IoT Security: Problems, Challenges and solutions

Gabriele Saturni

SAPIENZA
UNIVERSITÀ DI ROMA

W • SENSE
INTEGRATED CABLELESS SOLUTIONS

Overview

- Basic cryptography concepts
- IoT security challenges
- Common attacks for IoT Networks
- Security libraries: freeRTOS & RELIC

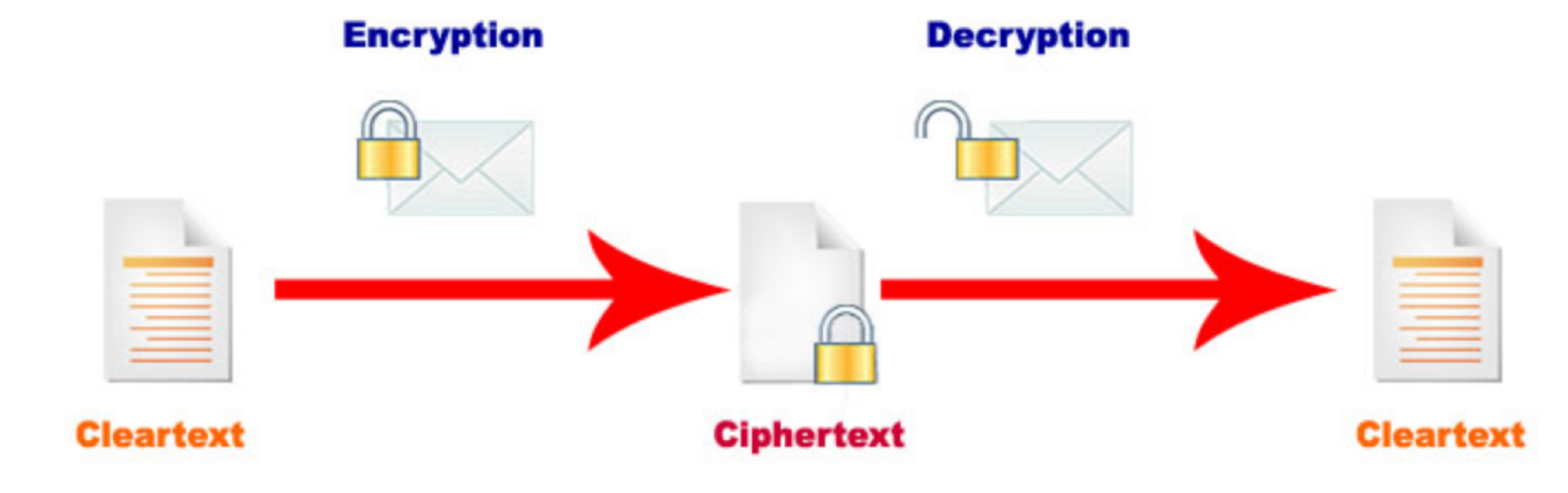
Basic security concepts

- *What is cryptography ?*
  - The cryptography is a science that uses the mathematics to encrypt and decrypt data
  - The goal is to protect the information, transmitted over an insecure channel, against attackers
  - We can not protect the data forever and ever **BUT** we can protect the data for enough time, making them useless → the information will be available to the attacker in 100000 years

Basic security concepts

- Some useful definitions:

  - *Plaintext:* The data can be read by anyone
  - *Encryption:* The method that is used to hide the plaintext to unauthorized users
  - *Chipertext:* The output of the encryption
  - *Decryption:* The method that makes the data readable from authorized users
  - *Key:* Secret information that is used to encrypt and decrypt the data. Only the authorized users can share the key.


Encryption
Decryption
Cleartext
Ciphertext
Cleartext

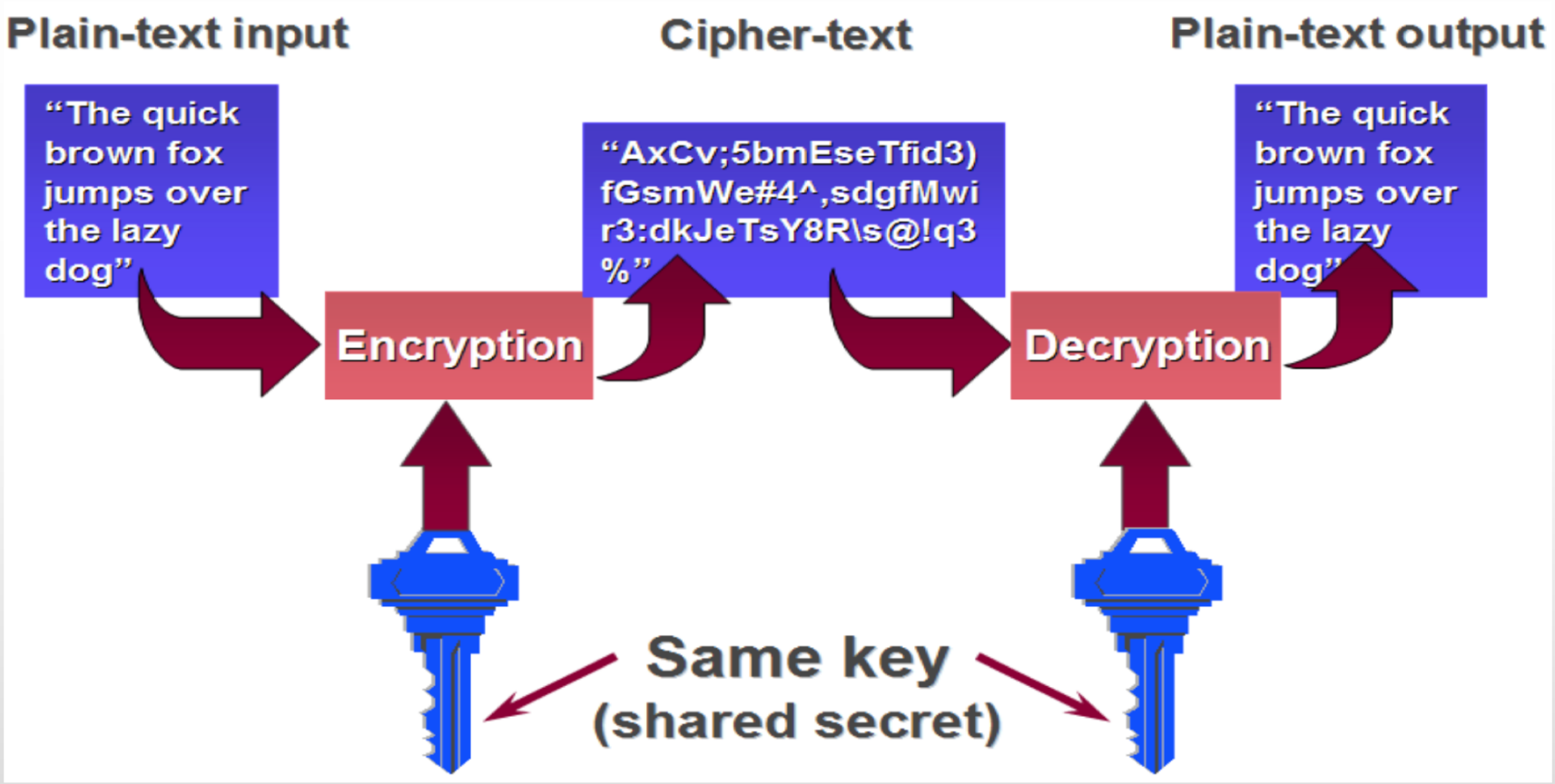
Basic security concepts

- As we can see all the security relies on the keys that are used during the encryption and decryption phases

- ***HOW TWO USERS CAN AGREE ON THIS SECRET OVER AN INSECURE CHANNEL ?***

- Two common strategies:
  - ***Symmetric cryptography*** VS ***Asymmetric cryptography***

Symmetric cryptography



Symmetric cryptography

- The **asymmetric cryptography** began thousands of years ago (Caeser chipher, Vigenere chiper, etc.)
- Today, the most common algorithms used in symmetric cryptography are based on **block ciphers**
- These algorithms take a number of bits and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size. Blocks of 64 bits are commonly used.
- The most used are:
  - **AES** (Advanced Encryption Standard)
  - **GCM** (Galois Counter Mode)

Symmetric cryptography– how it works

- Essentially, it is based on a XOR operation

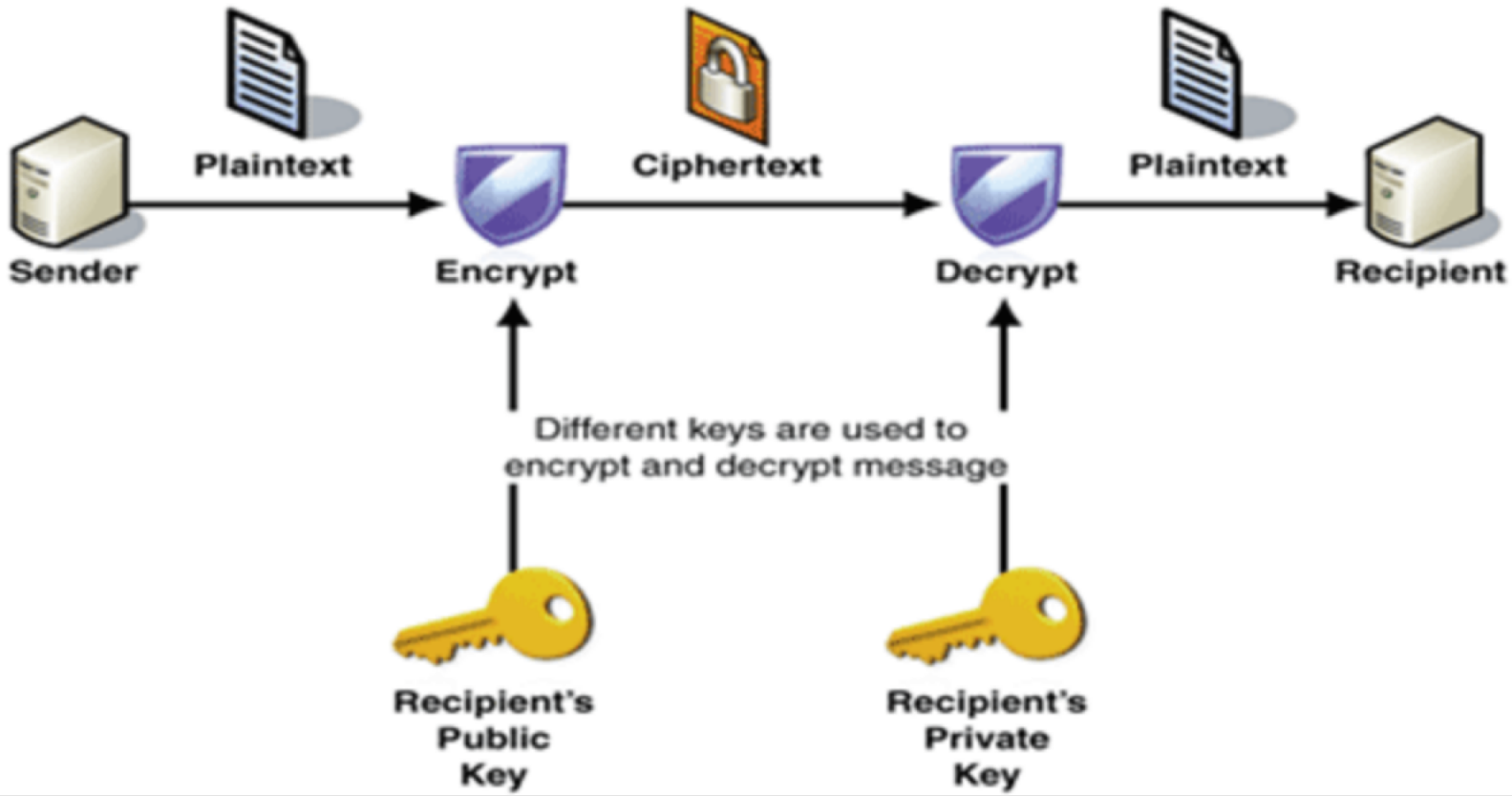
Message: 0 1 0 1 0 1 1 0 1 1 1 1 0
Key : 1 1 0 0 0 1 0 1 0 1 0 0 1
Output : 1 0 0 1 0 0 1 1 1 0 1 1 1

<u>The only way available for the attacker is to try all the possible combinations of the key !</u>

Symmetric cryptography

- **PROS** of symmetric cryptography:
  - Encryption and decryption are really fast
  - The encrypted data can be transferred on the link even if there is a possibility that the data will be intercepted. Since there is no key transmitted with the data, the chances of data being decrypted are null
  - Only the users that have shared the key can read the plaintext
- **CONS** of symmetric cryptography
  - Symmetric cryptosystems have a problem of key transportation. The secret key is to be transmitted to the receiving system before the actual message is to be transmitted. Every means of electronic communication is insecure as it is impossible to guarantee that no one will be able to tap communication channels. So the only secure way of exchanging keys would be exchanging them personally
  - There is not authentication. If the attacker stole the key of one user it can communicate without problems

Asymmetric cryptography



Asymmetric cryptography

- RSA is an **asymmetric cryptography** protocol

- RSA security relies on the computational difficulty of factoring large integers that are the product of two large prime numbers. Multiplying two large primes is easy, but it is hard to determine the original.

- RSA keys are typically 1024 bits long

RSA – How it works

- Key generation:
  - Select two random prime numbers p and q
  - Compute n=pq
  - Compute Φ = (p -1)(q –1)
  - Select e, such that 1 < e < Φ and gcd(e,Φ) = 1
  - Compute d =e^{-1} mod Φ
  - The public key is (n,e) and the private key is d
- Encryption
  - c = m^e mod n
- Decryption
  - m = c^d mod n

Asymmetric cryptography

- Another important family of **asymmetric cryptography** protocols are based on Elliptic Curve Cryptography (ECC)
- ECC security relies on elliptic curve discrete logarithm
  - significantly more difficult problem than factoring
  - ECC key sizes can be significantly smaller than those required by RSA yet deliver equivalent security with lower computing power and battery resource usage making it more suitable for mobile applications than RSA.
- The most common protocol is ECDH (Elliptic Curve Diffie Hellman)

ECDH – How it works

- Suppose that we have two users, Alice and Bob that need to share a common secret key and G is the generator of the elliptic curve
- Alice selects a random element a and computes A = aG. Then sends Ka to Bob
- Bob does the same, so takes randomly b and computes B = bG. Then sends Kb to Alice
- Then both Alice and Bob compute, respectively, Ka = (B)a and Kb = (A)b
- Notice that Ka = abG = Kb = baG

Asymmetric cryptography

- **PROS** of asymmetric cryptography:
  - In asymmetric cryptography there is no need of exchanging keys, thus eliminating the key distribution problem.
  - The primary advantage of public-key cryptography is increased security: the private keys do not ever need to be transmitted or revealed to anyone.
  - Provide mechanisms for repudiation of unauthorized users
- **CONS** of asymmetric cryptography
  - The encryption is slower than symmetric cryptography

Summarizing…

- Cryptography plays a crucial role for achieving secure communication
- There are two kinds of cryptography:
  - Symmetric: faster than asymmetric but there is the problem of key distribution
  - Asymmetric: slower than symmetric but it is safer
- ECC cryptography is very suitable for IoT due to constrained devices

IoT Security challenges

Security goals

Wireless sensor networks are vulnerable to many attacks, due to:

- broadcast nature of transmission medium;
- resource limitation on sensor nodes;
- uncontrolled environments where they are left unattended

Requirements for IoT Security

- **Availability:** the desired network services are available even in the presence of denial-of-service attacks
- **Authorization:** only authorized sensors can be involved in providing information to network services
- **Authentication:** the communication from one node to another node is genuine; a malicious node **can not** masquerade as a trusted network node

Requirements for IoT Security

- **Confidentiality:** a given message cannot be understood by anyone other than the desired recipients
- **Integrity:** a message sent from one node to another is not modified by malicious intermediate nodes
- **No repudiation:** a node cannot deny sending a message it has previously sent
- **Freshness**, which implies that the data is recent and ensures that no adversary can replay old messages

Requirements for IoT Security

- **Forward secrecy:** if a node is compromised by an attacker then there is no way (from the attacker's point of view) to recover the previous communication
- **Backward secrecy:** a joining sensor should not be able to read any previously transmitted message
- **Efficiency:** storage, processing and communication limitations on sensor nodes must be considered
- **Scalability:** Security solutions should support a large number of node and take into account that the topologies could rapidly change

IoT Security challenges

“standard” security solutions does not fit well for IoT

- Constrained devices with low computational power
- The longevity of the device (sometimes is impossible to update with new security patches)
- The fact that there is a device → Usually no User Interface for entering user ids and passwords
- Embedded systems are often developed by grabbing existing chips, designs, etc.
- The data could be highly personal
- The mindset → manufacturers do not thing like security expert. Often the security is not considered as an important aspect…

Foundation of IoT Security

- Build security in, **IT CAN NOT BE ADDED LATER**
- Use ECC that guarantees high security level and good performance
- Encrypt only the sensitive data
- Obscurity does not provide security
- Check the literature and use cryptographic primitives suitable for IoT (e.g key management protocol based on implicit certificate)

Common attacks for IoT Networks

Attacks Type

There are three mainly types of attacks against IoT networks:

- **Attacks on network availability**: DoS attacks
- **Attacks on secrecy and authentications**: the attackers try to exploit vulnerabilities on cryptographic primitives adopted
- **Stealthy attack against service integrity***:* in a stealthy attack, the goal of the attacker is to inject into the network false data values

Attacks type

Where is the level at which an attack can be performed ?

Attacks type

ANYWHERE!

Physical layer attacks

The physical layer is responsible for:

- frequency selection;
- carrier frequency generation;
- signal detection;
- Modulation

Possible attacks:

- Jamming
- Tampering

IoT NETWORKS ARE VERY SUSCEPTIBLE TO PHYSICAL LAYER ATTACKS!

Jamming

- The attacker interferes with the radio frequencies that the nodes use for communication
  - Essentially the attacker injects noise!
- A jamming source may be powerful enough to disrupt the entire network
- **It is very easy to implement**
- **The consequence on the networks can be catastrophic**

Tampering

- The attacker can **MANUALLY** steal the device and extract the cryptographic key used
- The attacker can tamper the circuits of the nodes
- Modify the source code and replace it with some malicious code
- Replace original devices with a malicious one

Link layer attacks

The link layer is responsible for:

- multiplexing of data-streams
- data frame detection
- medium access control, and error control

Attacks at this layer include:

- purposefully created collisions
- resource exhaustion
- unfairness in allocation

Link layer attacks

The attacker can produce collisions:

- Transmitting at the same time and frequency of networks node

This implies:

- Retransmissions of the packet → increased energy consumption of legitimate nodes → reduced network availability

Network layer attacks

The IoT networks are vulnerable to several network layer attacks:

- Spoofed routing information
- Selective packet forwarding
- Sinkhole attacks
- Sybil attacks
- Wormhole
- Hello flood attacks
- Acknowledgment spoofing

Network layer attacks

- **Spoofed routing information**: the most direct attack against a routing protocol is to target the routing information in the network. An attacker may spoof, alter, or replay routing information to disrupt traffic in the network
- **Selective packet forwarding:** in a multi-hop network all the nodes need to forward messages accurately. An attacker may compromise a node in such a way that it selectively forwards some messages and drops others

Sinkhole attack

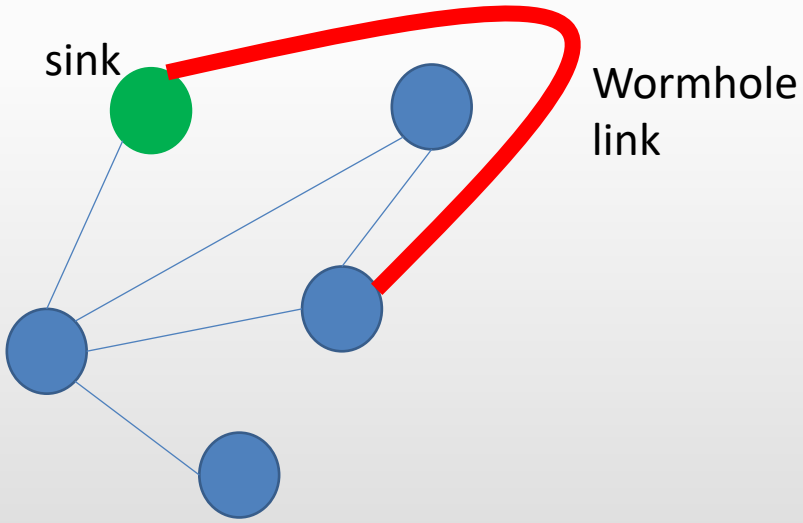
- It is a way to make very easy a selective forwarding attack
- The attacker builds a node that has a very convenient route
  - The neighbor nodes choose the compromised node as the next-hop node to route their data through
- **ALL THE TRAFFIC COULD BE PASS THROUGH THE COMPROMISED NODE**

Sybil attack

- The compromised node presents more than one identity in a network
- The attacker creates a very high number of fake identities
- The attacker uses those identities in order to gain a disproportionately large influence
- **This affects the routing protocols and voting applications**

Wormhole attack

A wormhole is low latency link between two portions of a network over which an attacker replays network messages



This attack would compromise seriously the network topopology

Hello flood attack

- The vast majority of the protocols use *Hello* packets for understanding which are the nodes that are placed within the same radio range
- An attacker may use a high-powered transmitter to fool a large number of nodes and make them believe that they are within its neighborhood

Acknowledgment spoofing

- The attacker can spoof and modify the acknowledgment packet
- An attacker, spoofing the acknowledgments, can thereby providing false information to the nodes

Transport layer attack

Flooding

- Whenever a protocol needs to maintain the information about the connections, it becomes vulnerable to memory exhaustion through flooding
- An attacker may repeatedly make new connection request until the resources required by each connection are exhausted or reach a maximum limit

Transport layer attack

De-synchronization

- The attacker disrupts the existing connections
- An attacker repeatedly spoofs messages to an end host causing the retransmission of missed frames
- This affects the ability of the end hosts to successfully exchange data
  - They waste energy attempting to recover from errors which never really exist

Security in freeRTOS

Amazon freeRTOS

Amazon freeRTOS is an operating system for microcontrollers that makes small, low-power edge devices easy to program, deploy, secure, connect, and manage.

We will focus on how to use it in order to make our IoT application secure

Documentation link: https://docs.aws.amazon.com/freertos/latest/lib-ref/index.html

Security in freeRTOS

- Amazon FreeRTOS has two libraries that work together to provide platform security: TLS and PKCS#11
- Amazon FreeRTOS provides a software security solution built on mbed TLS (a third-party TLS library)
- The TLS API uses mbed TLS to encrypt and authenticate network traffic
- PKCS#11 provides an interface to handle cryptographic material

TLS

The public interfaces of this library and a detailed explanation for each TLS interface are listed in lib/include/aws _tls.h (https://docs.aws.amazon.com/freertos/latest/lib-ref/aws__tls_8h.html)

PKCS#11

- PKCS#11 standard for cryptographic operations and key storage. To port PKCS#11, you must implement functions to read and write credentials to and from non-volatile memory (NVM)

- The functions you need to implement are listed in lib/third_party/pkcs11/pkcs11f.h

PKCS#11

The following functions are the minimum required to support TLS client authentication in Amazon FreeRTOS:

- C_OpenSession, C_CloseSession
- C_GetFunctionList, C_Initialize, C_GetSlotList
- C_FindObjectsInit, C_FindObjects, C_FindObjectsFinal
- C_GetAttributeValue
- C_FindObjectsInit
- C_FindObjects, C_FindObjectsFinal
- C_GetAttributeValue
- C_SignInit, C_Sign
- C_Finalize

RELIC toolkit

RELIC toolkit

- RELIC is a modern cryptographic meta-toolkit with emphasis on efficiency and flexibility. RELIC can be used to build efficient and usable cryptographic toolkits tailored for specific security levels and algorithmic choices
- This library provides cryptographic primitives that are tailored for IoT application
- Ease of portability and inclusion of architecture-dependent code
- Flexible configuration

link:https://github.com/relic-toolkit/relic