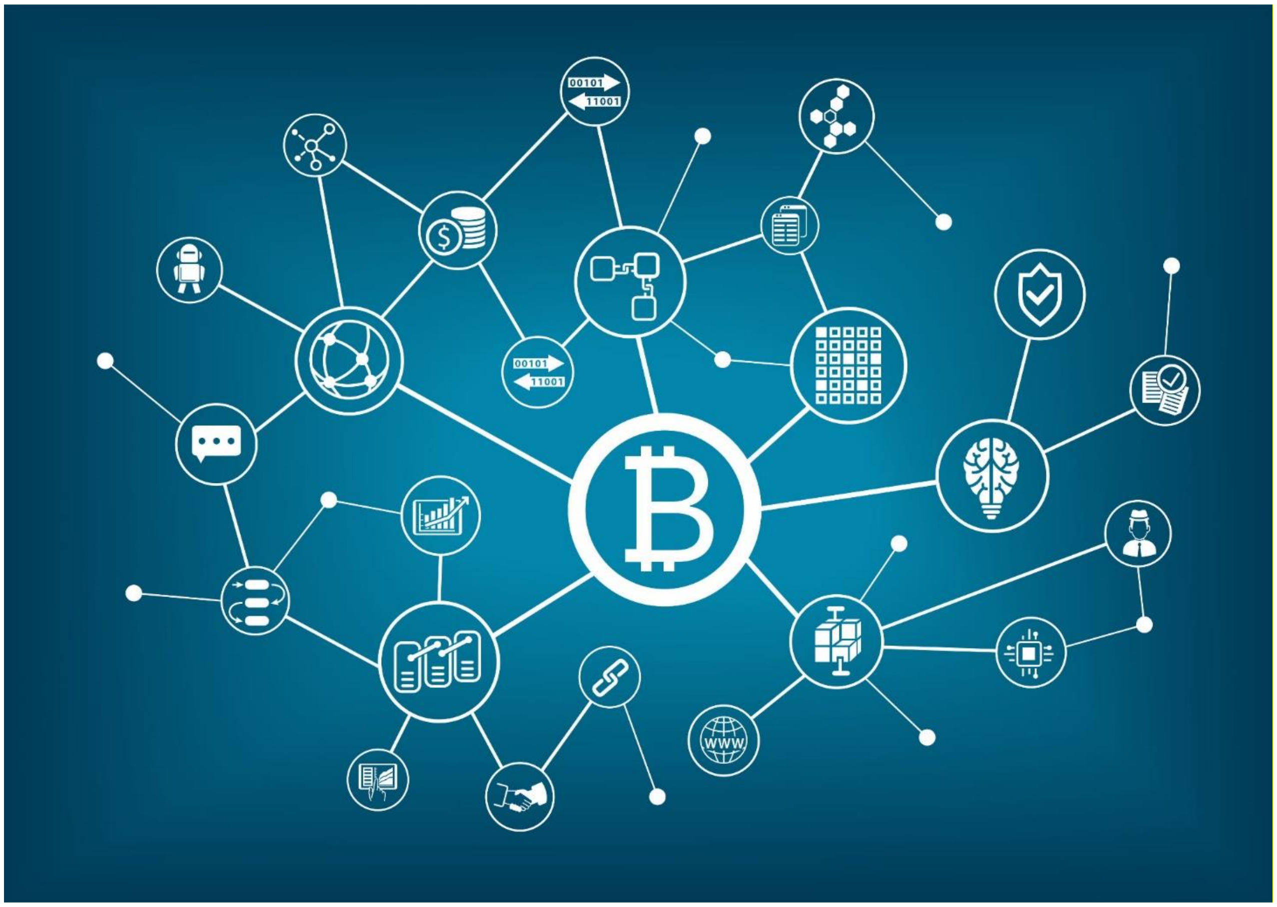


Customized Blockchain for SSL Certificates

By

Jahangir Maqsood (Bscs17026)

Muhammad Muzzamil (Bscs17065)

Muhammad Muzzamil (Bscs17073)

Muhammad Muneeb Yasir (Bscs17075)

Hassan Munir (Bscs17077)

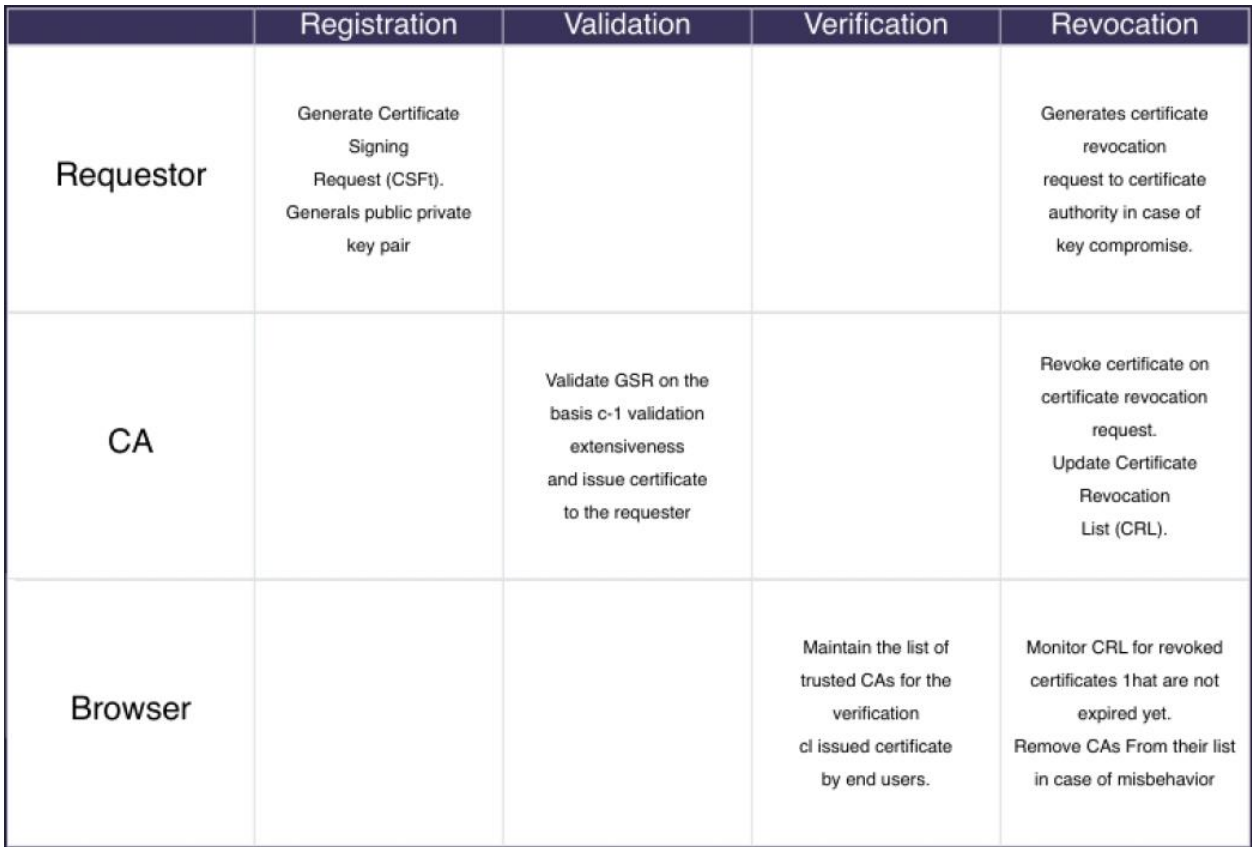
Table of Content

1. Summary

This paper is written for the absolute purpose of implementing the customized blockchain for SSL Certificates. Previously there was only one miner who was responsible to perform this. We have implemented it using decentralization and multiple miners can be used now. Centralized CA uses Public Key Infrastructure that verifies user's identity thus aims to provide secure communication channels. However. the security of PKI is profoundly reliant on the reliability of these third- party CAs which serves as a single point of fail u re for PKI. Over the past. there have been several incidents of popular CA breaches, where the centralized operation model of CAs ca u sed numerous targeted attacks due to the spread of rogue certificates. We use Proof-Chain as a decentralized certificate authority which provides the basic operations of CAs such as registration, validation, verification, and revocation. The validation and registration is the job of miners. In this way the validated certificate becomes a part of the ledger and becomes trusted. Moreover. Miners can also revoke a certificate upon the request of domain owner resulting in minimized cost, improved security and less reliability on a third-party Certificate Authority

2. Introduction

We have implemented this project using the basis of proofchain proposed algorithm. The implementation is an extended version of the code that we have been provided by our assigned project supervisor Ms Tania Saleem. CAs are the trusted third parties that play the role of verifying the key binding used in the public key infrastructure (PKI). CA mainly used four main processes for providing secure key binding: Registration, Validation, Verification, Revocation. There are three primary actors involved in these four processes: Requestor, Certificate Authorities (CAs) and Browsers.



	Registration	Validation	Verification	Revocation
Requestor	Generate Certificate Signing Request (CSFt). Generals public private key pair			Generates certificate revocation request to certificate authority in case of key compromise.
CA		Validate GSR on the basis c-1 validation extensiveness and issue certificate to the requester		Revoke certificate on certificate revocation request. Update Certificate Revocation List (CRL).
Browser			Maintain the list of trusted CAs for the verification cl issued certificate by end users.	Monitor CRL for revoked certificates 1hat are not expired yet. Remove CAs From their list in case of misbehavior

Figure 1: It shows the roles of primary actors involved in secure key binding.

3. Literature Review

3.1 What is SSL?

SSL, more commonly called TLS, is a protocol for encrypting Internet traffic and verifying server identity. Any website with an HTTPS web address uses SSL/TLS. SSL, or Secure Sockets Layer, is an encryption-based Internet security protocol. It was first developed by Netscape in 1995 for the purpose of ensuring privacy, authentication, and data integrity in Internet communications. SSL is the predecessor to the modern TLS encryption used today. A website that implements SSL/TLS has "HTTPS" in its URL instead of "HTTP."

3.2 How does SSL/TLS work?

1. In order to provide a high degree of privacy, SSL encrypts data that is transmitted across the web. This means that anyone who tries to intercept this data will only see a garbled mix of characters that is nearly impossible to decrypt.
2. SSL initiates an **authentication** process called a handshake between two communicating devices to ensure that both devices are really who they claim to be.
3. SSL also digitally signs data in order to provide **data integrity**, verifying that the data is not tampered with before reaching its intended recipient.

There have been several iterations of SSL, each more secure than the last. In 1999 SSL was updated to become TLS.

3.3 What information does an SSL certificate contain?

SSL certificates include:

4. The domain name that the certificate was issued for
5. Which person, organization, or device it was issued to
6. Which certificate authority issued it
7. The certificate authority's digital signature
8. Associated subdomains
9. Issue date of the certificate
10. Expiration date of the certificate
11. The public key (the private key is kept secret)

The public and private keys used for SSL are essentially long strings of characters used for encrypting and decrypting data. Data encrypted with the public key can only be decrypted with the private key, and vice versa.

3.4 Why do websites need an SSL certificate?

A website needs an SSL certificate in order to keep user data secure, verify ownership of the website, prevent attackers from creating a fake version of the site, and gain user trust.

Encryption: SSL/TLS encryption is possible because of the public-private key pairing that SSL certificates facilitate. Clients (such as web browsers) get the public key necessary to open a TLS connection from a server's SSL certificate.

Authentication: SSL certificates verify that a client is talking to the correct server that actually owns the domain. This helps prevent domain spoofing and other kinds of attacks.

HTTPS: Most crucially for businesses, an SSL certificate is necessary for an HTTPS web address. HTTPS is the secure form of HTTP, and HTTPS websites are websites that have their traffic encrypted by SSL/TLS.

In addition to securing user data in transit, HTTPS makes sites more trustworthy from a user's perspective. Many users won't notice the difference between an http:// and an https:// web address, but most browsers have started tagging HTTP sites as "not secure" in more noticeable ways, attempting to provide incentive for switching to HTTPS and increasing security.

3.5 How does a website obtain an SSL certificate?

For an SSL certificate to be valid, domains need to obtain it from a certificate authority (CA). A CA is an outside organization, a trusted third party, that generates and gives out SSL certificates. The CA will also digitally sign the certificate with their own private key, allowing client devices to verify it. Most, but not all, CAs will charge a fee for issuing an SSL certificate.

Once the certificate is issued, it needs to be installed and activated on the website's origin server. Web hosting services can usually handle this for website operators. Once it's activated on the origin server, the website will be able to load over HTTPS and all traffic to and from the website will be encrypted and secure.

3.6 How domain validation is performed

For a simple Domain Validated certificate, the CA sends email to your registered business email id with a verification link or uses HTTP/HTTPS File Verification or DNS verification method. The verification just confirms that you own the email address or website in question. For higher assurance digital certificates (such as Organization Validation or Extended Validation), the CA will verify details such as your business registration, physical address, the presence of your business on an approved online business directory website, phone number, and/or use other methods to verify the identity of the applicant. When the applicant passes the verification process, the CA issues a digital certificate for his/her website/software/email. One of the most popular among them is an SSL (Secure socket layer) certificate. An SSL certificate is a digital certificate that encrypts the data transferred between a website's server and the client/website visitor's browser. Suppose you visit an eCommerce website using Chrome (or any browser). Now, your browser does not know the website owner. But it knows the CA that issued the SSL certificate to that website. If that website has installed an SSL certificate, your browser will verify the website's certificate, then all data transferred between that website and your browser will be encrypted.

However, this whole process has a major flaw in it that is places the entire trust on a single point of on control and authority for certificate issuance. If the root CA makes a mistake and become untrustworthy then all the websites and sub-CA certificates become untrustworthy. There have been several popular incidents in the past about root CA mess-ups. One of these mess-ups includes the incident of security breach of DigiNotar CA in 2011. In this attack, the attacker maliciously issued rogue certificates for various domains including Google.com

3.7 What is Proof chain and its key Idea

Proof chain is a compatible, robust and decentralized PKI framework which overcomes the security issues in traditional and blockchain based PKI. Proof chain model is designed to meet following Objectives.

Compatibility: It is an enhanced and scalable scheme of PKI. It fulfills all the basic functionalities of standard PKI. It is also able to achieve the distribution and management of certificates which are completely relying on the ledger.

Decentralized Trust: It is reliably decentralized to avoid the security risk in centralization. It provides that the security of the network should be independent to CA.

Prevention from MITM Attacks: The sanity checks are performed during the validation of certificates process. This prevents forging keys in communication between domain and end-users.

Automation: All the certificate operations in blockchain are systematically automated using the consensus protocol and sanity checks.

Transparency: Proof chain maintains the whole history of certificates. In case, any change is being made to a certificate then it becomes visible to whole network.

3.8 Proof-Chain Operations:

1. Registration: request for the issuance of certificate against the domain name.
2. Validation: certificate requests validation to ensure that it is indeed requested from the valid entities and not from the adversary.
3. Revocation: request for the invalidation of the active issued certificate in case of the key compromise.
4. Verification: ensuring that the certificate associated with the domain name is part of the ledger and not expired or revoked.

4. Methodology

4.1 Proof-Chain Entities:

1. Domain Owner: is a registered member of Proofchain which owns the certificate and is able to perform certificate operations of registration and revocation. The domain owner also has access to the server linked to a DNS for which he/she requires a TLS certificate. Domain owner owns a unique public private key pair.
2. CA Miner: each CA miner is an independent node of our system that holds the Proof-Chain ledger state. CA miners posses a unique certificate that enables them to exercise their authority as a CA miner is Proofchain. Each CA miner also possesses a unique public/private key pair in Proofchain. CA miners are allowed to perform the operation of validation.
3. End-Users: are the entities that communicate with domains owned by domain owners. These entities need to ensure that they are communicating securely with the domain by checking the legitimacy of issued certificates. End users perform the operation of verification. \

```
class Proof chain : GENESIS = (
"indexl' : O,
"ert": None,
"gs": None,
"ids": None,
"domain": None,
"pie": None,
"sig": None,
"expiry": None, "CAsigll: None
)
```

4.2 Proof-Chain workings:

In our project. Miners are continuously looking for the requests to validate a certificate or revoke the certificate. Depending on the Proofchain in operation, the domain owner generates a certificate request (CRT) of either type initial or revoke. Proofchain converts this certificate request into a transaction for CA miners to process and the transaction goes into the Proofchain in validated transaction pool. For Simplicity, our project does not require a miner to perform validation by providing some proof-of-issuance. Every time a request is generated, the miner is bound to follow up on it.

```
def create cert for(self,crt):

self.cert.set subject(crt.get
subject())#req added

        self.cert.gmtime_adj_notAfter(365)

        self.cert.gmtime adj
notBefore(0)#valid after 0 seconds


self.cert.set issuer(self.identity.get subjec

t())
        print(self.identity.set_subject().CN)
        self.cert.set_pubkey(crt.get_pubkey())
```

4.3 Registration and Validation of a Certificate:

A domain owner who intends a certificate for his website submits CRT initial transaction for his domain

```
class Transaction: def   init (self, domain, pk,
sig, cert):
```

```
                    self.domain = domain

                    self.pk = pk

                    self.sig = sig

                    self.crt = cert
```

Our project tries the following checks for validation or registration of a certificate:

- Key Check: check for domain owner public key binding with the domain owner signature to ensure that the domain owner holds both public and private key for the domain.
- Domain Check: check for the existing domain public key binding in the Proofchain ledger. This is to ensure that another domain is not attempting to forge the existing domain.
- Domain ownership Check: ask the domain owner to provide signed token tO' on the DIMS associated with a domain, Only the domain owner has access to the associated DNS, therefore an adversary cannot fulfill this validation check. The token is the hash of the transaction under validation with the CA miner's public key collectively signed by the CA miner using his secret key.

```
           def validatel():

      token = request.files['token'].read()
      issued= request.files['issued'].read() crt
      = request.files['crt'].read()

               crt =

           crypto.load_certificaterequest(cryptoFILETY
           PE PEM,crt)

               domain=
               crt.get_subje
               ct().CN tx =
               proofchain.utp
               .get(domain)

               crt = ROOTCA.create_cert_for(crt)

               ROOTCA.mytokens.setdefault(domain,
           (token,issued,tx,crt))

           user returned
               tokens.setdefault(domain,token) return

           render_template("errusr.html",header="Request
           submitted",

           content="Waitfor Validation. You cannavigate to 'Get
           Certifcate'")
```

- Token Placement: The requester which is the domain owner in our system signs the token with his secret key, places it on his website and notifies the CA miner.

```
def signtoke n():

key= request.files'[key'].read() token=
request.files['token'J . rea  d () name   = req ues
t . for m[ ' name ' ]

key= crypto.load_privatekey(pctro.yFILEYTPE_PEMk,ey)
cert= crypt.oX509()

cert.set_pubkey(key)

f =

open(path.join(app.root_path,"tokens","signed_"+name
+".t"),'wb

' )

f.write(crypto.sign(key, token,'sha256')) f.flush()

return send file(

f.name,

as attachment=True
```

4.4 Revocation of a certificate:

To revoke an existing certificate in case of key compromise. the domain owner submits a transaction with CRT revoke.

- Key Check: check for the public key binding with the signature.
- Domain Check: Check the domain- public key binding that already exists in the ledger. In the case of CRT revoke. there must be a domain public key binding of the requested domain in the ledger.
- Validate revoke a request: ask the requester to place the hash of the last transaction of the same domain name plus the current revoke
- transaction under validation along with his public and notify the CA miner
- Token Placement: Do main Own er places the token on DNS signed by his
- secret key.

- Token Validation: CA miner gets the token in the form of a hash from the website Re-calculate the ha s h at his end a gain. If both the hashes match, then the CA miner validates the transaction, and the certificate is revoked.

```
def key_check(self, pk, sig):

#associates Sk with PK using Sig#SK is a secret
hence not required, sig is enoughif decrypt_sig(pk,
sig)  gen_sigstr(pk):

return True return False


def domain_check(self, domain):#, pk, sig):#returns
false if domain in blockchainfor block in
self.bc.chain:if block.domain== domain:#if
block.pk== user.pk:#if decrypt_sig(pk, block.sig) ==
gen_sigstr(pk):return False

return True


def token_placement(self, trans): #returns a signed
token of trans t = str(c rypto .bl 6e ncode(

self .sig n(hash(hash(t rans)+ self.pk))

) )

self.mytokens.setdefault(t,None) return t


def token validation(self, issuedtoken, signedToken,
cert):

#validates signedtoken using trans and

provided pubk#calculates the token again without
signature as the token issues by miner is not
saved#anywhere hence recalculate it. Not a bug;
decentralization may require it#any miner can verify
issued token
```

```
crypto .verify(cert,signedToken,issuedtoken,'s
ha256')
```

5. Conclusion

As we have seen that in decentralized CA, the cost is minimized because there is not any central company governing all the requests of the clients.

The CA Miners can be incentivized into mining more and more certificate requests. However, the cost of this incentive would still be lower that the cost of centralized Certificate Authorities because there can be any n number of miners whereas there are only few trusted certificate authorities.

Our project on Proof-Chain covers all the traditional PKI operations performed by CA:

- Registration
- Validation
- Verification
- Revocation

Our project does not cover the security aspects of the proof-chain as this was beyond the scope of our project. However, the decentralized PKI Model provides decentralized trust because in Proof-chain, the trust of any certificate or security of the network is not dependent on a single certificate authority. Every certificate issuance in the Proof-Chain is a result of the mutual consensus of certificate authorities in the system. Also, the corruption of one CA cannot do any harm to the certificate issuance process.

Appendix

Janjua, U., 2021. ProofChain: An X.509-Compatible Blockchain-based PKI Framework with Decentralized Trust. Phd. Information Technology University, Punjab.