A P2P Approach to issue, verify and revoke certificates over BLOCKCHAIN

Rohitangsu Das

MComp Graduate, NUS  
Singapore  
  
rohitangsudas91@gmail.com

Jayson Cena

MComp Graduate, NUS  
Singapore

jayson@cena.ph

Maddi Kamal Divya

MComp Graduate, NUS  
Singapore  
  
divyakamalmaddi@gmail.com

Hu Hui

MComp Graduate, NUS  
Singapore  
  
e0267822@u.nus.edu

**ABSTRACT**

A Certificate Authority (CA) is often defined as “trusted.” Despite a great deal of work to mitigate the insecurities in CA, CA misbehaviour continues to occur. There have been many examples showing the insecurities in a CA which shows the increasing probability of impersonations and MITM attacks. Unfortunately, CAs have shown to be prone to compromises and operational errors. These failures have occurred all around the world, and even Symantec, which has almost a quarter of the TLS-certificate market share, issued unauthorized certificates for Google and almost 2,500 unauthorized certificates. This is an active area of research, with many ideas and research in moving the PKI over the blockchain technology. We felt that placing the trust on a single CA entity provides a single-point of failure. In this paper, we address this issue by providing a distributed way to issue, verify and revoke certificate by leveraging the robustness of the Blockchain technology.

# INTRODUCTION

Parties are adapting TLS protocol to secure their connections. TLS and SSL, its predecessor, uses X.509 standard to authenticate remote parties by using the concept of chain-of-trust. Chain of trust is a security model where you trust the topmost party and you can only trust the other party by references that linked to the topmost, trusted party. But, this may lead to monopoly which resulted in other PKI models. In this paper, we will be proposing a stronger trust model by having multiple, trusted nodes to sign a certificate instead of a single entity certifying the validity of a certificate. We’ll also provide how the proposed system will do common tasks such as signing, verification, revocation and validation of ownership of the third party that requested for the signature from a root or CA certificate.

Aside from signing and verification, revocation is an important part of the public key infrastructure. Revocation is used to distrust a certificate that was previously signed by a root or any other CA certificate. Since the public certificate is already on the entity that requested for a signature, which in our example is Google, Inc., it will be almost impossible to stop google.com from serving the certificate. To solve this issue, the certificate that is used to sign the Google.com’s certificate will also have a field that has the data on where to get the list of certificates revoked. The list from the Certificate Revocation List distribution endpoints are regularly fetched by the browser and other TLS clients to verify that if the certificate that the other party provided is distrusted or not by its signer.

# RELATED WORKS

In view of the weakness in current PKI system, we studied some related works which aim to have improved design:

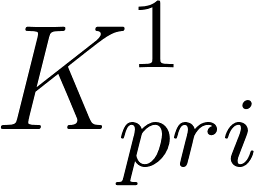
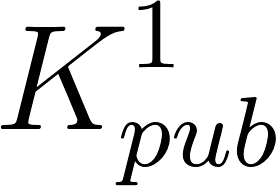
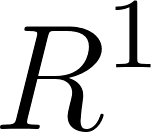
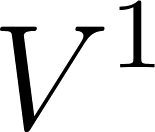
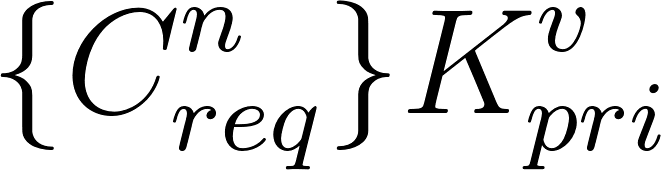
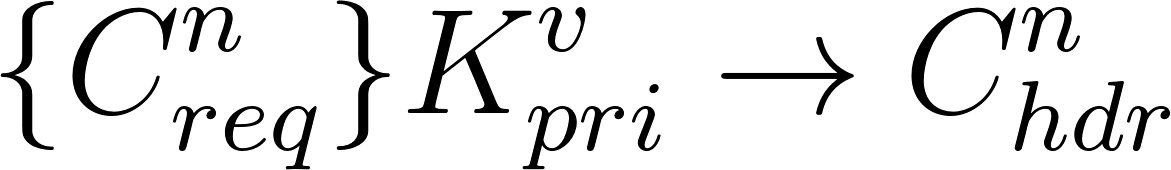
IKP (Instant Karma PKI): In this paper two main issues are addressed: what is a better way to incentivize CAs to improve security and report misbehaviour, and what can be implemented to automate the process to report unauthorized certificate. IKP is built on a blockchain based smart contract platform – Ethereum, and it consist of two main components, IKP Authority which maintains Cas’ information, and detector which reports suspicious certificate to IKP Authority. It uses RP (Reaction Policy) to automate reaction against unauthorized certificate and gives financial incentive to right-behaved CAs. However, IKP is still a centralized CA system.

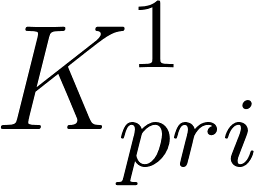
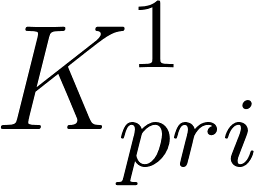
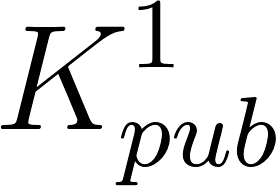
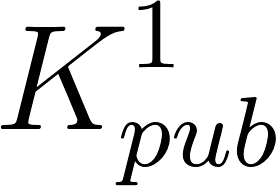
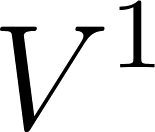
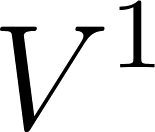
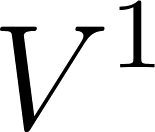
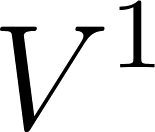
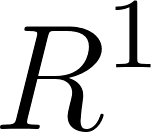
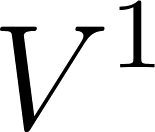
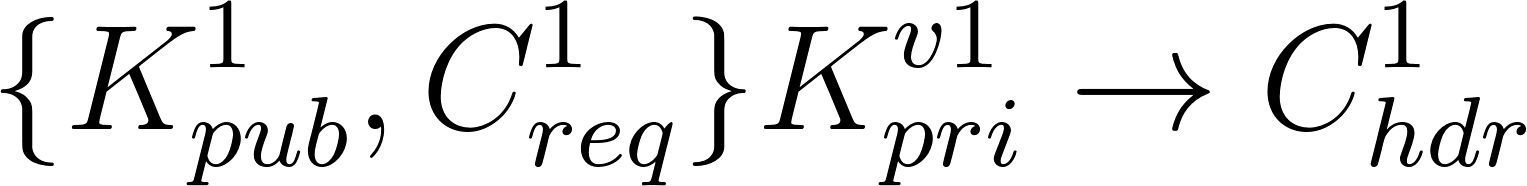
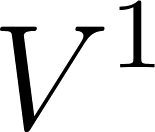
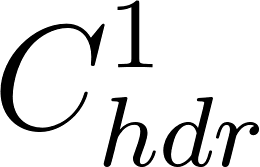
CertCoin/Namecoin: Certcoin is a system created to addresses the issue to offer guarantee of identity retention, which is that one cannot register a public key under an existing identity who has already registered. Certcoin is built upon Namecoin, while Namecoin is a cryptocurrency designed, and it is created to behave like decentralized Domain Name Server (DNS) for. bit addresses. Namecoin is used as a permanent ledger using it blockchain function to store Certcoin information, and Certcoin is designed with various versions which are using cryptographic primitives (digital signature), cryptographic accumulators (space-effective data structure for set membership verification) and distributed hash tables (provide faster way for public key lookup query) to achieve PKI functionalities.

Both solutions are effective but only addressed to partial of the equation, and we are going to design a system which is distributed, and addressing to most aspects including certificate signing, verification and revocation.

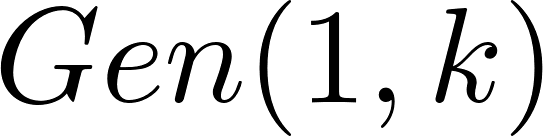
# ARCHITECTURE

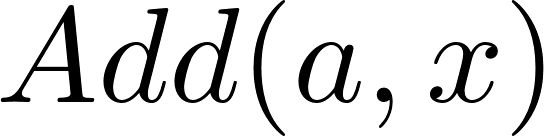
We propose a Decentralized CA consortium where every CA is considered as an Entity. Entities will have nodes to verify, validate and revoke certificates. Every entity should have at least a fixed number of minimum nodes to participate and it cannot exceed the maximum nodes. A new node can be added after verification of new node done by the existing nodes. Requester submits a certificate to the blockchain to verify and it is put on the current, open block. Once the block is closed, it will be visible to all nodes in the blockchain network. A browser/client will initiate the request and it will receive response with similar details as the original TLS specification along with updated certificate information after validation done by nodes and the transaction IDs on the blockchain where the certificate was signed. A browser/client can decide to accept or reject the certificate based on the number of signatures done by the validating nodes.

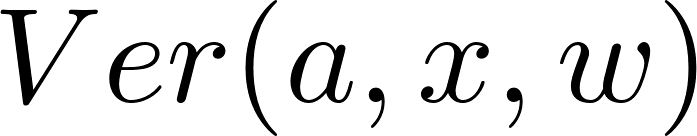
* 1. **Entities**
* Decentralized CA consortium - a group of organization or entity that wants to participate in the issuance, validation and revocation of certificates
* Organization - a CA/security/software company/organization that maintains the nodes that does the actual issuance, validation and revocation of certificates in the blockchain. Each organization in the consortium will have an equal number of nodes
  1. **Legend**
*  and  - private key and public key
*  - certificate signing request which contains identifying information such as domain i.e. google.com and hash of the public key
*  - any random string generated by the requester
*  - validating node, owned by an entity
*  - signature done on the certificate signing request using the validating node’s private key
*  - once the certificate signing request is signed by the validating node, it can now then be used for TLS handshake
  1. **Certificate Request**

1. Requester (e.g. google.com) creates or uses its own private key  and from the private key , the public key  is derived
2. Requester creates the certificate signing request which contains  and certificate header  which includes identifying information (such as domain name i.e. google.com), hash of the public key and the header. Certificate signing request
3. Requester submits the  to the blockchain and it is put on the current, open block (open block is the latest block in the blockchain where transaction can still be added. Once the time limit i.e. 10 mins is done, the block will close and is ready for hashing by the nodes in the blockchain network)
4. Once the block is closed, it will now become visible to all nodes in the blockchain network.
   1. **Certificate Validation**
5. The validating node  will look into the current block that is being mined and will search for any certificate signing request that is not yet signed by the validating node 
6.  will fetch the certificate signing request . Now, the validating node  will do the domain ownership validation by connecting to the domain over HTTP on a specific address (i.e. http://google.com/validation.txt) and see if the content of the response and  will be the same
7. Once it passes the validation, the validating node  will now sign the certificate signing request which will result into 
8. The validating node  will save the signed certificate  to the blockchain
9. The requester can check mined(confirmed) blocks until it finds the signed certificate on the blockchain
   1. **Certificate Revocation Request by the Owner**

In our modified PKI, the power of revocation rests with the owner of the certificate. Possible disadvantages of using a distributed list or a hash table to model CRL is the linearity in the number of certificates to maintain the storage. To ensure this, cryptographic accumulators can be used. Cryptographic accumulators can be thought of a set, to which one can add a member, and efficiently later whether the member is a part of the set. Formally, suppose A, is an accumulator object. It supports the following functions:

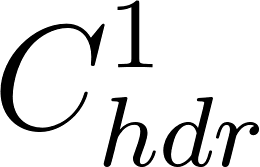
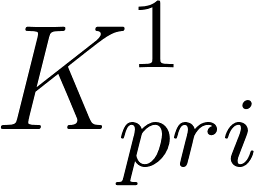
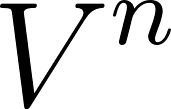
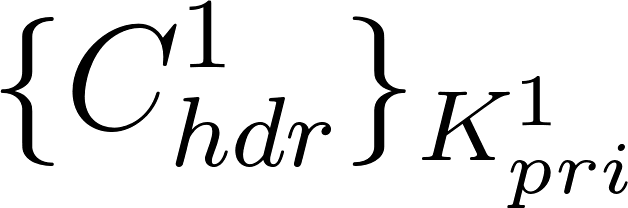
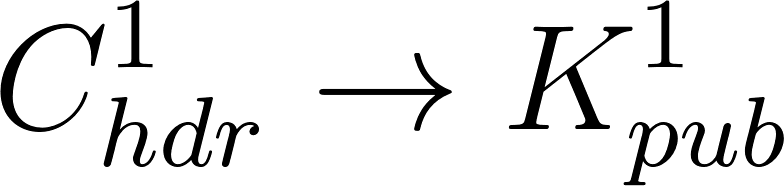
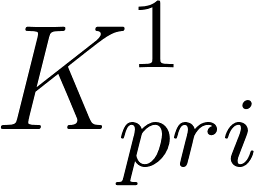
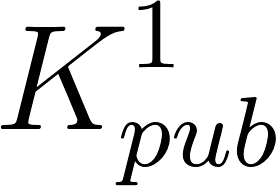
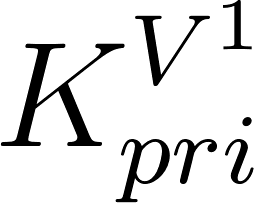
[](about:blank) -> Initializes the accumulator

[](about:blank) -> Add x to the accumulator

[](about:blank) -> Given the x, a, w, function returns whether the x is in a.

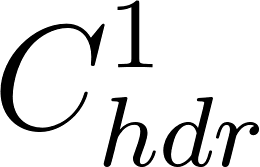
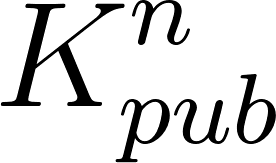
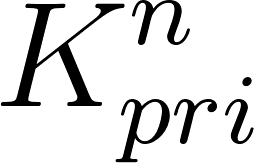
(w: witness)

A disadvantage of an accumulator is in maintaining and updating witness each time a new value is added to the accumulator. However, if there are many additions to the accumulator, there would be that many witness updates. However, it would be great if the witness updating can be bounded by logarithm of the additions of members to the accumulators. Maintaining a list of Merkle Tree accumulators is a possible solution, to reduce the number of witness updates.

1. Requester signs the certificate  using the certificate’s private key  and puts the revocation request to the blockchain
2. Once the block is closed and it became visible to all nodes in the blockchain network, validating nodes  can now fetch the revocation request 
3. At the same time, it shall fetch the latest accumulator which shall also be stored on the blockchain and are pinned together to avoid traversing the blockchain again.
4. The validating node will then extract the public key from the certificate  and will then use the public key to validate if the private key  used to sign and the public key  comes from the same private key.
5. Once it passes the check, the validating node will now sign the certificate  using the validating node’s private key  and tagged as revoked (revise: use txid instead of the actual cert)
6. Get the cryptographic hash  of the certificate and add it to the accumulator which we shall call as the revocation accumulator.
7. The node shall compute the witness of the revoked certificate and gossip it to the other nodes in the consortium. This way all the nodes, can store the witness of the revoked certificate.
8. The node shall add the revoked transaction to the blockchain.

However, as the accumulators are updated, the witness of the revoked certificates can get outdated. So, the witness shall be outdated according to the paper which we are not describing at this moment

* 1. **Certification Verification by the Browser**

1. Browser establishes a TCP connection to the web server by sending TLS ClientHello packet along with the certificate verification request to the blockchain nodes to verify.
2. Server replies with TLS ServerHello together with other details, the same as the original TLS specification along with . The blockchain node fetches the latest accumulator from the blockchain and use the locally stored witness to verify the membership from the accumulator.
3. Verify if the certificate is revoked by verifying if the (witness, certificate) leads to a successful membership, then the certificate is revoked.
4. If not successful, then it is not a part of the list.
5. If the certificate is not considered as revoked, actual validation will continue
6. Since the certificates are signed by multiple validating nodes, certificates will come with multiple transaction IDs that points to where in the blockchain the signatures (per validating nodes) are stored
7. Once the transaction IDs are located, the signature and certificate are verified one-by-one by:
   1. Retrieve the signature based from the transaction ID
   2. Verify if the certificate and public key of the validating node is trusted
   3. If trusted, increment the number of successfully validated signatures
8. If  where  is the browser’s lower-limit for the number of validated signature, then continue with the TLS handshake, else fail.
   1. **Validating Node Membership - Bootstrap the Network**
9. Each entity will generate a certificate with  attached and derived from .
10. Each entity will then request the remaining validating nodes for signature.
11. Signing of the certificate by the other validated entities will not happen on the network but rather, it will be done on the physical world where each entity will verify each other.
12. Once the identity of the entity is verified, they will sign each other’s certificate.
13. Once a certificate got all the signatures from all validating nodes, that’s the only time it can connect to the network.
14. Now, the entity can add nodes. A node submission request is generated.
15. The other nodes shall verify, whether the request is submitted by a valid entity, and whether it is within the maximum limit.
16. If all the nodes successfully validate the request, then the node is submitted and node membership metadata is updated.
    1. **Validating Node Membership - Joining the Network**

If entity joins the network for the first time, the number of nodes that it has to provide should be at least the lower bound. If an entity has already joined the network, then one cannot add more than a fix upper bound. All such events are gossiped to the entire network, to prevent addition of any invalidated node.

* 1. **Validating the Registration of the Domain**

1. It should be noted that when a blockchain node reads all the certificate issuance request from the current blockchain node, it should verify whether the same domain hasn’t been already registered.

2. Suppose a domain, D(some.xyz.com), creates a certificate and signs the certificate with its private key. This request is added to the blockchain, and to add such an issuance request, some fee is required on the part of the domain. This fee would then be distributed to the CA’s that is trying to validate the certificate request.

3. So, when this certificate is already validated and when the domain, tries to get another “similar” certificate. This time, the blockchain nodes should detect that there is already such a certificate registered for the domain, and ignore the certificate request.

4. Although, there are use cases where a company might need multiple certificates at any given time, we thought it is always better to perform a check to remove redundancy.

# REQUIREMENTS AND ASSUMPTIONS

* 1. **Domain Nameserver Security**

It is assumed that the domain’s nameservers are secure and no malicious DNS records can be added on the domain’s nameservers.

* 1. **Client Validating the Certificate is Secure**

Once the client is compromised, the adversary doesn’t need to find and abuse vulnerabilities on PKI. Adversary can just add a rogue CA to the computer’s trusted key store, get the unencrypted data from the memory, get the shared key to decrypt TLS communications, etc.

# THREAT MODEL

* 1. **Compromised Validating Node**

When a validating node is compromised, adversary can use the validating node’s private key to sign a rogue certificate submitted by the adversary. On our proposed decentralized PKI, the client will require signatures from more than 50% of total validating nodes in the network. As the client who

* 1. **DNS Hijacking**

Before a validating node signs, a certificate, it needs to confirm the ownership of the domain. This is done through HTTP request which is explained in part [3.4](https://docs.google.com/document/d/15xb6_DIk7ultEiZ9QztzYDliEKDPkK-t2kIWk_cuRVk/edit#bookmark=id.m0ejmmud6bjs). For an adversary to successfully obtain a signed rogue certificate, the adversary needs to submit a certificate signing request for the rogue certificate and a nonce to a validating node. The adversary will then do a DNS hijacking on a validating node, update the DNS record and point the HTTP server to the adversary’s HTTP server which contains the nonce which is used to prove domain ownership.

To prevent this type of attack, we can require the requester to fully support DNSSEC so that no matter the injection is, false DNS responses will not be accepted by the validating node.

Another security model of this paper that made this kind of attack less effective is that the adversary needs to hijack more than 50% of the validating nodes to be able to get the required signature from different validating nodes.

# EVALUATION METRIC

Although, our one of the main goal was to implement the system and deploy it in a controlled and test environment and then try to create adversary attacks and evaluate the resilience of the system. However, our original goal to implement smart contracts on Ethereum was met with a setback as we were trying to do more processing for a single transaction than allowed by the gas limit. Our new goal is to fork the implementation of the blockchain, and create an abstracted system tailored to our needs. These we are however keeping for future work. Theoretically, as said earlier, more computational power has to be used to achieve complete hijacking of the system. Secondly, we haven’t prepared a incentive model which would prevent a single domain, to continue send certificate request, because in that way, a single entity might occupy a large share of the storage space, as well as, much time will be spend by the CPU’s on request for a single domain holder. We have thought of a way, to ask for coins on each request, however, this we have kept for future work.

# REFERENCE

1. <https://www.ieee-security.org/TC/SP2017/papers/290.pdf>
2. <https://courses.csail.mit.edu/6.857/2014/files/19-fromknecht-velicann-yakoubov-certcoin.pdf>
3. <https://bitcoin.org/bitcoin.pdf>
4. <https://eprint.iacr.org/2015/087.pdf>
5. <https://eprint.iacr.org/2015/718.pdf>