symmetry

Article

ETHERST: Ethereum-Based Public Key Infrastructure Identity Management with a Reward-and-Punishment Mechanism

Chong-Gee Koa [1,*], Swee-Huay Heng [1,*] and Ji-Jian Chin [2,*]

[1] Faculty of Information Science and Technology, Multimedia University, 75450 Melaka, Malaysia
[2] Faculty of Computing and Informatics, Multimedia University, 63100 Cyberjaya, Malaysia
[*] Correspondence: 1181400347@student.mmu.edu.my (C.-G.K.); shheng@mmu.edu.my (S.-H.H.); jjchin@mmu.edu.my (J.-J.C.)

Abstract: Public Key Infrastructure (PKI) is the fundamental of secure digital communications. It provides a secure means to authenticate identities over the Internet. Symmetric or asymmetric encryption schemes are widely used in identity authentication in any kind of PKI. The conventional PKI has several drawbacks due to the centralized and non-transparent design. Several recent research works utilize blockchain technology to overcome the limitations of conventional implementations of PKI. Blockchain-based PKI integrates blockchain technology with PKI to form a new type of decentralized PKI (DPKI). Several works utilize the currency property in blockchains to implement the reward-and-punishment mechanism. In this paper, we propose a smart contract-based PKI which utilizes the Ethereum smart contract to build a new type of blockchain-based PKI with the reward-and-punishment mechanism using ERC-20 tokens. It has several advantages over previous implementations of similar research that use Ethereum's main currency—Ether.

Keywords: blockchain; PKI; Ethereum

check for updates

Citation: Koa, C.G.; Heng, S.H.; Chin, J.J. ETHERST: Ethereum-Based Public Key Infrastructure Identity Management with a Reward-and-Punishment Mechanism. *Symmetry* **2021**, *13*, 1640. https://doi.org/10.3390/sym13091640

Academic Editor: Lorentz Jäntschi

Received: 28 June 2021
Accepted: 30 August 2021
Published: 6 September 2021

1. Introduction

Public key infrastructure (PKI) is an indispensable infrastructure in the security of digital communications. One of the roles of PKI in digital security is to ensure that the identity of a website that we browse is genuine. The deployment of PKI has encouraged the growth of crucial applications such as e-commerce, online share trading and Internet banking. Implementation through Certificate Authorities (CAs) is the most common approach of the conventional PKI. These centralized PKIs (CPKIs) rely on a hierarchy of trusted CAs. CPKIs have some issues and the most obvious issue is single point-of-failure [1]. On the other hand, decentralized PKIs (DPKIs) were proposed to deal with the issues of centralized PKIs. However, both CPKIs and DPKIs still have several shortcomings [2]. There were some serious flaws found in even well-known CAs during the last few years. For example, the high profile DigiNotar incident final report on the DigiNotar hack showed a total compromise of CA servers, where a total of 531 fraudulent certificates were issued for famous domains such as *.windowsupdate.com, *.mozilla.com and *.google.com [3].

Many studies had proposed to use blockchain technology to build a secure PKI system [4] after the success of blockchain technology as a shared, immutable, public distributed journal. There are several proposed blockchain-based PKIs that implemented a reward-and-punishment mechanism to fill the missing part of conventional PKI. In this paper, we proposed a new blockchain-based PKI with smart contracts and a custom currency to improve the reward-and-punishment mechanism. With this improvement, we believe it will encourage the commercial adoption of the blockchain-based PKI.

1.1. Blockchain Technology

Blockchain technology is considered part of the fourth industrial revolution. Blockchain is the fundamental technology that was invented and used when Bitcoin, a cryptocurrency, was created through the maintenance of immutable distributed ledgers proposed by Satoshi Nakamoto in 2008 [5]. There are many potential applications of blockchain besides its initial usage as an alternative currency. The advantage of blockchain as an encrypted, secured and decentralized public ledger, has been implemented in many areas, such as judiciary, education and finance after it was proposed. Cryptocurrency blockchain transaction flow can be simplified as below: P initiates a transaction to Q through a peer-to-peer blockchain network. In order to identify P and Q uniquely, a pair of public and private keys are used. This transaction will then be broadcast to the blockchain network for the purpose of validation and verification [6].

The ability of blockchain to maintain a consistent view and agreement among the participants (i.e., consensus) is its key feature [4], even though some participants might not be honest [7]. The issue of consensus has been studied by many researchers. However, the use of consensus in blockchain has provided new motivation and stimuli, resulting in novel contributions in the design of blockchain systems. The most popular one, used in Bitcoin, is called Proof of Work (PoW), which requires miners to solve complex computational tasks before doing validation of transactions and then adding them to the blockchain [8]. The miner who manages to solve the task bundles the block to the chain, which is later validated by the other nodes, will be rewarded with newly mined coins together with a small transaction fee. Some research for the consensus mechanism for blockchains have been carried out to improve PoW. Among them is Ouroboros [9], a provably secure Proof-of-Stake (PoS) blockchain protocol. Besides that, the Hyperledger Sawtooth platform [10] developed by Intel, uses the proof-of-elapsed-time (PoET) consensus protocol which relies on SGX technology [11]. Since all the transactions executed in blockchain are transparent, trustworthy and immutable, this technology has many potential applications [12].

1.2. Ethereum, Smart Contract, Ether and ERC-20

Due to the success of Bitcoin and blockchain technology, many research works have been made toward improving blockchain technology. One of the proposals is to add program code into blockchain as a form smart contract running on blockchain. Bitcoin implements its own limited capability scripting language called Script [13]. However, Script is not a Turing-complete language [14] because nodes are required to process Script in order to verify transactions and this causes vulnerabilities where malicious Script transactions could cause nodes fall into state of infinite loop.

Ethereum [15], an open source, public blockchain platform which supports Turing-complete language was launched in 2015 and is still rapidly gaining attention of researchers and developers. In the Ethereum whitepaper, a smart contract is described as "complex applications involving having digital assets being directly controlled by a piece of code implementing arbitrary rules" [16].

Ethereum is not only a blockchain but also a platform or operating system known as the Ethereum Virtual Machine (EVM) that can execute byte-code scripts called Smart Contracts. Smart contracts can be coded in high-level languages i.e., Solidity, which can be compiled into EVM byte-code. Similar to Bitcoin, Ethereum has its own internal currency called Ether. Each smart contract can contain functions that have cost linked to them. The cost is named as gas fee which can be paid with Ether. Besides, each node can also transfer Ether among themselves.

Additionally, Ethereum allows users to create their own cryptocurrencies that run on its blockchain. The cryptocurrencies that are built on Ethereum are named as tokens. ERC-20 token is a type of token that can be created [17]. ERC-20 is not only functioning as a currency, it can also act as a share in the company for points in a loyalty programme.

1.3. Conventional Public Key Infrastructure (PKI)

Public key cryptography uses a pair of public key and private key to encrypt and decrypt data to protect it against unauthorized access. PKI is responsible to manage the public key based on certificates that verify whether a public key is owned by some entities. The following functionalities must be supported by a PKI [4]:

1. Registration of identity with a corresponding public key.
2. Updating the public key of a previously registered identity.
3. Looking up a public key for an identity.
4. Verification of the public key for an identity.
5. Revoking the public key for an identity.

Conventional PKIs can generally be categorized into two categories: Certificate Authorities (CAs) [18] and decentralized networks of P2P certification, often referred to as Webs of Trust (WoT) [18].

1.3.1. Certificate Authorities (CAs)

CAs are trusted authorities who can issue a signed certificate, to validate an entity's ownership of a public key as in Figure 1. A device receives a root certificate from a CA and accepts to save into itself to establish a trust to the CA. When a user browses to a website, such as google.com, via any web browser, the web browser will start with validation of claimed certificate which holds the public key of Google by checking with the issuer of certificate. To improve the running performance, usually web browsers are pre-configured with a list of trusted certificates from some known CAs.

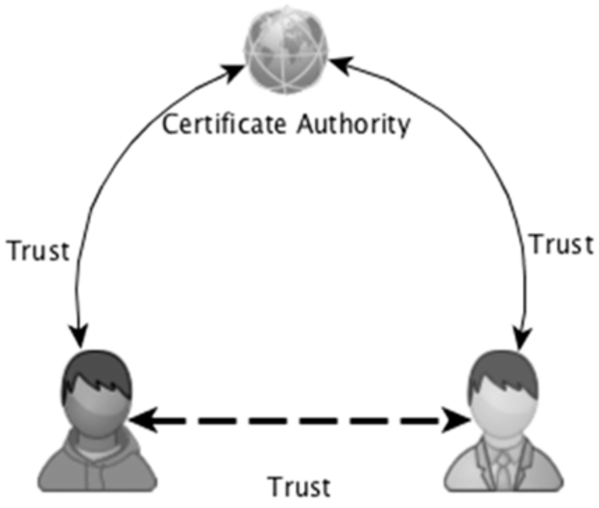

Figure 1. The management of trust by certificate authority between two parties.

1.3.2. PGP Web of Trust (WoT)

The approach of PGP WoT is not centralized because there is no central authority. Instead, all members are able to issue the public keys as introducers. Each member also has the endorsement to establish a trust [19]. Users of the decentralized network establish trust between each other by validating that others have a certificate signed by an entity in whom the validater has previously established trust. In a web of trust, there are no central authorities that manage certificates. Instead, any of the users in the WoT can sign each other's public key, resulting in a scenario where trust is decentralized in WoT. This means any parties in the WoT can issue a certificate, and if a signer is compromised, the signer's key will be revoked and it brings limited impact to the network.

In the PGP WoT model, four levels of trust are supported: no trust, marginal trust, complete trust, and legitimate.

In summary, the four levels of trust can be explained in short as below:

1. No trust: users are not sure whether this public key is valid or not.
2. Marginal trust: this public key may be valid but users cannot be too sure.
3. Complete trust: users are confident that this public key is valid.
4. Legitimate: deemed legitimate by users.

As a way to strengthen the security of WoT, real events named "key signing parties" are held to let the users in a WoT attend and meet each other at the party [20]. Participants at a key signing party are expected to bring along sufficient identity documents to prove their identity. They will also write down public key fingerprints generated by a cryptographic hash function. This fingerprint represents the participant's key and will be changed among participants to be verified. At the end of the party, participants get the public keys corresponding to fingerprints they received and sign them digitally.

1.4. Blockchain-Based PKI

Several Ethereum-based PKI have been proposed by researchers to overcome the open issues in conventional PKI [2]. Starting from CertChain [18], the first attempt in blockchain-based PKI in the year 2014, many researchers had contributed their effort on blockchain-based PKI to improve on conventional PKI. These implementations had been studied and compared by Brunner et al. [21]. They compared the blockchain-based PKI from the aspects of

1. Permission Type
2. Revocation
3. Blockchain Type
4. Certificate Format
5. PKI Type
6. Storage Type
7. Updateable Key
8. Privacy
9. Incentives
10. Evaluation

One of the improvements of blockchain-based PKI is implementing the reward-and-punishment mechanism by utilizing the currency property of blockchain. Among them are the Instant Karma PKI (IKP) [22] proposed by Matsumoto and Reischuk and Internet Web Trust System [23] proposed by Li et al. All of the proposed reward-and-punishment mechanisms use Ether as the currency. It leads to the problem of implementation costs that fluctuates with the Ether market price.

1.5. Our Contributions

In this paper, we introduce ETHERST, a new blockchain-based PKI which makes use of the currency property of blockchain to integrate incentivization and disincentivization. This provides a new method of improvement on the key validation problem that has been studied by Ulrich et al. [20]. A simple incentivization and disincentivization algorithm is developed to control the misbehaviour of participants in this new PKI. The introduction of the ERC-20 token in ETHERST also resolves the issues of previous implementations of reward-and-punishment mechanism that are using Ether by reducing the reliance on Ether which fluctuates in value with market price.

1.6. Organization

The rest of the paper is organized as follows: In Section 2, we analyze the evolution of blockchain-based PKI. In Section 3, we discuss the open issues of the existing blockchain-based PKI reward-and-punishment mechanism. We outline the ETHERST design in Section 4. The simulation settings and implementation detail is presented in Section 5. Finally, in Section 6, we conclude the paper with a discussion on possible future work on the blockchain-based PKI.

2. The Evolution of Blockchain-Based PKI

2.1. Beginning of Blockchain-Based PKI

2.1.1. CertChain

Fromknecht et al. introduced the first full-fledged blockchain-based PKI in 2014 [18]. The authors leveraged the advantages of immutability, transparency and decentralization of blockchain to implement a decentralized PKI with identity retention. Like WoT, a decentralized PKI, as depicted in Figure 2, overcomes the issue of single point of failure in conventional CA PKI.

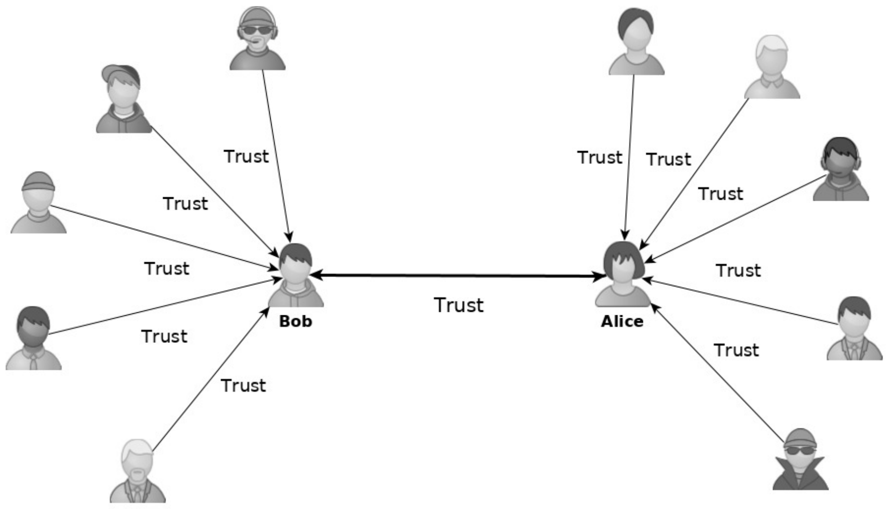

Figure 2. Decentralized of trust management between two parties.

Blockchain-based PKI improves over conventional decentralized PKI like WoT with consistency offering identity retention [18]. Unlike WoT, that trusting a user with a set of other users, Fromknecht et al. implemented a blockchain-based PKI which only required that users believe that many of the members are trustworthy. They also discussed how Certcoin, which is based on Namecoin [24], can efficiently implement each PKI functionality with the help of blockchain and cryptography accumulators. A compact, universal, publicly auditable and strong cryptography accumulator is used to lower the storage size of Certcoin. Fromknecht et al. compared four known accumulator constructions, i.e., the RSA construction [25], the Merkle Hash Tree construction [26], the Bilinear Map construction [27] and the Bloom Filter which technically is not a cryptographic accumulator [28]. The Merkle accumulator was proposed as the accumulator in the implementation of Certcoin. The structure of "a block" of blockchain-based PKI is depicted in Figure 3.

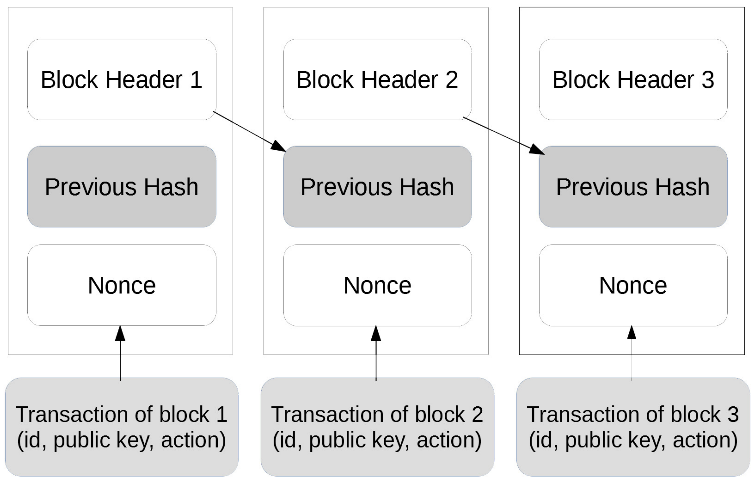
Block Header 1
Previous Hash
Nonce
Block Header 2
Previous Hash
Nonce
Block Header 3
Previous Hash
Nonce
Transaction of block 1 (id, public key, action)
Transaction of block 2 (id, public key, action)
Transaction of block 3 (id, public key, action)

Figure 3. Structure of Blockchain-based PKI.

2.1.2. Blockstack and AuthCoin

While Fromknecht et al. formalized the first blockchain-based PKI, Ali et al. produced another blockchain-based PKI, called Blockstack [29]. Around the same time, Leiding et al. proposed Authcoin [30], an alternative protocol to authenticate with a dynamic challenge and response scheme. The authors presented a process of authentication which starts with private key and public key generation, and their binding. The authors further elaborated a key validation flow which leads to validate and authenticate in AuthCoin. Finally, they focused on the discussion of revocation and expiration of key and signature of AuthCoin. Leiding et al. improved on the validation and authentication processes of Authcoin to combat malicious users compared to CertCoin [18]. They used a custom blockchain to implement the proposed protocol.

2.1.3. Pomcor

Lewison et al. proposed a blockchain-based PKI with custom certificates called Pomcor [31]. The two custom certificate formats, i.e., plain and rich format, were made with some special purpose for this proposed PKI. They comprise meta data, a public key, and asserted data but with no signature. It is different from the standard X.509 public key certificate where a signature is applied to a hash of a one-to-one encoding of the public key, the meta data and the asserted data, using the same ASN.1 DER encoding [32] used in conventional PKI [31]. According to Lewison et al., Pomcor solves the CRL problem of conventional PKIs because it does not require the CRL issuance for certificate revocation [31].

2.1.4. SCPKI

Smart contract-based PKI and Identity System (SCPKI) [33] was proposed by Al-Bassam with the objective to discover the wrong certificate issuance. He built a prototype of PKI that consists of two version of smart contracts. The smart contracts were implemented in Solidity and a command-line Python client. The difference between the two versions is, full version smart contract will keep all the attribute data within the contract itself, while the light version does not. Instead of keeping the data inside smart contract, the light version stores data into InterPlanetary File System (IPFS), a popular storage layer for decentralized application. In light version, with the help of IPFS, the gas fees associated with storage of blockchain is lower compared to full version. He did a gas costs analysis between the full version and the light version and found that the light version has 72% of cost reduction over the full version.

2.1.5. Instant Karma PKI (IKP)

IKP, a blockchain-based PKI by Matsumoto and Reischuk[22], is an automated platform that rewards good CA behaviour through incentives and notifies fraudulent certificates through disincentivization [22]. The IKP has the following properties:

- Public auditability
- Automation
- Incentivization
- Deterrence

They designed domain certificate policy (DCP) and reaction policy (RP) in smart contracts for the purpose of defining and processing misbehaviour automatically. Matsumoto et al. presented the design of IKP, together with a framework for reacting to CA misbehaviour. The authors also presented an analysis of cost on IKP to verify the reward-and-punishment on certificate authority. The IKP prototype has been implemented in Ethereum and finally, an analysis of real-world data from existing CAs has been done to determine the realistic value for RP which used to incentivize or disincentivize when an unauthorized certificate is reported.

2.1.6. Internet Web Trust System

Li et al. proposed an Internet Web Trust System based on smart contracts [23]. They had utilized blockchain to simulate conventional CA-based PKI but added a reward-and-punishment mechanism. The system proposed by Li et al. consists of five parts:

1. Admission mechanism for CA nodes.
2. Certificate chain structure.
3. Conventional certificate management function.
4. Error certificate feedback mechanism.
5. Automatic economic reward and punishment mechanism.

They classified nodes into three types of node in the proposed system. The three types of node are CA, end-user and ordinary nodes [23]. The role of CA nodes is that of a certificate authenticator while end-user nodes are those who request the certificate and submit a query on the certificate. On the other hand, ordinary nodes are permitted to view certificates and to maintain the blockchain [2].

The proposed system is implemented with smart contracts. Aside from conventional PKI functions, they also implemented new functions: question and vote to simulate an auto-response, incentivize and disincentivize mechanism by making use of the currency property of blockchain.

3. Incentivization and Disincentivization in Blockchain-Based PKI

There are several blockchain-based PKIs that implement the incentivization and disincentivization model. The models in IKP [22] and Internet Web Trust System [23] proposed the incentivization and disincentivization models as an improvement specifically on the conventional centralized PKI, which is CA. Both of the models proposed use the default currency in Ethereum—Ether as the medium to incentivize or disincentivize.

However, the success of deployment of the models commercially very much depends on the market value of Ether. Ether is traded on the open market, hence the value can fluctuate with the demand and supply. From the value history, Ether fluctuates unpredictably with the market: it was valued at 1500 USD in early 2018, 120 USD in January 2019 and 2603.17 USD in June 2021.

This can be a show stopper for commercial implementations of the model of those previous research [34]. Thus, we propose ETHERST, a new generation of blockchain PKI that implements incentivization and disincentivization with an ERC-20 token which has the following advantages:

1. Minimize the influence of fluctuations on the market value of Ether.
2. Allows the model owner to determine the value of reward/punish medium.
3. Allow better control over disputes.
4. Better adoption on commercial deployment.

Instead of implementation on centralized PKI, we implement ETHERST on distributed PKI, WoT as an attempt to integrate a reward-and-punishment mechanism to encourage the involvement of nodes in the WoT.

4. Design of ETHERST

ETHERST, blockchain PKI which will incentivize and disincentivize each node of a Web of Trust, was inspired by SCPKI [33] and the research by Li et al. [23]. This new blockchain PKI combines blockchain-based Web of Trust with the reward and punishment mechanism that is provided by the blockchain currency property as an attempt to improve the key validation problem with incentivization and disincentivization. This is the first proposal of integrating Web of Trust with the reward-and-punishment mechanism. Besides, it is also the first blockchain-based PKI that implements its reward-and-punishment mechanism with ERC-20 token.

ETHERST is a system running on Ethereum blockchain. In this research, we produce two models of ETHERST. We start with ETHERST version 1.0 which uses Ether as a means of reward and punishment. We later found that the Ether value is fluctuated with the demand and supply and can change tremendously over time. This caused the ETHERST version 1.0 to be not feasible in the commercial implementation. In ETHERST version 2.0, we introduce ERC-20 token in ETHERST to replace Ether as a mean of rewarding and punishment.

4.1. ETHERST Version 1.0

ETHERST version 1.0 comprises one smart contract, which we call the ETHERST smart contract. Below is the list of entities that exists in the ETHERST smart contract.

1. ETHERST entity
2. Attribute
3. Signature
4. Revocation

Figure 4 shows the Entity Relationship Diagram (ERD) of the ETHERST smart contract.

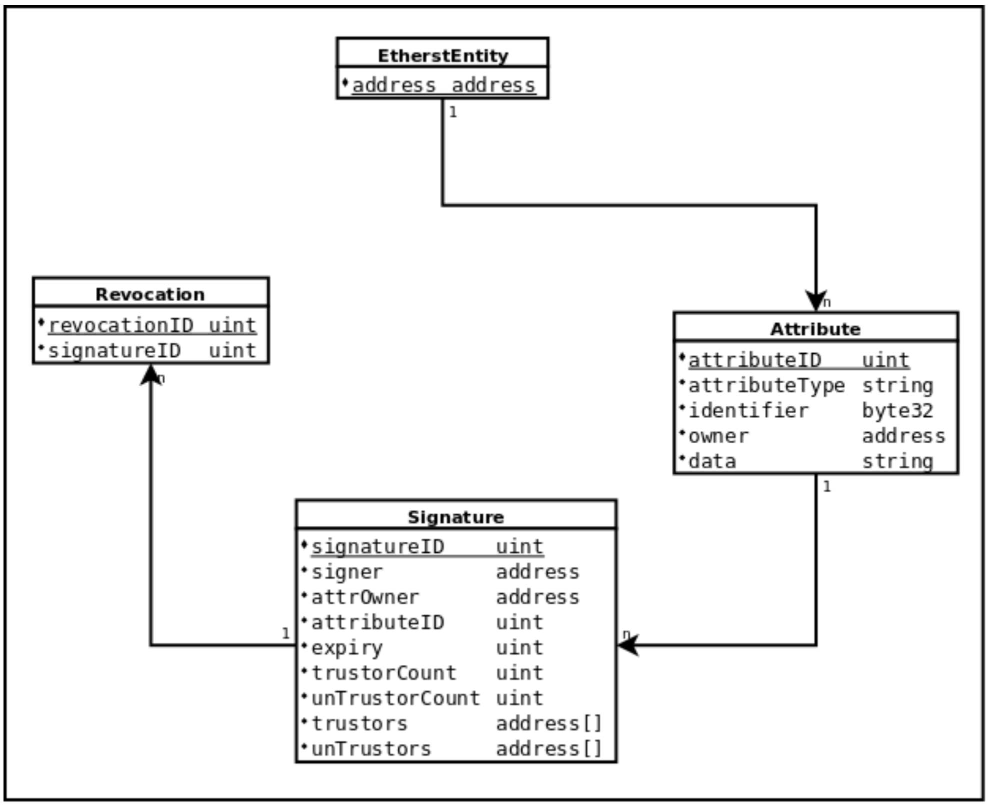

Figure 4. Entity relationship diagram for ETHERST.

The following are functions in the ETHERST smart contract:

1. Create attribute.
2. Sign attribute.
3. Revoke signature.
4. Trust signature.
5. Untrust signature.

In the following subsections, we describe the functions in detail.

4.1.1. Create Attribute

CreateAttribute is a smart contract function that allows a node to create attributes in the ETHERST for their identity.

The following parameters are passed to the function:

- Type of Attribute—the type of attribute that is represented, for instance, “key” or “name”.
- Identifier—a unique identifier for the attribute if there is any.
- Data—the attribute data which includes the proof of binding for the attribute.

A unique incrementing attribute ID will be created automatically for each new attribute by this function. This unique ID is used as a reference to the attribute. In each attribute created, we keep a value of the attribute owner by recording the address of caller who calls to CreateAttribute function in ETHERST smart contract.

AttributeWasCreated event is emitted in order to make a log in Ethereum with all of the input parameters for ETHERST clients to pick up for further processing. This event contains three indices that allow the client to do filtering: the ID of attribute, the owner of attribute and the attribute identifier.

This function keeps the data of every attribute in an array which is accessible by other smart contracts by ID. Attribute ID is the index of an attribute in the array.

4.1.2. Sign Attribute

After calling the CreateAttribute function, calling the SignAttribute function will allow an entity to sign the attributes of any other entities. It accepts the following arguments.

- Attribute ID—a unique identifier for the attribute.
- Expiry—the expiry timestamp for the signature. It represents the time at which the signature is considered invalid.

This function will generate a unique signature ID for the new signature. The signer of the new signature is the function caller’s transaction address. Prior to calling the SignAttribute function, each node needs to deposit an amount of 0.0005 ETH to ETHERST smart contract. Without enough balance, calling to this function will fail.

It also stores the data of the signers of the signatures in the signature array. The information allows the ETHERST signature revocation function to match the signers of signatures with the revocation function caller when revocation function is invoked.

The SignatureWasAdded event is then emitted to log all of the input arguments of the function call, the signature ID and signer. This event contains three indices that allow the client to do filtering: the signature ID, the signer and the attribute ID.

4.1.3. Revoke Signature

While SignAttribute allows entities to sign their attributes, RevokeSignature function allows entities to revoke own signatures.

It accepts only an argument.

- Signature ID—unique identifier for the signature.

This function will generate a unique revocation ID for the new revocations. A SignatureWasRevoked event is then emitted to log with the signature ID and the revocation ID.

The RevokeSignature function only keeps and increments the latest ID of revocation. It only revokes signatures if the function caller is matching the signer of the signature to be revoked.

4.1.4. Trust Signature

Trust signature is not an essential part of conventional PKI functionality. It is the part of the economic automation implementation of the reward-and-punishment mechanism which improves SCPKI. Each node can trust any signature of other nodes. For example, if node A trusts a signature X of node B, node A is named as “trustor” of signature X of node B. As a reward, the trustor will be rewarded with a specific fixed amount of Ether. The same amount is deducted from the Ether balance of the signature owner at the same time as a cost of getting trust in the ecosystem.

To control the overwhelming trusting actions, each signature can only be trusted by TRUST-LEVEL nodes where TRUST-LEVEL is a limit of the total of trustors. Only the first TRUST-LEVEL-th nodes will be rewarded. Besides, to avoid the dispute action from a specific node or nodes, we limit the maximum action of trust to one time per day from a node.

The pseudo-code for this function is shown in Algorithm 1.

Algorithm 1: TRUST SIGNATURE.

```
Result: Status of trusting
Input: SignatureID
Output: status of trusting
Initialization
Deny if mapLastTrustAction[msg.sender] less than 1 day
if $mapTrustor[signature.attributeID][msg.sender] != true$ then
    if $signature.trusteeCount < TRUST\text{-}LEVEL$ then
        $signature.trustorCount \leftarrow signature.trustorCount + 1$
        $signature.trustors[signature.trustorCount - 1] \leftarrow msg.sender$
        $mapTrustor[signature.attributeID][msg.sender] \leftarrow true$
        Reward Trustor with Ether
        $status \leftarrow true$
        emit event SignatureTrusted
    end
end
return $status$
```

4.1.5. Untrust Signature

Like trust signature, untrust is not an essential part of the conventional PKI functionality, but part of the automation economic rewarding mechanism. Each node can untrust any signature of other nodes. For example, if node untrusted signature Y of node B, node C is named as "untrustor" of signatures of node B. An untrustor for a node cannot be a trustor of the same node.

Each signature can only be trusted by UNTRUST-LEVEL number of nodes where UNTRUST-LEVEL is a limit of the total of untrustors for a specific node.

To avoid dispute, the reward will only be given out to untrustors when the count of untrustors reaches the UNTRUST-LEVEL. As a reward, all untrustors will be rewarded with a specific fixed amount of Ether equally when the total of untrustors for a specific signature reach UNTRUST-LEVEL. The same amount of Ether is deducted from the Ether balance of the signature owner at the same time, as a cost for being untrustworthy in the ecosystem.

Same as the trust signature, we also limit the maximum action of untrust to one time per day from a specific node. This helps to avoid several nodes colluding and take a node down in a short time.

The pseudo code for this function is shown in Algorithm 2.

We implemented ETHERST version 1.0 prototype and ran simulations with the prototype on a local Ethereum blockchain. The details of the implementation and simulation is presented in Section 5. During the process of developing the prototype and simulations, we notice some limitations of ETHERST version 1.0:

1. Fluctuation of the market value of Ether affects the reward-and-punishment mechanism feasibility and increases uncertainty.
2. Requirement of deposit before attribute creation discourages involvement of nodes.

Due to the limitation of ETHERST version 1.0, we propose an improved version, ETHERST version 2.0.

4.2. ETHERST Version 2.0

After reviewing the ETHERST version 1.0 limitation, we introduce the ERC-20 Token in ETHERST version 2.0. We create our own token named PKIToken which is a custom token that follows the specifications of ERC-20.

Algorithm 2: UNTRUST SIGNATURE

```
Result: Status of untrusting
Input: SignatureID
Output: status of untrusting
Initialization
Deny if mapLastUnTrustAction[msg.sender] less than 1 day
if mapUntrustor[signature.attributeID][msg.sender] != true &&
 mapUnTrustor[signature.attributeID][msg.sender] != true then
    if $signature.untrustorCount < UNTRUST\text{-}LEVEL$ then
        $signature.untrustorCount \leftarrow signature.untrustorCount + 1$
         $signature.untrustors[signature.untrustorCount - 1] \leftarrow msg.sender$
         $mapUntrustor[signature.attributeID][msg.sender] \leftarrow true$
        if $signature.unTrustorCount >= UNTRUST\text{-}LEVEL$ then
            for each untrustor do
                Reward each untrustor with Ether
            end
            emit event SignatureRevokedByOthers
        end
        $status \leftarrow true$ emit event SignatureUnTrusted
    end
end
return status
```

ETHERST version 2.0 comprises two smart contracts. Apart from the ETHERST version 2.0 smart contract, we create another smart contract to implement PKIToken.

1. PKIToken smart contract.
2. ETHERST version 2.0 smart contract.

The Entity Relationship Diagramme (ERD) of ETHERST version 2.0 smart contract is same as ETHERST version 1.0 smart contract which is shown in Figure 4.

All the name of functions in ETHERST version 1.0 smart contract were retained in ETHERST version 2.0 and no new function is created. The logic in the functions CreateAttribute and RevokeSignature are identical in both ETHERST version 1.0 and ETHERST version 2.0. The other three functions, SignAttribute, TrustSignature and UntrustSignature, were enhanced and modified to integrate PKIToken into ETHERST.

In the following subsections, we describe the enhancements and differences of three functions in detail.

4.2.1. Sign Attribute

With the same arguments as ETHERST version 1.0, calling the SignAttribute function will allow entities to sign the attributes of any entities.

This function will generate a unique signature ID for the new signature. The signer of the new signature is the function caller's transaction address. Unlike ETHERST version 1.0, in ETHERST version 2.0, the caller does not need to deposit any amount of Ether into ETHERST smart contract. The reason for the change is because the ETHERST smart contract can transfer ERC-20 tokens on behalf of any of the node in ETHERST, therefore, there is no prepaid deposit of the amount needed.

4.2.2. Trust Signature

Trust signature of ETHERST version 2.0 logic is same as ETHERST version 1.0. The difference is on the rewarding medium, where the trustor will be rewarded with a specific fixed amount of PKIToken instead of Ether. The same amount of PKIToken is deducted from the signature owner at the same time as a cost of getting trust in the ecosystem. We set this fixed amount as 10 PKIToken in our simulation which will be discussed in the next section.

Figure 5 shows the scenario when ETHERST version 2.0 trust signature happens. In the figure, assume if we fix the incentive amount as 10 PKIToken, Alice trusts Bob and Alice will be rewarded 10 PKIToken from Bob. Meanwhile, John trusts Alice and John will be rewarded 10 PKIToken from Alice.

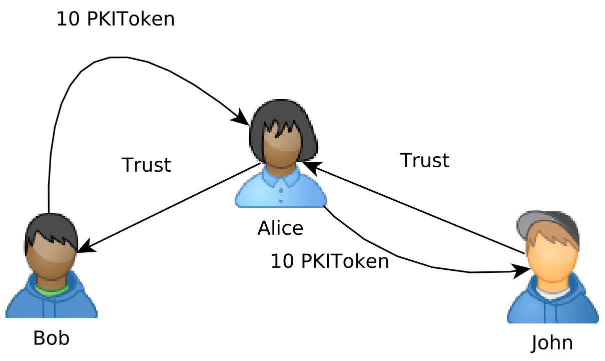

Figure 5. ETHERST version 2.0: trust a node.

Besides, to control the dishonest node who wishes to gain incentive by trusting without discretion, we improve the algorithm in ETHERST version 1.0 by implementing an incremental and random delay time for the next allowed trust action. We impose an incremental delay of 1 to $m - 1$ days, then followed by a random delay of m to n days between each trust action. For the dishonest node that wishes to gain incentive by trusting without discretion, the limit of trusting action within an incremental and random delay time creates an uncertain effect to discourage the dishonest node. The more dishonest the actions, the longer time they need to wait to perform the next trust action. The value of m and n is configurable by the implementor of ETHERST in the ETHERST smart contract before publishing to the Ethereum blockchain.

The value of delay between z-th and *(z+1)*-th trust action in unit of day is computed as below:

$$f(z) = \begin{cases} z & \text{for} \quad 0 <= z <= m \\ \textit{randomly choose between } m+1 \textit{ to } n & \text{for} \quad z > m \end{cases}$$

The pseudo code for the trust signature function is shown in Algorithm 3.

Algorithm 3: TRUST SIGNATURE

```
Result: Status of trusting
Input: SignatureID
Output: status of trusting
Initialization
if mapLastTrustAction[msg.sender] is after the allowed next-action time then
    if mapTrustor[signature.attributeID][msg.sender] != true then
        if signature.trusteeCount < TRUST-LEVEL then
            $signature.trustorCount \leftarrow signature.trustorCount + 1$
            $signature.trustors[signature.trustorCount - 1] \leftarrow msg.sender$
            $mapTrustor[signature.attributeID][msg.sender] \leftarrow true$
            Reward Trustor with PKIToken
            $status \leftarrow true$
            emit event SignatureTrusted
        end
    end
end
return status
```

4.2.3. Untrust Signature

Like trust signature, the difference between ETHERST version 2.0 with version 1.0 is in the rewarding medium. The untrustor will be rewarded with a specific fixed amount of PKIToken instead of Ether. The same amount of PKIToken is deducted from the signature owner at the same time as a cost for not being trustworthy in the ecosystem.

Figure 6 shows the scenario when ETHERST version 2.0 untrust signature happen. Assume that we fix disincentive amount as 50 PKIToken and in the figure, Dan is already untrusted by four users. When the fifth user, Tom, untrusted Dan, Dan will be punished and 50 PKIToken will be deducted from him. The 50 PKIToken will then be transferred as a reward equally among all the nodes that untrusted Dan before.

Like trust signature in ETHERST version 2.0, in order to control the dishonest node who wishes to gain incentive by untrusting without discretion, we implemented a random and incremental delay time for the next allowed untrust action. We imposed the same delay algorithm as in trust signature.

The value of delay between *z*-th and *(z+1)*-th untrust action in unit of day is computed as below:

$$f(z) = \begin{cases} z & \text{for} \quad 0 <= z <= m \\ \textit{randomly choose between } m+1 \textit{ to } n & \text{for} \quad z > m \end{cases}$$

The pseudo code for the untrust signature function is shown in Algorithm 4.

Algorithm 4: UNTRUST SIGNATURE

```
Result: Status of untrusting
Input: SignatureID
Output: status of untrusting
Initialization
if mapLastUnTrustAction[msg.sender] is after the allowed next-action time then
    if mapUntrustor[signature.attributeID][msg.sender] != true &&
      mapUnTrustor[signature.attributeID][msg.sender] != true then
        if signature.untrustorCount < UNTRUST-LEVEL then
            $signature.untrustorCount \leftarrow signature.untrustorCount + 1$
            $signature.untrustors[signature.untrustorCount - 1] \leftarrow msg.sender$
            $mapUntrustor[signature.attributeID][msg.sender] \leftarrow true$
            if signature.unTrustorCount >= UNTRUST-LEVEL then
                for each untrustor do
                    Reward each untrustor with PKIToken
                end
                emit event SignatureRevokedByOthers
            end
            $status \leftarrow true$
            emit event SignatureUnTrusted
        end
    end
end
return status
```

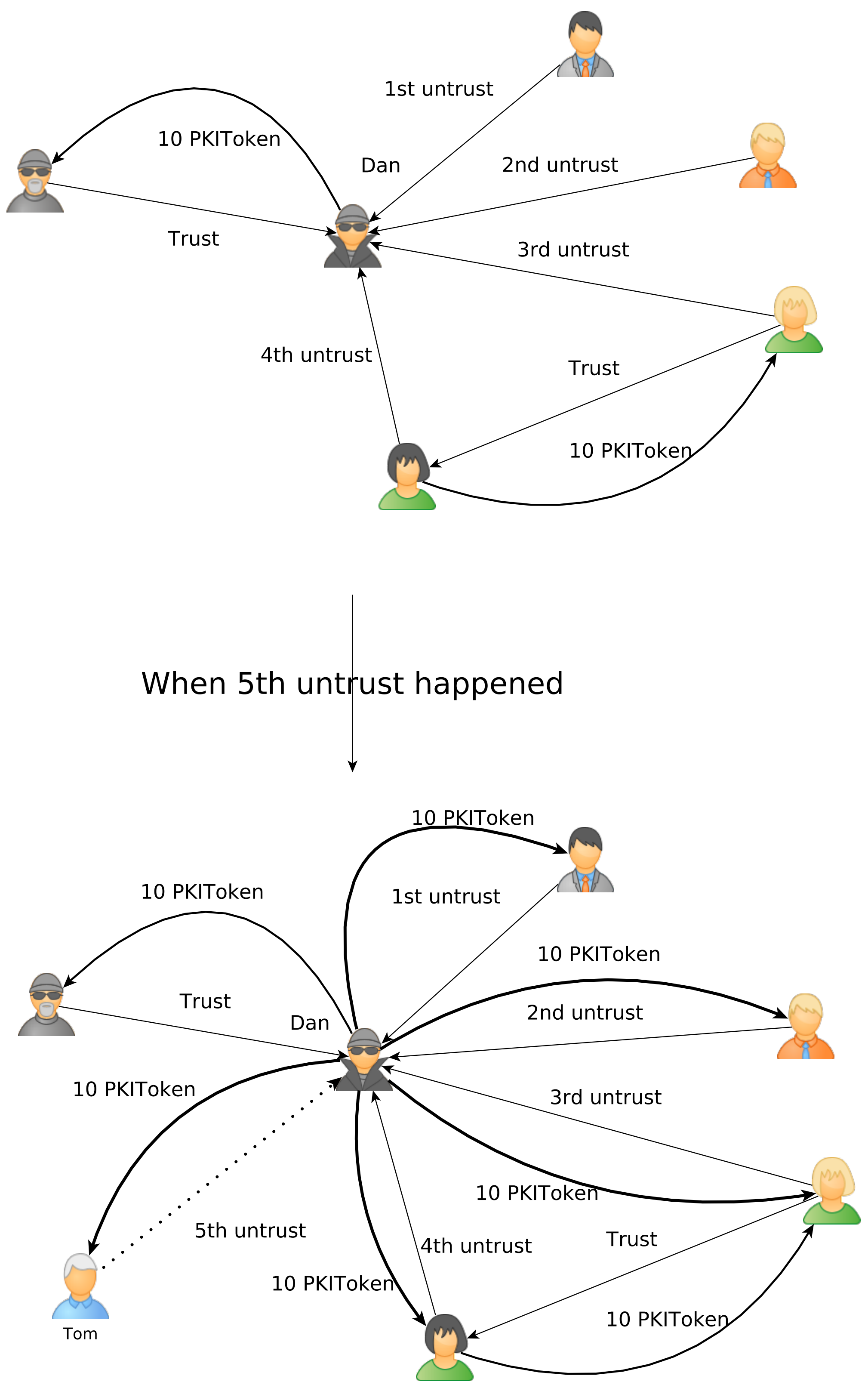

Figure 6. ETHERST version 2.0: Untrust a node. When the 5th untrust happens, a punishment is triggered. All the untrustors receive an amount of 10 PKIToken from user who has been untrusted.

5. Implementation and Evaluation

In this section, we present the prototype of the proposal and evaluate its output. The smart contracts were written in programming language Solidity v0.5.16 while the simulation script was written in Javascript. The whole prototype is developed with the Truffle Suite [35] and utilized the OpenZeppelin library [36]. Simulations were run on Ganache, a local Ethereum blockchain that comes with the Truffle Suite.

The simulation was carried out on a Dell Inspiron laptop (Intel i7-8565U/4.6 GHz Processor, 16 GB RAM and 512 GB M.2 PCIe NVMe Solid State Drive) and the local blockchain was running on the same laptop. We fixed several parameters in the simulations of both ETHERST version 1.0 and ETHERST version 2.0. They include

1. Size of attribute data is 500 bytes each.
2. TRUST-LEVEL is set as 5.
3. UNTRUST-LEVEL is set as 5.

5.1. ETHERST Version 1.0

The prototype for ETHERST version 1.0 consists of the following items

1. ETHERST version 1.0 smart contract
2. Simulation script

Smart contract of ETHERST version 1.0 is implemented in 166 lines of solidity.

We started the simulation with 100 account nodes with an initial balance of Ether of 100 ETH. The initial Ether is required because transferring any Ether required a gas fee which needs to be paid with Ether. Besides, before signing on an attribute, the node needs to deposit an amount of Ether to ETHERST smart contract. In the simulation, we fix the amount as 0.001 ETH. Besides, the rewarding amount is set as 0.0001 ETH and the punishment amount is set as 0.0005 ETH.

5.2. ETHERST Version 2.0

1. PKIToken smart contract
2. ETHERST version 2.0 smart contract
3. Simulation script

Smart contracts in ETHERST version 2.0 were implemented in a total of 200 lines of Solidity. In PKIToken smart contract, we created an ERC-20 token named "PKIToken", this token is attached to the Ethereum blockchain and will be used as an incentivize and disincentivize token to the PGP Web of Trust.

Similarly, we started the simulation with 100 account nodes with an initial balance of Ether of 100 ETH. The initial Ether is required because transferring any ERC-20 token like PKIToken required gas fee which needs to be paid with Ether. All the PKITokens are then allocated to the root or first account of the Ethereum which is owned by the implementor. Similar to the simulation of ETHERST version 1.0, in ETHERST version 2.0 we fixed the rewarding amount as 10 PKIToken and punishment amount as 50 PKIToken. Besides, each node that created the first attribute in ETHERST 2.0 will be given 50 PKIToken as an initial balance by the root account. Before the simulation, we deployed the PKIToken smart contract and ETHERST version 2.0 smart contract on our local Ethereum blockchain in Ganache.

This implementation of ETHERST version 2.0 has the following advantages over the ETHERST version 1.0:

1. Minimize the impact of fluctuation of market value of Ether on the ecosystem of ETHERST in a commercial implementation.
2. Lower the entry cost that will encourage participation of each node.

5.3. Simulation and Evaluation

We carried out simulations with ETHEREST version 1.0 and version 2.0. The simulations helped us to analyze the cost of implementing ETHERST and also to test the feasibility of the deployment of ETHERST.

5.3.1. Analysis of Cost

With this implementation, the total cost of deploying ETHERST version 1.0 and version 2.0 are 0.04546432 ETH and 0.08854386 ETH respectively. The cost was mainly to cover the gas fee to deploy the smart contracts to the blockchain network as transactions. It is a one time cost to implement ETHERST. As of June 2021, 1 unit of gas cost 15 gwei which is 0.000000015 ETH and 1 ETH = $2603.17.

Tables 1 and 2 display the estimated cost for creating attribute data with different sizes, from 100 bytes to 500 bytes for both ETHERST version 1.0 and ETHERST version 2.0 respectively. From the tables, we noticed that the more data there are, the higher the gas fee is. It means that the participants in the network need to pay a higher cost to add an attribute with the bigger size data.

Table 1. ETHERST version 1: gas used to create attribute.

Data (Byte)	Gas Used	Gas Fee in ETH	Gas Fee in USD
100	238,936	0.00358404	9.3299
200	306,951	0.00460427	11.9857
300	375,030	0.00562545	14.6440
400	443,044	0.00664566	17.2998
500	511,059	0.00766589	19.9556

Table 2. ETHERST Version 2: gas used to create attribute.

Data (Byte)	Gas Used	Gas Fee in ETH	Gas Fee in USD
100	280,329	0.00420494	10.9462
200	348,344	0.00522516	13.6020
300	416,423	0.00624635	16.2603
400	484,437	0.00726656	18.9161
500	552,452	0.00828678	21.5719

Apart from creating an attribute, there are other operations that involve cost in both ETHERST version 1.0 and version 2. They include

1. Publish contracts
2. Sign attribute
3. Revoke signature
4. Trust signature
5. Untrust signature

Firstly, the cost of publishing contracts is higher in ETHERST version 2.0 compared to version 1.0. The reason is version 2.0 consists of two contracts PKIToken and ETHERST and the costs are 0.05931191 ETH which is equivalent to 154.3990 USD as of June 2021, while version 1.0 used 2084797 unit of gas which costs 0.03127196 ETH equivalent to 81.4061 USD as of June 2021. Publishing contracts is a one-time cost—it will be incurred only when we need to update the smart contracts and the cost will depend on the content size of updated smart contract.

In contrast to publishing contracts, the other operations are not one-time costs. In order to investigate the cost of the remaining operations, we ran a simulation to analyze the cost of these operations. The simulation started with creating an attribute with 200 bytes of data. After creating the signature, we carried out the subsequent operations in the sequence of sign attribute, trust signature, untrust signature and finally revoke signature. The simulation was carried out with a test script written in Javascript with the TruffleSuite toolchain.

There is a difference between ETHERST version 1.0 and version 2.0 in the simulation algorithm, where we needed to deposit an amount of 0.0005 ETH to ETHERST version 1.0 before triggering the operations that we want to analyze.

Algorithm 5 shows the simulation algorithm for ETHERST version 1.0 while Algorithm 6 shows the simulation algorithm for ETHERST version 2.0.

Algorithm 5: ALGORITHM TO OBTAIN GAS USAGE OF ETHERST VERSION 1.0 OPERATIONS

```
    Result: Gas usage of each operations
    Input:
    Output: Gas usage of each operations
    Initialize nodes with random data
    Choose a dedicated node A, deposit 0.0002 ETH to ETHERST contract
    attrId ← node A create an attribute of 200 bytes data
    signatureId ← node A sign attribute
    Get the total gas used for signing attribute
    Choose a dedicated node B, trust Signature signatureId
    Get the total gas used for trusting signature
    Choose a dedicated node C, untrust Signature signatureId
    Get the total gas used for untrust signature from event
    node A revoke signature signatureId
    Get the total gas used for untrust signature from event
    Print usage of gas for all the operations.
    return status
```

Algorithm 6: ALGORITM TO OBTAIN GAS USAGE OF ETHERST VERSION 2.0 OPERATIONS

```
    Result: Gas usage of each operations
    Input:
    Output: Gas usage of each operations
    Initialize nodes with random data
    Choose a dedicated node A
    attrId ← node A create an attribute of 200 bytes data
    signatureId ← node A sign attribute
    Get the total gas used for signing attribute
    Choose a dedicated node B, trust Signature signatureId
    Get the total gas used for trusting signature
    Choose a dedicated node C, untrust Signature signatureId
    Get the total gas used for untrust signature from event
    node A revoke signature signatureId
    Get the total gas used for untrust signature from event
    Print usage of gas for all the operations.
    return status
```

Signing an attribute, which is an operation of conventional PKI, used 101,127 units of gas and cost 0.00151691 ETH (3.9488 USD as of June 2021) in ETHERST version 1.0. For ETHERST version 2.0, the cost was 100,631 units of gas and 0.00150947 ETH (3.9294 USD as of June 2020). The reduced cost from version 1.0 to version 2.0 is due to an extra checking logic that requires the signer to have a deposit of 0.0005 ETH in the ETHERST version 1.0 smart contract while there is no such requirement in version 2.0. The cost of revoking a signature which is another operation of conventional PKI, is the same between ETHERST version 1.0 and version 2.0 because the logic in both versions is identical.

Table 3 shows the total gas used and its equivalent cost of other operations in ETHERST. These two costs are only applicable when a node signs on an attribute and revokes the corresponding signature. The more attributes added, the more signatures to be signed.

Table 3. Gas used for other operations.

Operation	**Gas Used**		**Gas Fee in ETH**		**Gas Fee in USD**	
	.Ver 1	**.Ver 2**	**Ver 1**	**Ver 2**	**Ver 1**	**Ver 2**
Publish contracts	2,084,797	3,954,127	0.03127196	0.0531191	81.4062	154.3990
Sign Attribute	101,127	100,631	0.00151691	0.00150947	3.9488	3.9294
Revoke Signature	22,263	22,263	0.00033395	0.00033395	0.8693	0.8693
Trust Signature	101,047	147,072	0.00151571	0.00220608	3.9457	5.7428
Untrust Signature	97,189	97,189	0.00145784	0.00145784	3.7950	3.7950

The last two operations, trust signature and untrust signature, are not part of conventional PKI. They are the newly implemented functions in this new blockchain-based PKI to integrate automation economic reward-and-punishment mechanism. Each time a node, i.e, node A wants to trust a signature for another node, it will cost node A 0.00151571 ETH (3.9457 USD as of June 2021) and 0.00220608 (5.7428 USD as of June 2021) in ETHERST version 1.0 and version 2.0 respectively. The opposite operation—an untrust signature will cost 0.00145784 ETH (3.7950 USD as of June 2021) and is the same in both ETHERST version 1.0 and version 2.0.

5.3.2. Random Activities Simulation and Analysis

Simulations of the implementations of ETHERST version 1.0 and version 2.0 were carried out by running two test scripts in Javascript developed using the TruffleSuite toolchain.

The simulation algorithms for both ETHERST version 1.0 and version 2.0 are the same. The algorithm of main code block for the simulation is shown in Algorithm 7.

Algorithm 7: SIMULATION

```
Result: Status of simulation
Input:
Output: status of simulation
for each N in (10,20,30 ... 70,80,90) do
    Choose random N nodes
    Initialize nodes with random data
    activites ← a list of random activities
    for each node do
        Map to one activity in the activities
    end
    Run the activity for each node-activity in mapping asynchronously.
    Wait until all activities completely run, print balance of each node.
end
return status
```

The simulation was carried out on ETHERST version 1.0 to analyze the ETH balance of each node if N of the nodes are participating in the activities. This is to analyze the feasibility of the ETHERST version 1.0. The results were found not to be up to the expected accuracy because the balance of ETH for each node is not only affected by the reward and punishment activity but is also affected by activities carried out by each node because gas fee will be charged based on the activities carried out. Hence, we introduced ETHERST version 2.0 which uses a separate currency property in Ethereum, an ERC-20 token.

The same simulation was carried out on ETHERST version 2.0 to analyze the operation of reward-and-punishment and ERC-20 token distribution among the participants in ETHERST version 2 after the simulation.

As stated in the Algorithm 7, we carried out simulations for N nodes where $N = 10$, 20, 30 ... 70, 80, 90. Table 4 presents an analysis of the balance of PKIToken in each round of simulation. Since in ETHERST version 2.0, PKIToken is only to be used as a means of reward and punishment, we can use the balance of PKIToken as a measurement of the trustworthiness of a node in this new blockchain-based WoT. The higher the PKIToken balance, the higher level the trust of the node. In contrast, the lower the PKIToken balance, the trustworthy level is the lowest.

Table 4. Analysis for PKIToken balance of accounts.

N	Average	Minimum	Maximum	Time Used (Seconds)
10	38.89	30	50	6.02
20	42.11	30	60	17.58
30	45.17	30	60	19.58
40	44.87	10	90	21.87
50	44.69	10	70	28.44
60	44.41	10	80	44.17
70	46.81	10	90	47.23
80	47.72	10	120	48.54
90	48.88	10	90	61.85

From the average of balances, when the number N is increasing, so does the average. It indicates that the more activities in the ecosystem of ETHERST, the higher the accuracy of trust level in the ecosystem.

We observed that with the usage of ERC-20 token as currency for automation reward-and-punishment mechanism, we managed to reduce the dependency on Ether for the real-world implementation of a reward-and-punishment mechanism in blockchain-based distributed PKI. The balance of PKIToken can serve as a measurement of the trustworthiness level of this PKI model. If we utilize the attributes to keep the personal public key and other personal information, we can simulate a virtual key signing party held through the Internet. In this era of the coronavirus, ETHERST can be a good alternative to the face-to-face key signing party.

Compared to the previous blockchain-based PKI reward-and-punishment implementations proposed in IKP [22] and Internet Web Trust System [23], ETHERST version 2.0 encourages healthy growth of the ecosystem in the PKI because the value of reward-and-punishment medium PKIToken is under control by the model implementor. It helps to ensure the continuity of the ecosystem in a PKI.

6. Conclusions

In this paper, we proposed a new blockchain-based PKI as a WoT solution with a reward-and-punishment mechanism with ERC-20 token. The reward-and-punishment mechanism encourages the involvement of participants of the WoT and increases the activities among the community members. When deployed with an ERC-20 token, it resolves the problem of the old reward-and-punishment mechanisms that utilizes Ether, which depends on the market price. It also serves as a measurement of the trustworthiness of a participant in the ecosystem.

The ETHERST model can be a useful add-on module to real-world applications such as an e-commerce platform, where merchant and buyer can manage the trust themselves. Moreover, since the value of the token is defined by the e-commerce platform owner, the token can be used as an alternate currency to be consumed or exchange in the e-commerce platform.

For future work, we believe there is room for improvement in the reward-and-punishment algorithm and suggest looking into the possibility of minimizing the gas fee that is still being paid with Ether when transferring PKIToken for reward-and-punishment. Since the PKIToken amount in the reward-and-punishment in ETHERST version 2.0 is a fixed amount, we believe improvements with a dynamic amount based on a weighted average algorithm with the four trust levels in the WoT is an interesting direction. Besides, analyzing the dynamic value of PKIToken with a pair of optimum values for TRUST-LEVEL and UNTRUST-LEVEL, which can avoid collusion of the untrusting action is an area that can be further explored.

Author Contributions: Supervision, S.-H.H. and J.-J.C.; review and editing, S.-H.H. and J.-J.C.; writing-original draft, C.-G.K.; methodology, C.-G.K.; visualization, C.-G.K.; formal analysis, C.-G.K.; software, C.-G.K.; project administration, S.-H.H. and J.-J.C.; funding acquisition, S.-H.H. and J.-J.C. All authors have read and agreed to the published version of the manuscript.

Funding: The authors would like to acknowledge the Multimedia University's GRA scheme, and the Malaysia government's Fundamental Research Grant Scheme (FRGS/1/2018/ICT04/MMU/01/01) and Prototype Research Grant Scheme (PRGS/2/2019/ICT04/MMU/01/1) for supporting this work.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Yakubov, A.; Shbair, W.; Wallbom, A.; Sanda, D.; State, R. R: A blockchain-based PKI management framework. In Proceedings of the 2018 IEEE/IFIP Network Operations and Management Symposium, Taipei, Taiwan, 23–27 April 2018.
2. Koa, C.G.; Heng, S.H.; Tan, S.Y.; Chin, J.J. Review of Blockchain-Based Public Key Infrastructure. In Proceedings of the 7th International Cryptology and Information Security Conference 2020 (CRYPTOLOGY2020), 9–10 June 2020; pp. 20–31.
3. Prins, J.; Cybercrime, B. Technical Report;Diginotar certificate authority breach operation black tulip, 2012. Accessed on 31st March 2019.
4. Bano, S. *Consensus in the Age of Blockchains*; CoRR, 2017, abs/1711.03936 , https://arxiv.org/abs/1711.03936v2.
5. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system, https://bitcoin.org/bitcoin.pdf. Accessed on 28th February 2019.
6. Chen, G.; Xu, B.; Lu, M.; Chen, N.-S. Exploring blockchain technology and its potential applications for education. *Smart Learn. Environ.* **2018**, *5*, doi:10.1186/s40561-017-0050-x.
7. Castro, M.; Liskov, B. Practical Byzantine Fault Tolerance. *OSDI* **1999**, *99*, 173–186.
8. Bentov, I.; Gabizon, A.; Mizrahi, A. Cryptocurrencies Without Proof of Work. In *Financial Cryptography and Data Security*; Clark, J., Meiklejohn, S., Ryan, P.Y., Wallach, D., Brenner, M., Rohloff, K., Eds.; Springer: Berlin/Heidelberg, Germany, 2016; pp. 142–157.
9. Kiayias, A.; Russell, A.; David, B.; Oliynykov, R. Ouroboros: A provably secure proof-of-stake blockchain protocol. Proceedings of the 37th Annual International Cryptology Conference (CRYPTO). Springer.
10. Dhillon, V.; Metcalf, D.; Hooper, M. The Hyperledger Project. In *Blockchain Enabled Applications: Understand the Blockchain Ecosystem and How to Make It Work for You*; Apress: Berkeley, CA, USA, 2017; pp. 139–149, doi:10.1007/978-1-4842-3081-7_10.
11. Brandenburger, M.; Cachin, C.; Kapitza, R.; Sorniotti, A. *Blockchain and Trusted Computing: Problems, Pitfalls, and a Solution for Hyperledger Fabric*; CoRR, 2018, abs/1805.08541, https://arxiv.org/abs/1805.08541v1. Accessed on 26 June 2021.
12. Underwood, S. Blockchain beyond Bitcoin. *Commun. ACM* **2016**, *59*, 15–17, doi:10.1145/2994581.
13. Wiki, B. Script: simple, stack-based, https://en.bitcoin.it/wiki/Script. Access on 31 October 2020.
14. Wright, C.S. Turing Complete Bitcoin Script White Paper, 2016. http://dx.doi.org/10.2139/ssrn.3160279. Accessed on 27 June 2021.
15. Buterin, V. Ethereum: A Next-Generation smart contract and decentralized application platform, Whitepaper, https://ethereum.org/en/whitepaper/. Accessed 31 May 2020.
16. 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 of the IEEE International Conference on Computer Communications (INFOCOM), Toronto, ON, Canada, 27 April 2018; pp. 2060–2068.
17. ERC-20. EIP-20: ERC-20 Token Standard, https://eips.ethereum.org/EIPS/eip-20. Accessed 28 February 2020.

18. Fromknecht, C.; Velicanu, D.; Yakoubuv, S. A decentralized public key infrastructure with identity retention. *Cryptol. ePrint Arch.* **2014**, *2014*, 803.
19. Ryabitsev, K. PGP web of trust: Core concepts behind trusted communication, https://www.linux.com/training-tutorials/pgp-web-trust-core-concepts-behind-trusted-communication/. Accessed 31 December 2020.
20. Ulrich, A.; Holz, R.; Hauck, P.; Carle, G. Investigating the OpenPGP web of trust. *ESORICS LNCS* **2011**, *6879*, 489–507, doi:10.1007/978-3-642-23822-2_27.
21. Brunner, C.; Knirsch, F.; Unterweger, A.; Engel, D., A Comparison of Blockchain-based PKI Implementations. In *Proceedings of the 6th International Conference on Information Systems Security and Privacy, ICISSP 2020, Valletta, Malta, February pp. 333–340, 2020.* 2020;
22. Matsumoto, .; Reischuk, R. IKP: Turning a PKI around with decentralized automated incentives. In Proceedings of the IEEE Symposium on Security and Privacy (SP), San Jose, CA, USA, 22–24 May 2017; pp. 410–426.
23. Li.; Wang, N.; Du, X.; Liu, A. Internet Web Trust System Based on Smart Contract. *Commun. Comput. Inf. Sci.* **2019**, *1058*, 295–311.
24. Namecoin. Namecoin: Decentralized secure names, Feb. 2019.
25. Benaloh, J.; Mare, M.D. *One-Way Accumulators: A Decentralized Alternative to Digital Signatures*; Advances in Cryptology — EUROCRYPT '93, Springer Berlin Heidelberg, 1994;pp. 274–285.
26. Camacho, P.; Hevia, A.; Kiwi, M.; Opazo, R. Strong accumulators from collision-resistant hashing. *Lect. Notes Comput. Sci.* **2008**, *5222*, 471–486.
27. Camenisch, J.; Kohlweiss, M.; Soriente, C. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. *Lect. Notes Comput. Sci.* **2009**, *5443*, 481–500.
28. Burton, H.; Bloom. Space/timetrade-offs in hash coding with allowable errors. *Commun. ACM* **1970**, *13*, 422–426.
29. Ali, J.; Nelson, R.; Shea, M.; Freedman. Blockstack: A Global naming and storage system secured by blockchains. In Proceedings of the 2016 USENIX Conference on Usenix Annual Technical Conference, Denver, CO, USA, 22–24 June 2016.
30. Leiding, B.; Cap, C.; Mundt, T.; Rashidibajgan, S. Authcoin: Validation and Authentication in Decentralized Networks. *arXiv* **2016**, arXiv:1609.04955.
31. Lewison, K.; Corella, F. *Backing Rich Credentials with a Blockchain PKI*; Technical Report; Pomcor: https://pomcor.com/techreports/BlockchainPKI.pdf, 2016. Accessed 31 June 2020
32. Cooper, D.; Santesson, S.; Farrell, S.; Boeyen, S.; Housley, R.; Polk, W. Internet X.509 public key infrastructure certificate and CRL profile, RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>. Accessed on 27 June 2021
33. Al-Bassam, M. SCPKI: A smart contract-based PKI and identity system. In Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, Association for Computing Machinery, Abu Dhabi, UAE, 2–6 April 2017; pp. 35–40.
34. Kim, Y.; Kim, J.; Kim, W.; Im, J.; Kim, T.; Kang, S. Predicting Fluctuations in Cryptocurrency Transactions Based on User Comments and Replies. *PLoS ONE* **2016**, *11*, e0161197, doi:10.1371/journal.pone.0161197.
35. TruffleSuite. TruffleSuite: Sweet tools for smart contracts, https://www.trufflesuite.com/. Accessed 31 October 2020.
36. OpenZepplin. OpenZepplin: The standard for secure blockchain applications, https://openzeppelin.com/. Accessed 31 October 2020.