AN12120

A71CH for electronic anticounterfeit protection

Rev. 1.1 — 7 March 2018
458311

Application note
COMPANY PUBLIC

Document information

Info	Content
Keywords	Security IC, IoT, Product support package, Secure cloud connection, Anti-counterfeit, Cryptographic authentication.
Abstract	This document describes how the A71CH security IC can be used to implement a strong authentication mechanism based on ECC cryptography to prevent electronic counterfeit and verify proof-of-origin.

Revision history

Rev	Date	Description
1.0	20180219	First release
1.1	20180302	Updated section 3.2

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

This document provides an overview of the main security concepts and asymmetric cryptography fundamentals. It also describes how A71CH can be used for proof-of-origin verification or anticounterfeit protection. The document introduces ECC cryptography fundamentals, it describes the mechanisms and credentials involved to perform a cryptographic mutual authentication between a server and an IoT device. And finally, for A71CH evaluation and demonstration purposes, it details how a mutual authentication can be performed using the A71CH Configure tool, OpenSSL and A71CH OpenSSL Engine libraries.

The document is intended for IoT designers, who aim to implement a strong authentication mechanism based on ECC cryptography for preventing electronic counterfeiting between a server and IoT devices.

2. A71CH overview

The A71CH is a ready-to-use solution enabling ease-of-use security for IoT device makers. It is a secure element capable of securely storing and provisioning credentials, securely connecting IoT devices to public or private clouds and performing cryptographic device authentication.

The A71CH solution provides basic security measures protecting the IC against many physical and logical attacks. It can be integrated in various host platforms and operating systems to secure a broad range of applications. In addition, it is complemented by a comprehensive product support package, offering easy design-in with plug & play host application code, easy-to-use development kits, documentation and IC samples for product evaluation.

3. Public key infrastructure and ECC fundamentals

Security is an essential requirement for any IoT design. Thus, security should not be considered as differentiator option but rather a standard feature for the IoT designers. IoT devices must follow a secure-by-design approach, ensuring protected storage of credentials, device authentication, secure code execution and safe connections to remote servers among others. In this security context, the A71CH security IC is designed specifically to offer protected access to credentials, secure connection to private or public clouds and cryptographic device proof-of-origin verification.

Asymmetric cryptography, also known as public key cryptography, is any cryptographic algorithms based on a pair of keys: a public key and a private key. The private key must be kept secret, while the public key can be shared.

RSA (River, Shamir and Adleman) and Elliptical-Curve Cryptography (ECC) are two of the most widely used asymmetric cryptography algorithms. In the case of ECC cryptography, it is based on the algebraic structure of elliptic curves over finite fields. Therefore, each key pair (public and private key) is generated from a certain elliptical curve.

The digital signature, digital certificates, Elliptic Curve Digital Signature Algorithm (ECDSA) and Elliptic Curve Diffie-Hellman (ECDH) key agreement algorithm are briefly explained in the next sections.

3.1 Digital signature

A digital signature is used to guarantee the authenticity, the integrity and non-repudiation of a message. A signing algorithm generates a signature given a message and a private key. A signature verifying algorithm accepts or rejects a message given the public key and the signature.

Fig 1 illustrates an example of digital signature. In this case, the message is signed with the sender private key. The receiver will validate the signature using both the message and the sender public.

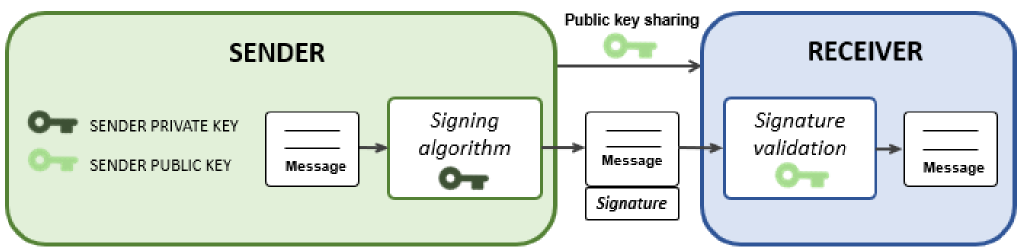


Fig 1. Digital signature diagram

3.2 Digital certificate, Certification Authority (CA) and Certificate Signing Request (CSR)

Digital certificates are used to prove the authenticity of shared public keys. Digital certificates are electronic documents that include information about the sender public key, identity of its owner and the signature of a trusted entity that has verified the contents of the certificate, normally called Certificate Authority (CA).

A Certificate Authority (CA) is an entity that issues digital certificates. The CA is trusted by both the certificate sender and the certificate receiver, and it is typically in charge of receiving a Certificate Signing Request (CSR) and generating a new certificate based upon information contained in the CSR and signed with the CA private key.

Therefore, a CSR is a request that contains all the necessary information, e.g., sender public key and relevant information to generate a new digital certificate.

Fig 2 shows digital certificates generation steps. First, the interested device (sender) creates a Certificate Signing Request. The CSR is then sent to the CA and a new digital certificate is created and signed with the CA private key. Also, the basic contents of this new digital certificate are illustrated in the figure.

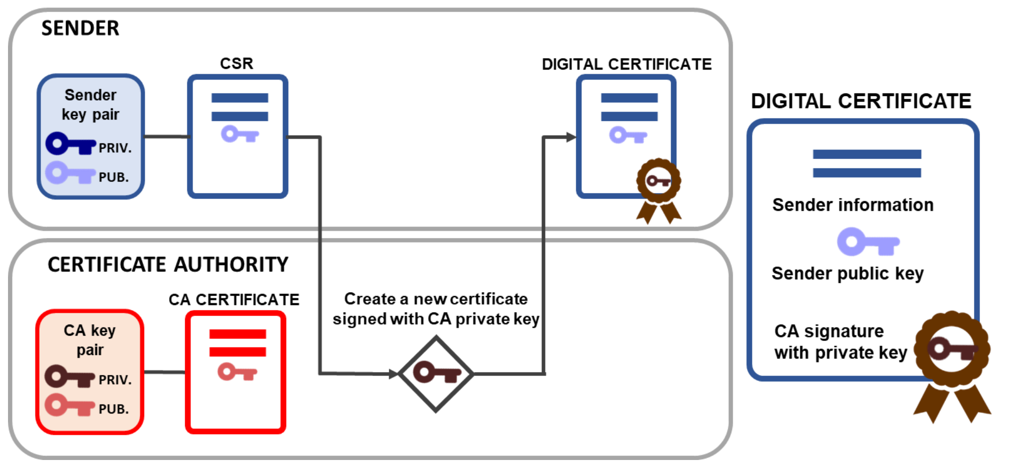


Fig 2. Digital certificate generation steps and contents

3.3 Elliptic Curve Digital Signature Algorithm (ECDSA)

The Elliptic Curve Digital Signature Algorithm (ECDSA) algorithm uses ECC to provide a variant of the Digital Signature Algorithm (DSA). A pair of keys (public and private) are generated from an elliptic curve, and these can be used both for signing or verifying a message's signature. Fig 3 illustrates an example of ECDSA application. In this example, the sender device generates a signature with its private key. The signed message is sent together with the sender digital certificate to the receiver. Finally, the receiver retrieves the sender public key from the digital certificate and uses it to validate the signature of the received message.

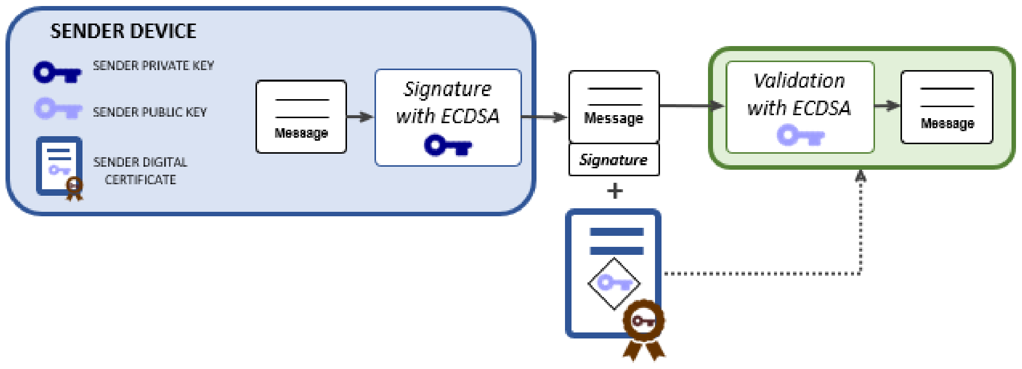


Fig 3. Elliptic Curve Digital Signature Algorithm (ECDSA) example

3.4 Elliptic Curve Diffie-Hellman (ECDH)

Elliptic Curve Diffie-Hellman algorithm (ECDH) is a key-agreement protocol. The goal of ECDH is to reach a key agreement between two parties, each having an elliptic-curve key pair generated from the same domain parameters. When the agreement has been reached, a shared secret key, usually referred to as the 'master key', is derived to obtain session keys. These session keys will be employed to establish a communication using symmetric-key encryption algorithms.

The sender and the receiver have its own elliptical key pair. Both the sender and receiver public keys are shared with each other. In this case, the exchange has been represented with digital certificates. Each party can compute the secret key using their own private key and the public key obtained from the received certificate. Due to the elliptical curve properties and the fact that both key pairs have been generated from the same domain parameters, the computed secret key is the same for both parties. This common secret key will be further used for establishing a communication and encrypt messages based on symmetrical cryptography. Fig 4 illustrates the usage of ECDH for a shared secret key agreement.

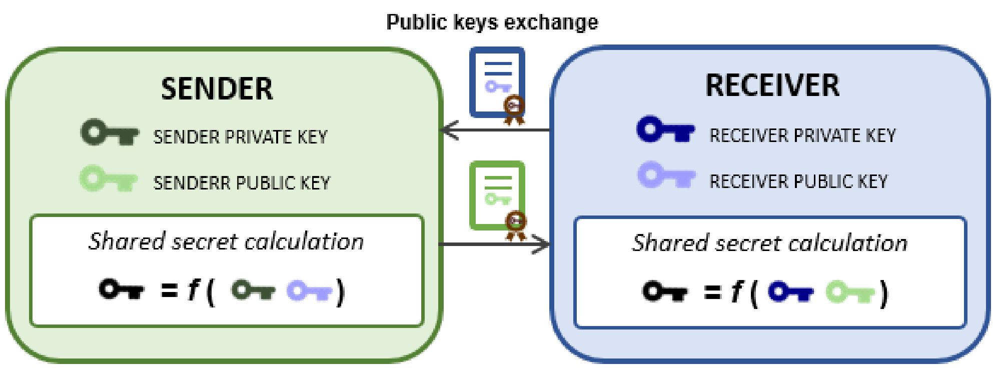


Fig 4. Elliptic Curve Diffie-Hellman Key Exchange (ECDH) example

In the Elliptic-curve Diffie-Hellman Ephemeral (ECDHE) algorithm case, a new elliptical key pair is generated for each key agreement instead of sharing the already existing public keys.

3.5 A71CH ECC supported functionality

The A71CH security IC supports the following ECC functionality:

- Signature generation and verification (ECDSA).
- Shared secret calculation using Key Agreement (ECDH or ECDH-E).
- Protected storage, generation, insertion or deletion of key pairs (NIST-P256 elliptical curve).

4. A71CH for electronic anticounterfeit protection

This section details how A71CH can be used for device proof-of-origin and anti-counterfeit protection. It describes the steps and credentials involved so that a server can verify the authenticity of an IoT device as well as the procedure so that the IoT device can also verify server's authenticity.

First, the IoT device verifies the server authenticity to avoid an untrusted server from trying to attack the system. Therefore, the complete IoT device authentication procedure is divided into the following steps:

1. Server authentication process
    a. Server certificate verification
    b. Server authenticity verification
2. IoT device authentication process
    a. IoT device certificate verification
    b. IoT device authenticity verification

4.1 Server credentials

The server stores a unique key pair (Server key pair) and a digital certificate (Server certificate) signed by a trusted CA. The server can either behave as the CA (thus store the self-signed root CA certificate and root CA key pair) or trust in a third-party CA.

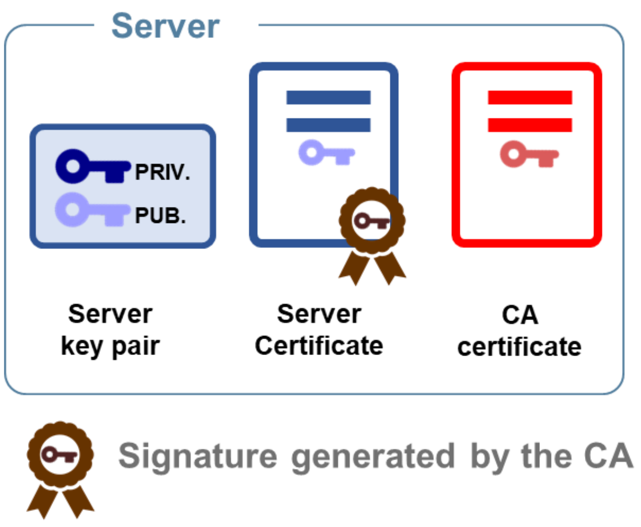
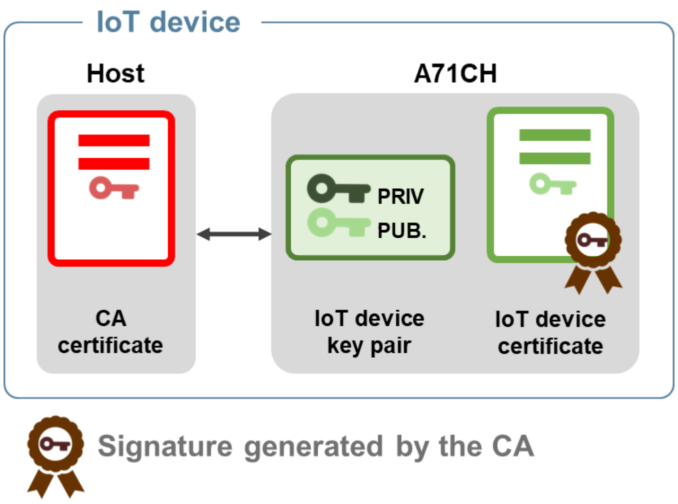


Fig 5. Server certificates

4.2 IoT device credentials

The IoT device stores a unique key pair (IoT node key pair) and its digital certificate (IoT certificate) signed by a trusted CA. It should also contain the CA certificate or CA public key for a Server authentication. The IoT device key pair and digital certificate will be stored in the A71CH.



(1) CA certificate could optionally be stored inside A71CH

Fig 6. IoT device certificates

4.3 Server authentication process

The server authentication consists of the following steps:

- **Step 1:** Server certificate verification. Validate the Server digital certificate.
- **Step 2:** Server authenticity verification. Validate the Server public key.

4.3.1 Step 1: Server certificate verification

In order to verify server’s authenticity, the IoT device validates the server digital certificate, e.g. the IoT device verifies that the server certificate is signed with the same CA and can therefore be trusted. Fig 7 shows the flow diagram corresponding to this step. The server certificate verification will only be successful if the certificate signature can be validated with the CA public key. Otherwise the process will be stopped. After the certificate signature validation, the IoT device is now able to trust the server public key.

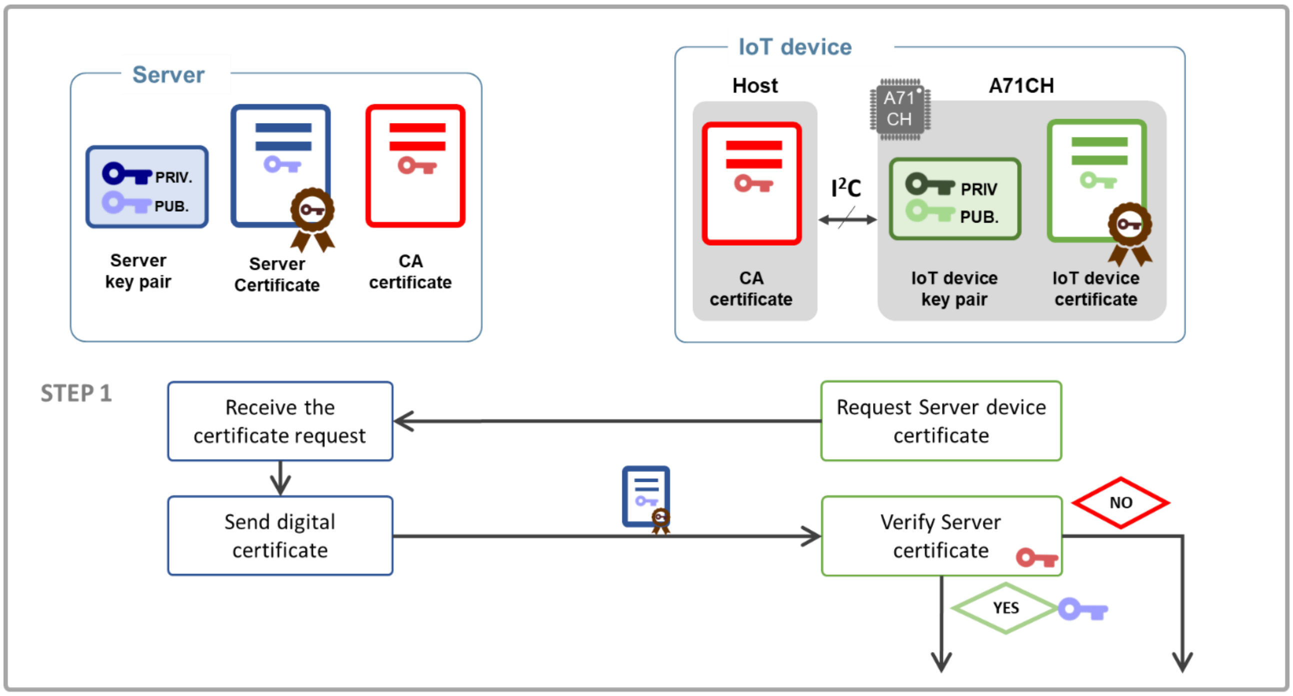


Fig 7. Step 1: Server certificate verification

4.3.2 Step 2: Server authenticity verification

In this step, the IoT device sends a random challenge to the server. This random challenge is signed by the server private key and returned to the IoT device. Finally, the IoT device verifies the authenticity of the server by validating the random message signature with the Server public key. The A71CH functionality includes generation of random messages.

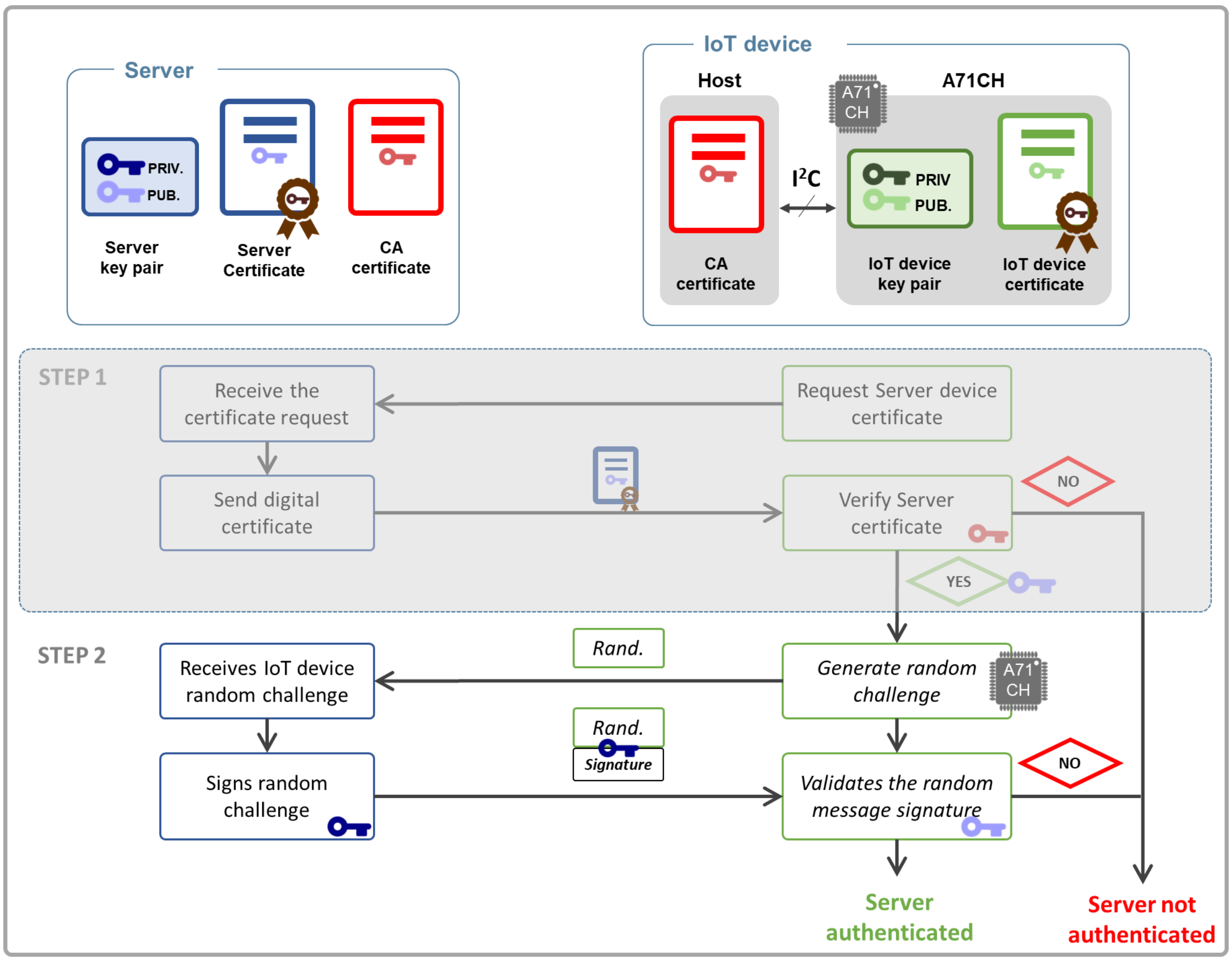


Fig 8. Step 2: Server authenticity verification

4.4 IoT device authentication process

The IoT device authentication consist of the following steps:

- **Step 3:** IoT device certificate verification**.** Validate the IoT device certificate with the CA public key.
- **Step 4:** IoT device authenticity verification. Validate the IoT device public key.

4.4.1 Step 3: IoT device certificate verification

The first step of the IoT device authentication procedure is the IoT device digital certificate verification. The server platform verifies that the IoT device certificate is signed with the same CA and can therefore be trusted.

First, the server requests the digital certificate of the IoT device. The IoT device then retrieves the certificate stored in the A71CH security IC and sends it to the server.

Finally, the IoT device certificate signature is validated with the CA public key (obtained from the CA certificate).

The IoT device certificate will be verified only if its certificate signature is validated, meaning that both the server and the IoT device are trusting in the same CA. Furthermore, the server will be able to trust the IoT device public key (pale green). Fig 9 shows the flow diagram of this first step.

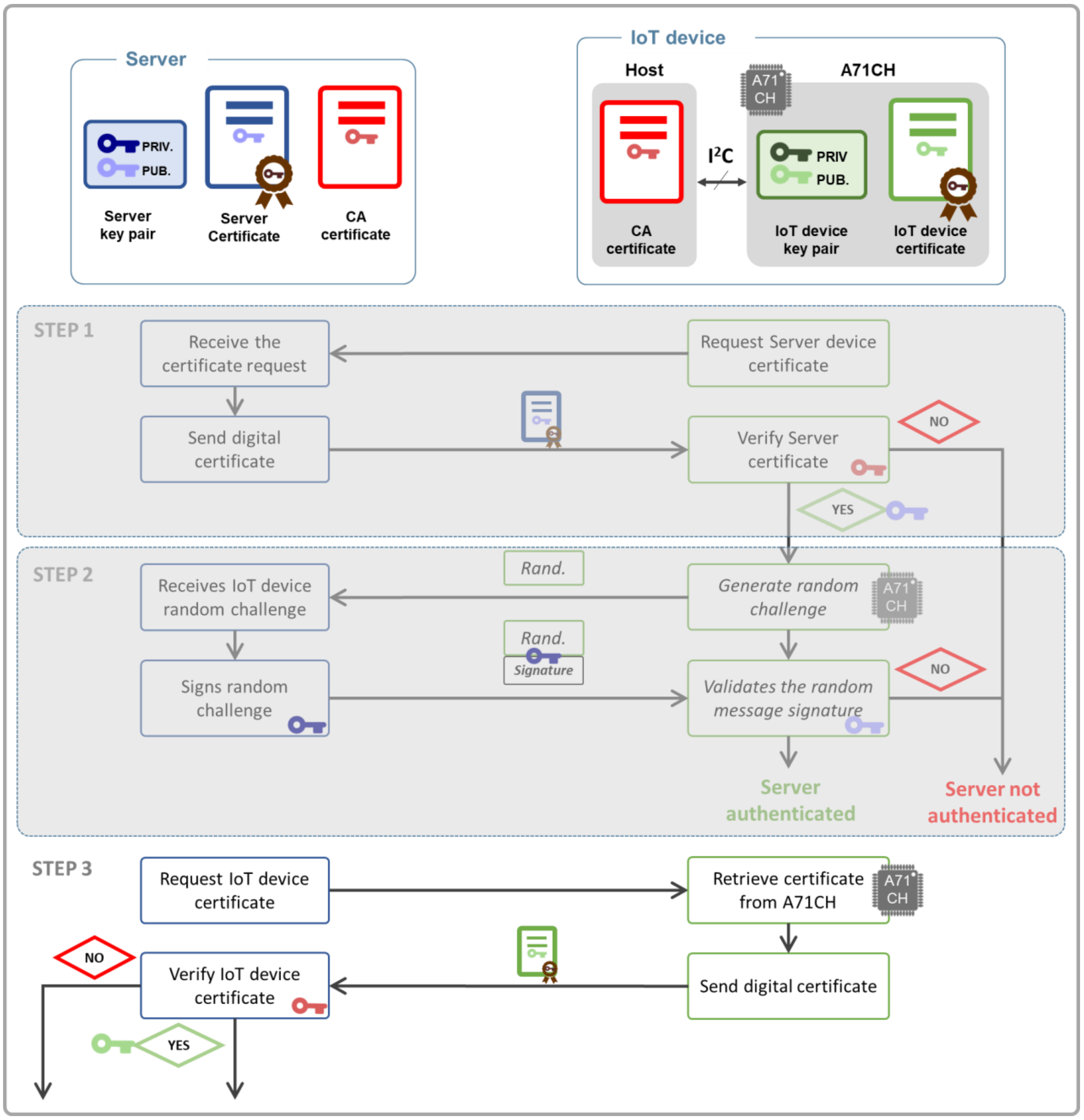


Fig 9. Step 3: IoT device certificate verification

4.4.2 Step 4: IoT device authenticity verification

Once the IoT device certificate has been verified, the server sends a random challenge to the IoT device. The IoT device signs the random challenge using its private key and returns it to the server. Finally, the server verifies the authenticity of the IoT device by validating the signature with the public key obtained from its certificate. Fig 11 shows the flow diagram of this step.

4.5 Host interface authentication (SCP03 secure channel)

Optionally, the communication channel between the IoT device and the A71CH can be secured by establishing an active SCP03 channel. The A71CH permits to establish an SCP03 secure channel between the IC and a given host [SCP03].

By doing this, the I^2C link is secured, avoiding eavesdropping on the I^2C channel and ensuring that only an authenticated IoT device MCU can exchange APDU commands with the A71CH, given that both elements must know the SCP03 session keys. As a consequence, replay attacks are not possible. Fig 10 illustrates the SCP03 channel established between the IoT device Host and the A71CH security IC. Both the Host and the A71CH have the SCP03 keys (ENC, MAC and DEK) so that the session keys can be generated according to [SCP03].

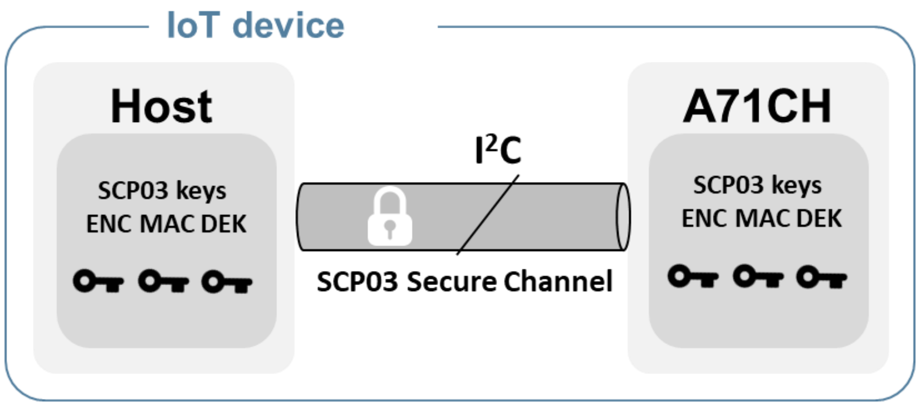


Fig 10. SCP03 secure channel establishment between the Host and the A71CH

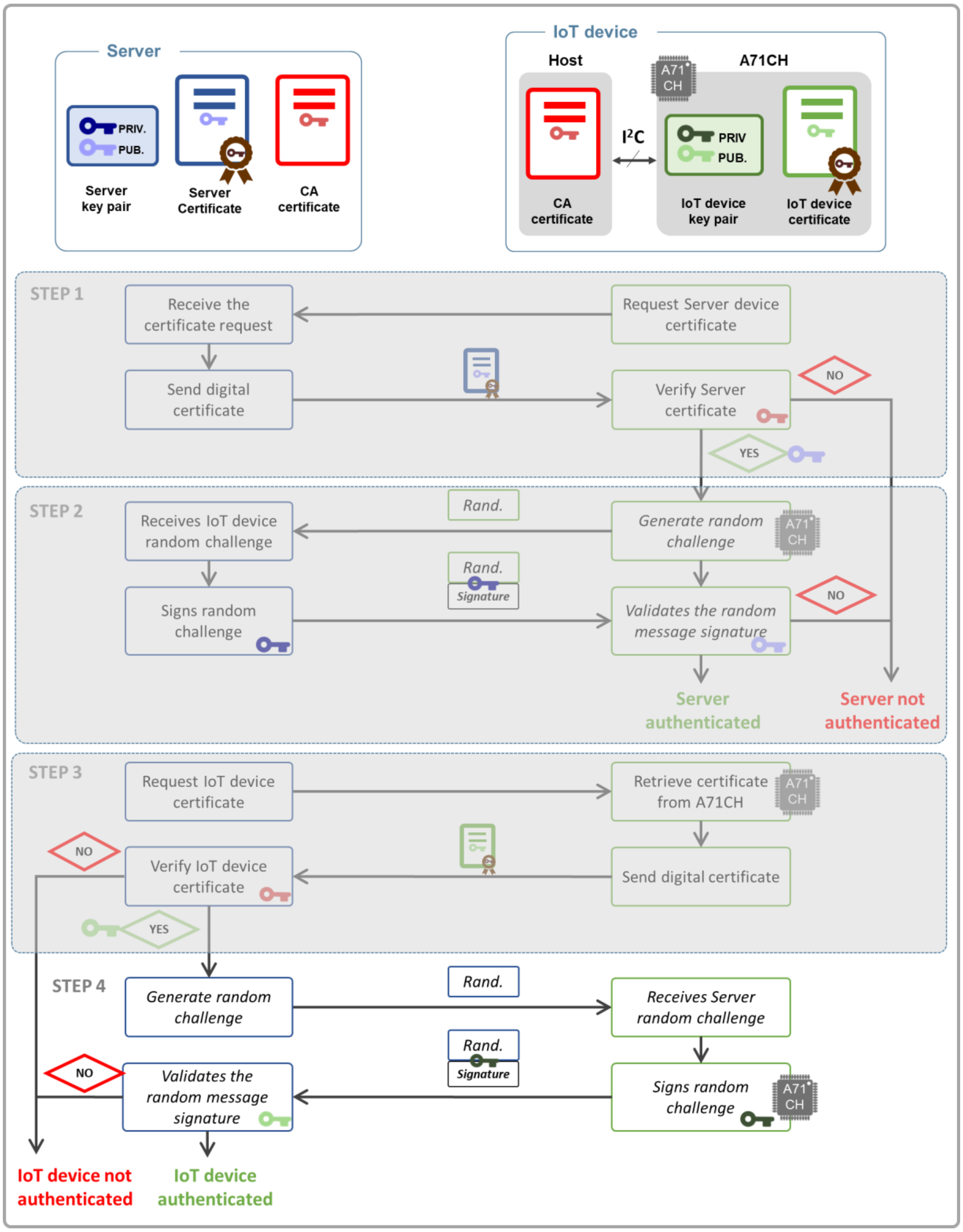

Server
PRIV.
PUB.
Server key pair
Server Certificate
CA certificate
IoT device
Host
A71 CH
A71CH
I²C
CA certificate
PRIV
PUB.
IoT device key pair
IoT device certificate
STEP 1
Receive the certificate request
Request Server device certificate
Send digital certificate
Verify Server certificate
NO
YES
STEP 2
Rand.
Receives IoT device random challenge
Generate random challenge
Rand.
Signature
Signs random challenge
Validates the random message signature
NO
Server authenticated
Server not authenticated
STEP 3
Request IoT device certificate
Retrieve certificate from A71CH
NO
Verify IoT device certificate
Send digital certificate
YES
STEP 4
Generate random challenge
Rand.
Receives Server random challenge
NO
Rand.
Signature
Validates the random message signature
Signs random challenge
IoT device not authenticated
IoT device authenticated

Fig 11. Step 4: IoT device authenticity verification

5. Evaluating A71CH security IC for anticounterfeit protection

This section presents how to use the tools available in the A71CH Host software package for electronic anticounterfeit protection. The section has been divided in four steps:

- IoT device A71CH key provisioning
- IoT device A71CH and Server certificate provisioning
- Server authentication process
- IoT device authentication process

Note: This section describes how to evaluate A71CH security IC for proof of origin validation or anticounterfeit protection based on A71CH product support package. This section illustrates how to perform mutual authentication between a development PC acting as a server and the i.MX6UltraLite embedded platform acting as an IoT device client using the A71CH Configure tool, OpenSSL and A71CH OpenSSL Engine libraries. The following description is provided only for demonstration. Therefore, the subsequent procedure must be adapted and adjusted accordingly for commercial deployment.

5.1 A71CH Security IC key provisioning

The A71CH security IC offers two options regarding key storage and generation:

- Keys can be externally generated (e.g. using OpenSSL functions) and injected into the A71CH module using the A71CH Configure Tool contained in the A71CH Host software package [A71CH_HOST_SW].
- Keys can be directly generated within the A71CH security IC using the A71CH Configure Tool contained in the A71CH Host software package.

In any case, the A71CH security IC will store a pair of keys, formed by a public key and a private key. For instance, the next steps can be followed to create a pair of elliptical keys using OpenSSL commands on the server, then send them to the IoT device and finally inject them into the A71CH security IC using the A71CH Configure Tool (Fig 12).

3. Generate elliptical key pair with OpenSSL functionality on the Server:

```
openssl ecparam -name prime256v1 -out eccparams.pem
openssl ecparam -in eccparams.pem -genkey -noout -out IoTecckeys.pem
```

4. Send the key pair to the IoT node, execute the following commands with the A71CH Configure Tool to provision the A71CH with the already generated elliptical private and public keys and obtain the reference key file:

```
debug reset
set pair -x 0 -k IoTecckeys.pem
refpem -c 10 -x 0 -k IoTecckeys.pem -r IoTecckeys_ref.pem
info pair
```

The keys reference file *IoTecckeys_ref.pem* will be stored in the IoT device memory and further used by the A71CH OpenSSL Engine to indicate which key pair stored in the

A71CH is used to sign or verify a document, or to create a Certificate Signing Request with a public key.

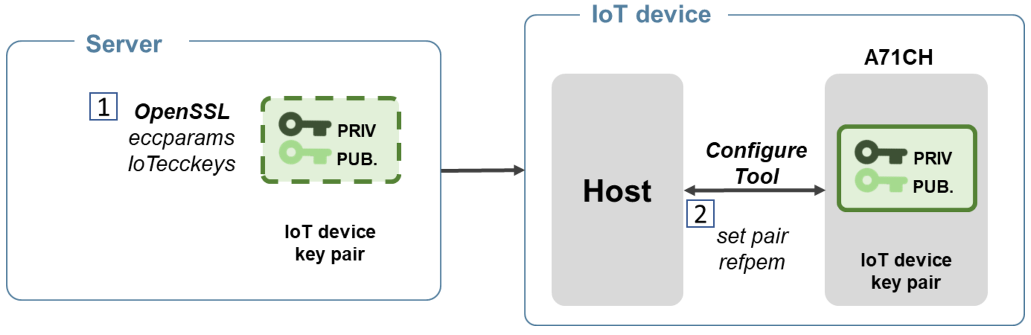


Fig 12. IoT device keys preparation. Option 1.

Alternatively, as has been mentioned, it is possible to generate an elliptical key pair directly in the A71CH security IC using only the A71CH Configure Tool on the IoT device. This would be the most secure option (Fig 13).

5. Execute the following commands with the A71CH Configure Tool to generate a key pair inside the A71CH and obtain the reference key file:

```
gen pair -x 0
refpem -c 10 -x 0 -r IoTecckeys_ref.pem
```

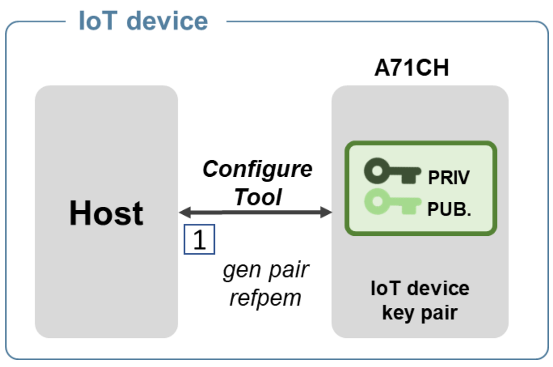


Fig 13. IoT device keys preparation. Option 2.

5.2 A71CH security IC and server digital certificate provisioning

The IoT digital certificate is generated by the server platform since it is signed by the CA. Once prepared, the signed certificate will then be sent to the IoT device, and optionally injected into the A71CH using the A71CH Configure Tool.

The A71CH Configure Tool provides support for certificate injection. It takes as input a certificate in PEM (Privacy Enhanced Mail) format, converts the certificate into DER (Distinguished Encoding Rules) and inserts it on an arbitrary offset position of the

General Purpose storage of the A71CH. Therefore, it is important to remark that every time the certificate is retrieved from the GP, it will have to be re-converted into PEM format. The following steps are followed for generating the IoT digital certificate.

6. Create a root CA key pair and root CA certificate on the Server in case it is needed (Fig 14).

```
openssl ecparam -in eccparams.pem -genkey -noout -out CArootecckeys.pem
openssl req -x509 -new -nodes -key CArootecckeys.pem \
-subj "/ CN=Host device" -days 2800 -out CArootCert.cer
```

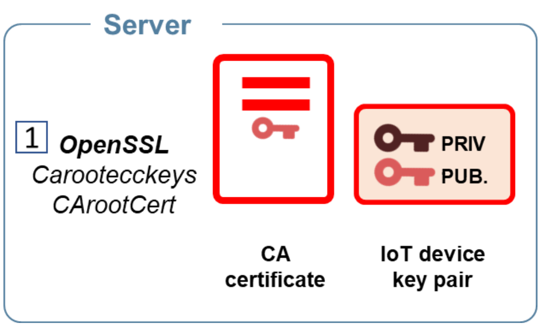


Fig 14. CA keys and certificate generation

7. Create the server key pair and digital certificate (Fig 15).

```
openssl ecparam -in eccparams.pem -genkey -noout -out Servereccкeys.pem
openssl req -new -key Servereccкeys.pem -subj "/CN=Server device info" -out
     ServerCSR.csr
openssl x509 -req -sha256 -days 2800 -in ServerCSR.csr -CAcreateserial -CA
     CArootCert.cer -CAkey CArooteccкeys.pem -out ServerCert.cer
```

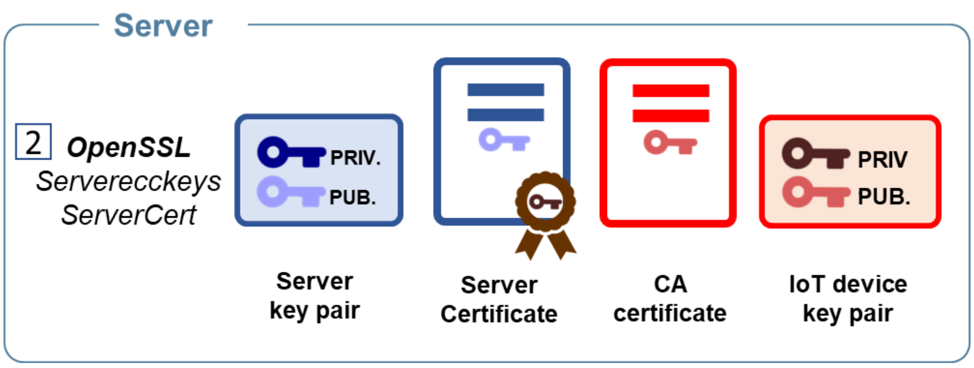


Fig 15. Host keys and certificate generation

8. Create the IoT CSR (3.2) in the IoT device using the A71CH OpenSSL Engine and the keys already stored in the A71CH module (Fig 16):

```
# configure OPENSSL_CONF environment variable with the A71CH OpenSSL Engine
# e.g., export OPENSSL_CONF = /etc/ssl/opensslA71CH_i2c.cnf
openssl req -new -key IoTecckeys_ref.pem -subj "/CN=IoT Device info" -out IoTCSR.csr
# Unset OPENSS_CONF variable
# e.g., unset OPENSSL_CONF
```

9. Send the IoT device CSR to the Server platform and create the IoT device certificate signed with the CA private key (Fig 16):

```
openssl x509 -req -sha256 -days 2800 -in IoTCSR.csr -CAcreateserial -CA
CArootCert.cer -CAkey CArootECCKeys.pem -out IoTCert.cer
```

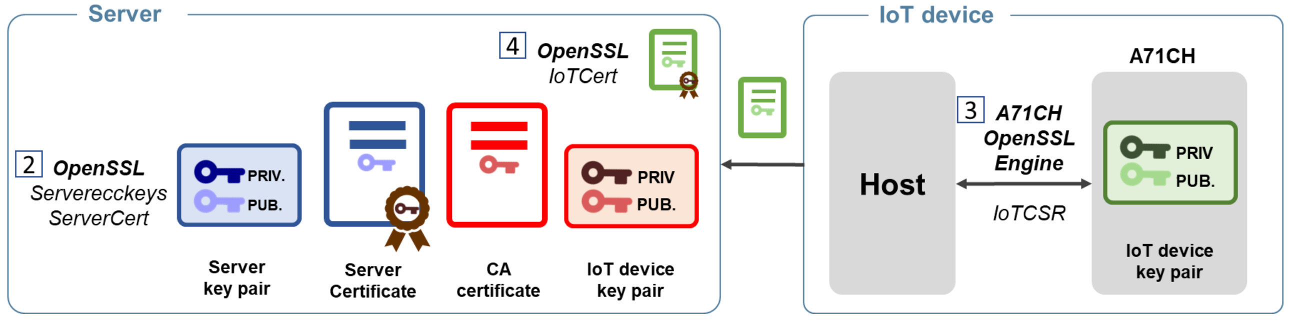


Fig 16. IoT device CSR creation and CA signature

10. Then, send the signed certificate back to the IoT device together with the CA certificate. Optionally, store both certificates in the A71CH security IC using the Configure Tool (Fig 17) as follows:

```
set gp -h 0400 -c CArootCert.cer
set gp -h 0000 -c IoTCert.cer
```

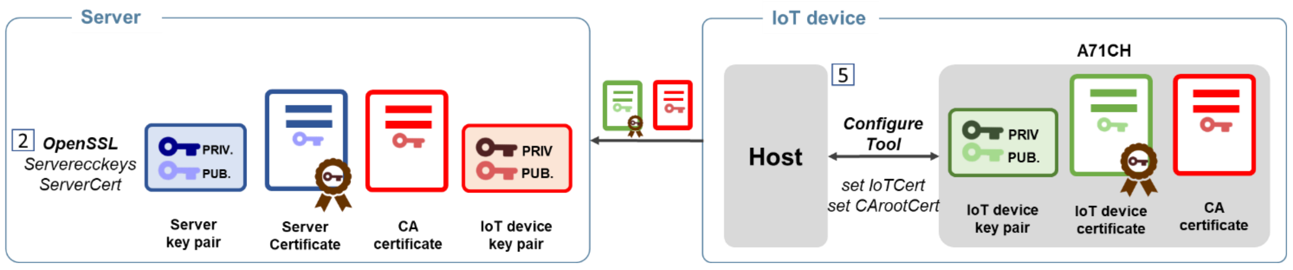


Fig 17. IoT certificate and CA certificate injection into the A71CH

An alternative method for creating the server and IoT device credentials (keys and certificates) and provision A71CH security IC can be found in the ***tlsCreateCredentialsRunOnClientOnce.sh*** and ***tlsPrepareClient.sh*** scripts included in the A71CH Host software package. In those scripts, all the credentials are generated in the same device, i.e., in the Server device (***tlsCreateCredentialsRunOnClientOnce)***.

Then, IoT device credentials are transferred from the server to the IoT device and further injected into the A71CH using the Configure Tool (***tlsPrepareClient.sh).***

5.3 Server authentication process

The server will be authenticated by the IoT device. Therefore, the IoT device will start by requesting the server digital certificate. Once the Server has sent its certificate, the IoT device will validate it using the CA public key contained in the CA certificate. Again, in case the CA digital certificate was stored into the A71CH GP Storage, it will be necessary to retrieve it using the Configure Tool and convert it from DER to PEM format. The following OpenSSL command verifies the signature of the Server certificate.

```
openssl verify -verbose -CAfile CArootCert.cer ServerCert.cer
```

If the server certificate has been validated, the IoT device will generate the random message to be sent to it, and again, the server will sign this message with its private key and send the signed message back to the IoT. Finally, the IoT device will attempt to validate the Host signature with the public key obtained from the Host certificate. Examples of how to sign and validate files using keys stored in the A71CH can be found in the OpenSSL example script ***a71chEccSignDemo.sh***

5.4 IoT device authentication process

Similarly, in order to authenticate the IoT node once the A71CH security IC has been provisioned, an IoT device public key verification is carried out. The process consists of making sure the IoT device public key belongs to the original IoT device OEM manufacturer. For this, the Server will verify the signature of the IoT digital certificate containing its public key.

The IoT device sends its certificate to the Server, which verifies it by using the CA public key. In case the IoT device digital certificate was stored into the A71CH GP Storage, it will be necessary to retrieve it using the Configure Tool and convert it from DER to PEM format. The IoT device certificate is authenticated if the signature is successfully verified and therefore its contents (device information and public key) can be correctly read.

The server device then generates a random message and sends it to the IoT device. This will sign the received random message (ECDSA) by using its private key stored in the A71CH. For this, the A71CH OpenSSL Engine functionality has to be enabled and will be used to access the A71CH and use the key reference file for signing:

```
# configure OPENSSL_CONF environment variable with the A71CH OpenSSL Engine

# e.g., export OPENSSL_CONF = /etc/ssl/opensslA71CH_i2c.cnf openssl dgst -ecdsa-with-
SHA1 -sign IoTecckeys_ref.pem -out signed_random.bin

# Unset OPENSS_CONF variable

# e.g., unset OPENSSL_CONF
```

The signed message will then be sent back to the Server, which will validate the signature.

More information about the OpenSSL, A71CH OpenSSL Engine and A71CH Configure Tool functionality can be obtained in A71CH Doxygen documentation [A71CH_HOST_SW] [OPEN_SSL] [A71CH_OPENSSL_ENGINE].

5.5 SCP03 Establishment between host MCU and A71CH

Optionally, the communication channel between the IoT device Host and the A71CH security IC can be secured by establishing a secure channel with SCP03. This can be done using the A71CH Configure Tool.

1. Write a set of SCP03 keys to the A71CH

```
scp put -h <hexvalue_keyversion> -k <keyfile>
```

Where the keyfile, containing ENC, MAC and DEK keys can be as follows:

```
# This is a comment, empty lines and comment lines allowed.
ENC AA112233445566778899AABBCCDDEEFF # Trailing comment
MAC BB112233445566778899AABBCCDDEEFF # Optional trailing comment
DEK CC112233445566778899AABBCCDDEEFF # Optional trailing comment
```

Once the set of SCP03 is stored in the A71CH. The secure channel can be established:

2. Establish an active SCP03 channel between the IoT device Host and the A71CH Security IC.

```
scp auth
```

6. Referenced documents

Table 1. Referenced Documents

[OPEN_SSL]	**OpenSSL Cryptography and SSL/TLS Toolkit information** - www.openssl.org
[SCP03]	**Global Platform Card Specification v2.3** – Amendment D v1.1.1.
[A71CH_OPENSSL_ENGINE]	**A71CH OpenSSL Engine** – DocStore, document number um4334**[1]
[RFC5246]	**The Transport Layer Security (TLS) Protocol** - Version 1.2, August 2008
[RFC4492]	**Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) -** May 2006
[A71CH_HOST_SW]	**A71CH Host Software Package (Windows Installer) –** DocStore, document number sw4673xx[1]**,** Version 01.03.00 (or later), available on www.nxp.com/A71CH **A71CH Host Software Package (Bash installer) –** DocStore, document number sw4672xx[1]**,** Version 01.03.00 (or later), available on www.nxp.com/A71CH

[1] **… document version number

7. Legal information

7.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

7.2 Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

7.1 Licenses

ICs with DPA Countermeasures functionality

NXP ICs containing functionality implementing countermeasures to Differential Power Analysis and Simple Power Analysis are produced and sold under applicable license from Cryptography Research, Inc.

7.2 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

FabKey — is a trademark of NXP B.V.

I²C-bus — logo is a trademark of NXP B.V.

8. List of figures

9. List of tables

10. Contents