Some Information the BB-PKI Implementation

Guidelines on how to test the code:

1. **Using Remix, Ropsten and Etherscan.io**

   https://remix.ethereum.org/#optimize=true&runs=200&evmVersion=null&version=soljson-v0.5.16+commit.9c3226ce.js
   https://etherscan.io/

2. **Via GitHGub**

- Clone the repository using the command, "git clone github.com/danbaba1882/bbpki.git"
- Install the applications development dependencies via "npm install"
- Set up your VPN
- Deploy your smart contract to your VPN
- Generate your contract byte code and replace it with the one in the *bbpki.json*
- Replace your contract address with the address of your deployed contract
- Use the IP address of your VPN as your RPC provider
- Run the *nodejs* application using the command *"node bbpki.js"*
- Call the function you need to test by performing a get request via the URL that serves the function via your web browser.

The bbpki model of managing certificates has four main components namely;

1. Nodejs Application: The nodejs application has the functions, libraries and modules that are responsible for the verification of a **CIR** by the RAs, signing of a certificate by the CAs, creating and saving a copy of the certificate and a successful inclusion of the transaction into the ethereum blockchain, revocation and certificate update, certificate client verification. It contains the modules for generating a public-private key pair, *bls multisignature* system and *Eth proof* of existence. Our nodejs application serves as a tool that facilitates the interaction and communication of the application functions and methods with the blockchain by using a JavaScript module web3.js via a VPN RPC provider. Our nodejs application is contained in the file *bbpki.js*

2. Smart contract: The smart contract is a piece of solidity code that is compatible with the Ethereum blockchain. The smart contracts contain the functions and the logics that are responsible for creating a copy of a certificate, revoking, updating and querying a certificate on the Ethereum blockchain. Our smart contract code is in the file *bbpki.sol.*

3. Our Ethereum blockchain: a virtual machine that runs on a network of distributed nodes that allows and facilitates the deployment and execution of smart contract logics by creating a transaction which is validated by a network of miners. It is also responsible for saving the created certificate on a network of distributed ledgers.

4. The virtual environmental features and setup (Local/ remote network from your side).

STEPS AND PROCEDURES INVOLVED IN THE WHOLE PROCESS

A domain owner creates a certificate signing request (CIR) generating a public/private key pair using the JavaScript crypto module through a number of trusted RAs (RAs≥ 3 and RAs ≤ 10) , after an automatic verification and vouching of a domain owner CIR by the RAs, Using an *out of bound secure communication channel* the certificate gets signed by a number of CAs (CAs≥ 3 and CAs ≤ 10) utilizing the concept of BLS-Multisignature system. After a verification of a CIR request by RAs a function named ***signCertificate*** from the file bbpki.js gets called via a *get request API*, on successful request, the certificate gets singed by using a *JavaScript bls multisignature library node-bls12-81*.

After a certificate has been signed, it is then forwarded into our deployed Ethereum smart contract with the following certificate parameters version, subject name, serial number, *multisignatures*, certificate signature, certificate public key, signature verified, certificate status, issuer, expiry *via web3.js*. A function ***issueCertificate*** on our smart contracts accepts the certificate parameters and saves a copy of the certificate in a mapping certificates. A transaction is being created on the process, upon a successful execution of the transaction, the following occurs sequentially;

- The certificates get included into the Ethereum blockchain returning a transaction receipt.
- The certificate is being returned to the RA off-blockchain by using an *out of bound secure communication channel.*
- The log included within our transaction receipt is being subscribed to getting the respective values of the block number, transaction hash and the block hash and gets included into the certificate returned to the RA.
- The RA finally issues the certificate to the owner in order to facilitate a secure connection between users and the certified domain server.
- At the point where the client is trying to establish a secure connection for users, it requests the issued domain certificate in order to check for the proof of existence and the proof of the certificates status and validity and thus does this in the following ways;
- The client uses the serial number of the given certificate to check for the existence of the certificate in our smart contract on the ethereum blockchain, if the certificate exist it is returned to the client, the client compares the values of the items of the returned certificate from our smart contract upon query with that of the certificate presented by the domain server, if found to be the same, the client then checks the validity of the certificate and its status in order to know if a certificate is active, expired or being revoked. If the status of the certificate is found to be active then a secure connection is established, if a certificate is expired and not revoked, an automatic request via the *nodejs* ***revokeCertificate*** function is made to the smart contract ***revokeCertificate*** function to revoke the certificate and therefore the secure connection establishment by the client fails, also if a certificate is found to be revoked the secure connection establishment by the client fails
- The client uses the block number of the transaction of the given certificate generated during its inclusion in the ethereum blockchain via our smart contract to query the ethereum blockchain and return a transaction, the client then uses the transaction

hash and block hash of the given transaction to verify the certificate by utilizing the concept of the proof of existence and proof of freshness by using a JavaScript module eth-proof. Passing the transaction hash and the block hash of the certificate as arguments into the ***txAgainstBlockHash*** function of the eth-proof module, the client then verifies the certificate via the block header returned from the ***txAgainstBlockHash*** function, thereby proving its existence on the Ethereum blockchain, validity and its status.

- At the point where a certificate is bound to be revoked, a domain owner, CA and any other authorized entity can call the ***revokeCertificate*** function in our smart contract by passing the certificates serial number upon a successful revocation, the function returns false and the certificate is deemed revoked.
- At the point where a certificate needs to be updated, the authorized entities call the ***issueCertificate*** function of our smart contract by passing the updated parameters as arguments, this results in the creation of a new transaction and the updated certificate gets included on the blockchain. On a successful update, the updated certificate is returned to the domain owner.

In bbpki.js package file in the BBPKI folder: Some features or routes has been inserted ***(e.g server.get (sign))*** in order to run all functionalities after deploying the (1.) smart contract and 2) Web3 server localhost:3000)

1. Smart contract can be deploy either using testnet Remix, Ropsten and etherscan.io or using remote/ virtual local network which you may be created from your side.
2. Web3 local host server: localhost:3000

All functionalities
For signing certificate: Localhost:3000/signCertificate/X.com
For cert revocation: localhost:3000/revoke-certificate/serialNumber
For client verify cert: Localhost:3000/client-verify-cert/serialnumber/blocknumber