correct public key. Refer figure 7.10

#### 7.2.4.2 Test Case 9 - When accessing a domain registered with TLSChain, an incorrect public key is sent and it will be indicated

- Current CA Behavior - Browser will indicate a warning in such situations only if the received certificate's CA is not trusted by the browser. If this is a fake certificate created by an attacker with a compromised CA browser will allow the communication until the certificate is revoked.
- TLSChain - The browser extension will fetch domain of the web page and retrieve the public key information and the domain and verify it against the information stored in the smart contract. If there is a valid entry in

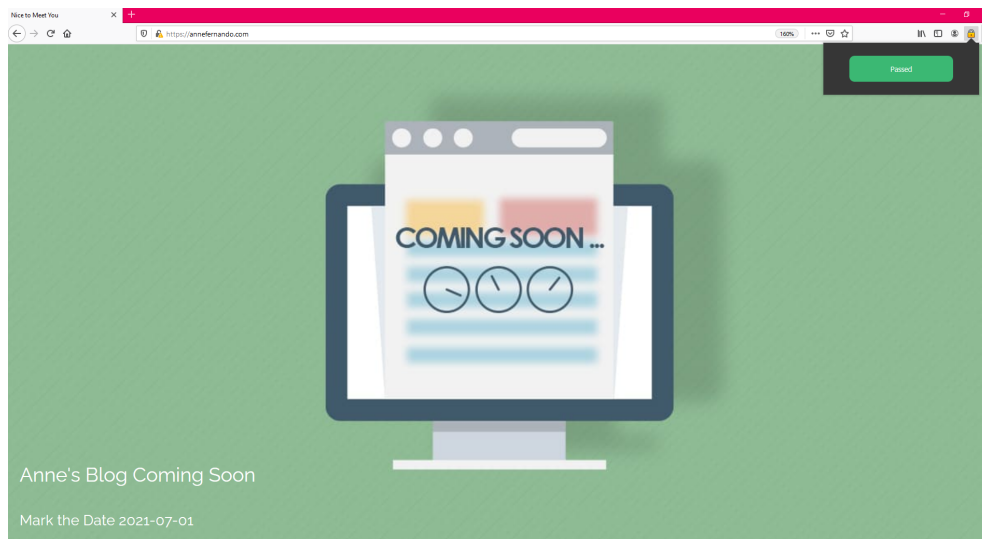


Figure 7.10: Test Case 8 - TLSChain Extension Verification

the smart contract for a particular domain but the public does not match with the received certificate this will be indicated. If the public key for the domain is correct but the entry is invalid this is also indicated.

- Proof - Domain owner registered the annefernando.com with TLSChain using the certificate annefernando.com.cer. But Eve hacked the web server and configured the webserver with her certificate eve.cer private key eve.private.key. Now during the HTTPS communication annefernando.com.cer will send the public key related to eve.cer. This is different what is registered in TLSChain hence the extension will indicate this. Refer Figure 7.11 and 7.12 and compare the public key hashes.

#### 7.2.4.3 Test Case 10 - When accessing a domain not registered with TLSChain it will be indicated

- Current CA Behavior - This will be indicated as a warning in the browser.
- TLSChain Behavior - The browser extension will fetch domain of the webpage and retrieve the public key information and the domain and verify it against the information stored in the smart contract. If there isn't any

```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19041.985]
(c) Microsoft Corporation. All rights reserved.

D:\TLSChain\ServerScript>TLSChainCLI.exe registerDomain -c selfsigned.cer -p private.key -d annefernando.com -wa 0x12e6EED8a7B2593649Aa3A75
912d449521044769 -wp 86924fFe8ff82494fbca5946ab9a56acb4447b6281c218c5c82aFebbaac3e108

Connected to Ethereum Network: True!

Public Key Hash of Certificate selfsigned.cer : b'TmO0EfgZdy1hPw1FCP2s6L8uH0v6vkBvs3qa7884Ks='

Register Domain Public Key in TLSChain: annefernando.com

Connected to Ethereum Network: True!

Query from Ethereum:
Domain Name: annefernando.com
Domain Is Valid: 1
Registered Date: 2021-05-31 13:13:03
Owner: 0x12e6EED8a7B2593649Aa3A75912d449521044769

D:\TLSChain\ServerScript>

```

Figure 7.11: Test Case 9 - TLSChain Extension Verification Incorrect Public Key Received from the Server

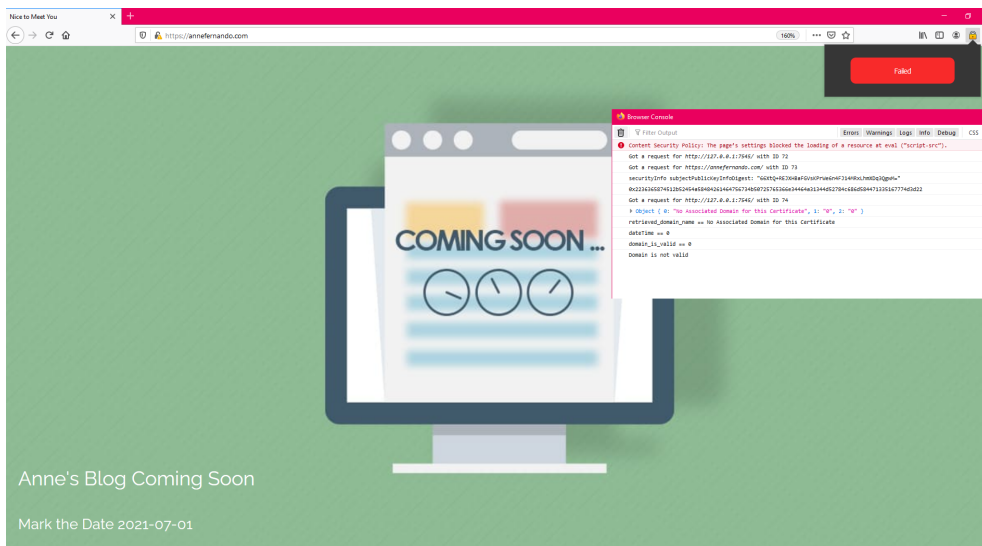


Figure 7.12: Test Case 9b - TLSChain Extension Verification Incorrect Public Key Received from the Server

registration for this domain (valid or invalid), it is also indicated. Then the user can determine whether to proceed with the connection or not.

- Proof - www.facebook.com yet to be registered with TLSChain. Hence TLSChain extension will indicate that it is not registered. Refer Figure 7.13



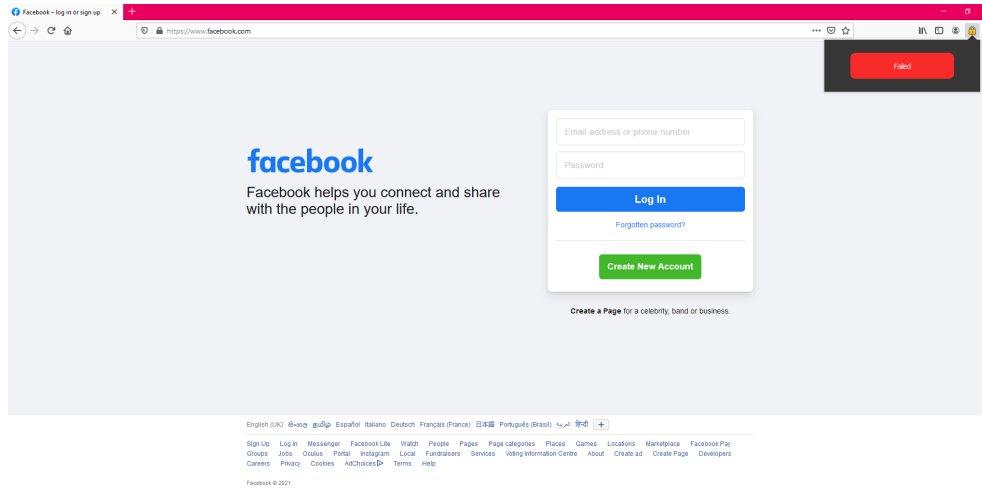


Figure 7.13: Test Case 10 - TLSChain Extension Verification

## 7.2.5 TLSChain Efficiency

### 7.2.5.1 Test Case 11 - Time taken for registering a public key for a domain

- Current CA Behavior - According Table 7.3 to issue a domain validation certificate it will takes around 5-30 mins.

Current DV Certificate Price and Issuance Time			
CA	Validation Method	Issuance Time	Price(USD)
AlphaSSL	Via Email	5 mins or less	16
RapidSSL	Automated domain control validation	10 mins	16
THAWTE	Domain Validation	1 Business day, or less	32
COMODO	Domain Validation	15 mins or less	32
GEOTRUST	Domain Validation	30 mins or less	40
GLOBALSIGN	Domain Validation	30 mins	100

Table 7.3: DV Certificate Providers

- TLSChain Behavior - Current implementation of TLSChain takes 2 minutes or less to register a domain public key mapping. Current implementation

---

takes around 1-2 for this. But this time will vary when there are several transactions are pending and the miner business. If we are providing a higher gwei value the time it will take would be reduced and for a lesser gwei time will be higher.

#### **7.2.5.2 Test Case 12 - Time taken for the validation in the client side**

- Current CA Behavior - Validation of the certificate performed by the web browser side. Trusted certificate authorities are hard-coded and if the received certificate is signed by one of the trusted certification authority the received certificate is also trusted. This take only few seconds.
- TLSChain Behavior - TLSChain web extension check for the validity of the received domain and public key mapping. This also takes few seconds.

Current browsers it won't much time for this validation as validation is inbuilt to the browsers. TLSChain extension will take sometime as it need to access the TLSChain Smart contract for current mappings.

#### **7.2.5.3 Test Case 13 - Registration cost in TLSChain compared to the existing systems**

- Current CA Behavior - According Table [7.3](#) it will cost around 16 USD -100 USD.
- TLSChain Behavior - There are two transactions happens when registration of the domain public key mapping, one related to the TLSChain smart contract **registration** function and when the Chainlink oracle access the web server. In the TLSChain client we have set the gas price to 50 gwei, but it is possible to change this gas price.
- Proof - Refer Figure [7.14](#) , [7.15](#) , [7.16](#)

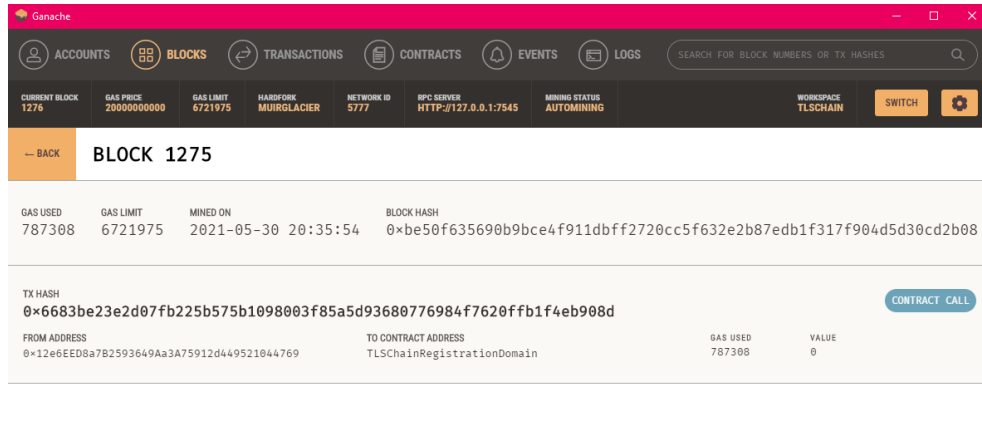


Figure 7.14: Test Case 14 - TLSChain Smart Contract Registration

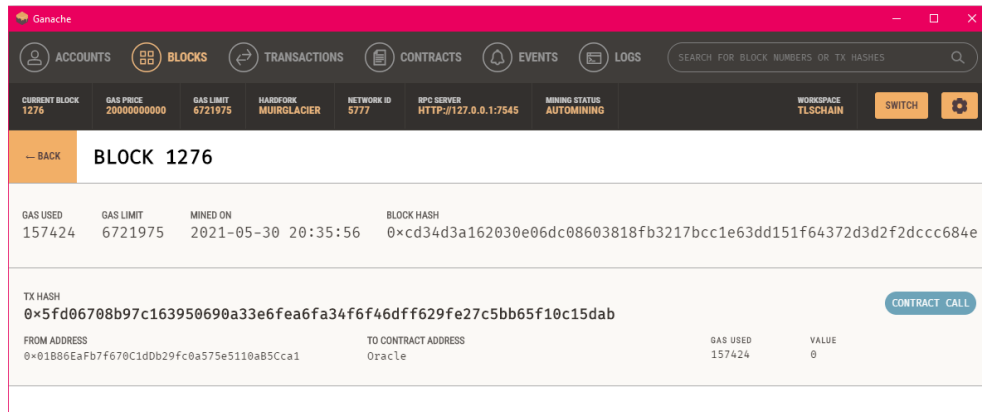


Figure 7.15: Test Case 14 - Chainlink Oracle

TLSChain Costs 2021 May 1ETH = 2447.92 USD					
TLSChain Func-tion	Gas Used	50(gwei)	Ether(ETH)	USD	Prediction
Registration	944,732	47,236,600	0.0472366	116	100
Revoke	35705	1,785,250	0.0017852	4.37	100

Table 7.4: TLSChain Costs Analysis

When referring to the above information, in 2020 May if a domain owner used TLSChain to register a domain public key with 50Gwei which will give 100% predictability ( Percentage of last 200 blocks accepting this gas price) he only

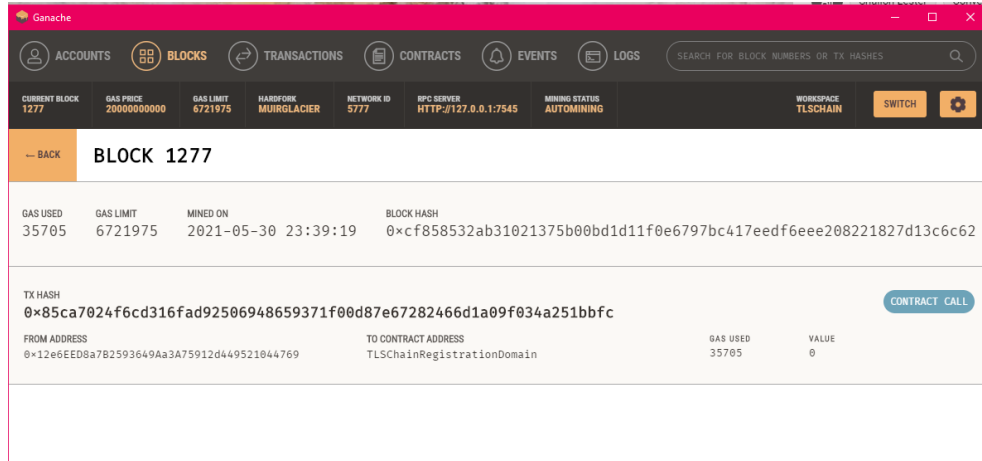


Figure 7.16: Test Case 14 - TLSChain Smart Contract Revocation

TLSChain Costs <b>2021 May</b> 1ETH = 2447.92 USD					
TLSChain Function	Gas Used	20(gwei)	Ether(ETH)	USD	Prediction
Registration	944,732	18,894,640	0.0188946	46.25	71
Revoke	35705	714,100	0.0007141	1.75	71

Table 7.5: TLSChain Costs Analysis

TLSChain Costs <b>2020 May</b> 1ETH = 241 USD					
TLSChain Function	Gas Used	50(gwei)	Ether(ETH)	USD	Prediction
Registration	944,732	47,236,600	0.0472366	11.38	100
Revoke	35705	1,785,250	0.0017852	0.43	100

Table 7.6: TLSChain Costs Analysis

TLSChain Costs <b>2020 May</b> 1ETH = 241 USD					
TLSChain Function	Gas Used	20(gwei)	Ether(ETH)	USD	Prediction
Registration	944,732	18,894,640	0.0188946	4.55	71
Revoke	35705	714,100	0.0007141	0.17	71

Table 7.7: TLSChain Costs Analysis

needed to spend 11 USD. If it was 20Gwei it was just 4USD with 71%.

But this was drastically changed in a year later and in 2021 May for the same

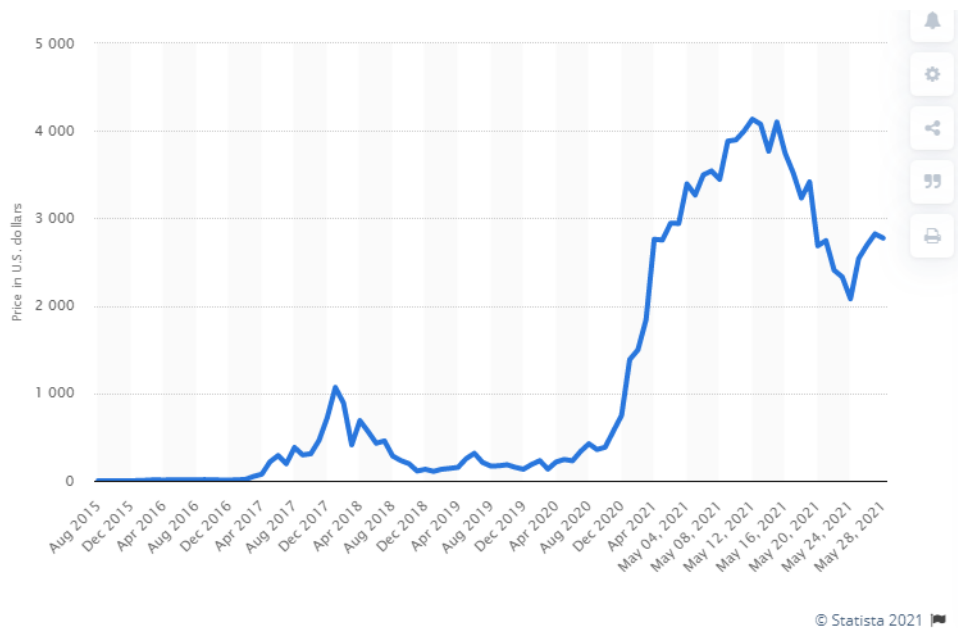


Figure 7.17: Test Case 14 - ETH to USD fluctuation

transaction the domain owner has to pay 116 USD for the 50 Gwei transaction and 46 USD if 20 Gwei was used.

Hence unpredictability of the transaction fee to pay is a drawback in TLSChain.

## 7.3 Security Analysis

### 7.3.1 Test Case 14 - Preventing Man in the Middle Attack

Typical man in the middle attacks can be carried out in two ways. 1) Obtain access to a fake web server certificate issued by a trusted CA 2) Creating a fake certificate and adding it to the trusted CA list.

These type of attacks are not possible when using TLSChain. Creating a fake web server certificate is not enough, to register an incorrect mapping in TLSChain hackers need to get hold of the actual webserver, and as well as the domain owner's Ethereum wallet.

---

### 7.3.2 Test Case 15 - Preventing Replay Attacks

Every Ethereum Transaction has a Nonce. Nonce is a property of transaction originating address. This value is not stored in the Ethereum blockchain. Value of nonce is calculate by counting the number of transactions sent from an address.

Sample Ethereum transaction

```
{ "nonce" : 'how many confirmed transactions this account has sent previously?',  
  "gasPrice",  
  "to",  
  "value",  
  "data",  
  "v,r,s"  
}
```

Value of Nonce can be used to prevent replay attacks. It is possible to make each transaction unique by including a Nonce. If an attacker performs a replay attack by using the same transaction miners will reject that transaction as it is a duplicate.

### 7.3.3 Test Case 16 - Preventing Intruder trying to register a domain that he does not own

A legitimate owner of a domain might send a request to the smart contract to register his domain and the public key. Due the unavailability of the hosted web server or some other reason, there could be a situation where Chain-Link nodes will not able to access the public folder of the webserver (Legitimate scenario) then an attacker might try to register that domain with his own public key.

Signature verification step would pass. But since he does not own the domain, he will not be able to place the necessary Hash of the public key to the public folder of the server. Hence the domain validation step would fail (comparing the content of the public folder and the public key provided for registration -in this case public key provided by the attacker), and attacker will not be able to register the domain with his public key.

---

### **7.3.4 Test Case 17 -Preventing DNS Poisoning attacks**

Attacker might try to poison DNS and try to direct the domain validation request to his server. Chain-link nodes are responsible for fetching the content of the public folder information. There are several chain link nodes involved in this process and they will access different DNS servers to resolve the domain name. Also, after fetching the web server public folder information chain link nodes needs to come into consensus. Hence carrying out this kind of attack is extremely difficult as it is not possible for predetermine which nodes will involve in the public folder value fetching process and each chain link node may access different DNS providers.

# Chapter 8

## Conclusion and Future Work

### 8.1 Revisiting Aims and Objectives

#### 8.1.1 Aims and Objectives

Eliminate the single point of failure in the PKI which is the Certification Authority.

#### 8.1.2 Addressing Identified Problem

1. CA compromised. Hackers issuing rouge certificates using the private key of the CA.

DigiNotar incident is a real-world example of this. More than 500 fake certificates were discovered after security breach in CA including “google.com”. Web browsers trusted these certificates as they were hard-coded to trust the certificate authority.

With TLSChain, a hacker creating a certificate to a domain is not enough. The domain registering party should prove that they have actual access to the domain that they are trying to register at the time of registration. Hence if the a hacker do not have access to the actual server, the registration process will not be successful.

For an already registered domain, only the domain owner will be able to



---

perform any renewal activity of the public key by using TLSChainCLI **requestRenewDomainPubKey** method. For a hacker, it will only be possible to perform this if the hacker get hold of the domain owner's Ethereum wallet. To secure the Ethereum wallet, domain owners can use cold storage or other secure mechanisms to keep their Ethereum wallet secure.

Also Just providing a fake certificate will not be able to get the needed validation from the TLSChain web extension side, as the registered certificate and the browser received certificate is different.

Defense in depth, layered security is in TLSChain.

2. CA accidentally issuing erroneous certificates to customers enabling customers to act as CA themselves.

TrustWave incident is a real-world example for this. They have issued a CA signing certificate to a customer allowing them act as a CA.

In TLSChain, trust and the responsibilities of the CA is distributed and transparent. Validation steps involved in the TLSChain smart contract would reject fake certificates (will fail validations) and will not include in the blockchain. These validations are performed by the miners and only be retained in the Ethereum blockchain, after coming into a consensus. All the validations are visible and it will provide greater transparency on the domain public key approval behavior.

3. Trusted root CAs are hard-coded to browsers and operating systems. In-order to remove a trusted CA security updates should be pushed and high certificate verification time.

CRLs do not operate in real time, they are commonly updated periodically by the CA. There could be a delay in CRL issuing. CRLs can get very large and they should be updates regularly. When it comes to OCSP, the service sometime become unavailable then the verification part ignored altogether.

In TLSChain if a domain owner revokes a certificate(domain public key mapping) this will be reflected to all the nodes when the transaction is mined and the blockchain is updated.

---

Domain public key mapping in TLSChain is stored on-chain across many different nodes. This prevents periodical updates for a off-chain storage and maintain data consistency and has an updated list of the certificate information among all the nodes. Hence even a non-updated browser will also be able to identify a revoked certificate if TLSChain extension is in use.

4. Certificate signing cost. A single certificate can cost between 100 USD to 1000 USD depending on the specific CA and the required certificate.

To obtain a domain verification certificate from an established certification authority it will take around cost. For an Organization Validation certificate or Extended Validation certificate it will cost around 100 USD to 1000 USD.

Certificate signing in TLSChain is related to registering domain and public key mapping. Due to the inflation of the Ethereum price, the TLSChain registration price is higher than the current approach. But this can be further reduced by Ethereum layer 2 solutions.

5. Slow certificate signing process.

Certificate signing in TLSChain is related to registering domain and public key mapping. This functionality is automated and only take few minutes. If there is a congestion in the Ethereum network to mine a transaction, then it will take some time.

## 8.2 Conclusion

Blockchain technology provides an enriched platform to create a public key infrastructure without an involvement of a central authority. In-built characteristics like distributed consensus, signed transactions and immutability provided a solid foundation to the TLSChain - an alternative to a certification authority based on decentralized public key infrastructure design.

Our novel approach of performing domain validation on-chain distinguish this project from related work in this area where most of the current approaches store certificate authority generated certificates on-chain or off-chain. As mentioned

---

in the “Statement of the problem” it was possible to design a novel Public Key Infrastructure by eliminating the centralized Certification Authorities by incorporating blockchain middleware to achieve on-chain domain verification.

Ethereum price fluctuation happened in late 2020 and early 2021 is an eye opener to all the Dapp developers how volatile the cypto currency market can be. Hence if a PKI system is to be launched based on a blokchain network despite of all the advantages that can be gained from the blockchain architecture, the scalability aspect of the solution should be address with priority.

## 8.3 Future Work

### 8.3.1 Interoperability Support

Ethereum is the most evolved smart contract echo system. But there are new smart contract platforms are emerged hence interoperability aspects also should be examined.

eg:- Hyperledger Fabric, Tezos, Polkadot, Solana

### 8.3.2 Domain Validation

Blockchain based PKI can achieve the functionalities of a Certification Authority. When it come to Domain Validating certificates Blockchain based PKI provides greater security as it is begin validated by several different miners. But our proof of concept only replace the Domain Validation certificates.

But there is a need to extend TLSChain to replace Organization and Extended validation certificates.

In our project domain owner is identified by the wallet address only. Hence the actual identity of the wallet address owner is anonymous. To simulate the Organization validation and the extended validation performed by the registration authority(Which is part of the CA) we can incorporate the wallet address with and real world identity. If protecting the privacy is a concern for the domain owner he can keep several different wallet addresses.

To solve the above we can integrate decentralized identity platform **uPort**

---

(Naik & Jenkins 2020). It is a self-sovereign identity, and user-centric data platform. With this users are able to register their own identity in Ethereum and sign transactions and request credentials and securely manage keys and data.

This decentralized identity platform builds a shared identity web of trust and integrating this to TLSChain would solve the Organization and Extended validation requirements, as the public key owner can be mapped to an Uport identity and yet there isn't any centralized governing body.

### **8.3.3 Ethereum Price**

Ethereum emerged on 2015 but 2021 is a significant year for Ethereum as it reached all time all time high value for ETH. In May 10th it reached all time high 4,196USD for 1ETH. Even though the high price indicate the popularity and the success of Ethereum, on the other hand as result of this Ethereum gas fees are also steadily increasing and it also continue to break all time highs. Many Ethereum based applications are becoming unusable.

To address this scaling issue, many scaling solutions for Ethereum network are produced. These are called layer 2 solutions and the main objective is to off-load the financial transaction to a different layer. OMG Network, SKALE Network, and IDEX are such layer 2 solutions.

Layer 2 scaling solutions were been in the work since 2017, but the actual need of such scaling solution is evident in 2021 more than ever (layer 2 ethereum.org 2021).

The price fluctuation of the Ethereum market does effect the transaction cost. The prise of Ethereum rose to all time high and the to stabilize this layer 2 Ethereum solutions should be Incorporated.

### **8.3.4 Validation to the browser and TLSChain support to metamask**

Normal Dapps typical interfaces uses HTML/Java-scripts. We have followed the same path and created an extension using javascript. But in our situation using Java script is not the most secure solution as we are involved with TLS

---

communication. Hence the ideal solution would be to in-cooperate this as an inbuilt function to the browser it self.

Current TLSChain CLI we have to provide the wallet details manually, but by integrating this with existing wallet solutions like Metamask, greater security can be achieved.

## **8.4 Contribution and Novelty**

In “TLSChain” blockchain based Public Key Infrastructure, we have proposed a novel on-chain domain verification and retention scheme without the usage of any attestors. This eliminated the need to having a Certification Authority.

# Appendix A : TLSChain Source Code

## TLSChain Smart Contract

```
pragma solidity >=0.4.24;

import "chainlink/contracts/ChainlinkClient.sol";
import "chainlink/contracts/vendor/Ownable.sol";
import "./SolRsaVerify.sol";

contract TLSChainRegistrationDomain is ChainlinkClient, Ownable {
    address ORACLE_ADDRESS = 0xC612350d871cda87fBFDafb2A9553b680707aca8;
    string constant JOBID = "3d583fa977f14919b01ee09f26f1bbab";
    uint256 private constant ORACLE_PAYMENT = 1 * LINK;

    struct RegInputDetails {
        string sPubHash;
        string oldPubHash;
        string domain;
        address owner;
    }
    mapping(bytes32 => RegInputDetails) InputRequests;

    enum State {Invalid,Valid,NotRegistered}
    struct Certificate {
        string domain;
        uint256 dateTime;
        State state;
        address owner;
    }
    mapping(string => Certificate) cerMapping;
    mapping(string => string) domainMapping;

    constructor(address _link) public Ownable() {
        setChainlinkToken(_link);
    }
}
```

---

```

}

function getChainlinkToken() public view returns (address) {
    return chainlinkTokenAddress();
}

/*****RequestRegister*****/

function requestRegister(
    string memory sPubHash_,
    string memory domain_,
    bytes memory message_,
    bytes memory exponent_,
    bytes memory modulus_,
    bytes memory signature_
) public {
    uint valid_sig_ = retrieveResult(message_,signature_,exponent_,modulus_);
    require (valid_sig_ == 0, "Signature Validation Failed");
    bytes memory tempEmptyPubHash = bytes(domainMapping[domain_]);
    require (tempEmptyPubHash.length == 0, "This Domain has a Certificate Attached");
    bytes memory tempEmptyDomain = bytes(cerMapping[sPubHash_].domain);
    require (tempEmptyDomain.length == 0, "This Certificate has a Domain Attached");
    Chainlink.Request memory req = buildChainlinkRequest(
        stringToBytes32(JOBID),
        address(this),
        this.fulfillRequestRegister.selector
    );
    string memory URL= concat(concat("http://",domain_),"/TLSChain.json");
    req.add("get", URL);
    req.add("path", "TLSChain");
    RegInputDetails memory regInputDetails = RegInputDetails({
        sPubHash: sPubHash_,
        oldPubHash: "",
        domain: domain_,
        owner: msg.sender});
    bytes32 requestId = sendChainlinkRequestTo(ORACLE_ADDRESS, req, ORACLE_PAYMENT);
    InputRequests[requestId] = regInputDetails;
}

function fulfillRequestRegister(bytes32 _requestId, bytes32 _result)
    public recordChainlinkFulfillment(_requestId)
{
    bytes32 fromSever = _result;
    register(_requestId,fromSever);
}

function register(bytes32 requestId_, bytes32 fromSever_) public {
    string sPubHash_ = InputRequests[requestId_].sPubHash;
    string domain_ = InputRequests[requestId_].domain;
    address owner_ = InputRequests[requestId_].owner;
}

```

---

```

require(fromSever_ == stringToBytes32(sPubHash_), "Server Value Not Matching");
    cerMapping[sPubHash_] = Certificate({
        domain: domain_,
        dateTime: now,
        state: State.Valid,
        owner: owner_
    });
    domainMapping[domain_] = sPubHash_;
}

/*****Request Register*****/

/*****Request Renew Domain Validity*****/
function requestRenewDomainValidity(
    string memory sPubHash_,
    string memory domain_,
    bytes memory message_,
    bytes memory exponent_,
    bytes memory modulus_,
    bytes memory signature_
) public {
    uint valid_sig_ = retrieveResult(message_, signature_, exponent_, modulus_);
    require (valid_sig_ == 0, "Signature Not Matching");
    require (msg.sender == cerMapping[sPubHash_].owner, "Only Domain Owner Can Perform
    This Function");
    Chainlink.Request memory req = buildChainlinkRequest(
        stringToBytes32(JOBID),
        address(this),
        this.fulfillRequestRenewDomainValidity.selector
    );
    string memory URL = concat(concat("http://", domain_), "/TLSChain.json");
    req.add("get", URL);
    req.add("path", "TLSChain");
    RegInputDetails memory regInputDetails = RegInputDetails({
        sPubHash: sPubHash_,
        oldPubHash: "",
        domain: domain_,
        owner: msg.sender});
    bytes32 requestId = sendChainlinkRequestTo(ORACLE_ADDRESS, req, ORACLE_PAYMENT);
    InputRequests[requestId] = regInputDetails;
}

function fulfillRequestRenewDomainValidity(bytes32 _requestId, bytes32 _result)
    public recordChainlinkFulfillment(_requestId)
{
    bytes32 fromSever = _result;
    renewDomainValidity(_requestId, fromSever);
}

```



---

```

function renewDomainValidity(bytes32 requestId_, bytes32 fromSever_) public {
    string sPubHash_ = InputRequests[requestId_].sPubHash;
    string domain_ = InputRequests[requestId_].domain;
    address owner_ = InputRequests[requestId_].owner;
    require(fromSever_ == stringToBytes32(sPubHash_), "Server Value Not Matching");
    cerMapping[sPubHash_] = Certificate({
        domain: domain_,
        dateTime: now,
        state: State.Valid,
        owner: owner_
    });
}

/*****Request Renew Domain Validity*****/

/*****Request Renew Domain Validity*****/

function requestRenewDomainPubKey(
    string memory sPubHash_,
    string memory oldSPubHash_,
    string memory domain_,
    bytes memory message_,
    bytes memory exponent_,
    bytes memory modulus_,
    bytes memory signature_
) public {
    uint valid_sig_ = retrieveResult(message_, signature_, exponent_, modulus_);
    require (valid_sig_ == 0, "Signature Not Matching");
    require (msg.sender == cerMapping[oldSPubHash_].owner, "Only Domain Owner
    Can Perform This Function");
    require ((keccak256(bytes(cerMapping[oldSPubHash_].domain))) ==
(keccak256(bytes(domain_)))));
    Chainlink.Request memory req = buildChainlinkRequest(
        stringToBytes32(JOBID),
        address(this),
        this.fulfillRequestRenewDomainPubKey.selector
    );
    string memory URL = concat(concat("http://", domain_), "/TLSChain.json");
    req.add("get", URL);
    req.add("path", "TLSChain");
    RegInputDetails memory regInputDetails = RegInputDetails({
        sPubHash: sPubHash_,
        oldPubHash: oldSPubHash_,
        domain: domain_,
        owner: msg.sender});
    bytes32 requestId = sendChainlinkRequestTo(ORACLE_ADDRESS, req, ORACLE_PAYMENT);
    InputRequests[requestId] = regInputDetails;
}

function fulfillRequestRenewDomainPubKey(bytes32 _requestId, bytes32 _result)

```

---

```

    public recordChainlinkFulfillment(_requestId)
    {
        bytes32 fromSever = _result;
        renewDomainPubKey(_requestId,fromSever);
    }

    function renewDomainPubKey(bytes32 requestId_, bytes32 fromSever_) public {
        string sPubHash_ = InputRequests[requestId_].sPubHash;
        string oldSPubHash_ = InputRequests[requestId_].oldPubHash;
        string domain_ = InputRequests[requestId_].domain;
        address owner_ = InputRequests[requestId_].owner;
        require(fromSever_ == stringToBytes32(sPubHash_), "Server Value Not Matching");
        revoke(oldSPubHash_);
        cerMapping[sPubHash_] = Certificate({
            domain: domain_,
            dateTime: now,
            state: State.Valid,
            owner: owner_
        });
    }

/*****Request Renew Domain Validity*****/

/****Signature Verification*****/
    function retrieveResult(bytes message_, bytes signature_, bytes exponent_, bytes modulus_)
    public returns (uint256) {
        uint256 i = SolRsaVerify.pkcs1Sha256VerifyRaw(
            message_,
            signature_,
            exponent_,
            modulus_
        );
        return i;
    }

/****Signature Verification*****/

/****Revoke a Domain*****/
    function revoke(string pubHash_){
        require (msg.sender == cerMapping[pubHash_].owner, "Only Domain Owner Can Perform
        This Function");
        cerMapping[pubHash_].state = State.Invalid;
        cerMapping[pubHash_].dateTime = block.timestamp;
    }

/****Revoke a Domain*****/

/****Retrieve a Domain*****/
    function retrieve(string pubHash_)
    public
    view
    returns (string, State, uint256, address)

```

---

```

    {
        if ((bytes(cerMapping[pubHash_].domain)).length == 0){
            return ('No Associated Domain for this Certificate', cerMapping[pubHash_].state, 0, 0);
        }else{
            return (cerMapping[pubHash_].domain, cerMapping[pubHash_].state,
                cerMapping[pubHash_].dateTime, cerMapping[pubHash_].owner);
        }
    }
}

/*****Retrieve a Domain*****/

/*****Supporting Functions*****/
function withdrawLink() public onlyOwner {
    LinkTokenInterface link = LinkTokenInterface(chainlinkTokenAddress());
    require(
        link.transfer(msg.sender, link.balanceOf(address(this))),
        "Unable to transfer"
    );
}

function stringToBytes32(string memory source)
    public
    pure
    returns (bytes32 result)
{
    bytes memory tempEmptyStringTest = bytes(source);
    if (tempEmptyStringTest.length == 0) {
        return 0x0;
    }
    assembly {
        result := mload(add(source, 32))
    }
}

function toBytes(bytes32 _data) public pure returns (bytes) {
    return abi.encodePacked(_data);
}

function bytes32ToStr(bytes32 _bytes32)
    public
    pure
    returns (string memory)
{
    bytes memory bytesArray = new bytes(32);
    for (uint256 i; i < 32; i++) {
        bytesArray[i] = _bytes32[i];
    }
    return string(bytesArray);
}

function concat(string memory a, string memory b) public view returns(string memory){

```

---

```

        return(string(abi.encodePacked(a,"",b)));
    }

    /*****Supporting Functions*****/
}

```

## TLSCChain CLI

```

import hashlib
import base64
import os
import argparse
import json
import time
from web3 import Web3
from Crypto.PublicKey import RSA
from OpenSSL import crypto

# Constants
ganache_url = "http://127.0.0.1:7545"
contract_address = "0xdF4FC5bE335DA5c2403916934f61A6690782c5DF"
path = "../ChainLinkProject2/truffle/build/contracts/TLSCChainRegistrationDomain.json"

def runConfig():
    web3_client = Web3(Web3.HTTPProvider(ganache_url))
    connected_to_ganache = web3_client.isConnected()
    print(f'\nConnected to Ethereum Network: {connected_to_ganache}!')
    if (connected_to_ganache):
        with open(path) as f:
            data = json.load(f)
            contract_abi = data['abi']
            contract_checksum_address = Web3.toChecksumAddress(contract_address)
            contract = web3_client.eth.contract(address=contract_checksum_address, abi=contract_abi)
        return connected_to_ganache, web3_client, contract

# Get the public key hash to place in the public folder of the server
def getPubHash(CERT_FILE):
    file_path = os.path.join(os.getcwd(), CERT_FILE)
    f = open(file_path, "r")
    cert = f.read()

    pubkey = RSA.importKey(cert)
    modulus = "{0:#0{1}x}".format(pubkey.n, 256)
    exponent = "{0:#0{1}x}".format(pubkey.e, 256)

    pub_key_obj = crypto.load_certificate(crypto.FILETYPE_PEM, cert).get_pubkey()

```

---

```

pub_key = crypto.dump_publickey(crypto.FILETYPE_ASN1, pub_key_obj)

m = hashlib.sha256()
m.update(pub_key)
digest = m.digest()
encoded = base64.b64encode(digest)
print(f'\nPublic Key Hash of Certificate {CERT_FILE} : {encoded}')

return modulus, exponent, encoded

def getSignature(KEY_FILE, plainText):
    key_file = open(KEY_FILE, "r")
    key = key_file.read()
    key_file.close()

    if key.startswith('-----BEGIN '):
        prkey = crypto.load_privatekey(crypto.FILETYPE_PEM, key)
    else:
        prkey = crypto.load_pkcs12(key).get_privatekey()

    sign = crypto.sign(prkey, plainText, "sha256")
    signature = sign.hex()
    return signature

#Register a public key with a domain
def registerDomain(CERT_FILE, KEY_FILE, domain_name, wallet_address, wallet_private_key):

    connected_to_ganache, web3_client, contract = runConfig();
    modulus, exponent, encoded = getPubHash(CERT_FILE)
    signature = getSignature(KEY_FILE, encoded)

    if (connected_to_ganache):
        print(f'\nRegister Domain Public Key in TLSChain: {domain_name}')
        txn_dict = contract.functions.requestRegister(encoded, domain_name, encoded.hex(),
            exponent, modulus, signature).buildTransaction({
            'from': wallet_address,
            'gas': 2000000,
            'gasPrice': web3_client.toWei('50', 'gwei'),
            'nonce': web3_client.eth.getTransactionCount(wallet_address)
        })
        try:
            signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
            result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
            tx_receipt = web3_client.eth.waitForTransactionReceipt(result)
        except ValueError as e:
            print(e.args[0]['message'])

    time.sleep(10)
    retrieveValidityHash(encoded)

```

---

```

#Renew a domain valifiy
def renewDomainValidity(CERT_FILE,KEY_FILE, domain_name ,wallet_address ,
wallet_private_key):

    connected_to_ganache,web3_client,contract = runConfig();
    modulus,exponent,encoded = getPubHash(CERT_FILE)
    signature = getSignature(KEY_FILE,encoded)

    if (connected_to_ganache):
        print(f'\nRenew Domain Public Key Validiy in TLSChain: {domain_name}')
        txn_dict = contract.functions.requestRenewDomainValidity(encoded, domain_name,encoded.hex(),
exponent,modulus,signature).buildTransaction({
            'from': wallet_address,
            'gas': 2000000,
            'gasPrice': web3_client.toWei('50', 'gwei'),
            'nonce': web3_client.eth.getTransactionCount(wallet_address)
        })
        try:
            signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
            result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
            tx_receipt = web3_client.eth.waitForTransactionReceipt(result)
        except ValueError as e:
            print(e.args[0]['message'])

        time.sleep(10)
        retriveValidityHash(encoded)

#Renew a domain public key
def RenewDomainPubKey(CERT_FILE,OLD_CERT_FILE,KEY_FILE, domain_name ,wallet_address ,
wallet_private_key):

    connected_to_ganache,web3_client,contract = runConfig();
    modulus,exponent,encoded = getPubHash(CERT_FILE)
    signature = getSignature(KEY_FILE,encoded)

    omodulus,oexponent,oencoded = getPubHash(OLD_CERT_FILE)

    if (connected_to_ganache):
        print(f'\nRenew Public for a Domain in TLSChain: {domain_name}')
        txn_dict = contract.functions.requestRenewDomainPubKey(encoded, oencoded,domain_name ,
encoded.hex(),exponent,modulus,signature).buildTransaction({
            'from': wallet_address,
            'gas': 2000000,
            'gasPrice': web3_client.toWei('50', 'gwei'),
            'nonce': web3_client.eth.getTransactionCount(wallet_address)
        })
        try:
            signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
            result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
            tx_receipt = web3_client.eth.waitForTransactionReceipt(result)

```

---

```

except ValueError as e:
    print(e.args[0]['message'])

time.sleep(10)
retriveValidityHash(encoded)

# Retrieve the validity using the certificate
def retriveValidityCer(CERT_FILE):

    connected_to_ganache,web3_client,contract = runConfig();
    modulus,exponent,encoded = getPubHash(CERT_FILE)

    if (connected_to_ganache):
        # Retrieve the value to verify
        retrieved_domain_name, domain_is_valid, dateTime, address =
        contract.functions.retrieve(encoded).call()
        print(f'\nQuery from Ethereum:')
        print(f'Domain Name: {retrieved_domain_name}')
        print(f'Domain Is Valid: {domain_is_valid}')
        if dateTime != 0:
            dateTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(dateTime))
            print(f'Registered Date: {dateTime}')
            print(f'Owner: {address}')

# Retrieve the validity using the public key hash
def retriveValidityHash(pubHash):

    connected_to_ganache,web3_client,contract = runConfig();
    # Retrieve the value to verify
    retrieved_domain_name, domain_is_valid, dateTime, address =
    contract.functions.retrieve(pubHash).call()
    print(f'\nQuery from Ethereum:')
    print(f'Domain Name: {retrieved_domain_name}')
    print(f'Domain Is Valid: {domain_is_valid}')
    if dateTime != 0:
        dateTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(dateTime))
        print(f'Registered Date: {dateTime}')
        print(f'Owner: {address}')

def revokeDomain(CERT_FILE,wallet_address,wallet_private_key):

    connected_to_ganache,web3_client,contract = runConfig ();
    modulus,exponent,encoded = getPubHash(CERT_FILE)

    if (connected_to_ganache):
        print(f'\n Revoke Domain Public Key Validiy in TLSChain')
        txn_dict = contract.functions.revoke(encoded).buildTransaction({
            'from': wallet_address,
            'gas': 2000000,

```

---

```

        'gasPrice': web3_client.toWei('50', 'gwei'),
        'nonce': web3_client.eth.getTransactionCount(wallet_address)
    })
    try:
        signed_txn = web3_client.eth.account.signTransaction(txn_dict, wallet_private_key)
        result = web3_client.eth.sendRawTransaction(signed_txn.rawTransaction)
        tx_receipt = web3_client.eth.waitForTransactionReceipt(result)
    except ValueError as e:
        print(e.args[0]['message'])

    time.sleep(10)
    # Retrieve the value to verify
    revokedDetails(encoded)

def revokedDetails(pubHash):

    connected_to_ganache, web3_client, contract = runConfig();
    # Retrieve the value to verify
    retrieved_domain_name, domain_is_valid, dateTime, address =
    contract.functions.retrieve(pubHash).call()
    print(f'\nQuery from Ethereum:')
    if dateTime != 0:
        dateTime = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(dateTime))
    if domain_is_valid == 1:
        print(f'Domain Public Key not Revoked')
        print(f'Registered Date: {dateTime}')
    else:
        print(f'Domain Public Key Revoked')
        print(f'Revoked Date: {dateTime}')
    print(f'Domain Name: {retrieved_domain_name}')
    print(f'Domain Is Valid: {domain_is_valid}')
    print(f'Owner: {address}')

if __name__ == '__main__':
    parser = argparse.ArgumentParser(prog='TLSSChain',
        usage='An Alternative to CA using Blockchain based Decentralized PKI',
        description='',
        '''
        *****
        *                                     *
        * This is TLSSChainCLI Decentralized PKI *
        *                                     *
        *****
        ''',
        epilog="Copyrights @ Anne Fernando",
        formatter_class=argparse.RawDescriptionHelpFormatter,
        add_help=True
    )

    parser.add_argument("command", type=str, help= "'getPubHash -c', 'registerDomain -c -p -d -wa -wp")

```



---

```

'retrieveValidityCer -c','retrieveValidityHash -hp','revokeDomain -c -wa -wp',
'renewDomainValidity -c -p -d -wa -wp' , 'requestRenewDomainPubKey -c -oc -p -d -wa -wp'",
choices= ['registerDomain','getPubHash','retrieveValidityCer','retrieveValidityHash',
'revokeDomain','renewDomainValidity','requestRenewDomainPubKey'], metavar="Commands")
parser.add_argument("-hp",type=str, help="Public Key Hash", required=False)
parser.add_argument("-c",type=str, help="Certificate File Name", required=False)
parser.add_argument("-oc",type=str, help="Old Certificate File Name", required=False)
parser.add_argument("-p",type=str, help="Private Key File", required=False)
parser.add_argument("-d",type=str, help="Domain Name", required=False)
parser.add_argument("-wa",type=str, help="Wallet Address", required=False)
parser.add_argument("-wp",type=str, help="Wallet Private Key", required=False)

arg = parser.parse_args()
if arg.command == 'getPubHash':
getPubHash(arg.c)
elif arg.command == 'registerDomain':
registerDomain(arg.c,arg.p,arg.d,arg.wa,arg.wp)
elif arg.command == 'retrieveValidityCer':
retrieveValidityCer(arg.c)
elif arg.command == 'retrieveValidityHash':
retrieveValidityHash(arg.hp)
elif arg.command == 'revokeDomain':
revokeDomain(arg.c,arg.wa,arg.wp)
elif arg.command == 'renewDomainValidity':
renewDomainValidity(arg.c,arg.p,arg.d,arg.wa,arg.wp)
elif arg.command == 'requestRenewDomainPubKey':
RenewDomainPubKey(arg.c,arg.oc,arg.p,arg.d,arg.wa,arg.wp)

```

## TLSCChain Web Extension

```

const Web3 = require("../web3");
const contractInterface = require("../abi/TLSCChainRegistrationDomain.json");

// Ganache Endpoint
const rpcURL = "http://127.0.0.1:7545";

// Web3 Client
const web3 = new Web3(rpcURL);

// SmartContract
const smartContractAddress = "0xdF4FC5bE335DA5c2403916934f61A6690782c5DF";
const smartContract = new web3.eth.Contract(
    contractInterface,
    smartContractAddress
);

```

---

```

// Current Active Tab URL
let currentTabUrl;

async function getSecInfo(requestDetails) {
  // console.log("inside getSecInfo");
  console.log(
    'Got a request for ${requestDetails.url} with ID ${requestDetails.requestId}'
  );

  // Yeah this is a String, even though the content is a Number
  var requestId = requestDetails.requestId;

  var securityInfo = await browser.webRequest.getSecurityInfo(requestId, {
    certificateChain: false,
    rawDER: false,
  });

  if (securityInfo && securityInfo.certificates) {
    // console.log('securityInfo : ${JSON.stringify(securityInfo, null, 2)}');

    const certificateKeyHash = JSON.stringify(
      securityInfo.certificates[0].subjectPublicKeyInfoDigest.sha256,
      null,
      2
    );
    console.log(
      'securityInfo subjectPublicKeyInfoDigest: ${certificateKeyHash}'
    );
    console.log(web3.utils.toHex(certificateKeyHash));

    const KeyHash = certificateKeyHash.replace(/['"]+/g, "");

    const result = await smartContract.methods.retrieve(KeyHash).call();
    const {
      0: retrieved_domain_name,
      1: domain_is_valid,
      2: dateTime,
    } = result;

    console.log(result);
    console.log('retrieved_domain_name == ${retrieved_domain_name}');
    console.log('dateTime == ${dateTime}');
    console.log('domain_is_valid == ${domain_is_valid}');

    if (requestDetails.url === currentTabUrl) {
      let $statusIndicator = $("#status");
      $statusIndicator.removeClass("loader active");

      if (domain_is_valid === "1") {
        $statusIndicator.addClass("success animated pulse");
      }
    }
  }
}

```

---

```

        $statusIndicator.text("Passed");

        console.log("Domain is valid");
    } else {
        $statusIndicator.addClass("failure animated pulse");
        $statusIndicator.text("Failed");

        console.log("Domain is not valid");
    }
}
}
}

async function loadCurrentCertData() {
    await new Promise((resolve) => {
        setTimeout(() => {
            let $statusIndicator = $("#status");

            $statusIndicator.addClass("active");
            $statusIndicator.addClass("loader");

            resolve();
        }, 1000);
    });

    await fetch(currentTabUrl);
}

async function main() {
    currentTabUrl = (
        await browser.tabs.query({ currentWindow: true, active: true })
    )[0].url;

    web3.eth.net.isListening().then((data) => {
        // console.log("Connected to Ganache ...");
    });

    browser.webRequest.onHeadersReceived.addListener(
        getSecInfo,
        {
            urls: ["<all_urls>"],
        },
        ["blocking"]
    );

    loadCurrentCertData();
}

main();

```

# Bibliography

- Brunner, C., Knirsch, F., Unterweger, A. & Engel, D. (2020). A comparison of blockchain-based pki implementations., *ICISSP*, pp. 333–340. [10](#)
- Buterin, V. et al. (2014). A next-generation smart contract and decentralized application platform, *white paper* **3**(37). [10](#), [13](#)
- Caronni, G. (2000). Walking the web of trust, *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*, IEEE, pp. 153–158. [19](#)
- chain.link (2020).  
**URL:** <https://chain.link/features/> [16](#)
- Constantin, L. (2012). Trustwave admits issuing man-in-the-middle digital certificate; mozilla debates punishment.  
**URL:** <https://www.computerworld.com/article/2501291/trustwave-admits-issuing-man-in-the-middle-digital-certificate-mozilla-debates-punishment.html> [3](#)
- ethereum.org (2020).  
**URL:** <https://ethereum.org/en/developers/docs/dapps/> [15](#)
- Fisher, D. (2012). Final report on diginotar hack shows total compromise of ca servers.  
**URL:** <https://threatpost.com/final-report-diginotar-hack-shows-total-compromise-ca-servers-103112/77170/> [3](#)

- Fromknecht, C., Velicanu, D. & Yakoubov, S. (2014). A decentralized public key infrastructure with identity retention., *IACR Cryptol. ePrint Arch.* **2014**: 803. [30](#)
- Kfoury, E. F., Khoury, D., AlSabeh, A., Gomez, J., Crichigno, J. & Bou-Harb, E. (2020). A blockchain-based method for decentralizing the acme protocol to enhance trust in pki, *2020 43rd International Conference on Telecommunications and Signal Processing (TSP)*, IEEE, pp. 461–465. [25](#)
- Laurie, B., Langley, A. & Kasper, E. (2013). Certificate transparency. [25](#)
- layer 2 ethereum.org (2021).  
**URL:** <https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/> [102](#)
- LetsEncrypt (n.d.). How it works.  
**URL:** <https://letsencrypt.org/how-it-works/> [26](#)
- Matsumoto, S. & Reischuk, R. M. (2017). Ikp: Turning a pki around with decentralized automated incentives, *2017 IEEE Symposium on Security and Privacy (SP)* . [25](#)
- Naik, N. & Jenkins, P. (2020). uport open-source identity management system: An assessment of self-sovereign identity and user-centric data platform built on blockchain, *2020 IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–7. [102](#)
- Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system, *Technical report*, Manubot. [5](#), [13](#)
- namecoin (n.d.).  
**URL:** <https://namecoin.org/> [30](#)
- Patsonakis, C., Samari, K., Kiayias, A. & Roussopoulos, M. (2020). Implementing a smart contract pki, *IEEE Transactions on Engineering Management* . [31](#)

- Patsonakis, C., Samari, K., Roussopoulos, M. & Kiayias, A. (2017). Towards a smart contract-based, decentralized, public-key infrastructure, *International Conference on Cryptology and Network Security*, Springer, pp. 299–321. [31](#)
- Rutland, E. (2018). Blockchain byte: R3 research, *Technical report*. [8](#)
- Schoen, S. (2017). New research suggests that governments may fake ssl certificates.  
**URL:** <https://www.eff.org/deeplinks/2010/03/researchers-reveal-likelihood-governments-fake-ssl> [3](#)
- Sermpinis, T., Vlahavas, G., Karasavvas, K. & Vakali, A. (2020). Detract: a decentralized, transparent, immutable and open pki certificate framework, *International Journal of Information Security* pp. 1–18. [29](#)
- Wikipedia contributors (2020). Public key infrastructure — Wikipedia, the free encyclopedia. [Online; accessed 16-November-2020].  
**URL:** [https://en.wikipedia.org/w/index.php?title=Public\\_key\\_infrastructure&oldid=986928996](https://en.wikipedia.org/w/index.php?title=Public_key_infrastructure&oldid=986928996) [19](#)
- Wolff, J. (2016). How a 2011 hack you’ve never heard of changed the internet’s infrastructure.  
**URL:** <https://slate.com/technology/2016/12/how-the-2011-hack-of-diginotar-changed-the-internets-infrastructure.html> [2](#)
- Yu, J. & Ryan, M. (2017). Evaluating web pkis.  
**URL:** <https://www.sciencedirect.com/science/article/pii/B9780128054673000077> [25](#)
- Zhao, J., Lin, Z., Huang, X., Zhang, Y. & Xiang, S. (2020). Trustca: Achieving certificate transparency through smart contract in blockchain platforms, *2020 International Conference on High Performance Big Data and Intelligent Systems (HPBDIS)*, IEEE, pp. 1–6. [14](#), [28](#)