Neural Computing and Applications
https://doi.org/10.1007/s00521-019-04403-6

GREEN AND HUMAN INFORMATION TECHNOLOGY 2019

A PKI without TTP based on conditional trust in blockchain

KyungHyun Han[1] · Seong Oun Hwang[2]

Received: 8 May 2019 / Accepted: 30 July 2019

Abstract
Many people have used public keys in various areas based on public key infrastructure (PKI). PKI provides a method to publicize public keys securely. However, existing PKI methods have a problem that they assume trusted third parties. Therefore, the existing PKIs cannot be used when users cannot trust certificate issuers. To solve this problem, we propose a new trust model and describe its implementation based on blockchain. Users can trust the certificate issued by full nodes even if they do not trust the full nodes themselves. We analyze the security of our model and show that its security can be achieved higher than existing models. This new model can be particularly useful in an environment where a third party cannot be easily trusted.

Keywords PKI · Blockchain · Trust model · Certificate

1 Introduction

In this modern era, there are two major encryption algorithms, which are symmetric key and asymmetric key cryptography. In the symmetric key cryptography, users use the same key for encryption and decryption. To prevent information leakage, the key must be kept as secret to both encrypting and decrypting parties only. Unfortunately, it is difficult for two or more parties that did not share any information to agree to the same secret in symmetric cryptography. In contrast, pairs of keys are used in asymmetric key cryptography, where private keys are kept secret by owners and their corresponding public keys are disseminated publicly. Therefore, it becomes easier to share keys among users in asymmetric key cryptography than in symmetric key cryptography. Therefore, asymmetric key cryptography has been widely used for digital signature as well as encryption.

However, sharing the public keys in asymmetric cryptography still raises a problem. That is, it is difficult for the receiver of a public key to believe that the received key is the right public key of the sender, because it may be possible for a sender to send a public key with wrong identity or for someone else to modify the public key or the sender's information in the middle. To use public key securely, public key infrastructure (PKI) is necessary. PKI provides a function that binds a public key with its owner. Most of the currently used public keys are shared under the control of the PKI.

Certificate authority (CA) and web of trust (WoT) are popularly used methods for PKI in the real world. But all of these models have a problem in common with trust. For example, CA assumes a trusted third party; WoT lacks a method to get trust from users. While they get someone else's public keys, however, users are forced to trust CA or WoT without evaluating how trustworthy they are.

To solve the aforementioned problem, this paper proposes a new trust model and describes the implementation of the proposed trust model through blockchain. Instead of providing a means of evaluating a third party, it provides a means of evaluating certificates. In the proposed model, users can trust some else's public keys, even though they don't trust the third party who issued the certificates on the public keys. Lastly, we will evaluate the proposed model and present its promising applications.

✉ Seong Oun Hwang
sohwang@hongik.ac.kr

KyungHyun Han
co112kr@mail.hongik.ac.kr

[1] Department of Electronics and Computer Engineering, Graduate School of Hongik University, Sejong, Korea

[2] Department of Software and Communications Engineering, Hongik University, Sejong, Korea

This paper consists of six sections. Section 2 explains the related work. Section 3 describes the background on blockchain. Section 4 describes the proposed model in detail. Section 5 evaluates the proposed model. Section 6 describes some applications of the proposed model. Finally, Sect. 7 concludes this paper with future work.

2 Related work

Each PKI has its own method of certification for gaining trust: certificate authority, web of trust and blockchain-based methods.

2.1 Existing methods of certification

Before blockchain appears, there have been two methods to get trust: certificate authority and web of trust [1].

CA is a method in which public keys are distributed by a trusted third party. The primary roles of the CA are to generate a certificate by binding a public key with its owner using the CA's digital signature and publish it on some public directories. The standard format of certificate is X.509 [2], where an issuer is an entity who generates a certificate. A subject is an entity who requests to generate his or her certificates. CA systems have tree structures. Therefore, there may exist many CAs, but only one root CA exists in a CA system. Users who trust the root CA can verify the certificates issued by the root CA and its subordinate CAs. However, users cannot verify the certificate issued by other CAs that they do not trust. Therefore, if a root CA loses trust, e.g., due to being hacked from attackers, the entire PKI system controlled under the root CA will be compromised.

Web of trust is a method which allows trusted people to exchange their public keys [3]. In this method, any users can act as a CA. However, authentication is entirely decentralized. Pretty Good Privacy (PGP) uses this method. A PGP system has a web structure. Any user generates his public key and sends it to other users. However, the user cannot trust public keys received from the other users who he or she does not trust, because the method does not provide any mechanism to establish trust among users. PGP systems have been used in a narrow range because new participants have difficulties in getting trust from existing ones.

2.2 Blockchain-based PKIs

Recently, PKIs based on blockchain technology are being proposed by many researchers. In a centralized system, one of the comprising nodes fails, and then the whole system will be highly likely to fail. To cope with the single-point-of-failure problem, blockchain stores the same data partitions on duplicate nodes in a distributive manner.

Certcoin is the PKI proposed by Fromknecht [4, 5]. Each user in Certcoin uses two pairs of keys. Blockchain publishes one pair of keys to disseminate public keys as typical CAs normally do. The other pair of keys is used for issuers (e.g., full nodes) on blockchain to identify subjects. Certcoin treats blockchain (e.g., Namecoin [6, 7]) as a public "bulletin boards," where information can be posted in a permanent way [4]. This system also has the same problem with CA systems: Users need to trust this system, but there is no way for users to distinguish authentic CA from malicious or fake CAs.

PB-PKI proposed by Axon and Goldsmith in [8] upgrades Certcoin by supporting privacy, which is enabled by avoiding the public linking of a public key with an identity. However, like other methods, it does not explore the management of group trust yet.

Cecoin is another PKI proposed by Qin [9]. Although this system does not require a trusted third party, it treats full nodes, i.e., entities who manage data on the blockchain, as CAs without providing any certification method.

3 Background

This section describes the blockchain technologies.

3.1 Blockchain

Blockchain is a continuously growing list of records (called blocks) which are linked and secured using cryptography [10]. A blockchain consists of full nodes and user nodes, as shown in Fig. 1. Blockchain is resistant to single point of failure because full nodes of a blockchain worked in a fairly distributed manner.

User nodes create transactions when a user uses services. For example, in Bitcoin [11, 12], if a user transfers

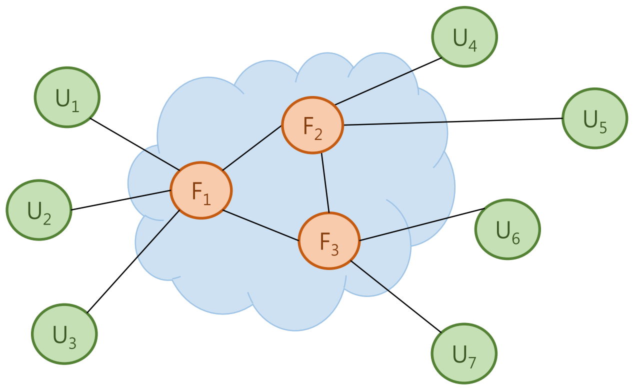


Fig. 1 The elements of blockchain (F_i: full node, U_i: user node)

some coins to another user, a transaction is created as proof of that transfer. Full nodes serve to store these transactions consistently. They receive all transactions and create blocks to store them.

Blockchain uses many cryptography techniques for storing consistent data securely over distributed full nodes. For example, digital signature to verify each transaction, Merkle tree to check all transactions in each block and consensus algorithms to check the stored block are used. All of these techniques commonly use hash functions.

3.2 Hash function

Hash functions are functions that take inputs of arbitrary length data and compress them into short, fixed-length outputs. An ideal hash function H should have the following properties:

1. Deterministic: If $x = x'$ then $H(x) = H(x')$.
2. Preimage resistance: Given z, we cannot find x with $z = H(x)$.
3. Collision resistance: A collision is a pair of distinct items x and x' for which $H(x) = H(x')$. In this case, we say that x and x' collide. A hash function should be difficult to find any collision.

A hash function is a good way to check whether data are modified or not. We usually compare the short hash values on behalf of data because of the above properties.

3.3 Digital signature

Digital signature is a value made by encrypting a hash value of data with a private key. In other words, a signature is made by a user who has the corresponding private key. Therefore, users cannot disclaim the transactions made by themselves later.

Each node in a blockchain network generates a pair of private and public keys. The private key is associated with a digital signature function, which outputs a fixed-length signature string for any arbitrary length input message. The public key is associated with a verification function, which takes as input the same message and the acclaimed signature for that message. The verification function returns true if and only if the signature is generated by the signature function with the corresponding private key and the input message. The nodes in the network identify themselves by revealing their public keys, in turn, the hash value of their public keys as their permanent addresses on the blockchain. Since each input tuple in a transaction is signed by the associated sending account, the network is able to publicly validate the authenticity of the input by verifying the signature based on the sender's public key.

3.4 Merkle tree

Merkle tree is also useful to check the modification of multiple data. Merkle tree has two properties: (1) The terminal node is data; (2) and each parent node is the hash value of their children nodes.

For example, we make a Merkle tree for eight transactions as shown in Fig. 2. Let us suppose that $h_{1\ldots8}$ is included in the block. If some data are altered, $h'_{1\ldots8}$ is calculated differently from $h_{1\ldots8}$. By comparing $h_{1\ldots8}$ and $h'_{1\ldots8}$, we can check modification without comparing all transactions. Therefore, the integrity checking process becomes faster.

3.5 Consensus algorithms

A block in blockchain includes information such as hash of the previous block, transactions, its Merkle tree and nonce. The hash value of the previous block is fed into the header of the current block to make sure that the previous block was not modified. In this way, the previous block is chained with the current block to ensure integrity. That is, if a modification happens in the previous block, then the chain will be broken. Since it calculates the hash values for all transactions on a block upward, the Merkle tree is able to check the integrity of valid transactions. The nonce can be determined according to the following consensus algorithm.

A consensus algorithm is necessary to select a full node which creates a next block. A malicious full node should not be allowed to create a block. The most representative consensus algorithm is Proof of Work [13]. All legitimate full nodes are able to find appropriate nonce in cooperation. A single full node which found a nonce creates a block to store new transaction. Appropriate nonce on each block exists, but it requires huge computing power and time for malicious full nodes to guess the nonce. It is because malicious ones have lower computing power than cooperating legitimate full nodes. There are other algorithms such as Proof of Stake [14] and Proof of Importance [15], where

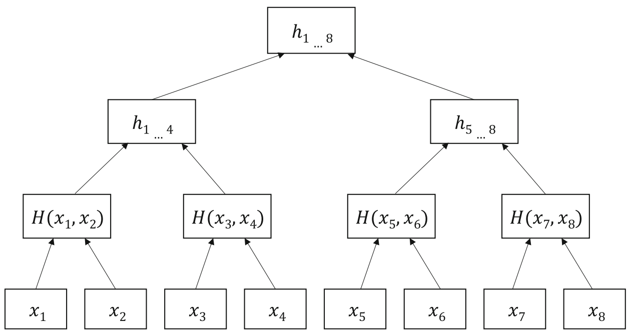


Fig. 2 Merkle tree

full nodes vote for a block which seems legitimate. It is hard for a malicious full node to get the most votes.

4 Proposed method

This paper proposes a new trust model named conditional trust and a PKI based on conditional trust in blockchain.

4.1 Conditional trust

4.1.1 Concept

Traditional centralized trust model of PKI assumes that users trust issuers. The basic concept of the proposed trust model called *conditional trust* is that each user presents a condition to trust a certificate. The condition is defined as a threshold value, the minimum number of verifiable signatures contained in a certificate, by the user. In other words, a user trusts a certificate in which the number of issuers' signatures verified exceeds the threshold. As shown in Fig. 3, Subject A submits his or her public key to full nodes so that they could make digital signatures on it. In general, one certificate is created, but wrong certificates can be generated by malicious full nodes or due to network transmission problem. In result, two certificates are produced and sent to user B. User B accepts a proper certificate whose number of verified signatures exceeds the threshold value (i.e., 5) set by himself or herself. Even though he or she does not trust any single centralized certificate issuer, User B trusts the above proper certificate containing multiple signatures made by full nodes.

It might be technically possible to attack all the issued signatures. However, it is not economical for attackers to launch such a high-cost attack. A malicious issuer needs to compromise a lot of the other issuers to add false certificates or remove true certificates.

We can find out the threshold value that makes our model more secure than previous systems. Therefore, users can use the certificates reliably by adjusting the threshold value even if they don't trust issuers.

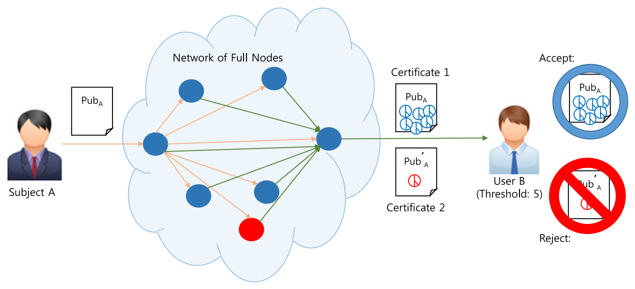


Fig. 3 Concept of conditional trust

4.1.2 Certification request for conditional trust

In the proposed trust model, each certificate must contain a number of verifiable signatures more than a condition which users present. To support this model, a method is needed to include a plurality of signatures in one certificate. To do this, we modified RFC 2986 [16] which is used to issue certificates from the traditional CA-based PKI.

The proposed method is as follows: (1) The subject generates a public key with his or her self-signature and requests a nearby issuer to issue a certificate. (2) The issuer broadcasts the public key with self-signature to other issuers. (3) All the receiving issuers verify the subject's self-signature and generate the issuers' signatures by using their own private keys. (4) All the issuers participating in the signature generation return the public key and their signatures to the broadcasting issuer. (5) Finally, the broadcasting issuer issues a single certificate containing all the signatures of participating issuers to the subject.

Step (2) and (4) in the above should be added to RFC 2986 to support our model. We note that the verification of multiple signatures in the resulting certificate can be done in the same way as RFC 2986 because the generated signatures do not affect each other.

4.1.3 Note

In this trust model, it is users who decide whether to trust the certificate or not. But it may be difficult in practice for users to determine the condition which affects the underlying PKI in terms of security. Therefore, we recommend that blockchain operators determine the condition by carefully considering the desired security level of the proposed trust model. The detailed analysis information to determine the condition is given in Sect. 5.

4.2 PKI based on conditional trust in blockchain

In order to use the proposed trust model, a number of issuers are involved as in shown Fig. 4. In addition, certificates issued by each issuer must be stored consistently. Therefore, we implement the proposed trust model on blockchain instead of central database.

4.2.1 Overview of the proposed PKI

This model has three steps such as initialization step, registration step and store step, which are briefly summarized as below.

The initialization step is performed on all full nodes when the PKI is first built. Afterward, this step can be performed whenever a new full node is added. This step is divided into two sub-steps. In the first step, full nodes

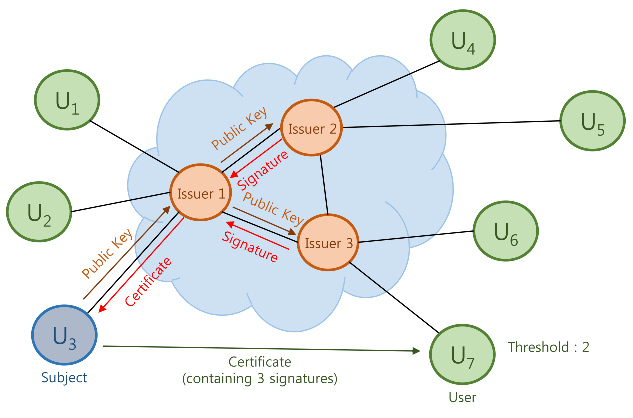


Fig. 4 Example of PKI based on conditional trust in blockchain

generate their certificates. It is because users need the certificate of full nodes when verifying subjects' certificates, or establishing encrypted communication channels with full nodes. The second step is performed to verify the generated certificates of full nodes by users.

The registration step is performed to register subjects' certificates to the full nodes. This step is further divided into three sub-steps. In the first sub-step, a subject sends its self-signed certificate to a full node. This is the same when a subject sends a public key to CA. In the second sub-step, all full nodes participate in forming the subject's certificate by putting their signatures inside the certificate. In the third sub-step, a user verifies the above-generated certificate of the subject.

In the store step, a new proposed consensus algorithm is performed to store all the generated certificates of the subjects by full nodes.

4.2.2 Implementation details

In this section, we explain the above PKI processes in detail by using functional diagrams and flowcharts.

(1) Initialization step

We assume that participating full nodes authenticate each other by exchanging their IDs and public keys by using direct delivery or over secure channel. The secure channel can be established using key establishment protocol like Diffie–Hellman [17], which is out of scope of this paper.

In the first step, full nodes generate their certificates as follows. Figure 5 shows the process of generating full node's certificates. First, a full node F_i generates a self-signed certificate and sends it to F_j, a full node except F_i. F_j generates a temp certificate through the private key of F_j. If the self-signed certificate is verified, then the full node F_j sends it to F_k, a full node except F_i and F_j. Note that F_k already has the public key of F_j due to our initial

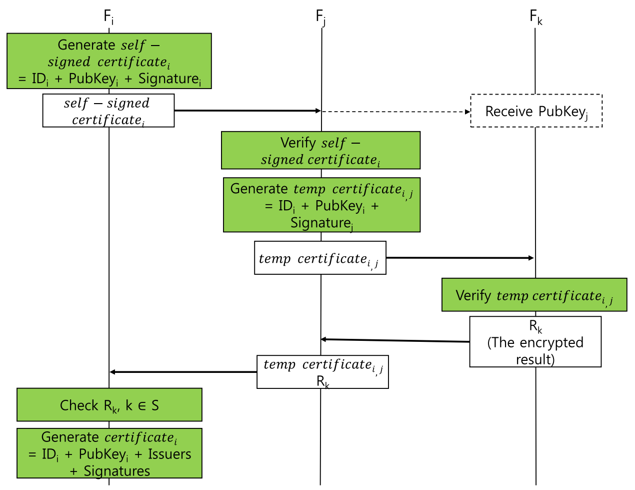

F_i
F_j
F_k
Generate self – signed certificate_i = ID_i + PubKey_i + Signature_i
self – signed certificate_i
Receive PubKey_j
Verify self – signed certificate_i
Generate temp certificate_i,j = ID_i + PubKey_i + Signature_j
temp certificate_i,j
Verify temp certificate_i,j
R_k (The encrypted result)
temp certificate_i,j R_k
Check R_k, k ∈ S
Generate certificate_i = ID_i + PubKey_i + Issuers + Signatures

Fig. 5 Process of generating full node's certificate

assumption. F_k verifies the temp certificate of F_j and sends the verification result encrypted by the public key of F_i to F_j. F_j repeats this verification process throughout all the other full nodes. These encrypted results are forwarded to F_i. F_i decrypts it and confirms that the signature on the temp certificate (that is, the F_j's signature) is valid. F_i repeats this verification process throughout all the other full nodes. If all the results are valid, F_i generates the final certificate of F_i because it means that the certificate of F_i is verified by the certificates of all the other full nodes. All the other full nodes should generate their final certificates by performing the above procedure in the same way as F_i does. In other words, full nodes cross-certify each other by digital signing and verification. Note that the full nodes can put more information in the certificate of F_i to meet X.509.

The second step is performed to verify the generated certificates of full nodes by users as follows. Figure 6 shows how a user verifies the certificates of full nodes. First, the user is required to get certificates from all full nodes on the blockchain web site. The user can compare the hash value of each certificate with the published hash value on the blockchain web site to thwart any possible man-in-the-middle attacks. If two hash values are different, the user discards the certificate, because the certificate may be modified by man-in-the-middle attack. Next, the user is required to verify all the certificates of full nodes whose signatures exist inside the received certificates as well as on the received certificates themselves of full nodes. For this verification purpose, a user checks that the number of verified signatures on the certificate is over the threshold value t. If it is greater than the threshold, the user stores the certificate. In this way, the user can trust certificates by the proposed trust model even if the user cannot trust full nodes.

(2) Registration step

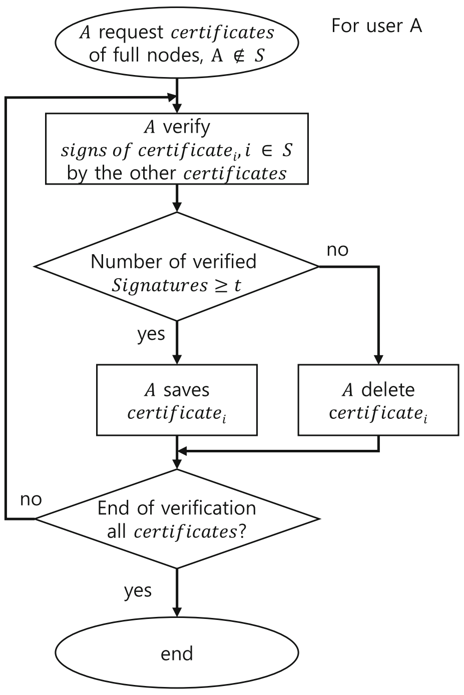

A request certificates of full nodes, A ∉ S
For user A
A verify signs of certificate$_i$, i ∈ S by the other certificates
Number of verified Signatures ≥ t
no
yes
A saves certificate$_i$
A delete certificate$_i$
no
End of verification all certificates?
yes
end

Fig. 6 Process of verifying the generated full node's certificate

This step is further divided into three sub-steps.

Figure 7 shows the process of generating a subject's certificate. In the first step, a subject A sends its self-signed certificate to a full node over a secure communication channel by using the full node's certificate. Then F_i authenticates subject A. Note that a subject is already authenticated from the full node when signing into the blockchain.

In the second step, all full nodes participate in forming the subject's certificate by putting their signatures inside the certificate as follows. F_i sends the received self-signed certificate to F_j. F_j generates a temp certificate through the private key of F_j. If the self-signed certificate is verified, then the full node F_j sends it to F_k. F_k verifies the temp certificate of F_j and sends the verification result encrypted by the public key of F_i to F_j. F_j repeats this verification process throughout all the other full nodes. These encrypted results are forwarded to F_i. F_i decrypts them and confirms that the signature on the temp certificate (that is, the F_j's signature) is valid. F_i repeats this verification process throughout all the other full nodes (including F_i). If all the results are valid, then F_i generates the final certificate of the subject because it means that the certificate of the subject is verified by the certificates of all full nodes. Note that the full nodes can put more information in the certificate of F_i to meet X.509.

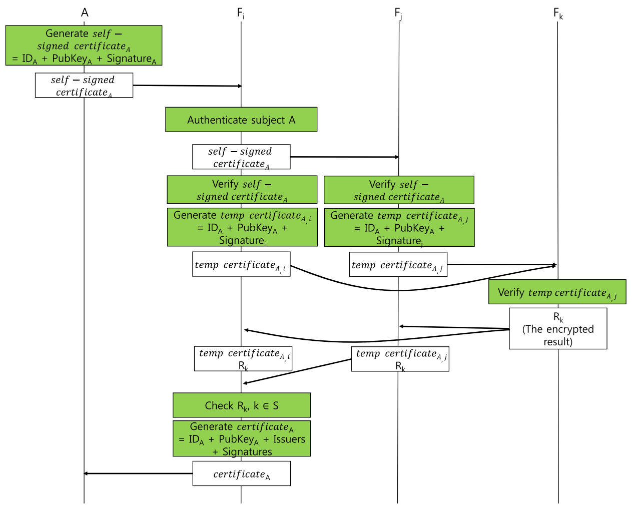

A
F$_i$
F$_j$
F$_k$
Generate self − signed certificate$_A$ = ID$_A$ + PubKey$_A$ + Signature$_A$
self − signed certificate$_A$
Authenticate subject A
self − signed certificate$_A$
Verify self − signed certificate$_A$
Verify self − signed certificate$_A$
Generate temp certificate$_{A,i}$ = ID$_A$ + PubKey$_A$ + Signature$_i$
Generate temp certificate$_{A,j}$ = ID$_A$ + PubKey$_A$ + Signature$_j$
temp certificate$_{A,i}$
temp certificate$_{A,j}$
Verify temp certificate$_{A,j}$
R$_k$ (The encrypted result)
temp certificate$_{A,i}$ R$_k$
temp certificate$_{A,j}$ R$_k$
Check R$_k$, k ∈ S
Generate certificate$_A$ = ID$_A$ + PubKey$_A$ + Issuers + Signatures
certificate$_A$

Fig. 7 Process of generating a subject's certificate

In the third step, a user verifies the above-generated certificate of the subject as follows. Figure 8 shows how a user verifies the certificates of a subject. A user checks that the number of verified signatures on the certificate of the subject is over threshold t by using the full node's certificates. If it is greater than the threshold value, the user stores the subject's certificate because it means that this certificate was authenticated by full nodes.

(3) Store step

The store step is performed to store all the generated certificates of the subjects by full nodes using the new proposed consensus algorithm as in the following three steps.

The first step is performed to generate a temp-block which includes newly issued certificates and a signature of each full node. Figure 9 shows how full nodes generate the temp-block. In order to confirm that each full node has the same certificates, each full node calculates and shares both the number of certificates n and the root hash of Merkle tree mk with the other full nodes. Full nodes select a pair of n and mk, which is the most popular among the entire set of

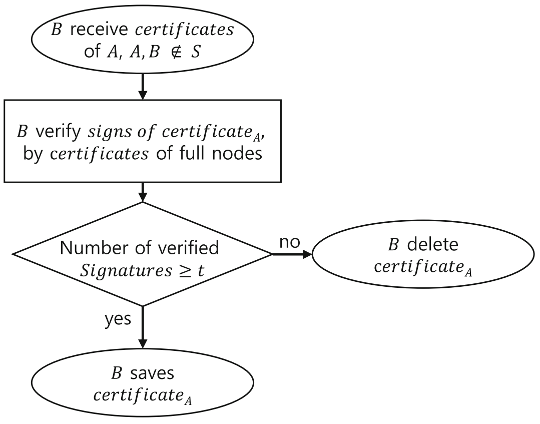

B receive certificates of A, A, B ∉ S
B verify signs of certificate$_A$, by certificates of full nodes
Number of verified Signatures ≥ t
no
B delete certificate$_A$
yes
B saves certificate$_A$

Fig. 8 Process of verifying the generated subject's certificate

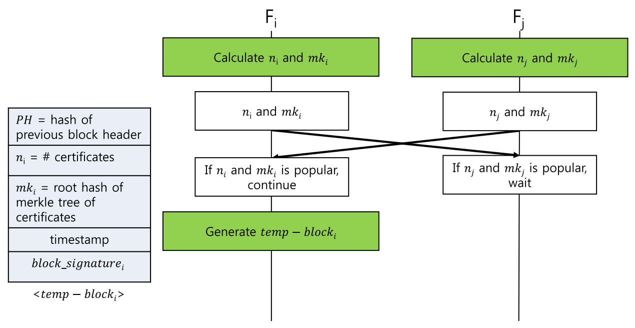


Fig. 9 Process of generating temp-block

pairs. Only full nodes which have all legitimate certificates (that is, full nodes with the pair of *n* and *mk*) can generate the temp-block. The full nodes which did not generate the temp-block start waiting.

The second step is performed to generate a block which includes certificates and signatures of all full nodes. In this step, full nodes must confirm the temp-blocks of the other full nodes and collect their signatures. Figure 10 shows how full nodes generate and share a block. First, full nodes must confirm that the signatures are created by the same pair of *n* and *mk* in temp-block and valid. Then, full nodes complete the generation of the block by adding the valid signatures to *block_signs* of their temp-blocks. Finally, they send the generated block to the waiting full nodes.

The third step is performed to chain the generated block to the previous block. Full nodes must confirm that the generated block is correct. Figure 11 shows how full nodes chain the generated block. In order to confirm the block is correct, all full nodes check that the number of verified signatures on *block_signs* of the block is over threshold *t*. If the number is greater than the threshold value, the generated block results in including the newly issued and correct certificates only by conditional trust. Finally, full nodes chain the new block to the blockchain. In this way, all full nodes keep consistency in the blockchain.

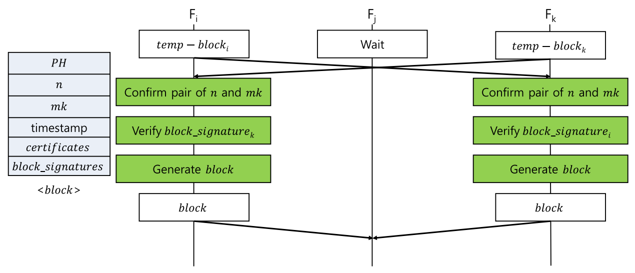
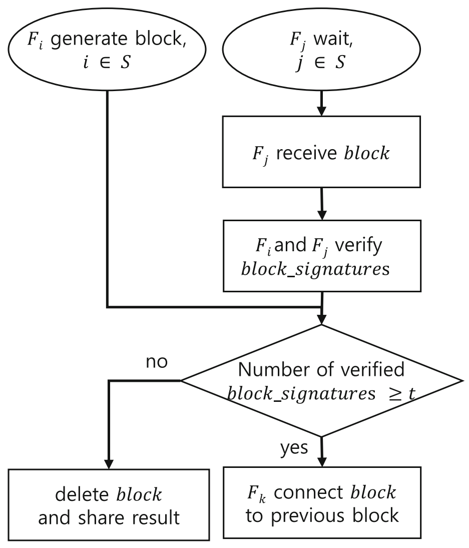


Fig. 10 Process of generating and sharing block



F_i generate block, i ∈ S
F_j wait, j ∈ S
F_j receive block
F_i and F_j verify block_signatures
Number of verified block_signatures ≥ t
no
yes
delete block and share result
F_k connect block to previous block

Fig. 11 Process of chaining the generated block

5 Analysis

5.1 Security

Property 1 *The security of conditional trust can be controlled by threshold t value.*

The process using conditional trust has two steps: one is for multiple nodes to make signatures, and the other for receiving nodes to verify the certificates. When verifying the certificates, the number of the verified signatures must be greater than threshold for getting trust.

Attacker can consider various attacks, which are defended as follows:

- Let us suppose that an attacker wants to implement full nodes more than *t*. But an administrator can make a policy to manage a list of full nodes including personal information because the manager of full node needs to have responsibility. The attacker cannot implement enough number of full nodes because duplication of personal information violates the policy.
- Let us suppose that the attacker wants to falsify packets transmitted between full nodes. However, since the important information (certificate or block, etc.) includes the signatures, the attacker cannot falsify these packets.
- Let us suppose that the attacker wants to delete a specific legitimate certificate when generating a block. But the attacker cannot generate that kind of block because legitimate full nodes use this certificate.
- Let us suppose that the attacker wants to send an abnormal certificate to deceive users and/or full nodes. However, the abnormal certificate needs to have multiple signatures generated by the other full nodes.

This requires the attacker to hack the full nodes more than t, which is very hard to achieve.

In other words, the attacker can attack only when the attacker has full nodes more than t. In this way, the security of conditional trust is controlled by threshold t value.

Property 2 *The threshold t can be set to implement a more secure system than existing systems.*

To compare the traditional CA and our system in terms of security, we first assume the invasion possibility for the node of each system and then calculate the abuse possibility of each system. Let the invasion possibility on a node in a CA be $a\%$. In other words, the attack possibility of an existing system is $a\%$. Let the invasion possibility of a full node in the proposed system be $b\%$. In Table 1, we statistically estimate the threshold values to prevent misuse by using binomial distribution for different invasion possibilities of attacker (e.g., 10%, 30%, 50% and 80%) at each node.

The proposed system is not compromised even if a node is invaded. An attacker needs to invade full nodes of more than threshold t to compromise the system. In other words, the attack possibility of our system is calculated by formula (1) because it is based on the attack or non-attack of each full node comprising the system. If formula (2) holds, the proposed system is secure than existing systems and achieves higher level of security as threshold t increases.

$$1-\sum_{i=0}^{t}\frac{n!}{i!(n-i)!}b^{i}(1-b)^{n-i} \tag{1}$$

$$a \geq 1-\sum_{i=0}^{t}\frac{n!}{i!(n-i)!}b^{i}(1-b)^{n-i} \tag{2}$$

Table 1 shows the minimum t values satisfying the condition that the proposed system works safer. These t values were calculated by R programming and formula (2). For example, for the value in bold, the invasion possibility of a node in CA is 0.05% and the invasion possibility on a full node is 30%. In the case that the number of full nodes is 30, the proposed system is safer if t is set by 0.633.

In addition, an administrator in CA should strongly protect all component nodes. Because, if one of the nodes are compromised, parts of existing systems related to the problematic node can be exploited. So, the administrator needs to pay more expensive cost. However, the proposed system is practical because it obtains security at a relatively low cost despite the fact that there are multiple full nodes.

Note that, if t increases, the number of certificates to be verified also increases. However, encryption or signature validation performed using once verified certificate does not take longer time. Therefore, the convenience is not much decreased in the proposed system.

For convenience, we may want to use lower threshold to keep the same security level. Table 1 shows that the more we increase the number of full nodes, the less t decrease.

Table 1 Minimum t values

t (%)	$b = 0.1$	$b = 0.3$	$b = 0.5$	$b = 0.8$	t (%)	$b = 0.1$	$b = 0.3$	$b = 0.5$	$b = 0.8$
$n = 10$					$n = 30$				
$a = 0.0005$	0.6	0.9	Over n	Over n	$a = 0.0005$	0.333	**0.633**	0.833	Over n
$a = 0.05$	0.4	0.6	0.9	Over n	$a = 0.05$	0.233	0.467	0.667	0.933
$a = 0.1$	0.3	0.6	0.8	Over n	$a = 0.1$	0.200	0.433	0.667	0.933
$a = 0.3$	0.2	0.5	0.7	1.0	$a = 0.3$	0.167	0.367	0.567	0.867
$a = 0.5$	0.2	0.4	0.6	0.9	$a = 0.5$	0.133	0.333	0.533	0.833
$a = 0.8$	0.1	0.3	0.5	0.8	$a = 0.8$	0.100	0.267	0.467	0.767
$n = 40$					$n = 50$				
$a = 0.0005$	0.300	0.575	0.775	1.000	$a = 0.0005$	0.28	0.54	0.74	0.98
$a = 0.05$	0.200	0.450	0.650	0.925	$a = 0.05$	0.20	0.42	0.64	0.90
$a = 0.1$	0.175	0.425	0.625	0.900	$a = 0.1$	0.18	0.40	0.62	0.90
$a = 0.3$	0.150	0.350	0.575	0.850	$a = 0.3$	0.14	0.36	0.56	0.86
$a = 0.5$	0.125	0.325	0.525	0.825	$a = 0.5$	0.12	0.32	0.52	0.82
$a = 0.8$	0.075	0.275	0.450	0.775	$a = 0.8$	0.08	0.26	0.46	0.78
$n = 100$					$n = 1000$				
$a = 0.0005$	0.22	0.47	0.67	0.93	$a = 0.0005$	0.133	0.349	0.553	0.842
$a = 0.05$	0.16	0.39	0.59	0.87	$a = 0.05$	0.117	0.325	0.527	0.822
$a = 0.1$	0.15	0.37	0.57	0.86	$a = 0.1$	0.113	0.320	0.521	0.817
$a = 0.3$	0.12	0.33	0.54	0.83	$a = 0.3$	0.106	0.309	0.509	0.808
$a = 0.5$	0.11	0.31	0.51	0.81	$a = 0.5$	0.101	0.301	0.501	0.801
$a = 0.8$	0.08	0.27	0.47	0.78	$a = 0.8$	0.093	0.289	0.488	0.790

Therefore, it is not necessary to implement too many full nodes to make our system secure.

5.2 Resource

We compare resource requirements in terms of storage, computation and communication with conventional CA systems.

- Storage: A certificate takes usually about 2–3 KB. The parts of issuer and signature in a certificate take about 70 Bytes. For $n = 30$, the size of each certificate increases up to 4–5 KB.
- Computation: The creation/verification process of a certificate in the proposed system takes n times more computations than that in the conventional CA system.
- Communication: When a full node issues a certificate, it requires four broadcast communications compared to one communication needed in the conventional CA system.

However, practitioners can ignore this resource issue. In the perspective of storage resource, the increase of certificate size is too small when considering the GB storage unit of mobile devices. We also note that the above computation and communication are required only at one time, respectively, in the certificate creation/verification process.

The proposed system has disadvantages in resource requirement per each certificate as above. However, we can save resource per each subject. A subject in the proposed system has only to save a certificate. In contrast, a subject in the conventional system needs to get as many certificates issued from relevant CAs. Hence, the proposed system is more efficient in this regard.

5.3 Blockchain storage

It is known there is a storage problem in the blockchain. However, this problem happens when we use blockchain for the purpose of ledger, which requires to save all data. On the other hand, the proposed method uses blockchain for PKI, where our blockchain can solve the storage problem by deleting block body (certificates) of old blocks for the following reasons. First, blockchain for PKI does not need to save all data because all certificates have a property named validity period. Second, we need a block header only when chaining a new block to previous block. Therefore, the proposed method doesn't have the storage problem.

6 Advantages

The proposed PKI has advantages as follows:

- There is no need to assume the existence of trusted third party as each full node is not a single trusted third party. But users can trust certificates if it has as many verified signatures as he finds trustworthy. Therefore, it is not mandatory for users to trust a single CA.
- There is a way for getting trust because users can trust certificates which have multiple verified signatures even though they do not trust every full node. Therefore, users can accept and use the certificates of the other communities.
- This is a decentralized system on blockchain with multiple full nodes. But each certificate does not need to be verified through the signatures of all full nodes. It is enough to verify signatures more than threshold. In other words, this system works even when some full nodes are broken. Therefore, users can use certificates without the single-point-of-failure problem.
- Integrity of certificates is supported. If an attacker invades a CA system, he can falsify all certificates. But even if he invaded full nodes in the proposed system, he cannot falsify certificates. To make the invasion successful, the attacker has to create valid hash values and signatures, which is very hard to achieve. Therefore, the attacker cannot falsify any certificates.

7 Conclusion

In this paper, we proposed a trust model and its implementation based on blockchain. The proposed trust model provides a method how users can trust the certificates of subjects outside the scope of user's trust. The theoretical security analysis shows that our system can work more securely than existing systems. Since the proposed model works under a multiple issuers, it naturally copes with a single-point-of-failure problem faced by traditional centralized PKI systems.

Acknowledgements This work was supported by the National Research Foundation of Korea (NRF) Grant funded by the Korea government (MSIP) (No. 2017R1A2B4001801).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Public key infrastructure. Wikipedia. https://en.wikipedia.org/wiki/Public_key_infrastructure. Accessed 22 Aug 2018
2. Cooper D, Santesson S, Farrell S, Boeyen S, Housley R, Polk W (2008) Internet X. 509 public key infrastructure certificate and

certificate revocation list (CRL) profile. Technical Report. https://tools.ietf.org/html/rfc5280. Accessed 22 Aug 2018
3. Abdul-Rahman A (1997) The pgp trust model, EDI-forum. J Electron Commer 10:27–31
4. Fromknecht C, Velicanu D, Yakoubov S (2014) CertCoin: a namecoin based decentralized authentication system 6.857 class project (Unpublished class project)
5. Fromknecht C, Velicanu D, Yakoubov S (2014) A decentralized public key infrastructure with identity retention. IACR Cryptol EPrint Arch 2014(2014):803
6. Loibl A, Naab J (2014) Namecoin, namecoin. info. https://namecoin.info/. Accessed 23 Aug 2018
7. Kalodner H, Carlsten M, Ellenbogen P, Bonneau J, Narayanan A (2015) An empirical study of namecoin and lessons for decentralized namespace design. In: Workshop on the economics of information security (WEIS), Citeseer
8. Axon LM, Goldsmith M (2017) PB-PKI a privacy-aware blockchain-based PKI. In: Proceedings of the 14th international joint conference on e-business and telecommunications
9. Qin B, Huang J, Wang Q, Luo X, Liang B, Shi W (2017) Cecoin: a decentralized PKI mitigating MitM attacks. Future Gener Comput Syst. https://doi.org/10.1016/j.future.2017.08.025
10. Blockchain. Wikipedia. https://en.wikipedia.org/wiki/Blockchain. Accessed 22 Aug 2018
11. Eyal I, Gencer AE, Sirer EG, Van Renesse R (2016) Bitcoin-NG: a scalable blockchain protocol. In: 13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16), pp 45–59
12. Nakamoto S (2008) Bitcoin: a peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf. Accessed 22 Aug 2018
13. Back A (2002) Hashcash-a denial of service counter-measure. http://www.hashcash.org/papers/hashcash.pdf. Accessed 20 Sept 2018
14. King S, Nadal S (2012) PPcoin: peer-to-peer crypto-currency with proof-of-stake (self-published paper)
15. Nikolakopoulos AN, Garofalakis JD (2013) NCDawareRank: a novel ranking method that exploits the decomposable structure of the web. In: Proceedings of the sixth ACM international conference on web search and data mining, 2013. ACM, pp 143–152
16. Nystrom M, Kaliski B (2008) PKCS# 10: certification request syntax specification version 1.7, technical report. https://tools.ietf.org/html/rfc2986. Accessed 22 Aug 2018
17. Diffie W, Hellman ME (1976) New directions in cryptography. IEEE Trans Inf Theory 22(6):644–654