Turning trust around: Smart Contract-assisted Public Key Infrastructure

Abu Shohel Ahmed
Department of Computer Science
Aalto University, Finland
Email: abu.ahmed@aalto.fi

Tuomas Aura
Department of Computer Science
Aalto University, Finland
Email: tuomas.aura@aalto.fi

***Abstract*—In past, several Certificate Authority (CA) compromise and subsequent mis-issue of certificate raise the importance of certificate transparency and dynamic trust management for certificates. Certificate Transparency (CT) provides transparency for issued certificates, thus enabling corrective measure for a mis-issued certificate by a CA. However, CT and existing mechanisms cannot convey the dynamic trust state for a certificate. To address this weakness, we propose Smart Contract-assisted PKI (SCP) - a smart contract based PKI extension - to manage dynamic trust network for PKI. SCP enables distributed trust in PKI, provides a protocol for managing dynamic trust, assures trust state of a certificate, and provides a better trust experience for end-users.**

***Index Terms*—PKI; Blockchain; Smart Contract; Policy; Decentralized Trust**

I. Introduction

Public Key Infrastructure (PKI) is a trusted third party that binds the name and meta-data with the public key of an entity. Web PKI (PKI and Web PKI are used interchangeably in this paper) ecosystem consists of 1) Certificate authority - responsible for issuing and managing certificates 2) Domain owners - owner of a certificate 3) Validators - applications used by end users to access a domain 4) End-users - interacts with the domain. These four entities form the PKI trust ecosystem. Additionally, auditors perfom scheduled audits to vouch the trustworthiness of a CA. In PKI, the trust provided by a certificate is one way and static i.e., validators and end-users trust any certificate issued by a trusted CA (once believed!).

In the past, several trust non-conformance by public CAs resulted in lack of trust in the PKI ecosystem. For example, DigiNotar (Dutch CA) [1] was compromised by hackers to issue several fake SSL certificates. The prevention and response to such scenarios are harder as removing a trusted CA certificate from the truststore of a validator results in breaking trust for thousands of domains, even for the valid ones. On another development, recently, we observe that application providers such as browser vendors are playing a critical role in managing trust in the PKI ecosystem. The recent dispute between Google and Symantec [2] shows how application providers are enforcing dynamic trust constraints for CAs and their issued certificates. For this instance, a set of constraints is placed in the truststore of Chrome browser to gradually decrease trust for Symantec CA certificates. This reveals several weaknesses of the existing PKI ecosystem: 1) manual auditing (CA/B audit) is not enough for discovering weaknesses of a CA 2) trust reversal is harder, e.g, removal of a popular root certificate from the trust store would break trust for many existing domains 3) domain owner (aka. certificate owner) is not aware of the trust level of an issued certificate i.e., what this certificate is good for and 4) no defined protocol to manage dynamic trust realtionship among application providers, CAs, domain owners and End-users.

We propose smart contract-assisted Public Key Infrastructure (SCP) protocol for managing dynamic trust in the PKI ecosystem. In SCP, application providers, domain owners, and CAs manages trust policies using smart contracts. From these trust policies, a certificate trust policy (CTP) contract is formed before issuing a certificate. The CTP contract acts as an autonomous agent for managing dynamic trust for a certificate. In SCP, certificates are issued under a CTP contract. This enables a domain owner to verify the current trust state of a certificate from the binded CTP contract. This paper has three main contributions.

1) Section III introduces SCP, a dynamic and transparent trust management protocol to set trust state for a certificate and convey this information to all parties. The smart contract approach enables transparent verification of trust using the bottom up approach based on who actually trust this certificate rather than the blind trust towards a CA.
2) Section IV and V model the SCP protocol and propose key characteristics of the system.
3) Section VI and VII evaluate the SCP from implementation and design perspective. An smart contract on top of Ethereum [3] is implemented for prototype evaluation.

II. Background

A. Trust in PKI

PKI provides a centralized and hierarchical trust chain from the issuer i.e., CA to the issued certificates. Figure 1 shows trust relationship in the Web PKI. According to Figure 1, A CA is a trusted entity to provide a certificate for a domain owner. The received certificate is used by the domain server to prove the ownership of a domain. Browser vendors/application providers set trusted CA certificates in a truststore based on the report from the Auditor and using additional proprietary ruleset. The truststore defines trust relationship between a CA

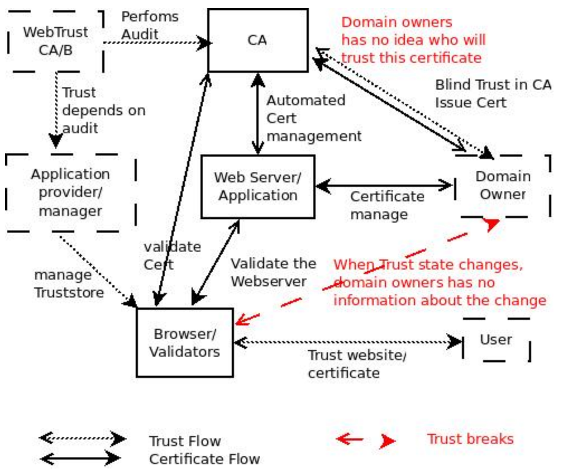
WebTrust CA/B
Perfoms Audit
CA
Domain owners has no idea who will trust this certificate
Blind Trust in CA Issue Cert
Trust depends on audit
Automated Cert management
Application provider/ manager
Web Server/ Application
Certificate manage
Domain Owner
validate Cert
Validate the Webserver
When Trust state changes, domain owners has no information about the change
manage Truststore
Browser/ Validators
User
Trust website/ certificate
Trust Flow
Certificate Flow
Trust breaks

Fig. 1. CA Trust Relationship

and a validator. End-user trusts a domain based on the trust representation of the domain certificate by the validiators e.g., browsers. Additionally, operations of the CA is audited by a third-party auditor using a set of guidelines (e.g., CA/B forum [4] guidelines). There is an assumed trust in the auditor's report by the application provider.

The trust relationship in Figure 1 works for a relatively static environment. However, it fails when trust state changes in a dynamic environment as shown using red color in Figure 1.

Certificate Transparency (CT): CT [5] logs issued certificates in a public log thus allowing scrutiny by domain owners, CAs, and auditors. CT makes hard for a compromised CA to issue forge certificate without being visible to the domain owners. However, CT does not deliver trust state of a certificate i.e., what this valid certificate is good for ?

B. Review of existing PKI trust improvement proposals

Perspective [6] uses a set of notary hosts observing the public key of a certificate via multiple path to detect an attack. However, Perspective fails to address dynamic trust state of a certificate from the viewpoint of an application provider.

Several earlier decentralized PKI proposals address the trusted third party weakness of a CA. **BlockStack [7]** binds a key to a name using a global append only log. It decentralizes the centralized domain name system. **SCPKI [8]** is a smart contract based PKI in which certificate verification is performed using web-of-trust approach. SCPKI smart contract does not comply with the existing PKI protocols and similar to PGP, bootstrapping trust for a new identity is hard. **BIX-Certificate [9]** proposes an equivalent of X.509 certificate implemented using a certificate public ledger. Similar to SCPKI [8], BIX suffers from compatability with existing PKI protocols.

Several earlier papers addresses dynamic trust management aspect of PKI eco-system. **IKP [10]** proposes a competitive and complementary solution for certificate transparency using blockchain. IKP relies on a smart contract between the CA and a Domain owner to manage dynamic trust. IKP has several weaknesses: 1) It fails to take into account the trust requirement of applications (or application provider), 2) the protocol is not suitable for trust delivery among multiple parties. **Certificate Limitation Policy [11] (CLP)** delivers a cryptographic protected trust policy for a certificate from the application provider to the application. CLP addresses only the format for a trust policy. **Policert [12]** allows domain owners to set policies for their certificates, thus browsers can use the certificate according to the defined domain policy during a TLS handshake. Policert does not handle trust view from application providers to others.

C. Truststore

A Truststore contains trusted CA certificates and policies used by applications to trust a certificate issued by a CA. For most cases, truststore management process is proprietary. For example, **Mozilla Trust Store** requires a CA to pass the WebTrust audit and additional criterias to include the CA certificate. Additionally, the truststore policy defines the purpose and usage of the CA certificate. For example, French and Turkish national CAs are restricted to issue certificates only for certain domains. This shows several weaknesses: 1) trust information is not propagated in a standardized way from the application truststore to other actors, e.g., to CA and domain owners, 2) the rule set for truststore management is proprietary, e.g., format and constraint rules.

D. Smart Contract and Blockchain

A smart contract is an agreement with automatable and enforceable policies [13]. A smart contract has several properties: 1) the smart contract code should be successfully run within a reasonable time, 2) the enforcement mechanism can be traditional with dispute resolution methods such as binding or non-binding arbitration logic in place, and 3) there can be non-traditional enforcement without dispute resolution, for example, to enforce actions of smart contract at the network level. The popularity of the smart contract is driven by its decentralized nature such as running on top of blockchain. A blockchain provides transparency, trust, and reliability of records over time [14] in a decentralized way.

III. Problem Definition and Objectives

Table I states three most critical PKI weaknesses [15] that are used as a problem definition for this paper. This paper proposes a dynamic trust management protocol to mitigate these weaknesses. The aims are:

1) A defined protocol for dynamic trust management among CAs, domain owners, and application providers
2) Use of smart contract to define trust behavior and trust agreement among parties
3) Enable trust verification of a certificate in a transparent way

TABLE I
EXISTING PKI WEAKNESSES

Weakness	Status
Blind trust in CA and Certificates	CA is assumed to be trusted but the question is by whom? How do we define trust level of a CA or compare trust level of a CA from one to another? **In a dynamic trust environment, existing slow auditing process to set trust level of a CA fails to meet the security requirements for browser vendors, domain owners, and end-users [2]**. Domain owners and end users have the right and need to know the current trust level of a CA as seen by other actors in the PKI ecosystem.
Malicious and non-conformant CA	When a trusted CA becomes malicious, validators revoke the CA certificate from their truststore. This results in lack of trust for already issued valid certificates. Often, trust for a compromised CA certificate is reduced gradually in a proprietary process by validators. Domain owners (which have certificates issued by a malicious CA) remain dark regarding this change of trust. **Current mechanism lacks transparency and automation for delivering trust state information when a CA becomes malicious**.
Non-conformant certificate	A valid certificate but not conforming to some validators parameters (e.g., validity period, basic constraint) is a non-conformant certificate. This results in a fragmented trust experience for a domain by end users. **PKI (including CT) fails to deliver a validator's trust constraint to a domain owner or domain viewer**.

IV. Architecture and Protocol Overview

This section provides an architectural overview and protocol modeling of SCP. For the architectural representation, first, we introduce the components of SCP along with their functionality. Figure 2 shows the architecture of SCP along with three active participants. The three active participants are application provider, domain owner, and certificate authority.

Application provider (A): An application provider manages truststore in an application. In SCP, application provider is responsible for managing the application trust policy, i.e., a policy for certificates trusted by the application.

Domain Owner (D): A domain owner is the user and owner of a domain certificate. In SCP, the domain owner defines trust requirements for a domain certificate.

Certificate Authority (CA): A CA issues and manages lifecycle of a certificate. In SCP, a CA is responsible for managing the CA trust policy. A CA also contains a signing server. In SCP, the CA signing server ($Sig(cert)$) binds address of a certificate trust policy (CTP) with the issued certificate.

The core of SCP is two smart contracts. First, `Policy contract`(B_p) for registerting and updating trust policies of application providers, domain owners and CAs. Second, `Certificate Trust Contract` (B_c) makes a CTP for a certificate using the trust policy of CAs, application providers and domain owners.

A. Security model

We provide a brief overview of the security model for SCP. The assumptions are the following:

1) Contract transactions are transparent to all and contract source code is published for verification. We assume smart contract acts honestly.

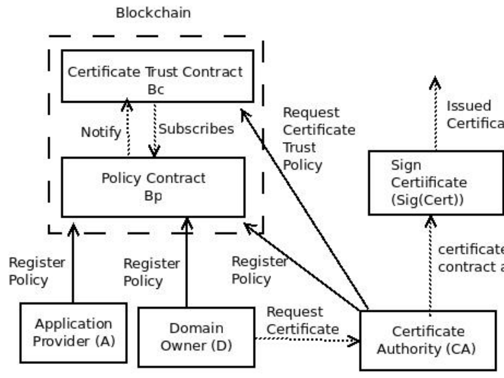

Fig. 2. Architecture of SCP along with participating actors

2) Blockchain communication to the external world using client, e.g., javascript client, and inter-communication from one contract to another are authenticated. Only owner and authorized parties can modify the storage of the smart contract. All transactions and messages are integrity protected without confidentiality.
3) We assume the data entered from the external world to the smart contract can be trusted.

B. SCP Protocl

Figure 3 models the core functions of the SCP protocol. The protocol is divided into two part. The first part, `RegisterTrustPolicies` uses the `Policy contract`(B_p) to register trust policies for CA, Application provider, and domain owners. The second part, `MakeCertificateTrustPolicy` uses both the `Policy contract`(B_p) and the `Certificate trust contract`(B_c) to form a certificate trust policy Id_c for a to be issued certificate.

1) RegisterTrustPolicies:

- Step 2: The `Policy contract`(B_p) accepts a request from a `CA` to register a `CA` trust policy Id_{ca}. The request includes a `CA` certificate `R` and the allowed usage of the `CA` certificate `U`.
- Step 3: An application provider `A` reads the `CA` trust policy Id_{ca} from the B_p contract.
- Step 4: The B_p contract accepts a request from an application provider `A` to register an application trust policy Id_a which binds to one or more existing `CA` trust policies Id_{ca}. At registration, an application trust policy has no `S[]` certificate trust policy subscribers.
- Step 5: A domain owner `D` registers a domain trust policy Id_d using the B_p contract. The domain trust policy includes the trust requirement and intended usage T_d, U_d of the domain.

2) MakeCertificateTrustPolicy:

Fig. 3. SCP Protocol

```
1: procedure RegisterTrustPolicies
2:     $CA \rightarrow B_{\mathrm{p}} : Id_{\mathrm{ca}}, P_{\mathrm{ca}}(\ TrustedCert\ R, Usage\ U)$
3:     $B_{\mathrm{p}} \rightarrow A : Id_{\mathrm{ca}}, P_{\mathrm{ca}}(\ TrustedCert\ R, Usage\ U)$
4:     $A \rightarrow B_{\mathrm{p}} : Id_{\mathrm{a}}, Id_{\mathrm{ca}}, P_{\mathrm{a}}(\ Trust\ T_{\mathrm{a\_ca}}, TrustedCert\ R, Usage\ U), S[]$
5:     $D \rightarrow B_{\mathrm{p}} : Id_{\mathrm{d}}, P_{\mathrm{d}}(\ Trust\ required\ T_{\mathrm{d}}, Usage\ U_{\mathrm{d}})$
6: procedure MakeCertificateTrustPolicy
7:     $D \rightarrow CA : Id_{\mathrm{d}}, P_{\mathrm{d}}(Trust\ required\ T_{\mathrm{d}}, Usage\ U_{\mathrm{d}})$
8:     $CA \rightarrow B_{\mathrm{c}} : Id_{\mathrm{ca}}, Id_{\mathrm{d}}, P_{\mathrm{d}}(T_{\mathrm{d}}, U_{\mathrm{d}})$
9:     $B_{\mathrm{c}} \rightarrow B_{\mathrm{p}} : Id_{\mathrm{ca}}$
10:    $B_{\mathrm{p}} \rightarrow B_{\mathrm{c}} : Id_{\mathrm{a}}, P_{\mathrm{a}}(Trust\ T_{\mathrm{a\_ca}}, TrustedCert\ R, Usage\ U)\ when\ \forall(Id_{\mathrm{a}}) \in B_{\mathrm{p}}\ such\ that\ Id_{\mathrm{a}} \in [Id_{\mathrm{a}}, Id_{\mathrm{ca}}]$
11:    $B_{\mathrm{c}} : Id_{\mathrm{c}}, Id_{\mathrm{d}}, Id_{\mathrm{a}}, P_{\mathrm{a}}(Trust\ T_{\mathrm{a\_ca}}, TrustedCert\ R, Usage\ U)\ when\ U == U_{\mathrm{d}}, T_{\mathrm{a\_ca}} == T_{\mathrm{d}}$
12:    $B_{\mathrm{c}} \rightarrow B_{\mathrm{p}} : Id_{\mathrm{c}}, (Id_{\mathrm{a}}, P_{\mathrm{a}})$
13:    $B_{\mathrm{p}} : Id_{\mathrm{a}}, S[Id_{\mathrm{c}}]$
14:    $B_{\mathrm{c}} \rightarrow CA : Id_{\mathrm{c}}$
15:    $CA \rightarrow D : Id_{\mathrm{c}}$
```

- Step 7: A domain owner requests a CA to make a certificate trust policy against a published domain trust policy Id_d.
- Step 8: The CA selects an appropriate CA trust policy Id_{ca} based on the received domain trust policy Id_d at Step 7. The certificate trust contract B_c accepts a request from the CA with the CA trust policy Id_{ca} and domains trust policy Id_d.
- Step 9: The B_p contract accepts a request from the B_c contract to find all existing application trust policies Id_a linked to the CA trust policy Id_{ca}. The linking occurs during the application trust policy registration (step no 4) described in procedure REGISTERTRUSTPOLICIES.
- Step 10: The B_p contract retuns all application trust polices Id_a previously linked to the CA trust policy Id_{ca}.
- Step 11: The B_c contract evaluates the trust requirement of the domain owner T_d against the application defined trust T_{a_ca} for a CA trust policy Id_{ca}. It also evaluates the certificate usage requirements from the domain against approved usage by an application. If T_{a_ca} satisfies T_d, the B_c contract creates a certificate trust policy Id_c by linking Id_{ca}, Id_a and id_d.
- Step 12: The B_c contract sends the newly created certificate trust policy Id_c and linked application trust policies Id_a to the B_p contract.
- Step 13: The B_p contract stores the certificate trust policy Id_c as a subscriber S[Id_c] for all linked application trust policies Id_a. This enables the application trust policy Id_a to act as a publisher for the certificate trust policy Id_c.
- Step 14: The B_c contract sends the address of the newly created certificate trust policy Id_c to the CA.
- Step 15: The retrun of certificate trust policy Id_c to the domain owner D from the CA indicates a successful establishment of a certificate trust policy.

Fig. 4. Modify Application Trust Policy changes certificate trust policy

```
1: procedure TriggerPolicyChange
2:     $A \rightarrow B_{\mathrm{p}} : Id_{\mathrm{a}}, Id_{\mathrm{ca}}, P_{\mathrm{a^1}}(\ Trust\ T_{\mathrm{a^1\_ca}}, P_{\mathrm{ca}})$
3:     $B_{\mathrm{p}} \rightarrow B_{\mathrm{c}} : Id_{\mathrm{c}}, Id_{\mathrm{a}}, P_{\mathrm{a^1}}(\ Trust\ T_{\mathrm{a^1\_ca}}, P_{\mathrm{ca}})\ when\ \forall Id_{\mathrm{c}} \in [Id_{\mathrm{a}}, S[Id_{\mathrm{c}}]]$
```

3) Modify application trust policy by an Application Provider: SCP supports dynamic trust state of a certificate trust policy Id_c by allowing an application provider to modify their trust policy for a CA. Figure 4 shows modification of the certificate trust policy by an application provider.

- Step 2: The Policy contract B_p accepts a modification request of application trust policy (Id_a, Id_{ca}, P_{a^1}) from the existing owner of the policy. Based on the request, the B_p contract updates the application trust policy.
- Step 3: The Policy contract B_p sends the modifed application trust policy Id_a, P_{a^1} to all subscribers of Id_a. Upon reception of the published message, the B_c contract updates the certificate trust policies Id_c using the updated application trust policy information.

C. Policies and Contracts

Trust policies are registered using the Policy contract and are readable by any party and updatable by the owner of the trust policy. A party registers a trust policy in the blockchain by performing a transaction at the Policy contract. Below, we provide a short description of each policy type with examples.

CA trust policy: A CA trust policy includes the CA stated trust obligation for a CA certificate. The CA makes a transaction at the SCP contract (Step 2 in Figure 3) to register a CA trust policy. The CA trust policy can include certificate policy statement (CPS) ID for a CA certificate and optionally other conditions such as the validity of the policy.

Application trust policy: An application provider reads a CA trust policy from the SCP contract, evaluates the CA trust policy, and request registering the application trust policy against a CA trust policy (Step 3, and 4 in Figure 3). An application trust policy includes trust decisions such as validity, and level of trust placed for a CA trust policy by an application provider.

Domain trust policy: A domain trust policy provides trust requirements for a domain certificate (Step 5 in Figure 3). For example, the domain certificate should be issued by CA1 and CA2, the issued certificate should be fully trusted by the Chrome browser.

Certificate Trust Policy (CTP): The core of SCP is a CTP contract formed by binding published application trust policies and CA trust policies with a domain policy using MakeCertificateTrustPolicy procedure of Figure 3. The CTP uses publish/subscribe paradigm, in which the CTP subscribes to application trust policies. This subscription enables the CTP gets updated for changes in application trust policies. The domain owner/auditor can monitor the current state for a domain certificate by monitoring the CTP contract.

Fig. 5. Certificate Issuance

```
1: procedure IssueCertificate
2:     $D \rightarrow CA : Id_{\mathrm{d}}, CSR(Id_{\mathrm{c}})$
3:     $CA : Id_{\mathrm{c}} \in CSR$
4:     $CA \rightarrow B_{\mathrm{c}} : Id_{\mathrm{c}}, Id_{\mathrm{d}}$
5:     $B_{\mathrm{c}} \rightarrow CA : Id_{\mathrm{c}}\ when\ Id_{\mathrm{c}} \in B_{\mathrm{c}} \wedge Id_{\mathrm{d}} \in Id_{\mathrm{c}}$
6:     $CA \rightarrow D : Cert(Id_{\mathrm{c}})_{\mathrm{Sig_{ca}}}$
```

Fig. 6. Certificate Trust Validation

```
1: procedure ValidateCertificate
2:     $V : Id_{\mathrm{c}} \in Cert(Id_{\mathrm{c}})_{\mathrm{Sig_{ca}}}$
3:     $V \rightarrow B_{\mathrm{c}} : Id_{\mathrm{c}}$
4:     $B_{\mathrm{c}} \rightarrow V : Id_{\mathrm{c}}, Id_{\mathrm{d}}, Id_{\mathrm{a}}, P_{\mathrm{a}}\ (Trust\ T_{\mathrm{a\_ca}},\ Usage\ U,\ TrustedCert\ R)$
```

V. External Operations

This section describes the usage aspect of the SCP protocol.

A. Request for certificate issuance under an existing Certificate trust policy

Similar to the existing certificate issuance process, a domain owner generates a certificate signing request (CSR) as a prerequisite to obtain a certificate. The generated CSR includes an existing certificate trust policy as an additional attribute [16] of the CSR. The certificate issuance process follows steps in Figure 5.

- Step 2: A CA accepts a CSR and a domain trust policy Id_d from the domain owner D.
- Step 3: The CA checks that the CSR includes a certificate trust policy Id_c and that the domain owner owns the domain trust policy Id_d.
- Step 4: The certificate trust contract B_c receives a request from the CA with the certificate trust policy Id_c and the domain policy Id_d.
- Step 5: The B_c contract checks the validity of the certificate trust policy Id_c and the domain's relation with the certificate trust policy, i.e., Id_d belongs to Id_c. A successful validation results in the B_c contract sends a success response to the CA.
- Step 6: The CA sigs a certificate including the certificate trust policy Id_c.

In SCP, the certificate issuance process is similar to the existing process with a few exceptions. These exceptions are: 1) The CSR includes a certificate trust policy contract address as an additional attribute 2) Requires an extra verification steps for the issuing CA during the certificate issuance process. However, this has minimal performance impact on the certificate issuance process because certificate trust policy check is a READ operation in the blockchain, thus no transaction delay and cost is associated with this step 3) The issued certificate includes the URL of application binary interface (ABI) defintion of the certificate trust contract, the address of the certificate trust contract and the location of the certificate trust policy. These attributes are included in the certificatePolicies [17] field of the issued X.509 certificate.

B. Trust validation for a certificate issued under a Certificate trust policy

In X.509 v3, a valid trust chain determines trust for a certificate. The validation process also checks certificate constraints such as certificate policies [17]. In SCP, the validation process (V) uses the certificate trust policy location embedded in the certificate to query the certificate trust contract B_c for obtaining the current state of trust. Figure 6 describes certificate trust validation steps using SCP.

C. Contract violation by a party

A valid certificate exists when a certificate trust policy is invalid: An invalid certificate trust policy can be easily identified from the location of a certificate trust policy embedded in the certificate. A domain owner, for example, can monitor the certificate trust policy and request to revoke the certificate when a violation is detected.

Policy trust rules are not followed by a policy owner: If an auditor monitoring both the SCP contract and real-world events finds that a valid certificate with a valid policy is not honored by a policy owner, the auditor can notify this violation to the certifcate trust contract. There can be potential penalty and dispute resolution mechanism in place within the smart contract. Currently, this paper does not address the auditor and penalty aspects.

VI. Prototype Implementation

SCP prototype consists of smart contracts and a javascript (JS) client for interacting with smart contracts. Smart contracts are implemented using Solidity - a high-level language designed to run on the Ethereum virtual machine. We have developed two smart contracts: Policy contract and Certificate trust contract based on the design described in section IV. The first contract consists of functions for registering trust policies of applications, domains and CAs. The second contract includes functions for registration and verification of a certificate trust policy. These two contracts work in a publish/subscribe model in which a certificate trust policy subscribes to published application trust policies. The JS client provides an interface for registering trust policies and creating a certificate trust policy. The JS client can also perform validation of a certificate trust policy and evaluate the current trust state of a certificate.

A. Certificate Trust Evaluation using the SCP prototype

An entity with a valid SCP enabled certificate or in possession of a certification trust policy location can evaluate the trust level (i.e., trusted by others) from the SCP smart contract. Our developed JS client takes as an input the certificate trust policy location from the certificatePolicies extension field of the issued certificate. The client uses this parameter to derive trust state of a certificate from the SCP contract. A sample trust evaluation output using the client is shown in Table II.

TABLE II

TRUST EVALUATION OF A CERTIFICATE

CAPolicyId	Chrome	Mozilla	IoS	Android OS
1.3.6.1.4.1. 311.94.1.1	Full	Full	Marginal Until Jan 2019	Unknown

TABLE III

REQUIRED GAS AND COST IN USD FOR RUNNING SCP PROTOTYPE

Operation	Gas	Price in USD
Deploy Policy contract	1423841	2.37
Register_policy	128430	0.21
Modify_policy (no subscriber)	62665	0.1
Get_policy_status (called by other contract)	2237	0.004
Deploy certificate Trust contract	957989	1.6
Make_certificate_trust_policy	271657	0.452

For trust evaluation, we have borrowed the trust level name from GNUPG [18]. Our definition for the trust level names are: Full = CA trust policy is fully trusted by the application trust policy; Marginal = CA trust policy is partially trusted by the application trust policy; None = Not trusted; Unknown = Application has no trust policy for this CA trust policy.

B. *Prototype evaluation*

The evaluation begins from the cost of running SCP prototype. Proof-of-work consensus system such as Ethereum uses economic incentive to make the system secure and DoS-resistant. Each transaction in the Ethereum blockchain requires a certain amount of Gas depending on the computational and storage used by the transaction. The user need to pay the price for each unit Gas usually measured in ”Gwei”. Table III states Gas consumption for deployment and operations for the SCP. Table III also shows cost in USD for each SCP operations using 2 Gwei per Gas (safe minimum according to [19]) and the current Ether price in [1] the market [2].

From Table III, deploying the Policy contract and Certificate trust contract costs around 2.37 USD and 1.6 USD respectively. The deployment is a one-time cost compared to the transaction cost for operations such as Register_policy, Modify_policy, Get_policy_status, Make_certificate_trust_policy . According to Table III, the transaction cost for the functions of SCP are relatively low. Thus, the cost of the transaction has minimal economic impact for participating actors.

In a smart contract, cost can vary based on the size of the input parameters, e.g., size of the trust policy. Figure 7 shows Gas cost increases linearly with the increase in trust policy size using the SCP prototype. This re-iterates the proof that Ethereum is not suitable for large data storage [3]. We assume a typical trust policy size is around 130 bytes which costs only 239261 Gas. If policy size is larger than this, data can be stored in an external storage such as InterPlenetary File System (IPFS) [20]. For such cases, the link to the data and the integrity hash are only stored in the blockchain.

[1]https://www.coingecko.com/en/price_charts/ethereum/usd

[2]The calculation is based on 1 Gwei equals to 0,000000832 USD

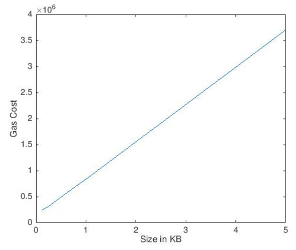

Fig. 7. Cost increases linearly with the increase of policy size

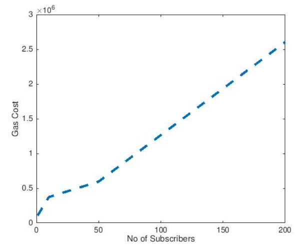

Fig. 8. Cost increases linearly with the increase of subscribers

In the SCP protocol, a certificate trust policy subscribes to one or more application policies. Any modification in a published application trust policy triggers changes to all subscribers of that application trust policy. This impacts on the cost of running SCP for an application provider. Figure 8 shows that the increase in the number of subscribers linearly increases the cost for the application provider to modify an application trust policy. In an experiment with the prototype, 400+ subscribers hit the default[3] maximum of 4.7 million Gas limit for a single transaction in Ethereum. This raises three concerns. First, fix a limit on the number of subscribers for an application trust policy. This can be achieved by placing a logic inside the smart contract to control the number of subscribers. Second, the additional cost incurred by a publisher to publish information to subscribers can be compensated by assigning a subscription cost. Third, and most importantly, the prototype needs design improvement to accomodate the number of subscribers to match real world scenarios.

An important angle of measurement is the time to complete an operation (transaction) in SCP. Transaction completion time depends multiple factors including Gwei offered per Gas, the block size, and the number of blocks added to the blockchain per second. Typically, with 2 Gwei per Gas, it would take 22 minutes to complete the transaction in Ethereum chain. We believe this completion time is tolerable as smart contract transactions are performed during the contractual phase before issuing the actual certificate. The existing certificate registration process in Web PKI takes a couple of minutes for issuing a domain validated (DV) certificate, around one day for issuing an organizational validation (OV) certificate, and one to five days for issuing extended validation (EV) certificate. Thus, SCP does not increase the issuance time for OV and EV certificates. For a strict transaction time critical systems, we can opt for a permissioned chain such as Hyperledger [21] or offer higher Gwei per Gas for each transaction.

VII. EVALUATION

A. *Comparison with other PKI trust improvement proposals*

This section evaluates SCP with other proposals for improving trust in the PKI eco-system. We evaluate the SCP and the existing proposals using the weakness matrix described in Table I. Table IV provides a summary of our evaluation.

[3]Based on default configuration in existing clients

TABLE IV
EVALUATION OF SCP WITH OTHER PROPOSALS

Weaknesses	**CT [5]**	**BlockStack [7]**	**IKP [10]**	**SCP**
Blind trust in CA or TTP (1)	limited	No	No	No
Non-conformant CA information (2)	No	No	limited	Yes
Certificate non-conformance information (3)	No	No	limited	Yes
Portability with existing PKI standards (4)	Yes	No	Yes	Yes

Notes:
(1) Trust in the CA is based on a trusted third party. CT brings transparency to this by publishing the name and key binding in a public log. BlockStack is fully decentralized while IKP tackles blind trust by monitoring domain contract using auditors. SCP adds a new dimension to the certificate trust evaluation, allowing application providers to define dynamic trust state for a CA trust policy.
(2) Non-conformance of a CA occurs when trust in a CA is lost conditionally from the viewpoint of a trusted actor. CT and BlockStack have no defined mechanism to support this until revocation occurs. IKP is limited to express non-conformant CA information by an application provider. SCP delivers notification for change of trust to the participating parties using a certificate trust policy contract. This enables domain owners to take an appropriate measure in trust breakout scenarios.
(3) In web browsing, lock or green bar typically act as a trust symbol for a domain. Currently, the end-user cannot verify the trust level of a certificate even when it fails to load in the browser. For this weakenss, the limitations of CT, BlockStack, and IKP are similar to the non-conformance of a CA. In contrast, SCP offers the end-user or domain viewers the means to validate the trust level for a certificate.
(4) SCP protocol is compatible with the existing X.509 protocols.

B. SCP design evaluation

The core of SCP is a trust contract among the application provider which manages the truststore of an application, the CA which manages certificates, and the domain owner which uses a certificate as a trusted identity for the domain. This is a case of multi-party system where trust is dynamic, e.g., a browser vendor can decide to remove or constrain trust for a CA certificate at any point of time. Smart contract based trust system allows 1) a pre-defined and automated management of trust for each actor and is 2) transparent to everyone. However, blockchain based smart contract has several weaknesses. The transaction time and incurring cost of a transaction act as deterrent factors. Additionally, maintaining a trusted data source is challenging, e.g., if an application provider updates the truststore without updating the application trust policy. This issue can be addressed using Town Crier [22] protocol which feeds data to the smart contract from an external trusted data source. Another alternative is using auditors to monitor the state between the external data source and the SCP contract. This, however, brings the question of economic incentives for the auditor role which requires further investigation.

We, now, focus on several design choices of SCP. First, we look at the usage aspect of SCP. The certificate issuance process uses the existing X.509 v3 protocol to embed SCP related trust vector in the certificate. The certificate verification process uses the embedded trust vector included in the certificate to derive trust for the certificate. Both the certificate issuance and verification process using SCP are compatible with the existing X.509 v3 protocol as SCP only adds additional checks on top of the existing issuance and verification process. Next, from a protocol perspective, SCP suffers from increased Gas cost and limitation on the number of subscribers for an application trust policy due to the publish/subscribe design pattern. An alternate design can use event monitoring for change in an application trust policy rather than subscribing to the application trust policy contract. However, in this approach, certificate trust policy may get out of sync with the current application trust state when multiple notifications arrive at the same time. Thus, further analysis is required before finalizing a design choice.

C. Relations with existings X.509 protocols

In X.509 v3, the trust anchor starts from an independent CA. Each independent CA commits to a trust policy by including certificate policies embedded in the certificate. Auditors audit the trustworthiness of a CA against the published certificate policy. Application software/Clients decide on the level of trust for a certificate by relating it to a certificate policy. The certificate policy [23] extension includes a set of one or more policy identifiers indicating under which terms and conditions a certificate is issued. Existing certificate policy extension or any other extension fails to take into account, how trustworthy a CA or a certificate is from the perspective of others. To address this, SCP uses the existing `certificate policy` field to include and deliver application provider stated trust state for a certificate.

D. Security analysis

We follows STRIDE [24] methodology to derive a threat model for the SCP protocol. The main actors in SCP are 1) Application Provider 2) Domain Owner 3) Certificate Authority 4) End-user and 5) Smart Contract provider. These entities interact with the following assets 1) Trust policies 2) Certificate trust policy and 3) Certificate. Based on these assets and actors, the identified major threats are discussed below. For each threat, we describe the threat, its classification, effects and possible remediation techniques:

An application provider's identity is spoofed by some other actors. This creates a false trust relationship between the CA and the application provider. For example, the identity of a browser vendor used for publishing a trust policy in the SCP contract is spoofed by another entity. This allows the rouge entity to publish an application trust policy in disguise. To remedy this, the application provider's identity should be published in a well-known media (e.g., newspaper, company web site). The same level of remediation is required for the identity of CAs.

Application provider does not follow the trust commitment recorded in the SCP contract. To remedy this, additional tools (e.g., auditors) or protocols are required to validate that a published application trust policy actually matches with the truststore of the application. One possible remediation is an improved smart contract to automatically validate the state of an application truststore against the published trust policy.

Denial of service on the SCP contract. In SCP, the certificate trust policy subscribes to multiple application trust policies to receive notifications of change in the application trust policy. The publisher requires Gas for publishing notification to each subscriber. Thus, an attacker can use multiple subscription to drain Gas from the publisher. By incorporating a subscription fee for each subscriber, this attack can be mitigated.

Several blockchain related attacks are also applicable for the SCP. For example, 51 percent miners node attack can be mounted to de-stabilize the system [25], and loss of private key means loss of controls for the account. These problems are outside the scope of this paper.

VIII. Conclusion

The paper proposes a smart contract-based PKI (SCP) for managing and delivering dynamic trust in the PKI eco-system. This is achieved by using a trust policy contract among the application provider, certificate authority and domain owner. A trust policy contract enables each participating party to stay up to date with the current state of trust for a certificate. The trust state for a certificate is transparent in the blockchain and is verifiable by all. The paper has three main contributions: 1) A model for the SCP protocol, 2) Concretize the SCP protocol with a prototype implementation, and 3) Evaluate SCP from design and implementation perspective. We have evaluated our proposal with certificate transparency (CT), BlockStack, and IKP at which our solution mitigates the identified risks compared to the existing one. Finally, SCP provides the capability to validate trust in a certificate in a transparent and an automated way, thus allowing faster detection of trust misbehavior in the PKI eco-system.

Acknowledgment

This paper has been produced in the context of project SOFIE (Secure Open Federation for Internet Everywhere), which has received funding from EUs Horizon 2020 programme, under grant agreement No. 779984. We also thank Miika Komu, Petri Laari, and Jimmy Kjallman for thoughts and insightful comments.

References

[1] J. Stapleton, "PKI under attack," available at www.issa.org/resource/resmgr/JournalPDFs/PKI_Under_Attack_ISSA0313.pdf.

[2] B. Dev, "Intent to deprecate and remove: Trust in existing symantec-issued certificates," 2017, available at https://groups.google.com/a/chromium.org/forum/#!msg/blink-dev/eUAKwjihhBs/rpxMXjZHCQAJ.

[3] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, pp. 1–32, 2014.

[4] "Baseline requirements for the issuance and management of publicly trusted certificates," available at https://cabforum.org/baseline-requirements-documents/.

[5] Laurie, Ben, A. Langley, and E. Kasper, "Certificate transparency. RFC 6962," 2013.

[6] D. Wendlandt, D. G. Andersen, and A. Perrig, "Perspectives: Improving ssh-style host authentication with multi-path probing," in *USENIX 2008 Annual Technical Conference*, ser. ATC'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 321–334. [Online]. Available: http://dl.acm.org/citation.cfm?id=1404014.1404041

[7] M. Ali, J. C. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A global naming and storage system secured by blockchains." in *USENIX Annual Technical Conference*, 2016, pp. 181–194.

[8] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*. ACM, 2017, pp. 35–40.

[9] S. Muftic, "Bix certificates: Cryptographic tokens for anonymous transactions based on certificates public ledger," *Ledger*, vol. 1, pp. 19–37, 2016.

[10] S. Matsumoto and R. M. Reischuk, "IKP: Turning a PKI around with blockchains." *IACR Cryptology ePrint Archive*, vol. 2016, p. 1018, 2016.

[11] Belyavskiy, *Certificate Limitation Policy*, 2017, available at https://www.ietf.org/id/draft-belyavskiy-certificate-limitation-policy-04.txt.

[12] P. Szalachowski, S. Matsumoto, and A. Perrig, "Policert: Secure and flexible tls certificate management," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 406–417.

[13] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 79–94.

[14] M. Swan, *Blockchain: Blueprint for a new economy.* " O'Reilly Media, Inc.", 2015.

[15] C. Ellison and B. Schneier, "Ten risks of PKI: What you're not being told about public key infrastructure," *Comput Secur J*, vol. 16, no. 1, pp. 1–7, 2000.

[16] Nystrom and K. et al., *RFC2986 PKCS 10: Certification Request Syntax Specification Version 1.7*, 2000.

[17] Housley and R. et al., *RFC5280 Internet x. 509 public key infrastructure certificate and crl profile.*, 2008.

[18] "Validating other keys on your public keyring," available at https://www.gnupg.org/gph/en/manual/x334.html.

[19] "Eth gas station," available at https://ethgasstation.info/.

[20] J. Benet, "IPFS-content addressed, versioned, p2p file system," *arXiv preprint arXiv:1407.3561*, 2014.

[21] C. Cachin, "Architecture of the hyperledger blockchain fabric," in *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, 2016.

[22] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 270–282. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978326

[23] S. C. et al., *RFC3647 Internet x. 509 public key infrastructure Certificate Policy and Certification Practices Framework*, 2003.

[24] "The STRIDE threat model," available at https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx.

[25] M. Bastiaan, "Preventing the 51%-attack: a stochastic analysis of two phase proof of work in bitcoin," in *Availab le at http://referaat. cs. utwente. nl/conference/22/paper/7473/preventingthe-51-attack-astochastic-analysis-of-two-phase-proof-of-work-in-bitcoin. pdf*, 2015.