ORIGINAL RESEARCH
published: 09 December 2021
doi: 10.3389/fbloc.2021.739431

Efficient Onboarding and Management of Members in Permissioned Blockchain Networks Utilizing TLS Certificates

Ulrich Gallersdörfer, Jan-Niklas Strugala and Florian Matthes*

Chair for Software Engineering for Business Information Systems (sebis), Faculty of Informatics, Technical University of Munich, Munich, Germany

Consortia blockchain networks face the issue of expanding their systems to new members. Onboarding processes are often cumbersome, as they require identifying the new participant, manually setting up rights, exchanging key material, and adding information about the new member to the consensus smart contract. Besides that, these processes are time-consuming and scale poorly. Identifying the members might be faulty as the pre-existing members might be deceived by malicious parties claiming to be someone else. This paper proposes a novel methodology to allow the onboarding of new parties without time-intensive off-chain processes. We establish identities of new consortia members by utilizing TLS certificates bound to publicly known domain names. With this identity scheme in place, the network operators can define rules such as only specific parties are allowed to join the network, e.g., only owners of *.edu domains. This methodology scales well, provides for extensive ruling and monitoring, and helps consortia blockchains to grow faster.

Keywords: smart contract, permissioned blockchain, authentication, authorization, TLS, PKI

OPEN ACCESS

Edited by:
Dimitris Kogias,
University of West Attica, Greece

Reviewed by:
Michael Xevgenis,
University of West Attica, Greece
David López Flores,
National Autonomous University of Mexico, Mexico

****Correspondence:***
Ulrich Gallersdörfer
ulrich.gallersdoerfer@tum.de

Specialty section:
This article was submitted to Blockchain Technologies, a section of the journal Frontiers in Blockchain

Received: *10 July 2021*
Accepted: *09 November 2021*
Published: *09 December 2021*

Citation:
Gallersdörfer U, Strugala J-N and Matthes F (2021) Efficient Onboarding and Management of Members in Permissioned Blockchain Networks Utilizing TLS Certificates. Front. Blockchain 4:739431. doi: 10.3389/fbloc.2021.739431

1 INTRODUCTION

Private or public permissioned blockchain networks are a compelling alternative to public permissionless blockchains such as Ethereum or Bitcoin for enterprises forming a consortium. Not only the issues of energy consumption and the respective carbon footprint Stoll et al. (2019); Gallersdörfer et al. (2020) or scalability Xin et al. (2017) are resolved, but also the entities setting up the network remain in full control over the circulating supply of cryptocurrency and who is able to join and participate in the network Androulaki et al. (2018). Privacy and access control are often required, as the members of the consortium store private information or handle otherwise proprietary data within the network.

As a critical difference to permissionless networks, not everyone can join and participate in the network at any time. The access to the network, either on an application-level (e.g., transactions and execution of smart contracts) or on a "mining" level[1] (e.g., proposing new blocks) is strictly limited and defined by the actors that set up the network.

[1]In permissioned networks, we refer to the role of miners as validators

In particular, the role of the validators is of interest. They can create new blocks, include transactions in the network, and secure the integrity of the network. However, the management of the set of validators has two downsides:

1) It is time-consuming, as every participant has to be onboarded to the network manually by communicating off-chain, e.g., exchanging information and addresses. Often, a party within the network is responsible for collecting information and ensuring the proper onboarding of the new participant.
2) There is a form of centralization, as 1) often a single entity is responsible for adding new validators to the network or 2) if amongst the existing validators voting is conducted, voters would need to verify who they are actually voting on (e.g., as there is no link between an address in a network and a real-world entity).

This paper proposes a solution to ease the management of validators in permissioned blockchain networks and address the previously mentioned issues. We do so by proposing the usage of TLS certificates, allowing the identification of a counterparty by a Fully Qualified Domain Name (FQDN), e.g., *www.example.org*. Thereby we allow adding entities to the validator set either by directly specifying the FQDN of the new validator or defining rules an FQDN needs to adhere to, e.g., the Top Level Domain (TLD) needs to be . *edu*. This novel authentication mechanism can also be used for FQDN-based voting, so voters are certain which new entity applies to become a validator.

Our work builds upon previously published research in Gallersdörfer et al. (2021); Gallersdörfer and Matthes (2021). Gallersdörfer and Matthes (2021) proposes the usage of TLS-certificates in the context of off-chain verification (e.g., in a wallet such as Metamask) to prevent address replacement attacks. These attacks aim at tricking end-users into sending funds such as ICO investments to wrong addresses, resulting in the loss of the respective funds. Gallersdörfer et al. (2021) proposes the mirroring of parts of the Public Key Infrastructure (PKI) on-chain that lies behind the TLS certificate structure. This allows for the usage of the certificates and signatures in an on-chain context. In this manuscript, we extend the second approach to be usable as a form of active authentication in the context of consortia membership management.

The paper is structured as follows: **Section 2** introduces concepts of the World Wide Web, outlines the previous approaches, and explains their limitations. In **section 3** we introduce the active usage of TLS-certificates in an on-chain context and extend the application to consortia membership management. In **section 4** we conclude the paper.

2 BACKGROUND

Several systems are the backbone of the World Wide Web, and we also rely on these technologies in this paper. First, we introduce components of the Web-PKI, namely PKIs in general, DNS, and TLS. Then, we discuss permissioned blockchain networks and how they are managed. Afterward, we present previous work done to utilize TLS-certificates and FQDNs in the context of the blockchain.

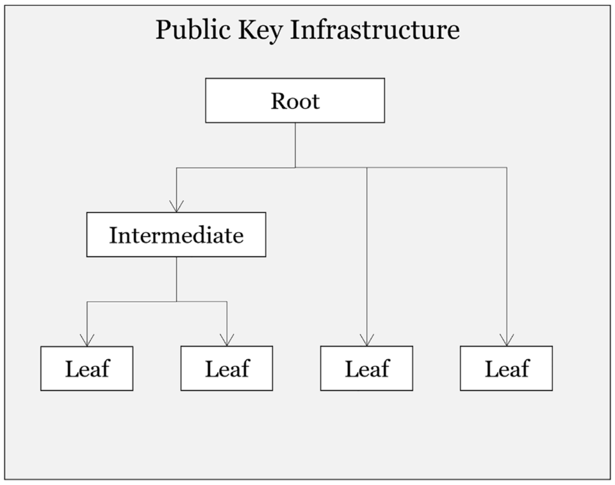

FIGURE 1 | Hierarchical public key infrastructure.

2.1 Web-PKI

On today's internet, there are a set of technologies and public key infrastructures in place that can be summarized as *Web PKI*. These include, amongst other, the Domain Name System, Transport Layer Security, Certificate Authorities, DNS Security Extensions, and Certificate Transparency. In particular, we cover regular PKIs, DNS, and TLS.

2.1.1 PKI

Public Key Infrastructures are systems in place to allow entities the management of key material for trusted entities, such they can sign, verify and communicate with other parties in a safe way Weise (2001). Integrity, privacy, and authenticity are of the highest importance for these systems.

Often, these PKIs are organized in a hierarchical order. There are several trust anchors, often called root certificates. All entities in the network know the certificates and trust the authorities managing these root certificates to behave honestly. These, in turn, issue new certificates for other entities in the network containing a unique identifier (e.g., in TLS, FQDNs are used).

The PKI presents itself as a tree-like structure. There are few roots but many leaves. First, this allows for broad usage. Millions of certificates can be managed with such a structure. Second, it also enables an efficient verification mechanism: For an entity to prove that his certificate is signed by one of the root certificates, it only has to prove the signing path to the root and the second entity, only aware of the root certificate, can verify the claims made by the first party. This is the basis for today's Internet. In **Figure 1**, we depict a form of hierarchical PKI.

2.1.2 DNS

The domain name system (DNS) is a naming service that allows resolving human-readable names to IP addresses. It was first described in rfc (1987a) and in rfc (1987b). These names follow a definition, containing out of Top-level domains (e.g., *com, edu, org*), second-level domains (e.g., *example, wikipedia*), and additional levels that are used to further divide the respective name. These domains are separated by a dot (.) and together form a unique identifier, the Fully Qualified Domain Name (FQDN). The second-level domains are managed by selected registrars, which sell the domains to interested buyers. They are able to use the domain in the context they desire, e.g., for website hosting or e-mail address usage. As this naming scheme is broadly established in the world and subject to daily usage for billions of people, we also rely on it for the purposes of this paper.

2.1.3 TLS

Transport-Layer-Security (TLS) was first introduced in Allen and Dierks (1999) and continually expanded in Dierks and Rescorla (2006), Rescorla and Dierks (2008) and Rescorla (2018) as a protocol to secure the communication between two parties via the World Wide Web. The identification of the counterparty relies on X.509 certificates (described in Housley et al. (1999)) which contains key material and a human-readable name established in the DNS to allow end users to easily recognize the party they are interacting with. As this system and the cryptography behind it is widely in use and no further bootstrapping is required, it provides a solid basis for integrating its signature schemes within a blockchain system.

2.2 Managing Permissioned Blockchain Networks

Permissioned blockchains are often considered an alternative to permissionless blockchains. They require different consensus mechanisms for creating new valid blocks in the networks than permissionless blockchains. While these systems often rely on mechanisms such as Proof-of-Work (PoW) or Proof-of-Stake (PoS), permissioned networks often depend on some form of Proof-of-Authority (PoA) or some form of Byzantine-Fault-Tolerant-Scheme (BFT). One difference between these schemes is, that PoA or BFT algorithms require the validating entities to be known prior to the launch of the network. Later on, if the set of validators changes, entities can be on- or offboarded.

In this paper, our work builds upon the Ethereum blockchain. Any other type of permissioned blockchain with Smart Contract support can be used, as our scheme can be applied regardless. We use Ethereum due to its simplicity as well as its higher prominence in the space.

For Ethereum-based permissioned networks, rules, the consensus mechanism, and other settings are defined within the genesis block of the respective permissioned blockchain. The genesis is the first block in the network which does not link to prior blocks. To allow new entities to join the network, members of the consortium need to share, inter alia, the genesis block with new participants. While the genesis block and an account funded with the respective currency is sufficient to read from and create transactions within the network, it is not sufficient for being able to become a validator in the network. An exemplary genesis block is displayed in Listing 1.[2]

Listing 1. A genesis. json file. The initial signer is defined within the extradata field. Two accounts receive an initial balance of 300,000 and 400,000 respectively.

```
{
  "config": {
    "chainId": 15,
    "homesteadBlock": 0,
    ...
    "petersburgBlock": 0,
    "clique": {
      "period": 5,
      "epoch": 30000
    }
  },
  "difficulty": "1",
  "gasLimit": "8000000",
  "extradata": "0x000000000000000000000000000000000000000000000000000
     00000000000007df9a875a174b3bc565e6424a0050ebc1b2d1d82000000000000
     00000000000000000000000000000000000000000000000000000000000000000
     00000000000000000000000000000000000000000000000000000",
  "alloc": {
    "7df9a875a174b3bc565e6424a0050ebc1b2d1d82": { "balance": "300000"
       },
    "f41c74c9ae680c1aa78f42e5647a62f353b7bdde": { "balance": "400000"
       }
  }
}
```

There are two main ways to manage the list of the validators in the network.

- Managing the set of validators in the genesis block: The entity creating the network can define within the genesis block the set of validators. This is a straightforward and easy way to set validators, but it comes with the downside of reduced flexibility. As blocks in the network cannot be changed after the fact, adding new validators or removing old ones require a hard fork. This hard fork needs to be administered off-chain, e.g., the parties involved in the network need to communicate with each other and define a new block that overrides the rules set in the genesis block.
- Use a contract-based validator set: Instead of just stating a list of validators, one can also define a specific smart contract for the management of the validator set. The contract is created with the genesis block and usually receives an easy to recognize address such as 0x000 ... 005. Within that contract, the creator of the network also defines the first validator and is able to add or remove new validators at a later point in time. This approach does not require a hard fork and is more flexible, e.g., as it also allows for the automatic removal of inactive validators.

Due to the clear benefits of a contract-based validator set, most networks opt for it and do not rely on a fixed set of validators. Several example contracts are available[3], which are also used for the Ethereum Tesnet Kovan.

Nonetheless, the existence of such smart contracts does not eliminate problems of centralization and the need for an off-chain synchronisation between the entities in the network. If only one entity is responsible for maintaining the contract-based validator set, the network is centralized and the main entity is able to manipulate the network in ways it desires. Another option would

[2]The file is taken from https://geth.ethereum.org/docs/interface/private-network

[3]https://github.com/openethereum/kovan-validator-set/

be to vote on new validators, as it is done within the Bloxberg network, a public permissioned network aimed at scientific applications[4]. The problem with such an approach is that 1) either voting happens on a name-basis and the central entity is trusted to adhere to the voting and adds the related address to the set of validators or 2) the voters elect on a pure address, without any further information whom it belongs to. In any case, both approaches have downsides.

2.3 Usage of TLS-Certificates in Blockchain Networks

In this section, we discuss the usage of TLS-certificates in blockchain networks as outlined in Gallersdörfer et al. (2021); Gallersdörfer and Matthes (2021). First, we give an overview of the components and design space of using TLS certificates in the context of blockchain and then discuss how both approaches account for their different requirements.

2.3.1 Components of the System

In both systems, there are three two components: An endorsement and the verifier.

2.3.1.1 Endorsement

If we want to use TLS certificates in the context of blockchains, it has to be understood how these certificates are applied. In a WWW-context, TLS certificates are used every time a client (e.g., a web browser) contacts a web server and the web server responds with a message signed with the private key belonging to the respective certificate. The signature itself is short-lived and is newly created every time a new client approaches the webserver. In the context of blockchains, we also need a form of signature. Together with the plaintext, we refer to the signature as endorsement.

An endorsement states that the owner of a certificate intends that an address is acting on behalf or in his name. As the signed address is also capable of creating signatures, endorsements can be seen as *sub-certificates* directly aimed at addresses for blockchain networks. For that, the endorsement contains the following information:

- Address**:** The to-be endorsed address
- Domain: A fully qualified domain name from the respective certificate
- Expiry date: A date after the respective endorsement is not valid anymore
- **Flags:** Flags allow for advanced settings, e.g., if subendorsements are allowed
- Signature: The signature created by the private key of the respective certificate over the above mentioned fields.

Depending on the use case, there can be more fields like the fingerprint of the certificate. That allows users to find the respective certificate even if it is not available any more on the web server through means like Certificate Transparency.

2.3.1.2 Verifier

The verifier is a software application that retrieves the endorsement and verifies it. It depends on the context the verifier runs in (see **Section 2.3.2** for further details), but usually it obtains the certificate, verifies if it trusts the certificate (by checking if there is a signed path between a trusted root certificate and the certificate in question) and verifies if the endorsement was actually created with the respective certificate. Other checks are also applied (e.g., expiry date). The verifier ensures that either an address can be trusted or that it has access to a specific service.

2.3.2 Strategies for Using TLS-Certificates in a Blockchain Context

Leveraging TLS-certificates in a blockchain environment poses the question of the specific needs of the participants of the system. Both Gallersdörfer and Matthes (2021) and Gallersdörfer et al. (2021) solve specific needs of the user: The first aims at the usage of certificates from an off-chain perspective, the second allows for the usage of certificates in an on-chain environment. To understand the context of off-chain or on-chain usage, we need to ask several questions:

- Q1: Where is the endorsement stored? The endorsement can reside off-chain (e.g., on a web server or on IPFS) or it can be stored on-chain, either in one centralized smart contract or in individual smart contracts.
- Q2: Where is additional information stored that is required for verification? This additional information includes the respective certificate, the certificate chain as well as the trusted root certificates as well as other information, such as time. This information can be obtained off-chain (from the webserver) or it can be made available in an on-chain environment.
- Q3: Where is the verification taking place? The verification can be done off-chain, e.g., in a browser or wallet, such that one single user can decide if she trusts the endorsement or not. Alternatively, the verification can take place on-chain, such that the authenticity of endorsements can be made subjects or requirements in smart contract code.

Depending on the goal of the system, these questions are answered differently. **Figure 2** (system A) and **Figure 3** (system B) display the respective architectural decisions. It becomes evident that while in system A most of the data handling and verification is done off-chain (e.g., in a browser or a wallet), in system B the verification of the work is done entirely on the blockchain, which also requires the data for the verification (e.g (root-) certificates, …) to reside on-chain.

Both systems store the endorsement on-chain; other storage places are unreliable and for on-chain verification (as in system B it is even required to have the information available). Additional information such as certificates in system A are just retrieved from the respective sources (e.g., webserver), while in system B this information needs to be stored on-chain, as otherwise it would not be possible to verify that data within a smart contract. As for Q3, the verification for system A happens off-chain and for system B on-chain. Both systems have their

[4]http://bloxberg.org/

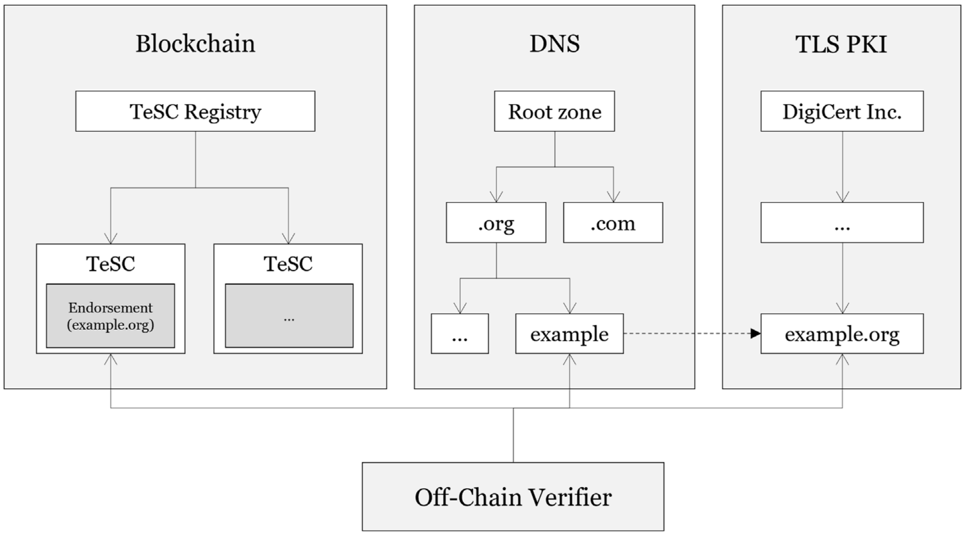

FIGURE 2 | Architecture of the system proposed in Gallersdörfer and Matthes (2021). The Off-chain verifier collects data from all sources and decides off-chain about the validity of an endorsement.

individual strengths and disadvantages, therefore it needs to be decided on the context which system to use.

In the context of this paper, we rely on system B as it allows for on-chain verification. As it only allows passive forms of validation, we extend system B by allowing entities to actively use their endorsement as a means of authentication and authorization at smart contracts. More concrete, they can use their TLS-certificates as a means to become a validator in a permissioned network. As previously described, the set of validators can be managed by a smart contract, and proposing a smart contract architecture to use TLS-certificates as a means for access control is the aim of this paper.

3 ARCHITECTURE EXTENSION

In this section, we first give more details about the design of Gallersdörfer et al. (2021). Then, we derive requirements from our use case and elaborate on how we augment the already existing architecture to enable our use case.

3.1 Previous Design

The design in Gallersdörfer et al. (2021) consists of three main components. We introduced the general notion of endorsements already in **section 2.3.1**. The x.509 certificate storage mirrors relevant certificates present in the TLS PKI and the on-chain endorsement storage stores and validates endorsements.

3.1.1 X.509 Certificate Storage

The X.509 certificate storage is a smart contract that stores, manages, and verifies X.509 certificates that are submitted to the contract. The smart contract is able to parse certificates and supports a set of cryptographic functions to validate their contents. Two characteristics are noteworthy: Bootstrapping the certificate storage and the CRUD operations it supports.

Bootstrapping a certificate storage smart contract is straightforward. Every self-signed certificate that is submitted to the contract is considered to be a root-certificate. The creator of the smart contract can begin with submitting well-known root certificate lists, e.g., by Mozilla Foundation[5]. As the verifying party later on defines which root certificates to trust, the insertion of additional, non-trusted root certificates is not an issue, as they are not considered when verifying certificates or lateron endorsements. Third parties, which are new in the network, are able to verify that the root certificates stored within the certificate storage smart contract are identical to the ones they have stored on their local machine.

The X.509 certificate storage covers the following CRUD operations:

- Create: Entities submit certificates to the smart contract one by one. Submitted certificates are parsed and validated in accordance to Housley et al. (1999) and Boeyen et al. (2008). If a parent certificate is referenced, either 1) it is already existing or 2) needs to be submitted beforehand. Then, the respective signature is verified as well. To ensure that only valid certificates are added to the storage, further checks such as the expiry of the certificate are also executed.
- Read: Single certificates can be read through a unique certificate identifier. This also allows to reference parent certificates and retrieve them as well.
- Update: Due to the nature of public key infrastructures, single properties of a certificate cannot be changed. Updated certificates rather have to be reissued to the contract. However, it is possible to update the revocation status of the certificate. To revoke a certificate, either a Certificate

[5]https://wiki.mozilla.org/CA

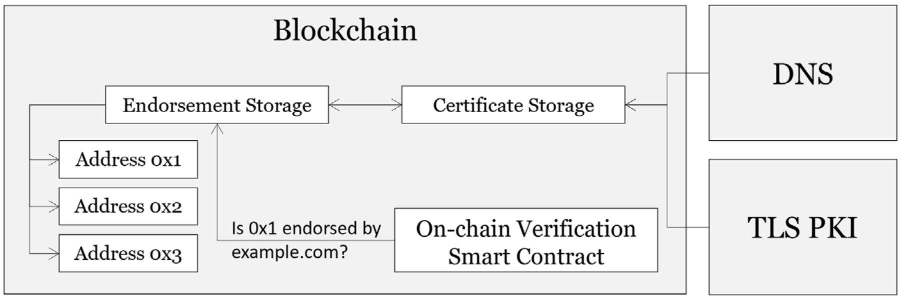

FIGURE 3 | Architecture of the system proposed in Gallersdörfer et al. (2021). All data, including certificates reside on-chain. One can ask the endorsement storage whether a certain address is endorsed by a domain. As only valid endorsements are stored within the endorsement storage, the verification only has to take place once.

Revocation List (CRL) Boeyen et al. (2008) or a Online Certificate Status Protocol (OCSP) Galperin et al. (1999) response has to be submitted to the contract.

- Delete: Certificates cannot be deleted, as the removal of single certificates could break the trust of other certificate chains.

Due to the structure of the certificate storage, it is not only possible to use certificates from the regular TLS PKI but also possible to establish own PKIs as well as use existing ones, e.g., within companies.

3.1.2 Endorsement Storage

The endorsement storage extends the certificate storage and allows for the management and verification of endorsements of respective TLS certificates. It thereby relies on the validity of three crucial parts of information: 1) the certificate of the entity signing the certificate, 2) the endorsement and the signature it contains, and 3) the root certificates the verifier trusts.

With this approach, the verification is highly efficient. All information that can be cryptographically verified is verified only once. Due to the immutability of the smart contract and the data it contains, the results of the verification can be stored within the contract and only minor details, such as the expiry date, need to be checked. These processes take place when the certificate or the endorsement is inserted in the table. Later verification requires only a minor effort.

Similar to the certificate storage, the endorsement storage also offers a set of CRUD operations, namely:

- Create: If a new endorsement is submitted to the endorsement storage, the referenced certificate is verified for its validity, it is checked if the domain names match and if the endorsement is not expired. If this is the case, the endorsement is submitted to the endorsement storage.
- Read: Similar to the certificate storage, the endorsement can also be retrieved by a key, either by the endorsed address or by the domain name. As multiple endorsements can exist, a list of endorsements is returned. The requesting party needs to ensure that an endorsement is (indirectly) signed by a root certificate that it trusts.
- Update: Endorsements itself cannot be updated. However, they can be revoked by the respective party.
- Delete: Endorsements face similar issues to certificates. It is not allowed to delete endorsements, as decentralized applications could rely on them for their functionality, even if they are expired.

3.2 Requirements

The next step is the definition of the requirements. Our goal is not only to cover the requirements for the features of our system (Functional Requirements (FR)), but also requirements in the context of usability, cost and blockchain technology (Non-Functional Requirements (NFR)). We derive basic functional and non-functional requirements for our use case consortia membership management.

We identified following functional requirements:

- FR1: Authenticate at application smart contract: An endorsed account should be able to authenticate itself at an application.
- FR2: Authorize at application smart contract: An authenticated account should be able to authorize itself at an application according to the rules of the smart contract.
- FR3: Use TLS certificate attributes for authorization: The application smart contract owner should be able to define the properties of the TLS certificate that are required for accessing specific functions.

Further, we identified the following non-functional requirements:

- NFR1: On-Chain access control decisions: To guarantee deterministic behavior, the rules, properties, and access control decisions need to be on-chain.
- NFR2: Access control without pre-provisioning of the subject at the application: An access request should not require any kind of pre-provisioning of the address at the application. This increases flexibility, decreases barriers of entry and increases the user-friendliness of the system.

TABLE 1 | Attribute types Cooper et al., 2008 supported by our ABAC system with respective OIDs Hoffman and Schaad (2010) and certificate types

OID	Attribute type	DV	OV	EV
2.5.4.3	commonName	X	X	X
2.5.4.6	countryName	—	X	X
2.5.4.7	localityName	—	X	X
2.5.4.8	stateOrProvinceName	—	X	X
2.5.4.10	organizationName	—	X	X
2.5.4.11	organizationUnitName	—	X	X

TABLE 2 | Comparison of Gallersdörfer and Matthes (2021), Gallersdörfer et al (2021) and this work.

	System A	System B	This work
Endorsement storage	on-chain	on-chain	on-chain
Certificate storage	off-chain	on-chain	on-chain
Endorsement verification	off-chain	on-chain	on-chain
Passive/active authentication	passive	passive	active

TABLE 3 | Fullfillment of requirements

	System A	System B	This work
On-chain Verification	✗	✓	✓
Openness	~	✓	✓
Compatibility	✗	✓	✗
Availability	✗	✓	✓

3.3 Attribute-Based Access Control (ABAC) and TLS Attributes

In our approach, we opt for an attribute-based access control approach. The reasoning behind this is straightforward: Our application smart contract (the contract managing the validator list) should decide on specific properties of the respective TLS certificate. Therefore, other approaches such as Discretionary Access Control (DAC) or Role-based Access Control (RBAC) are outside the scope of our approach.

In an ABAC system, the access requesting user is indirectly described by attributes from the respective TLS certificate, the attributes that describe the entity the certificate has been issued to are relevant. These attributes are stored in the *subject*, a sub-field of the *tlsCertificate* field. However, the supported attribute types differ depending on the type of the endorsing certificate. There are three different types of SSL/TLS certificates: Domain-validated (DV) certificates, Organization-validated (OV) certificates and Extended-validation (EV) certificates. Depending on the certificate type a more sophisticated validation process is required, in order to guarantee the credibility of the attributes. The DV certificates support the least attributes, while EV certificates support the most. **Table 1** depicts which attribute types are included in which type of TLS certificate. Further attribute types included in EV certificates and other certificate fields, as certificate extensions, are left out for clarity. However, if future work identifies such use-cases, our system can easily be extended to support more attribute types.

In the following we provide a brief overview of the supported attributes types, specify their OIDs and provide examples:

- commonName (2.5.4.3) - *e.g. www.example.edu*: The FQDN of the organization.
- countryName (2.5.4.6) - *e.g. DE*: The country the organization is located in.
- localityName (2.5.4.7) - *e.g. Muenchen*: The city the organization is located in.
- state Or Province Name (2.5.4.8) - *e.g. Bayern*: The state or province the organization is located in.
- organizationName (2.5.4.10) - *e.g. Blockchain University*: The name of the organization.
- organizationUnitName (2.5.4.11) - *e.g. IT*: The name of a business unit within the organization.

Access control evaluation requires access to the attributes of a specific TLS certificate that endorses an address. Therefore, for each endorsing certificate the attribute values have to be stored on the blockchain. As the certificate storage already stores most information of a TLS certificate and its certificate chain, we can retrieve all attributes required for our work from the central database.

For our use case at hand the commonName-attribute is the most relevant. Other attributes can be used as consortia blockchain operators see fit. However, given the low adoption of OV or EV certificates, we advise sticking to the commonName-attribute.

3.4 Processes to Enable TLS-Based Consortia Membership

To set up a system with the intended functionality, the following steps need to be considered. If the network is already set up and running, a hardfork is required to change the type of contract to a contract with TLS support.

- Deployment of certificate storage: At the beginning, the certificate storage contract needs to be deployed, as all other smart contracts depend on the correct management of certificates.
- Initial storage of trusted root certificates: Second, the trusted root certificates need to be stored within the certificate storage smart contract. The responsible entity can either rely on already well-known certificate authorities or define their own certificate authorities, e.g., if deployed within a company with existing PKI.
- Deployment of the endorsement storage: The endorsement storage contract is also required for the correct functioning of the validator smart contract. The responsible entity needs to deploy and reference it to the certificate storage smart contract.
- Deployment of the validator smart contract: The validator smart contract is the last contract to deploy. It references to the certificate storage as well as the endorsement storage.

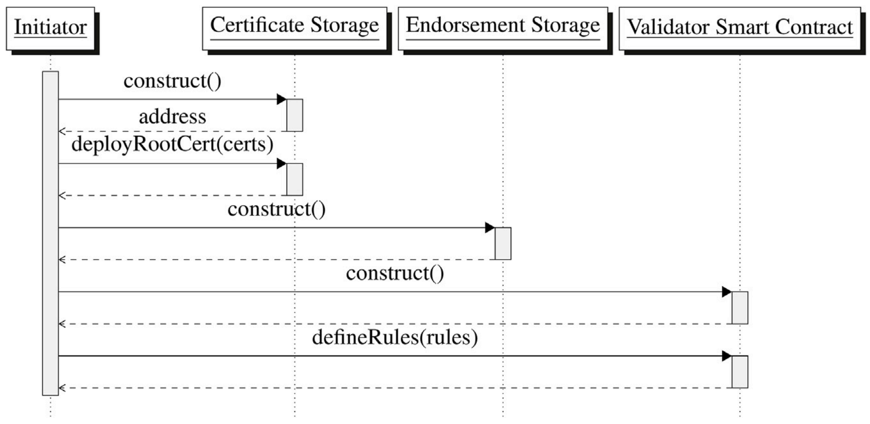

FIGURE 4 | Setup to enable automatic onboarding with TLS-certificates in the permissioned blockchain.

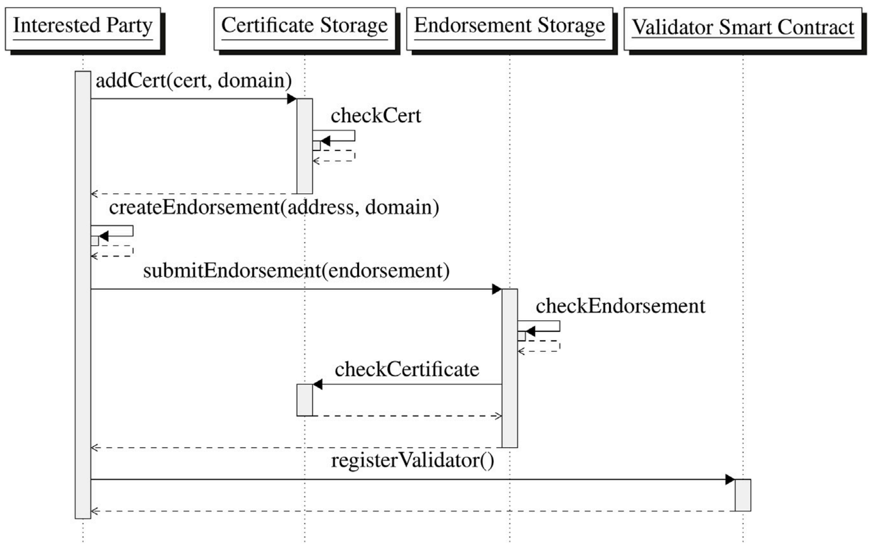

FIGURE 5 | Process for an interested party to register in the permissioned blockchain.

- Definition of rules for prospective validators: The owner of the validator smart contract should define rules and attributes that prospective validators need to fulfill.
- Optional: Establish hard fork: If the network already runs, it is advisable to upgrade all nodes in the network to the latest genesis. json-file in order to account for the future hard fork.

All contracts can be set up as precompiled contracts at the beginning of the network, rendering manual creation of contracts unnecessary. After finishing all steps, other prospective validators are able to join the network and begin their work as validators in the permissioned network. They have to follow these steps:

- Adding respective certificates: As previously outlined, TLS certificates need to be added on a one-by-one basis. A prospective validator needs to add all intermediate certificates and its own certificate to the certificate storage.
- Creating endorsement: The entity needs to use the private key of their certificate to create an endorsement including their address, domain and expiry date. As this verification happens only once, the period for expiry can be short.
- Submit endorsement: The endorsement needs to be submitted to the endorsement storage. It gets verified alongside the certificates previously submitted to the certificate storage.
- Register as validator: If the prospective validator fulfills the requirements set out by the owner of the network, it can register itself with the validator smart contract. The contract checks the requirements and adds the asking party to the set of validators.

It is up to the owners of the network to set the gas fees and if they allow the creation of transactions without fees. In this case it is advisable, as then new members of the network are not required to be pre-funded with the respective currency of the network.

The process for setting up the network, introducing respective contracts, allowing anyone to connect to the network and joining are depicted in **Figures 4**, **5**.

3.5 Evaluation

We briefly discuss architectural decisions and their impact on other use cases as well as the fulfillment of our requirements.

Albeit the architecture previously proposed in Gallersdörfer et al. (2021) seems complex, our proposal allows for the usage of TLS certificate properties to be used in the context of access control in smart contracts. For our use case we authorize the asking address for every call to the protected smart contract. In this example use case for managing validators in permissioned blockchain networks, only one call is required to gain access to the system.

Our previous work indicates that the initial filing of the certificate storage requires some gas; for 13 root certificates and 24 intermediate certificates the total gas amounts to 33 million gas in Ethereum Gallersdörfer et al. (2021). As this has to be paid only once, it might be worthwhile to deploy on the mainnet. Adding a single domain certificate alongside an endorsement consumes about 1.3 million gas, which might be a reasonable price for single actors.

However, in any permissioned blockchain in which gas costs do not play a relevant role, this is not an issue. As permissioned networks do not have the same requirements and transaction fees as permissionless networks, such a system can easily be deployed. Even storage wise, given that one certificate requires about 1 kb of storage, permissioned blockchains should be able to handle this amount of load easily.

Considering the requirements that we set out for our approach, the following statements can be made:

- FR1: An endorsed account can authenticate itself at an application smart contract. ✓
- FR2: An endorsed account can authorize itself at an application smart contract. ✓
- FR3: The entity owning the application smart contract can define the requirements for authorization. ✓
- NFR1: On-Chain access control decisions: All relevant data for decisions is stored on-chain in a trustless manner. ✓
- NFR2: Access control without pre-provisioning of the subject at the application: The account can directly call the respective smart contract without pre-provisioning. ✓
- NFR3: Minimal costs of user management, authentication and authorization: Transaction fees and costs were not considered or optimized for. ✓

3.6 Comparison With Previous Approaches

The systems described in this paper, in Gallersdörfer and Matthes (2021), and in Gallersdörfer et al. (2021) deviate from each other. We describe their differences from a functional, security and requirements perspective.

3.6.1 Functional Perspective

In **Table 2**, we display the key differences between the two previous systems Gallersdörfer and Matthes (2021), Gallersdörfer et al. (2021), and this work. System A and B deviate in where the verification of endorsements takes place; system A allows for a decentralized, off-chain verification in which every single user executes the verification. This has the advantage that different system configurations (e.g., trusted root certificates) are considered as well as the low cost of the approach Gallersdörfer and Matthes (2021). This also comes with the disadvantage that there is no on-chain information about the validity of the endorsement; this is crucial for applications that rely on such information, e.g., address independent payments. In this work, we extend system B by allowing to actively use the endorsement to authenticate and authorize at a Smart Contract, using attribute-based access control.

3.6.2 Security Perspective

An extensive analysis of the security implications of the underlying systems is given in the respective previously published work. Nonetheless, we highlight common security themes and provide an insight into where these systems deviate. As this work builds upon system B, potential attack vectors remain identical.

3.6.2.1 TLS as the Underlying System

TLS and DNS are relied upon in all three systems. It allows us to use cryptographic key material alongside human-readable information (e.g., the domain name) to assert this information to addresses within blockchain systems. Thus said, an error that occurs on the TLS layer (e.g., maliciously issued certificates) can be used to obtain a fraudulent endorsement, independent if it takes place in systems A, B, or this work. Attacks like these (especially on a large scale) are not common; nonetheless, due to the transparency of the blockchain, one can monitor for such attacks and mitigate their damages (e.g., by warning users).

3.6.2.2 Ways of Verification

One of the key differences is that in system A, the verification of information happens off-chain and uses live data from the WWW (e.g., by accessing the webserver or Certificate Transparency). In contrast, in system B and this work, the verification is done on-chain once and only if contradicting information (e.g., a certificate is revoked) exists, it needs to be pushed on-chain by the respective responsible parties.

Second, given the novelty of the solidity programming language in the case of Ethereum (or any other novel blockchain-based language), no libraries exist for verifying certificates. Contrary to that, programmers can access many libraries in other programming languages like Javascript, Python, or Java. Therefore, the off-chain verification mechanisms might be more robust than on-chain mechanisms.

3.6.2.3 Human Error

In all systems, humans are prone to be attacked or social-engineered into sending their funds regardless. First, typo-squatting is still a common theme, e.g., replacing an upper-case *I* for a lower-case *l* and tricking the users into believing that they are accessing the correct domain. Applying to system A, another approach that can also result in the loss of funds is just telling the users to ignore the warning of the browser-based wallet, sending their funds regardless. Educating users is an essential step in combating such approaches; given the complexity of the technology, significant efforts are required.

3.6.3 Requirements Perspective

The previously prop osed systems fullfil a set of functional and non-functional requirements. As this work builds upon system B, we do not introduce these requirements in this paper again. To better understand these systems and their differences, we first discuss differing requirements and then display them in **Table 3**

The first relevant functional requirement is introduced in system B: Allow the verification of an endorsement to take place on-chain. System A introduced the notion of endorsements but only allowed verification in an off-chain environment, as not all data was available on-chain. System B allowed for the verification of endorsements on-chain.

The second deviating functional requirement is the system's openness, meaning that anyone can participate and use the technology. As we rely on TLS, any certificate authority deemed trustworthy by a verifier can be used. There is a slight difference between systems A and B/this work, as in system B, the existence of alternative Certificate Authority (CA) is explicit, meaning that a verifier knows that an alternative CA is used. In system A, this information has to be communicated via other means. Therefore, it is less likely that alternative CAs are used within system A.

A third functional requirement differing between systems is compatibility: Does the system support contracts that have been deployed without the respective system A/B/this work in mind? System A does not, as it requires the smart contract to adhere to a specific interface standard. As contracts cannot be updated, once deployed, contracts cannot be made compatible to system A. The endorsements are stored in a centralized Smart Contract in system B, independent of the actual endorsed smart contract. Therefore, the respective contract can also be endorsed retroactively. This work, as it builds upon system B, also allows the retroactive endorsement of existing smart contracts but does not allow existing contracts to be used for verification purposes, as they still need to support the respective interface. In this specific use case, a new contract can easily be deployed and used to onboard new consortia members.

The availability requirement introduces the notion of dependency on other entities or systems. As system A does depend on the webserver and other systems, it might be possible that verification attempts fail due to missing data. This is not the case for system B and this work, as all information is stored and kept on-chain for verification purposes.

All additional requirements introduced in this paper do not apply to the previous systems. Table 3.6.3 contains the requirements deviating between different systems.

3.7 Related Work

This work builds upon previous work aiming at providing a way to utilize TLS-certificates in an on-chain context. To that system, related work exists: ENS delivers a means to register domain names with the TLD. *e*th, finding increasing adoption within the community Johnson et al. (2020). CertLedger shifts the processes of Certificate authorities on blockchain systems, allowing for these processes to be more transparent and, thus, more secure Kubilay et al. (2019). Systems like Instant Karma PKI (Matsumoto and Reischuk (2017)) introduce game-theoretical models that incentivize certificate authorities and offer insurance against fraudulent operations. While these systems exist and sometimes see some adoption, we are unaware that any is used for an active authentication mechanism or in a consortia membership context.

4 CONCLUSION

This manuscript extended previous research about mirroring and migrating parts of existing Public Key Infrastructures in a Blockchain environment to support the management and onboarding of validators in permissioned blockchains. We defined requirements and discussed architectural decisions to actively use endorsements stored within the blockchain. Our approach allows consortia blockchains to enhance their onboarding process and reduce administrative and communicational costs. Voting in such networks should be more trustworthy, as well as the centralization to one single party should be reduced.

We intend to extend the framework to support access control for multiple addresses for one endorsement as future research. In the current use case, only one entity must be endorsed, but there are other use cases in which firms could issue multiple entities the right to access specific resources. For example, a car manufacturer could decide to equip their cars with specific wallets. Endorsing these wallets with the respective TLS certificate could yield an ecosystem where vehicles have access to certain services or benefits, such as reduced parking fees. A system like that can both prevent the existence of malicious copies and an easy extension of the marketplace for the respective cars of the manufacturer.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

UG, J-NS and FM contributed to conception and design of the study. UG wrote the first draft of the manuscript. JNS developed the application and interfaces. UG and J-NS wrote sections of the manuscript. All authors contributed to manuscript revision, read, and approved the submitted version.

REFERENCES

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., et al. (2018). "Hyperledger Fabric: a Distributed Operating System for Permissioned Blockchains," in Proceedings of the thirteenth EuroSys conference, 1–15.

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *RFC* 5280. , 2008 Dataset. doi:10.17487/RFC5280

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and Polk, W. (2008). Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. *Tech. Rep*. doi:10.17487/rfc5280

Dierks, T., and Allen, C. (1999). The TLS Protocol Version 1.0. *RFC* 2246. , 1999 Dataset. doi:10.17487/RFC2246

Dierks, T., and Rescorla, E. (2006). The Transport Layer Security (TLS) Protocol Version 1.1. *RFC* 4346. , 2006 Dataset. doi:10.17487/RFC434610.17487/rfc4346

Gallersdörfer, U., Groschupp, F., and Matthes, F. (2021). "Mirroring Public Key Infrastructures to Blockchains for On-Chain Authentication," in International Conference on Financial Cryptography and Data Security (Springer), 415–430. doi:10.1007/978-3-662-63958-0_33

Gallersdörfer, U., Klaaßen, L., and Stoll, C. (2020). Energy Consumption of Cryptocurrencies beyond Bitcoin. *Joule* 4, 1843–1846. doi:10.1016/j.joule.2020.07.013

Gallersdörfer, U., and Matthes, F. (2021). "Tesc: Tls/ssl-Certificate Endorsed Smart Contracts," in IEEE DAPPS 2021 (The 3rd IEEE International Conference on Decentralized Applications and Infrastructures). doi:10.1109/dapps52256.2021.00016

Hoffman, P., and Schaad, J. (2010). New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX). *Tech. Rep*. doi:10.17487/rfc5912

Housley, R., Ford, W., Polk, W., and Solo, D. (1999). Internet X.509 Public Key Infrastructure Certificate and CRL Profile. *RFC* 2459. , 1999 Dataset. doi:10.17487/RFC2459

Johnson, N., Lau, J., Eigenmann, D., and Millegan, B. (2020). *Ethereum Name Service*. Dataset.

Kubilay, M. Y., Kiraz, M. S., and Mantar, H. A. (2019). Certledger: A New PKI Model with Certificate Transparency Based on Blockchain. *Comput. Security* 85, 333–352. doi:10.1016/j.cose.2019.05.013

Matsumoto, S., and Reischuk, R. M. (2017). "IKP: Turning a PKI Around with Decentralized Automated Incentives," in Proceedings - IEEE Symposium on Security and Privacy (IEEE), 410–426. doi:10.1109/sp.2017.57

Mockapetris, P. V. (1987a). Domain Names - Concepts and Facilities. *RFC* 1034. , 1987a Dataset. doi:10.17487/RFC1034

Mockapetris, P. V. (1987b). Domain Names - Implementation and Specification. *RFC* 1035. , 1987b Dataset. doi:10.17487/RFC1035

Myers, M., Ankney, R., Malpani, A., Galperin, S., and Adams, C. (1999). X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. *RFC* 2560. , 1999 Dataset. doi:10.17487/RFC2560

Rescorla, E., and Dierks, T. (2008). The Transport Layer Security (TLS) Protocol Version 1.2. *RFC* 5246. , 2008 Dataset. doi:10.17487/RFC5246

Rescorla, E. (2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446, Dataset. doi:10.17487/RFC8446

Stoll, C., Klaaßen, L., and Gallersdörfer, U. (2019). The Carbon Footprint of Bitcoin. *Joule* 3, 1647–1661. doi:10.1016/j.joule.2019.05.012

Weise, J. (2001). Public Key Infrastructure Overview. *Sun BluePrints OnLine* August, 1–27.

Xin, W., Zhang, T., Hu, C., Tang, C., Liu, C., and Chen, Z. (2017). "On Scaling and Accelerating Decentralized Private Blockchains," in 2017 ieee 3rd international conference on big data security on cloud (bigdatasecurity), ieee international conference on high performance and smart computing (hpsc), and ieee international conference on intelligent data and security (ids) (IEEE), 267–271. doi:10.1109/bigdatasecurity.2017.25

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.