KORGAN: An Efficient PKI Architecture Based on Permissioned-Blockchain by Modifying PBFT Through Dynamic Threshold Signatures

Murat Yasin Kubilay[1], Mehmet Sabir Kiraz[2], Haci Ali Mantar[1], Ramazan Girgin

[1] Department of Computer Engineering, Gebze Technical University, Kocaeli, Turkey,
[2] De Montfort University, School of Computer Science and Informatics, Leicester, UK
muratkubilay@gtu.edu.tr, mehmet.kiraz@dmu.ac.uk, hamantar@gtu.edu.tr, ramazangirgin@gmail.com

Abstract. During the last decade, several misbehaving Certificate Authorities (CA) have issued fraudulent TLS certificates allowing MITM kinds of attacks which result in serious security incidents. In order to avoid such incidents, Yakubov et al. recently proposed a new PKI architecture where CAs issue, revoke, and validate X.509 certificates on a public blockchain. In their proposal, each CA has a smart contract on the blockchain for publishing the hash values of its issued certificates and managing their revocation status. However, their proposal has several security and privacy issues. First, TLS clients can only validate certificates through either full nodes or web services, but cannot verify the correctness of the incoming responses. Second, certificate transparency is not fully provided because CAs do not store the certificates themselves but only their hash values in the blockchain which makes to detect fake ones impossible.
In this paper, we eliminate the issues of the Yakubov et al.'s scheme and propose a new PKI architecture based on permissioned blockchain with a modified PBFT consensus mechanism. In our modified PBFT, the validators (i.e., the consensus nodes) utilize a dynamic threshold signature scheme to generate signed blocks. In this way, the trust to external entities can be completely eliminated during certificate validation. More concretely, TLS clients can easily verify the genuinity of the final state of the TLS certificates using signed block headers and the Merkle proofs. Also, the privacy of the TLS clients is fully preserved during validation process by avoiding additional communication with the external entities. Our scheme enjoys the dynamic property of the threshold signature because TLS clients do not have to change the verification key even if the validator set is dynamic. Furthermore, TLS clients are also not required to be a peer of the blockchain network and avoid communication overhead. We implement our proposal on private Ethereum network to demonstrate the experimental results. The results show that our proposal has negligible overhead during TLS handshake. The certificate validation duration is less than the duration in the conventional PKI and Yakubov et al.'s scheme.

Keywords: SSL/TLS, PKI, Certificate Transparency, PBFT, Dynamic Threshold Signatures

1 Introduction

TLS is the most widely used protocol in today's internet providing a secure channel in terms of authentication, confidentiality, and integrity between two communicating peers [18]. Authentication of the peers takes place during TLS handshake protocol using X.509 certificates [7]. These certificates are issued, revoked, and managed under a set of policies, roles, and cryptographic methods which have been modelled under the so-called Public Key Infrastructure (PKI). In conventional PKI, CAs are assumed to be trusted organisations which verify the identity of subjects (e.g., domain names) and issue certificates to domains. However, during the last decade, some CAs issued fake but valid certificates for even the well-known domains such as Google, Facebook, Hotmail, GMail, Mozilla, Microsoft [10, 21] which could be used to apply MITM kinds of attacks [9].

In order to reduce the ultimate trust to CAs, several public log based PKI architectures are proposed such as Certificate Transparency (CT) [11], AKI [6], and DTKI [26]. The main goal behind these architectures is to store the certificates in public logs, so that any certificate which has not been added to the logs would be rejected by TLS clients during certificate validation, and a fake certificate could be immediately detected since these logs are publicly visible and monitored by all the related parties. However, these architectures still have several issues if a strong adversary controls all the trusted entities. For an up-to-date list of references, we refer to [9]. Recent studies [4, 9, 20, 24] show that blockchain seems to be a promising technology to eliminate the trust to the public logs by decentralizing their management.

1.1 Our Contributions

In this paper, we first revisit one of the most recent blockchain-based proposals for PKI (i.e., the Yakubov et al.'s scheme), and address its security and privacy issues within their certificate validation during TLS handshake.

More concretely, TLS clients can easily be deceived by fraudulent full nodes or web services during certificate validation, because they cannot verify the validity of the incoming responses. Besides, fake but valid certificates also cannot be detected since only the hash values of the certificates are stored in the blockchain.

In order to elimate these issues, we improve their scheme and propose a new PKI architecture. In summary, our scheme provides the following features.

– We use Practical Byzantine Fault Tolerance Algorithm (PBFT) [3] as the consensus mechanism and modify it in such a way that consensus nodes

hold a share of the blockchain signing key and a block can only be generated if a threshold number [17] of them approve the block by signing it by their key share. Since we use a dynamic threshold signature scheme, once the TLS clients receive the blockchain verification key they do not require to change it even if the set of validators is dynamic.
– During TLS handshake, TLS clients can validate the certificates without requiring to be a peer of the blockchain network. Moreover, they do not need to make any further network connection and query other entities during this process. In this respect, certificate and revocation transparency is now fully provided so that the TLS certificates and their revocation status are publicly monitored. Moreover, the privacy of the TLS clients is fully preserved during certificate validation.
– CAs are not the sole authority to revoke certificates anymore but also certificate owners can revoke their certificates.

We implement a prototype[3] of our proposal on Ethereum, and experiment certificate validation. Our experimental results show that TLS clients can validate certificates efficiently (in constant time) depending on only their processing power and memory. Moreover, TLS handshake overhead is insignificant in our scheme.

1.2 Roadmap

In Section 2, we briefly describe the most recently proposed blockchain based PKI architectures, and highlight their drawbacks. In Section 3, we revisit Yakubov et al.'s scheme and elaborate its security and privacy issues. In Section 4, we first describe our motivation to use dynamic threshold signatures based permissioned blockchains in our PKI architecture, and then describe our modified PBFT consensus mechanism using dynamic threshold signatures. Finally, we present our new PKI architecture, what we called KORGAN, which eliminates the highlighted issues and provides a more efficient construction than the existing schemes. We discuss our implementation and experimental results in Section 5, and conclude the paper with future works in Section 6.

2 Related Work: Blockchain Based PKI Architectures

2.1 Blockchain-based Certificate and Revocation Transparency [20]

Wang et al. in [20] proposed to put all issued TLS certificates and their revocation data (i.e., CRL, OCSP) to the blockchain by their corresponding web

[3] Our prototype is available on https://github.com/efficient-pki-blockchain/.

servers. Each web server has a publishing key pair which is used to sign transactions. A new publishing key has to be approved by a set of web servers using previously approved publishing keys. In this architecture, each transaction has a validity period and the validity period of a certificate addition transaction is shorter than the lifetime of a certificate. Therefore, a certificate is added to the blockchain several times throughout its lifetime. If a certificate is revoked, then the related OCSP response or CRL is also added to the blockchain in a new transaction. During a TLS handshake, a web server sends the Merkle audit proof (standard Merkle tree proof) of its latest certificate addition transaction to the TLS clients. The TLS clients verify the Merkle proof using the block headers which they receive from the P2P network asynchronously. However, this architecture is subject to MITM kinds of attacks in the period of certificate revocation and certificate addition transaction expiration time since certificate addition transaction can be still valid and used for certificate validation even though the certificate is revoked.

2.2 CertChain: Public and Efficient Certificate Audit based on blockchain for TLS Connections [4]

CertChain [4] proposes a certificate management framework to publicly and efficiently audit TLS certificates on a blockchain. In order to eliminate centralization problems of proof-of-work based consensus mechanisms [14, 22], the authors introduce a new consensus protocol based on Ouroboros [8] which incentivizes CAs and the miners for their honest behaviour. In this mechanism, they introduce a new transaction structure which makes possible to search the history of certificates without sequential traversal of all the blocks. Even though *CertChain* proposes to find the revocation status of a certificate efficiently through the bloom filters, it is not clear how the implementation of the bloom filters fit to its transaction structure. TLS clients in *CertChain* ask the validity of the certificates to the miners. They have to rely on their responses which can make them subject to MITM kinds of attacks.

2.3 CertLedger: A new PKI model with Certificate Transparency based on Blockchain [9]

The authors in [9] propose a new PKI architecture where all the TLS certificates are validated and stored in the blockchain. The entire certificate revocation process and trusted certificate management are also conducted in the blockchain. TLS clients are light nodes of the blockchain network and store block headers to make a successful TLS handshake. However, becoming a peer of the blockchain network brings overhead in terms of storage and network communication for many of the TLS clients.

2.4 A Blockchain-based PKI Management Framework [24]

Recently, in [24], Yakubov et al. proposed a new blockchain based PKI architecture for issuing, revoking, and validating X.509 certificates. In this architecture, certificate lifecycle is managed through smart contracts[4] on the blockchain. After creating a certificate, CA adds its hash value to an issuance list in its smart contract. Similarly, to revoke a certificate, CA adds its hash value to a revocation list in the smart contract. More concretely, a CA smart contract stores an array for all its issued certificates' hash values, a map for the revoked certificates which are referenced by the certificates' hash values, and the CA certificate itself. Clients can validate certificates through either sending requests to web services or triggering certificate validation smart contract. The CA smart contract is created in such a way that its methods can only be triggered by its owner CA.

3 Security and Privacy Analysis of the Yakubov et al.'s Scheme

In this section, we first briefly describe the Yakubov et al.'s Scheme and then point out its security and privacy issues.

3.1 High-Level Description of the Yakubov et al.'s Scheme

In this scheme, each CA has a dedicated smart contract for issuing and revoking certificates in the blockchain. More concretely, a CA smart contract contains the following features:

- $certList := \{< index_i, certHash_i, hashAlg_i, date_i >: 1 \leq i \leq \alpha\}$ where $index_i$ is the auto generated index for the i-th certificate, $certHash_i$ is the hash value of the i-th issued certificate using $hashAlg_i$, $date_i$ is the addition date of the certificate to the blockchain, and α is the number of certificates.
- $revocationMap := \{< index_i, revokeDate_i >: 1 \leq i \leq \alpha\}$ where $index_i$ is the index of i-th certificate in the $certList$, $revokeDate_i$ is the revocation date of the i-th certificate, and α is the number of certificates.
- Its CA certificate $Cert_{CA}$.

[4] A smart contract [19] is a self enforcing digital application which contains data and an immutable code to manage it. It can be triggered through transactions in the blockchain.

Once a certificate is issued, the CA adds the hash value of the certificate to $certList$. Similarly, to revoke a certificate, CA adds the index of the *certList* and the revocation date to the $revocationMap$. The certificates in this proposal basically comprise several custom X.509 extensions such as *CA key identifier* and *Issuer CA identifier*:

- *CA key identifier* is populated with the CA smart contract address in the CA certificates.
- *Issuer CA identifier* stores the smart contract address of the issuer of a certificate. This extension is populated for all the certificates apart from the root CA certificates. In fact, it is used for building a trusted path and finding the smart contract address of the issuer of a certificate during the certificate validation.

Certificate validation can be performed in two different methods.

- In the first scheme, the certificate validation algorithm is implemented in a smart contract. This smart contract triggers all the CA smart contracts in the trusted path of a certificate. It validates the existence and the revocation status of all the certificates within this path. This scheme can only be triggered through a full node of the blockchain.
- In the second scheme, certificate validation is delegated to a web service. The web service queries the revocation status of all certificates in the trust chain one by one from a full node.

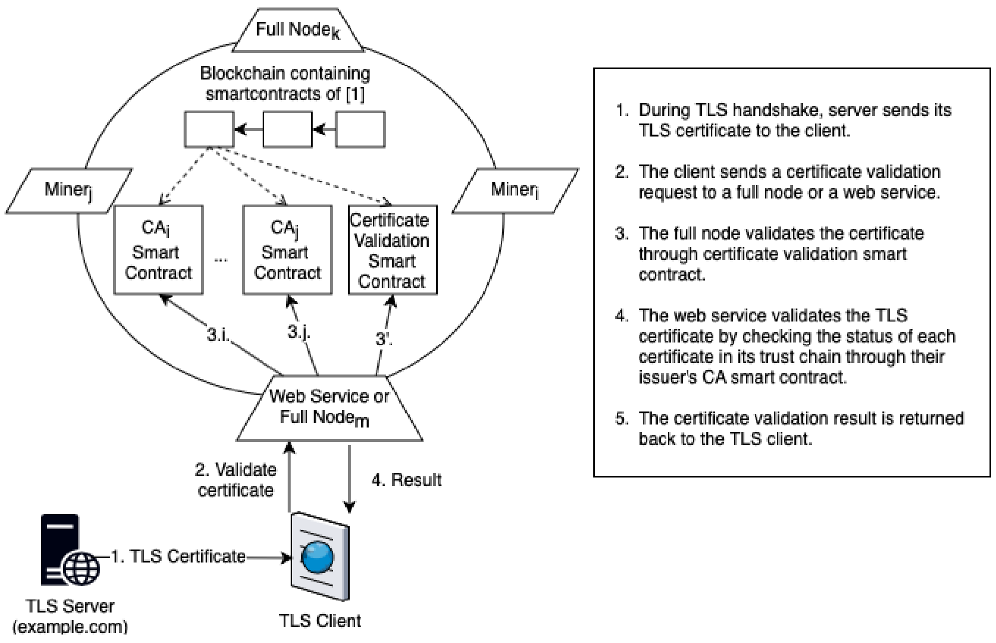
According to the experimental results, the performance of the second certificate validation scheme has a higher performance than the first one for the trust chains up to 400 sub-CAs. We depict the TLS system architecture with these certificate validation schemes in Figure 1.

3.2 Fake but valid certificates cannot be identified

The transparency of the certificates is not fully provided in Yakubov et al.'s scheme since only $certHash_i$s are stored in the CA smart contracts. Since it is infeasible to derive the subject of a certificate from $certList$, it would not be possible for a domain owner to identify and revoke a fraudulent certificate. Consequently, during a TLS handshake, clients could accept fake but valid certificates allowing MITM kinds of attacks.

3.3 Certificates may not be revoked in case of corrupted CAs

CAs may have to revoke their issued certificates for several reasons such as key compromise or information change (e.g., DNS name). However, if a CA is



1. During TLS handshake, server sends its TLS certificate to the client.
2. The client sends a certificate validation request to a full node or a web service.
3. The full node validates the certificate through certificate validation smart contract.
4. The web service validates the TLS certificate by checking the status of each certificate in its trust chain through their issuer's CA smart contract.
5. The certificate validation result is returned back to the TLS client.

Fig. 1: The TLS System of Yakubov et al.'s Scheme

compromised or does not have a proper revocation process, the certificate may not be revoked in a reasonable time frame or may not be revoked at all. This CA dependent process makes the TLS clients vulnerable to MITM kinds of attacks between the compromise and the revocation time of the certificate.

3.4 Certificate validation services can be compromised

The proposed certificate validation schemes (described in Section 3.1) are subject to MITM kinds of attacks. TLS clients have to rely on either the full nodes or the web service, and trust their certificate validation responses. Because these responses do not contain any cryptographic proofs of correctness. Hence, clients can easily be deceived if the full nodes[5] or web services are corrupted.

Building the trust chain of a certificate is one of the critical components of certificate validation. This process is not fully clarified in the proposal which may be a cause of MITM attacks as well. Namely, creation of CA smart contracts on the blockchain is not subject to authorization, therefore an adversary

[5] Full nodes do not execute a transaction which changes the state of the blockchain while running certificate validation smart contract, but only queries blockchain data. Therefore, its malicious behaviour does not have any impact on blockchain.

can deploy a smart contract for a malicious CA which stores fraudulent certificates issued by this fake CA. In this case, TLS clients could be subject to MITM kinds of attacks since they are going to validate these fake certificates through the proposed certificate validation schemes.

3.5 A privacy issue while certificate validation

As said before, TLS clients cannot validate the certificates themselves (trivially, if they are not a full node of the blockchain). In that case, they have to query a full node or a web service for this purpose. However, this process is not also privacy preserving since these intermediary entities can track the web addresses visited by the TLS clients.

4 Our Proposal: KORGAN

4.1 Our Motivation: Why Dynamic Threshold Signatures based Permissioned Blockchains?

In order to eliminate the security and privacy issues mentioned in Section 3, TLS clients should be able to use a publicly available blockchain which would include all issued certificates as well as their status without relying any external parties during a certificate validation process. However, this introduces an extra overhead for both permissionless and permissioned blockchains since they have to first verify the validity of the blocks (or only the headers) [23]. More concretely,

- In case of permissionless blockchains, there is going to be a significant network overhead for TLS clients because they need to be a peer of the blockchain network to determine the valid blocks due to the underlying consensus mechanism [8, 14, 22].
- In case of permissioned blockchains, they have to query a certain number of consensus nodes (i.e., $2N+1$ in PBFT [3] which requires $3N+1$ replicas to tolerate N Byzantine failures). This requirement would also become a burden with the increasing number of consensus nodes.

Therefore, becoming a peer of the blockchain network or querying consensus nodes would going to be infeasible for many of the TLS clients due to limited storage capacity, processing power, or low bandwith. In order to eliminate this overhead, we require authentic blocks which would enable TLS clients to verify their validity efficiently. However, permissionless blockchains are unfortunately not suitable for generating signed blocks because any peer could join the blockchain network and could generate a new block which makes determination of a signature (private) key and distribution of the verification (public) key to the TLS clients infeasible. On the other hand, permissioned blockchains would be more suitable for generating signed blocks since only a limited number of consensus nodes are authorized to generate the new blocks. Still the following requirements must be satisfied for an efficient and scalable solution:

1. Management of the verification key should not be a burden for the TLS clients. Namely, after they receive an authentic verification key they should not be able to change it frequently.
2. There must be only one signature on a block to be optimally scalable.
3. The verification key should not be also changed with the varying number of consensus nodes (i.e., in case of adding new nodes or removing the existing ones). Otherwise, all TLS clients must subsequently update the verification key which would also make the system practically infeasible.

To tackle these requirements, we propose to use dynamic threshold signature schemes among the consensus nodes for signing the new blocks [17]. As in a typical threshold signature scheme, there is going to be only one public key (for verifying a signature) of the overall system, and the private key shares will be owned and managed by the corresponding consensus nodes. If at least a threshold number of consensus nodes agree on a block, then they are going to sign the new block with their private key shares to generate a valid signature.

We highlight that the underlying consensus of Facebook Libra also utilizes threshold signatures [1, 2, 25], however, their solution does not propose dynamic versions of threshold schemes which would incurs significant overhead to our scheme.

This is because it does not the meet the above-mentioned third requirement, and a change in the underlying consensus nodes would result in an update in the clients' public (verification) keys. Thanks to the authors of [17], we have efficient threshold schemes which indeed meet all the requirements and ensure that the remaining consensus nodes are able to add new consensus nodes or remove the corrupted nodes efficiently through only updating their secret shares without changing the overall verification key.

4.2 Our Approach: A Modified PBFT Consensus Mechanism Through Dynamic Threshold Signatures

The seminal Practical Byzantine Fault Tolerance (PBFT) algorithm aims to reach consensus through Byzantine nodes that tolerates Byzantine failures with low overhead [3]. In particular, PBFT basically uses state-machine replication and replica voting for changing the state in the network. All nodes acting as validators[6] have equal votes, and validation is executed through multiple rounds to reach the consensus. PBFT utilizes digital signatures to ensure the authenticity of the messages. Nodes have to verify all the signatures received from their peers during each phase of the consensus rounds.

In our blockchain architecture, we utilize a dynamic threshold signature scheme on the PBFT consensus mechanism, and a valid block can only be generated if at least t-out-of-ℓ consensus nodes sign the new block [17]. A key generation setup for threshold signature scheme is going to be executed among predefined consensus nodes as follows:

Threshold Key Generation Setup Among Consensus Nodes We assume that an existing PBFT blockchain with $3N+1$ consensus nodes has been already setup [3]. Namely, the key generation ceremony will be completed using the underlying PBFT consensus mechanism. More concretely,

1. Each i-th consensus node randomly chooses its secret key share SK_i and executes the threshold signature steps (like in [17]), and publishes their intermediate outputs on the blockchain (as a transaction). In particular, the consensus nodes use the underlying blockchain as a public bulletin board to publish and retrieve the necessary data to execute the key generation setup properly.
2. Each consensus node queries the blockchain until all the consensus nodes publish their partial signatures. A key generation setup would be completed only after all the pre-defined consensus nodes participate to the ceremony[7].
3. Once all the consensus nodes publish their partial signatures to the blockchain, the selected leader constructs the overall public key and adds as a new transaction. More concretely, once the signature steps are validated through the consensus mechanism, $(PK, (SK_1,\cdots,SK_\ell))$ become the verification key and signing key share of the i-th consensus node, respectively.
4. Once the key generation ceremony is completed succesfully, the consensus nodes (i.e., the signers) will only generate and accept signed blocks.

Block Generation through PBFT with Dynamic Threshold Signatures Now, we have now a dynamic threshold signature variant of PBFT on our new modified chain. As in PBFT, the consensus mechanism would run in rounds with one

[6] Trivially, as in any permissioned blockchain, we require the validators to be selected from political and geographical disparate entities.

[7] If they do not take part in the ceremony, they will not be able to send signed messages during the consensus phase.

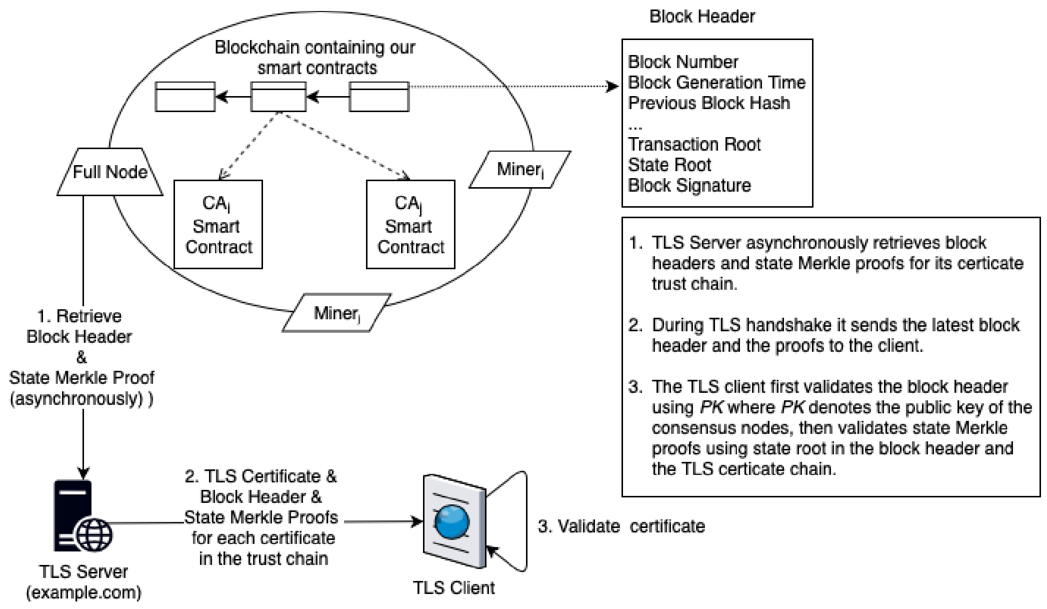

Blockchain containing our smart contracts
Block Header
Block Number
Block Generation Time
Previous Block Hash
...
Transaction Root
State Root
Block Signature
Full Node
Miner$_i$
Miner$_j$
CA$_i$ Smart Contract
CA$_j$ Smart Contract
1. Retrieve Block Header & State Merkle Proof (asynchronously))
2. TLS Certificate & Block Header & State Merkle Proofs for each certificate in the trust chain
3. Validate certificate
TLS Server (example.com)
TLS Client
1. TLS Server asynchronously retrieves block headers and state Merkle proofs for its certicate trust chain.
2. During TLS handshake it sends the latest block header and the proofs to the client.
3. The TLS client first validates the block header using PK where PK denotes the public key of the consensus nodes, then validates state Merkle proofs using state root in the block header and the TLS certicate chain.

Fig. 2: The TLS System of KORGAN

node acting as a leader and others as validators. In order to sign a new block in our new and modified PBFT, the consensus nodes (i.e., the signers) are going to execute the following threshold signature generation ceremony to confirm a block.

- **Pre-prepare phase:** As in PBFT, the goal of this phase is to ensure that a majority of honest nodes has agreed on a sequence number for a leader's request. In our flow, the leader initiates the consensus process by sending its partially signed PRE-PREPARE message (using $Sign_{SK_i}$) which is a block proposal containing a certain number of transactions.
- **Prepare phase:** Upon receiving the PRE-PREPARE message, every node in the consensus group checks the correctness and validity of the block and multicasts its partially signed PREPARE message (using $Sign_{SK_i}$) to all the other nodes.
- **Commit phase:** Based on the received PREPARE messages, each node combines t signatures, computes a valid signed message, and then multicasts a signed COMMIT message (i.e., "YES/NO") to the consensus group. The new block is committed to the blockchain only if a valid signature generated.

At the end of the commit phase, all honest nodes in the consensus group would have the same view regarding to the state of blockchain by either accepting or rejecting the block proposal, thereby achieving the confirmed transaction. Consequently, the authenticity of a block (or a block header) can easily be verified by any TLS cilent using the verification key PK. Since the underlying consensus mechanism relies on PBFT, up to N nodes could suffer from Byzantine failure. Therefore, $t = N+1$ out of $3N+1$ nodes would be sufficient to preserve the same security level of the underlying consensus mechanism.

Note that the feasibility of our consensus mechanism does not change if the validator set is dynamic because it only relies on an arbitrary set of signers participating in the signature process. Moreover, a correct leader, once selected, sends messages to only $O(n)$ validators to reach the consensus as in [25].

4.3 The architecture of KORGAN

In our PKI model, we use a permissioned blockchain with a modified PBFT consensus mechanism which enables TLS clients to easily verify the final state of a certificate.

To achieve this, consensus nodes maintain a State Merkle Tree (SMT)[8] which is used to store and verify the state of all accounts, smart contract codes and the data within the smart contracts. SMT is updated with each block according to the block transactions and its Merkle root is put into the generated block header[9]. Since we store all the issued certificates and their revocation status in the smart contracts, their state can also be tracked in every block. Moreover, their state can be verified using the Merkle proof generated from the SMT and the Merkle root. Since Merkle root is stored in the signed block header and its authenticity can be verified by PK, TLS clients can easily check the validity of a certificate without relying any parties during TLS handshake.

In order to implement our PKI model, we modify the CA smart contract in Yakubov et al.'s scheme in such a way that, instead of storing an array with certificate hash values and a map for the revoked certificates, we store two maps in it. The first map is $certURIMap := \{(uri_i, < certHash_{i,j} >) : 1 \leq i \leq \alpha, 1 \leq j \leq \beta\}$ where uri_i is the i-th certificate subject (or subject alternative name), $certHash_{i,j}$ is the hash of the j-th certificate of the i-th uri, and $\alpha, \beta \in \mathbb{N}^+$. This map provides the transparency of the certificates, and the domain owners can

8 Modified Merkle Patricia Tree [5] can be used for this purpose where search, insert and update operations can be performed in logarithmic time.

9 A sample block header of KORGAN's architecture is depicted in Figure 2.

use it to monitor[10] the blockchain if a fraudulent certificate is issued for their web servers.

The second map is used to track the revocation status of the certificates and represented as $certRevocationMap := \{< certHash_k, rs_k >: 1 \leq k \leq \gamma\}$ where $certHash_k$ is the k-th certificate hash, rs_k is the revocation status (“revoked", “valid") of the k-th certificate, and $\gamma \in \mathbb{N}^+$.

In our CA smart contract, a certificate can only be added to the blockchain by its issuing CA. However, the status of the certificate can be changed as “revoked" by both its issuing CA and owner. A certificate owner can only trigger the revocation method of the smart contract if he can prove his ownership to the certificate. Therefore, this method requires a signature generated by the private key of the certificate.

Note that block confirmation time must be short enough to discourage adversaries to perform a MITM kind of attack during block time. In this respect, due to the underlying PBFT mechanism and dynamic threshold signature scheme, our consensus scheme provides high throughput and low transaction latency similar to LibraBFT [12] which meets our requirements.

Algorithm 1 Verify Header

```
▷header denotes the block header of the block which will be used to validate the state of the
certificates, PK denotes the public key of the blockchain, latestAcceptableTime denotes the
latest generation date of the header acceptible by the TLS client
function VERIFYHEADER(header, PK, latestAcceptableTime)
    ▷verify that the block header is genuine
    if verifyBCHeaderSgn(header.signature, PK) = false then
        return false
    end if
    ▷verify that the block header is not too old
    if ($header.timestamp \geq latestAcceptableTime$ AND
      $header.timestamp < t_{now}) = false$ then
        return false
    end if
    ▷otherwise return true
    return true
end function
```

[10] Event listeners can be used for this purpose which triggers certain events (e.g., SMS, e-mail etc.) in case a certain condition is satisfied in the smart contract.

Algorithm 2 Validate Certificate Chain

```
▷$certChain$ denotes the CA certificates in TLS certificate trust chain, $proofForCertStatusList$ denotes the list of Merkle proofs for the status of each certificate in the trust chain except root CA certificate
function ValidateCertificateChain($certChain$, $proofList$, $trustedCAList$, $header$)
    ▷verify that the root CA in the certificate chain is in the trusted CA address list
    if $certChain[certChain.length].caKeyIdentifier \notin addrTrustedCAList$ then
        return $false$
    end if
    ▷verify that the certificates in the TLS chain (except root CA certificate) are not revoked. note that if the certificates doesn't exist in the smart contract merkle proofs can not be verified
    for $i \leftarrow 1$ to $certChain.length - 1$ do
        $cert \leftarrow certChain[i]$
        $certCA \leftarrow certChain[i+1]$
        if ValidateCertificate($cert$, $certCA$, $proofList[i]$, $header.stateRoot$) = false then
            return $false$
        end if
    end for
    ▷otherwise return true
    return $true$
end function
```

4.4 Certificate Validation of KORGAN in SSL/TLS

TLS clients are not required to be full or light nodes of the blockchain, thus do not have to retrieve any blocks (or headers) from the blockchain network. For certificate validation, they only need to store *PK* to verify the authenticity of the block headers and the trusted root CAs' smart contract addresses to construct the trust chain. On the other hand, TLS servers have to periodically retrieve the block headers and the Merkle proofs associated with their TLS certificate chain from a full node of the blockchain. The retrieval process is independent of the TLS handshake and can be conducted asynchronously.

KORGAN does not change the TLS handshake protocol but introduces new TLS extensions to be used during the *ServerCertificate* step of the protocol (see Figure 2). In these extensions, a TLS server sends the latest block header, certificate chain, and a list of Merkle proofs for the revocation status of each certificate in the chain to the TLS client. A TLS client performs the following steps to validate the TLS certificate.

1. Verifies the signature of the block header by *PK*, and reads the authentic block generation time and the SMT root from the block header.
2. Checks whether the block generation time is fresh enough according to its security settings. However the acceptable freshness period should not be shorter than the block time and it should not reject the latest block header.

Algorithm 3 Validate Certificate

```
▷$cert$ denotes the certificate to be validated, $certCA$ denotes the certificate of cert's issuer, $merkleProof$ is the merkle proof generated from SMT for cert, $smtRoot$ denotes the root hash value of SMT
function VALIDATECERTIFICATE($cert, certCA, merkleProof, smtRoot$)
    $caScAddr \leftarrow cert.issuerCAIdentifier$
    $certHash \leftarrow Hash(cert)$
    ▷check whether certCA is the the issuer of cert
    if $caSCAddr \neq certCA.caKeyIdentifier$ then
        return $false$
    end if
    ▷verify that certificate is valid using the state proof generated for the certificate
    if verifyStateMerkleProof ($merkleProof, smtRoot,$
      $certHash, caScAddr) \neq$ “valid" then
        return $false$
    end if
    ▷otherwise return true
    return $true$
end function
```

3. Checks whether the TLS certificate is issued from a trusted root CA by searching the smart contract address (*CA key identifier*) of the root CA in its trusted list.
4. Validates each certificate in the trust chain by validating the Merkle proofs using the certificate's hash value, its issuing CA's smart contract address (*Issuer CA identifier*) and the SMT root.
5. Checks the revocation status of each certificate in the trust chain (except root CA certificate) and verifies that none of them are revoked.

Table 1: TLS Handshake Experimental Results

Number of TLS Certificates in the CA Smart Contract	Header Data Size (bytes)	Account Proof Data Size (bytes)	Storage Proof Data Size (bytes)	TLS Handshake Overhead Total (bytes)
1	535	758	590	1883
100	535	758	1089	2382
1.000	535	758	1557	2850
10.000	535	758	2027	3320

The first and second step of our certificate validation algorithm is described in more detail in Algorithm 1, third step in Algorithm 2, and finally fourth and fifth steps in Algorithm 2 and 3. We would like to highlight that our algorithm does not require any further network connections, thus the privacy of the TLS clients is also fully preserved. The execution time of the algorithm is not effected from the network latency and only depends on the processing capability of the TLS clients.

5 Implementation and Experimental Results

We have implemented KORGAN by updating the CA smart contract of Yakubov et al.'s scheme[11] [24] and experimented certificate validation with our scheme. For the experiments, we deployed two smart contracts on private Ethereum network so that we had two CAs in the trust chain. For generation and verification of the state Merkle proofs, we used Eth-proof node-js API [27]. We executed our experiments on a Macbook Pro with Intel Core i7 (3.1 Ghz) CPU, 16 GB of memory and macOS Majove OS.

We demonstrate the experimental results for TLS handshake overhead in Table 1 and elaborate them as follows. First, $Header$ denotes the size of the block header in Ethereum, therefore its size is constant and independent of certificates in the smart contract. Second, $AccountProof$ is the proof generated to validate the overall state of the smart contract (i.e., the account) comprising its balance, code, and the stored data. Note that its size grows logarithmically with the number of smart contracts in the blockchain due to its Patricia Tree structure [5]. In our first experiment, there was only one CA smart contract, therefore, the size of the $AccountProof$ size is the same independently of number of TLS certificates. Third, $StorageProof$ is also generated from Storage Merkle-Patricia Tree which is different for all smart contracts in Ethereum. The root value of this tree is also used while computing the state of the account. The size of the $StorageProof$ is $log_2\ n \times c_1 + c_2$ where n is the number certificates in an account, c_1 is a constant calculated by adding hash length forming the Merkle proof with the path length between the nodes, and c_2 is the size of input where its hash is calculated to generate a leaf node in the Merkle tree. Hence, the overall TLS Handshake overhead in our scheme is calculated as

$$|Header| + (|AccountProof| + |StorageProof|) \times n$$

where n is the number of CAs in the TLS certificate chain.

We note that certificate validation network overhead is not given in Yakubov et al.'s scheme. On the other hand in the conventional PKI, the size of a CRL changes with respect to the number of certificates issued by the CA. Even though there are CRLs ranging up to 28 MB[12], the CRL size for the median certificate is

[11] https://github.com/snt-sedan/pki-blockchain
[12] Apple hosts 28MB of CRL at http://crl.apple.com/wwdrca.crl

Table 2: Certificate Validation Durations I

Number of TLS Certificates in the CA Smart Contract	Certificate Validation Duration (ms)
1	55,56
100	56,37
1.000	59,24
10.000	60,45

calculated as 51 KB in [13]. In case of OCSP usage for revocation checking, the average size for an OCSP response is about ~4KB. Moreover, the total network overhead increases with respect to the length of the trust chain.

Our experimental results in Table 2 demonstrate that the number of certificates in a CA smart contract does not significantly effect the certificate validation duration. On the other hand, in conventional PKI, if the revocation check of a certificate is performed through OCSP, then the latency only due to the network traffic is around 200 ms [16]. The total duration increases with respect to the number of certificates in the trust chain. Since the size of CRLs are much bigger than OCSP responses, their average downloading latency is also greater then OCSP [15].

Table 3: Certificate Validation Durations II

Trustchain Length (number of CA smart contracts*)	Certificate Validation Duration (ms)
1	59,24
2	59,85
3	60,13
5	60,67

* There are 1000 certificates in each smart contract.

6 Conclusion and Future Work

There have been recent serious security incidents due to misbehaving CAs which have issued fraudulent certificates. To make CAs more transparent, various public log based and blockchain based PKI models are proposed. In this paper, we point out the security and privacy issues of one of the most recent proposals (belonging to Yakubov et al.) and eliminate them by proposing a new PKI architecture, what we called KORGAN. KORGAN is based on permissioned blockchain with a modified PBFT where the blocks are signed through dynamic threshold signature scheme among consensus nodes. Due to the signed blocks, TLS clients can now easily verify the final states of certificates without requiring to be a peer of the blockchain network. Our experimental results on Ethereum demonstrate that KORGAN does not bring any significant computational and network overhead during certificate validation. Even more, the duration of our certificate validation is less than the previous schemes.

Further research work mainly includes modifying KORGAN in such a way that generating CA smart contracts could be restricted to only trustworthy CAs. For this purpose, an international board can be established to audit the CAs and sign the smart contract generation transaction using a threshold signature scheme as well.

Bibliography

[1] Boneh D, Lynn B, Shacham H (2001) Short signatures from the weil pairing. In: Advances in Cryptology — ASIACRYPT 2001, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 514–532

[2] Cachin C, Kursawe K, Shoup V (2005) Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. Journal of Cryptology 18(3):219–246, DOI 10.1007/s00145-005-0318-0

[3] Castro M, Liskov B (2002) Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems 20(4):398–461, DOI 10.1145/571637.571640

[4] Chen J, Yao S, Yuan Q, He K, Ji S, Du R (2018) Certchain: Public and efficient certificate audit based on blockchain for tls connections. In: IEEE INFOCOM 2018-IEEE Conference on Computer Communications, IEEE, pp 2060–2068

[5] Ethereum (2019) Patricia Tree. https://github.com/ethereum/wiki/wiki/Patricia-Tree (visited 2019-09-28)

[6] Hyun-Jin Kim T, Huang LS, Perrig A, Jackson C, Gligor V (2013) Accountable key infrastructure (AKI): A proposal for a public-key validation infrastructure. In: Proceedings of the 22nd international conference on World Wide Web, ACM, pp 679–690

[7] International Telecommunication Union (ITU-T) (2012) X.509: Information technology–open systems interconnection–the directory: Public-key and attribute certificate frameworks. Standard

[8] Kiayias A, Russell A, David B, Oliynykov R (2017) Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Advances in Cryptology – CRYPTO 2017, Springer International Publishing, pp 357–388

[9] Kubilay MY, Kiraz MS, Mantar HA (2019) CertLedger: A New PKI Model with Certificate Transparency Based on Blockchain. Computer & Security 85:333–352, DOI 10.1016/j.cose.2019.05.013

[10] Langley A (2015) Maintaining digital certificate security. https://security.googleblog.com/2015/03/maintaining-digital-certificate-security.html (visited 2019-09-28)

[11] Laurie B, Langley A, Kasper E (2014) Certificate transparency. ACM Queue 12(8):10–19

[12] Libra (2019) LibraBFT Consensus Performance. https://developers.libra.org/docs/crates/consensus (visited 2019-09-28)

[13] Liu Y, Tome W, Zhang L, Choffnes D, Levin D, Maggs B, Mislove A, Schulman A, Wilson C (2015) An end-to-end measurement of certificate revocation in the web's PKI. In: Proceedings of the 2015 Internet Measurement Conference, ACM, pp 183–196

[14] Nakamoto S (2008) Bitcoin: A peer-to-peer electronic cash system

[15] Netcraft (2019) NetCraft. CRL Sites in September 2019. https://uptime.netcraft.com/up/reports/performance/CRL (visited 2019-09-28)

[16] Netcraft (2019) NetCraft. OCSP Server Performance in September 2019. https://uptime.netcraft.com/up/reports/performance/OCSP (visited 2019-09-28)

[17] Noack A, Spitz S (2009) Dynamic threshold cryptosystem without group manager. Network Protocols & Algorithms pp 108–121, DOI 10.5296/npa.v1i1.161, URL https://doi.org/10.5296/npa.v1i1.161

[18] Rescorla E (2018) The Transport Layer Security (TLS) Protocol Version 1.3. Tech. rep., RFC 8446

[19] Szabo N (1997) Formalizing and securing relationships on public networks. First Monday 2(9)

[20] Wang Z, Lin J, Cai Q, Wang Q, Jing J, Zha D (2018) Blockchain-based certificate transparency and revocation transparency. In: Financial Cryptography and Data Security, Springer International Publishing

[21] Wikipedia contributors (2011) DigiNotar. https://en.wikipedia.org/wiki/DigiNotar (visited 2019-09-28)

[22] Wood G, et al. (2014) Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper 151(2014):1–32

[23] Wüst K, Gervais A (2018) Do you need a blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), IEEE, pp 45–54

[24] Yakubov A, Shbair WM, Wallbom A, Sanda D, State R (2018) A blockchain-based pki management framework. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium, pp 1–6

[25] Yin M, Malkhi D, Reiter MK, Gueta GG, Abraham I (2019) Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, ACM, PODC '19, pp 347–356, DOI 10.1145/3293611.3331591

[26] Yu J, Cheval V, Ryan M (2016) DTKI: A new formalized PKI with verifiable trusted parties. The Computer Journal 59(11):1695–1713

[27] Zac M (2019) Eth Proof 2.0.0. https://github.com/zmitton/eth-proof (visited 2019-09-28)