Security needs in embedded systems

Anoop MS
Tata Elxsi Ltd. India
anoopms@tataelxsi.co.in

Abstract: The paper discusses the hardware and software security requirements in an embedded device that are involved in the transfer of secure digital data. The paper gives an overview on the security processes like encryption/decryption, key agreement, digital signatures and digital certificates that are used to achieve data protection during data transfer. The paper also discusses the security requirements in the device to prevent possible physical attacks to expose the secure data such as secret keys from the device. The paper also briefs on the security enforced in a device by the use of proprietary security technology and also discusses the security measures taken during the production of the device.

1. Introduction

The embedded or handheld devices are getting increasingly connected and are more and more involved in network communications. The users of these devices are now able to execute almost all the network/internet applications that run in a PC on these devices. These devices are also increasingly involved in transfer of secure data through public networks that needs protection from unauthorized access and thus the security requirements in embedded devices have become critical.

The secure data falls in different categories requiring different levels of security. According to whose interest the protection of the data is, the secure data can be classified as two: the users private data and the user restricted data. The users private data are those data which when its security is compromised impacts directly on the user. A simple example of compromising such security is having access to a user's internet banking password. But in case of user restricted data, it's not the user but the content (data) provider who suffers direct loss on compromising the security of that data. The examples of such data are digital multimedia content such as copyrighted digital photos, audio and video contents.

The secure data not only requires protection during data transfer but also while handling the data at the end user devices. Vulnerability at the end user device, like easy access to the secret keys that are used to encrypt or decrypt the data, can easily turn down the entire security measures. The protocol involved for the secure transmission of either of the above mentioned contents through a public network uses more or less the same techniques but the handling of the user restricted data at the user's end involves much more care as the content is protected from the user itself!

Thus an embedded device must implement methods or protocol for secure data transfer and also should implement security methods to defeat attempts of unauthorized access of secure data from the device. The security needs for an embedded device thus can be classified into two:

- Security needs for data transfer and
- Security needs within the device

2. Security needs for data transfer

The data in a public network passes through a number of untrusted intermediate points. Therefore the secure data must be scrambled in such a way that the data will be useless or unintelligible for anyone who is having unauthorized access to the secure data. This can be achieved with the help of cryptographic methods such as Encryption/Decryption, Key Agreement, Digital Signatures and Digital Certificates. The use of these cryptographic methods in an embedded system to achieve data security is explained in the following sections.

2.1. Data Encryption

Encryption is the process of scrambling/encrypting any amount of data using a (secret) key so that only the recipient, who is having access to the key, will be able to descramble/decrypt the data. The algorithm used for the encryption can be any publicly available algorithm like DES [2], 3DES [8] or AES [7] or any algorithm proprietary to the device manufacturer. The key is known only between the communicating devices and will typically of length 100s of bits. If publicly available algorithms are used, the security of the transferred data totally depends on the secrecy of the keys used for the encryption. Sharing and maintaining the secret key between the communicating devices without any unauthorized entity getting access to the keys is important for foolproof secure data communication. These keys can be embedded in the device prior to the communication, exchanged offline in a secure manner or established online using any key agreement algorithm as explained in section 2.2.

The storage of the secret keys within the device is also critical for ensuring the complete protection of data. Security requirement for the storage of secret keys in the device is discussed in section 3.

2.2. Public-key Key Agreement Algorithm

When there are 100's of devices in a network, sharing and maintaining secret keys between all the devices for data encryption seems difficult, even unrealistic. This is where the Key Agreement Algorithm is used. Using Key Agreement algorithm, a shared secret can be established between communicating parties without the need for exchanging any secret keys or secret parameters online or offline. This works as follows.

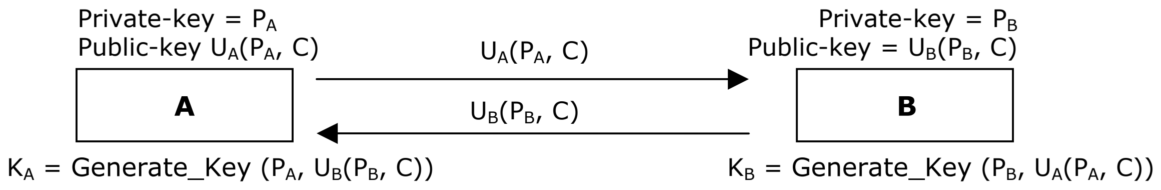
Key agreement algorithm is a public-key cryptography algorithm. For devices that use Key agreement algorithm will have a private-key and an associated public-key. The private-key is generally a random number of hundreds or few thousands of bits and the public-keys are derived from the private-key using the one-way function specified by the key agreement algorithm. One-way functions are mathematical function in which the forward operation can be done easily but the reverse operation is too difficult that it is practically impossible. The public-key is derived using private-key on the forward operation of the one-way function. The reverse operation of obtaining the private-key from the public-key is too difficult that it is practically impossible.

The devices that need to establish shared secret between them exchanges their public-keys and other public constants, if any. Both the device on receiving the other devices public-key performs key generation operation using its private-key to obtain the shared secret. The key agreement works such a way that the shared secret calculated by both the devices will be the same.

For e.g. let P be the private-key of a device and U(P, C) to be the public-key of the device, the representation U(P, C) is to show that the public-key of a device is derived from the private-key P of that device and some shared constants C known by all the device taking part in the communication.

Consider two devices A and B. Let P_A and $U_A(P_A, C)$ are the private-key and public-key of device A and P_B and $U_B(P_B, C)$ are the private-key and public-key of device B respectively. Both device exchanges their public-keys.

Device A, having got the public-key of B, calculates key K_A = Generate_Key(P_A, $U_B(P_B, C)$)
Device B, having got the public-key of A, calculates key K_B = Generate_Key(P_B, $U_A(P_A, C)$)


Private-key = P_A
Public-key $U_A(P_A, C)$
$U_A(P_A, C)$
Private-key = P_B
Public-key = $U_B(P_B, C)$
A
$U_B(P_B, C)$
B
K_A = Generate_Key (P_A, $U_B(P_B, C)$)
K_B = Generate_Key (P_B, $U_A(P_A, C)$)

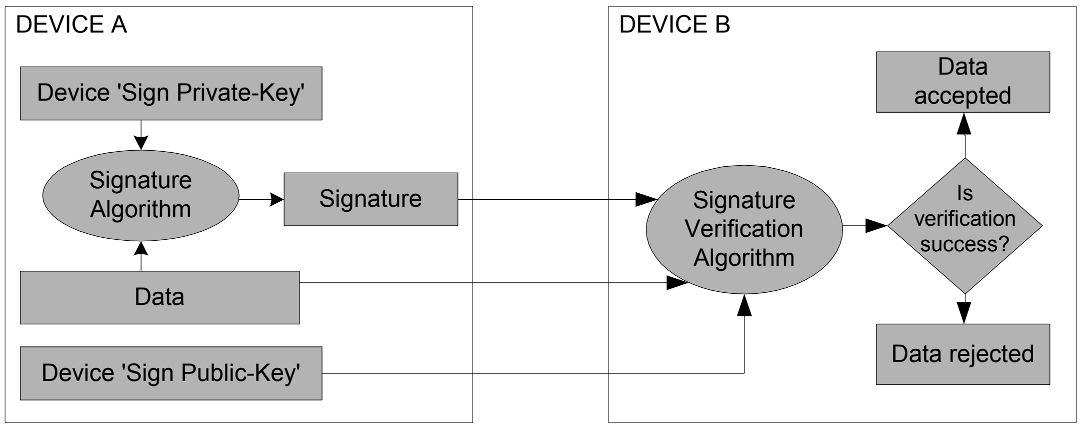
The key generation algorithm 'Generate_Key' will be such that the generated keys at the device A and B will be the same, that is shared secret $K_A=K_B=K(P_A, P_B, C)$.

It is impossible for any middleman, who only have access to the public-keys, $U_A(P_A, C)$ and $U_B(P_B, C)$, to obtain the shared secret K unless he has got the access to the private-key, P_A or P_B of any of the communicating device. Examples of key generation algorithms are DH [4] and ECDH [5].

In the key agreement algorithm the private-key used is a secret key and should not be shared with any other devices in the network, even to the device to which it is communicating. It is only the agreed shared secret key that is known between the communicating devices. It must also be ensured that the private-key is not disclosed from the device. The safe storage of the private-key of key agreement algorithm on the device is thus important. Security requirements for safe storage of secret keys are discussed in section 3.

2.3. Digital Signature

The device in a network may be communicating with the unknown or less familiar device located 100s of kilometers apart. The communication may also require routing through many intermediate points. During Key Agreement process, for establishing a secret key, any middlemen can substitute a devices public-key to its public-key and thus results in establishing a shared secret with the device. Therefore, for establishing shared secret using the key agreement algorithm, it is important for device to receive an authenticated public-key from the peer. For authenticated exchange of public-key, Digital Signature and Digital Certificates are used.



Digital signature is a public-key method to verify the authenticity of a received data from the peer. In digital signature, like the key agreement algorithm, a device uses a pair of keys, 'sign private-key' and 'sign public-key'. Only the device knows its sign private-key whereas the sign public-key is distributed to all the communicating devices. A device signs the message using a signatures algorithm with its sign private-key to generate a signature and any device that has got the access to the sign public-key of the signed device can verify the data with the signature using the signature verification algorithm. If any third party modifies the data or signature, the verification fails. Since only the signed device knows its sign private-key, it will be impossible for any other device to forge the signature. Examples of Digital Signature algorithms are RSA [3], DSA [2] or ECDSA [5].

2.4. Digital Certificate

Even while using the digital signature algorithm, the 'sign public-key' from a peer device has to be obtained by an authenticated way to ensure the authenticity of a received message. For key agreement or digital signature the authenticated transfer of public-key in a large network is difficult or even not possible without a centralized trusted authority. This centralized authority is trusted by all the devices in the network. This authority is generally known as trusted Certificate Authority or CA. The Certificate Authority (CA) signs the public-keys of devices along with the device ID using the CA's private-key to generate the signature. These CA signed data of a device (public-key, IDs etc.) along with the signature arranged in a standard format is called as a certificate. The certificate is issued by CA to all devices taking part in the communication. Any device, having the CA's public-key installed, can verify the authenticity of the received certificate and thus the public-key of the peer device. One popular certificate format is X.509 [6].

For obtaining the certificate, a device requests the certificate to the CA. The device sends its public-key, unique IDs and other information as required by the CA. The CA usually does some background check to ensure the device is not hostile before issuing the certificate. The certificate obtained by a device is rarely changed and the process of obtaining a digital certificate for an embedded device is usually done offline.

Once communicating devices obtains its certificate from the CA, the devices exchange their respective certificate before establishing a shared secret between them. Any device on receiving the peer certificate verifies it using the CA's public-key. Since the CA public-key common to all the devices taking part in the communication and is never changed, it is pre installed on all the devices in network generally through any offline trusted methods. Once a peer authenticates the device certificate, the device can use the public-key in the certificate to sign any message send by the device to the peer to prove the messages authenticity.

The CA may be different for different networks and for different communication protocol.

2.5. Certificate Hierarchy

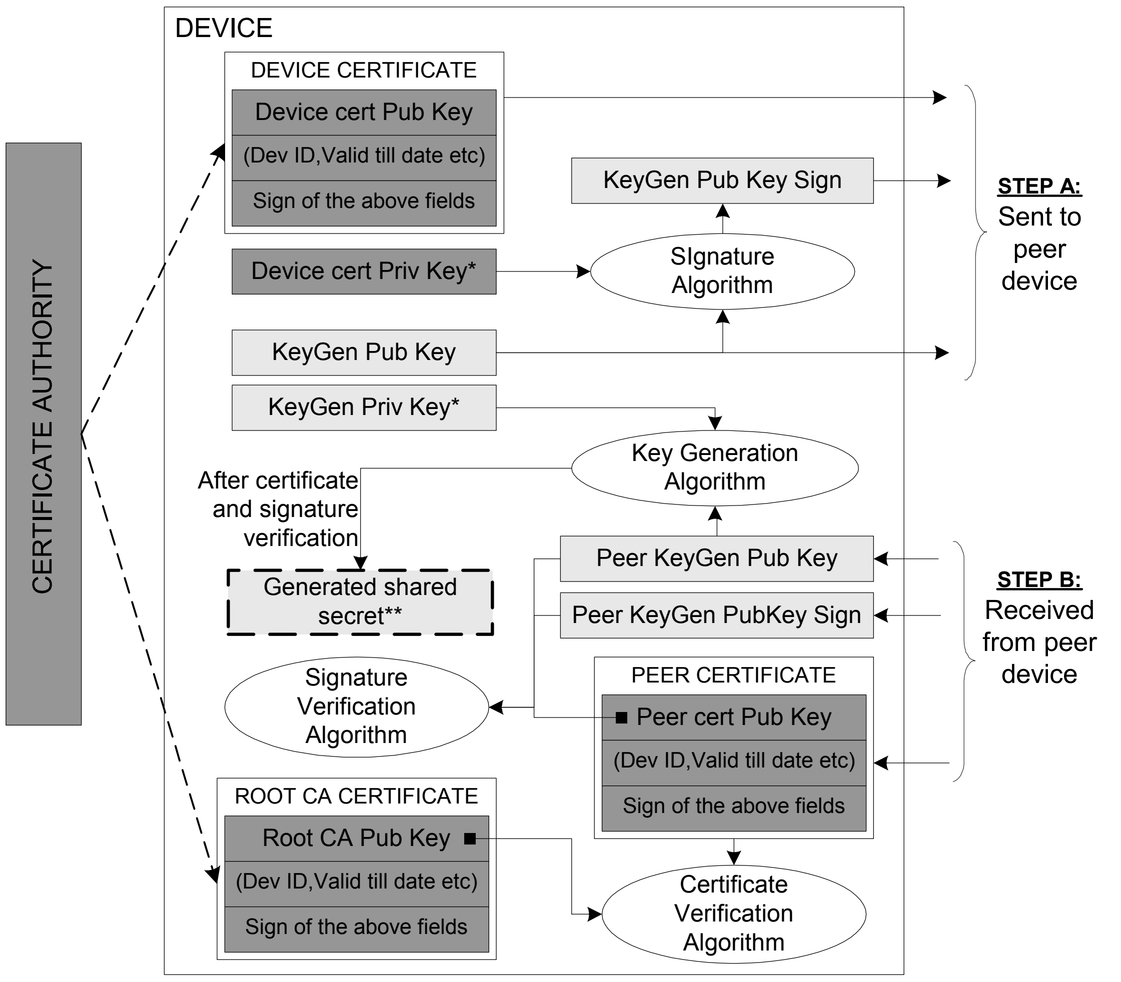
As the number of devices taking part in the communication increases and the location of these devices is distributed over different parts of the world, a single certificate authority may not suffice to issue and maintain certificates for all the devices. Certificate Hierarchy is the solution here.

In Certificate Hierarchy there will be a Trusted Root CA who will give permission other CA to issue certificate to the communicating devices. The Root CA will issue the certificate for the respective intermediate certificate authority. The intermediate CA will issue certificates to the device. In addition to issuing certificates to the devices, the intermediate CA's will also give its certificate, issued by root CA, to the devices. There can be multiple level of certificate hierarchy in which the intermediate CA's will give permission to other CA to issue certificate to the communicating devices.

If a device A obtained a certificate from an intermediate CA, then it not only receive its device certificate but also the certificate of the intermediate CA, which is issued by the root CA. If there is multiple level of intermediate CA above the CA that issued certificate to the device, the device will receive certificate of all intermediate CA's up to the root CA.
During a secure communication a device may ask all the intermediate certificates from its peer for successful verification of the received device certificate.

2.6. E.g. of Key Agreement algorithm

An example of key agreement protocol using digital certificates and digital signatures are explained below. A single certificate hierarchy is assumed in this example.



* Secrets known only by the device
** Secret known only between the communicating devices

Before connecting the device in the network, the device will acquire the device certificate from the CA. The certificate will have a 'sign public-key' and the device will have a 'sign private-key' corresponding to the sign public-key in the certificate stored securely in the device to prevent it from disclosure. The sign public-key and sign private-key can be of any digital signature algorithms such as RSA or DSA. The sign private-key can be used to sign any data sent by the device to prove the authenticity of its data.

For key agreement, the device uses any key agreement algorithms like DH or ECDH. The device generates a public-key and a private-key pair for key agreement algorithm. Let it be 'KeyGen public-key' and 'KeyGen private-key'. The device signs the KeyGen public-key with the sign private-key that corresponds to the sign public-key in the certificate. Once signed, the device sends the KeyGen public-key, its signature and its device certificate to the peer device. Similarly the device receives these sets of data from the peer device. On receiving the KeyGen public-key, signature and device certificate from the peer device, the device verifies the signature of the KeyGen public-key with the sign public-key in the peer device's certificate. The device then verifies the peer device's certificate using the root CA certificate stored in the device. Once the verification processes are successfully completed, the device proceeds with the key generation algorithm to obtain a shared secret using the device's KeyGen private-key and peer device's KeyGen public-key. The same process also takes place at the peer device and the shares secret generated by both the devices will be the same.

The certificate private-key and the generated shared secret are stored inside a device persistent memory and needs protection from disclosure. The KeyGen private-key is also not disclosed but is valid only till the shared secret generation and is generally disposed after the shared secret process is over. The KeyGen private-key is thus never stored.

3. Security needs within the device

Whether it is the private-key of any public-key algorithm as discussed in section 2 or it is any previously negotiated shared secret between the devices, the security of data transferred depends in the secrecy of these keys. To enforce additional security, some cryptographic algorithms may also specify a set of constant values that should not be disclosed from the device. These secret keys and secret values stored in the device that requires protection from unauthorized exposure are referred as 'secret keys' in this document.

The secret keys are stored inside the device, some even for the lifetime of the device. Hardware and software security measures implemented in the device must defeat any attempts of unauthorized access to retrieve these secret keys. Also, there are data such as the Root CA Certificate in the device that can be disclosed but should be prevented form unauthorized modification. If Root CA certificate can be modified, then the attacker can make the device to accept any certificate by substituting a fake root CA certificate and thus defeating the purpose certificate and secured communication. It is therefore also important that the security in the device is such that the data such as Root CA Certificates in the device is not subjected to unauthorized modification.

The level of security within the device varies depending on the nature of the protected content. The need for device security is more in the case of device handling user restricted data like copy-protected* video than in the case of user's private data like personal files or bank transactions. This is mainly because, in the case of user's private data since the user will suffer the direct loss on compromising such data, he/she will be responsible for restricting the physical access to the secret keys and other secured contents stored in the device. Also, the general implementation of secure data transfer protocols recommends a unique secret key for each device. Therefore if the hardware security of any of the device is compromised, it doesn't affect the security of other device in the network. But in the case of user restricted data, compromising the secret key of a single device results in the compromise of the security of all the copy-protected content handled by that device. One vulnerable device can thus results in helping an unauthorized device to access the copy protected content, decrypt it and distribute countless copy of the copy protected content.

The following section gives an example of prototype SoC to discuss the hardware and software support required to enforce the security within the device and thereby defeating the physical attack that compromises the security of the device.

3.1. Secure SoC

The Secure SoC provides physical protection to secret keys by keeping the components like Secure ROM, which is handling the secret keys, inside the Secure SoC.

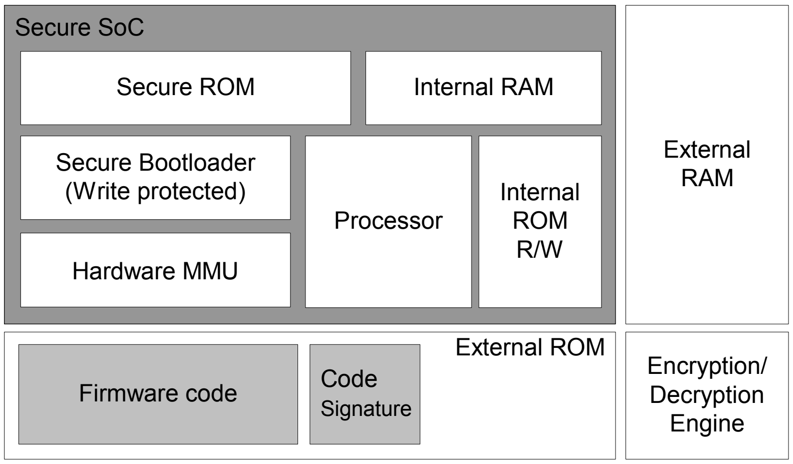
During execution time, the protected secure keys from the Secure ROM has to be loaded to the RAM in clear text and during that time the bus from the Secure ROM to the RAM can be monitored to access the secret keys. This can be prevented by allocating buffers for secret keys or intermediate values of cryptographic operations involving secret keys in

* Copy protected contents are multimedia content that have limited or no copying permission for the user.

the Internal RAM of the Secure SoC. This prevents the protected keys being available to any bus outside the Secure SoC.

The Secure Bootloader in the Secure SoC ensures that the device boots up with the genuine OS or firmware with right process privileges. The Memory Management Unit (MMU) configured by the OS permits the access to the buffers in the Internal RAM that involves secret key operations only to the secure processes with special OS privileges.

In the case where the Secure ROM is limited or pre-programmed by the hardware manufacturer, the Secure ROM can be programmed with a master key. This master key can be used to encrypt and store the device secret keys in the internal ROM.



In ideal case of a Secure SoC

- The Secure ROM cannot be physically accessed to retrieve the secret keys.
- The buses inside the Secure SoC cannot be monitored to obtain protected data or keys.
- The removal or replacement of any components in the Secure SoC should be impossible or should prevent the SoC from working.

The level of physical protection varies depending on the value of the protected content. The protection can be just tamper detection of SoC to zeroing of all the stored content in the SoC when a physical access attempt is made. Tamper detection protection method does not prevent a attacker from obtaining the data from the chip but will only makes it possible to know whether the chip is tampered or not. The zeroing requires special power supply and hardware support that makes the chip costlier. The NIST issued FIPS 140 [9] publication specifies different level of hardware and software security requirements for device that is involved in store and transfer of sensitive information.

The role of each component in the Secure SoC to ensure the secure storage of secret keys and other protected data are explained below.

3.2. Secure ROM

One method for storing the device secret keys securely in the persistent storage of a device is to encrypt the secret keys before storing. Thus even if anyone managed to get the data out of the persistent storage he/she will never be able to understand the secret keys. To encrypt any data generally two things are required, an encryption algorithm and a key for encryption. If any well-known algorithm like AES is used for encryption of the secret keys, then the strength of the encryption is only as strong as the secrecy of the key that used for the encryption. Thus the same problem faced for the storage of the secret keys is faced again for the storage of the key that is used for encrypting the secret keys. This problem is repeated unless an encryption algorithm is used that is known only to the device manufacturer. If the device proprietary algorithm is used for the encryption and storage of the secret keys, the security of the secret keys are only as strong as the secrecy of the algorithm. Since the code binary is stored in the clear text in the device memory and plenty of tools for reengineering the code like 'objdump' are available, the chance of exposing the secret keys cannot be neglected.

Another method to store the secret keys is to store it inside a Secure ROM. The Secure ROM resides inside the Secure SoC in the device. The hardware controller of the Secure ROM descrambles the data before retrieving it back from the ROM. This hardware support will prevent the unauthorized physical access to retrieve the secret key stored in the Secure ROM.

The buffers that hold the secret keys or the intermediate values of cryptographic operations involving the secret keys are allocated in the Internal RAM of Secure SoC. Thus the secret keys are prevented from being available to any bus outside the Secure SoC.

In the case where the Secure ROM is limited or pre-programmed by the hardware manufacturer, the Secure ROM can be programmed with a device master key. The device master key is a key unique to each device hardware or Secure SoC that can be further used to encrypt and store the device secret keys in a less Secure ROM.

The possible vulnerability in the implementation of Secure ROM can be:

1. The Secure ROM is physically removed from the Secure SoC, place it in another device and make it work to retrieve the protected keys.
2. The bus between Secure ROM and RAM is accessed to retrieve the protected keys.
3. An unprivileged/unauthorized application running on the device gets access to the API for retrieving the secret keys from the Secure ROM.

In an ideal case of Secure SoC, the first two vulnerability doesn't exist. The third vulnerability can be prevented by use of Secure Bootloader and implementation of right process privileges as explained in section 3.3 and 3.4 and thus not allowing an unprivileged application to run and access the restricted memory locations of the device.

3.3. Internal RAM and Secure Processes

The buffers for secret keys or intermediate values of cryptographic operations involving secret keys are allocated in the Internal RAM of the Secure SoC to prevent the secret keys being available to any bus outside the Secure SoC. Let this memory area in the Internal RAM be called as Secure Memory Area. Not every process should access this memory area. Only the processes with special OS privilege, Secure Process, should be able to access the Secure Memory Area. This is analogous to process with administrative privilege or root privilege in an operating system.

The OS during boot up configures the memory management unit to permit access to Secure Memory Area by only the Secure Processes. It is also important that the MMU configuration code in the OS is not modified by an unauthorized user to get access to the secure memory area. This can be ensured by the use of Secure Bootloader and code signing as discussed in section 3.4. The Secure Processes are configured to start during device bootup. The OS should disallow any unauthorized processes to run as Secure Process or to start a new Secure Process. This can prevent any downloaded application, if supported by the device, to access the Secure Memory Area to read the secret keys.

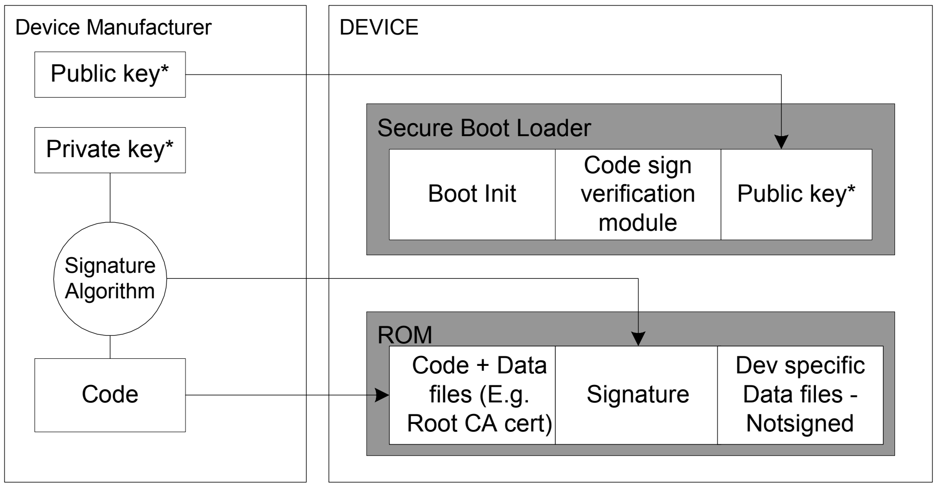
The results of an operation performed by a Secure Process on the secret keys are usually public data such as encrypted data or a public-key. These output data are requires by other less privilege processes to perform operation such as transmitting the output data to other device. There can be several ways of calling a Secure Process by a less privileged process. An example is explained below.

The Secure Processes waits, for e.g. on a semaphore, to get the input data and start executing. The input data buffer will be a non-secure memory area. Any less privilege can fill the input buffer and signal a Secure Process, by releasing the semaphore, to start execution. The Secure Process, on receiving the signal, does the corresponding cryptographic operation on the input data with the secret keys. The output of the operation is placed in the non-secure memory area from where the less privilege process can read the output.

3.4. Secure Boot-Loader and Code Signing

The secret keys, though protected by hardware security measure, have to be exposed for through some API's for use. It is necessary to ensure that the firmware of the device cannot be modified so that an unauthorized user can use these API's to extracts the secret keys from the device. The firmware also contains secure critical code such as code that handles security critical hardware configuration like Internal RAM configuration to specify access permissions. Any attempt on overriding the firmware components of the device thus must be turned down. The presence of Secure Bootloader can ensure this.

On startup before loading the firmware code, the Secure Bootloader checks whether the firmware is genuine or not and prevents the device from booting up if the device firmware is modified or replaced. An example of a Secure Bootloader implementation is discussed below.



* The public key and the private key are generated by device manufacturer and is used for the the signing and verification of the device firmware code.The private key has to be kept secret by the device manufacturer

Secure Bootloader resides on a write protected ROM inside the Secure SoC. Keeping Secure Bootloader in a write protected ROM ensures that the Secure Bootloader itself is never modified. In addition to the general boot initialization code, the Secure Bootloader contains a signature verification module of the firmware code and the code verification public-key to verify the firmware code.

The firmware code is signed using the device manufacturer's code verification private-key. The Secure Bootloader, on boot up checks the validity of the code by verifying the signature using the code verification public-key.

Though the private-key that used to sign the firmware code is never shipped along with the device, it has to be kept secret by the device manufacturer. The compromise in the secrecy of the private-key that used to sign the firmware code enables anyone, who is having access to the private-key, to write and sign a code that is acceptable by the Secure Bootloader.

If the firmware of the device is non-upgradeable then the level of security in boot loader can be enforced in much simpler method without the presence of a Secure Bootloader. In such case, writing the entire firmware in a read-only memory and configuring the boot loader to boot only from the given read-only memory area can prevent any unauthorized firmware component to run on the device. But in many case, non-upgradeable firmware brings too much limitations on a product.

There are files, like root CA certificate, which when modified can result in the compromise of the device security. Such files also need to be signed along with the firmware code of the device.

There are also file specific to each device, like encrypted secret keys or device certificates, which when modified prevents the device from secure data transfer but does not compromise the security of the device. The signing of these device specific files will cause overhead during the production of the device or during the upgrade of the device firmware where the device manufacturer needs to sign the firmware code for each device. Since the device security is not compromised by the modification of these files, these files can be kept in the device without being signed.

3.5. Encryption and decryption engine

In many secure protocol implementations, the shared secret generated by the Key Agreement algorithm as discussed in section 2.2 is used as a master key and the sub-keys are generated that are used for the process of encryption and decryption. When used as the master key, the shared secret is stored in the device till expiry time as specified as the protocol and the lifetime will be in the order of days or even months depending on the protocol of data transfer. But the lifetime of sub-keys will be generally small in the order of seconds. In this case the security requirement of the shared secret is higher and hence stored securely in the device. But the security requirement of the sub-keys is not that critical as that of the secret keys like shared secret or certificate private-keys. The sub-keys generation protocol also will be such that it will be impossible to derive the master key from the sub keys. In such cases where the security of keys (sub-keys) used for encryption and decryption is not so critical, the encryption and decryption engine (module) can reside outside the secure SoC. The sub-keys are generated inside the Secure SoC and are passed to the encryption/decryption engine outside the Secure SoC. In some other protocols the shared secret or secret keys itself for encryption/decryption. In such case, the encryption/decryption engine should reside inside the SoC to prevent the key from being available the bus outside Secure SoC.

The encryption and decryption engine thus have the choice to reside inside or outside the Secure SoC depending on the security need of the keys used for encryption/decryption.

3.6. System Time

The digital certificate of a device generally comes with validity period. The validity periods varies across different protocol implementation. Some protocols like SSL specify a fixed validity period of few years or decades whereas other protocols like DTCP specifies infinite validly for a certificate.

The system time in an embedded device will generally have interfaces to user to set or modify the system time. For certificate verification process the device should maintain a system time that is different from the system time modified by the user so that the users are not able to modify the system time and make the device accept an expired certificate. It is also important that the timer keeps counting even after when the device is in the switched off state.

Unauthorized modification of system time is not so critical in many cases where the device handles certificates having validity period of decades, i.e. 20-30 yrs or more. This timeframe is sufficiently larger than the lifetime of many devices. Also, the chance for root CA to update the root certificate within this time frame is also high. If the CA changes a root CA certificate, the devices must update the root and the intermediate CA certificates and should acquire a new device certificate from the CA.

4. Proprietary Technologies for secure data transfer

Proprietary security modules are implemented on some devices for the secure data transfer between the compliant devices. The modules can be some or all the modules mentioned in section 2 like Encryption/Decryption, Key Agreement, Digital Certificate or Digital Signature modules.

The proprietary technologies implemented in device are usually kept secret to enforce additional security for data transferred between the devices. It is therefore ensured by the device manufacturer to not to disclose the technology, from or outside the device, and thereby compromising the security enforced by these technologies. The device manufacturer should find a method to store the software module in the device so that the secrecy of the technology is not compromised from the device. Code (binary) of these proprietary technology software modules usually will be in clear text inside the device. Thus if someone can get access to the code in the device it may be easy to extract and understand the implementation of the software module with the help of code reengineering tools like 'objdump'. Thus the device must prevent access to retrieve the code by placing it inside Secure SoC or encrypting and storing the code using a secret key knowing only to the device manufacturer. If the code is stored encrypted, the boot loader of the device must support the decryption and loading of the code during execution and the secret key used for encryption and decryption of the code can be stored in the device by the methods specified in section 3.

A proprietary technology can be either a device specific or a standard specific. In the case of a device specific technology, the secure data transfer can happen only between the devices of same manufacturer. An e.g. can be Apple’s iPod music player. Apple can use their proprietary technology for secured transfer of files between their compliant devices or applications like iPod, iTunes and iTunes store. Since the devices are from a single manufacturer, keeping it secret, other than any user retrieves and understands the code through any weak links from the device, seems to be practically possible.

If the proprietary technology is specified by a standard body, it can be used to enforce additional security between the devices of different manufacturer. In this case the standard body will disclose the technology to the device manufacturer on an agreement, usually legal, to not to disclose the technology. Few e.g. of such technologies are M6 encryption technology used in DTCP [11] and DFAST scrambling algorithm used in OpenCable’s CableCARD-Host interface protocol [10]. Maintaining the secrecy of the technology becomes more and more difficult as the number of manufacturer to whom the technology is circulated increases. As the number of manufactures increases, the security provided by these technologies becomes minimal or even negligible. In this case It is therefore important that the security of data transfer of such device does not rely only on the secrecy of these proprietary technologies.

5. Revocation List

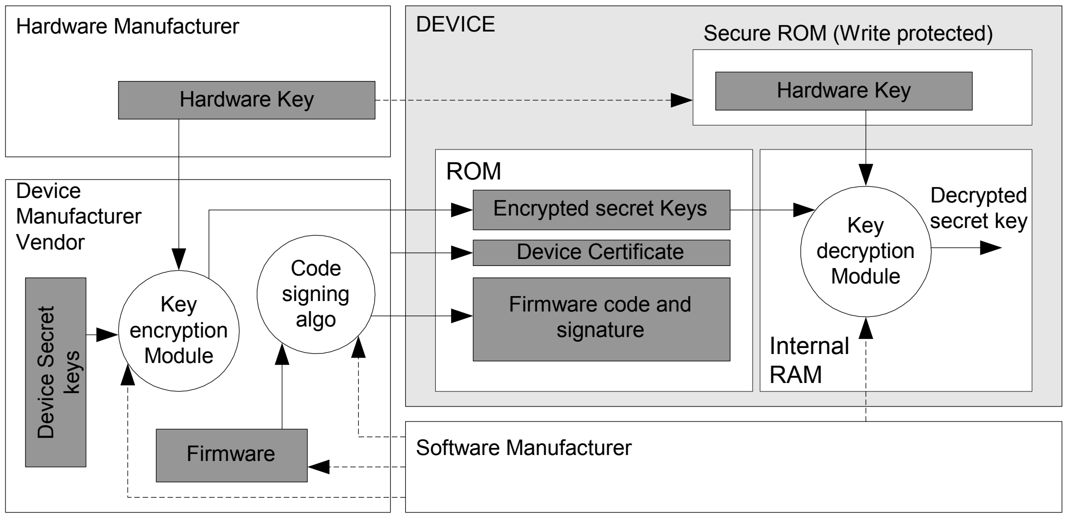
Though the security measures as explained in section 3 are used for secure storage of secret keys in the device, the chances of retrieving it cannot be ruled out completely. In devices with only 'tamper evident' security measures it is possible to retrieve the secret keys from the device's NVM but with some physical tampering of the device.

Generally each device will have a separate set of secret key so that compromising one devices secret key doesn’t compromise the security others. But in the case of devices handling the copy-protected digital multimedia content through network, compromising the secret key of a single device results in the compromise of the security of all the copy-protected content handled by that device. One vulnerable device can thus results in helping an unauthorized device to access the copy protected content, decrypt it and distribute countless copy of the copy protected content. If came to notice, a broadcaster can prevent such device to communicate with other devices in the network to get the copy protected content by adding the device ID in a list called as the revocation list. All trusted devices would have the access to revocation list from the broadcaster will stop communicating with the device whose security is compromised.

6. Keys and certificate handling during device manufacture

In many cases, the different hardware and software components in an embedded device are supplied by different vendors. The hardware vendors provide the hardware component and the associated drivers for the device whereas the software vendor provides the software components. The device manufacturer or the deice vendor assembles these hardware and software components to make the product, which is marketed with an aim of attaining revenue. The secret key for each device has to be loaded in to the device during manufacture. It is usually is the interest of the device vendors to protect the secret keys of the devices and thus the device vendors may refrain from sharing the secret keys to different hardware and software vendors. But atleast some part of the software has to use the shared secret and also as explained in section 3, the device need hardware support to store the secret key securely in the device. With the support of hardware and software vendors, the device manufacturer can store secret keys securely in the device, also not disclosing it to the hardware and software vendor. Two methods are explained here to handle the secret keys during production of the device.

1. The hardware vendor supplies hardware with write-protected Secure ROM, pre-programmed with unique random number for each device or for a set of devices. This random number can be used as a hardware master key to encrypt device secret keys.
2. The hardware vendor supplies hardware with programmable Secure ROM that can be programmed by the device manufacturer with device secret keys



Handling of secret keys when hardware vendor supplies the hardware with pre-programmed Hardware Key

In Case 1 where the hardware comes with a pre-programmed random number inside the Secure ROM, the random number acts as a hardware key or a master key that can be used to encrypt the secret keys of the device. The software vendor provides software methods/code to encrypt/decrypt secret keys using the hardware key. The device manufacturer can use the encryption method to encrypt and store the secret keys inside the device using the hardware key. The decryption method will be the part of device firmware that goes along with device to decrypt and use the stored secret keys using the hardware key.

In case 2 where the hardware comes with programmable Secure ROM, the hardware vendor also supplies the software module or drivers to program/write the Secure ROM with the keys. The software vendor or the device vendor now will have the flexibility to decide whether to store the secret keys or a master key to encrypt the secret keys to be stored in the Secure ROM.

Usually the device certificates of each device will be different. Since the device certificate and the encrypted secret keys are different for each device, these will not be the code-signed along with the firmware as discussed in section 3.4. Otherwise the device manufacturer needs to sign (using any software provided by the software manufacturer) the firmware code for each device. The device manufacturer will load the device certificate or encrypted keys for each device on the ROM location as specified by the software vendor. The certificate handling software component loads the certificate for processing from the specified location.

The root CA certificate is unique for all the devices communicating using the same protocol and its unauthorized modification or substitution results in compromise of device security, the root CA certificate is code-signed along with the firmware code.

7. Conclusion

The available security measures for secure transfer of data between two devices are matures enough to defeat any third party to decrypt and get access to the protected content. But the security measures available to protect the stored secure data, like secret keys, within the device are not yet foolproof. A tamper resistant protection mechanism in a device may require hardware circuit to zeroise the secret keys when a physical attack is made to the Secure SoC. The more the hardware security measures implemented in a device to protect its secret keys and other secure data, the more costly the device will be. Thus the hardware security measures implemented in the device are a trade of between the cost of implementation and the cost of the data protected. Achieving a cost effective yet foolproof method to protect the secret keys and secure data within the device will be a boon to the owner of the contents that needs security, especially to the content provider of copy-protected digital contents.

Reference

[1] Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, Handbook of Applied Cryptography, CRC Press, 1996

[2] FIPS PUB 186-2, Digital Signature Standard (DSS), January 2000, Available at http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf

[3] RSA Laboratories, PKCS#1 v2.1: RSA Cryptography Standard, June 2002, http://www.rsa.com/rsalabs/node.asp?id=2125

[4] RFC 2631, Diffie-Hellman Key Agreement Method, June 1999, Available at http://tools.ietf.org/html/rfc2631

[5] Certicom, Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography, Version 1.0, September 2000, Available at http://www.secg.org/download/aid-385/sec1_final.pdf

[6] ITU, Recommendation X.509, Available at http://www.itu.int/rec/T-REC-X.509-200508-I

[7] FIPS 197, Advanced Encryption Standard (AES), November 2001, Available at http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[8] NIST, Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher, May 2004, Available at http://csrc.nist.gov/publications/nistpubs/800-67/SP800-67.pdf

[9] FIPS 140-2, Security requirements for cryptographic modules, May 2001, Available at http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

[10] OpenCable, OpenCable System Security Specification, October 2006, Available at http://www.cablelabs.com/specifications/OC-SP-SEC-I07-061031.pdf

[11] DTLA, Digital Transmission Content Protection Specification Volume 1 (Informational Version), October 2007, Available at
http://www.dtcp.com/data/info%2020071001%20DTCP%20V1%201p51.pdf

[12] Anoop MS, Public Key Cryptography – Applications algorithm and mathematical explanations, May 2007,
Available at http://msitbox.blogspot.com/2008/03/public-key-cryptography.html

[13] Anoop MS, Elliptic Curve Cryptography - An implementation guide, May 2007,
Available at http://msitbox.blogspot.com/2008/03/elliptic-curve-cryptography.html

[14] Openssl, http://www.openssl.org

[15] Certicom, http://www.certicom.com/index.php?action=ecc_tutorial,home

[16] RSA Laboratories, http://www.rsa.com/rsalabs/node.asp?id=2193