Design-in Application Note for OPTIGA™ Trust E

Implementation guideline for System-Integration

About this document

Scope and purpose

This document explains the benefit of Certificates, the usage of a Public Key Infrastructure (PKI), secure storage of private keys and necessary assertions on host side. It explains which aspects must be considered when developing software which uses the OPTIGA™ Trust E.

Table of Contents

1 Public Key Infrastructure

Digital certificates are electronic documents which are readable and changeable. A certificate can be compared to an unprotected text file which may be saved in unprotected memory. The file contains a certain structure according to the ASN.1 standard. The top-level structure of a certificate contains three fields: tbsCertificate, Signature Algorithm and Signature.

Even though one can read and change the content of a certificate it is possible to use certificates to verify the authenticity of a device. This is done exploiting properties of asymmetric cryptographic algorithms.

This chapter explains basic definitions which are used throughout this Application Note (1.1), which fields are used in a certificate (1.2), how a certificate is created (1.3) and how the chain of trust is established (1.4).

1.1 Definitions

This Application Note uses terms, commands, abbreviations and symbols. To avoid misunderstanding this section provides explanations for them.

Issuer, Certificate Authority (CA): The issuer of a certificate has issued and signed a certificate with his private key. A root certificate authority (Root CA) is signing the own certificate (self-signed).

Client: A client needs either a CA or a Root CA to get his certificate signed. When a client signs other certificates with his private key, he becomes a certificate authority.

Client N: Since it is possible to establish a chain of trust, the members of a chain are called clients. To distinguish between clients in a chain, the clients have integers (Client 1, Client 2, …).

Certificate: A certificate is an electronic document.

1.2 Certificate Fields

A certificate in the X.509 standard (https://tools.ietf.org/html/rfc5280) contains at top-level three basic fields (Figure 1).

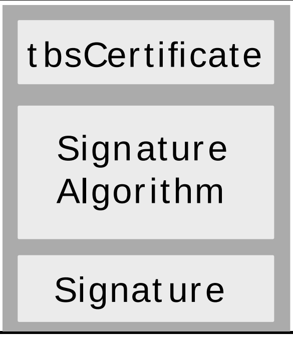

Figure 1 Basic Certificate Fields according to X.509 standard

1.2.1 To Be Signed Certificate Field

The "To Be Signed Certificate" Field (tbsCertificate) contains primarily information about the public key, the subject, the issuer and a validity period. All included values are hashed and used as input for the signature.

1.2.2 Signature Algorithm

This field contains information about the algorithm that was used by the certificate authority (CA) to sign the certificate.

1.2.3 Signature Value

The signature value contains the output of the private key operation (e.g. RSA decryption) of the signature algorithm applied by the Issuer. It is encoded as an ASN.1 string and set as signature value in the certificate.

1.3 Certificate Creation

This section describes how a certificate is created and signed by a certificate authority. In the following sections there are two parties which exchange files. There is always a client and a server which interact with each other. In the context of certificates, a client is usually called a subject and a server is usually called a certificate issuer or certificate authority. It is possible to establish an interaction chain where the server and a client exchange messages and also two clients can exchange messages in a strict top-down or bottom-up mechanism. To distinguish between clients, the clients are enumerated. A chain of four parties might look like this:

Server <-- >Client 1 < -- > Client 2 < -- > Client 3

After establishing the chain between the Server and Client 1, the latter operates as the Server and Client 2 is the client.

In the following steps we assume that both the Server and Client 1 do not have any certificates or private keys stored. The steps show how a non self-signed certificate is created.

- **Create asymmetric key pairs**
  The Server and Client 1 have to create a unique private and public key pair to be able to encrypt and decrypt data.

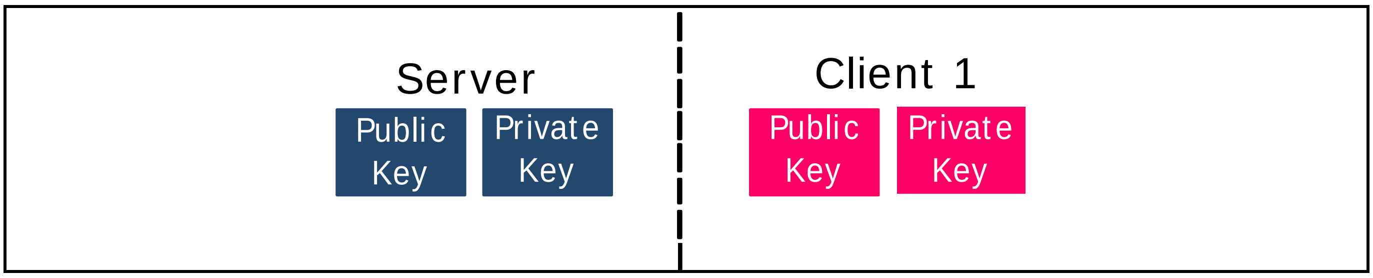

Figure 2 Unique asymmetric key pair required on Server and Client side

- **Certificate file creation**
  Client 1 creates a file according to the X.509 standard (ASN.1 encoded). It must contain all information necessary for the tbsCertificate field and may contain some additional information. Additionally, the signature algorithm must be set (Figure 3). To depict the key dependency of the client and server, Figure 3 shows the public key of the client which is stored in a certain sub-field of the tbsCertificate basic field.

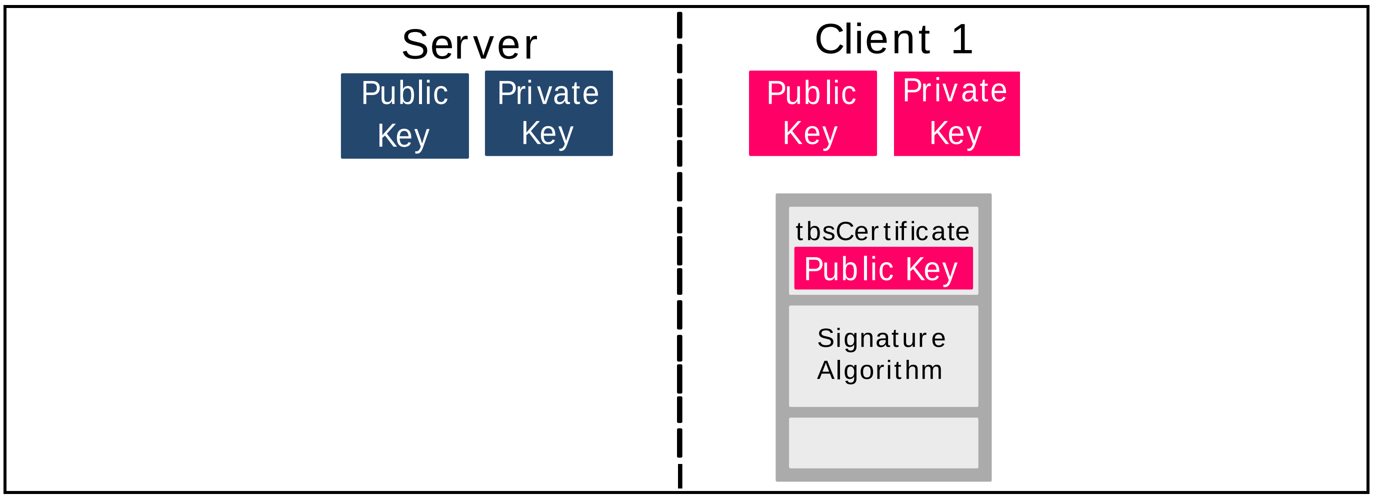

Figure 3 **Client 1 prepares unsigned Certificate content**

- **Send certificate file to Server**

  The certificate file contains two of the three basic certificate fields. The third field (signature value) can only be created by the Server with its private key. Therefore, Client 1 sends the unsigned certificate to the server on a secured channel to ensure integrity of the file.
- **Tasks on Server-Side**

  The Server takes the tbsCertificate field and calculates a hash value which is used as input for the signature algorithm specified in the second basic certificate field (Figure 4).

  Theoretically, it would also be possible to send the private key to Client 1. It could then calculate the signature value itself. As all trust depends on keeping the private key of the Server secret, sending it even on a secured channel must not be done.

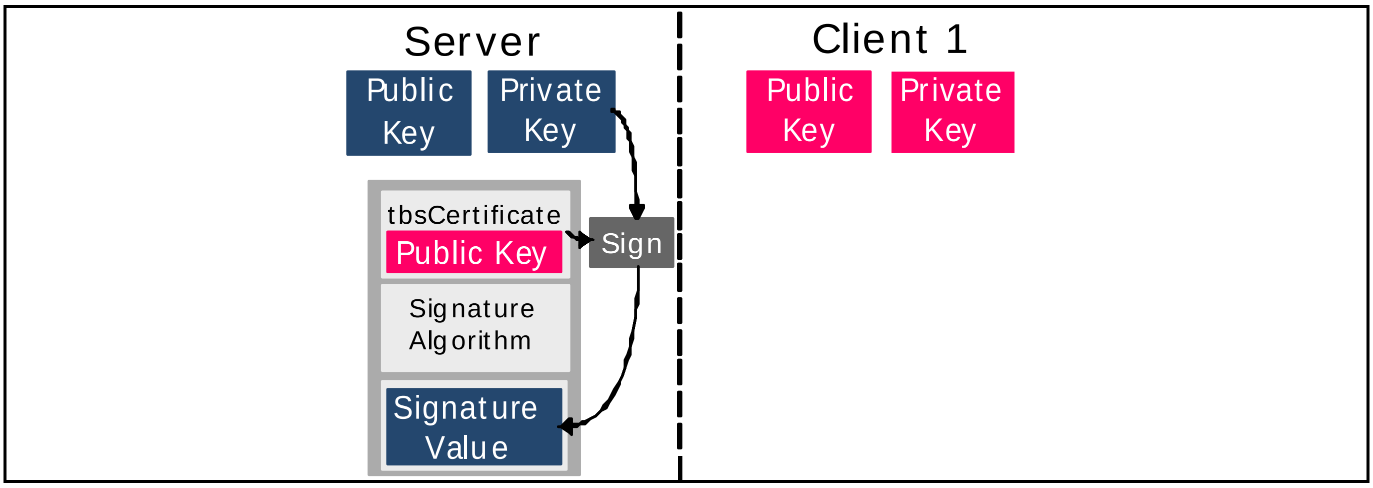

Figure 4 **Sign Certificate on Server side with private key of Server**

- **Send certificate back to Client 1**

  The signed certificate is now sent back to the client. To protect the integrity of the certificate after sending it over a communication channel, it can either be encrypted and decrypted or validated using the public key of the server.

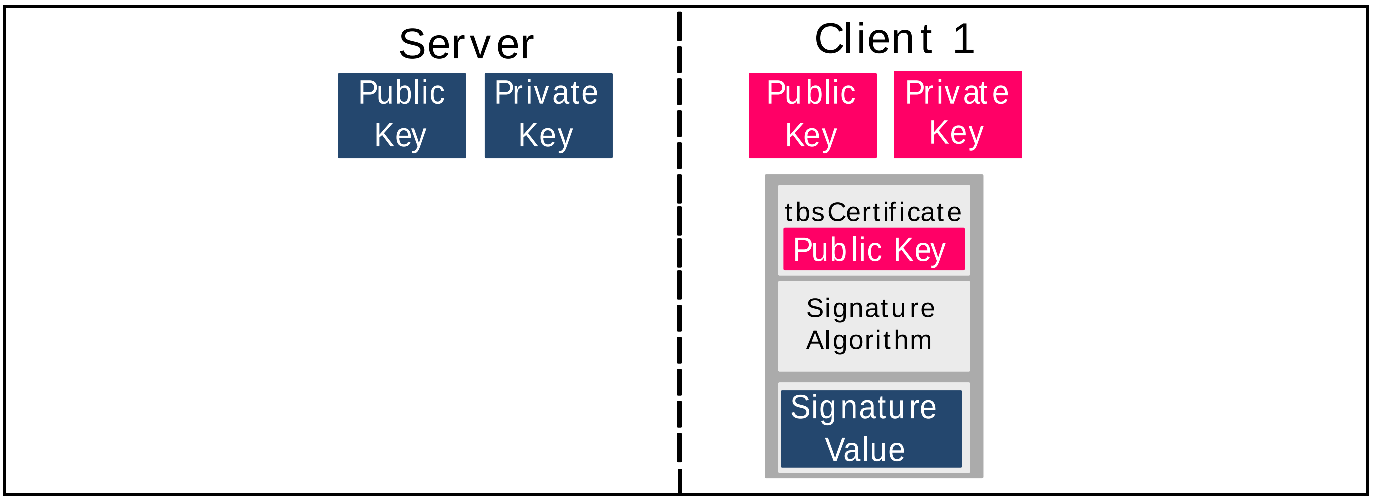

Figure 5 Signed Certificate is stored on Client 1

1.4 Chain of Trust

Getting a certificate with a signature from a server or certificate issuer can be used to create a Chain of Trust. The basic principle of signing a certificate is extended to more participants. Every Client which is not at the end of a chain (also called Leave) can sign a certificate from a "lower" participant. Figure 6 depicts the Chain of Trust for one Server and two Clients. Client 2 has a signature from Client 1, Client 1 has a signature from the Server and the Server has a self-signed certificate.

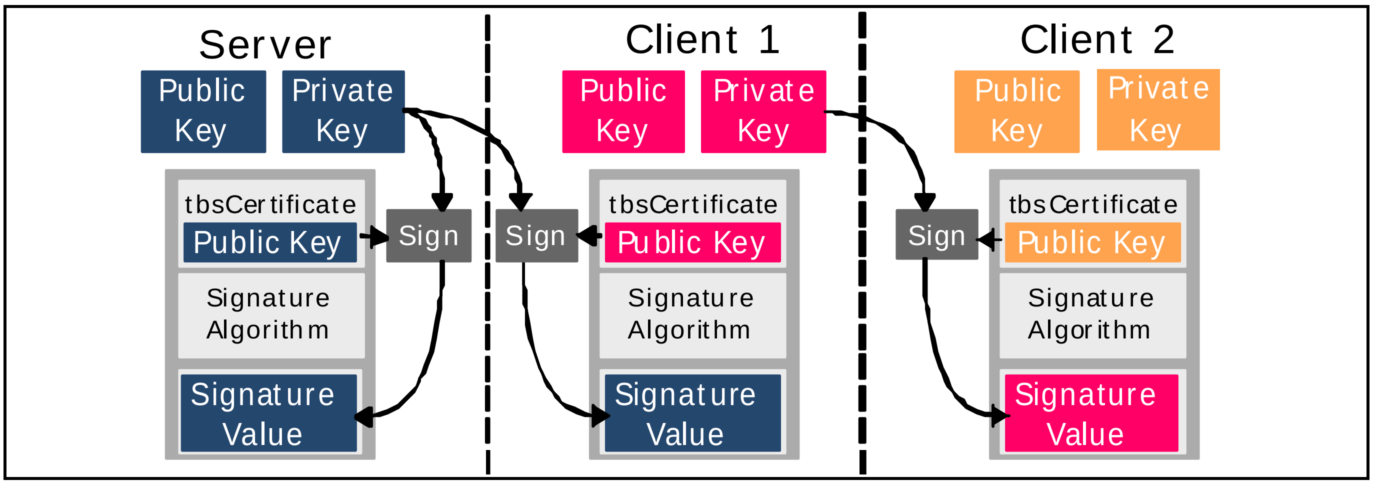

Figure 6 Chain of Trust with three participants

1.4.1 Verify Root Certificate

A Root Certificate is self-signed with the private key from the root certificate authority (Figure 6: Server). The integrity of the certificate can be verified by calculating the hash-value of the tbsCertificate field and encrypting the Signature Value using the public key from the root certificate. If both values are equal the content of the certificate was not altered. A Root Certificate must be provided in a trusted way that a recipient can check that the Root Certificate was not altered.

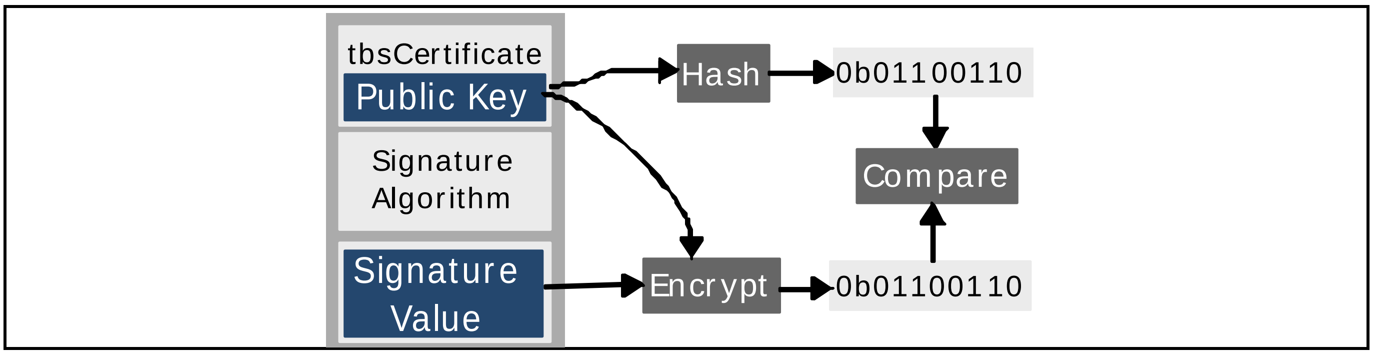

Figure 7 Verify self-signed Root Certificate

1.4.2 Verify Non-Root Certificate

A non-root certificate is signed with the private key of a certificate issuer. To verify the signature value, the tbsCertificate field of the client must be hashed and the signature value must be encrypted using the public key of the issuer. In Figure 8, the certificate issuer and client are the Server and Client 1. Server certificate must be locally available at Client 1 to extract the issuer's public key.

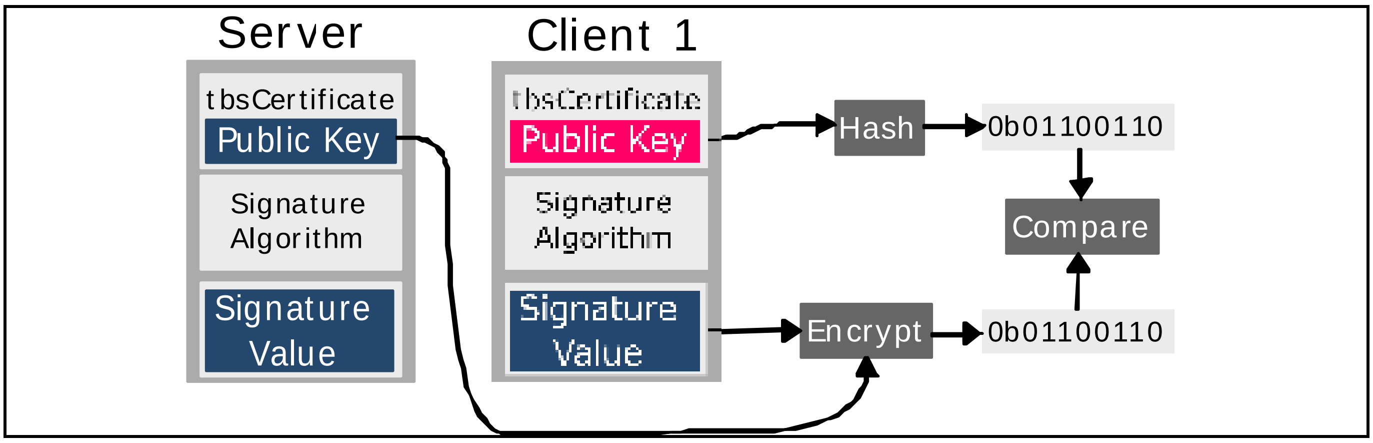

Figure 8 Verify certificate signed by certificate issuer

2 OPTIGA™ Trust E

In authentication applications, there are many different terms which are used to explain how the Public Key Infrastructure works or what a Host in a Sensor Network has to do. Table 1 maps different levels of a generic top-down view of a certificate hierarchy to the appropriate application levels. The first line shows a very generic approach how such an infrastructure can be named. The self-signed root certificate is located in "Root" and all subsequent certificates are located in lower levels where "Level 1" is the highest, "Level 2" is the second highest and so on. The lowest certificate level is "Level N". In Section 1, the different levels are named according to the second row in Table 1. The third row shows how certificates are called which belong to the different levels. One major scope of application for authentication devices are sensor networks, e.g. in smart homes. Usually, there is one Cloud service which allows monitoring status values or executing commands remotely via smartphone. The Host summarizes all measured values available in the sensor network and sends them to the Cloud. The values are delivered by the sensor nodes connected to the host.

Table 1 Mapping of different Certificate Levels to different Applications

Generic	Root	Level 1	Level 2	Level N
PKI	Server	Client 1	Client 2	Client N
Certificates	Root Certificate	First Intermediate Certificate	Second Intermediate Certificate	Leaf Certificate
Sensor Network	Cloud	Host	Sensor Node 1, Sensor Node 2, …	-

2.1 Commands

The OPTIGA™ Trust E has an API which features seven commands. They allow to read and store data objects, generate a bit stream with random numbers and to execute an one-way authentication scheme. All commands are shown in detail in the Solution Reference Manual [1].

In this Application Note, only the commands needed for the one-way authentication protocol are considered in more detail.

2.1.1 Open Application

The OpenApplication command is executed to open the authentication application. It is needed at the beginning of the authentication scheme.

2.1.2 Read General Purpose Data

This command is used to get the certificate from the OPTIGA™ Trust E. The certificate contains the public key and is stored inside the OPTIGA™ Trust E. It cannot be altered due to restrictive access rights. The belonging private key remains in the authentication device and cannot be read.

2.1.3 Set Authentication Scheme

This command is used to set the authentication scheme (elliptic curve cryptographic algorithm) for the following authentication sequence.

2.1.4 Set Authentication Message

This command sends a random number from the host to the OPTIGA™ Trust E.

2.1.5 Get Authentication Message

This command is used to get the calculated signature from the OPTIGA™ Trust E.

2.2 One-Way Authentication

The One-Way Authentication scheme needs the commands from subsection 2.1 and uses properties of certificates explained in section 1. Figure 9 depicts the right order of commands needed for the one-way authentication scheme.

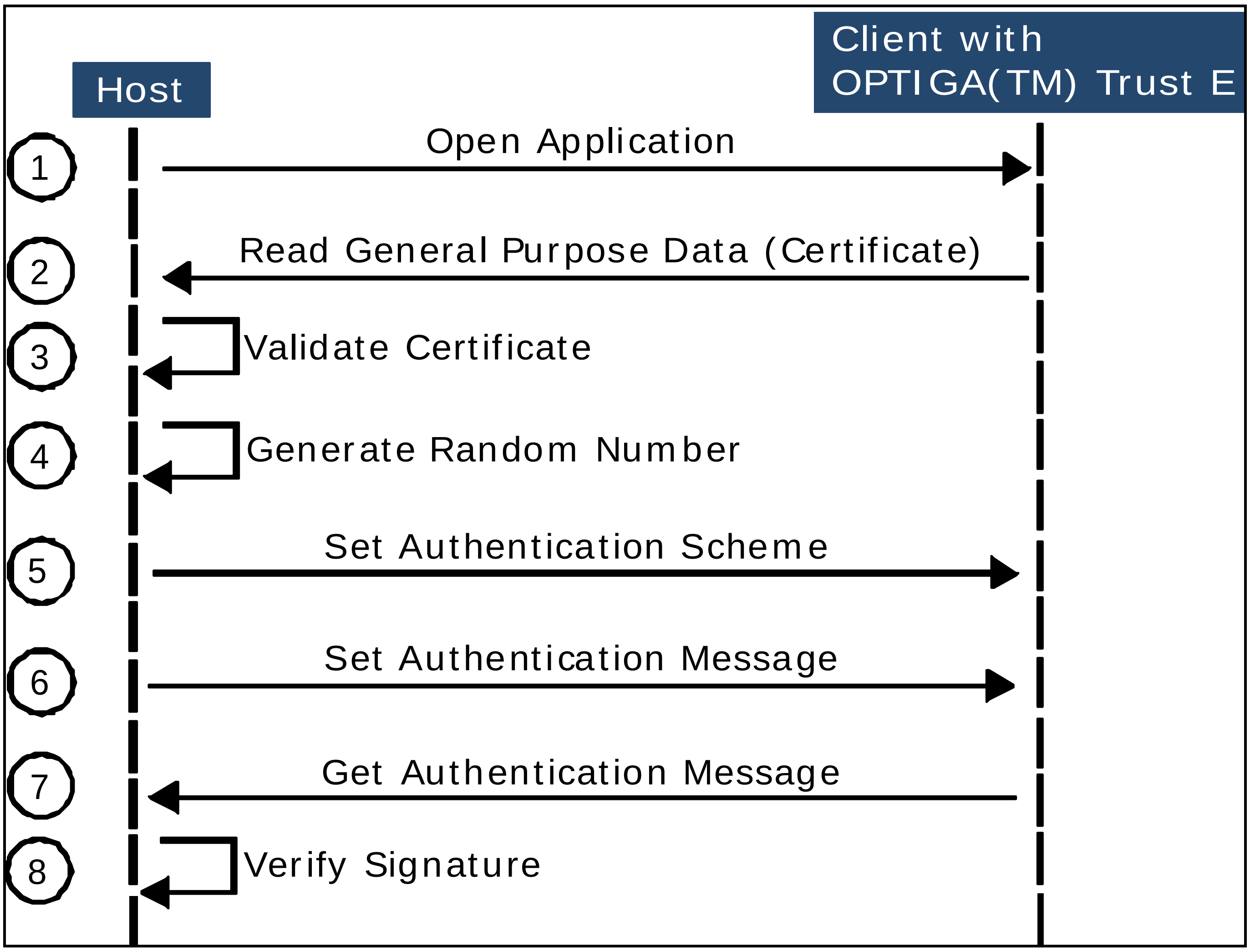
Host
Client with OPTIGA(TM) Trust E
1
Open Application
2
Read General Purpose Data (Certificate)
3
Validate Certificate
4
Generate Random Number
5
Set Authentication Scheme
6
Set Authentication Message
7
Get Authentication Message
8
Verify Signature

Figure 9 **One-Way Authentication Scheme for the OPTIGA™ Trust E**

- **1 – 3:** The Host sends an "Open Application" command to the Client to start the authentication process. It then reads (**Step 2**) and verifies the certificate. To verify the Leave Certificate from the sensor node all intermediate certificates and the (trusted) root certificate is required on Host side. The verification chain is shown in Figure 10. It is possible to store all certificates necessary for the verification process in memory permanently or reload them dynamically. In case of a dynamic reloading scheme due to limited memory size for example, a connection to certificate servers where the certificates are stored is mandatory. Whether the one or other solution is chosen doesn't increase or decrease the security level.
- **4 – 8:** The security of the authentication scheme depends heavily on the quality of random numbers. In case of a bad quality, e.g. a list of numbers which are repeatedly sent to the node can be exploited in replay attacks. As an attacker can read the content sent to the nodes and the reply message coming from the node, a look-up table with plain- and cipher values can be created and used for authentication of fake devices.

Therefore, **step 4** deserves the right amount of effort even it might look trivial at first. The random number is then sent to the sensor node and signed within the OPTIGA™ Trust E with its private key. In the last step the signature is verified using the public key from the previously received device certificate. To authenticate a client successfully, the decrypted signature message and the originally sent random number must be equal.

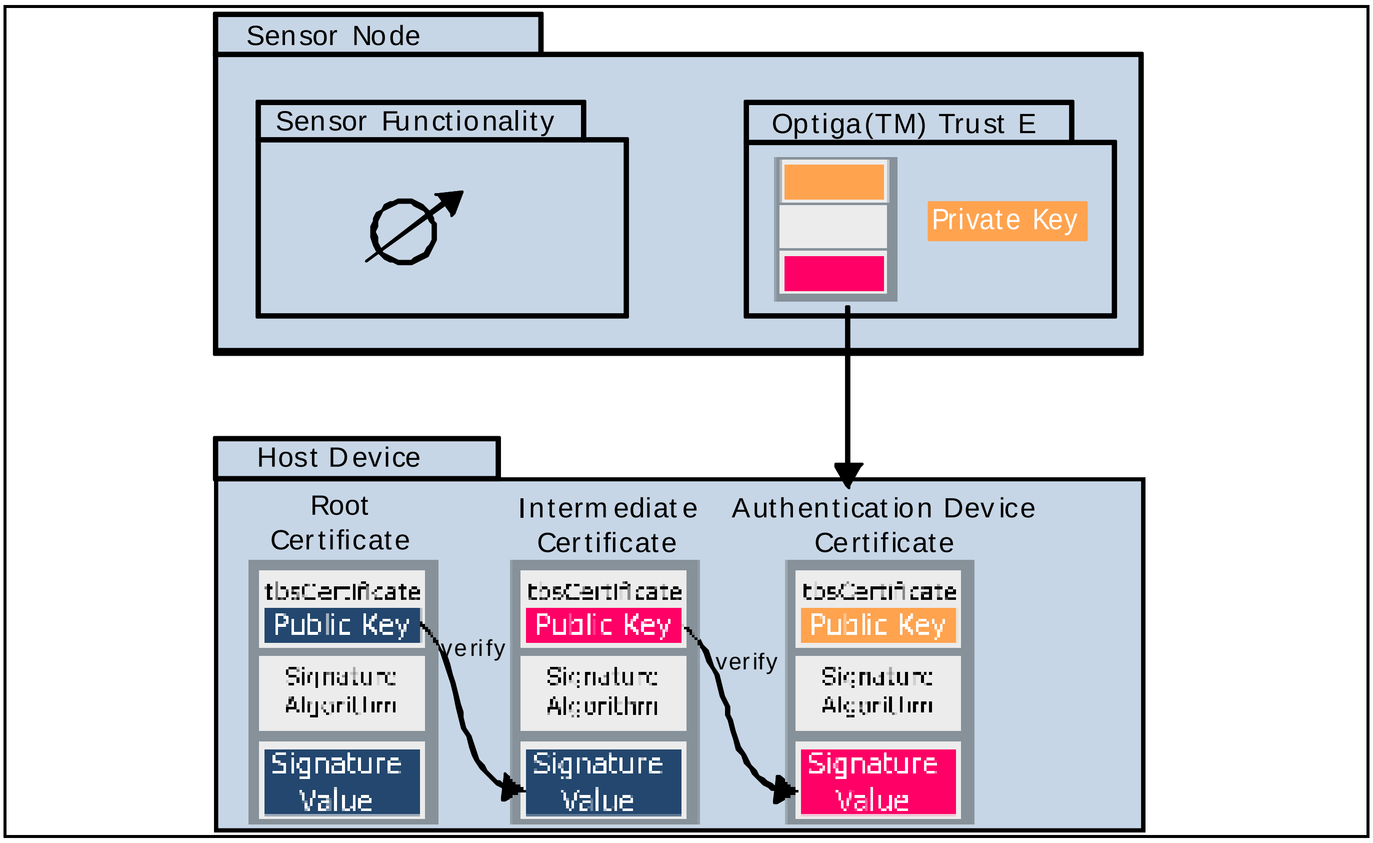

Figure 10 Sensor Note with OPTIGA™ Trust E for One-Way Authentication Scheme

3 Attack Vectors on Authentication Schemes

Public Key Infrastructures depend heavily on keeping the secret keys which belong to the certificates in a chain of trust secret. This can either be achieved keeping all devices with secret keys strictly separated in an encapsulated environment, or protecting the secret key from attackers on devices, even if they have physical access (Subsection 3.1).

The secret keys which belong to Leave Certificates can be used to clone devices. A host relying on the authentication scheme of e.g. sensor nodes cannot differentiate between an original device using the apparently unique key and clone devices using the same secret key. In case the secret key of an Intermediate Certificate is available for an attacker, faked Certificates can be created and signed. This makes it more difficult for a Host to recognize fake sensor nodes (Subsection 3.2).

Even though an attacker might not be able to read out the secret key of a sensor node, there are still possibilities how an authentication scheme might be compromised in case of a poor PKI verification implementation. This presumes that either the program flow can be changed or that memory locations can be overwritten (Subsection 3.3).

3.1 Beneficial Environment for an OPTIGA™ Trust E

Whenever two parties communicate with each other, there is a need to assert that the opposite communication partner is the one he intends to be. As an example, think about the communication between two persons. The authenticity of two persons talking to each other can be asserted easily when they talk face to face; it is still easy when they talk on the phone, because they recognized each other's voices and sentences. Writing messages using a digital communications channel makes it very difficult for persons to assert that a message was written by a certain person. These messages can be changed or faked. One possibility to verify the authenticity is to ask a certain question which can only be answered by that person. Adding the answer to the message increases the confidence in a message dramatically.

A company developing sensor nodes does not have to think about enhanced authentication mechanisms when the device will operate in an encapsulated environment where an attacker has no physical access. In this scenario two devices can simply exchange IDs to verify each other's identity. It can be compared to two persons talking face to face or on the phone.

As soon as a sensor node will operate in an unsecure or undefined environment, relying on the exchange of IDs is insufficient to verify each other's identity because every communication partner can send arbitrary IDs. Taking the example of two persons talking with each other, adding a question and a secret answer to the message can enhance the trust if there are enough questions and answers and if the answers are kept secret. Applying this analogy to the usage of embedded devices, the question is a random number and the secret answer is the encrypted random number using the secret key. As shown previously in section 2.2 comparing the original random number with the decrypted message is necessary to authenticate a communication partner unambiguously.

3.2 Attack Vectors on Secret Key Storage Devices

Whenever an attacker has physical access to a device which is used in an authentication scheme using a private key, the secret is a potential attack target. There are plenty of reasons, why an attacker might try to read out a secret key – the most reasonable is probably an economic one. Knowing the secret key allows reproducing fake devices which can be used to authenticate themselves successfully against a Host device.

The USB 3.1 Standard [2] includes the possibility to enhance devices with a hardware authentication scheme as it is shown in this document. The appendix of this document contains a very detailed list of potential attack vectors. A portion of it is listed here:

- Side Channel Attacks (e.g. Timing-, Power-, Electromagnetic Emanation, Photoemission analysis)
- Fault Induction Attacks (e.g. Temperature, Voltage, Electromagnetic Induction, Laser Attacks)
- Manipulative Attacks (e.g. Microprobing, modification of circuits using a focused ion beam)

3.3 Attack Vectors on Intermediate Certificate Devices

The assumption in the previous subsection is that an attacker has physical access to a device. This allows an attacker to apply highly sophisticated attack mechanisms. On top of that, another possibility to compromise a sensor network is to attack the Host via internet. A Host system must be designed in way that modifications can be identified. If an attacker can influence the random number generation on Host side or can skip the signature verification, the system security is directly affected.

References

[1] Title: OPTIGA™ Trust E SolutionReferenceManual, Version: 1.22, Release Date: 2016-01-20

[2] Title: Universal Serial Bus Type-C Authentication Specification, URL: http://www.usb.org/developers/docs/, Version: 1.0, Release Date: 2016-03-25

Revision History

Major changes since the last revision

Page or Reference	Description of change
V1.0	Initial Document

Edition 2017-01-20

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
Version 1.0