trust yourself.

# Distributed Root Certificate Store

Concept Paper

+++ DRAFT +++

*Version 0.1*
*2020-08-25*
*Gregor Jehle*

# Contents




*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*

# Document History

| | | |
|---|---|---|
| 2020-08-25 | 0.1 | Initial draft |

| | |
|---|---|
| GJ | Gregor Jehle, P3KI GmbH |


*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*

# 1 Introduction

## 1.1 Executive Summary

This document discussed the implementation of a distributed X.509 root certificate store to solve the specific requirements of the Maritime Connectivity Platform (MCP).

The goal of the MCP is to provide a unified but distributed – possibly decentralized – communication platform for international, maritime shipping operations. The MCP is designed and developed by members of the Maritime Connectivity Platform Consortium (MCC), an international expert group.

A key challenge at the time of writing of this document is the handling of trust anchors (or root certificates) used for the various MCP instances. The express goal of the MCC is to not hold any secret key materials or responsibility within the chain of operation of individual instances. A possible, quick solution to this challenge is presented in this document.

## 1.2 Focus & Scope

- To be used with existing X.509-based solutions
- Quick to implement
- Distributed or decentralized responsibility over a given entity's effective root certificate store configuration

## 1.3 Methodology

- OpenSSL-based Proof-of-Concept implemented as shell-script within a Docker container to showcase base functionality (NOT PRODUCTION READY)
- Documentation (this document) to provide additional context and insight regarding the applicability to production implementations

## 1.4 About P3KI

P3KI GmbH ("P3KI") specializes in the design and implementation of highly distributed and decentralized networked systems in the context of permission delegation and public-key infrastructure for IoT and IIoT. With a unique focus on verifiable and provable implementations of competently designed systems, P3KI's holistic solutions target long running and low level applications.

P3KI is a subsidiary of Recurity Labs GmbH, a renowned cyber security consultancy with strong experiences in the industrial and communication infrastructure markets. This experience is also the building block and driving force behind P3KI's design principles and work ethics.

With a vast network of excellent partners from industry, private businesses, as well as state and academic institutions, we're able to meet even the toughest challenges.


*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*

# 2 The Proof-of-Concept

## 2.1 Idea

## 2.2 Walkthrough

### 2.2.1 Prerequisites

- Docker
- git

### 2.2.2 Checkout

```
$ git clone https://github.com/hdznrrd/mccroot.git
$ cd mccroot
```

### 2.2.3 Building the Container

Building the containers

```
$ ./build.sh
```

```
Sending build context to Docker daemon  5.632kB
Step 1/7 : FROM ubuntu:latest
 ---> 4e2eef94cd6b
Step 2/7 : MAINTAINER Gregor Jehle <gregor@p3ki.com>
 ---> Using cache
 ---> c5aa578a7771
Step 3/7 : RUN apt-get update && apt-get upgrade -y
 ---> Using cache
 ---> 14eb6913117b
Step 4/7 : RUN apt-get install -y rng-tools openssl
 ---> Using cache
 ---> e56308967d13
Step 5/7 : ADD run.sh /mmc/
 ---> Using cache
 ---> 37f8a41a05ca
Step 6/7 : WORKDIR /mmc
 ---> Using cache
 ---> 750324346eb7
Step 7/7 : ENTRYPOINT rngd -r /dev/urandom
 ---> Running in bb01dba4bbf2
Removing intermediate container bb01dba4bbf2
 ---> bb3a910ce1e7
Successfully built bb3a910ce1e7
Successfully tagged mccroot:latest
```


Version 0.1, 2020-08-25, Gregor Jehle
P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com

### 2.2.4 Running the Proof-of-Concept

To start the container and get a shell in it simply use the outside runscript:

```
$ ./run.sh
```

You'll be greeted with a new shell prompt, this time from within the container:

```
root@416aca3a2df6:/mmc#
```

From inside the container, use the provided inside runscript:

```
root@416aca3a2df6:/mmc# ./run.sh
```

Let's walk through the output of that script together.

We'll start off with `rngd` complaining. Since we're only using `rngd` in this demo to ensure we have plenty – albeit bad – entropy, this is is of no concern. DO NOT USE `rngd` IN PRODUCTION. Be a good sport and create proper certificates!

```
unable to adjust write_wakeup_threshold: Read-only file system
```

Next up, the script generates a bunch of self-signed certificates. Please note that these are 1024 bit RSA keys for sake of not wasting entropy and the demo locking up on us. NEVER USE CERTIFICATES CONFIGURED LIKE THIS IN PRODUCTION!

```
CREATING CERTIFICATE FOR attester1
Generating a RSA private key
.....................+++++
...................+++++
writing new private key to 'attester1/attester1.key.pem'
-----
writing RSA key
CREATING CERTIFICATE FOR attester2
Generating a RSA private key
......+++++
.............................................+++++
writing new private key to 'attester2/attester2.key.pem'
-----
writing RSA key
CREATING CERTIFICATE FOR root1
Generating a RSA private key
........+++++
............+++++
writing new private key to 'root1/root1.key.pem'
-----
writing RSA key
CREATING CERTIFICATE FOR root2
Generating a RSA private key
..............+++++
.............................+++++
writing new private key to 'root2/root2.key.pem'
-----
writing RSA key
CREATING CERTIFICATE FOR root3
Generating a RSA private key
```


*Version 0.1, 2020-08-25, Gregor Jehle*
*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*

```
......+++++
.............................+++++
writing new private key to 'root3/root3.key.pem'
-----
writing RSA key
CREATING CERTIFICATE FOR root4
Generating a RSA private key
.....................................................+++++
.......................+++++
writing new private key to 'root4/root4.key.pem'
-----
writing RSA key
```

We now have self-signed certificates for a bunch of roles:

- **root1** through **root4** represent root certificates for four different MCP instances.
- **attester1** and **attester2** are instances an individual operator may accept as their trust anchor. This could be a government agency or possibly an insurance company.

Next up, we'll publish our root certificates. In practice this could be done by each MCP instance operator individually by uploading their respective certificates to a common webservice, possibly run by the MCC. As part of the proof-of-concept, this is simply simulated by copying the respective certificates into the `published/` folder.

```
PUBLISHING CERTIFICATE OF root1
'root1/root1.cert.pem' -> 'published/root1.cert.pem'
PUBLISHING CERTIFICATE OF root2
'root2/root2.cert.pem' -> 'published/root2.cert.pem'
PUBLISHING CERTIFICATE OF root3
'root3/root3.cert.pem' -> 'published/root3.cert.pem'
PUBLISHING CERTIFICATE OF root4
'root4/root4.cert.pem' -> 'published/root4.cert.pem'
```

Now the real work begins.

- *attester1* is going to attest both *root1* and *root2* to be from legitimate MCP instances
- *attester2* does the same for *root2* and *root3*
- Note that *root4* is not legitimized by anyone, this could be because they have not started the process of getting accredited yet (or haven't yet passed) or possibly because it's a rogue certificate by a malicious actor

For the demo this is done by creating a signature over the respective root certificates using the key of the respective attester and then publishing said signature to the same place the root certificates are published to.

The naming convention is as follows:

- Input: Certificate of *root1* named `published/root1.cert.pem`
- Input: Key from *attester1* (kept secret/unpublished)
- Output: Signature file named `published/root1.cert.pem.attester1.sha256.sign`

```
ATTESTING root1 BY attester1
ATTESTING root2 BY attester1
ATTESTING root2 BY attester2
ATTESTING root3 BY attester2
```


*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*

For the next step let's assume you believe in ***attester1***'s testimony on which root certificate is actually belonging to a vetted and trusted MCP instance. You can now enumerate all available certificate published on the webservice (here: `published/` folder) and see if there is a suitable signature available that was created by ***attester1***. The proof-of-concept implements this very naively by simply testing every available signature against all available certificates. This is an

$$O(n^2)$$

operation, so probably not the option one should pick for production.

The result we expect is that `Verified OK` shows up exactly twice, once for ***root1*** and again for ***root2***:

```
LISTING CERTIFICATES ATTESTED BY attester1
Testing signature ./published/root2.cert.pem.attester1.sha256.sign for all certs...
VERIFING SIGNATURE ./published/root2.cert.pem.attester1.sha256.sign by attester1 FOR
    root1...Verification Failure
VERIFING SIGNATURE ./published/root2.cert.pem.attester1.sha256.sign by attester1 FOR
    root2...Verified OK
VERIFING SIGNATURE ./published/root2.cert.pem.attester1.sha256.sign by attester1 FOR
    root3...Verification Failure
VERIFING SIGNATURE ./published/root2.cert.pem.attester1.sha256.sign by attester1 FOR
    root4...Verification Failure
Testing signature ./published/root1.cert.pem.attester1.sha256.sign for all certs...
VERIFING SIGNATURE ./published/root1.cert.pem.attester1.sha256.sign by attester1 FOR
    root1...Verified OK
VERIFING SIGNATURE ./published/root1.cert.pem.attester1.sha256.sign by attester1 FOR
    root2...Verification Failure
VERIFING SIGNATURE ./published/root1.cert.pem.attester1.sha256.sign by attester1 FOR
    root3...Verification Failure
VERIFING SIGNATURE ./published/root1.cert.pem.attester1.sha256.sign by attester1 FOR
    root4...Verification Failure
```

Next up we do the same, but using signatures created by ***attester2***.

Here the expected result is again two instances of `Verified OK` but this time around for ***root2*** and ***root3***:

```
LISTING CERTIFICATES ATTESTED BY attester2
Testing signature ./published/root3.cert.pem.attester2.sha256.sign for all certs...
VERIFING SIGNATURE ./published/root3.cert.pem.attester2.sha256.sign by attester2 FOR
    root1...Verification Failure
VERIFING SIGNATURE ./published/root3.cert.pem.attester2.sha256.sign by attester2 FOR
    root2...Verification Failure
VERIFING SIGNATURE ./published/root3.cert.pem.attester2.sha256.sign by attester2 FOR
    root3...Verified OK
VERIFING SIGNATURE ./published/root3.cert.pem.attester2.sha256.sign by attester2 FOR
    root4...Verification Failure
Testing signature ./published/root2.cert.pem.attester2.sha256.sign for all certs...
VERIFING SIGNATURE ./published/root2.cert.pem.attester2.sha256.sign by attester2 FOR
    root1...Verification Failure
VERIFING SIGNATURE ./published/root2.cert.pem.attester2.sha256.sign by attester2 FOR
    root2...Verified OK
VERIFING SIGNATURE ./published/root2.cert.pem.attester2.sha256.sign by attester2 FOR
    root3...Verification Failure
VERIFING SIGNATURE ./published/root2.cert.pem.attester2.sha256.sign by attester2 FOR
    root4...Verification Failure
root@416aca3a2df6:/mmc#
```

You can now exit the shell-inside-the-container using `exit`.


*Version 0.1, 2020-08-25, Gregor Jehle*
*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*

Congratulations, you've made it through the proof-of-concept. If you were to collect the root certificates for which `Verified OK` was the result and import them into your system's root certificate store, you can now talk to the respective vetted and attested MCP instance and verify MRN assignments issued by them as well.

To sneak a peek at how the thing works under the hood, check out `docker/run.sh`.

## 2.3 Shortcomings

The shortcomings of this proof-of-concept are plentyful, let's enumerate the most serious offenders first

- Do NOT fudge entropy using `rngd` as the demo does, this is purely so the demo does not block if your system were to run out of entropy.
- Do NOT using 1024 bit RSA keys, this is – again – purely to preserve entropy in the context of the demo.
- DO require all certificates involved to actually define a CRL or OCSP server so revocations can be done if needed.
- You might want to consider also using a different hashing scheme than SHA256.

Now that cryptographic fudging of the proof-of-concept is out of the way, some more items for your consideration:

- The roots used here should probably be intermediaries instead, technically this does not make a difference to the operation of the distributed root store and attestation mechanics, but if you need to perform a revocation and recovery, doing so with an intermedate CA instead of a root CA is still painful, but at least less so
- The same goes for attester certificates.
- When updating your list of roots, consider throwing out old ones from your local root store when importing the fresh set.
- Make sure you have a sound concept allowing revocation and revocation checks for both attesters, roots, and possibly even the attestations (the signatures) themselves depending on your threat model requirements. That last part will need some extra work that's left as an exercise for the implementer.
- There need to be suitable processes that cover end-to-end
  - upload of root certificate from MCP instances by their operators to the webservice
  - formal notification of attester(s) about which certificate was uploaded and should be attested
  - a formal process of fetting the MCP instances and operators
  - issuing an attestation signature and publishing it to the webservice
  - building lists of attested/vetted root certificates for consumers (MCP users)
- A proper solution that limits trust on a more fine-granular level than "has a valid certificate" would be appropriate. Something like P3KI's DPKI solution. However, one of the goals of this proof-of-concept is to arrive at a solution that works with many moving and already existing parts and quickly so. Therefore, X.509 is not what we deserve, but what we get.

## 2.4 Properties

After listing everything you should not do or do better than the proof-of-concept does, here's a list of the nice properties:

- The operator of the webservice everyone publishes to does not need to hold any secret material involved with the chain of trust between MCP instances providers and MCP users


*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*

- Attesters could be anyone from government agencies ("The Ministries of Fisheries says you are a vetted MCP instances, I believe them!") to insurance companies ("If you want to insure your shipping operation with us, we expect you to only talk to instances we or one of our partners vetted!")
- There does not need to be a limitation as to who can publish root certificates to the webserver (at least for cryptographic and security reasons), the security properties stem from attesters picking out the right certificates and signing them publicly
- The exact process and requirements for the attestation (as mentioned in the shortcomings paragraph above) can be offloaded to a large degree to attesters themselves

%%%


*Version 0.1, 2020-08-25, Gregor Jehle*
*P3KI GmbH c/o Recurity Labs GmbH, Wrangelstr. 4, 10997 Berlin, Germany, +49 (0)711 22051252 (Stuttgart office), +49 (0)30 695399933 (Berlin office), contact@p3ki.com*