Security in Embedded Computing

René Burger

Sommerakademie in Leysin, August 2016

Abstract

There are different ways to attack computer systems, but protection against many software attacks, like brute force or exploiting flaws in the design of a system, works in most of the cases on “normal” computers. Because of limitations in computational power, cost, and in the possibility of updating i.e. patching, the security in embedded systems has to be concerned in a different way. The heavily increasing usage of embedded systems, especially IoT devices, requires action. It is important to consider security during the design process of an embedded system, otherwise once found security flaws are really hard to patch. While the computational power of many embedded systems is not sufficient for most cryptographic tasks, TPM modules as an addition on an embedded system could enhance the possibilities of building a really secure embedded system.

Contents

1 Introduction

Today, more and more devices get connected to the internet. Usually there is no time we are not online anymore, but now also the devices surrounding us are using the internet or another network technology (Bluetooth, NFC, proprietary wireless protocols etc.). Our lighting, heating and home security devices, TV, Blu-ray Players, maybe in some time your washing machine and refrigerator will be controllable via internet. Tesla now introduced an App, which makes you able to move your Car forwards and backwards with your smartphone.

Also, many industrial machines or medical devices in hospitals are controllable over the World Wide Web today.

In nearly all those applications, security should be considered. If a malicious entity gets access to one of these devices, there will be problems. In the “best” case, your personal data can be accessed, or someone will be able to control the lights in your living room. In the worst case, the outcome could be life-threatening.

So, security should be concerned in embedded systems, especially in IOT devices.

But sometimes, this does not work as expected. In 2013, the computer magazine c’t gained access to several industrial control interfaces: They were able to manipulate the pumps of a brewery, a heating plant and to ring the bells and control the lighting of a church. This all just by using open port scans and trying out standard passwords.

In 2015, IP cams sold by ALDI were delivered with a firmware that could easily be manipulated. Everyone with basic technological knowledge could look into thousands of living rooms (and tilt the camera etc.) without great effort.

Many of these systems have not been patched yet, and with more and more end-users without much technological knowledge using those devices, the problems are growing even bigger.

In the following, general security terms and basic algorithms in cryptography will be explained. After that, an example for a secure communication will be shown.

The following sections will concern different kinds of attacks (on software level and very briefly on hardware level), explain why embedded devices are different from normal computers concerning security and will give an introduction into TPM and HSM- technology.

2 General information on security

2.1 Definitions of security terms (from [1])

To understand cryptography, it is important to know some cryptographic terms and definitions of them. Those will be shortly summarized in this section.

2.1.1 Definition of security

The most obvious at first: What Security means differs greatly from the point of view. An End-User may be concerned of his personal data being stored somewhere of being transported through the internet. The Manufacturer of a device could want to keep the secret of the proprietary firmware on his device, and a content provider wants copy protection of his media (DRM) at first place.

A malicious entity could steal your data, impersonate you or change documents without being detected. This malicious entity can also vary, depending on who wants which information. For example, if you want to duplicate a motion picture with a copy protection, you are the malicious entity, working against the content provider.

2.1.2 Secrecy

Secrecy is defined as the means of preventing an unauthorized observer of a message from determining its contents. A Message can be any array of bytes sent between two parties. To obtain secrecy, you can encrypt the message.

2.1.3 Shared secret

A shared secret is a value that is known only to two parties. This can be a password or an encryption key.

2.1.4 Integrity

Integrity is a means of indicating that a message has not been altered during storage or transmission. Even if a message is encrypted, it could have been corrupted by a malicious entity without having a precise indication for that.

To maintain integrity, messages usually get hashed using HMAC.

2.1.5 Authentication

Methods of authentication ensure that a message really was sent by the creator that the message claims to be sent by. This is to verify, that only the creator could have send the message, and is usually obtained by asymmetric encryption.

2.1.6 Authorization

Authorization proofs, that a user (or device) is permitted to perform an operation.

2.1.7 Anti-Replay
Anti-Replay is a means of preventing an attacker from reusing a valid message.

2.1.8 Nonrepudiation
Prevents the sender of a message from claiming that they did not send the message.

2.1.9 Example
As an example for all these terms, we can imagine an electronic purchase in an online-shop:

The message contains the number of items ordered and confidential customer information, like the credit card number and the address. Means to sustain integrity are used that the message cannot be altered during transit, for example from 3 items to 300. Authentication proves, that the order really came from the buyer and authorization checks that the buyer really is permitted to purchase these items. Anti-Replay prevents an attacker from sending the buyer's message again to purchase the items multiple times.

2.2 Key- algorithms in cryptography

2.2.1 Secure Hashes
A cryptographic hash is the digest of a message of any length. The hash has always a fixed length, and is always the same if the message is the same. It is computationally infeasible (so it takes very much time to compute) to construct two different messages with the same hash, and it is infeasible to derive the message from a given hash.

An example for a hashing algorithm is SHA-256. This algorithm produces hashes of 256 bits / 32 bytes.

Hashes are the building blocks of many other operations, such as HMACs, tickets and digital signatures.

2.2.2 HMAC
You can imagine HMACs (keyed-hash message authentication code) as keyed hashes. They perform a secure-hash operation, but mix in a shared-secret key (HMAC-key). Only a party knowing the HMAC-key is able to calculate the same hash from a given message. HMACs are used to check, whether a user is authorized (because he knows the key) and that the message has not been altered (integrity).

For Example, a user sends a message containing a command to a machine and HMACs the message with his "authorization code" that is a shared secret with the machine. The machine then knows that the user was authorized and can check if the message is integer, if the HMAC the machine computes equals the HMAC the user sent with it.

2.2.3 Symmetric encryption

Symmetric encryption algorithms use a symmetric key to encrypt a message. This means, that the same key is used to decrypt the message again. The most commonly used symmetric-key-algorithm is AES (advanced encryption standard). Symmetric encryption is used to provide secrecy, so it keeps the data secret from all observers to secure communication.

It does NOT provide integrity or authenticity, to ensure this, you need to use HMAC and/or asymmetric encryption

2.2.4 Asymmetric encryption

This technology uses two different keys: One public key and one private. Asymmetric algorithms basically use mathematical one-way functions, where it is easy to calculate one direction, but the other is computationally infeasible. In RSA, a message encrypted by the public key can be only decrypted by the corresponding private key and vice versa. Deriving the public key from the private key is relatively easy, but deriving the private key from the public key is computationally infeasible.

Asymmetric algorithms are much slower than symmetric algorithms, because the mathematics behind takes more computational power to execute. Besides RSA, there is also ECC (elliptic curve cryptography) as an asymmetric encryption technology, which is very powerful.

Asymmetric algorithms can e.g. be used to sign data. That means the owner of a message can encrypt data (usually a digest of the message) with his private key. Now everyone can decrypt that digest with the public key and knows that only the owner of the private key could have encrypted that data.

They are also used to share data: Everyone can encrypt the message with the public key of the recipient, and only the designated receiver of this message can decrypt it.

2.3 How can communication be secured

This is a theoretical example, how you could build up a secure way of communicating through a data path, where every message could be read or altered by malicious entities. This could be e.g. the communication between a server and a client through the internet.

The first idea is to just use asymmetric encryption. The client uses the server's public key to encrypt the message, so only the server will be able to read the plaintext, and the server can use the client's public key to encrypt the message so only the client can decrypt it. But asymmetric encryption is really slow. It would take much computational power to always en- and decrypt the messages with asymmetric encryption. The solution is to only use asymmetric encryption to exchange a symmetric key. After the key-exchange, the communication can be encrypted with a symmetric algorithm with the shared-secret key.

This seems like a legit solution, but there is another problem: How do we ensure, that the public key of the server is really belonging to the wanted server (and vice versa)? Secrecy won't be preserved if you encrypt a message with the attacker's public key. Man in the middle (MITM) attacks work like this (In this example, the client wants to send a message to the server): The MITM sends the client his public key (instead of the server's) and encrypts the message received by the client. Now he has the message in plaintext and can alter it in the way he wants. He then encrypts the altered message with the server's public key and sends it to him – and no one knows that the message has been altered.

The solution to prevent this method is to ensure that a key really belongs to its owner by certificates: If the public key of the server is certified and the client sends a symmetric key to the server encrypted by his (now certified) public key, a secure communication is obtained.

Digital certification works like this: They confirm, that a public key really is connected to the owner it states it came from. This ensures that a message (encrypted with the public key) really can only be decrypted by the declared owner. A certificate includes the public part of the key being certified and some attributes of that key. This is then signed by a so-called certificate authority (CA). The certificate is encrypted with the CA's private key, and can be decrypted by the (well-known) CA's public key. If you trust the CA (that is usually an independent organization), you can also trust that the public key of an owner really belongs to him.

The CA's public key can also be certified by another (higher-level) CA, forming a certificate chain. At the end of this chain, it terminates at a root certificate that is very well-known and trusted without any cryptographic proof.

3 Different kinds of attacks

3.1 Software attacks

3.1.1 Exploiting a flaw in the design

This is the more sophisticated way to get access to a system. To get rights you usually do not have, you have to find an exploit in the software. This might be a bug or security gap in the firmware of a system. For example an algorithm being used that is feasible today and thus not safe anymore (e.g. SHA-1), a security gap in the web interface of a router or printer or an unhandled variable overflow in the firmware (e.g. in C).

You can only be safe against this kind of attack, if your system design really is waterproof and uses no software that could have any security gaps.

3.1.2 Brute Force

The easier way to get into a system is to use brute force. That means you simply try any combination of a key or password many times, until you have successfully found the right one. A good cryptographic algorithm should be computationally infeasible, so it would take an impractical amount of time for trying every possible combination and brute force attacks won't success. But many passwords can be cracked quite easily by brute force attacks, because a human has to be able to remember the password. Often, passwords are not a random combination of characters but contain words that can be found in the dictionary. "Good" attackers first try dictionary words, then combinations of words and numbers, special characters and so on. With this way, you can crack a standard password in not much time, but usually, a server makes the login process much slower if the password was not entered right several times in a row that makes a brute force attack much slower and again (if a relatively good password is used) computationally infeasible.

3.1.3 How are embedded devices different from normal computers?

Embedded systems have, speaking in general, much slower processors than normal computers, so it is much more difficult to implement safe cryptographic systems because they often need a relatively high computing performance. In addition to that, embedded systems can't easily be updated or patched most of the time, because no one feels responsible for that. If the system works, why should I update it? This is an attitude that often leads to well-known security gaps that often are not patched for a very long time and can be exploited. In some cases, parts of a system can't even be patched if someone wanted to do that, because many drivers or components of a system only exist as a "binary blob" that can't be patched unless you reflash the chips. Even if an OS is updated, underlying systems could have security gaps that often can't be closed. Also, many manufacturers use older chips because they are cheaper and powerful enough for the application, but if a security gap is discovered in an older chip, no one will feel responsible to fix it. Even if you could patch the gap, maintaining older chips is not a priority for the manufacturers.

To conclude this, the security of an embedded system has to be considered during the design cycles. It is really hard to patch an embedded system after it has been deployed.

3.2 Hardware Attacks

Systems can not only be compromised with software, but also with hardware methods (and expensive equipment and sophisticated techniques): Invasive attacks include the probing of inter-component communication of a system, e.g. to read out the memory. With access to the internals of a device, the ability to manipulate and interfere with the system and some reverse-engineering, an attacker can be able to completely manipulate the operations of a system. Non-invasive attacks use the monitoring of timing, power and electromagnetic radiation to detect what a component is computing. Depending on which step (of a cryptographic algorithm) a component executes, it draws different currents, takes more time or emits different electromagnetic fields, what leads to understanding what a component is doing right now.

4 TPM and HSM

One example of a security system, that can be considered to be used in an embedded System are TPM modules. TPM means “trusted platform module”. They are a dedicated cryptographic coprocessor that are mounted on the mainboard of a device and used to perform cryptographic tasks. These can be used as a storage and management of keys, to perform cryptographic computations like the encryption or decryption of small messages, and to store and manage certificates. Another very important function of a TPM to ensure a secure cryptography is to generate random numbers. This is the base of the generation of new keys, that can’t be predicted by an observer.

A TPM can be useful for many tasks:
To identify a machine - not by their MAC of IP address, but by security identifiers – to grant access to networks, to authorize a payment or to allow an action only authorized machines are allowed to execute.
A TPM can also encrypt and decrypt (so manage) keys for the encryption of files on a hard drive or to decrypt files that the system got from another device or had to outsource. For example, full disk encryption (like BitLocker on Windows systems with a TPM) or the encryption of passwords in a password manager can securely be done by a TPM.
These modules can also be used to check if the boot sequence of a system has not been compromised by using PCR values. This ensures that a system has not been altered even before the OS has started.

While TPMs, that are small onboard modules with a very low cost-factor, are mostly used for authentication and to manage a small amount of keys/certificates on a single device, there are also bigger dedicated modules as extension of a workstation, server or network. These are then called HSM: Hardware Security Modules. As an external module, accountable for a PC or an entire network, they can be used for more power consuming cryptographic computations and key-management in large numbers. These HSM modules are also protected against tampers like bus probing (attacks against the hardware), so not only software attacks. They are mostly used in very high security environments like bank servers.

5 Conclusion

Security gaps in embedded systems are really hard to patch, and the low security standard of many embedded IOT devices is a growing problem. Many IOT devices are vulnerable and can be accessed through the internet with very low effort and some basic knowledge about computer systems.

Embedded systems can be successfully secured, but not after they have been deployed. Security has to be considered during the design process of a device, it is not possible to change much of the system after release. The consideration of using TPM modules in embedded devices is a way of making our industry 4.0 and IOT world a place that is not a playground for hackers, where we can enjoy all the advantages of IOT and smart home without having the fear of some people or companies being able to control all the devices around us.

Bibliography

Arthur, W., Challener, D., & Goldman, K. (2015). *A Practical Guide to TPM 2.0.* APress Media LLC.

Kocher, P. et al. (2004, 06). Security as a New Dimension in Embedded Systems Design. *DAC*, pp. 753-760.

Patterson, D. A., & Hennessy, J. L. (2014). *Computer Organization and Design.* Elsevier Inc.

Schneier, B. (2014). *https://www.schneier.com*. Retrieved 01 09, 2016, from https://www.schneier.com/blog/archives/2014/01/security_risks_9.html