ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Integrate Collective Certificate Management on Skipchains and on cross platform mobile application

Claudio Loureiro

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Semester Project

June 2018

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Linus Gasser
EPFL / DEDIS

Contents

1 Introduction **3**
1.1 Goals and motivation . . . 3
1.2 Background . . . 4
1.2.1 Cothority . . . 4
1.2.2 Cisc . . . 4
1.2.3 Let's Encrypt . . . 5
1.2.4 CPMAC . . . 5
1.3 Previous work . . . 5
1.3.1 Collective certificate management . . . 5
1.3.2 Cross platform application . . . 6

2 Main work **6**
2.1 Collective certificate management into Cisc . . . 6
2.2 Cross platform application development . . . 8
2.2.1 General improvements in Cisc . . . 8
2.2.2 Collective certificate management . . . 10
2.3 Limitations . . . 11
2.3.1 User name on the CPMAC . . . 11
2.3.2 NativeScript and Node.js . . . 11
2.4 Future Work . . . 11
2.5 Conclusion . . . 12

3 User guide **12**
3.1 Cisc guide . . . 12
3.1.1 Initialization . . . 12
3.1.2 Running collective certificate management . . . 13
3.2 Cross platform application guide . . . 14
3.3 Initialization . . . 14
3.4 Accessing the certificate module . . . 14

1 Introduction

In today's Internet, most of the communication needs to be encrypted to ensure confidentiality and integrity of the data. Before a secured communication channel can be open between two devices (for example, between Alice and Bob), they need to exchange their public keys. Those permits Alice to send encrypted messages to Bob ensuring that he will be the only one who can decrypt it and vice versa. The problem with this exchange is that Bob needs to be certain that the public key that he receives is indeed Alice key. For example, a man-in-the-middle attacker can usurp the identity of Alice and sent his public key to Bob. Because Bob believes that he gets Alice key, he will send her confidential encrypted messages using the attacker's key. Thus, the attacker will be able to read those messages using his private key. One solution that has been developed to prevent this type of attack was to create Certificate Authorities (CAs). Those entities deliver certificates that permit to prove that a key belongs to the appropriate device. Typically, a web server owner request a certificate from a CA so that he can prove to its clients that they use the right public key to communicate with him. Typically by now if a browser wants to establish a secure channel (using HTTPS) with a web server need to first get its certificate.

Then with this configuration new problems raise. How can we ensure that a particular CA is trustworthy? A CA could be attacked and his private key could be stolen. Moreover, Certificate Authorities could be improperly configured and thus deliver inappropriate certificates. Hence, users could accidentally or maliciously request a certificate for a domain that they don't own. One solution to solve this problem would be to let decentralized protocols involving multiple entities manage certificates. In this configuration, the certificate can be signed by multiple entities and then being considered as more trustworthy.

This project will be mainly based on this previous solutions. We will in a first place expand a previous student project which based its implementation in the following research paper named *Keeping authorities "honest or bust" with decentralized witness cosigning* [1] co-written by Bryan Ford et al. Our main task will be to integrate this collective certificate management in the current Cothority framework (explained in the later sections).

As this functionality is for now completely implemented only in the back-end, it is difficult for casual users to use these technologies without investing ample time in the hands-on process and understand the order of the different actions to realize a specific task. Therefore in a second place our project will consist in giving a front-end implementation of the collective certificate management.

Another previous student developed a cross-platform mobile application that implemented the core applications of our Cothority framework. We will benefit from this latter and integrate our front-end implementation to the application.

1.1 Goals and motivation

As stated previously our project will be divided in two parts. The first part consists in integrating into the current Cothority framework and improving the previous student's work on the collective certificate management [2]. The second part will involve the implementation in the already existing cross-platform mobile application of a functionality that permits to manage the different stored certificates.

Since the Cothority framework has been updated these last months, the previous code is no longer compatible thus an update was needed. Furthermore some other features could be added to turn the functionality more robust. The different commands and functions of the certificate management will be explained in the next sections.

At the moment the main applications of our framework are only available in the back-end. We felt the need as user to have a better interface to deploy the different decentralized protocols and applications. Thus the idea of having a smart phone application seemed logic to create such a front-end interface.

1.2 Background

In this section, we will describe our main framework and then we will focus on the main structures needed for our project.

1.2.1 Cothority

The collective authority (Cothority) project provides a framework for development, analysis, and deployment of decentralized, distributed (cryptographic) protocols. We can generate individual servers called by cothority servers or conodes. The Cothority project is developed and maintained by the DEDIS lab. Also, the Cothority project defines applications that can be run on those conodes [3].

- **Status report**: report the status of a conode
- **Onchains-secrets**: hides data on a blockchain and adds an access control to it
- **Proof of Personhood**: create a PoP party to distribute unique cryptographic tokens to physical people
- **E-voting**: E-voting following Helios to store votes on a blockchain, shuffle them and decrypt all votes
- **Cisc**: stores key/value keypairs on a skipchain, has special modules for handling ssh-keys, storing webpages and requesting certificates from letsencrypt

This project will focus more specifically in one of these applications called Cisc.

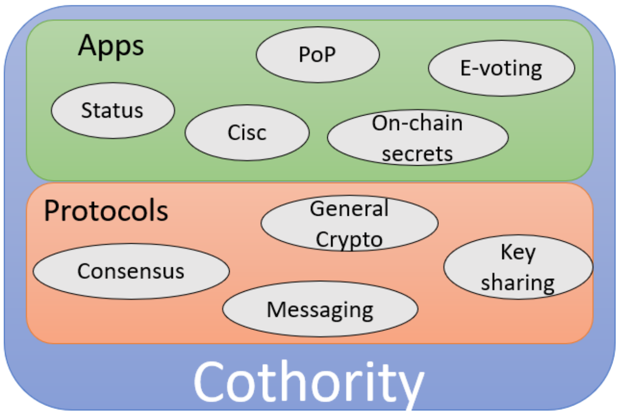


Figure 1: Cothority framework

1.2.2 Cisc

Cisc stands for Cothorithy Identity Skipchain. The main goal of this application is to provide a simple way to store data. The storing is based on the blockchain principle mostly used by crypto-currencies as the most famous one: the Bitcoin. The main advantage of the blockchain is that the infrastructure is decentralized. Therefore every user can have a local copy of it. Thus it is easy for a user to add a block. And inversely it is hard to add a false block since every user will figure out the problem coming from this one and they will ignore it. In addition this structure allows peer-to-peer and business-to-business without the need of a third party. For example when doing money transaction the third party is the bank. In the actual Cothority framework we call this chain a skipchain (based on the Chainiac-paper [4]). Our Cisc application, based upon these skipchains serves, a data-block with different entries that can be handled by a number of devices who propose changes and cryptographically vote to approve or deny those changes. Different data-types exist that will interpret the data-block and offer a service. Besides having devices that can vote on changes, simple followers can download the data-block and get cryptographically signed updates to that data-block to be sure of the authenticity of the new data-block [3].

1.2.3 Let's Encrypt

Let's Encrypt is a certificate authority that was launched in 2015 by the Internet Security Research Group (ISRG). The advantage of this Certificate Authority is that certificates are delivered for free and automatically. Let's Encrypt provide a protocol called Automatic Certificates Management Environment (ACME) which allows automatic communication and certificates requesting between a web server and a CA. However before giving the certificate Let's Encrypt verifies that the person requesting the certificate is the owner of the web server. The validation is realized through a challenge where the client needs to put a certain file on a certain place. This CA is used in our project due to the facility of requesting a certificate and it is easy to implement our challenge verification in our web server.

1.2.4 CPMAC

CPMAC stands for Cross-Platform Mobile Application. As the name indicates, this app is compatible with Android and iOS. The chosen framework to create this app is called NativeScript [5] and the used language is the JavaScript. It allows to have a real native app on both mobile operating systems. The main purpose of the app is to have user-friendly interface in order to complete Cisc or PoP tasks. To implement such an app we also need some "technologies" such as web sockets to send data back and forth to the server or cryptographic algorithms such as Schnorr signature or elliptic curve cryptography. The app structure can be summarized with the following picture taken from the previous student's work [6].

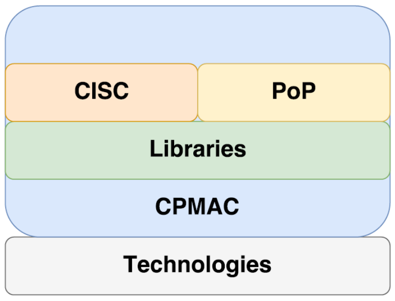


Figure 2: CPMAC structure

1.3 Previous work

In this section we describe the work already done on the collective certificate management and on the cross-platform application.

1.3.1 Collective certificate management

We can find here the different methods already implemented concerning the collective certificate management.

- **Request**: this functionality request a certificate to Let's Encrypt for a given domain. As explained earlier, the client needs to prove his authenticity by completing in our case a HTTP-challenge for our web server. Once we get the certificate this one goes to data proposal and the different users have to vote positively to store this certificate in the Skipchain.
- **Renew**: this functionality permits a client to renew a given certificate stored in the Skipchain. By doing this the expiration date of the certificate is extended by three months.
- **Verify**: this functionality verifies if a given certificate is valid.
- **Retrieve**: this functionality permits to retrieve the PEM file of a certificate stored in the Skipchain.
- **Add**: this functionality permits to add an already existing certificate to the Skipchain
- **List**: this functionality lists every certificate stored in the Skipchain.
- **Revoke**: this functionality revokes a certificate and suppress it from the Skipchain

For further details about the implementation of these methods refer to Robin Berguerand's work [2]. Our task will be to re implement these methods into the current Cisc framework.

1.3.2 Cross platform application

As stated earlier, the mobile application implemented the Cisc and the PoP functionalities from the Cothority framework. Again I will briefly describe these two applications.

PoP App
The PoP app of the Cothority framework is used to generate and verify proof of personhood, which indicates that a user is a human being. Personhood is proven through stating that a specific person was at a precise location at a particular time and thus that this user is not a bot or any other kind of human-simulating program. This application is already implemented in our mobile application. PoP is used to display the data shared between attendees and organizers, it shows the list of all fetched final statements and generated tokens. The Org tab contains all the functionalities needed by the organizers of PoP party. For further information you can refer to previous student's work : Cédric Maire and Vincent Pétri [6].

Cisc App
When navigating through the Cisc in the mobile app, we access a page where we need to establish a connection to a Skipchain. Once connected we have a new interface (UI) with three tabs: QR, Home, Data. The QR tab displays a QR code of the connected skipchain. The second tab is home, it shows the data stored in the skipchain as well as the proposed data. Finally the third tab permits the user to add a key value by pressing the respective button.

2 Main work

In this chapter we will describe the work realized on the collective certificate management and on the cross-platform mobile application.

2.1 Collective certificate management into Cisc

First, we describe the changes and/or improvements realized from the previous functionalities explained in. In addition to the explanation of the functionality of each command previously stated in 1.3.1 , we give the parameters for our CLI of the commands composing our actual Cisc certificate management.

- `cert request` : cert request domain-name cert-dir www-dir [Skipchain-ID]

- `cert list` : cert list [command options] [Skipchain-ID]
  OPTIONS:

  –verbose, -v Display the fullchain certificate

  –public, -p Display the public certificate

  –chain, -c Display the chain certificate

- `cert verify` : cert verify cert-key [Skipchain-ID]
- `cert renew` : cert renew cert-key [Skipchain-ID]
- `cert revoke` : cert retrieve [command options] key [Skipchain-ID]
- `cert add` : cert add domain path [Skipchain-ID]

In the following parts, I explain the main improvements acted to these commands.

Multiple Skipchains changes

In the previous implementations, the proposed code was only compatible with a framework containing one skipchain. In the next Cothority framework it was possible for a user to create and join different skipchains. Thus one has to manage in which skipchain they want the certificate to be stored or to be revoked. Therefore now when realizing a certificate operation, a skipchain ID has to be given if the user is connected to more than one skipchain.

Request certificate changes

When running the request command with previous implementation, the obtained certificates and the private key generated from registering into the ACME server were stored in the current folder (in addition to being stored in the skipchain). Thus if we executed the command in the `www` folder, the private key would be deposited in this folder and therefore everybody accessing for example our Cothority server `https://cothority.net/privkey.pem` could get access to the private key. To solve this issue, the user needs to explicitly give the path to the folder from where we wants to store the certificates and the keys. In addition, he also needs to give the path to the `www` folder (to complete the http-challenge) and the domain name. Also to help the user we store the path leading to certificates (which can be displayed with the certificate command list). Finally to make more robust the process of requesting the certificates if an error happens in the process. All the created files (certificate or private keys) are deleted. The process of requesting a certificate did not change from the previous implementation and works as follows. At first, the program creates a client for the ACME by querying the Let's Encrypt server resource directory. Then, the program generates a RSA key-pair that is sent to the server to register (see Figure 3 inspired by previous student project) [2].

Renew certificate changes

As for the request certificate command, when renewing a certificate, the physical certificate is stored in the folder from where the command is executed. Since we saved the certificate path (from the request command) the renewed certificate will automatically replace the old one. If for some reason the path of the certificate was not defined the file would be stored in the current directory.

Retrieve certificate changes

The command retrieves the fullchain and the public certificate of the given domain. The files relative to the certificate will be placed in the current directory. We can also specify to store in a special folder by giving a flag and giving the desired path.

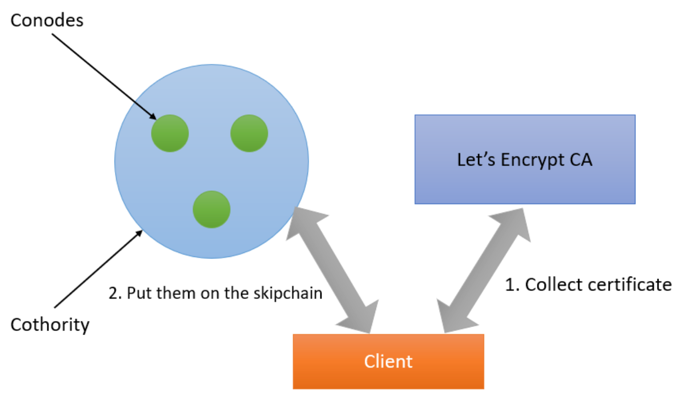


Figure 3: Requesting structure

Revoke certificate changes

To revoke a certificate the user needs to specify the path to the private key generated when the certificate was requested. In the previous implementation we needed to execute the command in the same directory as the private key.

List certificate changes

This command now only shows the key of the certificate, in other words it displays the domain name for which the certificate was requested. In addition to this, it also prints the expiry date of the certificate as well as the the path that leads to this certificate. By giving the flag `-c` it only shows the value of the chain certificate, by giving `-p` it shows the value of the public certificate of the domain and finally by giving `-v` it shows the entire fullchain certificate.

Add certificate changes

Now the command only accept a PEM file as argument instead of a raw text copied in the command line interface.

Verify certificate changes

Apart from changing this functionality to be compatible with multiple skipchains, we did not need to modify or improve this command.

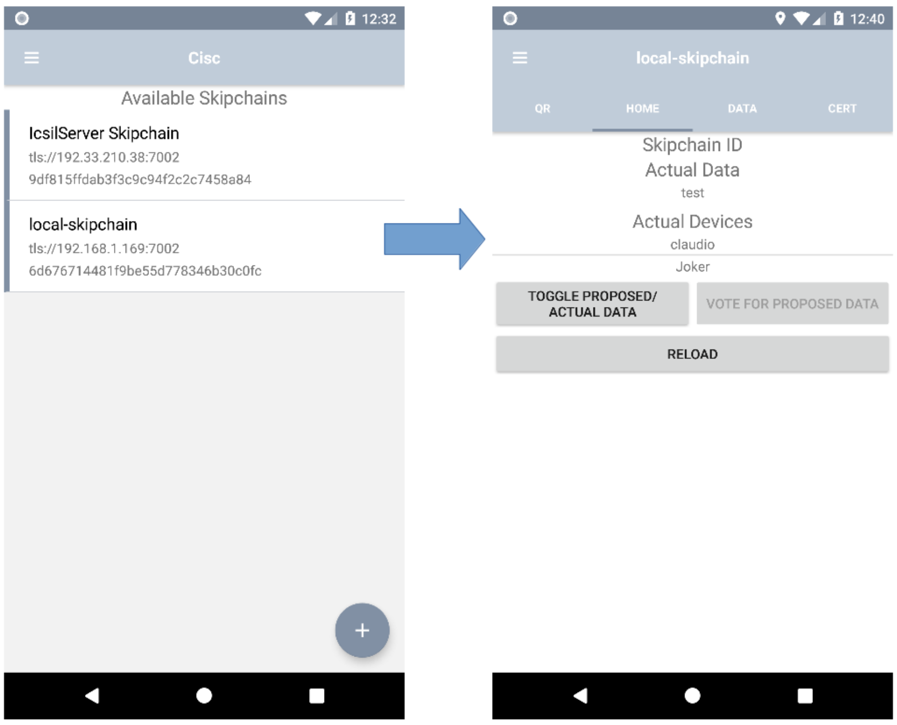
2.2 Cross platform application development

This next section will explain the work realized for the cross platform application. In a first part we will explain the improvements made to the general application mostly related to the Cisc. In a second part, we will explain the work realized for integrating the collective certificate management to the Cisc.

2.2.1 General improvements in Cisc

2.2.1.1 Multiple Skipchains

Beforehand our Cisc drawer was designed with three tabs (Qrcode, home and data) as explained previously. Also this Cisc drawer only allowed connecting to one skipchain. In order to implement multiple skipchains to our app, we had to find a way to store the data corresponding to each of the stored skipchain. Our solution was first to allow instantiating multiple classes (previous implementation only allowed one singleton class) and then save in a folder for each created skipchain the core data of this latter. So now when trying to join a new skipchain it now creates a folder and fill in automatically a file (`identity_link.js`) with its corresponding skipchain data. To ensure that we don't overwrite folders when joining a skipchain, we generate a Universally Unique IDentifier (UUID) as the name of our folder. Furthermore, for testing purposes one can create a skipchain and give as parameter a folder name already containing the necessary information of an existing skipchain. To reflect these changes we need to update our UI. The idea was to create a page for Cisc which would permit the user whether to choose between an existing skipchain(s) available or he could simply just add a new skipchain. To display the existing skipchain in the app we loop through all the saved folder attached to a skipchains and we send this to the UI. Concerning the skipchain join, there is a plus floating action button which allows to scan a skipchain QR code.



IcsilServer Skipchain
tls://192.33.210.38:7002
9df815ffdab3f3c9c94f2c2c7458a84

local-skipchain
tls://192.168.1.169:7002
6d676714481f9be55d778346b30c0fc

test

claudio
Joker

Figure 4: Cisc home page (left) and its corresponding result after pressing on the "local-skipchain" (right)

As depicted in the figure 4, the user can select the desired skipchain and the page of this latter will open. Also a delete feature has been implemented. By swiping the skipchain to the left a delete button appears. This functionality just removes the skipchain from the list but does not make the device leave from the skipchain.

2.2.1.2 Other improvements

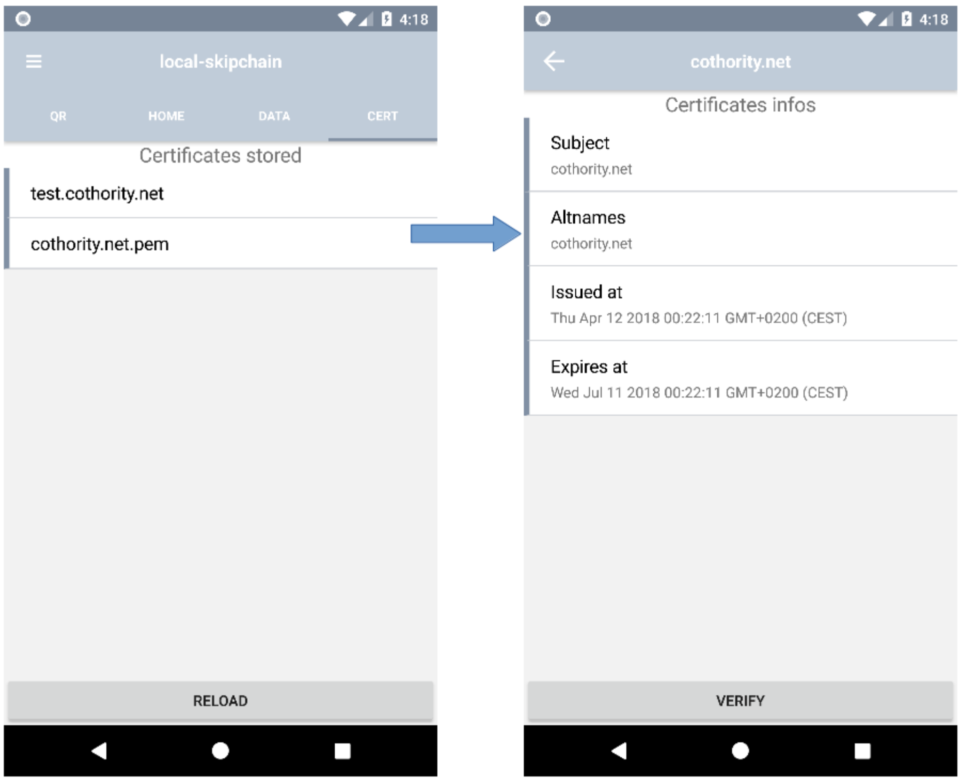
In the settings drawer, to adapt to the multiple-skipchain implementation we updated the user name of device to be global to all skipchains.

Always in the settings drawer, there was an issue with private and public keys, when signing the proposed data in the Cisc. In fact every time a user launch the app for the first time he will always get the same key pairs which caused problems in the data management. Now every time the user launches the application for the first time a new key pair is generated.

2.2.2 Collective certificate management

2.2.2.1 User Interface

In this part we will explain the implementation of our collective certificate management user interface. We decided to add a new tab called "Cert" in the skipchain page. Then pressing on this tab would list the current certificates stored in the skipchain. Then by pressing on a specific certificate, it opens a new page displaying certificate information like issued date, expiry date, altnames... From there the user can also verify if the certificate is valid by pushing the button (see figure 5).



Certificates stored

test.cothority.net

cothority.net.pem

Certificates infos

Subject
cothority.net

Altnames
cothority.net

Issued at
Thu Apr 12 2018 00:22:11 GMT+0200 (CEST)

Expires at
Wed Jul 11 2018 00:22:11 GMT+0200 (CEST)

Figure 5: List of stored certificates (left), details of the certificate "cothority.net.pem" (right)

2.2.2.2 Functionality

In this part we will explain how is implemented the back-end of our certificate management. The main involved files for this implementation are `cert-page.js` and `cert-details.js`. We first filter all the key(s)/value(s) that begin(s) with `---BEGIN CERTIFICATE---` and `---END CERTIFICATE---`. The values containing these keywords will then appear in our Cert tab (see figure 5). To request a certificate we need to pass through a CLI Cisc user on a server, this one requests a certificate and install it on the proposed list and finally let the device(s) approve(s) or disapprove(s) this certificate. **Remark**: with this actual implementation, it supposes that the devices could not request for a certificate. A more complicated process for requesting certificate will be explained in the next sections.

Another realized functionality is the display of certificate information. To execute this feature, we extract the string value of the certificate and convert it to a certificate type which let us easily get the basic information of this latter. Furthermore, in this same page there is a verify button. This functionality permits to verify the following points:

- The certificate validity period includes the current time.

- The certificate was signed by its parent (where the parent is either the next in the chain or from the CA store).
- The certificate issuer name matches the parent's subject name.

For this purpose we used the module `node-forge.js` [7].

2.3 Limitations

2.3.1 User name on the CPMAC

At the moment, the user name can be changed at any time, the problem in our actual implementation is that the same device could join many times the same skipchain with different names. By doing this we lose the principle of decentralization. A malicious user could then create a lot of user names linked to the same device so that he could get a key/value pair easily accepted to be stored in the skipchain.

2.3.2 NativeScript and Node.js

In our cross platform application we are using different npm modules. These ones were meant in the beginning for web programming. However NativeScript allows us to download these modules and since most of these modules were not optimized for NativeScript, they may not work on the application side. For example a module was found to realize the renew operation but was not working as expected.

2.4 Future Work

Request certificate: A more sophisticated system could be implemented for requesting certificates. Suppose we have a device 1, a device 2 and a Cisc user on a server.

1. **Device 1**: Request a certificate, enter a domain name and a server IP
2. **Device 2**: Accept the request from device 1
3. **Cisc user**: If he is responsible for that IP/domain, then request certificate from ACME server proposes new cert:domain_name with value "content of cert"
4. **Device 1**: Sees the request for new key/value pair and accept it
5. **Device 2**: Sees the request for new key/value pair, accept it and we suppose that the threshold was reached thus stores cert:domain_name / "content of cert" on the skipchain.
6. **Cisc user**: Periodically looks at skipchain to see if new keys are present if new certificate is present install the new certificate

Renew certificate: The certificate renewal could be still tried to be implemented. The module used for this purpose was the following "WeDeploy LetsEncrypt" [8].

Unique user: As explained in the limitations, one should not change its device user name. So it needs to be fixed and should be the same every time the app is run. One other solution would be to send a request update when a device changes its name. This part concerning the settings of the application could be improved.

Plugin on browser: In another context but still concerning certificates one could also try to look into adding a plugin to the browser to verify if the certificate is on the skipchain.

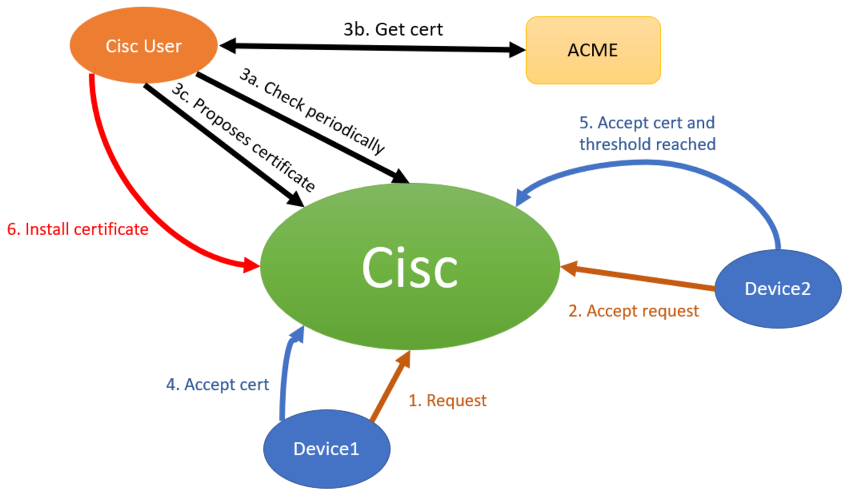


Figure 6: Schema corresponding to the improved request protocol

2.5 Conclusion

The implementation of certificate management to the actual Cothority framework added an additional brick to Cisc application more particularly in the certificate management. Nowadays multiple attacks have occurred against CA's so this decentralized protocols could help solving these issues. Also having a portable device to control or check information about certificates is a plus. It is much more practical to use an application to realize these decentralized protocols than using a command line interface. So was the purpose of the project to give facilities for users owning domains to have their certificates stored in a secure manner. But do not forget that everyone must participate in these protocols/applications.

3 User guide

In this last part we will explain how to run the certificate management on the command line interface and in the cross platform application.

At first, notice that you should better use a Unix environment to support the implementation of this project. You need to install the Golang language that is the used to implement the project. Then, you need to set up the environment variable `GOPATH` using the command `$ export GOPATH=goFolder` so that it points to your workspace directory and add `GOPATH/bin` to PATH using the command `$ export PATH=$PATH:$GOPATH/bin`. Also, you have to install and set up GitHub so that you can download the following package (Using `$ go get -u` command):

3.1 Cisc guide

In this part we will explain in how to work with the Cisc CLI specially for the certificate management.

3.1.1 Initialization

First run the following commands:

```
$ go get -u github.com/dedis/cothority/conode

$ go get -u github.com/dedis/cothority/cisc

$ go get -u github.com/ericchiang/letsencrypt
```

3.1.1.1 Run conodes

Then you need to run the conodes, the easiest way to do that is to run the bash program `run_conode.sh` (normally located in $GOPATH/github.com/dedis/cothority/conode).

```
$ ./run_conode.sh (local|public) (nbr_conode) [dbg_lvl]
```

So you need to choose between local or public depending if you want to run your conodes locally or publicly, then decide about the number of conodes you want to run and eventually a debug level. So you could execute to run three conodes locally.

```
$ ./run_conode.sh local 3 2
```

Then you need to be able to link with the conodes you execute

```
$ cisc link pin localhost:7002
```

A pin will appear when executing this command and you need to rerun the command but this time with the pin code

```
$ cisc link pin localhost:7002 PIN
```

Finally you can create a skipchain with the `public.toml` file (generated in the conode folder) and use the other available cisc commands.

```
$ cisc sc create public.toml
```

For further details about the Cisc commands check the Github repository of the cothority [3].

3.1.2 Running collective certificate management

Now that Cisc is working, let's focus on the Cisc cert commands. First of all for these commands to work, one should be connected to the server hosting a website. Here is a summary of the certificate commands.

- Request - request a certificate and add it to the skipchain. Request takes the domain the `cert` and the `www` path. The certificates and the private key are then saved in the cert folder with the name of the domain.
- List - return a list of all the certificates with their expiry date and the path of the certificate.
- Add - stores a certificate in the skipchain by giving the key and path to the certificate.
- Renew - renew a certificate stored in the skipchain by giving the key of this latter.
- Verify - verify a certificate in the skipchain.
- Retrieve - retrieve the certificate and save it in the specified directory.
- Revoke - revoke and remove the certificate from the skipchain.

As a starting point, if can request a certificate so you should run:

```
$ cisc cert request cothority.net path/to/cert path/to/www
```

If everything went well you should see the certificate listed after calling `cisc cert list`.
Remark: One last word though, make sure you know where all your keys are stored, they will be needed for these commands.

3.2 Cross platform application guide

In this part we will describe how to set-up our NativeScript environment.

3.3 Initialization

First part we need to install NativeScript. For this I will describe their own tutorial but it is important to check their website to always be up-to-date [5].

1. **NodeJS installation**
   This can be done by downloading the installer on their home page. Always install a long term service (LTS) version as it is the supported version for NativeScript.

2. **NativeScript CLI installation**
   This can be done by running the following command:

   ```
   $ npm install -g nativescript
   ```

   If an EACCES error is returned at some point of the installation, re-run the last command with administrator rights. If EACCES errors are still returned, run the command again with administrator rights and the unsafe permissions parameter of NPM:

   ```
   $ sudo npm install -g --unsafe-perm nativescript
   ```

3. **Android and iOS requirements**
   Since it depends on your operating system (OS), follow the official tutorial [1]. NativeScript provides scripts for Windows and macOS that will automatically setup most dependencies. It is still recommended to look at the advanced setups they provide to ensure that everything is correctly installed.

4. **TNS doctor**
   The last step is to check if all requirements are met, this can be done by running: `$ tns doctor`. If errors are returned, fix them before continuing.

Finally one last word about the editor for your code. In essence, any editor could be used. We recommend using Visual Studio Code since it is the officially supported editor and provides an official plugin to integrate with NativeScript.

3.4 Accessing the certificate module

Once everything is set-up we can launch the app and access our certificate module. First press the drawer icon on the top left then click on Cisc. A page will open with all the skipchains listed. If it's your first time running the app, you might expect finding this list empty. To join a skipchain with your device you need a QR code of a skipchain. To obtain this code you need to run a skipchain on your CLI for example following part 3.1.1 of the Cisc guide. Once the skipchain is created you can run:

```
$ cisc sc qr -e
```

If everything went well you should have your QR code displayed. Now your QR code has to be scanned by the app (you can do this by taking a picture of it for example). If the scan recognized your skipchain your device has successfully joined the skipchain. However the other users connected to the skipchain need to approve this device by running `$ cisc data vote`. Finally you can access the Cert tab and see the stored certificates if there is any.

[1]`https://docs.nativescript.org/start/quick-setup#`

References

[1] Bryan Ford, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, Ewa Syta, Iulia Tamas, Dylan Visher, and David Isaac Wolinsky. Keeping authorities “honest or bust” with decentralized witness cosigning. https://arxiv.org/abs/1503.08768.

[2] Robin Berguerand. Collective certificate management. https://github.com/dedis/student_17_certificate_skipchain/tree/master/cisc.

[3] DEDIS Lab EPFL. Cothority. https://github.com/dedis/cothority.

[4] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, Justin Cappos, and Bryan Ford. Chainiac: Proactive software-update transparency via collectively signed skipchains and verified builds. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/nikitin.

[5] Nativescript. https://www.nativescript.org/.

[6] Cédric Maire & Vincent Petri. Cross-platform mobile application for cothority. https://github.com/dedis/student_17_mobile/blob/master/report/report.pdf.

[7] Different collaborators. A native implementation of tls (and various other cryptographic tools) in javascript. https://www.npmjs.com/package/node-forge.

[8] Different collaborators. Wedeploy letsencrypt. https://www.npmjs.com/package/wedeploy-letsencrypt.