2014•2015
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: elektronica-ICT

Masterproef deel 1

Public-key cryptography with mutual authentication for IoT platform

Promotor :
Prof. dr. Nele MENTENS

Promotor :
Dhr. TEEMU HAKALA

Copromotor :
Dhr. EERO HAKALA

Jori Winderickx

Eerste deel van het scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

KU LEUVEN | universiteit hasselt

KU LEUVEN | universiteit hasselt

2014•2015

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: elektronica-ICT

Masterproef deel 1

Public-key cryptography with mutual authentication for IoT platform

Promotor :
Prof. dr. Nele MENTENS

Promotor :
Dhr. TEEMU HAKALA

Copromotor :
Dhr. EERO HAKALA

Jori Winderickx

Eerste deel van het scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: elektronica-ICT

universiteit hasselt | KU LEUVEN

Preface

At the start of this semester I had the opportunity to go abroad for my thesis in a beautiful country named Finland. This would not be possible without the UHasselt, I would like to thank them for this opportunity. And I am also grateful for the help of Vincent Boerjan and Dr. Tech. Antti Ylä-Jääski from Aalto University in helping me find the right thesis.

During my internship I was able to work among the great team of ELL-i with Teemu Hakala as my mentor. And I would like to express my gratitude to Teemu Hakala for his guidance to make the best out of this thesis. Another person who helped me with my thesis is D.Sc. Arto Karila and I would like to thank him for the insight he gave me about my thesis. Then I also would like to thank my internal promotor Dr. Nele Mentens for her help.

Contents

1 Introduction 15
1.1 Problem definition 15
1.2 Goal 16
1.3 Materials and methods 16

2 Theory 17
2.1 Constrained Application Protocol 17
2.2 Cryptography 17
2.2.1 Symmetric-key encryption 18
2.2.2 Asymmetric-key encryption 19
2.2.3 Public Key Infrastructure 21
2.3 IP security 24
2.3.1 Cypher suites of DTLS 25

3 IP security 27
3.1 PSK, RPK and certificates 27
3.2 Public-key algorithms 27
3.2.1 Key size 27
3.2.2 Efficiency 27
3.3 Certificates 28
3.3.1 Differences in certificate 28
3.3.2 Differences in network of trust 28
3.3.3 Differences in revocation procedure 28
3.3.4 Size comparison 29
3.4 Footprint 29
3.5 Libraries 30

4 The system 31
4.1 Cloud Service 31
4.2 Leaf node 32
4.3 Site Controller 33

5 Conclusion 35

Tables

Table 1: NIST-recommended key sizes in bits 27
Table 2: Public-key part of DTLS handshake using software ECC and RSA on OPAL*[17]*..... 28
Table 3: Comparison of certificate sizes of implicit certificate(ECQV), ECC based certificate(ECDSA) and RSA based certificate*[29]*..... 29
Table 4: Comparison of libraries..... 30

Figures

Figure 1: Diagram of the system of ELL-i[1] 15
Figure 2: A taxonomy of cryptographic primitives[30] 18
Figure 3: The structure of CBC encryption. The last cipher block servers as CBC-MAC[12] 19
Figure 4: CTR encryption and decryption mode[12] 19
Figure 5: Encrypt new key using the public key of B[31] 20
Figure 6: Diffie-Hellman key agreement scheme[31] 20
Figure 7: Representation of a X.509 certificate[22] 22
Figure 8: Network of trust of X.509 certificates[22] 22
Figure 9: The PGP certificate on the left; Web of trust on the right[22] 23
Figure 10: Flow of an DTLS handshake[15]. Messages with an '*' are optional 25
Figure 11: RAM overhead of the certificate based DTLS handshake, the symmetric-key based handshake and the delegation architecture[15] 29
Figure 12: ROM overhead of the certificate based DTLS handshake, the symmetric-key based DTLS handshake and the delegation architecture[15] 30
Figure 13: The Cloud Service program 31
Figure 14: The program of the Leaf node 33
Figure 15: Interface of DTLS in Node.js 33
Figure 17: Process of the Leaf node connecting to the Site Controller 34
Figure 16: The structure of the Site Controller's program 34

Abbreviations

AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CFB	Cipher Feed Back
CoAP	Constrained Application Protocol
CTR	Counter
DANE	DNS-Based authentication of Named Entities
DES	Data Encryption Standard
DH	Diffie-Hellman
DNSSEC	The Domain Name System Security Extensions
DoS	Denial-of-Service
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
HTTP	Hypertext Transfer Protocol
IETF	The Internet Engineering Task Force
IoT	Internet of Things
LDAP	Lightweight Directory Access Protocol
MAC	Message authentication code
MD5	Message-digest algorithm
MITM	Man In The Middle attack
OFB	Output Feed Back
PFS	Perfect Forward Secrecy
PKC	Public Key Cryptography
PKI	Public Key Infrastructure
PRF	Pseudorandom Function
PSK	Pre-Shared Key
RAM	Random-access memory
RC4	Rivest Cipher 4
REST	Representational State Transfer
ROM	Read-only memory
RPK	Raw Public Key
RSA	Ron Rivest, Adi Shamir and Leonard Adleman
SHA	Secure Hash Algorithm
SOAP	Simple Object Access protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
XOR	Exclusive or

Abstract

The distributed control system of ELL-i open source co-operative consists of 3 parts: the Leaf node, the Site Controller and the Cloud Service. The Leaf node is an constrained device with limited ROM and RAM and uses CoAP for M2M communication. The Site Controller can use the Leaf node and potentially the Cloud Service for establishing the communication with the Leaf node. The only problem with this situation is that they communicate over the unsecure local network and therefore requires additional security. Besides the security, the Leaf node should be able to automatically connect with the Site Controller and exists only of open source software.

The preferred security of CoAP is DTLS because of its handling of UDP. Most usages of DTLS are based on pre-shared key and raw public key in a constrained environment. A less used protocol for constrained environments is public-key cryptography and certificates. With WolfSSL, the best DTLS library as seen by the comparison, public-key cryptography can be implemented with a small footprint. With the X.509 certificate already embedded in the WolfSSL library, both devices can establish a mutual authenticated communication. However, according to the comparison of the certificates, the implicit certificate is more efficient for these environments. Furthermore, the elliptic curve cryptography is the best public-key algorithm for embedded devices.

Abstract in het Nederlands

Het gedistribueerd controlesysteem van ELL-i open source co-operative bestaat uit 3 delen: de Leaf node, de Site Controller en de Cloud Service. De Leaf node is een zeer simpel apparaat, heeft weinig RAM en ROM ter beschikking en gebruikt CoAP om te communiceren. De Site Controller kan de Leaf node aanspreken en eventueel de Cloud Service gebruiken om de communicatie te bevestigen met de Leaf node. Het enige probleem met deze situatie is dat de communicatie zich op het lokaal netwerk bevindt. Waardoor dit moet beveiligd worden tegen vreemden. Daarbovenop moet de Leaf node automatisch verbinden met de Site Controller en bestaan uit open source software.

De geprefereerde beveiliging voor CoAP is DTLS vanwege zijn omgang met UDP. De protocollen voor DTLS zijn vaak gebaseerd op pre-shared key en raw public key. Een veel mindere gebruikte vorm voor Leaf nodes met beperkingen is het gebruik van certificaten en public-key cryptografie. Met WolfSSL als meest geschikte DTLS bibliotheek voor embedded systemen kan public-key cryptografie geïmplementeerd worden met een kleine voetafdruk. Met de certificaat type X.509 dat al geïmplementeerd is in WolfSSL kan de wederzijdse verificatie bekomen worden. Echter geeft de vergelijking van certificaten aan dat impliciete certificaten efficiënter zijn voor deze situatie. Daarnaast komt elliptic curve cryptografie uit als de beste kandidaat voor public-key cryptografie.

1 Introduction

This thesis is made for ELL-i open source co-operative[1]. The idea of the system of ELL-i consists of constrained Leaf nodes controlled by the Site Controller and a Cloud Service which can be used as an trusted identity. The Leaf nodes are powered through the use of Power-over-Ethernet which means that the power is transmitted over the same cable as the communication. The software protocol used is Constrained Application Protocol(CoAP)[2], this protocol is specifically designed for lightweight communication. In addition this system is implemented within the local network. Furthermore, this structure causes the problem that the communication between the Leaf nodes and the Site Controller should be secure against outsiders trying to listen or even modify the data. Another aim of ELL-i is to have Leaf nodes that can automatically connect to the system. Their fundamental goal is to use only open-source code.

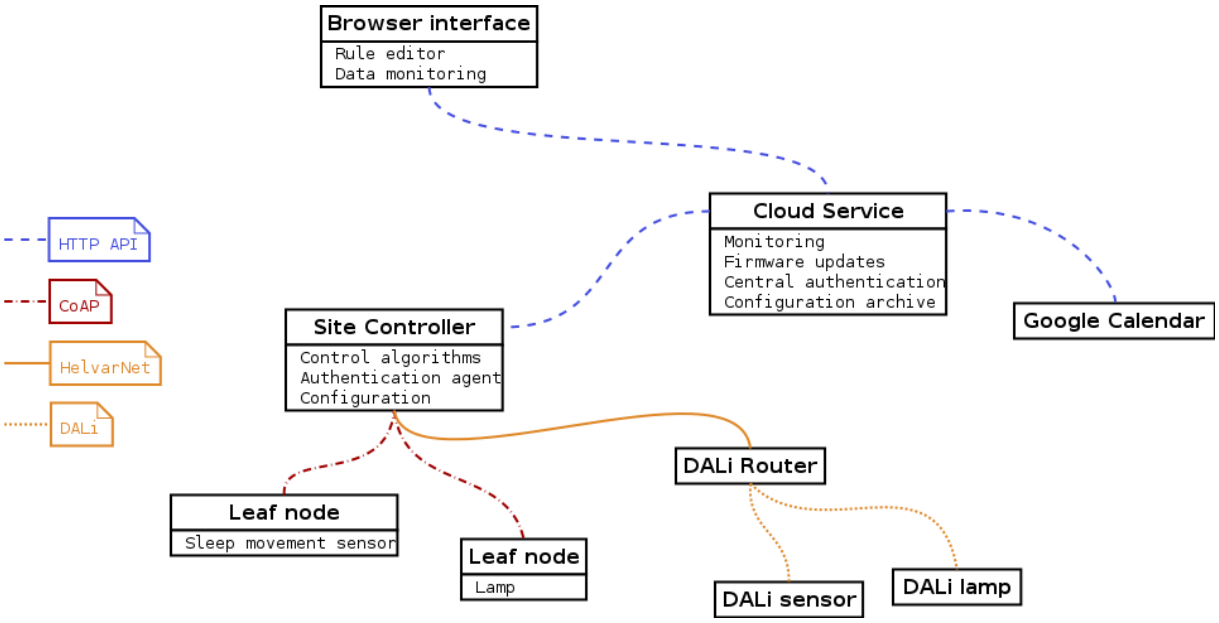

Figure 1: Diagram of the system of ELL-i[1]

1.1 Problem definition

The system of ELL-i is not safe to use in a network connected to the internet. The constrained Leaf node needs enough security with a strong defence against outsiders while permitting a Leaf node to be automatically configured into the system. The first problem is that the Leaf node is constrained and therefore should not use the entire storage of the Leaf node for security coding. Secondly, the Leaf node has to be authenticated by the Site Controller and vice versa so that the system has only trust worthy objects in its system. In order to have mutual authentication a trusted party or an common known key is needed. The trusted party or the common key has to be configured on the node prior to the bootstrap of the node with the Site Controller. If the Leaf node should work in any environment this trusted identity must be configured before use. Finally, because the Cloud Service does not yet exist, a basic web service should be implemented. In conclusion the most important part of this thesis is to implement the system with the goal of authentication and as a result having an automatically configured node.

1.2 Goal

The primary goal is to have the basic functionalities, firstly a Cloud Server that provides the Site Controller with the necessary certificates and the needed information for the communication between the Site Controller and the Leaf node. Secondly the Site Controller must be able to communicate with the Cloud Service over an Secure Shell(SSH) tunnel and with the Leaf node over CoAP with DTLS security. And lastly the Leaf Node should work like a RESTful(Representational state transfer) service where predefined resources and configurations can be read or even modified and in addition the Leaf node should have an reset functionality. The reset functionality erases all the keying material and certificates used in previous communications and lets the Node generate a new public-key pair in order for the Leaf node to work in all environments. For the DTLS secured communication, with and without the trusted identity stored in the Leaf node, two basic cipher suites should be implemented. The first one to configure an trusted identity and the second one to bootstrap the Leaf node with the Site Controller. Furthermore, the Leaf node should be developed using libraries made for embedded devices and consequently using no more than 42 kB of code for the entire certificate based handshake.

1.3 Materials and methods

Using the operating system of choice by ELL-i namely RIOT(The friendly Operating System for the Internet of Things)[3], cryptography libraries should be compared in function of size and peer reviewers. The size should be small and with a lot of peer reviewers so that the library is known to be safe.

The security should be implemented with a public-key based cryptography as bootstrap while using the selected cryptography library. In addition a symmetric cryptography can be used to configure the Leaf node with the right trusted identity for the asymmetric bootstrap to work. These cipher suites should also be chosen so that the cryptography components of the symmetric cryptography can be used in the asymmetric cryptography to reduce footprint of the security. To reduce the footprint further the following objectives can be considered. Combining a small and an efficient process with protocols designed for embedded devices and selecting the right ratio of size and speed while configuring the algorithms to be either more size efficient or speed efficient.

In order to retrieve certificates from the cloud service, the cloud service should be a RESTful web service written in JavaScript to maintain the preferred programming language of ELL-i. This Cloud Service will be based upon the node.js framework and potentially other well-known modules to ensure safety, simplicity and free to distribute within for example the GPL licence.

2 Theory

2.1 Constrained Application Protocol

“An application-layer protocol tailored for the use with constrained devices and constrained networks is the Constrained Application Protocol(CoAP)” [4]. CoAP is a web transfer protocol based on UDP similar to the Hypertext Transfer Protocol. CoAP provides like HTTP a RESTful interface. The only difference is that CoAP focuses on being more lightweight and cost-effective.

CoAP has four methods for a client to interact with a server: GET, PUT, POST and DELETE. They are respectively for retrieving, updating, creating and deleting a resource. In addition CoAP can offer binding methods. Three methods are available: polling, observe and push. The first method polling defines that the client should send periodically a GET request message for a specified CoAP resource. The next method, observe, defines that the CoAP client sends an initial GET request message with the Observe method and its attributes defined in the body of the message. Then the CoAP server should acknowledge this binding in order to accept the agreement. After that, the CoAP server can send periodically messages based upon the attributes. The last binding method is very similar to the observe binding method. With push, a resource can be preconfigured to be bound to a CoAP client’s address.

Another functionality of CoAP is the discovery mechanisms of resources. The built-in discovery of CoAP allows for retrieving existing resources of a CoAP server. Furthermore, Datagram Transport Layer Security(DTLS) is the favoured security protocol because CoAP runs on the unreliable UDP protocol and DTLS security can secure the UDP protocol by covering its unreliabilities and using cryptography to encrypt the data[2], [5]–[14].

2.2 Cryptography

Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication and data origin authentication. So there are four cryptographic goals: confidentiality, data integrity, authentication and non-repudiation.

The goal of confidentiality is to keep the content of information from all but those authorized to access it. There are two ways to provide confidentiality: physical and logical protection. Examples of logical protection are mathematical algorithms which render data unintelligible.

The data integrity goal is to have protection against alteration of data during transport. In order to assure data integrity, one must be able to detect an alteration.

Authentication of an message ensures the origin of the data and entity. Information delivered over a channel should be authenticated as to originating entity, date of origin, data content, time sent and etc. Because of the two sided goal, it is often separated into two classes: entity authentication and data origin authentication. Data origin authentication implicitly provides data integrity(if a message is modified, the source has changed).

And lastly the goal of non-repudiation prevents entities from denying previous commitments or actions. When a dispute arises due to an entity denying that certain actions were taken, a means to resolve the situation is necessary. Often a trusted third party is needed to resolve this dispute.

Figure 2 provides a schematic listing of the primitives that can be used and how they relate. Only the primitives useful for this thesis, e.g. symmetric-key and public-key primitives, will be explained in the following paragraphs.

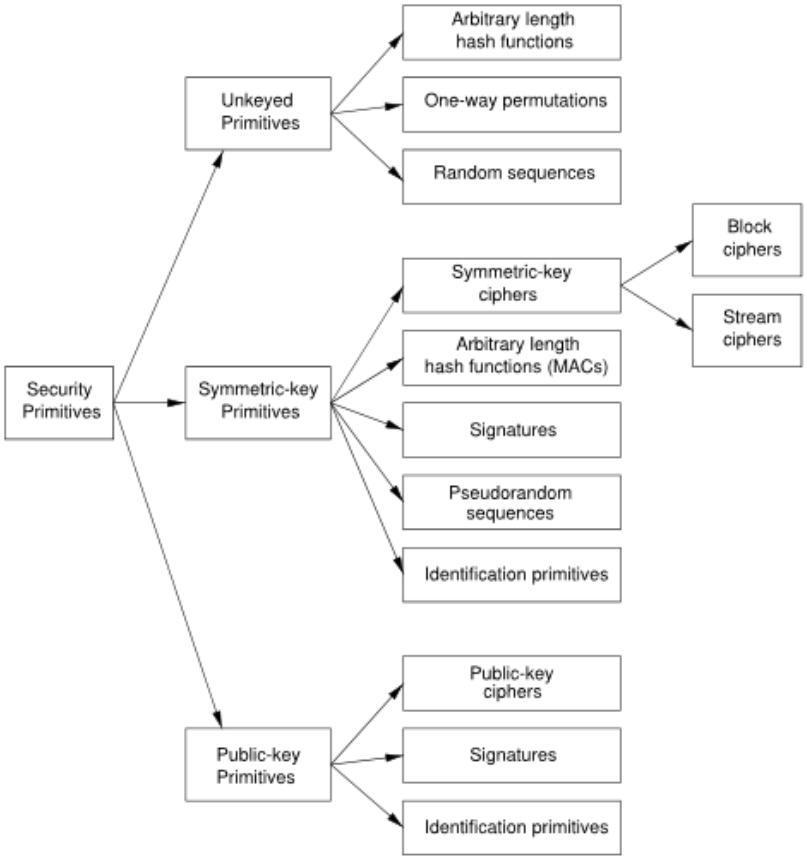

Figure 2: A taxonomy of cryptographic primitives[30]

2.2.1 Symmetric-key encryption

Symmetric-key cryptography or private-key cryptography uses a private key. All the peers of the communication know the private key in order to decrypt or encrypt data. Private-key encryption is used to provide confidentiality of messages. These symmetric-key encryption algorithms are usually grouped into stream ciphers and block ciphers. A stream cipher uses 1 byte at a time whereas a block cipher uses blocks of data. The RC4 algorithm is an example of a stream cipher. The Triple DES and the AES algorithm are block ciphers.

The process of encrypting a message is the following: the message will be first divided into blocks of data with the input length of the encryption function. Then the encryption will do a series of functions like XORing, permutations, bit-shifting and linear mixing. After these functions are done encrypting the block of data, the data will be known as ciphertext. Now there are the same amount of ciphertext blocks as plaintext blocks, the naïve way would be to just send them over to the other party. But this approach is dangerous because a stranger could use the lack of randomness to decipher the private key. In order to provide randomness to the messages a couple of basic encryption methods can be used: Cipher Block Chaining(CBC), Counter(CTR) , Output Feed Back(OFB) and Cipher Feed Back(CFB). Only the most relevant modes of encryption for this thesis will be discussed.

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This ensures that the ciphertext always changes even if the plaintext is repeated. To change the first block of data an initialization vector(IV) is used. Because all the cipher-blocks are dependent on all the previous cipher-blocks, the last block can be used as authentication and integrity protection. This feature is used in CBC-MAC mode. A Message Authentication Code provides authentication and integrity protection to a message.

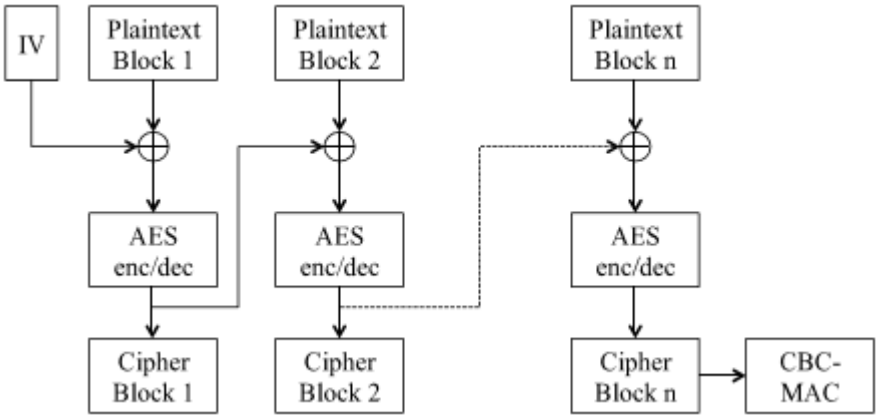

Figure 3: The structure of CBC encryption. The last cipher block servers as CBC-MAC[12]

The counter mode is an block cipher encryption algorithm that does not use the cross dependencies between cipher-blocks to provide randomness, instead it uses a nonce and a counter. The input block in Figure 4 can be a plaintext or cipher block and the output will be the corresponding cipher or plaintext block.

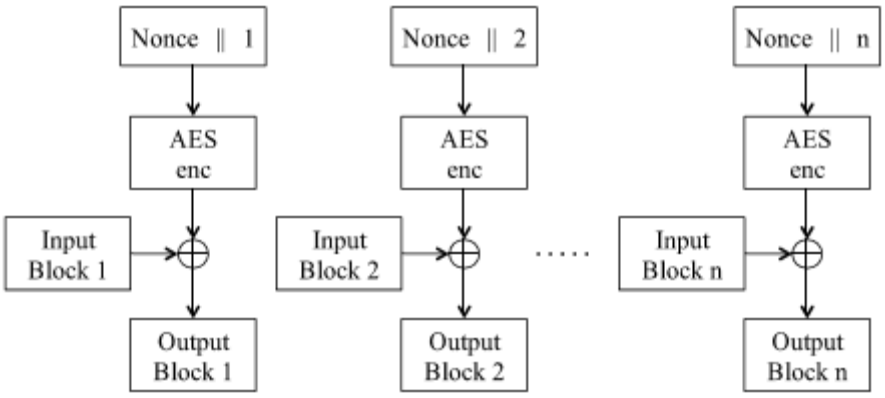

Figure 4: CTR encryption and decryption mode[12]

CCM mode is an adaptation of the CTR mode with CBC-MAC to provide authentication, integrity and confidentiality. The CCM mode uses the CTR mode to encrypt but also the CBC-MAC to create the MAC code. The CCM mode requires two block cipher encryption operations per each block of an encrypted-and-authenticated message and one encryption per each block of associated authenticated data.

2.2.2 Asymmetric-key encryption

The disadvantage of symmetric-key algorithms is the placement of the private-key on both ends. This can be solved with asymmetric-key algorithms because each device has its own set of keys. In this section, the concept of Public Key Cryptography and Public Key Infrastructure is explained with regards to securing IoT.

The concept of PKC is based on how hard a mathematical problem is to solve, as in the complexity of calculations. Examples of hard to solve problems are prime factorization and

discrete logarithm problems. The most used public-key algorithm is RSA but an alternative is ECC. The first step of PKC is to generate key pairs, the public key and the private key. The private key is only used by the identity who is defined by the keys and it should be impossible for anyone to derive the private key from the public key. When a message is encrypted by the public key, only the private key can decrypt that message and the other way around.

"Public-key cryptography is the de-facto standard for peer authentication across independent network domains in the Internet" [15]. Public-key cryptography is generally much harder to process than symmetric-key cryptography and they are therefore mostly used together. The process of public-key cryptography is often only used for establishing trust and an shared secret for the symmetric-key cryptography to use. The private key of PKC must be kept private at all times because the use of this key can indicate the identity of the peer. If someone would get hold of one another's key, he could impersonate the original holder of the key. For this purpose there is the Public Key Infrastructure.

RSA

The RSA algorithm is based upon the problem of finding the prime factors of a number. For RSA, a key length of 1024 bit is required in order to have the same level of security as symmetric key cryptography with a key length of 128 bit.

- Key exchange/agreement RSA

The key agreement process is as displayed in Figure 5. Peer 'A' sends the new key encrypted with the public key of peer 'B' so that only 'B' can read the new key. The issue with this procedure is a Man in the middle attack. Peer 'B' cannot verify with this process who had send the new key. A more secure key exchange is the Diffie-Hellman key exchange[16]–[19].

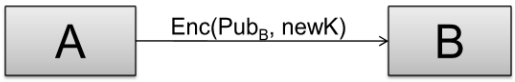

Figure 5: Encrypt new key using the public key of B[31]

Diffie-Hellman key exchange

Diffie-Hellman(DH) key agreement scheme is the most famous key agreement protocol. It is based upon the hard to solve computational problem from logarithms. The process of DH is shown in Figure 6. Both Alice and Bob generate an unique large number(a and b). Both peers will calculate the same equation with the unique number being the only difference. The result of these calculations are seen as almost impossible to factor back into their begin form and are therefore secure for transportation. And then lastly due to mathematical properties of the earlier equation, both parties can derive the same secret by using the retrieved number as base of an exponentiation.

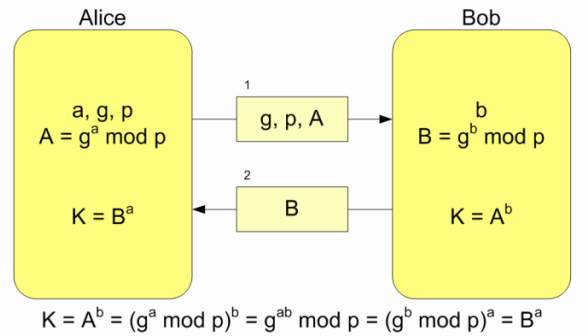

Figure 6: Diffie-Hellman key agreement scheme[31]

ECC

Elliptic Curve Cryptography is also a public-key cryptography, ECC is based on elliptic curves over finite fields. An 256 bit key length for ECC is equivalent to a 128 bit symmetric-key security. The standardization community IETF recommends AES-CCM with ECC for constrained devices[2].

- ECDH

The Elliptic curve Diffie-Hellman(ECDH) is a variant of the Diffie-Hellman protocol using the elliptic curve cryptography. It is used to securely exchange a secret key in an unprotected connection. First each party generates their own pair of private-public keys based upon an already agreed elliptic curve. If new keys are generated for the session the protocol is also defined as ephemeral ECDH(ECDHE) but static keys can be used. However this will not provide forward security. Next, the public keys are exchanged and the common point K on the elliptic curve is calculated with the following equation:

$$d_s Q_c = K = d_c Q_s$$

Equation 1: Calculation of common point K[4]

This checks out because the pre-defined point on the curve is known to both parties and the following equation is valid:

$$d_A\, Q_B \;=\; d_A\, d_B\, G \;=\; d_B\, d_A\, G \;=\; d_B\, Q_A$$

Equation 2: The deriven shared secret of both parties are equal. Where Q_x: public key of x and d_x: secret key of x[4]

After computing K, the x-coordinate value of K serves as a common secret between both parties. This secret is then used to derive the private key. An expansion function for the shared secret can be used to generate a shared-key with proper length.

- ECDSA

Elliptic Curve Digital Signature Algorithm(ECDSA) provides both authentication and integrity protection to a message. It is used in the key exchange of DTLS in order to deal with MITM attacks. The peer uses its private key to sign the message. This message can then be opened with only the corresponding public key and therefore provides identification of the sender.

In order to provide perfect forward secrecy(PFS), ephemeral keys can be used. This practice is mostly used in conjunction with ECDH or DH. At the moment of creating the shared secret, newly generated keys are used instead of the static certified keys. This results in a secure communication even if the static keys are compromised. The only disadvantage of ephemeral keys is the extra computation cost of generating them[14], [18]–[21].

2.2.3 Public Key Infrastructure

As described before, public-key cryptography uses sets of keys to identify, encrypt and decrypt data. So it is possible to use a set of keys to identify a peer but how do you know which set belongs to which peer? For this purpose certificates were designed. In today's internet X.509 is the most used certificate to link a set of keys to an identity but there are other types like PGP, SPKI, Raw public key and Implicit certificates.

X.509

"X.509 Version 1 has been available since 1988, is widely deployed and is the most generic" [22]. A X.509 certificate can be represented by Figure 7. The X.509 certificate has resources identifying the subject and the issuer but because the X.509 is generic, additional resource, that are specific to the situation, can be added. The standard resources are the following:

- Version
- Serial number
- Issuer name
- Signature algorithm identifier
- Validity period
- Subject name
- Subject public key information
- Issuer digital signature

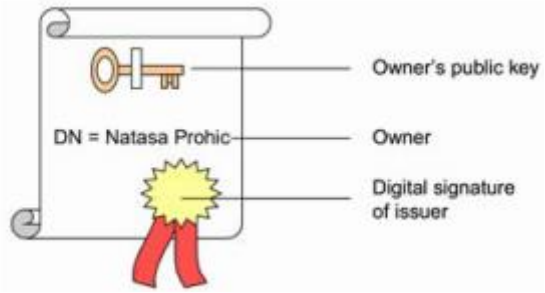

Figure 7: Representation of a X.509 certificate[22]

The public key infrastructure of a X.509 certificate consists of a Certification Authority and a Registration Authority, Figure 8. The CA generates the certificates and are well known e.g. VeriSign. The RA is an authority that can verify user requests for a digital certificate so that the CA can make the certificate for this user[22].

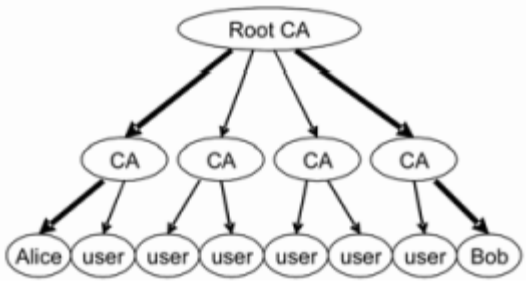

Figure 8: Network of trust of X.509 certificates[22]

PGP

The resources of a PGP certificate are very similar to the resources of a X.509 certificate. A PGP certificate includes the following resources: PGP version number, certificate holder's public key, certificate holder's information, digital signature of the certificate owner, validity period and the preferred symmetric encryption algorithm for the key.

Figuratively speaking is a PGP certificate a public key with one or more labels tied to it as shown in Figure 9. The labels tie the identifying information of the owner to its public key. Each label can contain different means of identifying the key's owner with for example private information. Besides the private information everybody may sign the key/identification pair to attest to their own assurance that the certificate is valid. This validating scheme is called web of trust as seen in Figure 9[22].

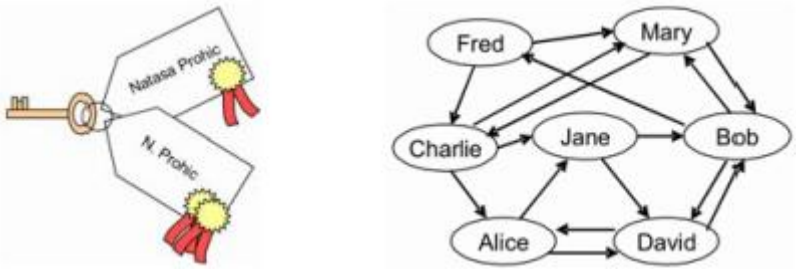

Figure 9: The PGP certificate on the left; Web of trust on the right[22]

SPKI

"A certificate of SPKI carries as few information on clients as possible compared to a certificate of PKIX(Public Key Infrastructure with X.509).[23]" According to Takamichi et al. there are three kinds of certificates: ID certificate, Attribute Certificate and an Authorization Certificate. The ID certificate guarantees the binding of an identity and a public key. The attribute Certificate guarantees the binding of an identity and an authority. And lastly the Authorization Certificate guarantees the binding of an authority and a public key. An authorization certificate is generated by a server and is held by a client who can use it to utilize the service of that server. This authorization certificate is based upon the SPKI certificate.

The contents of a SPKI certificate include: Issuer, Subject, Delegation, Authorization, Validity. The first field, issuer, specifies who has created and signed the certificate. The issuer is represented as a public key or hash of a public key. The next field, subject, defines to whom the certificate has been issued. The subject is also represented by a public key or hash of a public key. The Delegation field specifies whether the authority specified in the certificate is delegatable or not, represented by a Boolean value. The authorization field specifies the authority of the certificate. The last field, validity, defines until when the certificate is valid[23].

Raw public key

The raw public key method sends the public-key raw over the channel. This causes the problem that verifying the raw public key has to be installed out-of-band. There are three available methods: firstly retrieved from a DNSSEC secured resource record using DNS-Based authentication of Named Entities (DANE), secondly obtained from a certificate chain from a Lightweight Directory Access Protocol (LDAP) server and thirdly the public-keys should be provisioned into the operating system firmware image and updated via software updates[24], [25].

Implicit certificate

With a conventional certificate or explicit certificate like X.509, the public key and the digital signature are distinct data elements. An implicit certificate has in contrast to explicit certificates no public key element but uses a superimposed signature and public key. The certificate is built so that the public-key can be derived while verifying the certificate with the superimposed data element. The implicit certificate has also like an explicit certificates, identification data to identify the subject of the certificate[26], [27].

2.3 IP security

The most widely deployed protocol for securing IP-enabled devices is Transport Layer Security(TLS). TLS runs over the reliable Transmission Control Protocol (TCP) and provides protection against attacks such as eavesdropping, tampering and message forgery. In constrained environments however is UDP the most favoured transmission protocol because of its less complex structure and smaller overhead. This is also why CoAP runs over UDP and this has led to the development of Datagram Transport Layer Security(DTLS). Because UDP does not guarantee delivery, ordering or duplicate protection, DTLS has resources build in to cope with the disadvantages of having a connectionless communication protocol. In addition, most elements used by DTLS are based upon TLS thus providing the same security guarantees as TLS. This was done to minimize new security inventions and to maximize the amount of code and infrastructure reuse.

The DTLS protocol consists of two objectives: the handshake layer to establish a secure connection and the traffic encryption layer to protect the communication via the secure connection. The handshake is always the first to execute. It sets up a secure communication for the encryption of messages. Both parties who are involved in the handshake have to agree on the security details they both will use and in addition the handshake can provide mutual authentication if needed. In the next paragraph all the flights during the handshake will be discussed to point out the capabilities of this handshake.

Handshake

Three different types of handshakes are possible in DTLS, it depends on what kind of authentication is required: no authentication, server authentication and mutual authentication. In this thesis mutual authentication is required. The mutual authentication can be achieved by Pre-Shared Key(PSK), Raw Public Key(RPK) or certificates but it will be discussed later. As seen in Figure 10, the handshake consists of six flights and each flight is a group of multiple messages. This handshake is similar to the TLS handshake except from the first two flights. These are sent in order to prevent Denial of Service(DoS) attacks.

The handshake begins with a ClientHello message to start the handshake. This message also includes the security parameters supported by the client and a random value that is used later. When the server receives this message it will create a stateless cookie with parameters from the ClientHello. The cookie is used to prevent DoS attacks because the client has to send another ClientHello as a response of the server sending an HelloVerifyRequest to create a state on the server. With flight 3, the ClientHello, the handshake can really begin.

Flight 4 consists of up to five messages: ServerHello, Certificate, ServerKeyExchange, CertificateRequest and ServerHelloDone message. The ServerHello message contains the security parameters that will be used during the handshake and traffic encryption layer and the random value from the ClientHello. If both peers do not have an common known security suite, the handshake will be terminated. The security suite defines the algorithms used during and after the handshake layer. The next message of flight 4 is the optionally Certificate message. If the Client wants to verify the certificate of the server, to authenticate the server, the certificate is send. The ServerKeyExchange message is an cipher suite depended message which is only send with some suites that require additional parameters for deriving the premaster secret. For example, the cipher suite TLS_ECDHE_ECDSA_AES_128_CCM_8 uses ECC for the key establishment and authenticating message. It uses AES with CCM_8 mode for bulk encryption and authentication. This cipher suite needs the ServerKeyExchange to send the ephemeral ECDH public key of the server and the specification of the corresponding elliptic curve. The CertificateRequest message can be send in order to enable the authentication of the client, resulting in mutual authentication if both parties have verified each other. To finish flight 4, the ServerHelloDone message is send.

The messages of flight 5 are the Certificate, the ClientKeyExchange, the CertificateVerify, the ChangeCipherSpec and the Finished message. The first message, Certificate message, contains the certificate or another form of authentication data. This message is only send when the CertificateRequest from the server was send. Then the ClientKeyExchange is send. The ClientKeyExchange is the same as the ServerKeyExchange except for the sender and receiver. After the ClientKeyExchange comes the CertificateVerify message, it is used by the client to proof his possession of his private key. The client computes the hash of all exchanged flight messages so far and then computes the signature of the digest using the corresponding private key. Before flight 5 is finished with the Finished message an ChangeCipherSpec message is send to signal the server that the client will encrypt the subsequent messages with the negotiated algorithms.

To end the handshake, flight 6 is send by the server with its own ChangeCipherSpec and Finished message. With the handshake being completed, both peers can communicate using the negotiated cipher suite.

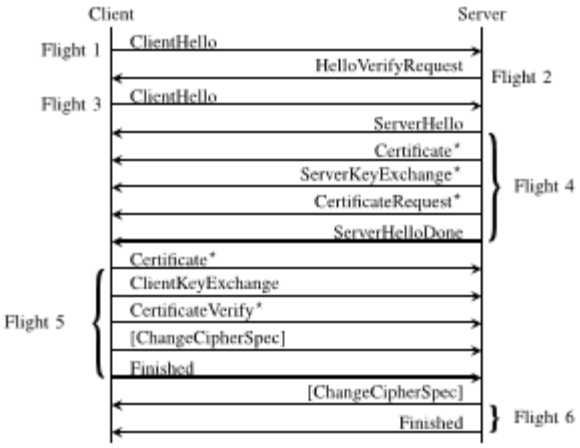

Figure 10: Flow of an DTLS handshake[15]. Messages with an '*' are optional

2.3.1 Cypher suites of DTLS

TLS and DTLS uses standard cipher suites to enable easy configuration of the communication. Each named cipher suite defines a key exchange algorithm, bulk encryption algorithm, message authentication code(MAC) and a pseudorandom function.

- The key exchange algorithm determines how the client and server will authenticate during the handshake. This can also be divided into an key exchange and an authentication algorithm.
- The bulk encryption algorithm defines which algorithm is used to encrypt the message stream. It also includes the key size and the lengths of explicit and implicit initialization vectors.
- The message authentication code algorithm creates the message digest, a cryptographic hash of each block of the message stream.
- The pseudorandom function(PRF) creates the master secret. The master secret is used as a source of entropy when crating session keys, such as the one used to create the MAC.

The application-layer protocol CoAP defines two mandatory cipher suites: TLS_PSK _AES_128_CCM_8 for PSK mode and TLS_ECDHE_ECDSA _AES_128_CCM_8 for RPK

mode. This thesis will compare the algorithms that can be used for constrained environments so to choose the best cipher suite.

Cryptographic hash

A cryptographic hash is often used to verify the data's integrity because these hashes are a one-way message-digest algorithm. The output hash value that comes out of the cryptographic hash cannot be used to retrieve the original input value. It is therefore only used to verify the integrity of the message. The most common used cryptographic hashes are MD5 and SHA but other methods are available as explained in the symmetric-key cryptography chapter.

3 IP security

In order to provide IP security for constrained devices using CoAP as communication protocol, DTLS is preferred. DTLS however has a lot of ways to provide the necessary security objectives. In this system three objectives are considered: lightweight, mutual authentication, interoperability and automatic integration.

3.1 PSK, RPK and certificates

The advantage of using public-key cryptography is the identity based establishment. If a peer has the right identity it can be allowed to communicate, and this makes the system more flexible in the sense of automatic configuration.

PSK has in comparison to public-key cryptography the advantage of performance. The shared secret is based upon the PSK and therefore does not need to use the more resource expensive public-key cryptography.

RPK is in a way the combination of certificates and PSK. The connection establishment is identity based but the public-key of the corresponding peer needs to be preconfigured by an out-of-the-band method.

3.2 Public-key algorithms

If public-key cryptography can be used in an constrained environment, which one would suite best? In order to compare the two public-key algorithms, two aspects can be compared. First of all the size of the key used in the algorithm and secondly the needed processing capability.

3.2.1 Key size

The security of the encryption is dependent on the length of the key. According to the National Institute of Standards and Technology(NIST), Table 1, is a 128-bit symmetric-key of similar security level as an 3072 bit RSA key. This RSA key is really large for a constrained device as the lowest end of devices have only 8kB of RAM. That would mean that only the key uses almost half of the volatile memory. Thankfully there is a possible substitute for RSA that uses much smaller keys, Elliptic curve cryptography. For an 128 bit of symmetric key security the elliptic curve key has to be 256 bit.

Table 1: NIST-recommended key sizes in bits

Symmetric-key (AES)	RSA key	Elliptic curve key
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	521

3.2.2 Efficiency

Table 2 provides the processing comparison between an RSA 2048 bit key against an ECC 224-bit key based upon the NIST curve. The benchmark runs on the OPAL node[28] which features an Atmel SAM3u micro-controller and the Atmel AT97SC3203S TPM. The benchmark measures one Private Key and two public-key operations. The RSA and ECC implementations are based upon the WolfSSL project.

It is notable how faster the ECC based handshake is than the RSA counterpart. In addition, the ECC 224-bit key provides better security over the 2048-bit key according to the NIST-recommended key sizes in Table 1.

Table 2: Public-key part of DTLS handshake using software ECC and RSA on OPAL[17]

Handshake	Processor clock(MHz)	Computation time(ms)
RSA – 2048-bit	48	5165
RSA	96	2583
ECC – 224-bit	48	1614
ECC	96	772

3.3 Certificates

In the environment of IoT, everything needs to be as constrained as possible. Therefore it is good to think about what certificates should be able to offer in the IoT environment. A certificate in an IoT environment needs to verify the trustworthiness of peers. A question that a peer could ask is: Is this peer allowed to communicate with me? There are four elements where the certificates could be different: the certificate, network of trust and revocation procedure. At the end of this chapter, a footprint comparison of three types of certificates is shown.

3.3.1 Differences in certificate

Every PGP certificate contains a self-signature and optionally multiple third party signatures. For a X.509 certificate supports only a single digital signature from the CA. Both the X.509 and PGP certificates can have a lot of 'personal' information to verify the peer. A SPKI certificate on the contrary has as little as possible 'personal' information and is only signed by the issuer. Then there is the Implicit certificate, it has also identification data elements but in contrast to explicit certificate the signature and the public key are not separate data elements. The public key can be derived from the signature.

3.3.2 Differences in network of trust

A peer can create his or her own PGP certificate. This certificate can then be signed by everyone who wants to act as an introducer of that key and therefore contributing to the web of trust. In the case of X.509 certificates, only the CA can issue a certificate and it is the job of the RAs to approve the certificate request. This is similar with SPKI certificates, a trusted identity can sign and manage the certificates. The network of trust of implicit certificates are also based upon the X.509 network of trust with an issuer wo can sign the certificate.

3.3.3 Differences in revocation procedure

A revoked signature is the same as a revoked certificate in a X.509 certificate infrastructure provided that the signer(CA) the only is that made it valid. With X.509 certificates, only the issuer(CA) can revoke the certificate. PGP certificate is in contrast to X.509 certificates revoked by the user(subject). In case of the PGP, the user posts the revoked certificate on a certificate server. With X.509 the revoked certificate is put in the CRL. The revocation procedure for SPKI is similar to X.509.

The revocation procedure of implicate certificates is non-existent. It is not done because implicit certificates can be generated more frequently so the CA can simply stop refreshing a user's certificate.

3.3.4 Size comparison

The X.509 and PGP certificates are almost similar in the data elements and will therefore not differ a lot. The SPKI and implicit certificate have less data elements and are therefore more likely to be smaller in size. But the most size expensive element of a certificate is the public key and the signature of the certificate. In Table 3 two certificates are compared, the X.509 certificate and an implicit certificate (ECQV). In addition the X.509 certificate will be based upon a RSA and ECC certificate. When comparing a couple of security levels it is notable that the RSA certificate is much larger than the ECC certificate. This is because the keys of RSA cryptography need to be longer than ECC cryptography in order to achieve the same security level. Because the public-key and signature are combined in one data element with implicit certificates, another size reduction can be achieved.

Table 3: Comparison of certificate sizes of implicit certificate(ECQV), ECC based certificate(ECDSA) and RSA based certificate[29]

Security Level	Certificate size (bits)			Ratio ECQV/RSA certificates
	ECQV	ECDSA	RSA	
80	193	577	2048	10x
112	225	673	4096	18x
128	257	769	6144	23x
192	385	1153	15360	39x
256	522	1564	30720	57x

3.4 Footprint

To make a good comparison, the overall footprint can be considered. Figure 11 compares the RAM overhead of three different systems. The first certificate based system uses the certificate based handshake. The next system uses a symmetric-key based handshake PSK to establish a connection. And the last system is the proposed system of René Hummen et al[15]. The delegation system uses a special designed architecture based upon DTLS to enable the nodes in communication with over-the-internet devices. This system defeats the interoperable objective of this thesis and therefore cannot be used. Another system, presented by Thomas Kothmayr et al[17], that implements X.509 certificates and DTLS system indicated an total usage of approximately 20kB of RAM and 67kB of ROM for the entire implementation including the networking and system code.

The RAM overhead of the certificate based system uses almost three times more RAM than the symmetric based system. This is a considerable amount and can be directly linked to the storage space of certificates and public-keying resources. The smallest of current constrained devices have a minimum of 8kB of RAM, in comparison to the total implementation usage of 20kB . These devices will not be able to run X.509 certificate based DTLS.

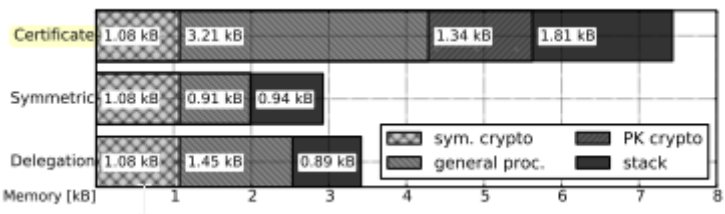

Figure 11: RAM overhead of the certificate based DTLS handshake, the symmetric-key based handshake and the delegation architecture[15]

The footprint on the storage is also considerable larger. Figure 12 compares the ROM usages of the same systems as the RAM overhead comparison. The ROM overhead of an certificate based DTLS handshake is yet again almost three times as high as the symmetric-key based DTLS handshake. To compare with a fully implemented system, the total system indicated an usage of 67kB of ROM.

Is the ROM overhead significantly compared to the total storage space of today's constrained devices? The lowest end devices have around 64kb ROM and 8 kb of RAM. Midrange devices can have up to 256 KB of ROM and from 4 to 32 KB of RAM. The high end devices have 128KB up to 1MB of ROM and up to 128 KB of RAM. In the case of really low end devices public-key cryptography is almost not possible to implement because there is also the code in order to use Ethernet communication and application specific coding. If the node is rather from the mid-range or the high range of constrained devices, public-key crypto based DTLS is definitely a considerable option.

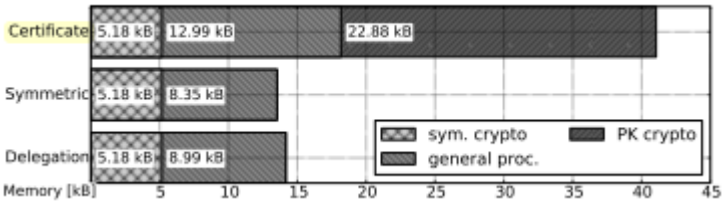

Figure 12: ROM overhead of the certificate based DTLS handshake, the symmetric-key based DTLS handshake and the delegation architecture[15]

3.5 Libraries

A lightweight security can only be achieved by using code optimized for size. And to not reinvent code, libraries can be used. When comparing libraries for a company that wants to sell their product a couple of objectives can be considered: the number of algorithms, small footprint and peer reviewers. The amount of algorithms contribute to the easiness of adapting the product. The small footprint is to make the security in this case as small as possible. And the last objective peer reviewers is to make sure that most bugs are an can be easily resolved.

The comparison of the libraries in Table 4 is done by comparing the objectives of the libraries by visiting their website. The peer reviewers is mostly based upon GitHub followers and Google search hits. The footprint is based upon their objectives.

Table 4: Comparison of libraries

Name	Start date	#Algorithms	Small footprint	Peer reviewers
ARM mbed TLS	2006	++	+(+)	+++
Cryptlib	2003	++	+	+++
NaCL	2009	++	++	++
Libsodium	2013	-	+	+/-
Libtomcrypt	2003	+	+	-
wolfSSL	2004	++	++	++
Relic-toolkit	2009	+	++	+/-
Tinydtls	2011	-	++	++

4 The system

As explained in the introduction the system of ELL-I consists of three objects: Leaf node, Site Controller and the Cloud Service. In this chapter all the parts will be explained and how they relate to each other.

4.1 Cloud Service

The purpose of the cloud service is to act as an trusted identity/certificate authority. It needs to provide the Site Controller with the needed information for the automatic integration of the Leaf node in the local network. The cloud service is written in the preferred programming language by ELL-i, JavaScript. And to make a program with JavaScript Node.js must be used. Node.js provides an asynchronous coding style with basic functionalities to write programs.

To provide the clients of ELL-I of an easy to use interface when they desire to use their own Site Controller, an interface must be made. To make such a interface a Web Service like interface can be coded, there is REST and SOAP. Because REST is surpassing SOAP in a lot of ways and is the most popular web API style, the choice is easy. So the Cloud Service's API will be a REST API.

If the Cloud Service needs to help the Site Controller connecting to a Leaf node, the Cloud Service needs to know which Leaf node is contacted. Therefore a database is used. The Leaf nodes of ELL-i will be stored into this database so that the Cloud Service can verify to which Leaf node the Site Controller is connecting. The database is the PostgreSQL database that can provide a lot of functionalities to ELL-i and it is free to use.

To act as an certificate authority, the cloud service needs a certificate generator. Traditionally to make a certificate you would need to have the public key of the subject, data to identify the subject and then the CA keys to sign the certificate. The most secure way is for the subject to create a certificate request(CSR). A certificate request is a message send from the subject to the issuer to apply for a certificate. The most common format for CSR is PKCS #10. It contains the public key, identification material and his own signature so that the CA can verify the request.

In Figure 13 the diagram of the web service is presented. The Site Controller will need to connect to the Cloud Service using an SSH connection so that the connection is secured. The Site Controller can use the Cloud Service by using the routes provided by the REST API. This interface makes sure that the Site Controller cannot directly modify or create any data from the database or certificate generator. The API can use the database manager to look up the Leaf node the site controller is trying to connect with. If the Site Controller is allowed to connect with the Leaf node, the API can generate a certificate for the Site Controller specific to the Leaf node if needed.

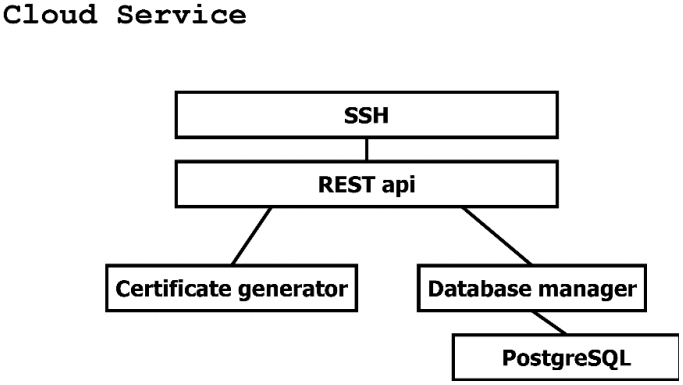

Figure 13: The Cloud Service program

4.2 Leaf node

The constrained Leaf node uses CoAP to communicate with DTLS security. The CoAP protocol provides the Leaf node with REST like interface for the Site Controller to retrieve and update resources. In the next paragraphs a couple of objectives are explained: service announcement, DTLS handshake and structure of the Leaf node.

For the Site Controller to find the Leaf node an indicative message must be send. With the use of CoAP and IPv6 this can be done. If the Leaf node is connected to the local network it will need to receive an IP address. With this address it can then multicast into the local network to let know that it exists. This message needs to indicate to the Site Controller which Leaf node he will be talking to. For this CoAP can be used, the Leaf node will act as a CoAP client and will send a request to 'PUT' a new Leaf node with his identification as payload into the resources of the Site Controller. After that, the Site Controller can initiate communication with the Leaf node.

The DTLS handshake will use public-key cryptography to authenticate and to establish an shared key. The identification certificate of the Leaf node and the trusted identity has to be preconfigured on the Leaf node, so it needs an alternative to the public-key cryptography for connecting and configuring the certificates on the Leaf node. Another advantage of enabling the Leaf node to have no certificates is that the Leaf node can reset and generate new keys so that the user of the Leaf node can configure it to use another trusted identity. With the reset functionality, the Leaf node can be renewed to make sure that the public keys are less likely to be exposed.

If the Leaf node has all the certificates, the site controller can connect to it with its DTLS client. The Site Controller needs to support ECC(ECDHE and ECDSA) and AES with CCM_8 mode. This is the best cipher suite for constrained devices. The Leaf node can verify the Site Controller's certificate with the pre-installed certificate of the Cloud Service and a secure connection can be established.

If the Leaf node has no certificates, a person of machine needs to be able to connect with the Leaf node in order to configure the certificates. This can be done by partly using the already available algorithms used by the public-key connection and a bit of additional code to support the PSK mode of DTLS. The first step for the person configuring the Leaf node is to make sure the network is secure. Then a pre-shared key that was configured during manufacturing and for example printed onto the Leaf node with QR-code can then be used to establish a secure connection with the Leaf node. After that, the user of the Leaf node can configure the certificates by using the CoAP interface of the Leaf node.

Figure 14 provides the structure of the program on the Leaf node and the process of connecting to the Leaf node. The CoAP server of the Leaf node will run on top of the DTLS implementation while the CoAP client will only run on top of the UDP implementation because the service announcement will not be secured with DTLS. The process of the Leaf node connecting to the Site Controller is as explained before. The Leaf node will first send an service announcement. Then the Site Controller will initiate the DTLS handshake with the Leaf node to establish a secure communication channel between the Site Controller and the Leaf node. Finally when the secure connection is established, the Site Controller can start using the Leaf node's resources.

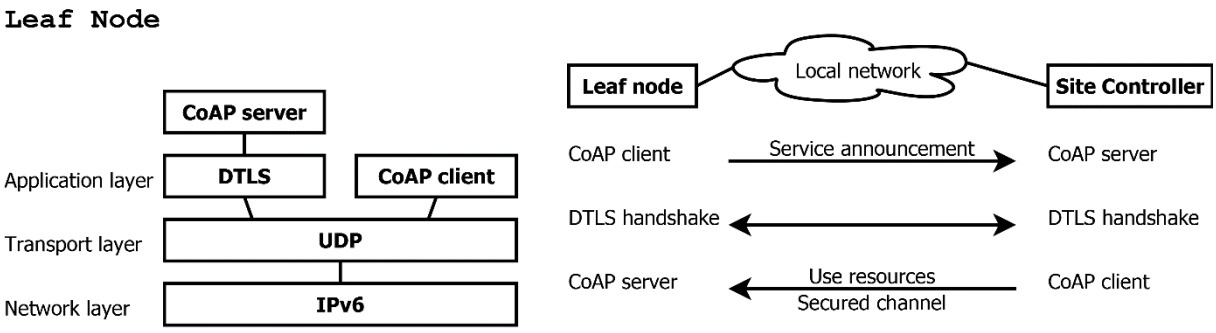

Figure 14: The program of the Leaf node

4.3 Site Controller

The Site Controller is the hearth of the system, it uses the Leaf nodes and the Cloud Service. It will use the Leaf node to read resource, e.g. temperature of a room, and the Site Controller can configure the Leaf node to for example close the blinds. If an installer needs to install this system, this person only needs to configure the Site Controller's link to the Cloud Service if the Leaf nodes are pre-configured with the right certificates. The only remaining installation work is to insert the Leaf nodes into the local network.

The configuration of the communication between Site Controller and Cloud Service is done at the moment by setting up a SSH connection to the Cloud Service and configuring the right key to start the communication with the Cloud Service.

For the Site Controller to contact the API of the Cloud Service, SOCKS5 protocol is used. The SOCKS5 protocol enables the usage of an distant network. With this protocol the SSH tunnel can be used to connect to a specific port of the Cloud Service so that the API of the Cloud Service can be used.

Node.js does not yet support DLTS. DTLS needs therefore to be implemented for the communication with the Leaf node. It is done by using Node.js's ability to interface other languages. With node-ffi, C and C++ code can be interfaced in Node.js. To make sure the functions of the library does not provide additional problems, a more simplified interface is built on top of the library, see Figure 15. With the functions: initDTLS, connectToServer, writeDTLS and readDTLS a simple and asynchronous interface was build.

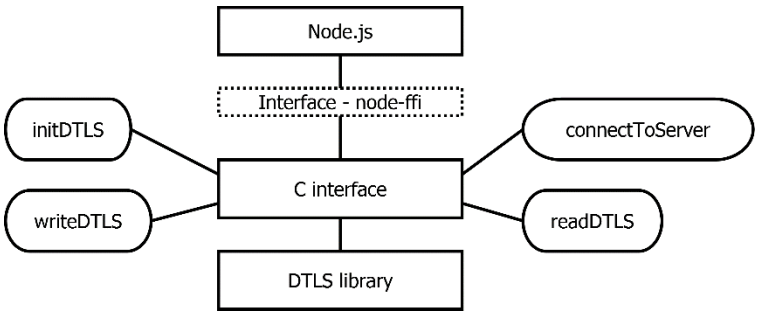

Figure 15: Interface of DTLS in Node.js

To connect with the Leaf node, the Site Controller needs to have a CoAP client on top of a DTLS implementation and a CoAP server without DTLS that is listening for service announcements. This is also shown in Figure 17. When the Leaf node contacts the Site Controller about his service, it can retrieve the necessary information to initiate the DTLS connection with the Leaf node.

Figure 16 provides the full process of a Leaf node connecting to the Site Controller. The first step of the system is for the Leaf node sending a Service announcement to the Site Controller.

Now that the Site Controller knows of the Leaf node, it can potentially request a certificate signed by the Cloud Service. When the Site Controllers possesses all the information needed to contact the Leaf node, it will initiate the DTLS communication. If the handshake is finally finished, it can start using the Leaf node.

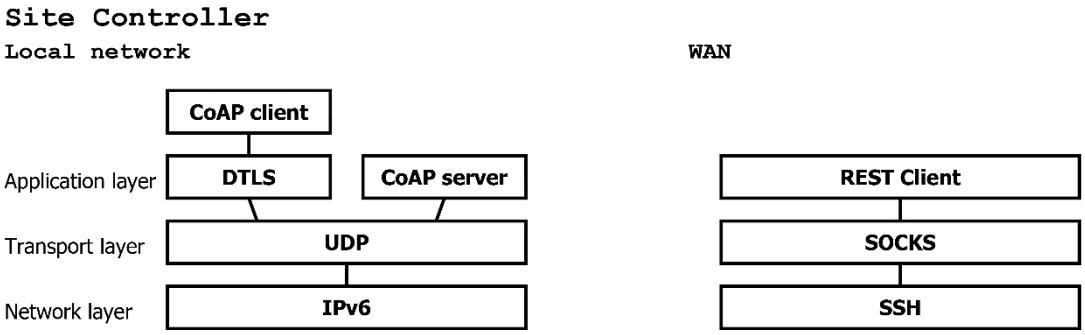

Figure 17: The structure of the Site Controller's program

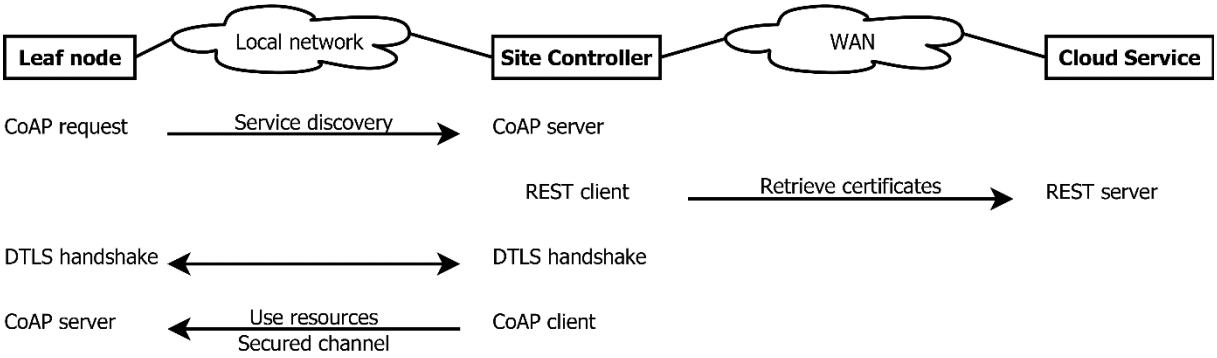

Figure 16: Process of the Leaf node connecting to the Site Controller

5 Conclusion

An objectives for the system in this thesis is to make the Leaf nodes automatically configurable and to make them usable in every system. This can be achieved by implementing the certificate support with mutual authentication. If the peer communicating with the Leaf node has a certificate signed by the trusted identity, trusted by the Leaf node, then the Leaf node will continue with the communication. If the Site Controller locates a Leaf node in the local network, it can verify the Leaf node to be trustable by verifying the certificate of the Leaf node. The best certificate to use in an IoT environment is an implicit certificate. However, implicit certificates are not yet wide spread and X.509 certificates are therefore better for interoperability.

To make the footprint as small as possible, a good library for DTLS can be chosen. The best library for these environments that has a lot of algorithms, a small footprint and lots of peer reviewers is WolfSSL. The best cipher suites for the communication with the Leaf node is ECDHE_ECDSA_AES_CCM_8 if there are certificates available. ECC is the lightest and smallest of public-key cryptography and therefore suitable for an IoT environment and AES_CCM_8 for its size in comparison to SHA based cipher suites. PSK_AES_CCM_8 would be the best for establishing a connection without certificates, most of the algorithms and processes used in PSK mode with AES_CCM_8 are already available in the public-key mode handshake.

At the moment of this writing the system is being implemented by using JavaScript for the Leaf node. The code of RIOT-os for ipv6 and Ethernet capabilities is being rewritten and therefore hard to use. In the future a lot of improvements could be done, for example implementing implicit certificates in WolfSSL and by optimizing the code in the Leaf Node.

Bibliography

[1] "ELL-i open source co-operative." [Online]. Available: http://ell-i.org/.

[2] C. Bormann, K. Hartke, and Z. Shelby, "CoAP: RFC 7252 Constrained Application Protocol." [Online]. Available: https://tools.ietf.org/html/rfc7252.

[3] "Riot: The friendly Operating System for the Internet of Things." [Online]. Available: http://www.riot-os.org/.

[4] H. Shafagh, "Leveraging Public-key-based Authentication for the Internet of Things."

[5] O. Bergmann, S. Gerdes, S. Sch, F. Junge, and C. Bormann, "Secure Bootstrapping of Nodes in a CoAP Network."

[6] S. Raza, H. Shafagh, K. Hewage, R. Hummen, and T. Voigt, "Lithe: Lightweight secure CoAP for the internet of things," *IEEE Sens. J.*, vol. 13, no. 10, pp. 3711–3720, 2013.

[7] A. Bhattacharyya, S. Bandyopadhyay, A. Ukil, T. Bose, and A. Pal, "Lightweight mutual authentication for CoAP (WIP) draft-bhattacharyya-core-coap-lite-auth-00," no. September, pp. 1–11, 2014.

[8] A. Ukil, S. Bandyopadhyay, A. Bhattacharyya, A. Pal, and T. Bose, "Lightweight security scheme for IoT applications using CoAP," 2014.

[9] A. Eriksson, A. Keranen, and J. Arkko, "Building Power-Efficient CoAP Devices for Cellular Networks," pp. 1–16, 2015.

[10] U. B. Tzi and G. Selander, "ACE use cases; use cases for the application of authentication and authorization in constrained environments(CoAP).," pp. 1–24, 2015.

[11] J. Park and N. Kang, "Lightweight Secure Communication for CoAP-enabled Internet of Things using Delegated DTLS Handshake," pp. 28–33, 2014.

[12] S. Jucker, "Securing the Constrained Application Protocol," no. October, pp. 1–103, 2012.

[13] M. Vucinic, B. Tourancheau, F. Rousseau, A. Duda, L. Damon, and R. Guizzetti, "OSCAR: Object Security Architecture for the Internet of Things," Apr. 2014.

[14] J. Granjal, E. Monteiro, and J. S. Silva, "End-to-end transport-layer security for Internet-integrated sensing applications with mutual and delegated ECC public-key authentication," pp. 1–9, 2013.

[15] R. Hummen, H. Shafagh, and S. Raza, "Delegation-based Authentication and Authorization for the IP-based Internet of Things," *Ieee …*, 2014.

[16] T. Wollinger, J. Guajardo, and C. Paar, "Cryptography in Embedded Systems : An Overview," 2003.

[17] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based security and two-way authentication for the Internet of Things," *Ad Hoc Networks*, vol. 11, no. 8, pp. 2710–2723, 2013.

[18] G. S. Tanwar, G. Singh, and V. Gaur, “SECURED ENCRYPTION - Concept and Challenge,” *Int. J. Comput. Appl.*, vol. 2, no. 3, pp. 89–94, 2010.

[19] Y. Choi, D. Lee, J. Kim, J. Jung, J. Nam, and D. Won, “Security enhanced user authentication protocol for wireless sensor networks using elliptic curves cryptography.,” *Sensors (Basel).*, vol. 14, pp. 10081–106, 2014.

[20] H. Y. Chien and C. S. Laih, “ECC-based lightweight authentication protocol with untraceability for low-cost RFID,” *J. Parallel Distrib. Comput.*, vol. 69, no. 10, pp. 848–853, 2009.

[21] R. Afreen and S. C. Mehrotra, “A REVIEW ON ELLIPTIC CURVE CRYPTOGRAPHY FOR EMBEDDED SYSTEMS,” *Ijcsit*, vol. 3, no. 3, pp. 84–103, 2011.

[22] N. Prohic, “Public Key Infrastructures–PGP vs. X. 509,” *INFOTECH Semin. Adv. Commun. …*, pp. 1–10, 2005.

[23] T. Saito, K. Umesawa, and H. G. Okuno, “Privacy enhanced access control by SPKI,” *Proc. Seventh Int. Conf. Parallel Distrib. Syst. Work.*, pp. 301–306, 2000.

[24] S. L. Keoh, S. Kumar, and H. Tschofenig, “Securing the Internet of Things: A Standardization Perspective,” *IEEE Internet Things J.*, vol. 1, no. 3, pp. 1–1, 2014.

[25] P. Wouters, H. Tschofenig, J. Gilmore, S. Weiler, and T. Kivinen, “Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS),” pp. 1–18, 2014.

[26] M. Campagna, “SEC 4 : Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV),” vol. 4, p. 32, 2013.

[27] “Certicom implicit certificates.” [Online]. Available: https://www.certicom.com/index.php/explaining-implicit-certificate.

[28] R. Jurdak, K. Klues, B. Kusy, C. Richter, K. Langendoen, and M. Brunig, “Opal: A multiradio platform for high throughput wireless sensor networks,” *IEEE Embed. Syst. Lett.*, vol. 3, no. 4, pp. 121–124, 2011.

[29] Certicom, “An Introduction to the Uses of ECC-based Certificates,” *Certicom*, vol. 2, no. 2, 2005.

[30] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, vol. 106. 1997.

[31] U. of Maryland, “Key Agreement Protocols,” 2008.

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Public-key cryptography with mutual authentication for IoT platform

Richting: **master in de industriële wetenschappen: elektronica-ICT**
Jaar: **2015**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Winderickx, Jori

Datum: **1/06/2015**