Mirroring Public Key Infrastructures to Blockchains for On-chain Authentication

No Author Given

No Institute Given

Abstract. In blockchain systems, the lack of established identity management processes poses a problem for applications requiring smart contract owners to be authenticated. One issue that previously proposed solutions face is the accumulation of a critical mass of trusted data that makes the system usable. In this work, we propose an identity assertion and verification framework for Ethereum that overcomes this bootstrapping problem. It achieves this by leveraging TLS certificates, which are part of the established infrastructure that is commonly used for authenticating internet connections. We design and implement a TLS certificate-based authentication framework whose key features are the smart contract-based validation and storage of certificates and address-identity bindings. Looking at the current TLS ecosystem, we find that a large share of all domain certificates is issued by a small number of intermediate and root certificates. Therefore, we decide to store and maintain certificates in one smart contract to minimize processing costs. The evaluation of our prototype implementation shows that the associated cost of our system is within a feasible operating range, with the costs of submitting a new certificate currently averaging around 1.81 \$ and the cost of creating an address-identity binding averaging around 1.32 \$. Our system is a pragmatic and, most importantly, quickly bootstrapped method for an identity assertion and verification framework for Ethereum.

Keywords: Blockchain · Public Key Infrastructure · Authentication · Smart Contracts · Ethereum · Certificates.

1 Introduction

The world wide web relies on public key infrastructures (PKI) to reliably identify and authenticate remote communication partners, enabling the Internet as we know it. The Domain Name System (DNS) allows the user to identify and direct their requests to the respective party behind a domain name (e.g., *example.org*) [19]. TLS/SSL-certificates[1], which are distributed securely through the TLS-PKI, map public cryptographic keys to these domains names to enhance the communication by ensuring the privacy and integrity of messages as well as confirming that the user is talking to the intended party [23]. In

[1] Often, the terms *TLS* and *SSL* are used interchangeably. In this paper, we only use the term TLS.

contrast, blockchain networks do currently not offer a well-established identity management system with human-friendly names. The native decentralized identity management solely depends on private and public key pairs registered on the blockchain[2]. While this prevents malicious parties from directly interfering with transactions, it does not facilitate the authentication of the counter party, as the public key is not mapped to a real-world identity.

Efforts to bridge this gap currently focus on establishing new identity management solutions which are guided by the blockchain core principles of decentralization and trustlessness: No party should be required to solely run the system or be able to interfere with its operation. A well-known example for such a system is Ethereum Name Service (ENS) [13]. ENS allows for the decentralized registration of domain names with the top level domain *.eth* to be used within the Ethereum blockchain. However, such newly-established and decentralized systems face huge bootstrapping issues.

Bootstrapping is a serious issue for these projects, as they face a lack of adoption from two distinct groups: users and service providers, e.g., companies. First, enterprises need to support these upcoming standards and integrate them in their applications and wallets. Second, users need to install and use these wallets, understand the functionality, and recognize the implications of these specific standards. Both groups are hesitant to invest their time and money in these systems as long as it lacks adoption of the respective counter party.

For that reason, we explore and evaluate the mirroring of existing, established PKIs and their certificates (alongside unique attributes such as a name) to blockchain networks. We rely on a two-fold approach:

1. Set up an on-chain structure to insert and verify the validity of the certificates managed in the PKI, resulting in the existence of trustworthy certificates and their attributes in the blockchain, and
2. enable the signature verification of these certificates, such that statements signed by the private key of these certificates can be verified on-chain.

This system allows us to verify on-chain statements[3] made with certificates that bind the respective unique attribute of the certificate to a smart contract or Externally Owned Account (EOA). Afterwards, third party smart contracts can verify whether the binding is valid and commence interactions with the account.

Several different PKIs are suitable for investigation in our work. Therefore, we allow the usage of any PKI that supports X.509 certificates [8]. In this paper, we evaluate our approach with the broadly used TLS public key infrastructure. Certificates issued via the TLS PKI are bound to fully qualified domain names (FQDNs) as unique attributes. This allows us to bind a smart contract to FQDNs. As 98% of all website visits rely on TLS [11], bootstrapping issues are eased.

In this work, we investigate two questions:

[2] Often, a subset of the hash of the public key is used, e.g., `0x42Ff4fa0...89024`

[3] We refer to these statements as *endorsements*. The definition follows in section 2.

– How can naming attributes of existing PKIs in an on-chain blockchain context be leveraged?
– What are the constraints of leveraging existing PKIs in a blockchain environment?

The following paper is organized as follows: Section 2 introduces the key concepts of the design and architecture of the on-chain PKI verification system. In Section 3, we discuss the suitability of the TLS PKI and evaluate key metrics of the system, such as costs and fulfillment of the requirements. In Section 4 we contrast our approach with related work and conclude the paper in Section 5.

We refer readers who are not familiar with the TLS ecosystem to [23], and readers who are not familiar with blockchain technology and Ethereum to [4,21] for background information on these topics.

2 System Design and Architecture

The aim of our system is to enable users or smart contracts to verify that an Ethereum account (an EOA or a smart contract) is assigned the name attribute of a X.509 certificate. Trusted root certificate authorities sign (indirectly via intermediary certificates when applicable) domain certificates, which we afterward use to create signatures that allow us to verify the assignment of an account to a domain. To verify that such an assignment is created and is valid **on-chain**, we require several components in our system: First, endorsements (section 2.1) contain details about the assignment of an account to a name attribute. The on-chain certificate database (section 2.2) ensures the validity of newly added root, intermediary or server certificates. The on-chain endorsement database (section 2.3) checks the attribution as well as context-dependent properties of the respective certificates such as time or trusted root certificates[4].

2.1 Endorsement

We define the endorsement of an Ethereum address as the signature of the address value together with optional associated data. An endorsement indicates that the endorser claims to own the address, i.e. that they receive ingoing funds, control outgoing funds, vouch for data associated with the address, and are the originator of outgoing transactions. Endorsements need to present some kind of liability and make only sense in scenarios where an adversary cannot gain advantage by signing an address they do not control.

To standardize the endorsement and avoid misuse, we specify the format and content of endorsements. This means the endorsements must be unambiguous. In particular, endorsements need to meet the following requirements:

– An endorsement issued for one Ethereum address may not be reused for another Ethereum address.

[4] We provide the option to define the set of trusted issuers to keep the system open and flexible for any other X.509-based PKI.

– It must be clear to which web domain an address is linked. The situation when this is unclear might arise when a certificate is issued for multiple web domains.
– It must be possible to identify the domain certificate with which the endorsement was created in order to retrieve the public key and to check the validity and revocation status of the certificate.
– It should be possible for the issuer to specify an expiration date of the endorsement.

An endorsement comprises a signature and an associated claim. The signature is computed over the hash of the claim. The claim contains the address account $addr$, the web domain ID_{domain}, the unique certificate identifier ID_{cert}, and the optional expiration date $date_{exp}$. We informally characterize the claim C in (1) and the endorsement E in (2).

$$C = \{addr|ID_{domain}|ID_{cert}|date_{exp}\} \tag{1}$$
$$E = \{C, sign(hash(C), key_{priv})\} \tag{2}$$

2.2 On-chain X.509 Certificate Storage and Validation

There is no absolute truth on the validity of TLS certificates as different entities might trust different root certificate authorities. Therefore, we need to design a mechanism that allows TLS certificate validation solely based on user preferences and in the context of the Ethereum blockchain. Only if this is possible, we can later validate endorsements on-chain.

To remove any dependency on external systems, the information that is required for certification validation needs to reside on-chain. We call this approach mirroring (a part of) the TLS PKI to Ethereum. The information that is required to validate a certificate is the **whole certificate chain** from server to root certificate, the **set of trust anchors** as defined by the verifier, and the **validation procedures**.

The validation procedure for X.509 certificates can be implemented and offered on-chain as Ethereum library. It is a security-critical component, needs to be carefully implemented, and the source code needs to be openly available to be trusted by users. As the set of trust anchors is specific to the verifier, each contract that acts as verifier needs to declare their own set.

Due to the nature of the TLS-PKI (which we further evaluate in section 3) we store certificate chains in one central contract. Previously added certificates need to be stored only once. Additionally, if the validity of a certificate and its chain is asserted by the database when it is submitted and only valid certificates are accepted, the validation of the certificate needs to be performed only once and can be shared by multiple server certificates. When verifiers are interested in the validity of an endorsement, they are not required to verify the certificate chain again.

A difficulty with the migration approach is the assertion of the revocation status of certificates. Both common revocation mechanisms, namely Certificate Revocation Lists (CRLs) and Online Certificate Status Protocol (OCSP) responses, are documents that are valid for a certain time period and are commonly signed with the issuer private key and can consequently be verified on-chain. The idea is that the database entries of certificates can be updated with the current corresponding revocation information. This needs to be repeated while the certificate is valid and the validity period of the revocation status information expires. If a certificate is revoked, its status cannot be changed anymore.

In the following, we describe the CRUD (create, read, update, delete) operations for certificates stored on the certificate database contract.

Create Certificates are submitted to the database one-by-one. Anyone can submit certificates. Before a certificate is stored, it is confirmed that it is valid. This check is performed in accordance to RFC 5280 [8]. As the signature of the certificate needs to be verified, the certificate must either be a self-signed certificate or the certificate's issuer's certificate must already be stored in the database. The validity period of the certificate must not be expired. If the certificate validation is successful, the relevant information is retrieved from the certificate and stored in the database. This includes a pointer to the entry of the issuer certificate; in the case of self-signed certificates, it is the certificate itself. The revocation status information is set to *unknown*. If the certificate validation is not successful, the certificate is rejected.

Any self-signed certificate with valid format and content can be added to the database and subsequently act as trust anchor. This enables anyone to create and maintain their own application-specific PKI.

Read Certificate information can be retrieved from the database with a unique certificate identifier. The certificate chain can be retrieved thanks to the pointers that refer to the issuer of each certificate.

Update The only information that can be updated is the revocation status of certificates. For this purpose, either the CRL or the OCSP response corresponding to a certificate can be submitted. The submitted information is only used to update the revocation status information if it is valid and signed by the certificate's issuer. For the CRL, the certificate status is considered as *not revoked* when its serial number is not contained in the CRL and considered as *revoked* when it is contained. For OCSP responses, the certificate status is updated to the status that is contained in the response. In both cases, information about the time of the last update and the expiry date are stored. Once a certificate is marked as *revoked* in the database, the state cannot be reversed to *unknown* or *not revoked*.

Other certificate attributes cannot be updated in the database as all information reflects the information of the submitted certificate. If altered information is required, a newly issued certificate must be submitted with a new unique certificate identifier.

Delete Once submitted, certificates cannot be deleted. This is because other certificates and endorsements may rely on this certificate and their validity and revocation status cannot be verified sufficiently if certificates and their chain of trust are missing.

2.3 On-chain Endorsement Validation

To make a decision on trust based on an account endorsement, the verifier relies on three components: 1) The validity of the **signer certificate** including its chain, the **validity of the signature in the endorsement**, and the verifier's **trusted root authorities**. The verifier can define its own set of trusted roots, if they want to rely on alternative X.509 based PKIs. We also need to take into account the context, especially the time of the verification. Both certificates and endorsements contain information about their expiry.

The validity of the **signer certificate** and its chain is ensured by the on-chain certificate storage (see section 2.2) which a potential verifier relies on. As the endorsement itself links to the respective certificate, the certificate and its public key can be obtained cost efficient. The verifier only needs to verify that the certificate is present in the database, that the root certificate is part of the previously defined trusted roots, that the validity period has not expired, and that the certificate has not been revoked. These operations are significantly cheaper than performing the full validation for a certificate chain.

The validity of the **endorsement** follows a similar approach: Upon retrieval of the public key of the respective certificate, the verifier is able to check if the private key of the respective certificate actually created the signature in the endorsement. Again, they need to validate that the validity period has not expired and that the endorsement has not been revoked. As this signature verification is an expensive operation in blockchains, we further propose a central database for endorsements to execute these operations only once.

Endorsement Database In addition to the certificate database, we propose a central database for storing endorsements. Besides the cost-reduction of verifying endorsements, providing a central database empowers verifiers to proactively and conveniently search for endorsements. Such a database query can have two distinct goals: The verifier might either be interested whether and by whom a specific Ethereum address was endorsed or whether there exist endorsements for a specific web domain. In addition, a central database facilitates the revocation of endorsements. Another advantage is that the endorsement can be validated upon submission. Subsequent parties interested in the endorsement do not need to perform the validation again.

The external-endorsement database provides the following functionality:

Create An endorsement E, as defined in section 2.1, is submitted to the database. The validation procedure retrieves the certificate with the certificate ID ID_{cert}, checks that the certificate is issued for the web domain ID_{domain}, and obtains the public key key_{pub}. If the endorsement's signature is valid and not expired, the endorsement it stored in the database.

Read Endorsements can be retrieved with $addr$ or ID_{domain} as key. As multiple endorsements per account or web domain may exist, the query returns a set of endorsements. The querying party is responsible for checking the endorsements for one that is signed by a certificate whose root certificate they trust.

Update Endorsements themselves are immutable information. The only associated information that may change is the revocation status. If the original issuer of an endorsement wants to revoke it, they sign the respective information and store it with the endorsement.

Delete Unexpired endorsements may not be deleted from the database. Some applications might also accept expired endorsements, therefore, expired endorsements should not be deleted while it is allowed to do so. However, if an endorsement was revoked, the revocation information should persist.

Revocation of Endorsements External endorsements can be revoked by updating their revocation flag. For this purpose, the corresponding certificate owner can create a ”revocation signature” which has the following format:

$$R = sign(hash(addr|ID_{domain}|date_{exp}|0xFF), key_{priv}) \tag{3}$$

This revocation information is submitted to the endorsement database. The smart contract verifies the correctness of the provided signature and, if the signature is valid, marks the endorsement as revoked. Again, a previously revoked endorsement (similar to certificates) can never be valid again. A new endorsement has to be created.

3 Evaluation & Discussion

To assess our system in the context of the TLS-PKI, we first need to understand the structure and organization of the TLS certificate hierarchy. This PKI is a well-established system that is omnipresent in today's world wide web. Since 2013, over 3,7 billion certificates have been logged in Certificate Transparency [16]. As this enormous data set is not really accessible[5], we use certificate data provided by Censys [9] and define two subsets of certificates: Subset S_1 contains all root, intermediary, and domain certificates that 1) belong to a commonly trusted certificate chain[6] 2) were added to Censys before the 21st of April 2020 and (3) expire after this date. In total, 204,166,070 certificates fulfill these requirements. Subset S_2 contains the domain certificates of the top 1,000 most visited websites[7]. After eliminating invalid, expired and duplicate certificates and certificates with an invalid trust path, S_2 contains 869 unique domain

[5] Assuming conservatively 1500 bytes per certificate, this data set would amount roughly to 5 Tebibyte.

[6] We use the Mozilla NSS root store: https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/, accessed 09/05/2020

[7] https://www.alexa.com/topsites, accessed 09/05/2020

certificates. We use S_1 for a general understanding of the TLS ecosystem (section 3.1 and use S_2 to test and evaluate compatibility (section 3.2) and costs of our system (section 3.3). Afterward, we discuss security implications of our system in section 3.4.

3.1 TLS-PKI Structure

Out of the 204,166,070 certificates in S_1, 3,345 are certificate authority (CA) certificates, 204,162,724 are domain certificates, and one certificate is of version X.509 v1 and does therefore not include this information. We define a level-x certificate as a certificate where the shortest trusted path to the root certificate contains x certificates. In S_1, 153 certificates are level-1 certificates (meaning that 153 certificates are in the root store), 2,387 are level-2 certificates, 203,838,127 are level-3-certificates, 325,196 are level-4 certificates, and a negligible number of 207 are level-5 certificates. This means that the most common structure for chains of trust is "domain certificate – intermediate certificate – root certificate". There are no certificates that are level 6 or higher.

It can be expected that each CA certificate is responsible for issuing and maintaining a significant amount of certificates. To find out whether there are differences regarding the number of certificates depending on one CA certificate, we examine S_1 in a bottom-up approach: We group domain certificates by their issuer and count the number of certificates in each group. From the cardinality of the groups, we can derive the number of intermediate and, ultimately, the number of root certificates required to cover a certain percentage of domain certificates.

At first, we take a look at intermediate certificates that issue domain certificates and order them by how many valid certificates they issued. The by far most prevalent issuer of domain certificates is "Let's Encrypt Authority X3", the currently active intermediate for Let's Encrypt with 123,826,849 issued certificates, a share of over 60%. The top five intermediates together cover over 91% of domain certificates, eight intermediates are required for 95% and 26 for 99%.

Of course, these numbers do not represent the total numbers of CA certificates required to cover the domain certificates as we must take root certificates and, in case of chains containing more than three certificates, additional intermediate certificates into account. A first look at the data shows that root certificates do not scale quite as well as the intermediate certificates: The top six intermediate certificates are all signed by unique roots. This means that in total, $2 \times 6 = 12$ CA certificates are required to cover 93% of certificates. The share of 98% of certificates can be covered by at most 37 CA certificates, divided into 24 level-2 (intermediate) certificates and 13 level-1 (root) certificates.

The numbers show that it is possible to validate the vast majority of certificates even when only a small subset of root and intermediate certificates are available. A very high coverage of domain certificates can be achieved with a low number of CA certificates. In addition, one root or intermediate certificate is the issuer to huge number of server certificates. Therefore, centralizing the validation and storage of certificates – in contrast to validating the whole chain for each server certificate anew – is more cost-efficient than autonomous approaches within single smart contracts as outlined in 2.2.

3.2 Compatibility

To test the compatibility of our prototype Solidity implementation[8] of our system and to measure its performance with real and commonly used certificates, we rely on data set S_2. Out of this set, we remove certificates whose certificate chain contains signatures that are using algorithms not yet supported by our implementation, such as ECDSA or SHA-384. Our final testing set is comprised of 576 certificates that serve 660 different domains, in addition to 47 intermediate and 21 root certificates that are required for valid trust chains. This means that our testing set contains 644 certificates in total.

We create a fresh instance of our system and consecutively add all root, intermediate, and domain certificates. All certificates are accepted as valid and added to the database. This complies with the desired behavior, as we have only included valid certificates in this test data set. Furthermore, none of the certificates contains a critical extension that our validation routine does not support (as we describe in section 3.4). Considering the nature of our data set, this is a good indicator that special critical extensions are uncommon for TLS certificates and that our implementation is compatible with most certificates.

3.3 Costs and Performance

The usage cost in form of transaction fees of the Ethereum smart contract is an important factor to the success and viability of our system and demand cost-efficiency, especially for the verifier. To gain a perspective on the cost to be expected, we once again consider the modified data set S_2 from section 3.2.

Certificates We submit all certificates in this set with one certificate per transaction. Table 1 displays the observed gas usage, ether, and USD cost by transaction, grouped by root, intermediate, and domain certificates[9]. We observe the following results: The median value of the root certificates is the highest. We conclude that this is the case because the majority of root certificate is self-signed using SHA-1, whose computation on Ethereum costs significantly more than SHA-256. The cost for intermediate certificates is relatively homogeneous, with some outliers that are signed using SHA-1. For domain certificates, the submission cost differs significantly. As domain certificates are commonly issued using the up-to-date SHA-256, the choice of algorithm is not the source of this circumstance. Instead, the reason for this occurrence is the size of domain certificates, especially the number of subject alternative names it specifies. The larger a certificate, the more it costs to parse and validate it, and the larger the SAN

[8] An implementation of our prototype is available in [5].

[9] We assume a gas fee of 11.1 Gwei and a conversion rate of 206 US dollar per ether, as observed on the 30th of April 2020 on `https://coinmarketcap.com`.

field, the more gas is paid for writing it to storage. The most costly certificate specifies 225 subject alternative names.

	Root certificate			Intermediate certificate			Domain certificate		
	gas	ether	$	gas	ether	$	gas	ether	$
min	705,035	0.0078	1.60	750,584	0.0083	1.70	544,777	0.0060	1.23
1st	770,455	0.0086	1.77	762,129	0.0085	1.75	733,073	0.0081	1.66
med	1,105,114	0.0123	2.53	783,324	0.0087	1.79	793,954	0.0088	1.81
3rd	1,170,981	0.0130	2.67	832,031	0.0092	1.89	903,813	0.0100	2.06
max	1,537,513	0.0171	3.52	1,233,724	0.0137	2.82	4,503,213	0.0500	10.3

Table 1. Cost of certificate submission in gas usage, ether, and US dollar.

In section 3.1, we showed that by adding 13 root and 24 intermediate certificates, we can cover 98% of all certificates. Calculating with an average gas usage of 1,041,580 for submitting a root certificate and 825,926 for submitting an intermediate certificate, an initial investment of $(1,041,580 \cdot 13) + (825,926 \cdot 24) = 33,362,764$ gas (equivalent to 75.60 $) would mean that afterwards 98% of all current certificates can be added and only incur the cost for the domain certificate submission.

Endorsements The cost of adding an endorsement does not fluctuate as much as for certificates as only one signature algorithm is used (RSA-SHA256), and endorsements are constant in size except for the length of the domain name. For submitting an endorsement to the external database, we measure a cost of around 577,219 gas (1.32 $).

3.4 Security Considerations

The security of our system relies on three pillars: (i) the implementation of the certificate validation routine and the databases, (ii) the integrity of the TLS system and its certificate authorities, and (iii) the ability of users to map domain names to real-world identities. We briefly discuss these three aspects in this section.

Security of the Certificate and Endorsement Frameworks We purposefully designed and implemented our system in a way that does not give one or a number of entities privileges for the system. Once the system is deployed, it is an immutable piece of code. On the one side, this means that our system cannot be subject to any censorship and or can be influenced by an authorized party. On the other side, this means that errors and vulnerabilities cannot be patched. Therefore, the system must be crafted cautiously.

In the past, the validation of TLS certificates has been a troublesome topic: Many TLS certificate verifier applications have been shown to have critical flaws that lead to invalid certificates being accepted. We aim to minimize the possibility of such critical flaws with two methods. Firstly, we keep the capability of our validation routine purposefully small and support only the most important extension types. Less functionality means less surface for errors and attack vectors. Secondly, make sure that our implementation does not repeat mistakes that were made in the past [1,7,18]. However, this is no guarantee for the correctness, and in the future, code audits, further testing, and possibly formal verification should be performed before the system is deployed.

Security of the TLS Ecosystem In the past, the TLS PKI has been under criticism as all trust is transferred to CAs, which makes them a single point of failure, and CA misbehavior has not been unobserved in the past. However, as the TLS system is widely adopted and ”too big to fail”, in the past a lot of considerations have been made to improve its security. For example, with the introduction of CT [16], a large step has been made towards the transparency of the TLS PKI and the issuance processes of CAs. It is no longer possible for a CA to issue a fraudulent certificate undetected. Furthermore, due to its wide deployment, the TLS is thoroughly investigated by security researchers in the past and in the present. A system that is set-up newly does not profit from these efforts but still requires trust anchors for bootstrapping and endorsing identity information.

Mapping Domain Names to Real-world Identities The foundational assumption of using TLS certificates for an authentication framework is that domain names can be linked reliably to real-world identities. This assumes that users have the ability and knowledge to connect a domain name to an organization or person and vice versa. Usually, this is the case as users have experience with using domain names on the internet and as domain names are constructed to be human-friendly, for example, by consisting of the company name.

One threat to this approach is *typosquatting*, the intentional registration of slight misspelling of well-known domain names [25]. While these domains are often used to display advertisements on the web [20], they pose a risk to our system. An attacker might use a typosquatting domain and trick users into using their similar domain or count on users accidentally misspelling a domain. However, we deem the chance of mistyping or misreading an Ethereum address higher and the use of domain names as identifying information more reliable.

4 Related Work

In this section, we introduce previous work and ongoing efforts with goals or approaches similar to ours. In section 4.1 we briefly describe several proposals that aim to improve certain properties of PKIs by relying on blockchain technology. We discuss the Ethereum Name Service in section 4.2.

4.1 Blockchain-based PKI Solutions

There exist numerous proposals to integrate blockchains and existing PKI infrastructure. However, the focus of these approaches is not to provide identity solutions for blockchain applications but to leverage the blockchain for improving the properties of (the TLS) PKI. These works are nevertheless relevant as migrating part of the PKI on-chain for Internet purposes has the side effect of using the information for on-chain authentication as information is readily available. Giving an overview of all research that has been done in this field is out of the scope of this paper, so we focus on approaches that target Ethereum or Ethereum-like blockchains and include CAs for issuing certificates. Various other approaches [2, 3, 10, 12, 22, 24, 26] do not include CAs in their design and introduce web-of-trust like solutions instead, which means the incompatibility with existing protocols does not solve the inherent bootstrapping problem, or they rely on newly designed blockchains.

CBPKI [14] is a proposal for a cloud blockchain-based public key infrastructure where stateless CAs residing in the cloud are combined with certificate information stored on a blockchain. The approach does not fit our requirements as only the certificate hash is stored on-chain, but not relevant information such as the subject name or the public key, and as it relies on CAs adapting to it and issuing a new type of certificate. CertChain [6], a decentralized and tamper-proof tool for auditing certificates, does not meet our requirements as it is built on a new certificate format, an adapted implementation of Ethereum, and a new type of CAs that also act as miners in the blockchain network. Instant Karma PKI (IKP) [17] is a smart contract-based incentivization platform aiming to prevent fraudulent issuing of TLS certificates: Clients can define policies concerning certificates issued for their domain and CAs can sell insurance against misbehavior. IKP focuses strongly on improving the security of the TLS ecosystem, but does not align with our goals as certificates are not presented to the blockchain unless they are fraudulent and CAs have to take significant action to make the system work. A blockchain-based PKI management framework is presented in [27]. CAs create smart contracts corresponding to store information about the issuance and revocation of certificates. When a verifier receives a certificate, they refer to the smart contract and verify that the hash is contained, that the certificate is not revoked, and that the chain of trust is valid. Just as the approaches before, this proposal relies on proactive CAs. Additionally, a new certificate format is required and only the certificate hash is stored on-chain, which is not sufficient for an on-chain authentication framework. Kubilay et al. introduce CertLedger, a PKI system with the intention of shifting trust from CAs to the blockchain and providing certificate and revocation transparency [15]. CertLedger manages the validation, storing and revocation of certificates. Clients do not validate certificates or maintain their own root store any longer, they simply refer to CertLedger for certificate-related information. In addition, CertLedger provides a transparent revocation system and allows owners of certificates – not just the issuers – to revoke them. While this proposal fulfills many of our goals, it does not allow open participation: The set of trusted CAs is defined by CertLedger board and all validation decisions are made depending on it. This means that (i) the CertLedger board needs to be fully trusted by clients, (ii) clients cannot distrust individual CAs, and (iii) clients cannot add root certificates for specific applications.

4.2 Ethereum Name Service

Ethereum Name Service (ENS) was launched in 2017 and aims to provide a decentralized way to address blockchain resources in a human-friendly way [13] by resolving human-readable names to Ethereum addresses. ENS is curated by the Ethereum Foundation and is described in three Ethereum Improvement Proposals: EIP-137, EIP-162, and EIP-181 [4]. ENS names are dot-separated hierarchical names called domains; currently, the only supported top-level domain (TLD) is ".eth". TLDs are owned by smart contracts called registrars. The owner of a domain can create subdomains and transfer the ownership of the subdomains to other parties.

The ENS architecture consists of two central components: Registries and resolvers. As ".eth" is currently the only supported TLD, there exists one registry. The ".eth".registry is currently controlled by a 4-of-7 multi-sig. It is planned to transfer control to a decentralized account in the future [4]. A registry contains a list of all its subdomains and their respective owners, resolvers, and cache expiration. All Ethereum accounts that support the relevant standards can be the owner of a domain. Resolvers are responsible for translating the domain to an actual Ethereum address.

Once ENS is established, it is a cost-efficient and decentralized system providing human-readable identities to Ethereum addresses. One problem, however, remains: Domain ownership can be acquired through auctions and the highest bidder wins. This means that ENS domain names cannot be intuitively mapped to real-world identities. Furthermore, there is no judicial system in place which would allow the seizing of individual domains, for example, in the case of copyright disputes.

5 Conclusion and Future Work

In this work, we present the conceptual idea, design, and evaluation of a TLS certificate-based authentication framework for Ethereum. In our framework, identities can be asserted and verified based on TLS certificates that are submitted to and validated by a central database. Identity owners that want to link their identity to an Ethereum account can create endorsements. An endorsement links information about the account address and the domain name, and contains a signature that was created with the certificate's private key and confirms the identity binding. Subsequently, users can obtain this endorsement to authenticate Ethereum accounts they aim to interact with.

The great strength of our system is that it overcomes the bootstrapping problem: Any identity owner can submit their certificate and endorsement without depending on other stakeholders. Under the assumption that certificate authorities are trusted, we can leverage a massive amount of verifiable/verified identity information that is readily available. However, we also acknowledge that our system comes with drawbacks: The TLS system is considered fragmented and not secure enough by some researchers, our system enables authentication only for certificate owners, the on-chain validation of TLS certificates is costly, and storing certificate information increases the size of the Ethereum blockchain. However, we believe that solutions or mitigations can be found to lower the negative impact of these drawbacks. Overall, our framework serves as a pragmatic and feasible approach to establish a system for the identity assertion and verification on Ethereum in a timely manner.

One main goal of future work should be to investigate whether a TLS certificate-based authentication framework can be used in combination with an identity management system or naming service developed specifically for Ethereum. A combination of the approaches could utilize the strengths of both: The certificate-based approach can boost the bootstrapping phase of the system. The information acquired in the bootstrapping phase can then be used to populate the system with further, certificate-independent information. The aim is to make the system gradually independent from the TLS ecosystem, thereby improving the security of the framework.

References

1. Akhawe, D., Amann, B., Vallentin, M., Sommer, R.: Here's my cert, so trust me, maybe? Understanding TLS errors on the web. In: Proceedings of the 22nd international conference on World Wide Web. pp. 59–70 (2013)
2. Al-Bassam, M.: SCPKI: A smart contract-based PKI and identity system. In: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts. pp. 35–40 (2017)
3. Ali, M., Nelson, J., Shea, R., Freedman, M.J.: Blockstack: A global naming and storage system secured by blockchains. In: USENIX Annual Technical Conference. pp. 181–194 (2016)
4. Antonopoulos, A.M., Wood, G.: Mastering Ethereum: Building Smart Contracts and DApps. O'Reilly Media, Beijing, first edit edn. (2018)
5. Author Citation: Prototypical Implementation (2020), `https://github.com/`
6. Chen, J., Yao, S., Yuan, Q., He, K., Ji, S., Du, R.: CertChain: Public and Efficient Certificate Audit Based on Blockchain for TLS Connections. In: Proceedings - IEEE INFOCOM (10 2018)
7. Chen, Y., Su, Z.: Guided differential testing of certificate validation in SSL/TLS implementations. In: 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2015 - Proceedings (8 2015)
8. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. Tech. rep. (2008)
9. Durumeric, Z., Adrian, D., Mirian, A., Bailey, M., Halderman, J.A.: A Search Engine Backed by {I}nternet-Wide Scanning. In: 22nd {ACM} Conference on Computer and Communications Security (2015)

10. Fromknecht, C., Velicanu, D.: A Decentralized Public Key Infrastructure with Identity Retention. Cryptology ePrint Archive (2014)
11. Google: HTTPS encryption on the web (2020), `https://transparencyreport.google.com/https/overview`
12. Hammi, M.T., Bellot, P., Serhrouchni, A.: BCTrust: A decentralized authentication blockchain-based mechanism. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC). pp. 1–6. IEEE (2018)
13. Johnson, N., Lau, J., Eigenmann, D., Millegan, B.: Ethereum Name Service (2020), `https://github.com/ensdomains`
14. Khieu, B., Moh, M.: CBPKI: Cloud blockchain-based public key infrastructure. In: ACMSE 2019 - Proceedings of the 2019 ACM Southeast Conference. pp. 58–63 (4 2019)
15. Kubilay, M.Y., Kiraz, M.S., Mantar, H.A.: Certledger: A new PKI model with certificate transparency based on blockchain. Computers & Security **85**, 333–352 (2019)
16. Laurie, B., Langley, A., Kasper, E.: Certificate Transparency. Tech. Rep. 6962, RFC Editor (2013)
17. Matsumoto, S., Reischuk, R.M.: IKP: Turning a PKI Around with Decentralized Automated Incentives. In: Proceedings - IEEE Symposium on Security and Privacy. pp. 410–426. IEEE (5 2017), `http://ieeexplore.ieee.org/document/7958590/`
18. Meyer, C., Schwenk, J.: SoK: Lessons learned from SSL/TLS attacks. In: International Workshop on Information Security Applications. pp. 189–209. Springer (2013)
19. Mockapetris, P., Dunlap, K.J.: Development of the domain name system, vol. 18. ACM (1988)
20. Moore, T., Edelman, B.: Measuring the perpetrators and funders of typosquatting. In: Lecture Notes in Computer Science. vol. 6052 LNCS, pp. 175–191. Springer, Berlin, Heidelberg (2010)
21. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press (2016)
22. Patsonakis, C., Samari, K., Roussopoulos, M., Kiayias, A.: Towards a smart contract-based, decentralized, public-key infrastructure. In: International Conference on Cryptology and Network Security. pp. 299–321. Springer (2017)
23. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. Tech. Rep. 8446, RFC Editor (2018)
24. Singla, A., Bertino, E.: Blockchain-Based PKI solutions for IoT. In: Proceedings - 4th IEEE International Conference on Collaboration and Internet Computing, CIC 2018. pp. 9–15. Institute of Electrical and Electronics Engineers Inc. (11 2018)
25. Spaulding, J., Upadhyaya, S., Mohaisen, A.: The landscape of domain name typosquatting: Techniques and countermeasures. In: Proceedings - 2016 11th International Conference on Availability, Reliability and Security, ARES 2016. pp. 284–289. Institute of Electrical and Electronics Engineers Inc. (12 2016)
26. Wang, Z., Lin, J., Cai, Q., Wang, Q., Zha, D., Jing, J.: Blockchain-based certificate transparency and revocation transparency. IEEE Transactions on Dependable and Secure Computing (2020)
27. Yakubov, A., Shbair, W.M., Wallbom, A., Sanda, D., State, R.: A blockchain-based PKI management framework. In: NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium (2018)