Università di Pisa

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Security in automotive microcontrollers of next generation

Candidato: Giorgio Boccini

……

Relatori: Prof. Ing. Luca Fanucci

……

Ing. Massimiliano Donati

……

Anno accademico 2013/2014

Summary

Chapter 1
Introduction

This thesis deals with the implementation of a hardware security module that will be used to ensure the privacy in a bus communication in the automotive environment. This activity is born in collaboration with the RENESAS Electronics, an industry that produces automotive components and more else. The issues addressed are continually developing and they will be incorporated in the in the next generation of automotive microcontrollers. The security in the automotive environment is an emerging problem and today does not exist a standard solution to solve this problem, for this reason this thesis activity start with a search on the state of the art of the automotive microcontrollers, to understand the various solutions on the market. Afterwards, following the RENESA's directives the activity continues with a deep study of the embedded security and specifically, on the security in a CAN to CAN bus communications. The scope of this study is the realization of a hardware module, to be inserted in an existing CAN IP, which ensure the privacy aspect.

In detail, the thesis is organized as follows:

- Chapter 2, **Security**. Issues on security in computer systems have been known for a decade. After a brief summary of the concepts of security, we will see, as the embedded security is different under many aspects to the well-known security in computer systems.
- Chapter 3, **Automotive security**. The Automotive security is a particular field of application of the more general embedded security that we have just introduced in the previous chapter. This chapter answers to an important question: how did the issue of security bear in the automotive environment?
- Chapter 4, **State of art.** After the two previous chapters we have a good background in automotive security, we can understand the choices made by other companies about security in their microcontroller automotive, and then we can draw a good state of the art.
- Chapter 5, **CAN protocol**. The themes covered so far are general, and they forms a good starting point to understand the security problem in the automotive environment, if we want to narrow down the security problem to the communication bus CAN, we need a fast review of the relative standard to understand its weakness regard the security.
- Chapter 6, **Advance Encryption Standard**. As will be clearer in the following, to ensure the privacy in our environment we need to use the cryptography and in particular the AES (Advance

Encryption Standard), to implement as best we can this algorithm in hardware, we need a good knowledge of the relative standard.

- Chapter 7, **AES-128 HW Implementation.** Finally, we will see two hardware implementation of the AES-128 algorithm that they could be a starting point to a feasibility study for the introduction of the privacy aspect in an existing bus communication IP.
- Appendix, models (System Verilog) and code (Verilog) developed.

Chapter 2
Security

Since the early days of writing, politicians, diplomats and military commanders understood that it was necessary to provide some mechanism to protect the confidentiality of correspondence and to have some means of detecting tampering. Julius Caesar is credited with the invention of the Caesar cipher ca. 50 B.C., which was created in order to prevent his secret messages from being read should a message fall into the wrong hands, but for the most part protection was achieved through the application of procedural handling controls. Sensitive information was marked up to indicate that it should be protected and transported by trusted persons, guarded and stored in a secure environment or strong box. As postal services expanded, governments created official organisations to intercept, decipher, read and reseal letters (e.g. the UK Secret Office and Deciphering Branch in 1653).

In the mid-19th century, more complex classification systems were developed to allow governments to manage their information according to the degree of sensitivity. The British Government codified this, to some extent, with the publication of the Official Secrets Act in 1889. By the time of the First World War, multi-tier classification systems were used to communicate information to and from various fronts, which encouraged greater use of code making and breaking sections in diplomatic and military headquarters. In the United Kingdom, this led to the creation of the Government Code and Cypher School in 1919. Encoding became more sophisticated between the wars as machines were employed to scramble and unscramble information. The volume of information shared by the Allied countries during the Second World War necessitated formal alignment of classification systems and procedural controls. An arcane range of markings evolved to indicate who could handle documents (usually officers rather than men) and where they should be stored as increasingly complex safes and storage facilities were developed. Procedures evolved to ensure documents were destroyed properly and it was the failure to follow these procedures, which led to some of the greatest intelligence coups of the war (e.g. U-570).

The end of the 20th century and early years of the 21st century saw rapid advancements in telecommunications, computing hardware and software, and data encryption. The availability of smaller, more powerful and less expensive computing equipment made electronic data processing within the reach of small business and the home user. These computers quickly became interconnected through the Internet.

The rapid growth and widespread use of electronic data processing and electronic business conducted through the Internet, along with numerous occurrences of international terrorism, fuelled the need for better methods of protecting the computers and the information they store, process and transmit. The academic disciplines of computer security and information assurance emerged along with numerous professional organizations – all sharing the common goals of ensuring the security and reliability of information systems.

2.1 Information security

Information security, sometimes shortened to InfoSec, is the practice of defending information from unauthorized access, use, disclosure, disruption, modification, perusal, inspection, recording or destruction. It is a general term that can be used regardless of the form the data may take (electronic, physical, etc…)

Two major aspects of information security are:

1. IT security: Sometimes referred to as computer security, Information Technology Security is information security applied to technology (most often some form of computer system). It is worthwhile to note that a computer does not necessarily mean a home desktop. A computer is any device with a processor and some memory (even a calculator). IT security specialists are almost always found in any major enterprise/establishment due to the nature and value of the data within larger businesses. They are responsible for keeping all of the technology within the company secure from malicious cyber-attacks that often attempt to breach into critical private information or gain control of the internal systems.

2. Information assurance: The act of ensuring that data is not lost when critical issues arise. These issues include but are not limited to natural disasters, computer/server malfunction, physical theft, or any other instance where data has the potential of being lost. Since most information is stored on computers in their modern era, information assurance is typically dealt with by IT security specialists. One of the most common methods of providing information assurance is to have an off-site backup of the data in case one of the mentioned issues arise.

2.2 IT security

IT security has gained central importance for many new automotive applications and services. On the production side, we observe that the cost for electronics and IT is approaching the 50% threshold of all manufacturing costs. Perhaps more importantly, there are estimates that already today more than 90% of all vehicle innovations are centered around software and hardware (admittedly not only digital hardware, though). IT systems in cars can roughly be classified into three main areas:

- Basic car functions, e.g., engine control, steering, and braking.
- Secondary car functions, e.g., window control and immobilizers.
- Infotainment applications, e.g., navigation systems, music and video entertainment, and location-based services.

Almost all such applications are realized as embedded systems, that is, as devices, which incorporate a microprocessor. The devices range from simple control units based on an 8-bit micro controllers to infotainment systems equipped with high-end processors whose computing power approaches that of current PCs. The number of processors can be 80 or more in high-end cars. In a typical automobile, the devices are connected by several separate buses. Not surprisingly, many classical IT and software technologies are already well established within the automotive industry, for instance hardware-software co-design, software engineering, software component re-use, and software safety. However, one aspect of modern IT systems has hardly been addressed in the context of automotive applications: IT security. Security is concerned with protection against the manipulation of IT systems by humans. The difference between IT safety and security is depicted in Fig 2.1

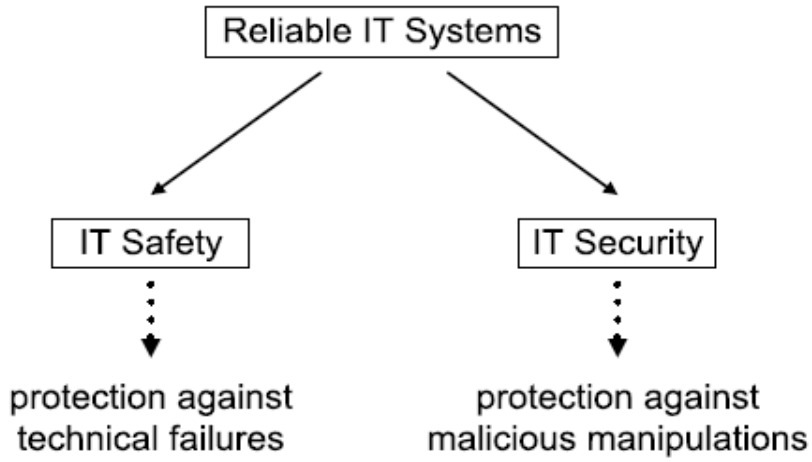


Figure 2.1: The relationship between IT safety and security

Software and hardware safety is a relatively well-established (if not necessarily well-understood) field in the automotive industry, IT security, on the other hand, is just beginning to emerge as a proper sub-

discipline within the field of automotive IT. Of course, there have been niche applications in the automotive domain, especially concerned with electronic immobilizers that have always relied on security technologies. However, the vast majority of software and hardware systems in current cars are not equipped with security functionality. This is not entirely surprising for two reasons:

1. Many past car IT systems did not need security functions as there was very little incentive for malicious manipulation in traditional applications.

2. Security tends to be an afterthought in any IT system. Achieving the core function, i.e., getting a telematic system working or enabling remote software updates is the primary goal of every system designer and implementer.

A prime example of such an IT-system is the Internet, which is only, from few years after some decades of existence, being equipped with rudimentary security functions.

The situation has changed dramatically with respect to the first argument given above. Already today, there is a multitude of quite different car sub-systems that are in desperate need for strong security functions in order to protect the driver, the manufacturer and the component supplier. Current examples of car functions with need for security include the large field of software updates, also known as "flashing" or "chip tuning". Future cars will become even more dependent on IT security due to the following developments:

- It is predicted that an increasing number of ECUs (electronic control units) will be reprogrammable, a process that must be protected.
- Many cars will communicate with the environment in a wireless fashion, which makes strong security a necessity.
- New business models (e.g., time-limited flash images or pay-per-use infotainment content) will become possible for the car industry, but will only be successful if abuse can be prevented.
- There will be an increasing number of legislative demands, which can only be solved by means of modern IT security functions, such as tamper resistant tachographs, secure emergency call functions, secure road billing etc.
- Increasing networking of cars will allow the collection of data for each driver (e.g., driving behaviour, locations visited), which will put high demands on privacy technology.
- Future cars will often be personalized, which requires a secure identification of the driver.
- Electronic anti-theft measures will go beyond current immobilizers, e.g., by protecting individual components.

As we can see from the, not necessarily complete, list in the previous page, IT security will be an important topic for many future car technologies. For some future applications, such as business models based on Digital Rights Management, IT security will even play the role of an enabling technology.

We would like to stress at this point that almost all target platforms with in cars, which will incorporate security functions, are embedded systems, rather than classical PC-style computers. Hence, the technologies needed for securing car applications belong often, but not always, to the field of embedded security. The difference between embedded security vs. general IT security will be discussed in more detail afterwards.

2.3 Embedded security

2.3.1Embedded Security vs. General IT Security

Since the late 1990s embedded security, sometimes also referred to as security engineering or cryptographic engineering, has emerged as a proper sub discipline within the security and cryptography communities. Embedded security is often quite different from the security problems encountered in computer networks such as LANs or the Internet. For such classical networks, there exist established and relatively mature security solutions, e.g., firewalls, encryption software, and intrusion detection systems. The topics with which embedded security deals are, generally speaking, closer related to the underlying software and hardware of the target device, which is to be protected. Arguably, the most important event at which embedded security technologies are treated from a scientific viewpoint is the CHES (Cryptographic Hardware and Embedded Systems) Workshop, which started in 1999. Even though there are certainly many aspects of security that are shared by embedded devices and general computers, there are a number of key differences:

- First, embedded devices tend to have small processors (often 8-bit or 16-bit micro-controllers) which are limited with respect to computational capabilities, memory, and power consumption. Modern PCs, on the other hand, are very powerful and in most cases do not limit the use of cryptographic functions.
- Second, potential attackers of embedded systems have often access to the target device itself, e.g., an attack of a smart card only makes sense if one actually has physical control over the smart card. On the other hand, attacks against traditional computer networks are almost always performed remotely.
- Third, embedded systems are often relatively cheap and cost sensitive because they often involve high-volume products, which are priced competitively. Thus, adding complex and costly

security solutions is not acceptable. By comparing typical prices (e.g., a laptop vs. an ECU) one easily notices a ratio of 1–2 orders of magnitude which, of course, limits the costs that can be spent on security for embedded solutions.

2.3.2Cryptographic Algorithms in Constraint Environments

Even though security depends on much more than just cryptographic algorithms - a robust overall security design including secure protocols and organizational measures are needed as well - crypto schemes are in most cases the atomic building blocks of a security solution. The problem in embedded applications is that they tend to be computationally and memory constrained due to cost reasons. (Often they are also power limited, but since automotive applications are often powered by their own battery, low-power crypto is not such an important topic in the car context.) It is now the task of the embedded security engineer to implement secure crypto algorithms on small devices at acceptable running times. Crypto schemes are divided into two families: symmetric and asymmetric algorithms. The first group is mainly used for data encryption and message integrity checks. Symmetric algorithms tend to run relatively fast and often need little memory resources. There exists a wealth of established algorithms, with the most prominent representatives being the block ciphers DES (Data Encryption Standard) and AES (Advanced Encryption Standard.) The family of stream ciphers can be even more efficient than block ciphers and are, thus, sometimes preferred for embedded applications. In almost all cases, it is a wise choice to use established, proven algorithms rather than unproven or self-developed ones. More on the state-of-the-art of symmetric algorithms will be said in the contribution Fundamentals of Symmetric Cryptography of this volume. The second family of schemes, asymmetric or public-key algorithms, are very different. They are based on hard number theoretical problems and involve complex mathematical computations with very long numbers, commonly in the range of 160–4048 bits, depending on the algorithm and security level. Their advantage, however, is that they offer advanced functions such as digital signatures and key distribution over unsecure channels. For common automotive applications such as secure flashing, public-key algorithms are often preferred. The problem here is the computational requirement of public key schemes. Embedded processors in the automotive domain are often only equipped with 8-bit and 16-bit processors clocked at moderate frequencies of, say, below 10 MHz Running computationally expensive public-key algorithms on such processors can result in unacceptably long execution times, for instance several seconds for the generation of a digital signature. For this reason, it is very important that a smart parameter choice together with the latest implementation techniques are being employed.

2.3.3 Physical Security: Side Channel Attacks and Reverse Engineering

A central tool for providing security are cryptographic algorithms. Both symmetric and asymmetric algorithms are based on the fact that the protected device (e.g., tachograph, an ECU, or an infotainment device) is equipped with a secret cryptographic key. “Secret” means in this context also that it can not be read out by an attacker. If an attacker obtains knowledge of the key, the device can usually be manipulated and/or cloned. Many of the potential attackers – which includes in particular the owner and maintenance personnel – have physical access to the device. One family of attacks which attempt to recover the key from the device are side channel attacks, which were first proposed in the open literature in 1996. Side channel attacks observe the power consumption, the timing behaviour or the electromagnetic radiation of an embedded device. These signals are recorded while the cryptographic algorithm with the secret key is executed. The attacker then tries to extract the key by means of signal processing techniques. Side channel attacks are a serious threat in the real world unless special countermeasures have been implemented.

A related family of attacks are fault injection attacks, sometimes referred to as active side channel attacks. Fault injection attacks force the device to malfunction, for instance by spikes in the power supply, through overclocking, or through overheating of the embedded device. The goal is often to create an incorrect output of the cryptographic algorithm, which leaks information about the key used. Quite different from side channel and fault injection attacks are reverse engineering attacks. The goal here is to read the cryptographic keys directly from the RAM, EEPROM, FlashROM, or ROM of the embedded device.

Unlike classical reverse engineering of code it is in this context sometimes sufficient to recover a single cryptographic key for a successful attack, which is often only 16 bytes long or less. Of course, there is tamper-resistant memory available but it is for automotive systems often not available for cost or legacy reasons.

2.3.4 Digital Rights Management (DRM)

DRM has become a very important technology for applications such as audio and film distribution over the Internet. DRM systems can enforce rules such as the time period for which access to a music file is granted or to which device a digital movie is allowed to be copied. It is perhaps a bit surprising that DRM should become important for vehicles as well. However, as soon as digital data used for car applications represent financial values, e.g., flash software, digital location-based services or entertainment content, DRM will be the technology that enforces the envisioned use of the data. DRM technologies are required

to prevent the customer from unauthorized copying or an unauthorized extension of the usage period of the content. In order to realize a proper DRM platform in a vehicle, we need trusted computing functions, which in turn are based on physical secure components such as secure memory, true random number generators and cryptographic algorithms.

2.4 Automotive Applications and IT Security

As sketched above, embedded IT security will be a crucial part of many future automotive features. IT security offers a wide variety of functions that can improve products. In the context of embedded automotive systems, the advantages of strong IT security can be summarized in two main categories.

- Increased reliability: Innovative IT applications must be protected against targeted manipulations. For instance, manipulation of an otherwise robust electronic engine control system may result in an unreliable engine (e.g., shortened engine life span). Another example is a highly fault tolerant telematic system. Manipulation of messages to and from the car, however, may result in a very unreliable system. IT security can prevent those and many other abuses. It is important to stress that from this viewpoint security can also be interpreted as being part of reliability.
- New business models: Cars equipped with state-of-the-art IT technology will open up opportunities for a multitude of new business models. In times where international competition is putting increasing pressure on car manufacturers, novel IT-based business models are tempting options. Examples include fee-based software updates, navigation data, location-based services and multimedia content. It is of crucial importance to stress that virtually all such business models rely heavily on strong IT security.

Admittedly, this is a rather broad classification. In the following, we will list more concrete application domains within cars that rely heavily on IT security.

Software Updates. In the last few years, the topic of software updates of ECUs (electronic control units) has gained crucial importance. The reasons why ECUs that can be updated are attractive are multitude, and a few important ones are given in the following: many software bugs are only found after shipment of the car; cars can be configured differently for different customers, reducing the variety of cars that have to be manufactured; features can be activated based on a pricing policy. Unfortunately, unauthorized software updates can pose an equal number of problems for manufactures and, to a lesser degree, for owners. For instance, it is obviously quite attractive to activate certain car features (e.g., a stronger engine or comfort functions) without paying the associated fees. This cannot only result in financial losses due to missed business transactions, but also in an increase of warranty cases. In order

to enable software update in a manner controlled by the manufacturer, one needs embedded security technologies such as digital signature, tamper-resistant hardware and encryption.

Theft Prevention. The electronic immobilizer is possibly the oldest incarnation of IT security mechanisms in the automotive. The latest generation of immobilizer has been quite a success, with the damage from car thefts reduced by 50% over the last decade or so. This proves that classical IT security (here: strong cryptographic identification) can have an immediate benefit in today's world. It is very tempting to generalize immobilizer solutions to car components. By using strong cryptography, one could identify valuable or crucial components and, thus, protect against illegal exchange of components, and enforce the usage of original manufacturer spare parts. The techniques required from embedded security are identification protocols and tamper resistance.

Business Models for Infotainment Content. It seems almost certain that the majority of future cars will be equipped with powerful infotainment devices in the dashboard. The functionality of the infotainment systems will be a fusion of

- Home entertainment (e.g., radio, music and video for the rear seats),
- Telecommunication (e.g., cell phone and email function),
- Car-specific information systems (e.g., navigation data, smart traffic routing, emergency calls).

There will be opportunities for content providers, car manufacturers and possibly for other parties to create innovative business models around the digital content mentioned above. There are already systems in use today, which provide navigation data on a time-limited basis. Another indication for the opportunities ahead is that after the 2004 more than 50% of all mini vans sold in the USA were equipped with rear seat video screens. Adding new business models, for instance fee-based video downloads at hot spots, seems not entirely unrealistic.

The topic of security plays a crucial role here. It should be noted that there is a built-in incentive for users (i.e., business partners!) to behave dishonestly, e.g., by copying content in an unauthorized manner or by using content beyond the paid-for period. This situation is similar to the hotly debated topic of content distribution via the Internet. In order to prevent abuse and, thus, to enable new business models, strong embedded security technologies are needed. First, communication security (i.e., protection of the link between car and the environment) is needed in order to transport valuable digital content to the customer. Second, digital rights management (DRM) technologies are required to prevent the customer from unauthorized copying or an unauthorized extension of the usage period of the content. Third, privacy-preventing technology will be required in order to limit the collection of customer data. Without the latter measure, user acceptance of new technologies can quickly diminish. Finally, secure hardware components are required in order to prevent manipulation of the IT security mechanisms and demolishing the business model.

Personalization of Cars. Car functions that can be updated open up a wide variety of new possibilities such as personalization of car features, from your favourite radio station and seat position to your favourite suspension setting. There is a multitude of options for realizing a recognition of the driver.

One class of approaches is token-based, e.g., through car keys, smart cards, or cell phones. Other approaches make use of biometrics, e.g., fingerprint recognition.

Another class simply requires active user input in order to communicate the person's identity to the vehicle. Again, we will need security technologies here in order to prevent abuse. Technologies needed included identification techniques, biometrics and tamper resistance hardware.

Access Control for Car Data. Already today, many cars are equipped with event data recorders. This can be as simple as tachograph data, or more advanced systems, which record a wide variety of information about the car subsystems and driving behaviour. Currently, such data can usually only be accessed via diagnosis interfaces which have to be attached physically to the car. Furthermore, today many vehicles can be equipped with wireless interfaces such as Bluetooth or GSM. It becomes crucial now to tightly control access to both technical data about the car and stored information about driving behaviour. Relevant security functions are authentication and identification protocols and communication security.

Anonymity. Cars filled with IT systems offer several possibilities for violating driver's privacy rights. The above-mentioned recording of driving behaviour is one example. Navigation data used or requests for other location based services (e.g., the purchase of certain navigation data, requests for the nearest gas station or requests for the nearest restaurant) is another example.

It is also imaginable that even traffic violations, e.g., driving beyond the allowed speed limit, are recorded. These can all be serious threats in an information society and it will be crucial to prevent abuses by incorporating technologies such as access control and anonymization.

Legal Obligations. Already today, there are several regulations that dictate the inclusion of IT security functions in cars. An example is Toll Collect, the German road toll system or the European tachograph.

In the future, there will be more applications, which require IT, security due to legal regulations. Possible examples include emergency call systems, immobilizers or other theft control measures, and event data recorders.

We do not claim that the listing above is complete. However, we can believe that embedded security is already an important technology for a host of diverse car functions, and its impact will increase in the future. In summary, it can be claimed that IT security will play the role of an enabling technology for numerous future car applications.

After this brief introduction, the motivations that stay behind the security in the automotive environment should be clearer.

Slowly, the various automotive companies, started to insert the "security" in their product, but the real push came from an article, Experimental Security Analysis of a Modern Automobile. To evaluate the importance of this article we will see briefly its contents in the next chapter.

Chapter 3

Automotive Security

3.1 Experimental Security Analysis of a Modern Automobile

Their studies focus on what an attacker could do to a car if she was able to maliciously communicate on the car's internal network thus they haven't done a full analysis of the modern automobile's attack surface.

However, we can distinguish roughly two type of vectors by which one might gain access to a car's internal networks:

Car's internal networks.

The first is physical access. Someone—such as a mechanic, a valet, a person who rents a car, an ex-friend, a disgruntled family member, or the car owner—can, with even momentary access to the vehicle, insert a malicious component into a car's internal network via the ubiquitous OBD-II port (typically under the dash). The attacker may leave the malicious component permanently attached to the car's internal network or, he may use a brief period of connectivity to embed the malware within the car's existing components and then disconnect. A similar entry point is presented by counterfeit or malicious components entering the vehicle parts supply chain—either before the vehicle is sent to the dealer, or with a car owner's purchase of an aftermarket third-party component (such as a counterfeit FM radio).

The other vector is via the numerous wireless interfaces implemented in the modern automobile. In their car, we can identify no fewer than five kinds of digital radio interfaces accepting outside input, some over only a short range and others over indefinite distance.

Their experimental analyses focus on two 2009 automobiles of the same make and model, and they selected those vehicles because they contained both a large number of electronically controlled components (necessitated by complex safety features such as anti-lock brakes and stability control) and a sophisticated telematics system. They purchased two vehicles to allow differential testing and to validate that their results were not tied to one individual vehicle. They also purchased individual replacement ECUs via third-party dealers to allow additional testing.

Component	Functionality	Low-Speed Comm. Bus	High-Speed Comm. Bus
ECM	*Engine Control Module* Controls the engine using information from sensors to determine the amount of fuel, ignition timing, and other engine parameters.		✓
EBCM	*Electronic Brake Control Module* Controls the Antilock Brake System (ABS) pump motor and valves, preventing brakes from locking up and skidding by regulating hydraulic pressure.		✓
TCM	*Transmission Control Module* Controls electronic transmission using data from sensors and from the ECM to determine when and how to change gears.		✓
BCM	*Body Control Module* Controls various vehicle functions, provides information to occupants, and acts as a firewall between the two subnets.	✓	✓
Telematics	*Telematics Module* Enables remote data communication with the vehicle via cellular link.	✓	✓
RCDLR	*Remote Control Door Lock Receiver* Receives the signal from the car's key fob to lock/unlock the doors and the trunk. It also receives data wirelessly from the Tire Pressure Monitoring System sensors.	✓	
HVAC	*Heating, Ventilation, Air Conditioning* Controls cabin environment.	✓	
SDM	*Inflatable Restraint Sensing and Diagnostic Module* Controls airbags and seat belt pretensioners.	✓	
IPC/DIC	*Instrument Panel Cluster/Driver Information Center* Displays information to the driver about speed, fuel level, and various alerts about the car's status.	✓	
Radio	*Radio* In addition to regular radio functions, funnels and generates most of the in-cabin sounds (beeps, buzzes, chimes).	✓	
TDM	*Theft Deterrent Module* Prevents vehicle from starting without a legitimate key.	✓	

Table 3.1 lists some of the most important ECUs in their car

How we can see from the table 3.1the various ECU have a different type of bus and two separate physical layers—a high-speed bus which is differentially-signalled and primarily used by powertrain systems and a low-speed bus (SAE J2411) using a single wire and supporting less-demanding components. When necessary, a gateway bridge can route selected data between the two buses.

Before experimentally evaluating the security of individual car components, they assessed the security properties of the communication bus inside the car.

There are a variety of protocols that can be implemented on the vehicle bus, but for example, starting in 2008 all cars sold in the U.S. are required to implement the Controller Area Network (CAN) bus (ISO 11898) for diagnostics. As a result, CAN—roughly speaking, a link-layer data protocol—has become the dominant communication network for in-car networks (e.g., used by BMW, Ford, GM, Honda, and Volkswagen).

The underlying CAN protocol (and the others link-layer data protocol like Ethernet) has a number of inherent weaknesses that are common to any implementation. Key among these:

Broadcast Nature. Since CAN packets are both physically and logically broadcast to all nodes, a malicious component on the network can easily snoop on all communications or send packets to any other node on the network. To facilitate their experimental analysis, they wrote CARSHARK, a custom CAN bus analyser and packet injection tool. CARSHARK allowed them to observe and reverse-engineer packets, as well as to inject new packets to induce various actions.

Fragility to DoS. The CAN protocol is extremely vulnerable to denial-of-service attacks. In addition to simple packet flooding attacks, CAN's priority-based arbitration scheme allows a node to assert a "dominant" state on the bus indefinitely and cause all other CAN nodes to back off. While most controllers have logic to avoid accidentally breaking, the network this way, adversarially controlled hardware would not need to exercise such precautions.

No Authenticator Fields. CAN packets contain no authenticator fields—or even any source identifier fields— meaning that any component can indistinguishably send a packet to any other component. This means that any single compromised component can be used to control all of the other components on that bus, provided those components themselves do not implement defences.

Weak Access Control. The protocol standards for their car specify a challenge-response sequence to protect ECUs against certain actions without authorization. A given ECU may participate in zero, one, or two challenge-response pairs:

- Reflashing and memory protection. One challenge response pair restricts access to reflashing the ECU and reading out sensitive memory. By design, a service shop might authenticate with this challenge-response pair in order to upgrade the firmware on an ECU.
- Tester capabilities. Modern automobiles are complex and thus diagnosing their problems requires significant support. Thus, a major use of the CAN bus is in providing diagnostic access to service technicians. In particular, external test equipment (the "tester") must be able to interrogate the internal state of the car's components and, at times, manipulate this state as well.

Now we see the various type of attacks that they done at the cars under test.

Disabling Communications. For example, the standard states that, ECUs should reject the "disable CAN communications" command when it is unsafe to accept and act on it, such as when a car is moving. However, they experimentally verified that this is not actually the case in their car: they were able to disable communications to and from all the ECUs in Table 3.1 even with the car's wheels moving at speed on jack stands and while driving on the closed road course.

Reflashing ECUs While Driving. The standard also states that ECUs should reject reflashing events if they deem them unsafe. In fact, it states: “The engine control module should reject a request to initiate a programming event if the engine were running.” However, they experimentally verified that they could place the Engine Control Module (ECM) and Transmission Control Module (TCM) into reflashing mode when their car was at speed on jack stands. When the ECM enters this mode, the engine stops running. They also verified that they could place the ECM into reflashing mode while driving on the closed course.

Noncompliant Access Control: Firmware and Memory. The standard states that ECUs with emissions, anti-theft, or safety functionality must be protected by a challenge response access control protocol even disregarding the weakness of this protocol, they found it was implemented less broadly than they would have expected. They verified experimentally that they can load their own code onto their car’s telematics unit without authenticating, and the Device Control keys for the ECM and TCM just by authenticating with the reflashing key. They were also able to extract the telematics units’ entire memory, including their keys, without authentication.

Noncompliant Access Control: Device Overrides. Recall that the Device Control service is used to override the state of components. However, ECUs are expected to reject unsafe Device Control override requests, such as releasing the brakes when the car is in motion. Some of these unsafe overrides are needed for testing during the manufacturing process, so those can be enabled by authenticating with the Device Control key. However, they found during the experiments that certain unsafe device control operations succeeded without authenticating.

Imperfect Network Segregation. The standard implicitly defines the high-speed network as more trusted than the low-speed network. This difference is likely due to the fact that the high-speed network includes the real-time safety critical components (e.g., engine, brakes), while the low speed network commonly includes components less critical to safety, like the radio and the HVAC system.

The standard states that gateways between the two networks must only be re-programmable from the high-speed network, presumably to prevent a low-speed device from compromising a gateway to attack the high-speed network. In their car, there are two ECUs, which are on both buses and can potentially bridge signals: the Body Controller Module (BCM) and the telematics unit. While the telematics unit is not technically a gateway, it connects to both networks and can only be reprogrammed (against the spirit of the standard) from the low-speed network, allowing a low speed device to attack the high-speed network through the telematics unit.

They verified that they could bridge these networks by uploading code to the telematics unit from the low-speed network that, in turn, sent packets on the high-speed network.

There are three major approaches to bring the attacks just seen:

Packet Sniffing and Targeted Probing. To begin, they used CARSHARK to observe traffic on the CAN buses in order to determine how ECUs communicate with each other. This also revealed to them which packets were sent as they activated various components (such as turning on the headlights). Through a combination of replay and informed probing, they were able to discover how to control the radio, the Instrument Panel Cluster (IPC), and a number of the Body Control Module (BCM) functions. This approach worked well for packets that come up during normal operation, but was less useful in mapping the interface to safety-critical powertrain components.

Fuzzing. Much to their surprise, significant attacks do not require a complete understanding or reverse engineering of even a single component of the car. In fact, because the range of valid CAN packets is rather small, significant damage can be done by simple fuzzing of packets (i.e., iterative testing of random or partially random packets). Indeed, for attackers seeking indiscriminate disruption, fuzzing is an effective attack by itself. (Unlike traditional uses of fuzzing, they use fuzzing to aid in the reverse engineering of functionality.)

Reverse Engineering. For a small subset of ECUs (notably the telematics unit, for which they obtained multiple instances via Internet-based used parts resellers) they dumped The ECU's code via the CAN Read Memory service and used a third-party debugger (IDA Pro) to explicitly understand how certain hardware features were controlled.

This approach is essential for attacks that require new functionality to be added (e.g., bridging low and high-speed buses) rather than simply manipulating existing software capabilities. Afterwards there are the results of their experiments with controlling critical components of the car.

All initial experiments were done with the car stationary, in many cases immobilized on jack stands for safety. Some of their results are summarized in the following tables. The tables indicate the packet that was sent to the corresponding module, the resulting action, and four additional pieces of information:

1. Can the result of this packet be overridden manually, such as by pulling the physical door unlock knob, pushing on the brakes, or some other action? A No in this column means that they have found no way to manually override the result.
2. Does this packet have the same effect when the car is at speed? For this column, "at speed" means when the car was up on jack stands but the throttle was applied to bring the wheel speed to 40 MPH.
3. Does the module in question need to be unlocked with its Device Control key before these packets can elicit results?

4. Additional column reflects their experiments during a live road test.

Packet	Result	Manual Override	At Speed	Need to Unlock	Tested on Runway
07 AE ... 1F 87	Continuously Activates Lock Relay	Yes	Yes	No	✓
07 AE ... C1 A8	Windshield Wipers On Continuously	No	Yes	No	✓
07 AE ... 77 09	Pops Trunk	No	Yes	No	✓
07 AE ... 80 1B	Releases Shift Lock Solenoid	No	Yes	No	
07 AE ... D8 7D	Unlocks All Doors	Yes	Yes	No	
07 AE ... 9A F2	Permanently Activates Horn	No	Yes	No	✓
07 AE ... CE 26	Disables Headlights in Auto Light Control	Yes	Yes	No	✓
07 AE ... 34 5F	All Auxiliary Lights Off	No	Yes	No	
07 AE ... F9 46	Disables Window and Key Lock Relays	No	Yes	No	
07 AE ... F8 2C	Windshield Fluid Shoots Continuously	No	Yes	No	✓
07 AE ... 15 A2	Controls Horn Frequency	No	Yes	No	
07 AE ... 15 A2	Controls Dome Light Brightness	No	Yes	No	
07 AE ... 22 7A	Controls Instrument Brightness	No	Yes	No	
07 AE ... 00 00	All Brake/Auxiliary Lights Off	No	Yes	No	✓
07 AE ... 1D 1D	Forces Wipers Off and Shoots Windshield Fluid Continuously	Yes†	Yes	No	✓

Packet	Result	Manual Override	At Speed	Need to Unlock	Tested on Runway
07 AE ... E5 EA	Initiate Crankshaft Re-learn; Disturb Timing	Yes	Yes	Yes	
07 AE ... CE 32	Temporary RPM Increase	No	Yes	Yes	✓
07 AE ... 5E BD	Disable Cylinders, Power Steering/Brakes	Yes	Yes	Yes	
07 AE ... 95 DC	Kill Engine, Cause Knocking on Restart	Yes	Yes	Yes	✓
07 AE ... 8D C8	Grind Starter	No	Yes	Yes	
07 AE ... 00 00	Increase Idle RPM	No	Yes	Yes	✓

Packet	Result	Manual Override	At Speed	Need to Unlock†	Tested on Runway
07 AE ... 25 2B	Engages Front Left Brake	No	Yes	Yes	✓
07 AE ... 20 88	Engages Front Right Brake/Unlocks Front Left	No	Yes	Yes	✓
07 AE ... 86 07	Unevenly Engages Right Brakes	No	Yes	Yes	✓
07 AE ... FF FF	Releases Brakes, Prevents Braking	No	Yes	Yes	✓

Destination ECU	Packet	Result	Manual Override	At Speed	Tested on Runway
IPC	00 00 ... 00 00	Falsify Speedometer Reading	No	Yes	✓
Radio	04 00 ... 00 00	Increase Radio Volume	No	Yes	
Radio	63 01 ... 39 00	Change Radio Display	No	Yes	
IPC	00 02 ... 00 00 27 01 ... 65 00	Change DIC Display	No	Yes	
BCM	04 03	Unlock Car†	Yes	Yes	
BCM	04 01	Lock Car†	Yes	Yes	
BCM	04 0B	Remote Start Car†	No	No	
BCM	04 0E	Car Alarm Honk†	No	No	
Radio	83 32 ... 00 00	Ticking Sound	No	Yes	
ECM	AE 0E ... 00 7E	Kill Engine	No	Yes	

In the article, they deepen these attacks, but for our scope is enough understand the problem of the security and as a starting point, we can take the conclusions of this article.

We can summarize the conclusions in the following eight points

Extent of Damage. Past works discuss potential risks to cyber-physical vehicles and thus we knew that adversaries might be able to do damage by attacking the components within cars. The authors of the

article did not, however, anticipate that we would be able to directly manipulate safety critical ECUs (indeed, all ECUs that we tested) or that we would be allowed to create unsafe conditions of such magnitude.

Ease of Attack. In starting this project, they expected to spend significant effort reverse engineering, with non-trivial effort to identify and exploit each subtle vulnerability. However, they found existing automotive systems—at least those they tested—to be tremendously fragile. Indeed, their simple fuzzing infrastructure was very effective and to their surprise, a large fraction of the random packets they sent resulted in changes to the state of their car. Based on this experience, they believe that a fuzzer itself is likely be a universal attack for disrupting arbitrary automobiles (similar to how the "crashme" program that fuzzed system calls was effective in crashing operating systems before the syscall interface was hardened).

Unenforced Access Controls. While they believe that standard access controls are weak, they were surprised at the extent to which the controls that did exist were frequently unused. For example, the firmware on an ECU controls all of its critical functionality and thus the standard for their car's CAN protocol variant describes methods for ECUs to protect against unauthorized firmware updates. They were therefore surprised that they could load firmware onto some key ECUs, like their telematics unit (a critical ECU) and their Remote Control Door Lock Receiver (RCDLR), without any such authentication. Similarly, the protocol standard also makes an earnest attempt to restrict access to Device Control diagnostic capabilities. They were therefore also surprised to find that critical ECUs in their car would respond to Device Control packets without authentication first.

Attack Amplification. They found multiple opportunities for attackers to amplify their capabilities—either in reach or in stealth. For example, while the designated gateway node between the car's low-speed and high-speed networks (the BCM) should not expose any interface that would let a low-speed node compromise the high-speed network, they found that they could maliciously bridge these networks through a compromised telematics unit. Thus, the compromise of any ECU becomes sufficient to manipulate safety-critical components such as the EBCM. As more and more components integrate into vehicles, it may become increasingly difficult to properly secure all bridging points.

Diagnostic and Reflashing Services. Many of the vulnerabilities they discovered were made possible by weak or unenforced protections of the diagnostic and reflashing services. Because these services are never intended for use during normal operation of the vehicle, it is tempting to address these issues by completely locking down such capabilities after the car leaves manufacturing.

Aftermarket Components. Even with diagnostic and reflashing services secured, packets that appear on the vehicle bus during normal operation can still be spoofed by third party ECUs connected to the bus. Today a modern automobile leaves the factory containing multiple third party ECUs, and owners often

add aftermarket components (like radios or alarms) to their car's buses. This creates a tension that, in the extreme, manifests itself as the need to either trust all third-party components, or to lock down a car's network so that no third-party components—whether adversarial or benign—can influence the state of the car.

One potential intermediate (and backwards compatible) solution they envision, is to allow owners to connect an external filtering device between an untrusted component (such as a radio) and the vehicle bus to function as a trusted mediator, ensuring that the component sends and receives only approved packets.

Detection versus Prevention. More broadly, certain considerations unique to cyber-physical vehicles raise the possibility of security via detection and correction of anomalies, rather than prevention and locking down of capabilities. For example, the operational and economic realities of automotive design and manufacturing are stringent. Manufacturers must swiftly integrate parts from different suppliers (changing as needed to second and third source suppliers) in order to quickly reach market and at low cost. Competitive pressures drive vendors to reuse designs and thus engenders significant heterogeneity. It is common that each ECU may use a different processor and/or software architecture and some cars may even use different communications architectures—one grafted onto the other to integrate a vendor assembly and bring the car to market in time. Today the challenges of integration have become enormous and manufacturers seek to reduce these overheads at all costs—a natural obstacle for instituting strict security policies. In addition, many of an automobile's functions are safety critical, and introducing additional delay into the processing of, say, brake commands, may be unsafe. These considerations raise the possibility of exploring the trade-off between preventing and correcting malicious actions: if rigorous prevention is too expensive, perhaps a quick reversal is sufficient for certain classes of vulnerabilities.

Several questions come with this approach: Can anomalous behaviour be detected early enough, before any dangerous packets are sent? Can a fail-safe mode or last safe state be identified and safely reverted to? It is also unclear what constitutes abnormal behaviour on the bus in the first place, as attacks can be staged entirely with packets that also appear during normal vehicle operation.

Toward Security. These are just a few of many potential defensive directions and associated tensions. There are deep-rooted tussles surrounding the security of cyber physical vehicles, and it is not yet clear what the "right" solution for security is or even if a single "right" solution exists. More likely, there is a spectrum of solutions that each trade off critical values (like security vs. support for independent auto shops). Thus, they argue that the future research agenda for securing cyber-physical vehicles is not merely to consider the necessary technical mechanisms, but to also inform these designs by what is feasible practically and compatible with the interests of a broader set of stakeholders. This work serves

as a critical piece in the puzzle, providing the first experimentally guided study into the real security risks with a modern automobile.

To conclude with this article we can say that its purpose is to awareness the automotive manufactures that the 'car system' presents some flaws about the security, which may bring damage to various levels, without going into detail of the attack channel type.The same editors of this article, to fill the previous gap, wrote another significant article about the attack channel types: Comprehensive Experimental Analyses of Automotive Attack Surfaces.

3.2 Comprehensive Experimental Analyses of Automotive Attack Surfaces

Although the purpose of this thesis is not to analyze issues relating to how the attack is carried, it is important view some principal aspects of this article, to understand the importance of the security in the automotive environment.

The previous work shown how to attack a car by the On-Board Diagnostics (OBD-II) port. This is a physical access and not ever the malicious user can have it. Thus, the threat do not seems much dangerous, because the malicious user need to dismount some part of the car and connect itself to On-Board Diagnostics port. Indeed, there are many type of channel that can be used to a malicious user to create a breach in the system and in this article, as we will see; they explored all the available channels.

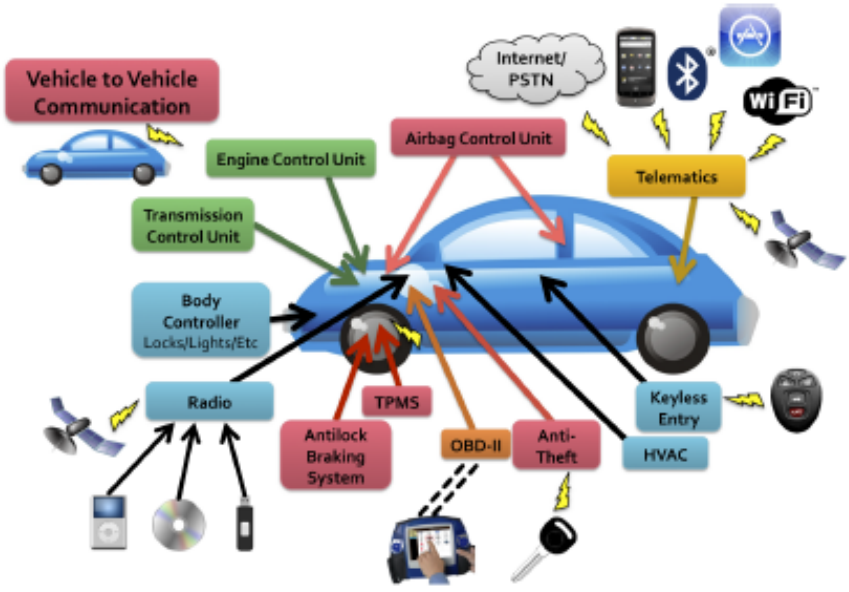


Figure 3.1: Digital I/O channels appearing on a modern car. Colours indicate rough grouping of ECUs by function.

They divide the attack channels type in three categories

1. Indirect physical access

Modern automobiles provide several physical interfaces that either directly or indirectly access the car's internal networks. They consider the full physical attack surface here, under the constraint that the adversary may not directly access these physical interfaces herself but must instead work through some intermediary.

OBD-II. The most significant automotive interface is the OBD-II port, federally mandated in the U.S., which typically provides direct access to the automobile's key CAN buses and can provide sufficient access to compromise the full range of automotive systems.

While their threat model forbids the adversary from direct access herself, we note that the OBD-II port is commonly accessed by service personnel during routine maintenance for both diagnostics and ECU programming. Historically this access is achieved using dedicated handheld "scan" tools such as Ford's NGS, Nissan's Consult II and Toyota's Diagnostic Tester, which are themselves, programmed via Windows-based personal computers. For modern vehicles, most manufacturers have adopted an approach that is PC-centric. Under this model, a laptop computer interfaces with a "Pass Thru" device (typically directly via USB or Wi-Fi) that in turn is plugged into the car's OBD-II port. Software on the laptop computer can then interrogate or program the car's ECUs via this device (typically using the standard SAE J2534 API). Examples of such tools include Toyota's TIS, Ford's VCM, Nissan's Consult 3 and Honda's HDS among others.

In both situations, Windows-based computers directly or indirectly control the data to be sent to the automobile. Thus, if an adversary were able to compromise such systems at the dealership she could amplify this access to attack any cars under service. Such laptop computers are typically Internet-connected (indeed, this is a requirement for some manufacturers' systems), so traditional means of personal computer compromise could be employed. Further afield, electric vehicles may also communicate with external chargers via the charging cable. An adversary able to compromise the external charging infrastructure may thus be able to leverage that access to subsequently attack any connected automobile.

Entertainment: Disc, USB and iPod.

The other important class of physical interfaces are focused on entertainment systems. Virtually all automobiles shipped today provide a CD player able to interpret a wide variety of audio formats (raw "Red Book" audio, MP3, WMA, and so on). Similarly, vehicle manufacturers also provide some kind of external digital multimedia port (typically either a USB port or an iPod/iPhone docking port) for allowing users to control their car's media system using their personal audio player or phone. Some

manufacturers have widened this interface further; BMW and Mini recently added to their cars, their support for “iPod Out,” a scheme whereby Apple media devices will be able to control the display on the car’s console.

Consequently, an adversary might deliver malicious input by encoding it onto a CD or as a song file and using social engineering to convince the user to play it. Alternatively, she might compromise the user’s phone or iPod out of band and install software onto it that attacks the car’s media system when connected.

Taking over a CD player alone is a limited threat; but, for a variety of reasons, automotive media systems are not standalone devices. Indeed, many such systems are now CAN bus interconnected, either to directly interface with other automotive systems (e.g., to support chimes, certain hands-free features, or to display messages on the console) or simply to support a common maintenance path for updating all ECU firmware. Thus, counterintuitively, a compromised CD player can offer an effective vector for attacking other automotive components.

2. Short-range wireless access

Indirect physical access has a range of drawbacks including its operational complexity, challenges in precise targeting, and the inability to control the time of compromise. Here we weaken the operational requirements on the attacker and consider the attack surface for automotive wireless interfaces that operate over short ranges. These include Bluetooth, Remote Keyless Entry, RFIDs, Tire Pressure Monitoring Systems, Wi-Fi, and Dedicated Short- Range Communications. For this portion of the attack surface, we assume that the adversary is able to place a wireless transmitter in proximity to the car’s receiver (between 5 and 300 meters depending on the channel).

Bluetooth.

Bluetooth has become the de facto standard for supporting hands-free calling in automobiles and is standard in mainstream vehicles sold by all major automobile manufacturers. While the lowest level of the Bluetooth protocol is typically implemented in hardware, the management and services component of the Bluetooth stack is often implemented in software. In normal usage, the Class 2 devices used in automotive implementations have a range of 10 meters, but others have demonstrated that this range can be extended through amplifiers and directional antennas.

Remote Keyless Entry.

Today, all but entry-level automobiles shipped in the U.S. use RF-based remote keyless entry (RKE) systems to remotely open doors, activate alarms, flash lights and, in some cases, start the ignition (all typically using digital signals encoded over 315 MHz in the U.S. and 433 MHz in Europe).

Tire pressure.

In the U.S., all 2007 model year and newer cars are required to support a Tire Pressure Monitoring System (TPMS) to alert drivers about under or over inflated tires. The most common form of such systems, so called "Direct TPMS," uses rotating sensors that transmit digital telemetry (frequently in similar bands as RKEs).

RFID car keys. RFID-based vehicle immobilizers are now nearly ubiquitous in modern automobiles and are mandatory in many countries throughout the world. These systems embed an RFID tag in a key or key fob and a reader in or near the car's steering column. These systems can prevent the car from operating unless the correct key (as verified by the presence of the correct RFID tag) is present.

Emerging short-range channels.

A number of manufacturers have started to discuss providing 802.11 Wi-Fi access in their automobiles, typically to provide "hotspot" Internet access via bridging to a cellular 3G data link. In particular, Ford offers this capability in the 2012 Ford Focus. (Several 2011 models also provided WiFi receivers, but we understand they were used primarily for assembly line programming.)

Finally, while not currently deployed, an emerging wireless channel is defined in the Dedicated Short-Range Communications (DSRC) standard, which is being incorporated into proposed standards for Cooperative Collision Warning/Avoidance and Cooperative Cruise Control. Representative programs in the U.S. include the Department of Transportation's Cooperative Intersection Collision Avoidance Systems (CICAS-V) and the Vehicle Safety Communications Consortium's VSC-A project. In such systems, forward vehicles communicate digitally to trailing cars to inform them of sudden changes in acceleration to support improved collision avoidance and harm reduction.

Summary. For all of these channels, if a vulnerability exists in the ECU software responsible for parsing channel messages, then an adversary may compromise the ECU (and by extension the entire vehicle) simply by transmitting a malicious input within the automobile's vicinity.

3. Long-range wireless

Finally, automobiles increasingly include long distance (greater than 1 km) digital access channels as well. These tend to fall into two categories: broadcast channels and addressable channels.

Broadcast channels. Broadcast channels are channels that are not specifically directed towards a given automobile but can be "tuned into" by receivers on demand.

In addition to being part of the external attack surface, long-range broadcast mediums can be appealing as control channels (i.e., for triggering attacks) because they are difficult to attribute, can command multiple receivers at once, and do not require attackers to obtain precise addressing for their victims.

The modern automobile includes a plethora of broadcast receivers for long-range signals: Global Positioning System (GPS), Satellite Radio (e.g., SiriusXM receivers common to late-model vehicles from Honda/Accura, GM, Toyota, Saab, Ford, Kia, BMW and Audi), Digital Radio (including the U.S. HD Radio system, standard on 2011 Ford and Volvo models, and Europe's DAB offered in Ford, Audi, Mercedes, Volvo and Toyota among others), and the Radio Data System (RDS) and Traffic Message Channel (TMC) signals transmitted as digital subcarriers on existing FM-bands.

The range of such signals depends on transmitter power, modulation, terrain, and interference. As an example, a 5W RDS transmitter can be expected to deliver its 1.2 kbps signal reliably over distances upto 10 km. In general, these channels are implemented in an automobile's media system (radio, CD player, satellite receiver) which, as mentioned previously, frequently provides access via internal automotive networks to other key automotive ECUs.

Addressable channels.

Perhaps the most important part of the long-range wireless attack surface is that exposed by the remote telematics systems (e.g., Ford's Sync, GM's OnStar, Toyota's SafetyConnect, Lexus' Enform, BMW's BMW Assist, and Mercedes-Benz' mbrace) that provide continuous connectivity via cellular voice and data networks. These systems provide a broad range of features supporting safety (crash reporting), diagnostics

(early alert of mechanical issues), anti-theft (remote track and disable), and convenience (hands-free data access such as driving directions or weather). These cellular channels offer many advantages for attackers. They can be accessed over arbitrary distance (due to the wide coverage of cellular data infrastructure) in a largely anonymous fashion, typically have relatively high bandwidth, are two-way channels (supporting interactive control and data exfiltration), and are individually addressable.

For each category of access vector, they will explore one or two aspects of the attack surface deeply, identify concrete vulnerabilities, and explore and demonstrate practical attacks that are able to completely compromise their target automobile's systems without requiring direct physical access.

To be clear, for every vulnerability they demonstrate, they are able to obtain complete control over the vehicle's systems

Vulnerability Class	Channel	Implemented Capability	Visible to User	Scale	Full Control	Cost
Direct physical	OBD-II port	Plug attack hardware directly into car OBD-II port	Yes	Small	Yes	Low
Indirect physical	CD	CD-based firmware update	Yes	Small	Yes	Medium
	CD	Special song (WMA)	Yes*	Medium	Yes	Medium-High
	PassThru	WiFi or wired control connection to advertised PassThru devices	No	Small	Yes	Low
	PassThru	WiFi or wired shell injection	No	Viral	Yes	Low
Short-range wireless	Bluetooth	Buffer overflow with paired Android phone and Trojan app	No	Large	Yes	Low-Medium
	Bluetooth	Sniff MAC address, brute force PIN, buffer overflow	No	Small	Yes	Low-Medium
Long-range wireless	Cellular	Call car, authentication exploit, buffer overflow (using laptop)	No	Large	Yes	Medium-High
	Cellular	Call car, authentication exploit, buffer overflow (using iPod with exploit audio file, earphones, and a telephone)	No	Large	Yes	Medium-High

Table 3.2: Attack surface capabilities. The Visible to User column indicates whether the compromise process is visible to the user (the driver or the technician); we discuss social engineering attacks for navigating user detection in the body. For (*), users will perceive a malfunctioning CD. The Scale column captures the approximate scale of the attack, e.g., the CD firmware update attack is small-scale because it requires distributing a CD to each target car. The Full Control column indicates whether this exploit yields full control over the component's connected CAN bus (and, by transitivity, all the ECUs in the car). Finally, the Cost column captures the approximate effort to develop these attack capabilities.

Their experimental results give us the unique opportunity to reflect on the security and privacy risks with modern automobiles. They also synthesize concrete, pragmatic recommendations for future automotive security, as well as identify fundamental challenges. They disclosed their results to relevant industry and government stakeholders.

Chapter 4

State of art

4.1 Evolution of the standard solutions

This excursus is important to understand how the manufacturer of automotive component have responded to the problem of security in the automotive environment.

4.1.1 SHE

The first that tried to solve the security problem is the "HIS" workgroup that is composed by some vehicle manufacturers like Audi, BMW, Daimler, Porsche, and Volkswagen.

The HIS Working Group Security focuses on:

- Software Security
- Specification of common security classes
- Recommended Algorithms
- Public Key Infrastructures

Thus, they done some research about:

- Possible attacks
- Benefit for the attacker
- Damage for the user and so on

After those analyses they tried to give a mixed hardware/software solution based on a Secure Hardware Extension and the last review is the Functional Specification v1.1, rev 439 from Oct. 16th 2009.

In the portal of the workgroup is available all the documentations, for further information, but is all in German. However, in internet, we can find everything about this topic and afterwards we will see briefly their work.

Without enter in detail we can see that they provide:

- A cryptographic services based on AES-128.

- The CMAC (Cipher-based MAC) that is a block cipher-based message authentication code algorithm, and it may be used to provide assurance of the authenticity and, hence, the integrity of binary data.
- They also provide a secure storage of critical information like keys for the cryptography.
- Secure boot to ensure that at the start of the system in a particular ECU run the right software.

Summarizing, they focused on:

- Add a Secure Zone
- Prevent user access to security functions other than those given by logic

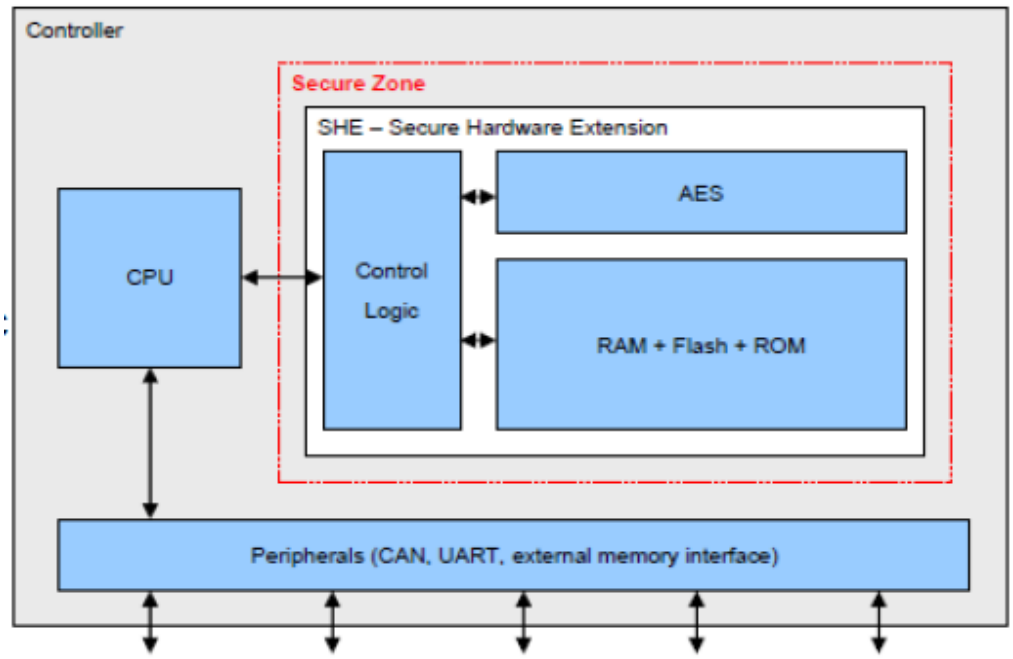


Figure 4.1: Simplified logical structure of SHE

SHE specifies Secure Zone components and algorithms

- Cryptography
  - En-/decryption unit
  - AES 128 algorithm
- ROM
  - Secret key storage SECRET_KEY
  - Unique key storage UID
- RAM
  - RAM key storage
  - PRNG key storage

- NV-Memory
    - Boot key & MAC storage
    - Master key, general purpose key storage

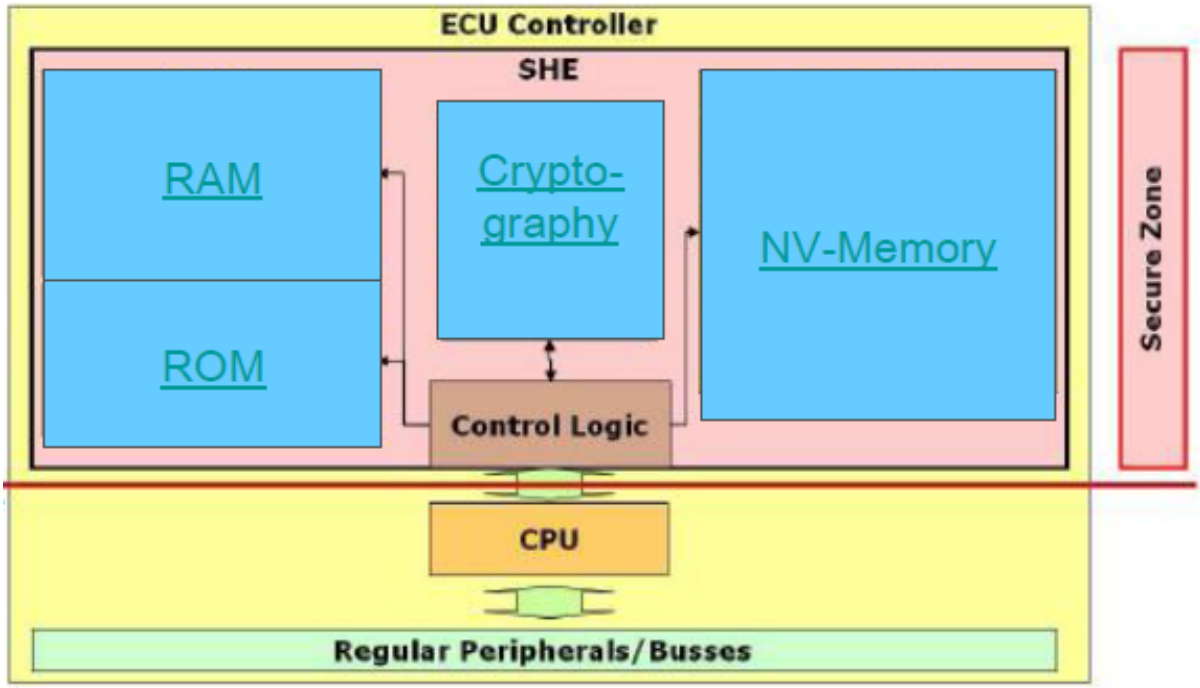
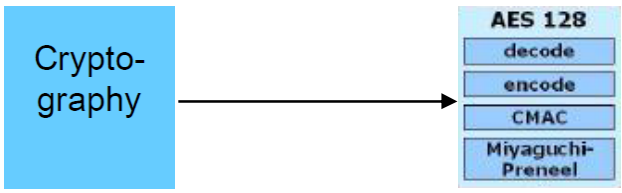


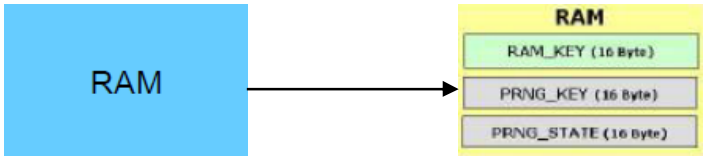
Figure 4.2: Detailed logical structure of SHE



Cryptography carries

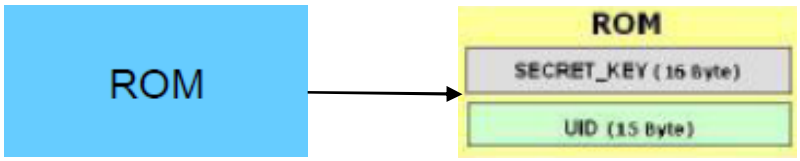
- Encryption unit
    - AES 128-based
- Decryption unit
    - AES 128-based
- CMAC
    - Cipher-based Message Authentication Code generator
- Miyaguchi-Preneel
    - One-way compression function; compressed data cannot be recovered

- Input requests 128-bit wide chunks of data stream
- Outputs Hash-values to en-/decoding unit



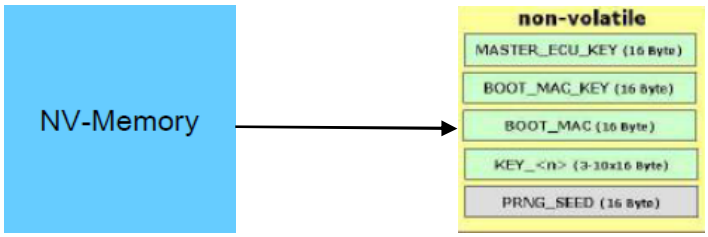
RAM carries

- RAM_KEY
  - Temporary key used for arbitrary operations
- PRNG_KEY
  - Key used by the Pseudo Random Number Generator
- PRNG_STATE
  - Keeps status o Pseudo Random Number Generator



ROM carries

- SECRET_KEY
  - Unique key
  - Used for im-/export of all other keys
  - Has to be created with true random number generator (off-chip TRNG) at production
- UID
  - Unique identifier
  - Authenticates MCU
- Both SECRET_KEY and UID have to be fixed at production time
  - 16 byte for SECRET_KEY and ≤15 byte for UID



NV-Memory carries

- MASTER_ECU_KEY
  - Set up by OEM (owner)
  - Enables change of other keys
- BOOT_MAC_KEY
  - Enables particular boot request and thus establishing secure boot
- BOOT_MAC
  - Authentication of boot code
- KEY_<n>
  - Dedicated key storage for arbitrary functions
  - 3 – 10 keys
- PRNG_SEED
  - Starting value for pseudo random number generator
- Irreversible Write Protection of keys in NV-memory
  - Any key in NV-memory area shall not be changeable throughout life time of the device once write-protection was applied by uses

The Secure hardware extension was the first attempt to create a standard solution to solve the security problem in the automotive environment, however the SHE, specializes the solution on a C2C communication. Moreover, the SHE does not provide all the security features.

For those and others reasons, many solutions available today on the market do not use this module

4.1.2Project EVITA

Project objectives

A significant further reduction of road traffic fatalities is expected from introducing vehicle-to-vehicle and vehicle-to-infrastructure (V2X) communication. Examples for electronic safety aids deployed in vehicles (e-safety applications) are local danger warnings, traffic light pre-emption, and electronic emergency brakes. While these functionalities inspire a new era of safety and efficiency in transportation, new security requirements need to be considered in order to prevent attacks on these systems. Attacks may originate outside or inside the vehicle, resulting for instance in the injection of illegitimate messages influencing the traffic flow. While related projects such as NoW (Network on Wheels) and SeVeCom(Secure Vehicular Communication) focussed on the security challenges of the external communication and proposed solutions for privacy-preserving trustworthy V2X communication, the EVITA project focused on securing the internal on-board system in order to prevent, or at least detect, illegal tampering. Attacks on V2X communication can only be averted if trustworthy V2X communication is combined with on-board security avoiding the transmission of manipulated messages to the external communication partners. The objective of the EVITA project was to design, to verify, and to prototype building blocks for secure automotive on-board networks protecting security-relevant components against tampering and sensitive data against compromise. This is an essential prerequisite to the safe operation of V2X communication. Thus, EVITA addressed “advanced, reliable, fast and secure vehicle-to-vehicle and vehicle-to-infrastructure communication for new functionalities” as stated in the objective ICT-2007.6.2 “ICT for Cooperative Systems” of the European Union’s Seventh Framework Programme for research and technological development.

The following requirements had to be met in order to achieve secure on-board networks:

- Distributed security: All electronic components of a vehicle and the connections between them need to be protected because a network is only as secure as its weakest part.
- Real-time capability: A vehicle performing V2X communication needs to sign and verify up to several thousand messages per second.
- Cost–effectiveness: In the automotive industry, cost effective solutions tailored to the specific needs are necessary. It is not acceptable to pay for unused hardware capabilities.

These requirements are not met by today’s off-the-shelve security solutions such as smart cards and Trusted Platform Modules (TPMs) because they are not designed to fit to the automotive system environment. The existing security solutions had to be adapted to the automotive domain.

Security Requirements Analysis

Starting from relevant use cases and security threat scenarios, security requirements for automotive on-board networks are specified. Also legal requirements on privacy, data protection, and liability issues are considered.

This chapter describes use cases of automotive on-board networks that are expected to require security measures. The use cases serve as a basis for the deduction of security requirements for automotive on-board networks. The security requirements serve as input for the design of a secure on-board architecture and the development of appropriate security measures in order to prevent, or at least detect, attacks on automotive on-board networks.

Communication Architecture Car-to-Car and Car-to-Infrastructure Communication Architecture

All of the communication entities depicted in Figure 4.3 will be taken into account in EVITA. However, the in-vehicle communication system and the communication interface to the outside world are of main interest.

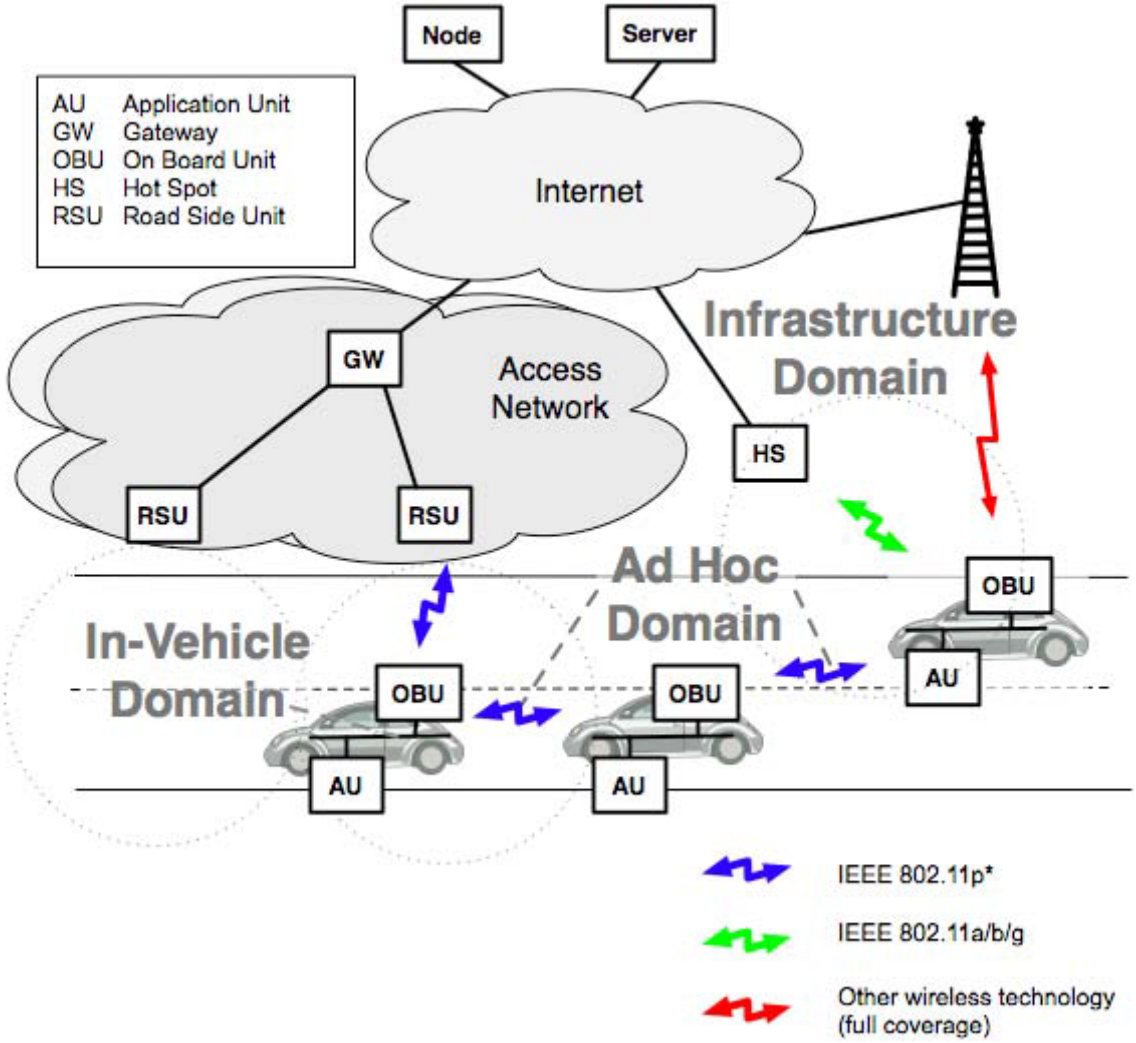


Figure 4.3 Car2X Communication System Architecture

On-Board Network Architecture

The Automotive on-board networks consist of

- electronic control units (ECUs), comprising a CPU, memory, and I/O devices,
- electronic sensors, and
- electronic actuators

that are connected with each other via some bus systems. The on-board network may possess wireless interfaces to the outside for communicating with service providers, road side units, and other vehicles and a wire-bound diagnostic interface. The on-board network may also possess wireless or wire-bound interfaces for connecting with mobile devices inside the car.

The embedded ECUs run both

- safety critical software applications and
- non-safety critical software applications.

Figure 4.4 shows a generalised on-board network architecture

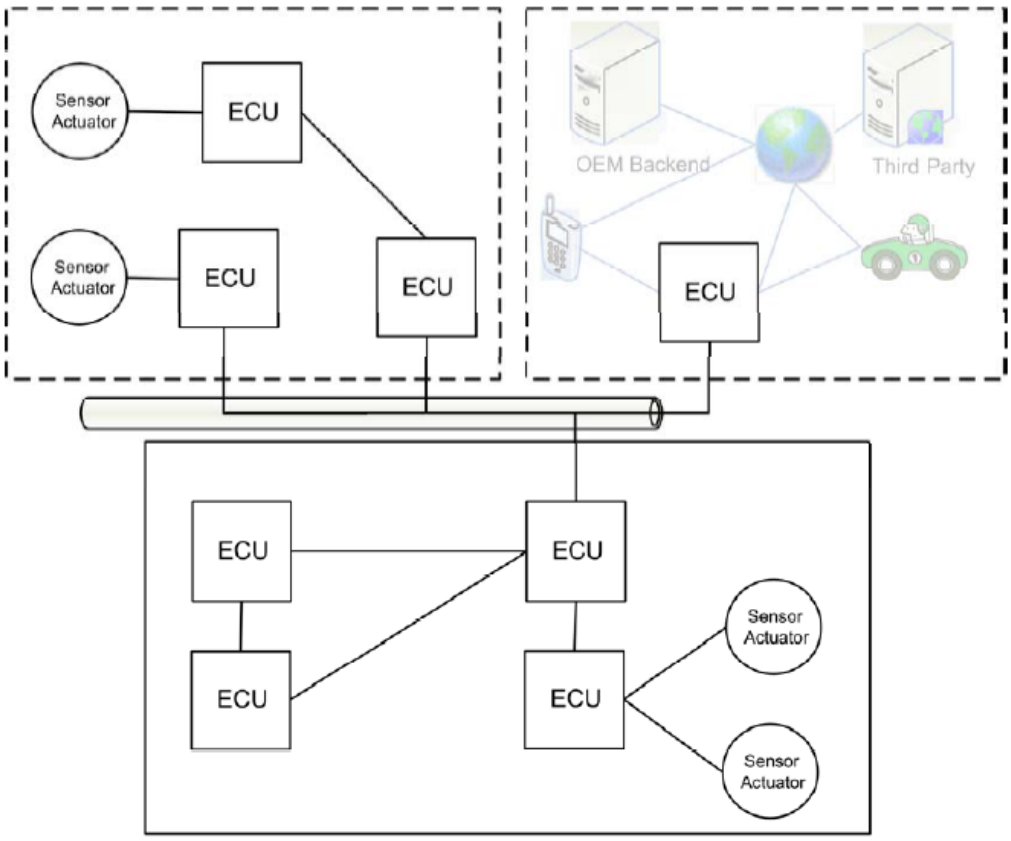


Figure 4.4 **Generalised on-board network architecture**

Figure 4.4 shows that ECUs, sensors and actuators can be clustered in domains and subdomains that can be interconnected with each other via various communication links. Thereby, different architectural topologies, e.g. line or star topology, are possible. The on-board network is assumed to operate in an uncontrolled environment. Therefore, its assets must be protected against a variety of threats.

Reference Architecture for EVITA Use Case Descriptions

The use case descriptions are based on a common architecture and topology for the in-vehicle communication networks consisting of ECUs, sensors, and actuators. This reference architecture for the use case descriptions is shown in Figure 4.5.

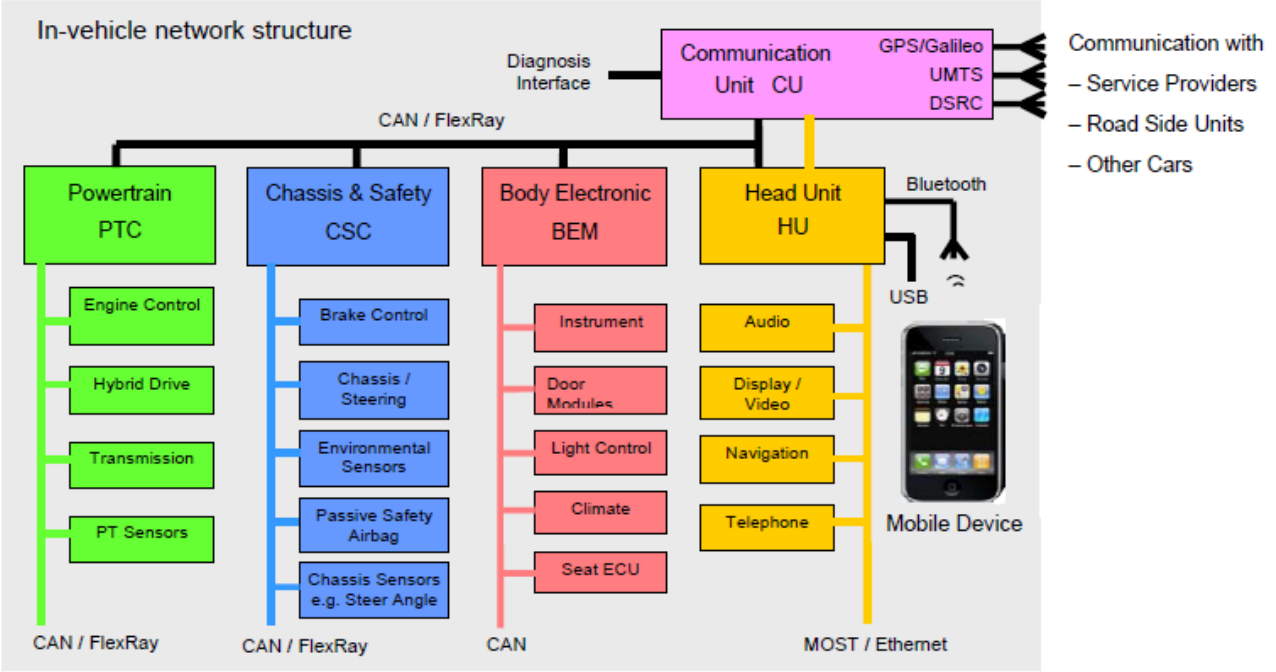


Figure 4.5 EVITA Use Cases Reference Architecture

Figure 4.5 shows a generalized topology for on-board networks of contemporary or next generation

The architecture represents an instantiation of the generalised on-board network architecture. This instantiation is used for describing the use cases related to a concrete exemplary in-vehicle network. However, the use cases of course can also be mapped to other instantiations of the reference model.

Within the exemplary instantiation, the control systems are clustered into different domains for Powertrain, Chassis & Safety, Body Electronics and Infotainment (Head Unit). The domain control units provide separate communication networks for their domains, control the high-level domain specific functions and are linked together via a backbone bus system. The communication unit that provides the wireless communication to the outside world on a cellular basis or via digital short-range communication (DSRC) is also connected to the backbone.

The Head Unit provides an interface to connect mobile devices to the Infotainment domain e.g. via Bluetooth or USB. With this instantiated architecture, use cases can be described and intrusion scenarios can be investigated in order to deduce security requirements.

Use Cases

Overview and categories

Use Cases where an e-safety related intrusion can likely happen are clustered as follows in categories:

- Car2MyCar (communication from other car to own car),
- MyCar2Car (communication from own car to other car),
- Car2I and I2Car (communication from car to infrastructure and from infrastructure to car),
- Nomadic Devices / USB Sticks / MP3,
- Aftermarket,
- Diagnosis.

The use cases are grouped into a number of categories for which e-security related intrusions were considered to be possible issues. The use cases that they developed include the following:

- Car2MyCar (communication from other car to own car)

  Use case 1: Safety reaction: Active brake

  Use case 2: Local Danger Warning from other Cars

  Use case 3: Traffic Information from other Entities

- MyCar2Car (communication from own car to other car)

  Use case 4: Messages lead to safety reaction

  Use case 5: Local Danger Warning to other Cars

  Use case 6: Traffic Information to other Entities

- Car2I and I2Car (communication from car to infrastructure and from infrastructure to car)

  Use case 7: eTolling

Use case 8: eCall

Use case 9: Remote Car Control

Use case 10: Point of Interest

- Nomadic Devices/USB Sticks/MP3

  Use case 11: Install applications

  Use case 12: Secure Integration

  Use case 13: Personalize the car

- Aftermarket

  Use case 14: Replacement of Engine ECU

  Use case 15: Installation of a Car2x Unit

- Workshop/Diagnosis

  Use case 16: Remote Diagnosis

  Use case 17: Remote Flashing

  Use case 18: Flashing per OBD

Now we will see three examples of the uses cases to understand how they approached them

Use Case 1 – (Car 2 My Car) Safety reaction: Active brake

The car receives a message that indicates that the car is in immediate danger of collision with an object. The only way to avoid the collision is an instant brake manoeuvre.

The emergency message contains longitude, latitude and altitude of the dangerous object, the time of message generation, the expiry time of the message, an indicator for the reliability of the information, a code that is classifying the object, an Id that is identifying the sender of the message and an event code that is classifying the emergency situation. All this information is packed in a message frame that adds checksum, information for protocol processing and if necessary security information. The receiving communication unit (CU) will check the message for correctness and then pass the information together with additional relevant information to the chassis safety controller (CSC). The additional information

consists of data about the position, speed, heading, type and size of communicating objects nearby; further attributes may also be added. The additional information was collected from older received messages and stored in the neighbourhood table. The neighbourhood table is a list of communication nodes from which the CU received messages in the past. The list contains the nodes Id, position, type and other available attributes. Nodes that are more than 1 km distant will be deleted from the list.

The information is provided in regular intervals (ca. 2/s). In parallel to the following action, the CU will assess whether it has to send the information out to other nodes. The assessment depends on the position of nearby CUs, the received RF power of the message and the type of the message. Only event messages will be rebroadcast. If it is obvious that all affected units that are even further away from the sender have received the same message, the message will not be rebroadcast, otherwise it is sent out again. The GPS unit of the CU is used to determine the position; this position is used internally and for car2X communication. The CSC will use further information that is available to perform a plausibility check. This information may be object lists from radar, lidar or video sensors together with data from digital maps, driver status information and status data of the car like position, speed, heading, steering angle etc. Except for the car status, all these data sources are optional.

If the plausibility check confirms the danger for the car, the CSC decides on appropriate action, which mainly depends on the possibilities that the vehicle dynamics and the neighbourhood conditions permit. If the CSC decides that a braking manoeuvre is the best solution, it will send a braking command and information concerning the best deceleration to the brake control unit. In addition, information about the emergency-braking manoeuvre will be sent to the CU. The CU will then broadcast an emergency braking message to warn following cars.

The brake control unit (BCU) will adjust the braking mechanics to get a deceleration as close as possible to the desired value, while keeping the car in a controllable state by executing ABS/TCS/ESC algorithms. When starting the braking, the BCU will send a message to the powertrain domain to reduce the driving power. This message is forwarded by the CSC and the Powertrain controller (PTC). The PTC will decide how best to comply with this request and will send the necessary commands to the units of the powertrain domain. The CSC will update the plausibility check and the concluding braking commands in regular intervals. The braking commands will be adapted to the situation assessment.

When the CSC gets information from environmental sensors (radar, lidar, video) and/or car internal sensors (digital map, speed, yaw rate, etc.) that show that the dangerous situation is no longer existent or that the driver is fully able and ready to cope with the danger, it returns control to the driver by adapting the deceleration to the braking pedal pressure.

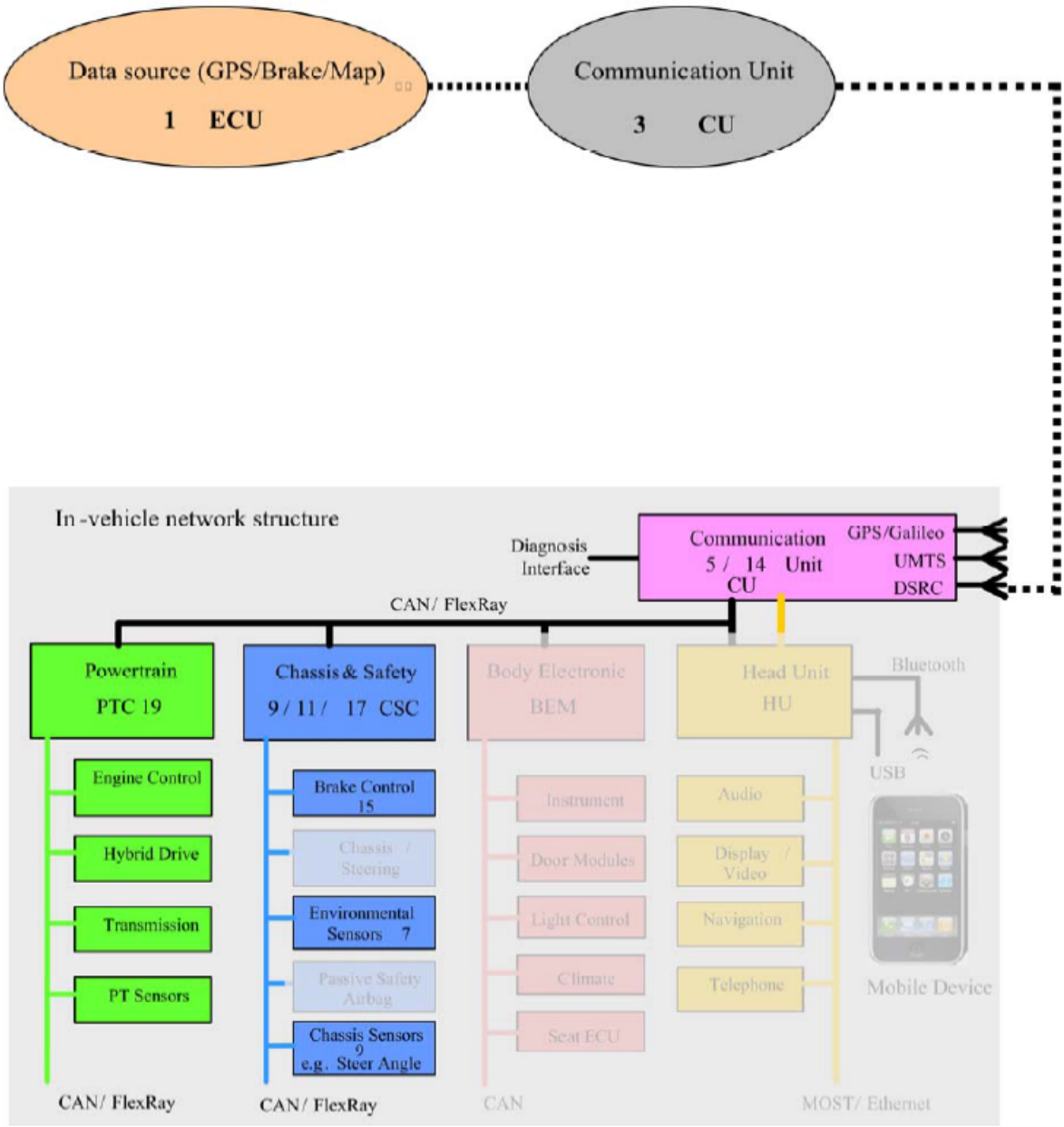


Figure 4.6 Communication Entities and Relations: Safety Reaction: Active Brake

Requirements

The requirements, for this use case, are divided into two categories: Functional requirements and Technical Requirements. We will not delve into these issues because they are beyond the scope of this thesis work, instead for us it is important to investigate the security aspect related to this use case.

Security Aspects

- The information received from another car needs to be evaluated regarding security andtrust (e.g. authenticity of data).
- Privacy of the broadcast car information has to be guaranteed

Use Case 7 – (Car 2 I and I 2 Car) eTolling

Car tolling is already in use in different countries using different techniques. Most are based on the same principle: The use of an extra On-Board Unit (OBU). In Germany, for example, the service called "Toll Collect" is used to account trucks. The use case will be described based on the German system. According to the EVITA use cases reference architecture the OBU can be seen as an enhanced CU.

The Toll Collect system provides two types of accounting: the manual accounting and the automatic one. Just the automatic one will be considered within the description of this use case.

To be able to automatically account the trucks, Toll Collect system used the combination of two positioning systems: the Global System for Mobile Communication GSM and the Global Positioning System GPS. Those two technologies are implemented in the Road Side Units (RSU) of the toll provider. In the vehicle, the OBU is equipped with a GPS antenna and GSM antenna in order to communicate with the RSU and to send the relevant information. With the position technologies, the OBU is then able to determine the driven distance in order to calculate the bill based on the driver contract information and in order to send it per mobile phone technology (GSM) to the data processing center of the toll provider.

Considering the fact that for car2X communication a communication unit will be introduced in the car, the logical consequence will be the use of this unit for toll purposes. Therefore, in the description it is assumed that the OBU as part of the Communication Unit will handle the communication with the RSU.

In this use case, the RSUs of the toll system provider are continually broadcasting a kind of wake-up signal. Depending on its position, an OBU recognises a toll road and automatically saves the necessary data for the accounting. Passing the toll provider RSU, the vehicle receives the control signal and the OBU automatically calculates the toll fee. Before sending the needed data for toll accounting, the CU checks the origin of the message (authentication of the RSU). If the RSU cannot be identified, the CU does not send any message after the check. Otherwise, it sends the data needed for accounting the driver: the type of the car, toll contract identification, pay method, and the signed bill of the last paid toll.

All the data sent by the CU are signed and encrypted in order to ensure that the driver will be correctly accounted and that only an allowed control center can process the data.

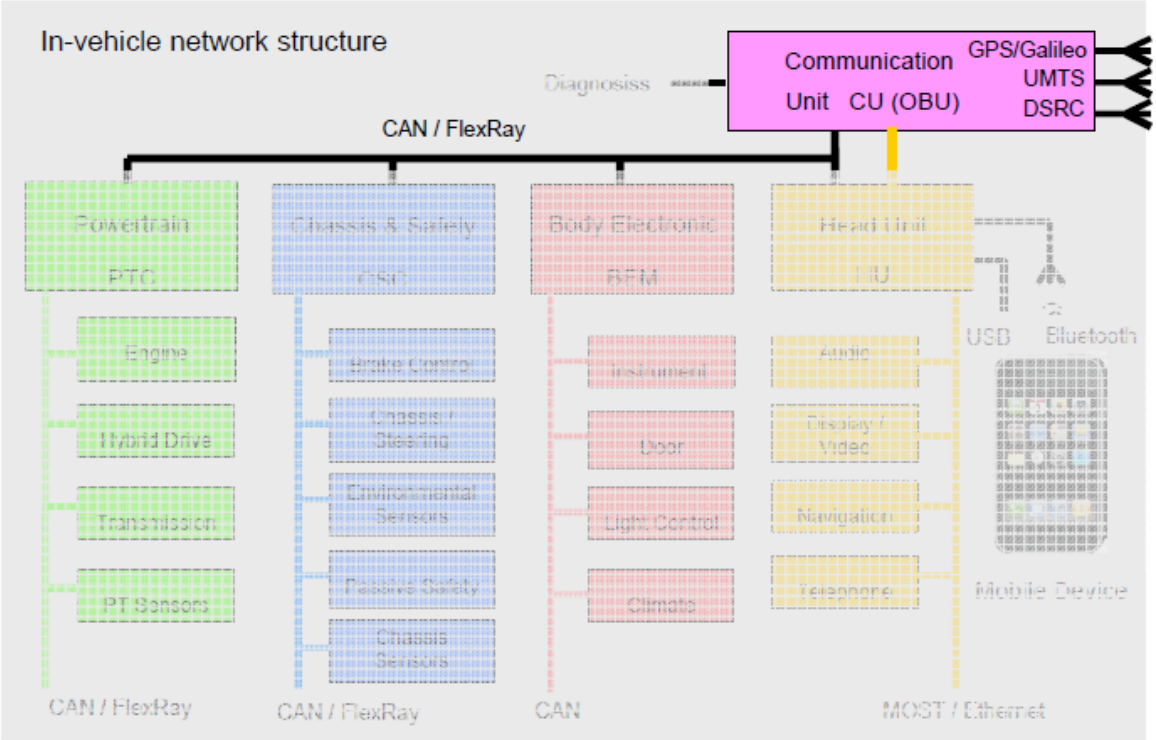


Figure 4.7 Communication Entities and Relations: eTolling

Security Aspects

- Authentication is an issue since the vehicle must check if the RSU is allowed to receive his billing information.
- Confidentiality is necessary because the information is sensitive.
- Data integrity is necessary to make sure the account data sent from the car is not manipulated.
- Non-repudiation is an issue since the driver should not be able to contest a correctly generated bill.
- Anonymity is an issue. For example, the RSU must be able to recognize the vehicle/owner without allowing an eavesdropper who overhears the connection to gain private information about the owner.

Use Case 18 – (Diagnosis) Flashing per OB

In use cases “Remote flashing” and “Remote Diagnosis”, the connection for diagnosis purpose is done wirelessly. Nowadays in Europe, car diagnosis is done hardwired. It is interesting to take a closer look at the use case, to identify the security issues service stations and vehicles owner are already confronted with. The description is based on the Standard Unified Diagnostics Services UDS. In this use case, an ECU firmware of a vehicle will be updated hardwired from a service station. A car owner takes his car to the area of a service station. To start the diagnosis session the car has to be activated. The ECU initializes its software and starts the diagnosis function, called diagnosis server. In this state, the diagnosis server is in the default mode (this is defined in the standard). The service station employee connects his diagnosis tool to the on-board diagnosis interface in the vehicle. This is done by plugging a cable to the diagnosis connector, which is different from car to car. A diagnosis request is then sent via the Communication Unit CU (on-board diagnosis interface) to the ECU. The ECU authenticates the diagnosis tool and checks the data integrity. If the request is successful, the ECU opens a programming session. The service station employee begins his diagnosis by checking the ECU type and firmware version. Assuming the ECU type is known, a comparison is also made to figure out the need of an update of the version. The diagnosis tool then sends the encrypted packets of the new firmware to the ECU, which stores it in the RAM. The new firmware is decrypted at ECU level and flashed in the ROM packet wise. The date of the update is written in the ECU and the programming session is closed by sending an EcuReset request to the ECU.

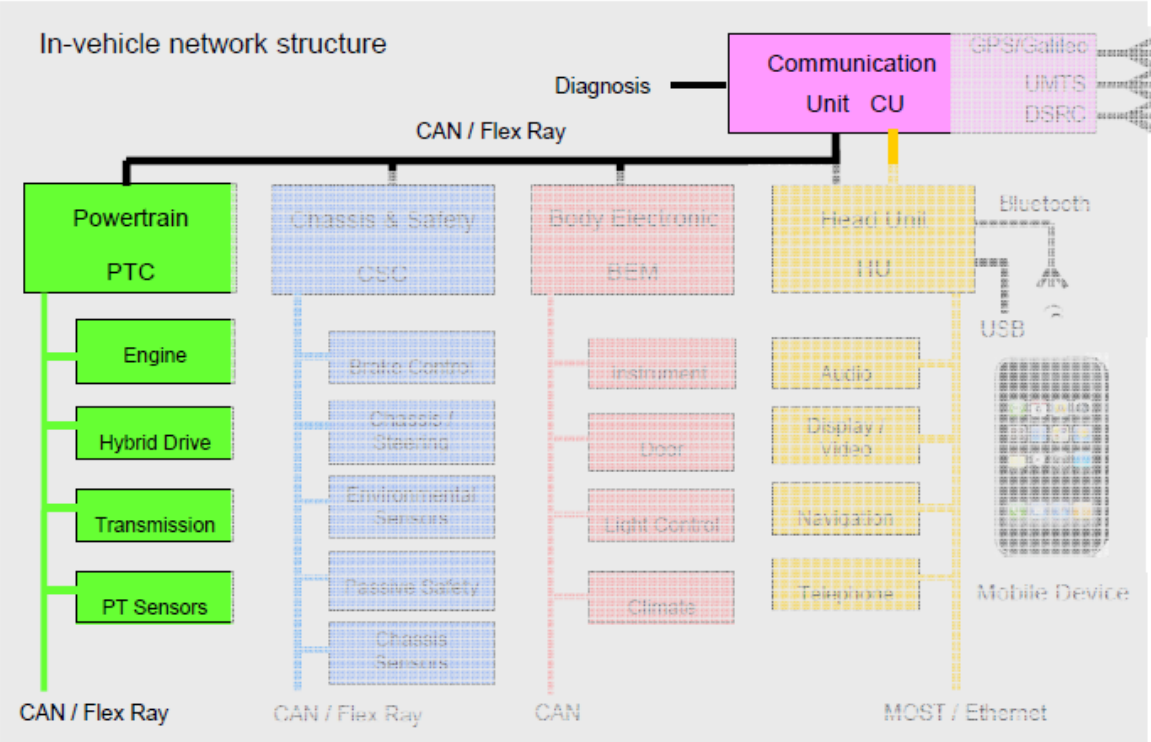


Figure 4.9 Communication Entities and Relations: Flashing per OBD

Security Aspects

- Authentication is required to avoid the compromising of the in-vehicle system by malicious code.
- Confidentiality is required to protect the know-how in the ECU update software.
- Data Integrity is necessary to make sure the update software was not manipulated before being executed.
- Freshness is required for the messages to be protected against replay attacks.
- Non-Repudiation is needed for example to avoid the repudiation of a wrongdoing by the service station.

Security requirements for automotive on-board networks based on dark-side scenarios

Future visions of road transportation include the networked vehicles and intelligent transport systems (ITS) that will enhance the safety of drivers and other road users, minimize pollution and maximize the efficiency of travel. The nature and interests of the stakeholders involved in future road transport systems therefore include:

- vehicle users – safe and efficient driving, valid financial transactions, personal privacy, protection of personal data;
- other road users – safe and efficient transport;
- vehicle/sub-system manufacturers – successful and affordable satisfaction of customer expectations, protection of IPR;
- ITS system operators – safe and efficient operation of systems, valid financial transactions, protection of user data;
- civil authorities – safe and efficient transportation networks, reliable financial transactions, data protection.

For the networked vehicles and intelligent transport systems (ITS) envisaged for the future, unauthorized access to vehicle or personal data might become possible, while the corruption of data or software could result in anomalies in vehicle function or traffic behaviour. Potential threat agents and their objectives may include:

- dishonest drivers – avoid financial obligations, gain traffic advantages;
- hackers – gain/enhance reputation as a hacker;
- criminals and terrorists – financial gain, harm or injury to individuals or groups;
- dishonest organisations – driver profiling, industrial espionage, sabotage of competitor products;
- "rogue states" – achieve economic harm to other societies.

Security functional requirements for information systems are broadly categorized into three types:

- confidentiality – prevention of unwanted/unauthorized disclosure of data;
- integrity – prevention of unwanted/unauthorized alteration or creation of data;
- availability – prevention of unwanted/unauthorized loss of data or access to data.

The EVITA project is concerned specifically with on-board networks within individual vehicles, rather than the wider ITS systems. In future road transport scenarios, breaches in the security of vehicle information or functions could lead to possible issues for stakeholders in four main areas:

- privacy – unwanted/unauthorized acquisition of data relating to vehicle/driver activity, vehicle/driver identity data, or vehicle/sub-system design and implementation;
- financial – unwanted/unauthorized commercial transactions, or access to vehicle;
- operational – unwanted/unauthorized interference with on-board vehicle systems or Car2X communications that may impact on the operational performance of vehicles and/or ITS systems (without affecting physical safety);
- safety – unwanted/unauthorized interference with on-board vehicle systems or Car2X communications that may impact on the safe operation of vehicles and/or ITS systems.

In order to define security and safety requirements for a system it is necessary to have an understanding of the operating environment and intended behaviour of the system. This is achieved through the specification of use cases for automotive on-board networks [Cap 4 Sect 1]. These and many other use cases may themselves suggest a number of security-related user requirements. However, the use cases also provide the basis for investigating a number of "dark scenarios" (threats), which are intended to establish ways in which the system could become a target for malicious attacks. The security issues identified from the dark scenarios are likely to include examples that also have safety implications.

Purpose and scope

The aim of the security requirements analysis is to derive, justify and prioritise IT security requirements and IT security related safety requirements for automotive on-board networks.

The inputs to the security requirements analysis are example use cases, dark-side scenarios, and the state of the art in standards and research. These inputs are viewed as the rationale for the requirements. The use cases require certain security functions in order to protect identified assets within the use case scenarios. The use cases also provide constraints and assumptions, such as performance constraints for the security functions. Security risk analysis of the threats identified in the dark-side scenarios will be documented as the rationale for the security objectives and security requirements. Traceability between the threats, objectives and requirements is accomplished by a structured approach.

This procedure defines a process for identifying vehicle security requirements, for assessing the relative risks of possible threats, and for addressing the subset of these security requirements that may be safety related. This process is then piloted to formulate requirements for the countermeasures needed to reduce the vulnerability of the vehicle's on-board architecture to threats that may lead to possible safety concerns and risks to assets.

Before detailing the security engineering process, we introduce classes of security requirements that are relevant for automotive on-board networks. The (informal) explanations reflect the way these concepts are generally understood.

Data origin authenticity

A data origin authenticity property applies to a quantum of information and a claimed author. The property is satisfied when the quantum of information truly originates from the author. The property can be made more specific by providing an observation of the quantum of information (defined, e.g., by a time and a location in the system). The author can also be constrained by adding a time and/or a place of creation of the quantum of information. Note that in most security-oriented frameworks data origin authenticity implies integrity.

Integrity

An integrity property applies to a quantum of information between two observations (defined, e.g., by a time and a location in the system). The property is satisfied when the quantum of information has not been modified between the two observations. It guarantees for instance that the content of a storage facility has

not been modified between two given read operations, or that a message sent on a communication channel has not been altered during its journey.

Controlled access (authorization)

A controlled access property or requirement applies to a set of actions and/or information and a set of authorized entities. The property is guaranteed if the specified entities are the only entities that can perform the actions or access the information. The property can be further detailed with time constraints on the period of authorization. Controlled access is needed to ensure that stakeholders only have access to information and functions that they are authorized to access as appropriate to their expected activities.

Freshness

A freshness property or requirement applies to a quantum of information, a receiving entity and a given time. The property is satisfied if the quantum of information received by the entity at the given time is not a copy of the same information received by the same or another entity in the past. Ensuring freshness can be used to prevent replay attacks.

Non-repudiation

A non-repudiation property or requirement applies to an action and an entity performing the action. The non-repudiation of the action is guaranteed if it is impossible for the entity that performed the action to claim that it did not perform the action. This property can be further detailed with a set of entities for which the action needs to be undeniable, with a time limit, etc.

There may be specific legal requirements for non-repudiation. However, non-repudiation may also be introduced for convenience, for example, as an aid in providing evidence or proving liability.

Privacy/anonymity

A privacy property or requirement applies to an entity and a set of information. Privacy is guaranteed if the relation between the entity and the set of information is confidential. Anonymity, for instance, is the property that the relation between an entity and its identity is confidential.

Privacy is frequently a major concern when the entity involved is an individual or a vehicle owned by an individual. For example, an adversary constantly recording the location of a vehicle and knowing the identity of the driver may be considered as violating the driver's privacy with respect to her movements.

Privacy requirements are needed to ensure that the anonymity of stakeholders and confidentiality of their sensitive information are assured. Sensitive information introduced by the application shall be identified. For users, sensitive information may include (but is not limited to) the following:

- identity of a specific car and/or driver,
- current location of a specific car and/or driver,
- past locations of a specific car and/or driver,
- properties of the vehicle that can be used for tracking a specific car and/or driver (e.g. car manufacturer, model, colour),
- behaviour of a specific car and/or driver (e.g. number of critical situations, speeding),
- records of telephone calls, internet activity, email messages, account information and driving characteristics,
- identity of specific cars and/or drivers involved in particular C2X transactions.

For vehicle manufacturers and system suppliers, sensitive information may include (but is not limited to) the following:

- identity of a specific car,
- car manufacturer and model,
- design information (algorithms, control parameters),
- performance data.

Privacy requirements must be made consistent with potentially conflicting requirements for identification, auditing, non-repudiation and jurisdictional access, which may require users to be identified and information about their interactions to be stored.

Confidentiality

A confidentiality property applies to a quantum of information and a set of authorized entities. The property is satisfied when the authorized entities are the only ones that can know the quantum of information. Privacy relies on confidentiality and can be considered as a special case of confidentiality.

Availability

An availability property or requirement applies to a service or a physical device providing a service. The property is satisfied when service is operational. Denial of service attacks aim at compromising the availability

of their target. The property can be further detailed with the specification of a period during which the availability is required and of a set of client entities requesting the availability.

Approach

The basic elements that are required of security and safety engineering processes are similar and include the following activities:

- develop a high-level (functional) model of the system to be analysed;
- identify safety hazards or security threats;
- classify the safety hazards or security risks;
- assess the associated risks;
- derive requirements for specific functions and assurance levels to mitigate the risks;
- evaluate the design and implementation for compliance with the requirements.

The methodology for inferring security functional requirements involves the following Steps:

1. Description of system under investigation and its environment;
2. Description of relevant use cases;
3. Identification of the assets to be protected within the described use cases (e.g. ECU, application/ process, sensor, data, communication between system entities, etc.);
4. Identification of the threats posed to each asset in order to infer basic security functional requirements;
5. Evaluation and assignment of respective risks (probability of threat, cost/loss, risk classification);
6. Identification of respective security functional requirements for each threat according to risk analysis.

The system under investigation is an automotive on-board network consisting of embedded electronic control units (ECUs), sensors, and actuators that are connected with each other via some bus systems. Figure 4.5 shows the assumed on-board network architecture.

The on-board network is assumed to possess interfaces to the outside for communicating with mobile devices, service providers, roadside units, and other vehicles:

- wireless interfaces such as GSM, UMTS, Bluetooth, W-LAN and DSRC and
- a wire-bound diagnostic interface.

For example, the on-board network may possess a Bluetooth interface in order to connect with mobile devices inside the car.

The generalised architecture of Figure 4.4 is too abstract for describing use cases. Therefore, use cases are described in the chapter 4 in terms of the reference architecture shown in Figure 4.5, which is an instantiation of the generalised architecture in Figure 4.4.

System assets

The main components of an automotive on-board network (see Figure 1 and Figure 2) that may become targets of attacks are:

- In-vehicle devices: ECUs, sensors and actuators,
- Safety critical and non-safety critical applications running on in-vehicle devices,
- Communication links internally within ECUs, between ECUs, between ECUs and sensors, between ECUs and actuators and between applications running on in-vehicle devices.

Threat identification (dark-side scenarios)

The purpose of developing the "dark-side" scenarios is to identify possible security threats and to allow aspects such as the desirability (to the attacker), opportunity, probability and severity of attacks to be assessed in order to support the security risk assessment activities. The approach adopted in developing the dark-side scenarios for the EVITA project is based on the following elements:

- identification and classification of possible attack motivations;
- evaluation of associated attacker capabilities (e.g. technical, financial);
- attack modelling, comprising:
    - identification of specific attack goals that could satisfy the attack motivations;
    - construction of possible attack trees that could achieve attack goals, based on the functionality identified in the use cases.

This approach has already been used in the Network-on-Wheels project. The attack trees are interpreted in terms of an initiating "attack goal", providing the attacker with an illegitimate benefit, which can be satisfied

by one or more “attack objectives” that have a negative impact on the stakeholders. Each “attack objective” could be achieved by one or more attack methods, which may consist of one or more combinations of attacks on specific system assets.

Attacks that could have an impact on the safety of a car based on direct physical access in order to manipulate the hardware of that car (e.g. modification of ECUs or other electronic components) are excluded from this analysis as they are beyond the scope of the EVITA project.

These classes of attacks are already feasible and probably always will be. While some outcomes of EVITA will help in the detection of malevolent modifications to a vehicle, this is not a specific objective of the project. Consequently, direct physical attacks against the hardware of the targets of attacks are out of scope. However, manipulations of devices that are under the control of the attacker are within the scope of EVITA (e.g. side channel attacks or extraction of keys); since attackers may modify their own vehicle in order to perform attacks against others.

Overview of risk analysis

In order to assess the “risk” associated with an attack it is necessary to assess the “severity” of the possible outcome for the stakeholders, and the “probability” that such an attack can be successfully mounted.

At the highest level, the security objectives are:

- operational – to maintain the intended operational performance of all vehicle and ITS functions;
- safety – to ensure the functional safety of the vehicle occupants and other road users;
- privacy – to protect the privacy of vehicle drivers, and the intellectual property of vehicle manufacturers and their suppliers;
- financial – to prevent fraudulent commercial transactions and theft of vehicles.

These security objectives counter generic security threats, as outlined in Table 4.1.

Generic Security Threats				Security Objectives
Aims	**Target**	**Approach**	**Motivation**	
Harming individuals	Driver or passenger	Interference with safety functions of a specific vehicle	Criminal or terrorist activity	Safety Privacy
Harming groups	City or state economy, through vehicles and/or transport system	Interfere with safety functions of many vehicles or traffic management functions	Criminal or terrorist activity	Safety Operational
Gaining personal advantage	Driver or passenger	Theft of vehicle information or driver identity, vehicle theft, fraudulent commercial transactions	Criminal or terrorist activity	Privacy Financial
	Vehicle	Interference with operation of vehicle functions	Build hacker reputation	Operational Privacy
	Transport system, vehicle networks, tolling systems	Interference with operation of traffic management functions or tolling systems	Enhanced traffic privileges, toll avoidance,	Operational Privacy Financial
Gaining organizational advantage	Driver or passenger	Avoiding liability for accidents, vehicle or driver tracking	Fraud, criminal or terrorist activity, state surveillance	Privacy Financial
	Vehicle	Interference with operation of vehicle functions, acquiring vehicle design information	Industrial espionage or sabotage	Privacy Operational Safety

Table 4.1 Generic security threats and security objectives

The severity of an attack is considered in terms of the four different aspects that may be associated with harm to the stakeholders (operational, safety, privacy, and financial aspects), as a 4-component vector with a range of qualitative levels that are based on the severity classifications used in vehicle safety engineering. The severity of an attack is assessed using the attack trees, by considering the potential implications of the attack objectives for the stakeholders.

The probability of a successful attack is also derived from the attack trees, by identifying combinations of possible attacks on the system assets that could contribute to an attack method. Thus, the risk analysis is organized by attack tree, and decomposed down to asset level. However, further decomposition may be helpful in estimating the probability of success (which is related to the "attack potential") for attacks on specific assets.

The probability and severity combinations are mapped to a series of risk levels ranging from 0 (lowest) to 6 (highest) in order to rank relative risks. In this scheme, high probability attacks with the severest outcomes have the highest risk levels, while low probability attacks with the least severe outcomes have the lowest risk levels. Between the extremes, the risk levels increase with rising probability and severity. As severity is

expressed in the form of a 4-component vector, the risk measure associated with an attack is also a 4-component vector. Furthermore, as several different attack methods may achieve the same attack objective, the result of the risk assessment is a set of risk vectors.

This provides a convenient basis for systematically identifying threats that need to be countered with priority:

- Where a number of possible attack objectives may achieve the attack goal, the attack objective with the highest perceived risk level is the priority for countermeasures to reduce the risk level for the attack goal;
- Where a number of possible attack methods may lead to the same attack objective, the attack method with the highest perceived combined attack probability is the priority for countermeasures to reduce the risk level for the attack objective;
- Where a number of asset attacks may lead to the same attack method, the asset attack with the highest perceived attack probability (i.e. lowest attack potential) is the priority for countermeasures to reduce the risk level for the attack method.
- The repeated occurrence of particular attack patterns in attack trees is a further indicator for prioritising countermeasures that are likely to provide favourable cost-benefit properties.

The security requirements are based on the use cases and attack trees (Dark-side scenarios) and derived in a systematic manner. The level of detail directly originates from the size of the use case model. The level of coverage is restricted to the amount of information that was input to the security analysis.

The fulfilment of security requirements is not measurable beyond Boolean (i.e. true or false). The fulfilment of security requirements in on-board architecture and protocol specifications will be verified by formal methods.

The following subsections list the security requirements determined using the two approaches outlined in the previous chapters, classified according to security properties.

Privacy

These requirements target every relation between identity and privacy-relevant information that are not already covered by the anonymity requirements.

Requirement reference: Confidentiality_3
Informal description: The position of a car at a certain point in time must be confidential.
Semi-formal description: *confidential({GPS-Sensing(car,Position), GSM-Send(car, Billing-Information)}, Position, allPositions, Lzero, car)*
Use case references: 7

Requirement reference: Confidentiality_4
Informal description: The personal information stored within the car shall remain confidential even during exchange of ECUs.
Semi-formal description: *confidential({ExchangeECU(Maintanance,car,ECU(Data)},Data, allData, Lzero, car)*
Use case references: 14

Requirement reference: Confidentiality_5
Informal description: The PoI-Configuration (the driver's preconfiguration regarding the reception of PoI- (Point of Interest) information) stored within the vehicle for a driver shall remain confidential even during exchange of data with an RSU.
Semi-formal description: *confidential({Receive(car,PoI-Info)}, PoI-Configuration, allPoIConfs, Lzero, car)*
Use case references: 10
Notes: The configuration of the vehicle can reveal personal information. In the case of PoIs, for example, personal preferences may be revealed.

Priority of security requirements

Analysis of the attack trees demonstrates that specific asset attacks may contribute to different attack objectives within the same attack tree, and may contribute to attack objectives associated with other attack trees. For a particular asset attack, both the risk level (which reflects the severity of outcome for an attack, and the attack potential associated with the asset attacks that contribute to it) and the number of instances from the collection of attack trees are indicators of the importance of the asset attack and the likely benefits of countermeasures for reducing the probability of successful attacks of this nature.

The severity measure is considered in terms of a four-component vector that reflects potential safety, operational, privacy and financial aspects that may be associated with a security attack. For safety-related security threats, the "controllability" of the hazard by the driver constitutes an additional dimension for the probability contribution to the relative risk level. The proposed mapping of these parameters to relative risk level is summarised in Table 4.2, where non-safety risks and highly controllable safety-related risks are associated with controllability C=1, and only safety–related risks are associated with the higher controllability measures. In principle, the relative risk is also a four-component vector, inheriting this property from the severity, although in the EVITA analysis it is usually found to be of lower order. The class "R7+" that is used in Table 4.2 denotes levels of risk that are unlikely to be considered acceptable, such as safety hazards with the highest severity classes and threat levels, coupled with very low levels of controllability.

Controllability (C)	Severity (S_i)	Combined Attack Probability (A)				
		A=1	A=2	A=3	A=4	A=5
C=1	S_i=1	R0	R0	R1	R2	R3
	S_i=2	R0	R1	R2	R3	R4
	S_i=3	R1	R2	R3	R4	R5
	S_i=4	R2	R3	R4	R5	R6
C=2	S_S=1	R0	R1	R2	R3	R4
	S_S=2	R1	R2	R3	R4	R5
	S_S=3	R2	R3	R4	R5	R6
	S_S=4	R3	R4	R5	R6	R7
C=3	S_S=1	R1	R2	R3	R4	R5
	S_S=2	R2	R3	R4	R5	R6
	S_S=3	R3	R4	R5	R6	R7
	S_S=4	R4	R5	R6	R7	R7+
C=4	S_S=1	R2	R3	R4	R5	R6
	S_S=2	R3	R4	R5	R6	R7
	S_S=3	R4	R5	R6	R7	R7+
	S_S=4	R5	R6	R7	R7+	R7+

Table 4.2 Combined risk graph for safety-related (C≥1) and non-safety (C=1) security threats

Analysis of the attack trees, which were based on the EVITA use cases and an assumed architecture based on the EASIS project, has identified small numbers of possible attack methods on various system assets that could lead to the achievement of potential attacker objectives. These "asset attacks" represent the terminal nodes of the attack trees, and specific subsets of the security requirements that are considered to be necessary to protect against such attacks have been identified. The risk analysis identifies severity at the higher levels of the attack trees and works up associated probability measures from the asset attacks that terminate the lower levels of the attack trees. Thus, the attack trees, risk analysis and security requirements are mapped to each other via the concept of asset attacks.

The same asset attacks often appear in more than one of the attack trees, but may be associated with different risk levels because the severity measures differ between trees. The results of the EVITA risk analysis activity are therefore summarized in terms of the number of occurrences of particular risk levels associated with specific asset attacks in Table 4.3, which also lists the security requirements to counter each such asset attack. The risk level reported in Table 4.3 is based on the worst case where more than one element of the risk vector is present in the risk analysis tables. Where alternative attack routes are available, the associated risk level is adjusted to reflect the attack probability for the asset attacks involved. Consequently, Table 3 indicates the worst-case risk estimates for all of the attack alternatives listed in the risk analysis tables. Thus, if high-risk asset attacks are mitigated by appropriate security countermeasures, the only change required to Table 3 is to remove or modify the entries corresponding to the risks that have been mitigated. The risk levels associated with lower risk attack alternatives remain unchanged.

Identified threats		**Risk analysis results**		**Security requirements**
Asset	***Attack***	***Risk level***	***Number of instances***	
Chassis Safety Controller	Denial of service	1 2	3 1	Authenticity_6, Availability_102, Availability_106
	Exploit implementation flaws	4 5	1 1	Authenticity_1, Authenticity_2, Authenticity_3, Authenticity_4, Authenticity_5, Authenticity_101, Authenticity_102, Authenticity_103, Confidentiality_101, Confidentiality_102, Integrity_101, Integrity_102, Integrity_103, Integrity_104, Integrity_105, Freshness_101, Freshness_102, Freshness_103, Availability_101, Availability_102, Availability_103, Availability_106, Availability_107, Availability_108, Privacy_101, Privacy_102, Privacy_103, Privacy_104, Privacy_105
	Corrupt data or code	3	1	Authenticity_1, Authenticity_2, Authenticity_5, Authenticity_6, Authenticity_101, Authenticity_102, Authenticity_103, Confidentiality_101, Confidentiality_102, Integrity_101, Integrity_102, Integrity_103, Integrity_104, Integrity_105, Freshness_101, Freshness_102, Freshness_103, Availability_101, Availability_102, Availability_103, Availability_106, Availability_107, Availability_108, Access_101, Access_102, Privacy_101, Privacy_102, Privacy_103, Privacy_104, Privacy_105
	Flash malicious code	4 5 6	1 1 1	Authenticity_1, Authenticity_2, Authenticity_3, Authenticity_4, Authenticity_5, Authenticity_101, Authenticity_102, Authenticity_103, Confidentiality_101, Confidentiality_102, Integrity_101, Integrity_102, Integrity_103, Integrity_104, Integrity_105, Freshness_101, Freshness_102, Freshness_103, Availability_101, Availability_102, Availability_103, Availability_106, Availability_107, Availability_108, Access_101, Access_102, Privacy_101, Privacy_102, Privacy_103, Privacy_104, Privacy_105

Table 4.3 Summary findings of risk analysis

The EVITA security architecture provides security functions with respective modules and interfaces, which can be easily integrated by application developers. Some of these security functionalities convey:

- Flexible on-/off-board user/identity authentication and authorization (access control)
- Communication filtering, malware protection, intrusion detection and response
- Strong separation and compartmentalization of data, applications and processes
- Privacy concealment services for identities and data
- Consistent central point for (automated) policy integration and security updates,
- Sandboxed runtime environments for in-vehicle execution of untrusted applications,

The set of EVITA modules comprises software and hardware modules realizing a security framework for automotive environments. Specifically, this security framework is comprised of:

- Software security modules providing security functionality, such as authentication, authorization, establishment of authentic and confidential communication channels, etc. as well as the software module including the API in order to access the functionality provided by the EVITA hardware security modules.
- Hardware security modules providing security functionality, such as secure storage of keys, secure operation of cryptographic algorithms, etc.
- Separation mechanism, e.g. based on virtualization or microkernels.

Within this chapter, we address the key design approaches for the EVITA security architecture.

We show how recent developments in automotive on-board communication and security systems influenced our design decisions and provide a refinement of security requirements, which were previously identified. Consequently, we analyse possible design goals for separation and hardware/software co-design and present the need for security services, represented as a framework. This provides a modular and flexible methodology for incorporating security deployment for upcoming automotive on-board networks.

Summary of EVITA Security Requirements

After a deep analysis, EVITA has inferred the following set of security requirements in order to satisfy the stated security objectives.

SR.1 Integrity/Authenticity of e-Safety related events. Actions depending on critical information should be decided based on assurances about integrity and authenticity in terms of origin, content, and time. Forgery of, tampering with, or replay of such information should at least be detectable.

SR.2 Integrity/Authenticity of ECU/firmware installation/configuration. Any replacement or addition of an ECU and/or its firmware or configuration to the vehicle shall be authentic in terms of origin, content, and time. In particular, the upload of new security algorithms, security credentials, or authorizations should be protected.

SR.3 Secure execution environment. Compromises to ECUs should not result in system wide attacks, primarily with regard to e-safety applications. Successful ECU attacks should have limited consequences on separate and/or more trusted zones of the platform.

SR.4 Vehicular access control. Access to vehicular data and functions should be controlled (e.g. for diagnosis, resources, etc.).

SR.5 Trusted In-Vehicle ECU platform. The integrity and authenticity of operated software shall be ensured. An altered platform might be prevented from running in an untrusted configuration (e.g. via comparison with a trusted reference) if so required.

SR.6 Secure in-vehicle data storage. Applications should be able to use functionality in order to ensure access control to as well as the integrity, freshness and confidentiality of data stored within a vehicle, especially for personal information and security credentials.

SR.7 Confidentiality of in-vehicle and external communication. The confidentiality of existing software/firmware as well as updates and security credentials shall be ensured. Some applications might additionally require that part of the traffic they receive or send internally or externally should remain confidential.

SR.8 Privacy. A privacy policy shall be enforceable on personal data stored within a vehicle or contained in messages sent from a vehicle to the outside. For example, some applications should limit the ability to link sent messages.

The corresponding fine-granular requirements are linked to possible attacks on the vehicle infrastructure. These attacks are associated with a certain risk level. They have analysed their condensed security requirements according to the EVITA risk and security analysis, reflecting the gravity of the individual security requirements SR.1 to SR.8.

We can derive two main security requirement classes from the occurrences of the risk levels.

Class one in the upper part of the tables contains those requirements that reach a maximum level of 7 and class two enlists those requirements that reach a maximum level of 6. The table is sorted by importance, i.e. by the last components of the vector. When we additionally interpret the vector of occurrences, we can define the importance of an individual requirement.

The vector reads as follows:

$[x_1, x_2, x_3, x_4, x_5, x_6, x_7]$

Each component x_n of the vector reflects the number of occurrences of a specific risk level n (See Table 4.3). The graphical representation of Table 4.4 is shown in Figure 4.10. We can see the distribution of the occurrences of the specific risk levels at a glance in this histogram.

Security requirement	Risk level range of attack per requirement	Occurrences of security levels per requirement as a vector
SR.1 SR.7 SR.2 SR.8 SR.4	[1–7]	[10,19,24,15,11,05,03] [09,17,21,11,10,05,02] [10,19,24,15,11,05,01] [09,14,17,09,08,04,01] [03,09,09,05,03,02,01]
SR.5	[1–6]	[08,11,13,06,05,03,00]
SR.3 SR.6	Not specified in D3.2	[n/a]

Table 4.4: Risk levels of broken down requirements

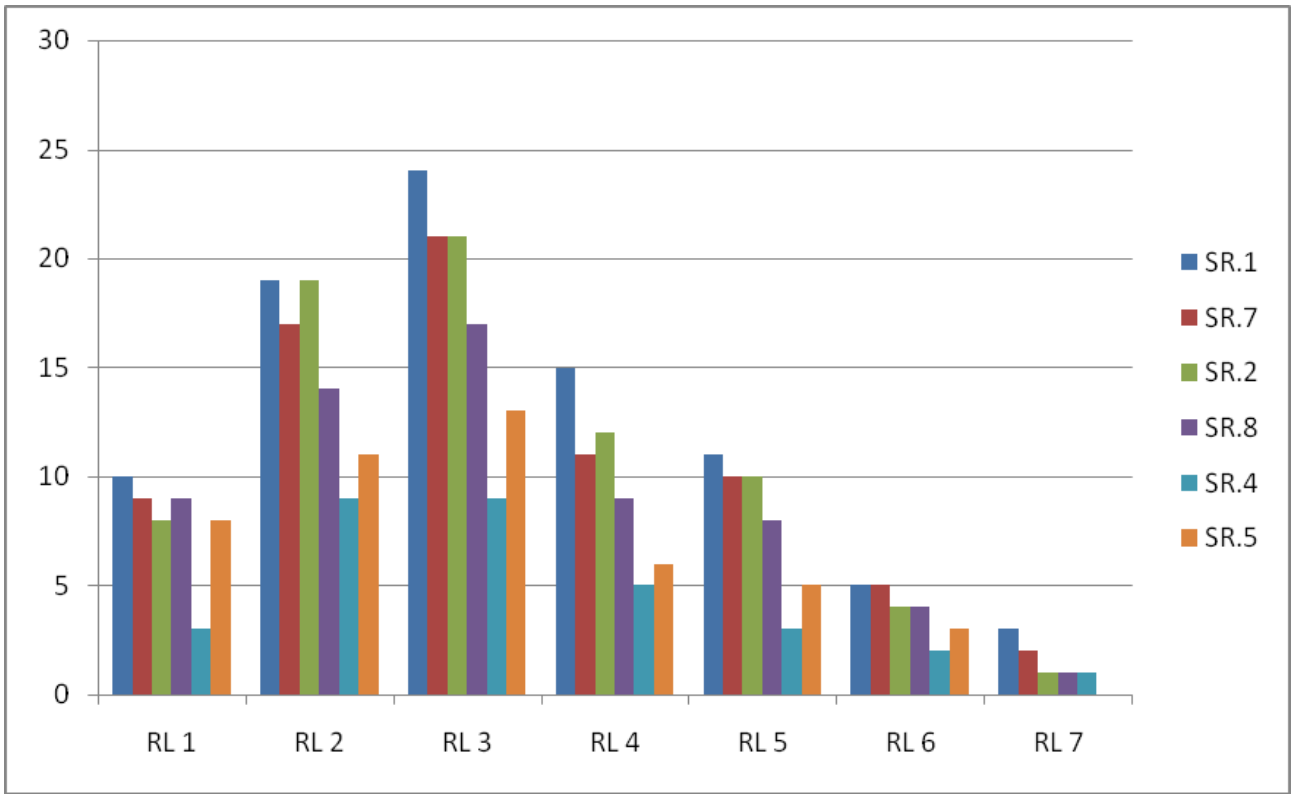


Figure 4.10: Risk levels associated with granular Security Requirements

Many of the security requirements defined target a very abstract architecture and concentrate on actions at the system borders triggered by or influencing the system environment.

These requirements need to be refined into requirements that can be satisfied by cryptographic mechanisms or security protocols.

The result of this process is categorized into three sets of requirements that are recapitulated in the following.

The first set targets the requirements of a vehicle's driver towards its own vehicle. Especially for the presented use cases, the driver must be assured of the correct behaviour of the Environmental Sensor, the Chassis Sensor, the GPS Sensor and the Communication Unit, namely that they forward only those data packets onto the vehicle's internal bus that were actually measured/received which corresponds to SR.2 and SR.3. Further, the Application ECU must base its decisions and therefore issue messages to the on-board display or the brake controller only based on according incoming data, which also corresponds, to SR.2 and SR.3.

Finally the on-board display and the brake controller shall only act, if they received a message on the bus that includes this command, which corresponds to SR.2 and SR.3 as well.

The driver also requires assurance regarding the on-board communication. Specifically this includes messages sent from the Sensors, the GPS and the CU to the Application ECU and messages from the

Application ECU sent to the display and the brake controller to originate from the specific senders, which corresponds, to R.1 and in case of cryptographic protocols to SR.6.

The second set of requirements target the communication between vehicles. The driver of a vehicle must be assured that messages received by its vehicle's Communication Unit originate from the Communication Unit of another vehicle or from a Road Side Unit, which corresponds to SR.1 and in case of cryptographic protocols to SR.6.

The third set of requirements target this other vehicle where a message is received from.

The driver of the receiving vehicle must be assured that within the sending vehicle the GPS Sensor, the Chassis Sensor and the Communication Unit will behave correctly, namely that they forward only data that was sensed/received before which corresponds to SR.2 and SR.3.

The Application ECU must base the issuing of a warning only on according sensor data, which also corresponds to SR.2 and SR.3 and finally the communication among the ECUs of the sending car to originate from specific senders which corresponds to SR.1 and in case of cryptographic protocols to SR.6. However, these are the assurances that the receiving vehicle's driver will require in order to believe the functional path leading to a warning or active brake, such that an assurance of these requirements fulfilment SR.1, SR.2, SR.3 and SR.6 must be available at the receiving vehicle.

At this point, a top-down approach has been applied with a deep study of the possible partitioning between hardware and software. The final result are three different design.

Towards the security modules

At this point having drawing up a "ranking" of the risks and required safety levels, three general hw architectures are designed to solve the security problem in the automotive environment

Of course, the solutions developed are the result of careful choices of co-design that will not be explored in this thesis as it is beyond the scope of the same. So even if it should be necessary a further explanation, we will now revise the three modules made, only to highlight some unique aspects that will be useful in the course of treatment.

EVITA Hardware Security Modules

As security in hardware is also always a matter of cost, they have decided to create three different variants of their hardware security module: Full, medium, and light EVITA HSMs. As depicted in Figure 4.11, these variants offer different levels of security functionality and performance. They are specifically designed to fit the individual needs of:

- V2X messages (high-speed asymmetric encryption and key storage)
- On-Board communication between ECUs (low-speed asymmetric cryptography, high-speed symmetric encryption and key storage; dynamic communication requirements)
- On-Board communication to Sensors and Actuators (symmetric encryption and key storage; static communication requirements)

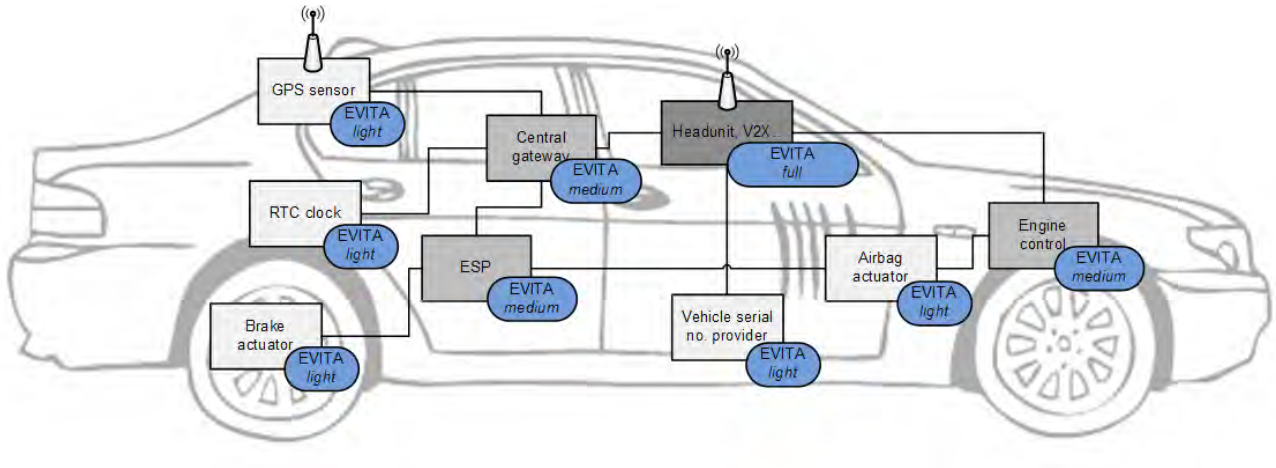


Figure 4.11: Example of EVITA hardware security modules deployment architecture

EVITA full hardware security module (V2X level)

In order to satisfy the performance requirements for signing and verifying messages for V2X communications, a very efficient asymmetric cryptographic engine is required. Thus, the EVITA full HSM is applied in this case, which provides the maximum level of functionality, security, and performance.

Figure 4.12 depicts the architecture of a full EVITA HSM, which generally consists of two

Parts, the cryptographic building block that realizes all cryptography hardware operations and the logic building block that connects the EVITA hardware with the normal ECU application core.

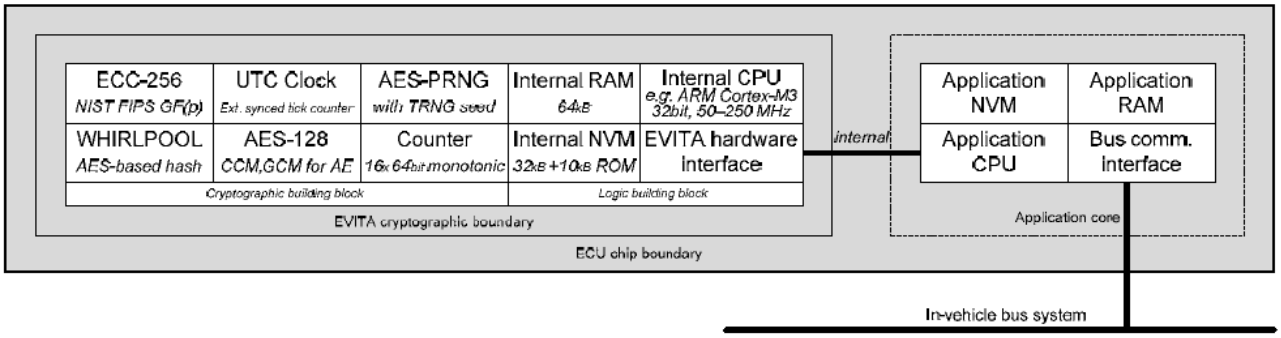


Figure 4.13: EVITA full hardware security module (V2X level)

The EVITA full HSM (see Figure 7) provides the following cryptographic building blocks:

- ECC-256-GF(p) is a high-performance asymmetric cryptographic engine based on a high-speed 256-bit elliptic curve arithmetic using NIST approved prime field parameters.10
- WHIRLPOOL is an AES-based hash function as proposed by NIST.11
- AES-128 is a symmetric block encryption/decryption engine using the official NIST advanced encryption standard. It supports not only standard block encryption modes of operation such as ECB and CBC, but also advanced encryption as used, for instance, in authenticated encryptions schemes such as GCM (Galois/Counter Mode) or CCM (Counter with CBC-MAC Mode).
- AES-PRNG is a pseudo random number generator, which is nonetheless seeded with a true random seed from a true internal physical random source

Finally, the EVITA full HSM uses its own independent internal CPU that can directly access its internal RAM and non-volatile memory. Separating the CPU prevents any malicious interference from the application CPU and the application software. The application CPU and its applications, however, can access the EVITA HSM only using the secure EVITA hardware interface which enforces a well-defined access (e.g., to prevent read-out of secret keys).

EVITA medium hardware security module (ECU level)

For vehicle-internal communication protection and integrity enforcement, EVITA deploys a medium HSM as depicted in Figure 4.14. The medium HSM is identical to the full HSM except for stripping the ECC-256 and the WHIRLPOOL cryptographic building blocks and including a little less performing CPU.

Thus, the medium HSM has no hardware acceleration for asymmetric cryptography and hashing. However, it is able to perform some non-time-critical asymmetric cryptographic operations in software, for instance for establishment of shared secrets. As, for efficiency and cost reasons, virtually all internal communication

protection is based on symmetric cryptographic algorithms, leaving out the ECC-256 block is reasonable to save costs and hardware size.

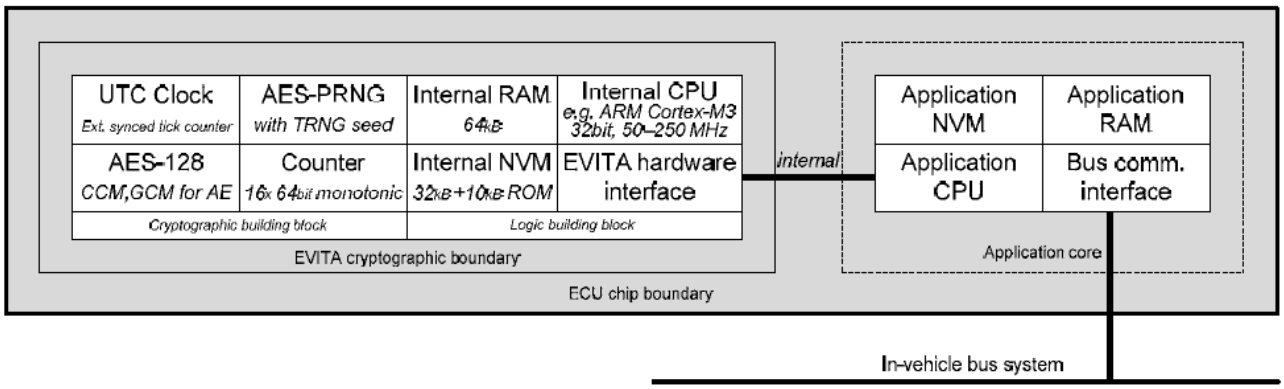


Figure 4.14: EVITA Medium hardware security module (ECU level)

EVITA light hardware security module (sensor/actuator level)

To enable reliable end-to-end protection, sensors with critical input data, critical actuators, and even very small ECUs may also have to be extended with basic security functionalities.

As depicted in Figure 4.15, an EVITA light HSM only has AES-128 symmetric encryption/decryption building block and the corresponding functionally shortened hardware interface.

Hence, the EVITA light HSM is able to fulfil the strict cost and efficiency requirements (e.g., regarding message size, timings, protocol limitations, or processor consumption) that are more typical for sensors and actuators.

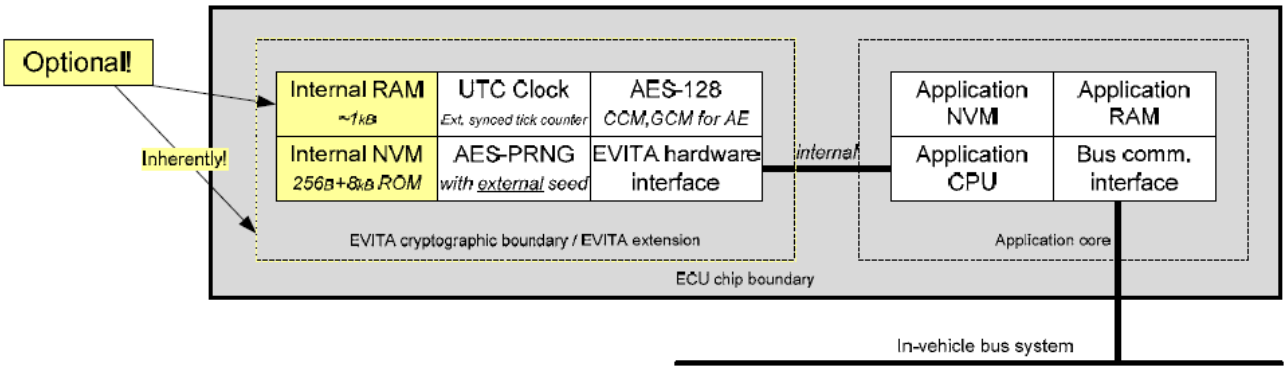


Figure 4.15: EVITA light hardware security module (sensor/actuator level)

Thus, the EVITA light HSM enables sensors and actuators at least to enforce authenticity, integrity, and confidentiality for their communicated data. The necessary shared secret can be established in various manners, for instance, by pre-configuration during manufacturing, by self-initialization (e.g., based on physical unclonable functions) or even by running a key establishment protocol in software at the attached

application processor. It should be noted that the light version of the EVITA HSM, in contrast to the full and medium version, per default does not provide isolated processing and storage. Since the default, light HSM has no internal NVM and no internal RAM; all secret material is fully accessible by the application processor and its application software. However, optionally the light HSM can provide also strong hardware security. In order to provide strong hardware security on sensor/actuator level as well, they propose to include a small internal NVM, a small internal RAM, and an AES-based PRNG with TRNG seed. Thus, the EVITA light HSM could generate, process, and store its secrets (i.e., AES keys and secure boot references) more securely in hardware and hence be able to enforce the cryptographic boundary and secure boot.

Comparison of HSM approaches

HSMs are designed and optimized for different application scenarios. The EVITA HSMs have been specifically designed for e-Safety use cases. Table 4.5 gives a comparison of different HSM approaches.

HSM	EVITA *full*	EVITA *medium*	EVITA *light*	SHE	TPM	Common Smartcard
Boot integrity protection	Auth. & Secure	Auth. & Secure	Auth. & Secure (opt.)	Secure	Auth	None
HW crypto algorithms (incl. key generation)	ECDSA,ECDH, AES/MAC, WHIRLPOOL/ HMAC	ECDSA,ECDH, AES/MAC, WHIRLPOOL/ HMAC	AES/MAC	AES/MAC	RSA, SHA-1/ HMAC	ECC, RSA, AES, 3DES, MAC, SHA-x..
HW crypto acceleration	ECC,AES, WHIRLPOOL	AES	AES	AES	None	None
Internal CPU	Program-mable	Program-mable	None	None / Preset	Preset	Program-mable
RNG	PRNG w/ TRNG seed	PRNG w/ TRNG seed	PRNG w/ ext. seed	PRNG w/ TRNG seed	TRNG	TRNG
Counter	16x64bit	16x64bit	None	None	4x32bit	Yes
Internal NVM	Yes	Yes	Optional	Yes	Indirect (via SRK)	Yes
Internal Clock	Yes w/ ext. UTC sync	Yes w/ ext. UTC sync	Yes w/ ext. UTC sync	No	No	No
Parallel Access	Multiple sessions	Multiple sessions	Multiple sessions	No	Multiple sessions	No
Tamper Protection	Indirect (passive, part of ASIC)	Indirect (passive, part of ASIC)	Indirect (passive, part of ASIC)	Indirect (passive, part of ASIC)	Yes (mfr. dep.)	Yes (active, up to EAL5)

Table 4.5: Comparison of HSM approaches

4.2 Comparison table

Now, we have sufficient knowledge to understand the choices made by various companies in the automotive sector about the security in their microcontrollers. In the following table, we can see some products with their features with more attention on their security aspects

State of art of automotive security microcontrollers

List of acronyms and abbreviations

AES = Advanced Encryption Standard
CMAC = Cipherbased Message Authentication Code
CBC = Cipher-block chaining
ECB = Electronic Code Book
RNG = Random Number Generator
PRNG = Pseudo Random Number Generator
TRNG = True Random Number Generator
ECC = Elliptic Curve Cryptography
SHA = Secure Hash Algorithm
MD5 = Message Digest algorithm 5
ECDSA = The Elliptic Curve Digital Signature Algorithm

Pre-Production
/n production

	Company	Family	Microcontroller (MCU)	Security module	Reference Standard	Principal Features
	Freescale	32-bit Qorivva micro controllers	MPC564xB-C	CSE Option (crypto service engine)	Secure Hardware Extension (SHE) functional specification Version 1.1	ECB and the CBC mode secure key storage, AES encryption, secure boot, AES CMAC authentication and RNG (PRNG and TRNG)
	Freescale	32-bit Qorivva micro controllers	MPC5748G	Hardware Security Module (HSM)		AES cryptographic algorithms, secure memory and secure boot capabilities
	Fujitsu	Fujitsu Cortex R4 Family	MB9EF226 (Titan)	Secure Hardware Extension (SHE) module	Secure Hardware Extension (SHE) functional specification 1.1	AES-128, ECB and CBC modes , CMAC, Implements Miyaguchi-Preneel compression function, RNG,Secure key storage implemented in EEFLASH,Secure boot mode

	Infineon	AUDO MAX	TC1798, TC1793, TC1791	Secure Hardware Extension (SHE)	Secure Hardware Extension (SHE) functional specification 1.1	encrypts access codes with up to 128 bits, Key programming by OEM, TRNG, Manipulation protection, Authentication, Secure boot, Secure hash Secure keys stored in secure Flash, Prevention of access by hardware or software MPU (Memory Protection Unit), Protected memory to store the cryptographic code & keys, „„ AES-128 Hardware Accelerator for symmetric block ciphers, TRNG, Support following crypto standards: ECB, CBC, CTR, OFB, CFB, GCM & XTS
	Infineon	AURIX	TC275T, TC277T	Hardware Security Module (HSM)		
	ST	SPC56 32-bit MCUs	SPC56ECxx, SPC564Bxx	Cryptographic Service Engine (CSE) module	Secure Hardware Extension (SHE) Functional Specification Version 1.1	AES-128 en/decryption, CMAC auth, Secured device boot mode
	Analog Devices	Blackfin	ADSP-BF54x, ADSP-BF52x	Lockbox™ Secure Technology		Elliptic Curve Cryptography (ECC) asymmetric cipher, SHA-1 Secure One-Way HASH2 , ECDSA Signature Verification
	TI	OMAP™ Applications Processors	DRAx (Jacinto)	TI's M-Shield™		RNG, AES and public-key accelerator (PKA), as well as DES/3DES, SHA and MD5 hardware accelerators
	Renesas	RH850/F1x Series		ICU (intelligent cryptographic unit)		encryption circuit that provides stronger security

Table 4.6 Comparison table

What do this comparison table can show to us?

We may note three main factors:

1. The security in the automotive environment is a emerging problem, for this reason there are several solutions at the moment on the market.
2. Since there is no real standard, some companies have tried to readjust products already established in other environments.

3. For the next generation of microcontrollers is expected a certain degree of standardization on the line of the project EVITA.

Therefore, it seems that the EVITA project is a first step towards a global standardization of the security problem in the automotive environment

This brief review concludes the first part of this thesis activity, whose main purpose is to shed light on how the various companies operating in the automotive sector are gearing up to enter in the market with competitive products that include security features.

Now the activity, always following the Renesas' directives, will be restricted to the security in a CAN to CAN communication, and in particular on the implementation of an IP that ensure the Privacy aspect in a CAN bus.

The Renesas, after a power point presentation and a conference call on these topics, has confined the security problem on the CAN bus communications. At this stage their goal, was the insertion in an existing CAN IP of a certain level of security.

To complete this task successfully we need to follow the following steps:

1) A brief study of the CAN standard to understand the peculiarities and the weakness as regards the security aspect.
2) An analysis of the level of security required.
3) Implementation of an IP cell in Verilog language that meets the level of security identified in the previous analysis

The next part of this thesis activity, will follow the previous steps.

Chapter 5
CAN protocol

The Controller Area Network (CAN) is a serial communications protocol, which efficiently supports distributed real-time control with a very high level of security.

Its domain of application ranges from high speed networks to low cost multiplex wiring. In automotive electronics, engine control units, sensors, anti-skid-systems, etc. are connected using CAN with bitrates up to 1 Mbit/s. At the same time, it is cost effective to build into vehicle body electronics, e.g. lamp clusters, electric windows etc. to replace the wiring harness otherwise required.

5.1 Basic Concepts

To achieve design transparency and implementation flexibility CAN has been subdivided into different layers.

- the (CAN-) object layer
- the (CAN-) transfer layer
- the physical layer.

The object layer and the transfer layer comprise all services and functions of the data link layer defined by the ISO/OSI model. The scope of the object layer includes

- finding which messages are to be transmitted,
- deciding which messages received by the transfer layer are actually to be used,
- providing an interface to the application layer related hardware.

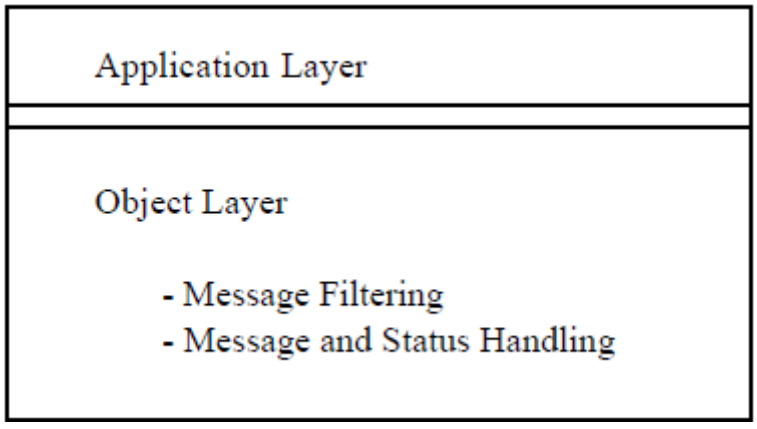
There is much freedom in defining object handling. The scope of the transfer layer mainly is the transfer protocol, i.e. controlling the framing, performing arbitration, error checking, error signaling and fault confinement. Within the transfer layer, it is decided whether the bus is free for starting a new transmission or whether a reception is just starting. In addition, some general features of the bit timing are regarded as part of the transfer layer. It is in the nature of the transfer layer that there is no freedom for modifications.

The scope of the physical layer is the actual transfer of the bits between the different nodes with respect to all electrical properties. Within one network the physical layer, of course has to be the same for all nodes. There may be, however, much freedom in selecting a physical layer.

CAN has the following properties

- prioritization of messages
- guarantee of latency times
- configuration flexibility
- multicast reception with time synchronization
- system wide data consistency
- multimaster
- error detection and error signaling
- automatic retransmission of corrupted messages as soon as the bus is idle again.
- distinction between temporary errors and permanent failures of nodes and autonomous switching off of defect nodes.

5.1.1Layered Structure of a CAN Node



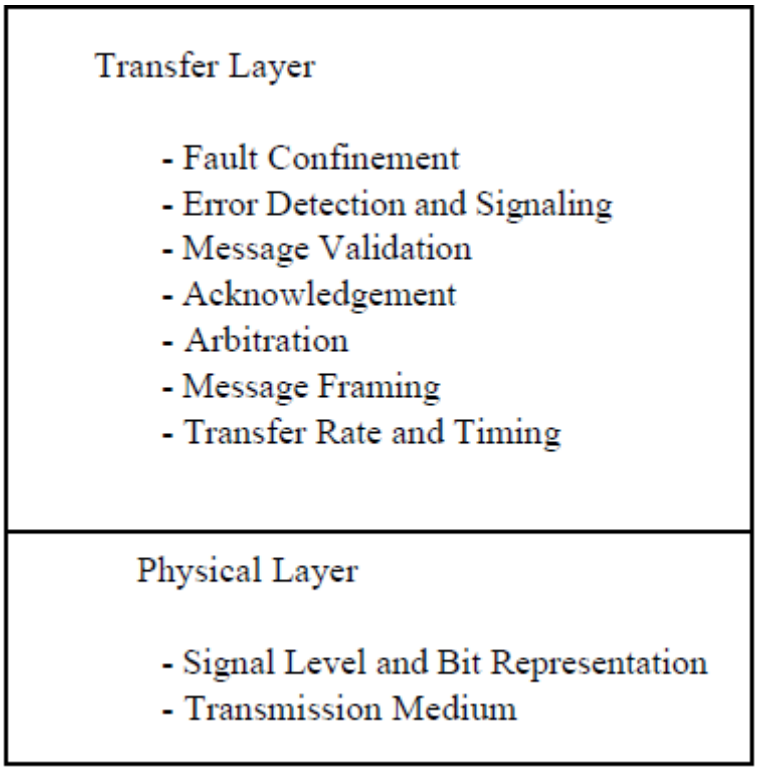


Figure 5.1 layered structure of a CAN node representation

The Physical Layer defines how signals are actually transmitted. Within this specification, the physical layer is not defined so as to allow transmission medium and signal level implementations to be optimized for their application.

The Transfer Layer represents the kernel of the CAN protocol. It presents messages received to the object layer and accepts messages to be transmitted from the object layer. The transfer layer is responsible for bit timing and synchronization, message framing, arbitration, acknowledgement, error detection and signalling, and fault confinement.

The Object Layer is concerned with message filtering as well as status and message handling.

The scope of the specification is to define the transfer layer and the consequences of the CAN protocol on the surrounding layers.

5.1.2 Frame Structure

Message transfer is manifested and controlled by four different frame types: A DATA FRAME carries data from a transmitter to the receivers.

A REMOTE FRAME is transmitted by a bus unit to request the transmission of the DATA FRAME with the same IDENTIFIER.

An ERROR FRAME is transmitted by any unit on detecting a bus error.

An OVERLOAD FRAME is used to provide for an extra delay between the preceding and the succeeding DATA or REMOTE FRAMEs.

For our purposes is sufficient examine the DATA FRAME. A DATA FRAME is composed of seven different bit fields: START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD, CRC FIELD, ACK FIELD, and END OF FRAME. The DATA FIELD can be of length zero.

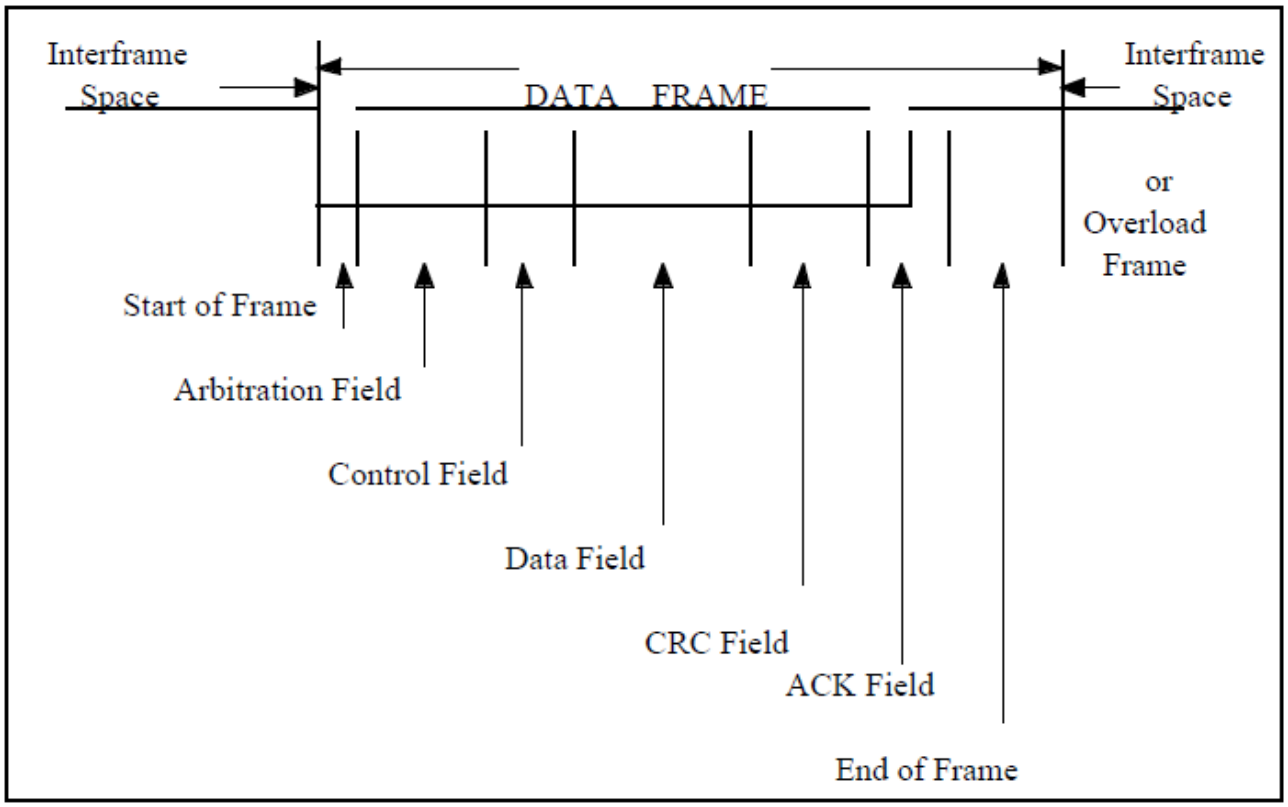


Figure 5.2 Data Frame structure

5.2 CAN security features

CAN provides users with a special kind of service for data transfer, named safety service, which includes the following procedures: error detection, error signaling, and self-checking. For error detecting the following measures are taken into account: Monitoring (transmitters compare the bit levels to be transmitted with the bit levels detected on the bus), Cyclic Redundancy Check (CRC), Bit stuffing, and Message Frame Check.

The CRC FIELD contains the CRC SEQUENCE followed by a CRC DELIMITER.

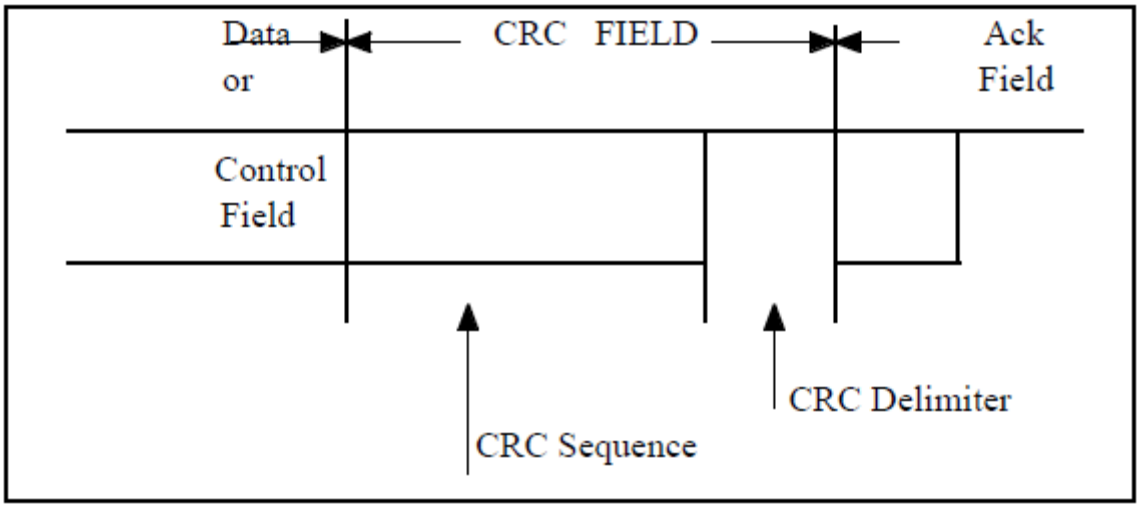


Figure 5.3 CRC structure

CRC SEQUENCE

The frame check sequence is derived from a cyclic redundancy code best suited for frames with bit counts less than 127 bits (BCH Code). In order to carry out the CRC calculation the polynomial to be divided is defined as the polynomial, the coefficients of which are given by the de-stuffed bit stream consisting of START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD (if present) and, for the 15 lowest coefficients, by 0. This polynomial is divided (the coefficients are calculated modulo 2) by the generator-polynomial:

$$X^{15} + X^{14} + X^{10} + X^{8} + X^{7} + X^{4} + X^{3} + 1.$$

The remainder of this polynomial division is the CRC SEQUENCE transmitted over the bus. In order to implement this function, a 15-bit shift register CRC_RG (14:0) can be used. If NXTBIT denotes the next bit of the bit stream, given by the de-stuffed bit sequence from START OF FRAME until the end of the DATA FIELD, the CRC SEQUENCE is calculated as follows

```
CRC_RG = 0;                                          // initialize shift register
REPEAT
        CRCNXT = NXTBIT EXOR CRC_RG(14);
        CRC_RG(14:1) = CRC_RG(13:0);          // shift left by
    CRC_RG(0) = 0;                                  // 1 position
    IF CRCNXT THEN
            CRC_RG(14:0) = CRC_RG(14:0) EXOR (4599hex);
    ENDIF
UNTIL (CRC SEQUENCE starts or there is an ERROR condition)
```

After the transmission / reception of the last bit of the DATA FIELD, CRC_RG contains the CRC sequence.

CRC DELIMITER

The CRC SEQUENCE is followed by the CRC DELIMITER which consists of a single 'recessive' bit.

Nevertheless, these procedures do not provide the integrity service, which can be achieved by using cryptographic mechanisms such as one-way hash functions.

Furthermore, CAN does not provide the confidentiality service. All the data transfers are made in plaintext and in broadcast mode. Therefore, in order to avoid possible passive and active attacks from intruders that have managed to gain access to the bus, CAN must instrument a data confidentiality service via feasible encryption/decryption schemes.

In short the CAN protocol is practically devoid of security mechanisms, the main reason behind this question is that the CAN protocol, like many other Communication protocols, is a low level protocol and defines only two OSI layers, the physical, and the data link (Logical Link Control and Medium Access Control) layers. Thus, the security mechanism are demanded to the higher levels.

5.3 Analysis of the security level requested

This analysis is very important because introduce the security in automotive environment involves various problems:

1. deterioration of performance
2. increased costs
3. increased time to market and many others...

Therefore, it is important to properly evaluate the level of security required for the given application

How we can assess the security level?

To justify a certain level of security we need to know:

1. the scope of the application domain
2. tools necessary to carry out the attack
3. who may be interested in bringing the attack
4. the benefits that can be achieved by the attacker
5. the damage that the attacker can cause

The concept is simple, also introducing a suitable level of security a malicious user can always compromise the system, and the goal is to make sure that the attack (necessary to compromise the system) becomes too burdensome for the attacker in terms of cost and time.

This analysis requires great knowledge in various fields and a lot of time to complete it. Then to continue with the activities of the thesis, it was necessary to rely on the work done by the employees of the EVITA project.

The only step that remains to be done is to identify the “Functional Domain” of the CAN bus.

The division into levels of security and countermeasures to be taken in the project EVITA refers to particular use cases as we have seen in the corresponding chapter, then we need to understand what possible use cases can be referred to the CAN bus.

To begin this analysis we can see initially a general scheme of the interconnections inside of a modern car.

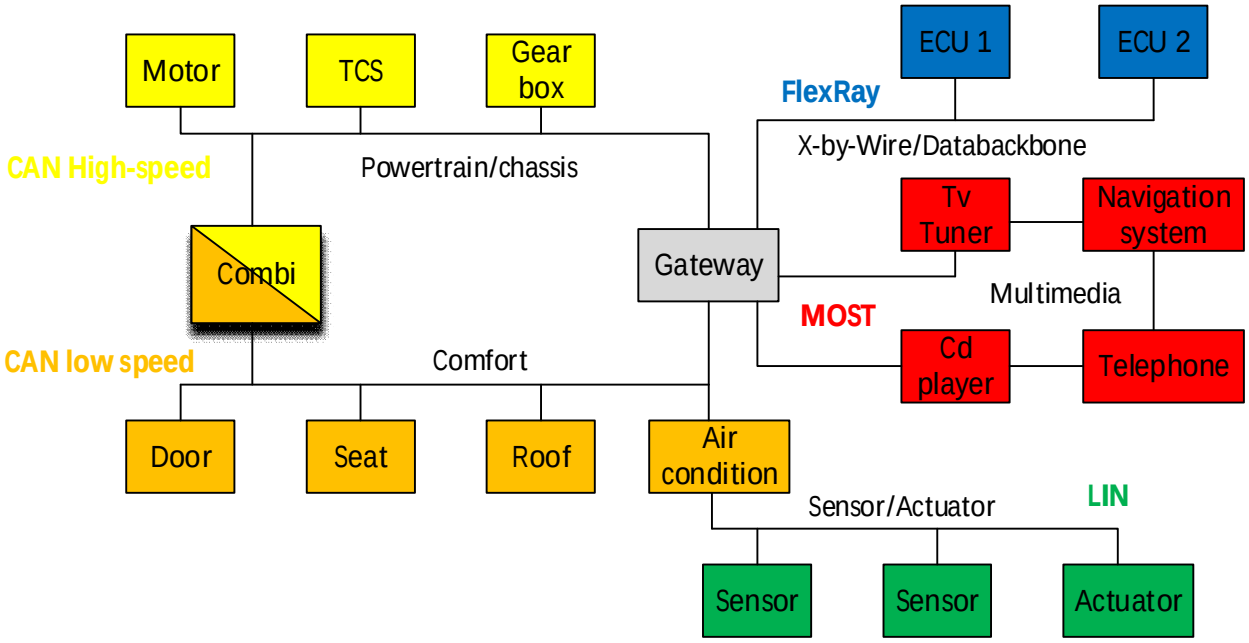


Figure 5.4 example of interconnection in an automotive environment

Once we figured out what is attached to the CAN network, we can compare this functional domain with the security level assigned to a particular domain in the EVITA project.

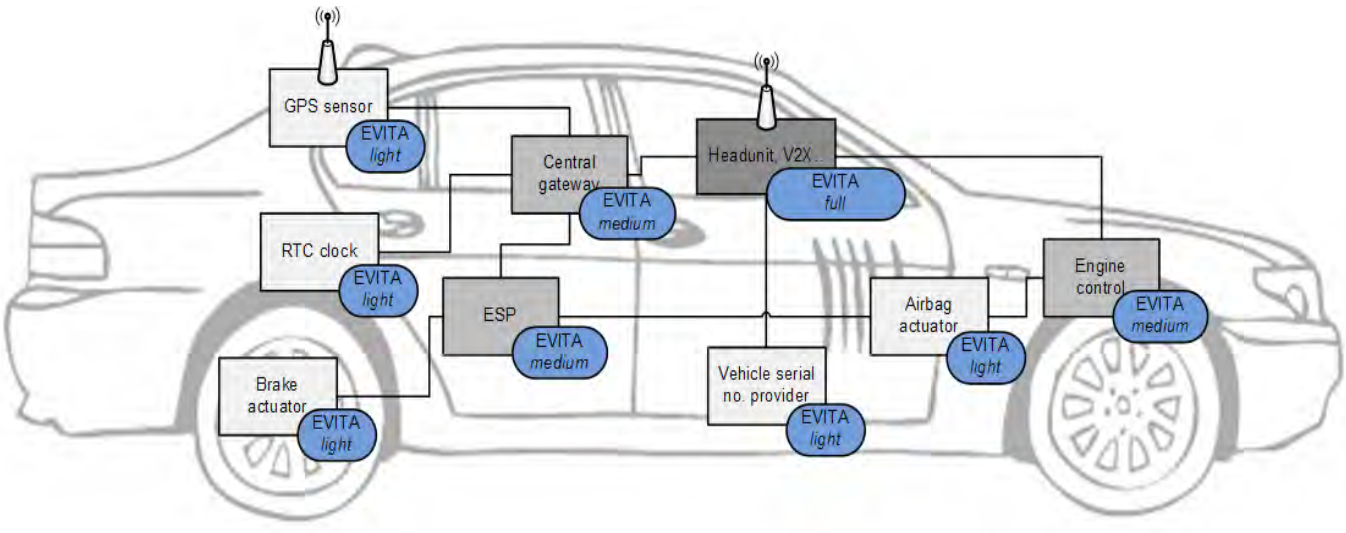


Figure 5.5 example of applications of the three types of the HSMs

Without going into excessive details we can see that the functional domain of the CAN bus, is covered by the medium and the light EVITA.

In the following comparison table are the main features of the two modules (highlighted columns)

	Full	Medium	Light	SHE	TPM	SmC
Cryptographic algorithms						
ECC/RSA	■/■	■/■	□/□	□/□	□/■	⊞/⊞
AES/DES	■/⊞	■/⊞	■/□	■/□	□/□	⊞/⊞
WHIRLPOOL/SHA	■/■	■/■	□/□	□/□	□/■	⊞/⊞
Hardware acceleration						
ECC/RSA	■/□	□/□	□/□	□/□	□/□	□/□
AES/DES	■/□	■/□	■/□	■/□	□/□	□/□
WHIRLPOOL/SHA	■/□	□/□	□/□	□/□	□/□	□/□
Security features						
Secure/authenticated boot	■/■	■/■	⊞/⊞	■/□	□/■	□/□
Key AC per use/bootstrap	■/■	■/■	■/⊞	□/■	⊞/■	□/□
PRNG with TRNG seed	■	■	■	■	■	■
Monotonic counters 32/64 bit	■/■	■/■	□/□	□/□	■/□	□/□
Tick/UTC-synced clock	■/■	■/■	■/■	□/□	□/□	□/□
Internal processing						
Programmable/preset CPU	■/⊞	■/⊞	□/⊞	□/■	□/■	⊞/⊞
Internal V/NV (key) memory	■/■	■/■	⊞/⊞	■/■	■/□	■/□
Asynchronous/parallel IF	■/⊞	■/□	■/□	■/□	□/□	□/□
Annotation: ■ = available, □ = not available, ⊞ = partly or optionally available						

Table 5.1 comparison of the various security solutions

As regards the security, we can see that both the modules have and hardware implementation of the AES (Advanced Encryption Standards). The AES seems to be the best tool to ensure a certain level of security in the CAN functional domain. For this and many other reasons, a first step, to ensure a certain level of security in a CAN bus, is the realization of a hardware module that performs the AES algorithm to encrypt/decrypt a message.

Thus, in the next chapter, we will see briefly the AES standard to understand how it operates and after, how it can be implemented in hardware.

Chapter 6
Advance Encryption Standard

The Advanced Encryption Standard (AES) is the most widely used symmetric cipher today. Even though the term “Standard” in its name only refers to US government applications, the AES block cipher is also mandatory in several industry standards and is used in many commercial systems. Among the commercial standards that include AES are the Internet security standard IPsec, TLS, the Wi-Fi encryption standard IEEE 802.11i, the secure shell network protocol SSH (Secure Shell), the Internet phone Skype and numerous security products around the world. To date, there are no attacks better than brute-force known against AES.

6.1 Introduction

In 1999, the US National Institute of Standards and Technology (NIST) indicated that DES should only be used for legacy systems and instead triple DES (3DES) should be used. Even though 3DES resists brute-force attacks with today’s technology, there are several problems with it.

First, it is not very efficient with regard to software implementations. DES is already not particularly well suited for software and 3DES is three times slower than DES. Another disadvantage is the relatively short block size of 64 bits, which is a drawback in certain applications, e.g., if one wants to build a hash function from a block cipher. Finally, if one is worried about attacks with quantum computers, which might become reality in a few decades, key lengths on the order of 256 bits are desirable. All these consideration led NIST to the conclusion that an entirely new block cipher was needed as a replacement for DES. In 1997, NIST called for proposals for a new Advanced Encryption Standard (AES).

Unlike the DES development, the selection of the algorithm for AES was an open process administered by NIST. In three subsequent AES evaluation rounds, NIST and the international scientific community discussed the advantages and disadvantages of the submitted ciphers and narrowed down the number of potential candidates. In 2001, NIST declared the block cipher Rijndael as the new AES and published it as a final standard (FIPS PUB 197). Rijndael was designed by two young Belgian cryptographers.

Within the call for proposals, the following requirements for all AES candidate submissions were mandatory:

- block cipher with 128 bit block size
- three key lengths must be supported: 128, 192 and 256 bit
- security relative to other submitted algorithms
- efficiency in software and hardware

The invitation for submitting suitable algorithms and the subsequent evaluation of the successor of DES was a public process. A compact chronology of the AES selection process is given here:

- The need for a new block cipher was announced on January 2, 1997, by NIST.
- A formal call for AES was announced on September 12, 1997.
- Fifteen candidate algorithms were submitted by researchers from several countries by August 20, 1998.
- On August 9, 1999, five finalist algorithms were announced:
    - RC6 by RSA Laboratories
    - Mars by IBM Corporation
    - Rijndael, by Joan Daemen and Vincent Rijmen
    - Serpent, by Ross Anderson, Eli Biham and Lars Knudsen
    - Twofish, by Bruce Schneier, John Kelsey, Doug Whiting, DavidWagner, Chris Hall and Niels Ferguson

6.2 Overview of the AES Algorithm

The AES cipher is almost identical to the block cipher Rijndael. The Rijndael block and key size vary between 128, 192 and 256 bits. However, the AES standard only calls for a block size of 128 bits. Hence, only Rijndael with a block length of 128 bits is known as the AES algorithm. In the remainder of this chapter, we only discuss the standard version of Rijndael with a block length of 128 bits.

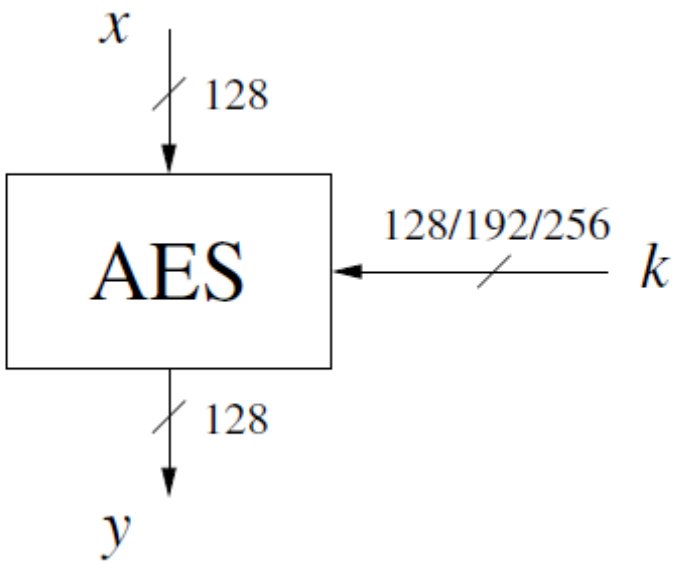


Figure 6.1 AES input/output parameters

As mentioned previously, three key lengths must be supported by Rijndael as this was an NIST design requirement. The number of internal rounds of the cipher is a function of the key length according to Table 6.1.

key lengths	# rounds = n_r
128 bit	10
192 bit	12
256 bit	14

Table 6.1 Key lengths and number of rounds for AES

In contrast to DES, AES does not have a Feistel structure. Feistel networks do not encrypt an entire block per iteration, e.g., in DES, 64/2 = 32 bits are encrypted in one round. AES, on the other hand, encrypts all 128 bits in one iteration. This is one reason why it has a comparably small number of rounds. AES consists of so-called layers. Each layer manipulates all 128 bits of the data path. The data path is also referred to as the state of the algorithm. There are only three different types of layers. Each round, with the exception of the first, consists of all three layers as shown in Fig. 6.2: the plaintext is denoted as x,

the cipher text as y and the number of rounds as nr. Moreover, the last round nr does not make use of the MixColumn transformation, which makes the encryption and decryption scheme symmetric.

We continue with a brief description of the layers:

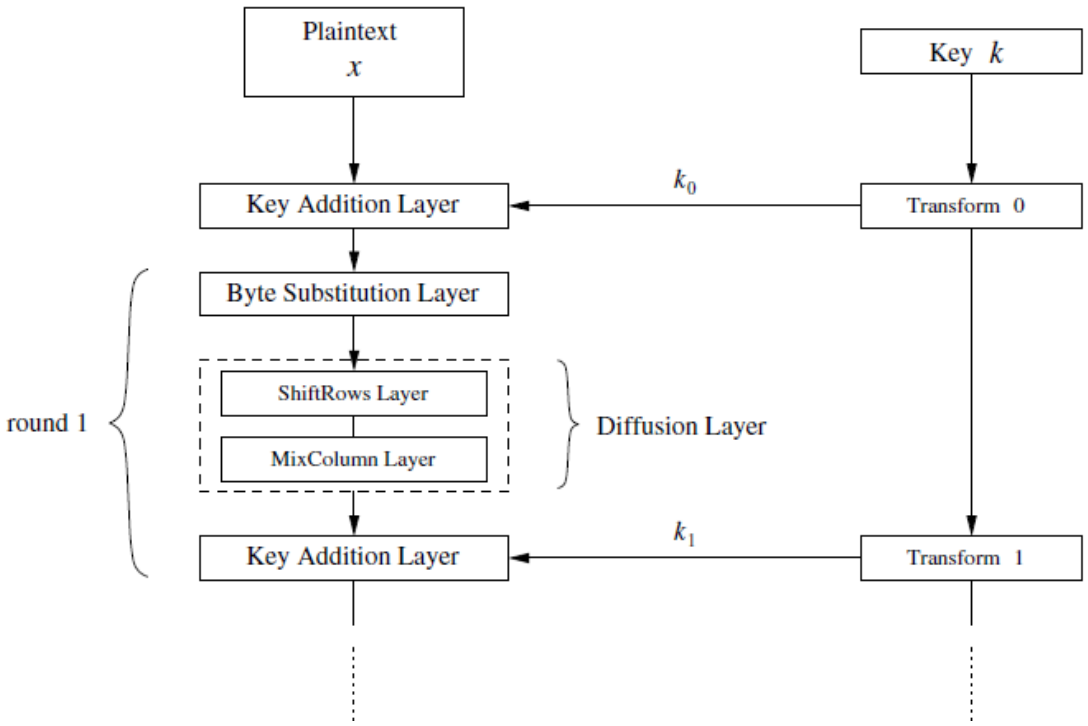
Key Addition layer A 128-bit round key, or sub key, which has been derived from the main key in the key schedule, is XORed to the state.

Byte Substitution layer (S-Box) Each element of the state is nonlinearly transformed using lookup tables with special mathematical properties. This introduces confusion to the data, i.e., it assures that changes in individual state bits propagate quickly across the data path.

Diffusion layer It provides diffusion over all state bits. It consists of two sub layers, both of which perform linear operations:

- The ShiftRows layer permutes the data on a byte level.
- The MixColumn layer is a matrix operation, which combines (mixes) blocks of four bytes.

Similar to DES, the key schedule computes round keys, or sub keys, (k_0, k1, . . . ,k_{nr}) from the original AES key. Before we describe the internal functions of the layers, we have to introduce a new mathematical concept, namely Galois fields. Galois field computations are needed for all operations within the AES layers.



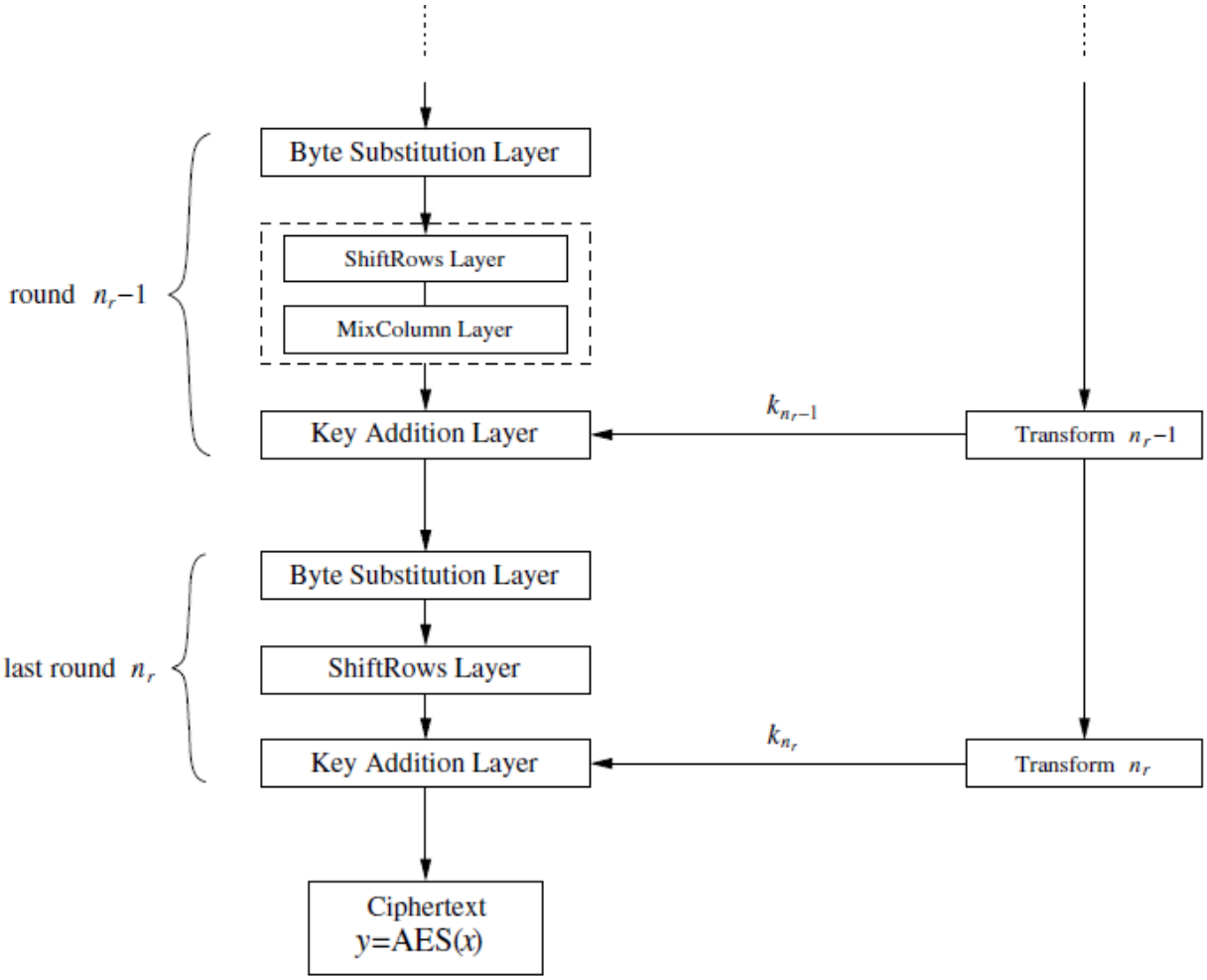


Figure. 6.2 AES encryption block diagram

6.3 Some Mathematics: A Brief Introduction to Galois Fields

In AES, Galois field arithmetic is used in most layers, especially in the S-Box and the MixColumn layer. Hence, for a deeper understanding of the internals of AES, we provide an introduction to Galois fields as needed for this purpose before we continue with the algorithm. A strong background on Galois fields is not required for a basic understanding of AES.

6.3.1 Existence of Finite Fields

A finite field, sometimes also called Galois field, is a set with a finite number of elements. Roughly speaking, a Galois field is a finite set of elements in which we can add, subtract, multiply and invert. Before we introduce the definition of a field, we first need the concept of a simpler algebraic structure, a group.

Definition 6.1 **Group**

A group is a set of elements G together with an operation $\circ$, which combines two elements of G. A group has the following properties:

1. The group operation $\circ$ is closed. That is, for all $a,b, \in G$, it holds that $a \circ b = c \in G$.
2. The group operation is associative. That is, $a \circ (b \circ c) = (a \circ b) \circ c$ for all $a,b,c \in G$.
3. There is an element $1 \in G$, called the neutral element (or identity element), such that $a \circ 1 = 1 \circ a = a$ for all $a \in G$.
4. For each $a \in G$ there exists an element $a-1 \in G$, called the inverse of a, such that $a \circ a-1 = a-1 \circ a = 1$.
5. A group G is abelian (or commutative) if, furthermore, $a \circ b = b \circ a$ for all $a,b \in G$.

Roughly speaking, a group is set with one operation and the corresponding inverse operation. If the operation is called addition, the inverse operation is subtraction; if the operation is multiplication, the inverse operation is division (or multiplication with the inverse element).

Example: The set of integers $Zm = \{0,1, \ldots ,m-1\}$ and the operation addition modulo m form a group with the neutral element 0. Every element a has an inverse –a such that $a+(-a) = 0 \bmod m$. Note that this set does not form a group with the operation multiplication because most elements a do not have an inverse such that $a*a-1 = 1 \bmod m$.

In order to have all four basic arithmetic operations (i.e., addition, subtraction, multiplication, division) in one structure, we need a set which contains an additive and a multiplicative group. This is what we call a field.

Definition 6.2 **Field**

A field F is a set of elements with the following properties:

1. All elements of F form an additive group with the group operation "+" and the neutral element 0.
2. All elements of F except 0 form a multiplicative group with the group operation "×" and the neutral element 1.
3. When the two group operations are mixed, the distributivity law holds, i.e., for all a,b,c ∈ F: a(b+c) = (ab)+(ac).

Example: The set R of real numbers is a field with the neutral element 0 for the additive group and the neutral element 1 for the multiplicative group. Every real number a, has an additive inverse, namely $-a$, and every nonzero element a has a multiplicative inverse $1/a$.

In cryptography, we are usually interested in fields with a finite number of elements, which we call finite fields or Galois fields. The number of elements in the field is called the order or cardinality of the field. Of fundamental importance is the following theorem:

A field with order m only exists if m is a prime power, i.e., $m = p^n$, for some positive integer n and prime integer p. p is called the characteristic of the finite field.

This theorem implies that there are, for instance, finite fields with 11 elements, or with 81 elements (since $81 = 3^4$) or with 256 elements (since $256 = 2^8$, and 2 is a prime). However, there is no finite field with 12 elements since $12 = 2^2 \cdot 3$, and 12 is thus not a prime power.

6.3.2 Prime Fields

The most intuitive examples of finite fields are fields of prime order, i.e., fields with n=1. Elements of the field GF (p) can be represented by integers $0,1, \ldots, p-1$. The two operations of the field are modular integer addition and integer multiplication modulo p.

Theorem 6.1 Let p be a prime. The integer ring Z_p is denoted as GF(p) and is referred to as a prime field, or as a Galois field with a prime number of elements. All nonzero elements of GF(p) have an inverse. Arithmetic in GF(p) is done modulo p.

In order to do arithmetic in a prime field, we have to follow the rules for integer rings: Addition and multiplication are done modulo p, the additive inverse of any element a is given by $a+(-a) = 0 \bmod p$, and the multiplicative inverse of any nonzero element a is defined as $a \cdot a^{-1} = 1$. Let's have a look at an example of a prime field.

In AES the finite field contains 256 elements and is denoted as GF (2^8). This field was chosen because each of the field elements can be represented by one byte. For the S-Box and MixColumn transforms, AES treats every byte of the internal data path as an element of the field GF (2^8) and manipulates the data by performing arithmetic in this finite field.

Example: We consider the finite field GF (5) = {0, 1, 2, 3, 4}. The tables below describe how to add and multiply any two elements, as well as the additive and multiplicative inverse of the field elements. Using these tables, we can perform all calculations in this field without using modular reduction explicitly

addition

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

additive inverse

$-0 = 0$
$-1 = 4$
$-2 = 3$
$-3 = 2$
$-4 = 1$

multiplication

×	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

multiplicative inverse

0^{-1} does not exist
$1^{-1} = 1$
$2^{-1} = 3$
$3^{-1} = 2$
$4^{-1} = 4$

A very important prime field is GF (2), which is the smallest finite field that exists. Let's have a look at the multiplication and addition tables for the field.

addition

+	0	1
0	0	1
1	1	0

multiplication

×	0	1
0	0	0
1	0	1

GF(2) addition, i.e., modulo 2 addition, is equivalent to an XOR gate and the GF(2) multiplication is equivalent to the logical AND gate. The field GF(2) is important for AES.

6.3.3 Extension Fields GF (2^m)

In AES, the finite field contains 256 elements and is denoted as GF (2^8). This field was chosen because each of the field elements can be represented by one byte. For the S-Box and MixColumn transforms, AES treats every byte of the internal data path as an element of the field GF (2^8) and manipulates the data by performing arithmetic in this finite field. However, if the order of a finite field is not prime and 2^8 is clearly not a prime, the addition and multiplication operation cannot be represented by addition and multiplication of integers modulo 2^8. Such fields with m > 1 are called extension fields. In order to deal with extension fields we need (1) a different notation for field elements and (2) different rules for performing arithmetic with the elements. We will see in the following that elements of extension fields can be represented as polynomials, and that computation in the extension field is achieved by performing a certain type of polynomial arithmetic.

In extension fields GF (2^m) elements are not represented as integers but as polynomials with coefficients in GF (2). The polynomials have a maximum degree of m−1, so that there are m coefficients in total for every element. In the field GF (2^8), which is used in AES, each element A ∈ GF (2^8) is thus represented as:

$$A(x) = a_7x^7 + \cdots + a_1x + a_0, \quad a_i \in GF(2) = \{0, 1\}.$$

Note that there are exactly 256 = 2^8 such polynomials. The set of these 256 polynomials is the finite field GF (2^8). It is also important to observe that every polynomial can simply be stored in digital form as an 8-bit vector

In particular, we do not have to store the factors x7, x6, etc. It is clear from the bit positions to which power x^i each coefficient belongs.

6.3.4 Addition and Subtraction in GF (2^m)

Let's now look at addition and subtraction in extension fields. The key addition layer of AES uses addition. It turns out that these operations are straightforward. They are simply achieved by performing standard polynomial addition and subtraction: We merely add or subtract coefficients with equal powers of x. The coefficient additions or subtractions are done in the underlying field GF (2).

Definition 6.3 **Extension field addition and subtraction**

Let $A(x), B(x) \in GF(2^m)$. The sum of the two elements is then computed according to:

$$C(x) = A(x) + B(x) = \sum_{i=0}^{m-1} c_i x^i, \quad c_i \equiv a_i + b_i \bmod 2$$

and the difference is computed according to:

$$C(x) = A(x) - B(x) = \sum_{i=0}^{m-1} c_i x^i, \quad c_i \equiv a_i - b_i \equiv a_i + b_i \bmod 2.$$

Note that we perform modulo 2 addition (or subtraction) with the coefficients. Addition and subtraction modulo 2 are the same operation. Moreover, addition modulo 2 is equal to bitwise XOR. Let's have a look at an example in the field GF (2^8) which is used in AES:

$$\begin{array}{rl} A(x) = & x^7 + x^6 + x^4 + \quad 1 \\ B(x) = & \quad x^4 + x^2 + 1 \\ \hline C(x) = & x^7 + x^6 + \quad x^2 \end{array}$$

Note that if we computed the difference of the two polynomials A(x)−B(x) from the example above, we would get the same result as for the sum.

6.3.5 Multiplication in GF(2^m)

Multiplication in GF (2^8) is the core operation of the MixColumn transformation of AES. In a first step, two elements (represented by their polynomials) of a finite field GF (2^m) are multiplied using the standard polynomial multiplication rule:

$$A(x) \cdot B(x) = (a_{m-1}x^{m-1} + \cdots + a_0) \cdot (b_{m-1}x^{m-1} + \cdots + b_0)$$
$$C'(x) = c'_{2m-2}x^{2m-2} + \cdots + c'_0,$$

Where:

$$\begin{aligned} c'_0 &= a_0 b_0 \bmod 2 \\ c'_1 &= a_0 b_1 + a_1 b_0 \bmod 2 \\ &\vdots \\ c'_{2m-2} &= a_{m-1} b_{m-1} \bmod 2. \end{aligned}$$

Note that all coefficients a_i, b_i and c_i are elements of GF (2), and that coefficient arithmetic is performed in GF(2). In general, the product polynomial C(x) will have a degree higher than m−1 and has to be reduced. The basic idea is an approach similar to the case of multiplication in prime fields: in GF (p), we multiply the two integers, divide the result by a prime, and consider only the remainder. Here is what we are doing in extension fields: The product of the multiplication is divided by a certain polynomial, and we consider only the remainder after the polynomial division. We need irreducible polynomials for the module reduction. Irreducible polynomials are roughly comparable to prime numbers, i.e., their only factors are 1 and the polynomial itself.

Definition 6.4 **Extension field multiplication**

Let $A(x), B(x) \in GF(2^m)$ *and let*

$$P(x) \equiv \sum_{i=0}^{m} p_i x^i, \quad p_i \in GF(2)$$

be an irreducible polynomial. Multiplication of the two elements $A(x), B(x)$ *is performed as*

$$C(x) \equiv A(x) \cdot B(x) \bmod P(x).$$

Thus, every field GF (2^m) requires an irreducible polynomial P(x) of degree m with coefficients from GF(2). Note that not all polynomials are irreducible. For example, the polynomial x^4+x^3+x+1 is reducible since

$$x^4 + x^3 + x + 1 = (x^2 + x + 1)(x^2 + 1)$$

and hence cannot be used to construct the extension field GF(2^4). For AES, the irreducible polynomial

$$P(x) = x^8 + x^4 + x^3 + x + 1$$

is used. It is part of the AES specification.

Example: We want to multiply the two polynomials $A(x) = x^3 + x^2 + 1$ and $B(x) = x^2+x$ in the field GF (2^4). The irreducible polynomial of this Galois field is given as $P(x) = x^4 + x + 1.$

The plain polynomial product is computed as: $C'(x) = A(x) \cdot B(x) = x^5 + x^3 + x^2 + x.$

We can now reduce C(x) using the polynomial division method

1 0 1 1 1 0	($x^5+x^3+x^2+x$)	+	(xor)
1 0 0 1 1	(x^4+x+1)	=	
0 0 1 0 0 0	(x^3)		

It is important not to confuse multiplication in GF (2^m) with integer multiplication, especially if we are concerned with software implementations of Galois fields. Recall that the polynomials, i.e., the field elements, are normally stored as bit vectors in the computers. If we look at the multiplication from the previous example, the following very atypical operation is being performed on the bit level:

$$\begin{array}{ccccc} A & \cdot & B & = & C \\ (x^3+x^2+1) & \cdot & (x^2+x) & = & x^3 \\ (1\,1\,0\,1) & \cdot & (0\,1\,1\,0) & = & (1\,0\,0\,0). \end{array}$$

This computation is not identical to integer arithmetic. If the polynomials are interpreted as integers, i.e., $(1101)_2 = 13_{10}$ and $(0110)_2 = 6_{10}$, the result would have been $(1001110)_2 = 78_{10}$, which is clearly not the same as the Galois field multiplication product. Hence, even though we can represent field elements as integers data types, we cannot make use of the integer arithmetic provided

6.3.6 Inversion in GF (2^m)

Inversion in GF (2^8) is the core operation of the Byte Substitution transformation, which contains the AES S-Boxes. For a given finite field GF(2^m) and the corresponding irreducible reduction polynomial P(x), the inverse A^{-1} of a nonzero element $A \in GF(2^m)$ is defined as:

$$A^{-1}(x) \cdot A(x) = 1 \bmod P(x).$$

For small fields (in practice this often means fields with 2^{16} or fewer elements) lookup tables which contain the precomputed inverses of all field elements are often used. Table 6.1 shows the values which are used within the S-Box of AES. The table contains all inverses in GF (2^8) modulo $P(x) = x^8 + x^4 + x^3 + x + 1$ in hexadecimal notation. A special case is the entry for the field element 0, for which an inverse does not exist. However, for the AES S-Box, a substitution table is needed that is defined for every possible input value. Hence, the designers defined the S-Box such that the input value 0 is mapped to the output value 0.

	Y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	01	8D	F6	CB	52	7B	D1	E8	4F	29	C0	B0	E1	E5	C7
1	74	B4	AA	4B	99	2B	60	5F	58	3F	FD	CC	FF	40	EE	B2
2	3A	6E	5A	F1	55	4D	A8	C9	C1	0A	98	15	30	44	A2	C2
3	2C	45	92	6C	F3	39	66	42	F2	35	20	6F	77	BB	59	19
4	1D	FE	37	67	2D	31	F5	69	A7	64	AB	13	54	25	E9	09
5	ED	5C	05	CA	4C	24	87	BF	18	3E	22	F0	51	EC	61	17
6	16	5E	AF	D3	49	A6	36	43	F4	47	91	DF	33	93	21	3B
7	79	B7	97	85	10	B5	BA	3C	B6	70	D0	06	A1	FA	81	82
X 8	83	7E	7F	80	96	73	BE	56	9B	9E	95	D9	F7	02	B9	A4
9	DE	6A	32	6D	D8	8A	84	72	2A	14	9F	88	F9	DC	89	9A
A	FB	7C	2E	C3	8F	B8	65	48	26	C8	12	4A	CE	E7	D2	62
B	0C	E0	1F	EF	11	75	78	71	A5	8E	76	3D	BD	BC	86	57
C	0B	28	2F	A3	DA	D4	E4	0F	A9	27	53	04	1B	FC	AC	E6
D	7A	07	AE	63	C5	DB	E2	EA	94	8B	C4	D5	9D	F8	90	6B
E	B1	0D	D6	EB	C6	0E	CF	AD	08	4E	D7	E3	5D	50	1E	B3
F	5B	23	38	34	68	46	03	8C	DD	9C	7D	A0	CD	1A	41	1C

Table 6.1 Multiplicative inverse table in GF(2^8) for bytes xy used within the AES S-Box

From Table 6.1 the inverse of $x^7+x^6+x = (11000010)2 = (C2)_{hex} = (xy)$

is given by the element in row C, column 2: $(2F)_{hex} = (00101111)_2 = x^5+x^3+x^2+x+1$.

This can be verified by multiplication: $(x^7+x^6+x) \cdot (x^5+ x^3+x^2+x+1) \equiv 1 \bmod P(x)$.

As an alternative to using lookup tables, one can also explicitly compute inverses. The main algorithm for computing multiplicative inverses is the extended Euclidean algorithm, which is not introduced in this thesis activity.

6.4 Internal Structure of AES

In the following, we examine the internal structure of AES. Figure 6.3 shows the graph of a single AES round. The 16-byte input $A_0, \ldots, A15$ is fed byte-wise into the S-Box. The 16-byte output $B_0, \ldots, B_{15}$ is permuted byte-wise in the ShiftRows layer and mixed by the MixColumn transformation c(x). Finally, the 128-bit sub key k_i is XORed with the intermediate result. We note that AES is a byte-oriented cipher.

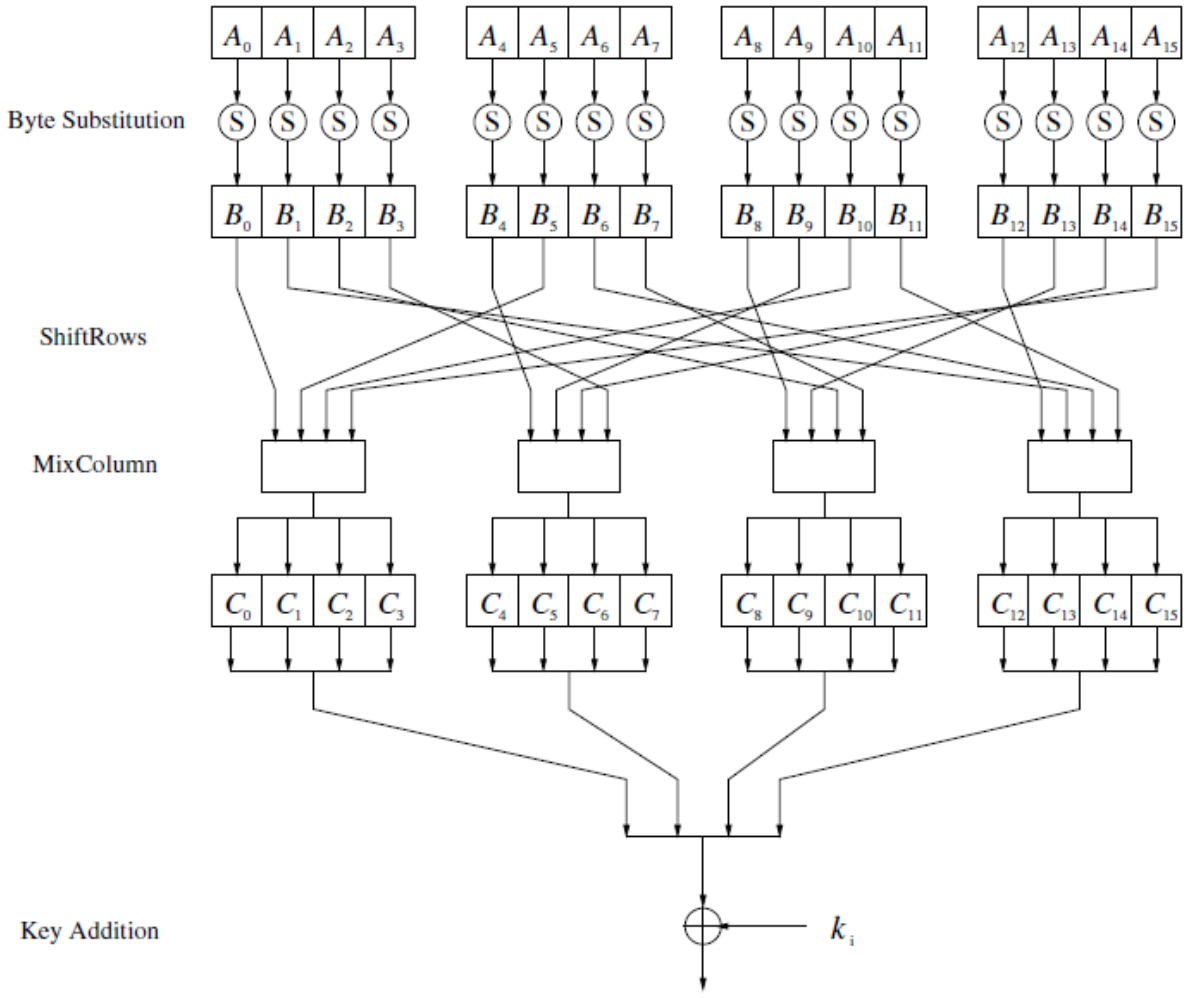


Figure. 6.3 AES round function for rounds 1,2, . . . ,$n_r - 1$

This is in contrast to DES, which makes heavy use of bit permutation and can thus be considered to have a bit-oriented structure.

In order to understand how the data moves through AES, we first imagine that the state A (i.e., the 128-bit data path) consisting of 16 bytes $A_0, A_1, \ldots, A_{15}$ is arranged in a four-by-four byte matrix:

A_0	A_4	A_8	A_{12}
A_1	A_5	A_9	A_{13}
A_2	A_6	A_{10}	A_{14}
A_3	A_7	A_{11}	A_{15}

As we will see in the following, AES operates on elements, columns or rows of the current state matrix. Similarly, the key bytes are arranged into a matrix with four rows and four (128-bit key), six (192-bit key) or eight (256-bit key) columns. Here is, as an example, the state matrix of a 192-bit key:

k_0	k_4	k_8	k_{12}	k_{16}	k_{20}
k_1	k_5	k_9	k_{13}	k_{17}	k_{21}
k_2	k_6	k_{10}	k_{14}	k_{18}	k_{22}
k_3	k_7	k_{11}	k_{15}	k_{19}	k_{23}

We discuss now what happens in each of the layers.

6.4.1 Byte Substitution Layer

As shown in Fig. 6.3, the first layer in each round is the Byte Substitution layer. The Byte Substitution layer can be viewed as a row of 16 parallel S-Boxes, each with 8 input and output bits. Note that all 16 S-Boxes are identical. In the layer, each state byte Ai is replaced, i.e., substituted, by another byte Bi: S(Ai) = Bi.

The S-Box is the only nonlinear element of AES, i.e., it holds that ByteSub(A) + ByteSub(B) 6= ByteSub(A+B) for two states A and B. The S-Box substitution is a bijective mapping, i.e., each of the 2^8 = 256 possible input elements is one-to-one mapped to one output element. This allows us to uniquely reverse the S-Box, which is needed for decryption. In software implementations the S-Box is usually realized as a 256-by-8 bit lookup table with fixed entries, as given in Table 6.2.

	y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
x 8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Table 6.2 AES S-Box: Substitution values in hexadecimal notation for input byte (xy)

6.4.1.1 Mathematical description of the S-Box

This description, however, is not necessary for a basic understanding of AES, and the remainder of this subsection can be skipped without problem. Unlike the DES SBoxes, which are essentially random tables that fulfill certain properties, the AES S-Boxes have a strong algebraic structure. An AES S-Box can be viewed as a two-step mathematical transformation (Fig. 6.4)

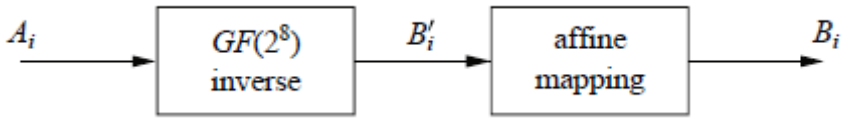


Fig. 6.4 The two operations within the AES S-Box which computes the function Bi = S(Ai)

The first part of the substitution is a Galois field inversion, the mathematics of which were introduced in Sect. 6.3.6. For each input element Ai, the inverse is computed: $B'_i = A_i{-}1$, where both A_i and B'_i are considered elements in the field $GF(2^8)$ with the fixed irreducible polynomial $P(x) = x^8 + x^4 + x^3 + x + 1$. A lookup table with all inverses is shown in Table 6.1. Note that the inverse of the zero element does not exist. However, for AES it is defined that the zero element Ai = 0 is mapped to itself.

In the second part of the substitution, each byte Bi is multiplied by a constant bit matrix followed by the addition of a constant 8-bit vector. The operation is described by:

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} \equiv \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} \bmod 2.$$

This second step is referred to as affine mapping. Let's look at an example of how the S-Box computations work.

We assume the S-Box input $A_i = (11000010)_2 = (C2)_{hex}$. From Table 6.1 we can see that the inverse is:

$$A_i^{-1} = B'_i = (2F)_{hex} = (00101111)_2.$$

We now apply the B'_i bit vector as input to the affine transformation. Note that the least significant bit (lsb) b'_0 of B'_i is at the rightmost position. $B_i = (00100101) = (25)_{hex}$

If one computes both steps for all 256 possible input elements of the S-Box and stores the results, one obtains Table 6.2.

The advantage of using inversion in GF (2^8) as the core function of the Byte Substitution layer is that it provides a high degree of nonlinearity, which in turn provides optimum protection against some of the strongest known analytical attacks. The affine step "destroys" the algebraic structure of the Galois field, which in turn is needed to prevent attacks that would exploit the finite field inversion.

6.4.2 Diffusion Layer

In AES, the Diffusion layer consists of two sub layers, the ShiftRows transformation and the MixColumn transformation. We recall that diffusion is the spreading of the influence of individual bits over the entire state. Unlike the nonlinear S-Box, the diffusion layer performs a linear operation on state matrices A,B, i.e., DIFF(A) + DIFF(B) = DIFF(A+B).

6.4.2.1 ShiftRows Sub layer

The ShiftRows transformation cyclically shifts the second row of the state matrix by three bytes to the right, the third row by two bytes to the right and the fourth row by one byte to the right. The first row is not changed by the ShiftRows transformation.

The purpose of the ShiftRows transformation is to increase the diffusion properties of AES. If the input of the ShiftRows sub layer is given as a state matrix B = (B_0, B1, . . . , B15):

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

the output is the new state:

B_0	B_4	B_8	B_{12}	no shift
B_5	B_9	B_{13}	B_1	⟵ one position left shift
B_{10}	B_{14}	B_2	B_6	⟵ two positions left shift
B_{15}	B_3	B_7	B_{11}	⟵ three positions left shift

6.4.2.2 MixColumn Sub layer

The MixColumn step is a linear transformation, which mixes each column of the state matrix. Since every input byte influences four output bytes, the MixColumn operation is the major diffusion element in AES. The combination of the ShiftRows and MixColumn layer makes it possible that after only three rounds every byte of the state matrix depends on all 16 plaintext bytes.

In the following, we denote the 16-byte input state by B and the 16-byte output state by C:

MixColumn(B) =C,

where B is the state after the ShiftRows operation. Now, each 4-byte column is considered as a vector and multiplied by a fixed 4×4 matrix. The matrix contains constant entries. Multiplication and addition of the coefficients is done in GF (2^8). As an example, we show how the first four output bytes are computed:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}$$

The second column of output bytes (C_4, C5, C6, C7) is computed by multiplying the four input bytes (B_4,B_9,B_{14},B_3) by the same constant matrix, and so on. Figure 6.3 shows, which input bytes, are used in each of the four MixColumn operations. We discuss now the details of the vector–matrix multiplication, which forms the MixColum operations. We recall that each state byte C_i and B_i is an 8-bit value representing an element from GF (2^8). All arithmetic involving the coefficients is done in this Galois field. For the constants in the matrix a hexadecimal notation is used: “01” refers to the GF(2^8) polynomial with the coefficients (00000001), i.e., it is the element 1 of the Galois field; “02” refers to the polynomial with the bit vector (00000010), i.e., to the polynomial x; and “03” refers to the polynomial with the bit vector (00000011), i.e., the Galois field element x+1.

The additions in the vector–matrix multiplication are GF (2^8) additions, that is simple bitwise XORs of the respective bytes. For the multiplication of the constants, we have to realize multiplications with the constants 01, 02 and 03. These are quite efficient. Multiplication by 01 is multiplication by the identity and does not involve any explicit operation. Multiplication by 02 and 03 can be done through table look-up in two 256-by-8 tables. As an alternative, multiplication by 02 can also be implemented as a multiplication by x, which is a left shift by one bit, and a modular reduction with $P(x) = x^8 + x^4 + x^3 + x + 1$. Similarly, multiplication by 03, which represents the polynomial (x+1), can be implemented by a left shift by one bit and addition of the original value followed by a modular reduction with P(x).

For example we assume that the input state to the MixColumn layer is B = (25, 25, . . . , 25).

In this special case, only two multiplications in GF (2^8) have to be done. These are 02 · 25 and 03 · 25, which can be computed in polynomial notation:

$$\begin{aligned} 02 \cdot 25 &= x \cdot (x^5 + x^2 + 1) \\ &= x^6 + x^3 + x, \\ 03 \cdot 25 &= (x+1) \cdot (x^5 + x^2 + 1) \\ &= (x^6 + x^3 + x) + (x^5 + x^2 + 1) \\ &= x^6 + x^5 + x^3 + x^2 + x + 1. \end{aligned}$$

Since both intermediate values have a degree smaller than 8, no modular reduction with P(x) is necessary. The output bytes of C result from the following addition in GF (2^8):

$$\begin{array}{rcccccccc} 01 \cdot 25 = & & & x^5 + & & x^2 + & & 1 \\ 01 \cdot 25 = & & & x^5 + & & x^2 + & & 1 \\ 02 \cdot 25 = & x^6 + & & x^3 + & & x & & \\ 03 \cdot 25 = & x^6 + & x^5 + & x^3 + & x^2 + & x + & 1 \\ \hline C_i = & & & x^5 + & & x^2 + & & 1, \end{array}$$

6.4.3 Key Addition Layer

The two inputs to the Key Addition layer are the current 16-byte state matrix and a sub key which also consists of 16 bytes (128 bits). The two inputs are combined through a bitwise XOR operation. Note that the XOR operation is equal to addition in the Galois field GF (2). The sub keys are derived in the key schedule that is described below in the next Sect.

6.4.4 Key Schedule

The key schedule takes the original input key (of length 128, 192 or 256 bit) and derives the sub keys used in AES. Note that an XOR addition of a sub key is used both at the input and output of AES. This process is sometimes referred to as key whitening. The number of sub keys is equal to the number of rounds plus one, due to the key needed for key whitening in the first key addition layer, cf. Fig. 6.2.

Thus, for the key length of 128 bits, the number of rounds is nr = 10, and there are 11 sub keys, each of 128 bits. The AES with a 192-bit key requires 13 sub keys of length 128 bits, and AES with a 256-bit key

has 15 sub keys. The AES sub keys are computed recursively, i.e., in order to derive sub key k_i, sub key k_i-1 must be known, etc.

The AES key schedule is word-oriented, where 1 word = 32 bits. Sub keys are stored in a key expansion array W that consists of words. There are different key schedules for the three different AES key sizes of 128, 192 and 256 bit, which are all fairly similar. We introduce only the key schedules for the AES-128.

6.4.4.1 Key Schedule for 128-Bit Key AES

The 11 sub keys are stored in a key expansion array with the elements W[0], . . . ,W[43]. The sub keys are computed as depicted in Fig. 6.4

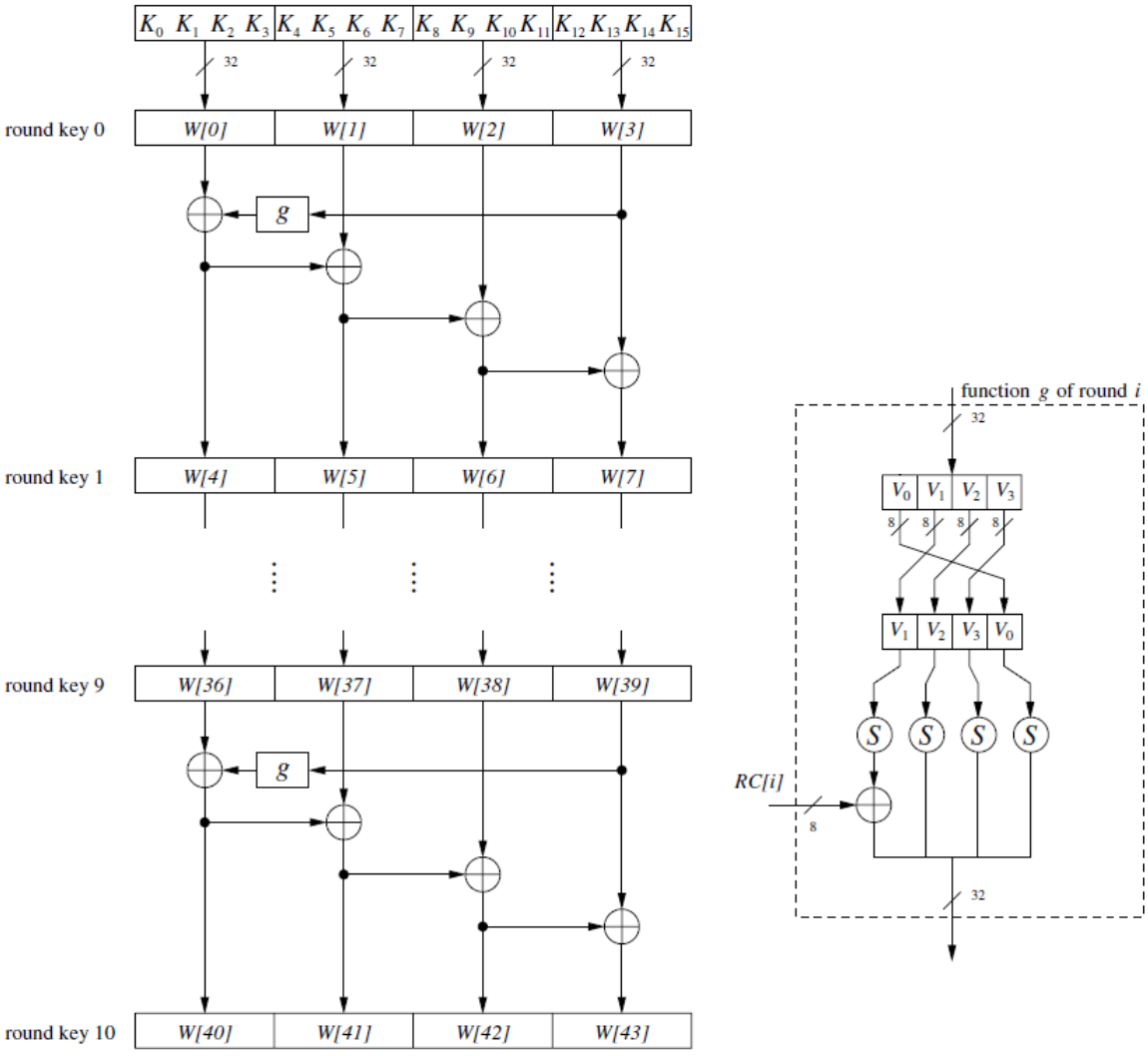


Figure 6.4 AES key schedule for 128-bit key size

The elements $K_0, \ldots, K15$ denote the bytes of the original AES key. First, we note that the first sub key k_0 is the original AES key, i.e., the key is copied into the first four elements of the key array W. The other array elements are computed as follows. As can be seen in the figure, the leftmost word of a sub key

W[4i], where i = 1, . . . , 10, is computed as: $W[4i] = W[4(i-1)] + g(W[4i-1])$.

Here g() is a nonlinear function with a four-byte input and output. The remaining three words of a sub key are computed recursively as:

$W[4i+j] = W[4i+j-1] + W[4(i-1)+j]$,

where i = 1, . . . ,10 and j = 1,2, 3. The function g() rotates its four input bytes, performs a byte-wise S-Box substitution, and adds a round coefficient RC to it. The round coefficient is an element of the Galois field GF (2^8), i.e, an 8-bit value. It is only added to the leftmost byte in the function g(). The round coefficients vary from round to round according to the following rule:

$$\begin{aligned} RC[1] &= x^0 = (00000001)_2, \\ RC[2] &= x^1 = (00000010)_2, \\ RC[3] &= x^2 = (00000100)_2, \\ &\vdots \\ RC[10] &= x^9 = (00110110)_2. \end{aligned}$$

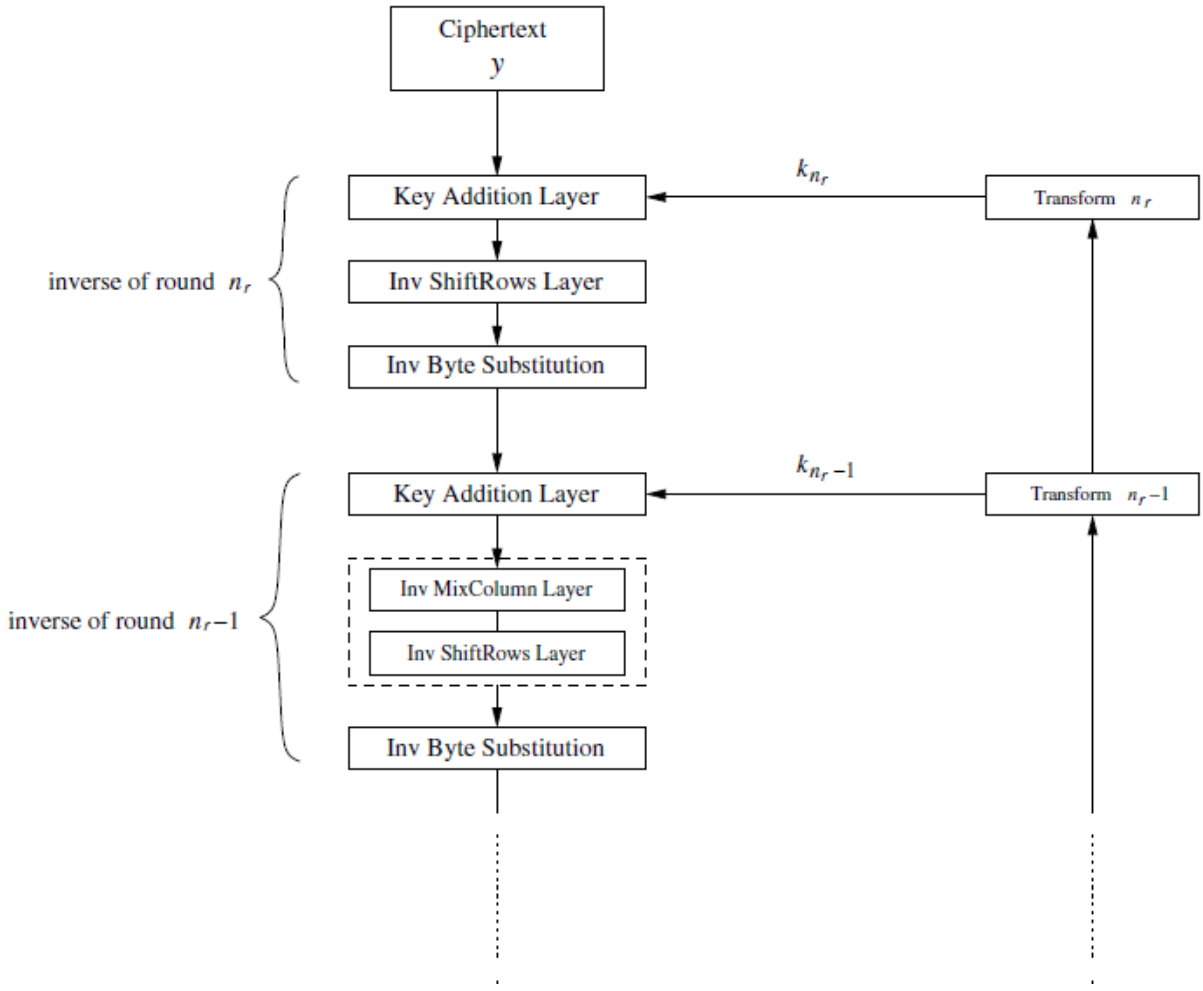
The function g() has two purposes. First, it adds nonlinearity to the key schedule. Second, it removes symmetry in AES. Both properties are necessary to thwart certain block cipher attacks.

In general, when implementing any of the key schedules, two different approaches exist:

1. Pre computation, all sub keys are expanded first into the array W. The encryption (decryption) of a plaintext (ciphertext) is executed afterwards. This approach is often taken in PC and server implementations of AES, where large pieces of data are encrypted under one key. Please note that this approach requires (nr +1) · 16 bytes of memory, e.g., 11 · 16 = 176 bytes if the key size is 128 bits. This is the reason why such an implementation on a device with limited memory resources, such as a smart card, is sometimes not desirable.
2. 2. On-the-fly, a new sub key is derived for every new round during the encryption (decryption) of a plaintext (ciphertext). Please note that when decrypting cipher texts, the last sub key is XORed first with the cipher text. Therefore, it is required to recursively derive all sub keys first and then start with the decryption of a cipher text and the on-the-fly generation of sub keys. As a result of this overhead, the decryption of a cipher text is always slightly slower than the encryption of a plaintext when the on-the-fly generation of sub keys is used.

6.4.5 Decryption

Because AES is not based on a Feistel network, all layers must actually be inverted, i.e., the Byte Substitution layer becomes the Inv Byte Substitution layer, the ShiftRows layer becomes the Inv ShiftRows layer, and the MixColumn layer becomes Inv MixColumn layer. However, it turns out that the inverse layer operations are fairly similar to the layer operations used for encryption. In addition, the order of the sub keys is reversed, i.e., we need a reversed key schedule. A block diagram of the decryption function is shown in Fig. 6.5.



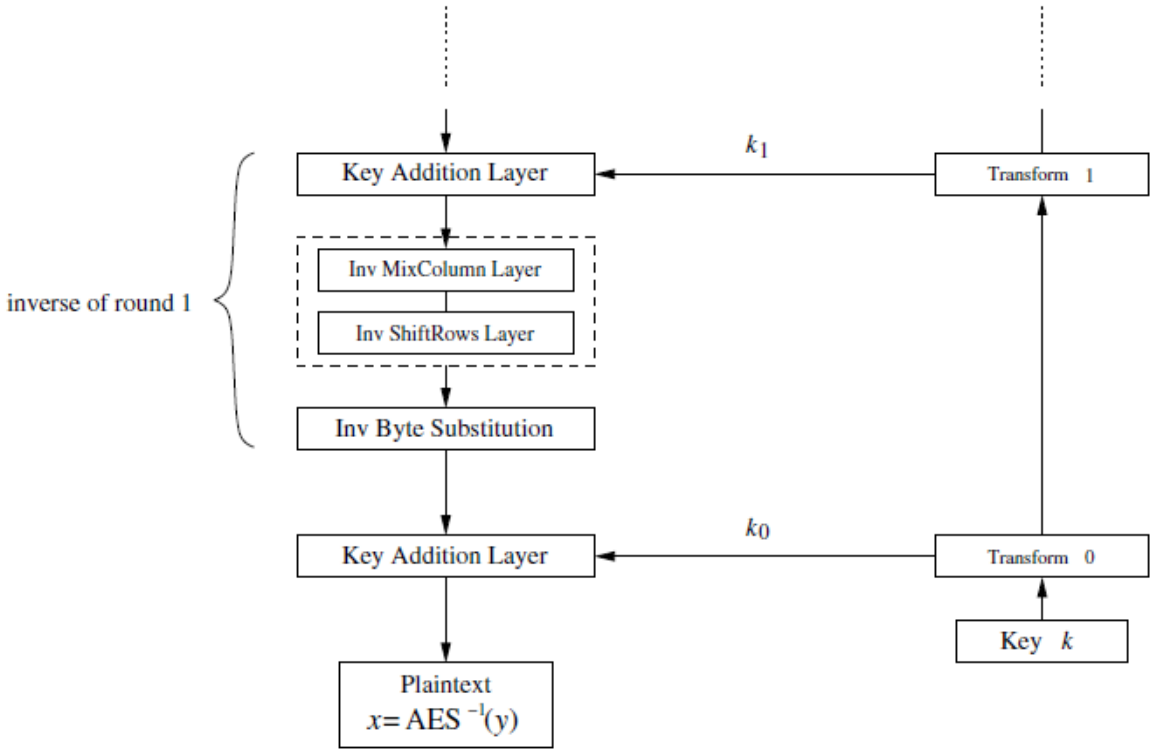


Figure 6.5 AES decryption block diagram

Since the last encryption round does not perform the MixColum operation, the first decryption round also does not contain the corresponding inverse layer. All other decryption rounds, however, contain all AES layers.

6.4.5.1 Inverse MixColumn Sublayer

After the addition of the sub key, the inverse MixColumn step is applied to the state (again, the exception is the first decryption round). In order to reverse the MixColumn operation, the inverse of its matrix must be used. The input is a 4-byte column of the State C, which is multiplied by the inverse 4×4 matrix. The matrix contains constant entries. Multiplication and addition of the coefficients is done in GF(2^8).

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

The second column of output bytes (B_4, B5, B6, B7) is computed by multiplying the four input bytes (C_4, C5, C6, C7) by the same constant matrix, and so on. Each value B_i and C_i is an element from GF (2^8). In addition, the constants are elements from GF (2^8). The notation for the constants is hexadecimal and is the same as was used for the MixColumn layer, for example: $0B = (0B)_{hex} = (0000\,1011)_2 = x^3 + x + 1.$

6.4.5.2 Inverse ShiftRows Sublayer

In order to reverse the ShiftRows operation of the encryption algorithm, we must shift the rows of the state matrix in the opposite direction. The first row is not changed by the inverse ShiftRows transformation. If the input of the ShiftRows sub layer is given as a state matrix B = (B_0, B1, . . . , B15):

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

the inverse ShiftRows sub layer yields the output:

B_0	B_4	B_8	B_{12}	no shift
B_{13}	B_1	B_5	B_9	$\longrightarrow$ one position right shift
B_{10}	B_{14}	B_2	B_6	$\longrightarrow$ two positions right shift
B_7	B_{11}	B_{15}	B_3	$\longrightarrow$ three positions right shift

6.4.5.3 Inverse Byte Substitution Layer

The inverse S-Box is used when decrypting a cipher text. Since the AES S-Box is a bijective, i.e., a one-to-one mapping, it is possible to construct an inverse S-Box such that: $A_i = S^{-1}(B_i) = S^{-1}(S(A_i))$

where A_i and B_i are elements of the state matrix. The entries of the inverse S-Box are given in the Table in the next page. In order to reverse the SBox substitution, we first have to compute the inverse of the affine transformation. For this, each input byte Bi is considered an element of GF (2^8). The inverse affine transformation on each byte Bi is defined by:

	y															
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
x 8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

$$\begin{pmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{pmatrix} \equiv \begin{pmatrix} 0&1&0&1&0&0&1&0 \\ 0&0&1&0&1&0&0&1 \\ 1&0&0&1&0&1&0&0 \\ 0&1&0&0&1&0&1&0 \\ 0&0&1&0&0&1&0&1 \\ 1&0&0&1&0&0&1&0 \\ 0&1&0&0&1&0&0&1 \\ 1&0&1&0&0&1&0&0 \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \mod 2,$$

where ($b_7, \ldots, b_0$) is the bitwise vector representation of Bi(x), and ($b'_7, \ldots, b'_0$) the result after the inverse affine transformation. In the second step of the inverse S-Box operation, the Galois field inverse has to be reversed. For this, note that $A_i = (A_i^{-1})^{-1}$. This means that the inverse operation is reversed by computing the inverse again. In our notation, we thus have to compute $A_i = (B'_i)^{-1} \in GF(2^8)$ with the fixed reduction polynomial $P(x) = x^8+x^4+x^3+x+1$. Again, the zero element is mapped to itself. The vector $A_i = (a_7, \ldots, a0)$ (representing the field element $a_7x^7+\cdots+a_1x+a_0$) is the result of the substitution: $A_i = S^{-1}(B_i)$

6.4.5.4 Decryption Key Schedule

Since decryption round one needs the last sub key, the second decryption round needs the second-to-last sub key and so on, we need the sub key in reversed order as shown in Fig. 6.5. In practice, this is mainly achieved by computing the entire key schedule first and storing all 11, 13 or 15 sub keys, depending on the number or rounds AES is using (which in turn depends on the three key lengths supported by AES). This pre computation adds usually a small latency to the decryption operation relative to encryption.

In the Fig 6.6, there is a general AES decryption round.

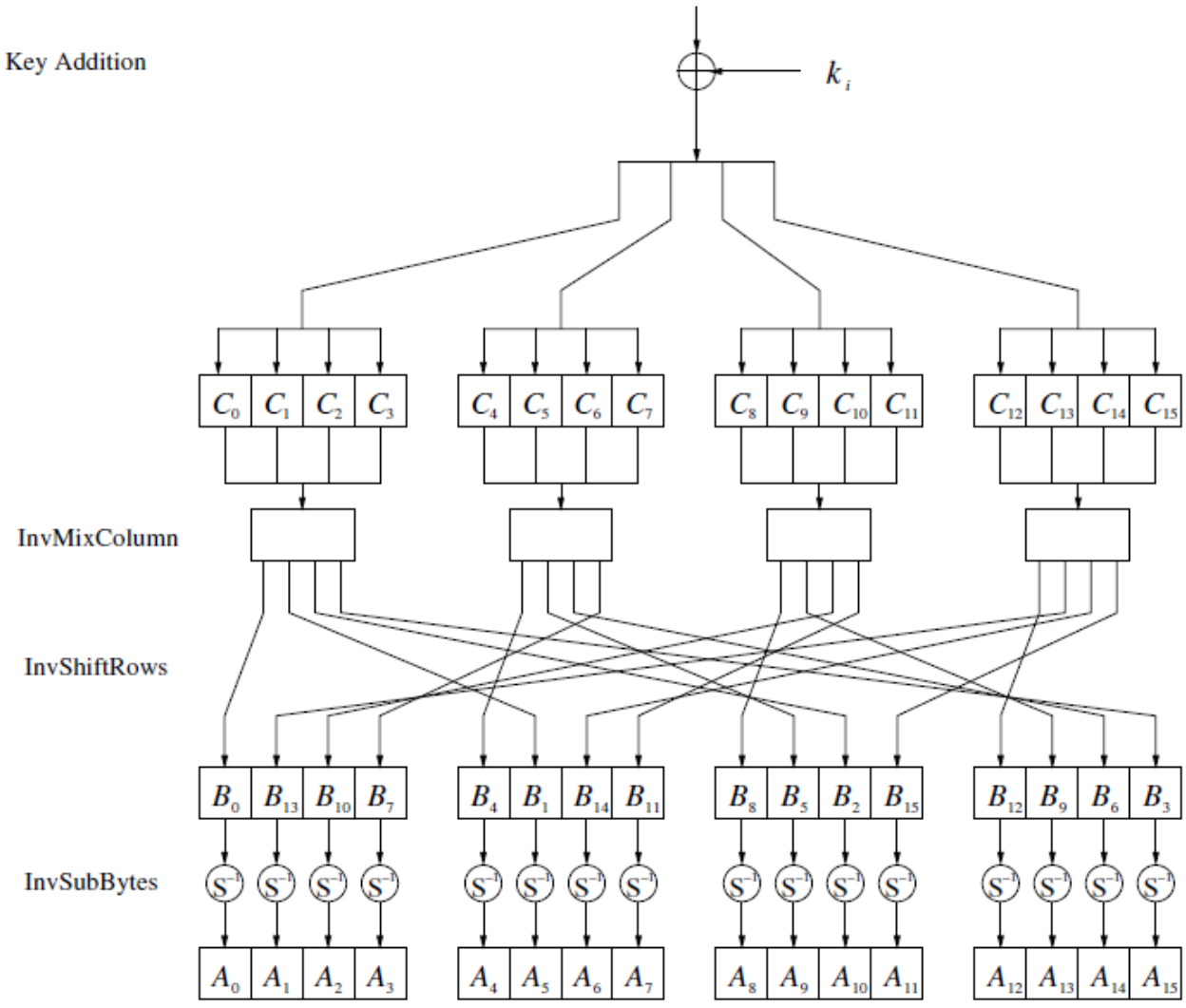


Figure 6.6 AES decryption round function 1, 2. . . nr −1

Chapter 7
AES-128 HW Implementation

At this point, we have all the tools and knowledge to implement a hardware module that guarantees the privacy aspect of security.

The implementation of the module is not driven by particular constraint, for this reason, the module will be developed following an approach of type "best effort". This type of analysis can be a starting point for a feasibility study, to see if it is possible insert a certain level of security without violating the constraints imposed by the automotive environment.

7.1 System Verilog model

With a look at the subsequent test phases that follow the development of the hardware module, has been realized a model in System Verilog.

The model will initially serve as a basis for the realization of the hardware module. Subsequently it will be used as golden unit or golden device for the testing phase.

Some important aspects of the model will be taken up later after we will discuss the hardware module. For more information and details on this model, refer to the complete code in Appendix

Naturally, the test environment will be developed in System Verilog. One of the many benefits that we can have, writing in System Verilog, are the randomization functions, that allow to create automated tests.

For fast functional verification, a simple direct test in Verilog has been implemented.

The automated tests introduce an initial overhead of programming but bring great advantages for debugging. on the contrary, the direct tests are fast to implement, but are used only when there are only a few case studies to explore, otherwise the phase of testing and debugging can become endless.

7.2 Hardware module

This section is intended to show the hardware module implemented, starting with the code Verilog written, and coming to the synthesis of the module.

As a platforms for the synthesis, was chosen the Virtex-6, a Virtex-5 and a Spartan-6, all XILINX's FPGA. The choice fell on those platforms because there are other studies and articles on this topic (AES-128 HW implementation) that use the same platforms (comparative purposes).

7.2.1 Overview

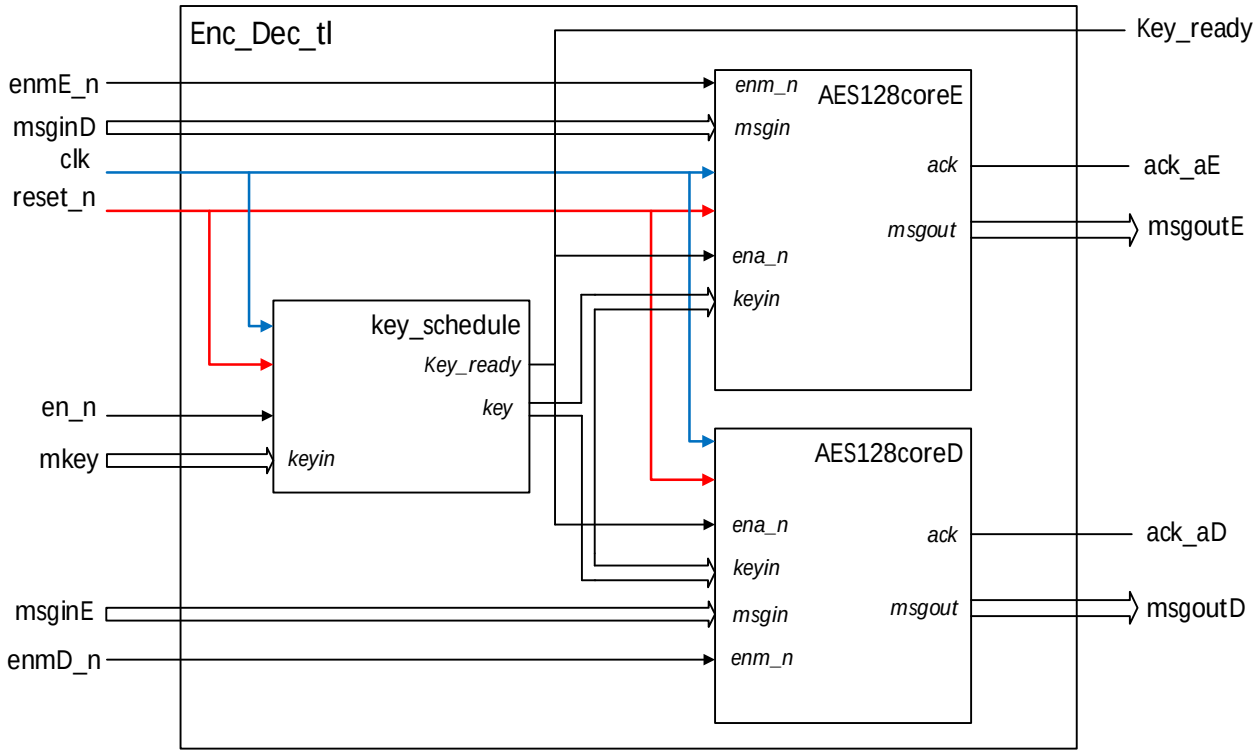


Figure 7.1scheme of the top level

As we can see in the previous figure, there are three distinct modules:

1. *Key_schedule*: performs the key generation algorithm

2. *AES128coreE*: it takes a plaintext in input and performs the encryption
3. *AES128coreD*: it takes an encrypted text in input and performs the decryption.

Encryption and decryption work both in parallel, in this way is possible perform two different operations at the same time. For this reason, there are two different input for the messages and in the same way, there are two different output.

We have opted for a pre computation phase where, starting from a master key, the other keys are computed. In this way, how we have just discuss in the AES chapter, we have an initial latency in encryption and decryption but only for the first transaction. An “on the fly” solution decrease the memory use but increase the computation time, and it could be not acceptable in an environment with many hard real time applications.

How we will see, there are two version of the same module, both solution have the same goal: perform the AES 128 algorithm as soon as possible,

The first solution has better performance in terms of clock cycles, while the second has better performance in terms of max clock frequency achievable.

However there are very similar and for this reason we will see deeply only one of the two modules (the first).

7.2.2 key_schedule module

This module as its name suggests, performs the key schedule algorithm, starting from a master key that is given from another entity (for example a secure memory) . When it finishes its computation, it enables the others two module to start the key transfer protocol. After these operations, the module can start with the encryption/decryption operations.

The signal *key_ready*, as well as activate the other modules, is sent in output to advise the external world that the module is ready to start the communications.

The change of the *master key* is possible only after a reset of the module.

This module is realized by a FSM that performs the operations descripted in the standard []. In the next figure, there is a state diagram of the FSM mentioned.

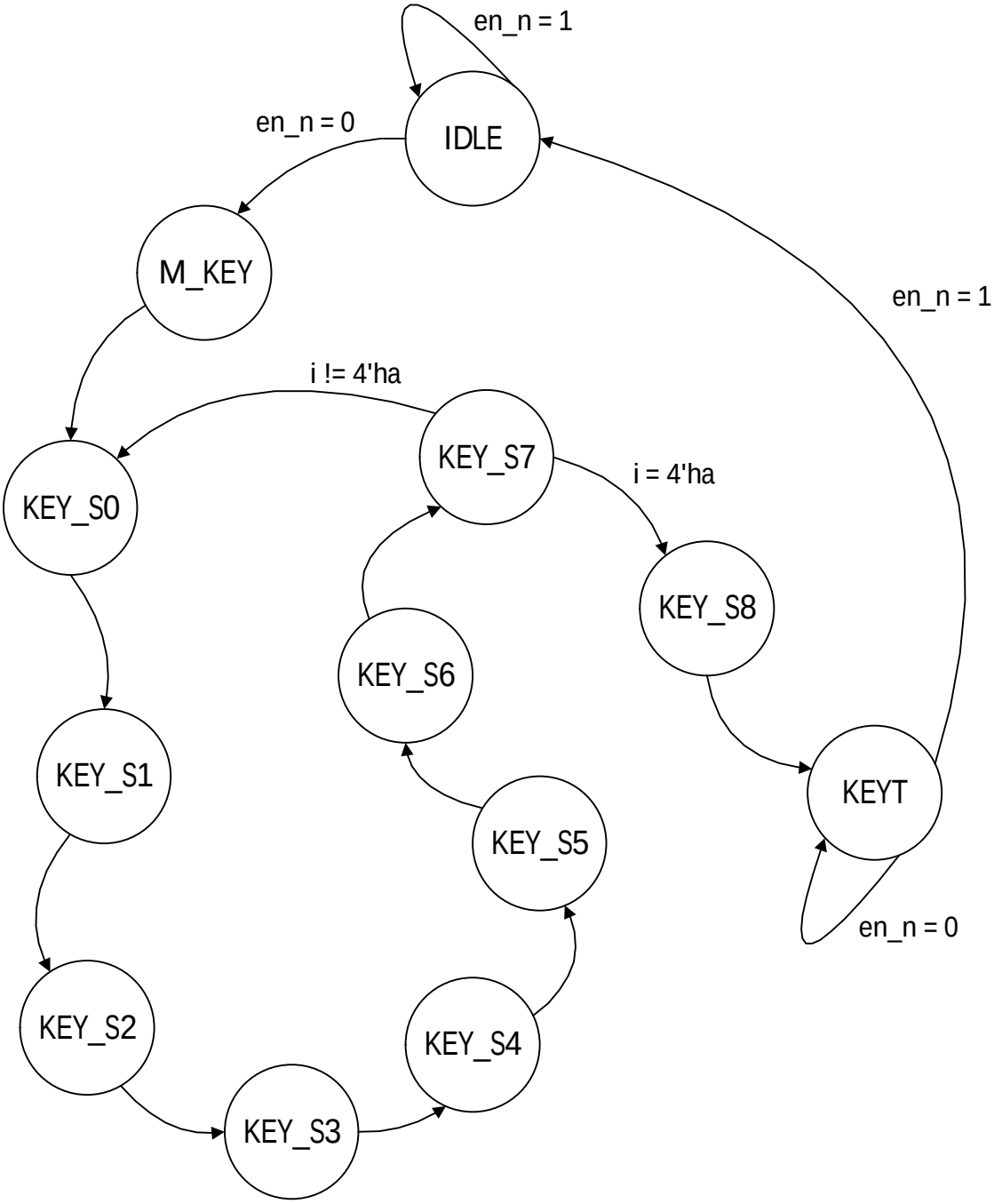


Figure 7.2 key_schedule FSM

For further information about the Verilog code, see the Appendix.

7.2.3 Encryption/Decryption module

As we have just seen these modules are activated the first time by the key_sceduler, which passes the computed keys at these modules. Finished this procedure the entire module is ready and these modules wait an incoming message.

There is a singular section for both modules because they are very similar. As we have already see , the AES algorithm is not a Feistel network, however the two modules can't be the same but, due to an easy hardware implementation requirements of the AES challenge, the two modules have a few differences.

The most significant difference is an implementation of an important transformation in the AES algorithm. In the Encryption case the implementation of this transformation (*mix column layer*) pass through the implementation of the "*xtime"* function, that it is explained in the standard.

The same function is not efficient for the decryption algorithm. In this case for the implementation of the same transformation is used a "LUT approach".

This implementation is the fastest achievable in term of clock ticks needed to perform the algorithm because each transformation is performed in one clock cycle.

To overcome this result is possible use some mathematical techniques that allow to interlacing some operations. However it is complicated and do not bring big advantages in terms of performance achievable.

For the rest, there are no other differences to emphasize and also in this case, for further information about the Verilog code, consult the Appendix.

In the following pages there the state diagram of the two FSM of the two modules.

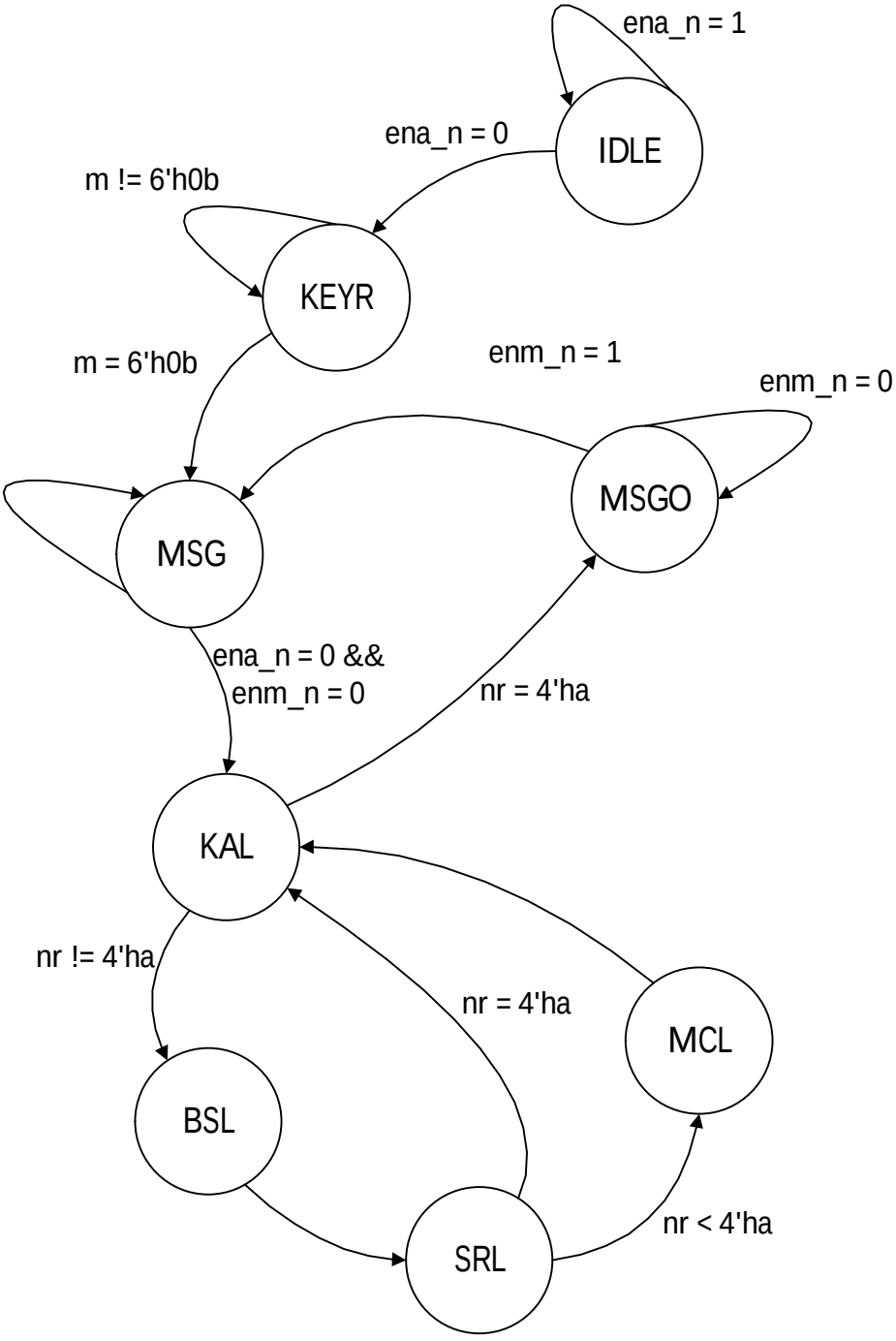


Figure 7.3 encryption FSM

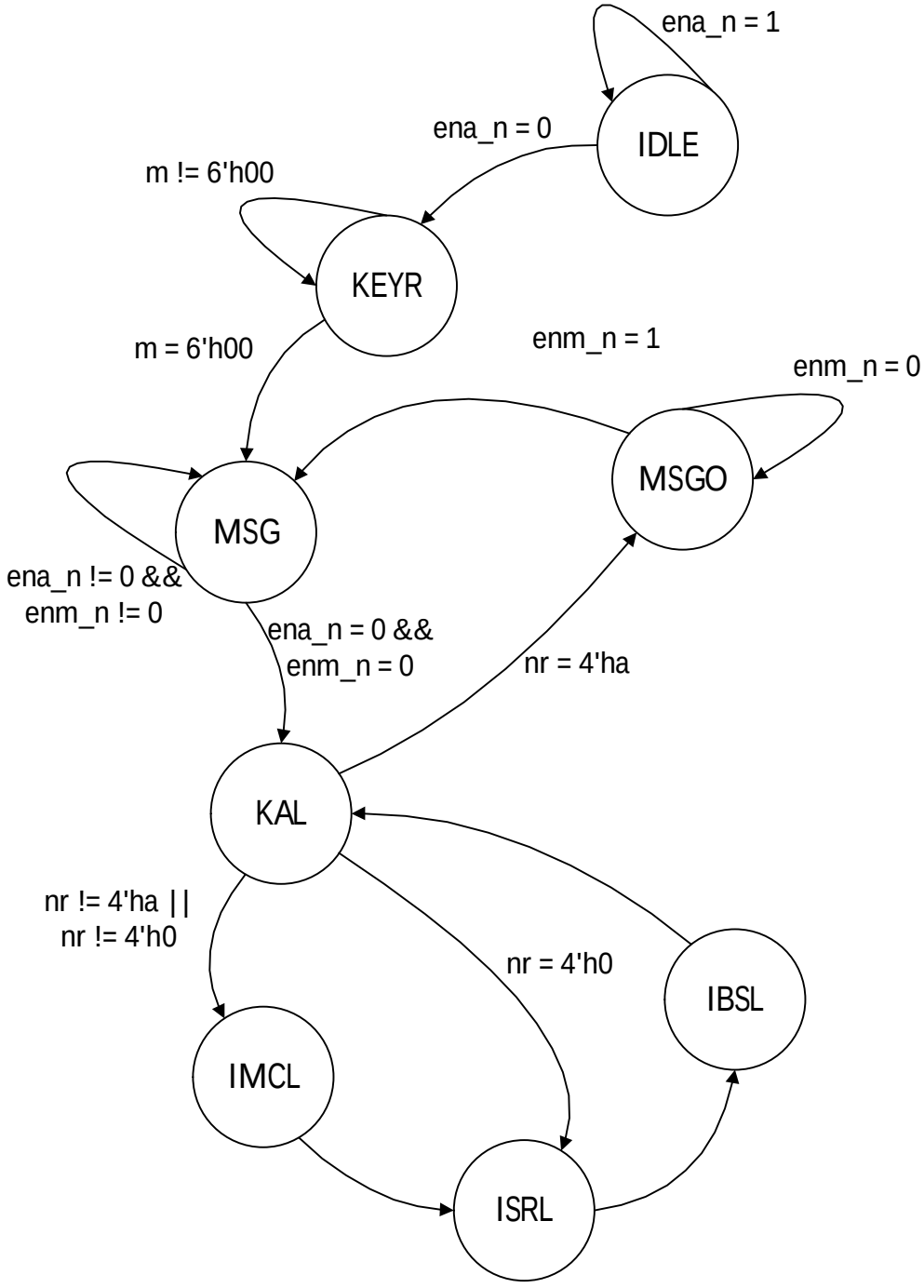


Figure 7.4 decryption FSM

7.3 Functional simulations

In this section are presented and discussed the functional simulations (performed with ModelSim) of the two versions of the module.

In both the version, we execute two simple direct test in succession. In the first, we insert a plaintext "*primo test*" in input, we wait the computation of the encrypted text and then we reuse this encrypted text like input for the decryption path. If the module work correctly, the output at the end of the procedure must be the same of the first plaintext input (*primo test*).

In the second test, we need to verify the parallel behavior of the model and thus we load at the same time the two input for the messages. In the encryption path, the input will be "*secondo test*" in the decryption path will be the AES-128 encrypted output for "*secondo test*". In this way in output we will wait the two input inverted, or rather like output of the encryption path there will be the input of the decryption path and like output of the decryption path there will be the input of the encryption path.

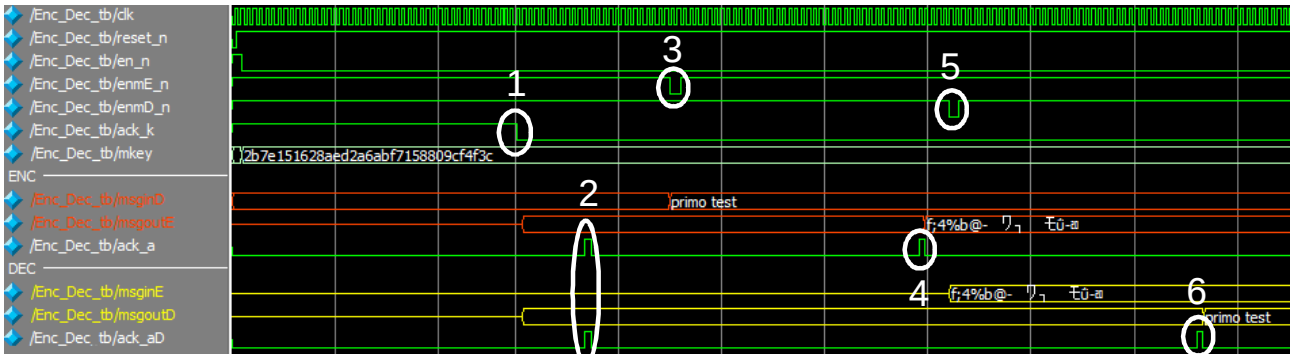


Figure 7.5 Modelsim waveform of the simulated module

(The text is displayed in ASCII)

The previous figure represent the top-level signals of the version 1 of the module for the first direct test, in the following there is a brief description of the main events (transactions) of the signals.

1. ack_k (1 -> 0) this transaction, indicates that the key schedule module has just terminated its execution and the others two modules can be wake up for the transfer keys protocol.
2. ack_a/ ack_D (1 for one clock tick) indicates that the transfer keys protocol procedure has just ended for both modules (encryption/decryption path).
3. enmE_n (1 -> 0) indicates that there is a message incoming in the encryption path and the corresponding module performs the message acquisition.

4. ack_a (1 for one clock tick) indicates that the encryption path has just ended the encryption algorithm.
5. enmD_n (1 -> 0) indicates that there is a message incoming in the decryption path and the corresponding module performs the message acquisition.
6. ack_aD (1 for one clock tick) indicates that the decryption path has just ended the decryption algorithm.

The final output "*msgoutD*" has the value expected

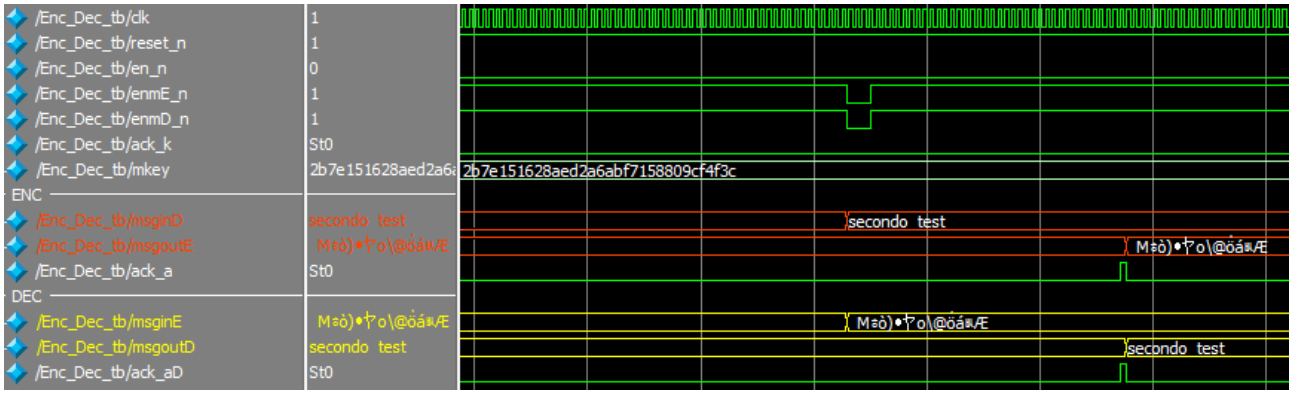


Figure 7.6 Modelsim waveform of the simulated module

The previous figure represent the top-level signals of the version 1 of the module for the second direct test. The behaviors of the signals, is the same of the previous case the only difference, is that the execution of both path is done in parallel. From this picture, we can note that the ack of both internal module are synchronized, this means that the execution of both modules require the same time in terms of clock cycle. This is not a surprise, because the algorithms are similar and they have the same number of operation to perform.

After these simple direct tests, we can say that the functionality of the module is proven. To explore all possibly case of utilization of the module we must go towards the automated tests.

7.4 Synthesis

In this section, we will see the performance of our module. For this purpose we used the tool **Precision** (Mentor Graphics) for synthesis processes and like platform, the **Virtex 6** already mentioned previously (the others two platforms are not displayed).

In the following, we will see the result of the process of synthesis in term of max clock frequency achievable

7.4.1 Version 1

7.4.1.1 Timing Report

```
-- Device: Xilinx - VIRTEX-6 : 6VLX75TFF484 : 3
-- CTE report summary...
-- POST-SYNTHESIS TIMING REPORTS ARE ESTIMATES AND SHOULD NOT BE RELIED ON TO MAKE QoR DECISIONS. For accurate timing information,

================================================================================================
                  Clock Frequency Report

        Domain                  Clock Name                      Min Period (Freq)           Required Period (Freq)
        ------                  ----------                      -----------------           ----------------------
        ClockDomain0            clk                             3.752 (266.525 MHz)         2.000 (500.000 MHz)


================================================================================================
Setup Timing Analysis of clk

Setup Slack Path Summary

                Data                                                                                                    Data
       Setup    Path    Source  Dest.                                                                                  End
Index  Slack    Delay   Clock   Clock           Data Start Pin                          Data End Pin                    Edge
-----  ------   -----   ------  -----   ---------------------------------------  ---------------------------------------  ----
  1    -1.752   3.391   clk     clk     AD/reg_state(7)/C                        AD/Statematrik/ix56310z19870/ADDRBWRADDR(5)  Rise
  2    -1.716   3.355   clk     clk     K0/reg_state(11)/C                       AD/Statematrik/ix56310z19870/ADDRBWRADDR(5)  Rise
  3    -1.699   3.338   clk     clk     K0/reg_state(10)/C                       AD/Statematrik/ix56310z19870/ADDRBWRADDR(5)  Rise
  4    -1.477   3.116   clk     clk     AD/reg_state(0)/C                        AD/Statematrik/ix56310z19870/ADDRBWRADDR(5)  Rise
  5    -1.231   3.243   clk     clk     K0/modgen_counter_rtlc_wadd_n62_reg_q(2)/C  K0/reg_W(1)(24)/D                      Rise
  6    -1.231   3.243   clk     clk     K0/modgen_counter_rtlc_wadd_n62_reg_q(5)/C  K0/reg_W(1)(24)/D                      Rise
  7    -1.230   3.242   clk     clk     K0/modgen_counter_rtlc_wadd_n62_reg_q(3)/C  K0/reg_W(1)(0)/D                       Rise
  8    -1.230   3.242   clk     clk     K0/modgen_counter_rtlc_wadd_n62_reg_q(4)/C  K0/reg_W(1)(0)/D                       Rise
  9    -1.198   3.210   clk     clk     AE/Statematrik/ix25675z19870/CLKBWRCLK   AE/reg_SM(1)(3)(0)/D                   Rise
 10    -1.198   3.210   clk     clk     AD/Statematrik/ix25675z19870/CLKBWRCLK   AD/reg_SM(1)(0)(2)/D                   Rise

                CTE Path Report

Critical path #1, (path slack = -1.752):

SOURCE CLOCK: name: clk period: 2.000000
     Times are relative to the 1st rising edge
  DEST CLOCK: name: clk period: 2.000000
     Times are relative to the 2nd rising edge
```

.....

```
AD/state(7)                                    (net)          0.775                 137
AD/ix11118z63781/I3                            LUT5                   1.055   up
AD/ix11118z63781/O                             LUT5           0.053   1.108   up
AD/nx11118z5                                   (net)          0.775                 133
AD/ix11118z63779/I5                            LUT6                   1.883   up
AD/ix11118z63779/O                             LUT6           0.053   1.936   up
AD/nx11118z2                                   (net)          0.246                   4
AD/ix34479z23076/I0                            LUT4                   2.182   up
AD/ix34479z23076/O                             LUT4           0.053   2.235   up
AD/nx34479z1                                   (net)          1.156                   5
AD/Statematrik/ix56310z19870/ADDRBWRADDR(5)    RAMB36E1               3.391   up

                Initial edge separation:       2.000
                Source clock delay:      -     0.000
                Dest clock delay:        +     0.000
                                         ----------
                Edge separation:               2.000
                Setup constraint:        -     0.361
                                         ----------
                Data required time:            1.639
                Data arrival time:       -     3.391   ( 12.95% cell delay, 87.05% net delay )
                                         ----------
                Slack (VIOLATED):             -1.752
```

Figure 7.6 Precision Timing report

Naturally, this simulation produces incorrect data regards the Slack because to find the max clock frequency achievable we have forced the required period to 2 ns (500 MHz) and this constraint for our module is unreachable.

7.4.1.2 Area Report

```
*******************************************************************
Device Utilization for 6VLX75TFF484
*******************************************************************
Resource                          Used    Avail   Utilization
-------------------------------------------------------------------
IOS                               648     -           -
Global Buffers                    0       32        0.00%
LUTs                              4826    46560    10.37%
CLB Slices                        1207    11640    10.37%
Dffs or Latches                   2619    93120     2.81%
Block RAMs                        8       156       5.13%
   RAMB18E1                       0
   RAMB36E1                       8
DSP48E1s                          0       288       0.00%
-------------------------------------------------------------------

*************************************************

Library: work    Cell: Enc_Dec_tl    View: INTERFACE

*************************************************

  Cell             Library  References     Total Area

 AES128coreD_0     work     1 x    661    661 Dffs or Latches
                                   128    128 MUXF8
                                   352    352 MUXF7
                                  1561   1561 LUTs
                                     8      4 Block RAMs
 AES128coreE_0     work     1 x    128    128 MUXF8
                                   352    352 MUXF7
                                     8      4 Block RAMs
                                   420    420 Dffs or Latches
                                  1174   1174 LUTs
 GND               xcv6     1 x

 Key_schedule_0    work     1 x    352    352 MUXF8
                                   704    704 MUXF7
                                  1538   1538 Dffs or Latches
                                  2091   2091 LUTs

 Number of ports :                        648
 Number of nets :                        1110
 Number of instances :                      4
 Number of references to this view :        0

Total accumulated area :
 Number of Block RAMs :                     8
 Number of Dffs or Latches :             2619
 Number of LUTs :                        4826
 Number of MUXF7 :                       1408
 Number of MUXF8 :                        608
 Number of gates :                       4839
 Number of accumulated instances :       9474
```

7.4.2 Version 2

7.4.2.1 Timing Report

```
-- Device: Xilinx - VIRTEX-6 : 6VLX75TFF484 : 3
-- CTE report summary..
-- POST-SYNTHESIS TIMING REPORTS ARE ESTIMATES AND SHOULD NOT BE RELIED ON TO MAKE QoR DECISIONS. For accurate timing information,

================================================================================================
                  Clock Frequency Report

  Domain                Clock Name                          Min Period (Freq)          Required Period (Freq)
  ------                ----------                          -----------------          ----------------------
  ClockDomain0          clk                                 3.231 (309.502 MHz)        2.000 (500.000 MHz)


================================================================================================
Setup Timing Analysis of clk

Setup Slack Path Summary

       Setup   Data Path   Source  Dest.                                                                    Data End
Index  Slack   Delay       Clock   Clock  Data Start Pin                          Data End Pin              Edge
-----  ------  -----  ------  -----  ---------------------------------------  ---------------------------------------  ----
  1    -1.231  3.243  clk     clk    modgen_counter_rtlc_wadd_n54_reg_q(2)/C  reg_W(1)(24)/D                           Rise
  2    -1.231  3.243  clk     clk    modgen_counter_rtlc_wadd_n54_reg_q(5)/C  reg_W(1)(24)/D                           Rise
  3    -1.230  3.242  clk     clk    modgen_counter_rtlc_wadd_n54_reg_q(4)/C  reg_W(1)(7)/D                            Rise
  4    -1.230  3.242  clk     clk    modgen_counter_rtlc_wadd_n54_reg_q(3)/C  reg_W(1)(7)/D                            Rise
  5    -1.198  3.210  clk     clk    StatematrikE/ix60576z19870/CLKBWRCLK     reg_SME(2)(0)(1)/D                      Rise
  6    -1.198  3.210  clk     clk    StatematrikE/ix47012z19870/CLKBWRCLK     reg_SME(3)(0)(3)/D                      Rise
  7    -1.198  3.210  clk     clk    StatematrikE/ix25675z19870/CLKBWRCLK     reg_SME(1)(0)(4)/D                      Rise
  8    -1.012  3.024  clk     clk    StatematrikE/ix56310z19870/CLKBWRCLK     reg_SME(0)(1)(1)/D                      Rise
  9    -0.957  2.596  clk     clk    req_state(14)/C                          StatematrikE/ix56310z19870/ADDRBWRADDR(8)  Rise
 10    -0.743  2.458  clk     clk    modgen_counter_rtlc_wadd_n60/reg_q(3)/C  StatematrikE/ix56310z19870/DIADI(0)      Rise

                  CTE Path Report

Critical path #1, (path slack = -1.231):

SOURCE CLOCK: name: clk period: 2.000000
     Times are relative to the 1st rising edge
  DEST CLOCK: name: clk period: 2.000000
     Times are relative to the 2nd rising edge


...
```

Figure 7.7 Precision Timing report

As expected this version of the same module is faster in terms of max clock frequency achievable, to realize a fair comparison between the two solutions we need to consider the throughput.

Version 1

-clock cycles to produce output: 41

-time to produce output: 41* (1/266,525 MHz) = 153 ns

Version 2

-clock cycles to produce output: 42

-time to produce output: 42* (1/309,502 MHz) = 135 ns

7.4.2.2 Area Report

With this configuration, we have better performance regard the max clock speed achievable and in terms of area, the result is similar. It seems that this second version is overall better than the previous but indeed; we have renounced to have the same key_schedule for both paths. (Thus, the area consumption is greater in the second version).

```
*************************************************************
Device Utilization for 6VLX75TFF484
*************************************************************
Resource                          Used    Avail   Utilization
-------------------------------------------------------------
IOS                               648     -           -
Global Buffers                    0       32        0.00%
LUTs                              3157    46560     6.78%
CLB Slices                        790     11640     6.79%
Dffs or Latches                   1943    93120     2.09%
Block RAMs                        4       156       2.56%
   RAMB18E1                       0
   RAMB36E1                       4
DSP48E1s                          0       288       0.00%
-------------------------------------------------------------

*********************************************

Library: work    Cell: BlackV    View: INTERFACE

*********************************************

  Cell                Library  References     Total Area

 FD                   xcv6        141 x      1    141 Dffs or Latches
 FDE                  xcv6       1760 x      1   1760 Dffs or Latches
 FDR                  xcv6         19 x      1     19 Dffs or Latches
 FDS                  xcv6          1 x      1      1 Dffs or Latches

 INV                  xcv6          1 x      1      1 LUTs
 LD_1                 xcv6         18 x      1     18 Dffs or Latches
 LUT2                 xcv6        193 x      1    193 LUTs
 LUT3                 xcv6        113 x      1    113 LUTs
 LUT4                 xcv6        548 x      1    548 LUTs
 LUT5                 xcv6        236 x      1    236 LUTs
 LUT6                 xcv6       2061 x      1   2061 LUTs
 MUXF7                xcv6       1216 x      1   1216 MUXF7
 MUXF8                xcv6        544 x      1    544 MUXF8
 modgen_counter_4_0   OPERATORS     1 x      5      5 LUTs
                                             4      4 Dffs or Latches
 ram_dq_128_0         OPERATORS     1 x      8      4 Block RAMs

 Number of ports :                   648
 Number of nets :                   7244
 Number of instances :              6854
 Number of references to this view :   0

Total accumulated area :
 Number of Block RAMs :                4
 Number of Dffs or Latches :        1943
 Number of LUTs :                   3157
 Number of MUXF7 :                  1216
 Number of MUXF8 :                   544
 Number of gates :                  3033
 Number of accumulated instances :  6867
```

7.5 Conclusion

With this module, we can ensure the privacy aspect in any type of communication; naturally, this is not enough to ensure “security” but is a good starting point. Many others features could be add with the help of the software layer (key exchange protocol, authenticity…). A complete project on a secure system needs a deep study of the partitioning between the hardware and software as we have introduced in the EVITA project section.

However also we haven’t ensure all the security aspects, this work can be a starting point for a feasible study on security in a bus communication related to an automotive environment.

Certainly, we can state, that an integration of the privacy feature, on an existing IP could be possible because the module presented reaches higher speeds than the existing automotive microcontrollers and also the module uses “little area” to realize its functions.

Appendix

System Verilog model

```
module AESE(input bit ed, // 0: encryption 1: decryption
        input reset,
        input bit [127:0]msgin,
        input bit [127:0]mkey,
        input bit go,
        output bit [127:0]msgout,
        output bit endop);


parameter HALFWORD = 16;
parameter BYTE = 8;
parameter NIBBLE = 4;
parameter KEYSIZE = 128;
parameter ROUND = 11;
parameter WORD = 32;
parameter REGLENGHT = 10;
parameter REGKEYLENGHT = 44;

integer mcd,number;

bit [BYTE-1:0] S_box [HALFWORD-1:0][HALFWORD-1:0];
bit [BYTE-1:0] X09 [HALFWORD-1:0][HALFWORD-1:0];
bit [BYTE-1:0] X0B [HALFWORD-1:0][HALFWORD-1:0];
bit [BYTE-1:0] X0D [HALFWORD-1:0][HALFWORD-1:0];
bit [BYTE-1:0] X0E [HALFWORD-1:0][HALFWORD-1:0];
bit [BYTE-1:0] IS_box [HALFWORD-1:0][HALFWORD-1:0];
//bit [KEYSIZE-1:0] msg = 128'h3243f6a8885a308d313198a2e0370734;
bit [KEYSIZE-1:0] msge;

//KEYSCHEDULER
bit [WORD-1:0] W [REGKEYLENGHT-1:0];
bit [BYTE-1:0] RC [REGLENGHT-1:0];
int i,j,Nk,Nr,Nb,count,nr;
```

```
bit [WORD-1:0] temp;

bit [BYTE-1:0] SM    [NIBBLE-1:0][NIBBLE-1:0];
bit [BYTE-1:0] SMA   [NIBBLE-1:0][NIBBLE-1:0];
bit [BYTE-1:0] StatematrikE [ROUND-1:0][NIBBLE-1:0][NIBBLE-1:0];
bit [BYTE-1:0] StatematrikD [ROUND-1:0][NIBBLE-1:0][NIBBLE-1:0];

// Clock generator
bit clk;
initial forever #1 clk = !clk;

initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/sbox.txt",S_box);
initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/RC.txt",RC);
initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/invsbox.txt",IS_box);
initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/09.txt",X09);
initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/0B.txt",X0B);
initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/0D.txt",X0D);
initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/0E.txt",X0E);
initial count = 0;

initial begin
 count = 0;
 i = 3;
 Nk= 4;
 Nr= 10;
 Nb= 4;
 nr= 0;
 endop = 0;
end

//_________rotword_____________________________________________________
function automatic  bit[31:0] rotword;
  input bit[WORD-1:0] Wr;
  bit [WORD-1:0] Wr1;
            Wr1[BYTE-1:0]                       = Wr[(BYTE*4)-1:BYTE*3];
            Wr1[(BYTE*2)-1:BYTE]    = Wr[BYTE-1:0];
            Wr1[(BYTE*3)-1:BYTE*2]          = Wr[(BYTE*2)-1:BYTE];
            Wr1[(BYTE*4)-1:BYTE*3]          = Wr[(BYTE*3)-1:BYTE*2];
            //$display ("Current Value of Wr1 = %h", Wr1);
            return Wr1;
```

```
endfunction

//_____subword_____
function automatic   bit[31:0] subword;
 input bit[WORD-1:0] Ws;
 Ws[(BYTE*4)-1:BYTE*3]   = S_box[Ws[(NIBBLE*8)-1:NIBBLE*7]][Ws[(NIBBLE*7)-1:NIBBLE*6]];
          Ws[(BYTE*3)-1:BYTE*2]   = S_box[Ws[(NIBBLE*6)-1:NIBBLE*5]][Ws[(NIBBLE*5)-1:NIBBLE*4]];
          Ws[(BYTE*2)-1:BYTE]      = S_box[Ws[(NIBBLE*4)-1:NIBBLE*3]][Ws[(NIBBLE*3)-1:NIBBLE*2]];
          Ws[BYTE-1:0]                          = S_box[Ws[(NIBBLE*2)-1:NIBBLE]][Ws[NIBBLE-1:0]];
          return Ws;
endfunction

//_____MsgtoState_____
function automatic bit[BYTE-1:0]MsgtoState;
 input bit [KEYSIZE-1:0] msgin;
 input integer i,j;
 bit [BYTE-1:0]SM;

 if(i==3 && j==3)SM  = msgin[7:0];
          if(i==2 && j==3)SM  = msgin[15:8];
          if(i==1 && j==3)SM       = msgin[23:16];
          if(i==0 && j==3)SM       = msgin[31:24];
          if(i==3 && j==2)SM  = msgin[39:32];
          if(i==2 && j==2)SM       = msgin[47:40];
          if(i==1 && j==2)SM  = msgin[55:48];
          if(i==0 && j==2)SM       = msgin[63:56];
          if(i==3 && j==1)SM       = msgin[71:64];
          if(i==2 && j==1)SM       = msgin[79:72];
          if(i==1 && j==1)SM       = msgin[87:80];
          if(i==0 && j==1)SM       = msgin[95:88];
          if(i==3 && j==0)SM       = msgin[103:96];
          if(i==2 && j==0)SM       = msgin[111:104];
          if(i==1 && j==0)SM       = msgin[119:112];
          if(i==0 && j==0)SM       = msgin[127:120];
 return SM;
endfunction

bit clock;
initial begin clock = 1'b0;
forever #1 clock = !clock;
```

```
end

//_____StatetoMsg_____
function automatic bit[KEYSIZE-1:0]StatetoMsg;
input bit [BYTE-1:0]SM[NIBBLE-1:0][NIBBLE-1:0];
bit [KEYSIZE-1:0]msg;
msg =
{SM[3][3],SM[2][3],SM[1][3],SM[0][3],SM[3][2],SM[2][2],SM[1][2],SM[0][2],SM[3][1],SM[2][1],SM[1][1],SM[0][1
],SM[3][0],SM[2][0],SM[1][0],SM[0][0]};
return msg;
endfunction

//_____KeytoStateE_____
function automatic bit[BYTE-1:0]KeytoStateE;
 input bit [WORD-1:0] Wkts [REGKEYLENGHT-1:0];
 input integer n,i,j;
 bit [BYTE-1:0]Ke;
 case (j)
  0:Ke = i==3?Wkts[((n+1)*4)-4][7:0]:i==2?Wkts[((n+1)*4)-4][15:8]:i==1?Wkts[((n+1)*4)-
4][23:16]:Wkts[((n+1)*4)-4][31:24];
  1:Ke = i==3?Wkts[((n+1)*4)-3][7:0]:i==2?Wkts[((n+1)*4)-3][15:8]:i==1?Wkts[((n+1)*4)-
3][23:16]:Wkts[((n+1)*4)-3][31:24];
  2:Ke = i==3?Wkts[((n+1)*4)-2][7:0]:i==2?Wkts[((n+1)*4)-2][15:8]:i==1?Wkts[((n+1)*4)-
2][23:16]:Wkts[((n+1)*4)-2][31:24];
  3:Ke = i==3?Wkts[((n+1)*4)-1][7:0]:i==2?Wkts[((n+1)*4)-1][15:8]:i==1?Wkts[((n+1)*4)-
1][23:16]:Wkts[((n+1)*4)-1][31:24];
 endcase
 return Ke;
endfunction

//_____KeytoStateD_____
function automatic bit[BYTE-1:0]KeytoStateD;
 input bit [WORD-1:0] Wkts [REGKEYLENGHT-1:0];
 input integer n,i,j;
 bit [BYTE-1:0]Ke;
 case (j)
  0:Ke = i==3?Wkts[((11-n)*4)-4][7:0]:i==2?Wkts[((11-n)*4)-4][15:8]:i==1?Wkts[((11-n)*4)-
4][23:16]:Wkts[((11-n)*4)-4][31:24];
  1:Ke = i==3?Wkts[((11-n)*4)-3][7:0]:i==2?Wkts[((11-n)*4)-3][15:8]:i==1?Wkts[((11-n)*4)-
3][23:16]:Wkts[((11-n)*4)-3][31:24];
```

```
  2:Ke          =          i==3?Wkts[((11-n)*4)-2][7:0]:i==2?Wkts[((11-n)*4)-2][15:8]:i==1?Wkts[((11-n)*4)-
2][23:16]:Wkts[((11-n)*4)-2][31:24];
  3:Ke          =          i==3?Wkts[((11-n)*4)-1][7:0]:i==2?Wkts[((11-n)*4)-1][15:8]:i==1?Wkts[((11-n)*4)-
1][23:16]:Wkts[((11-n)*4)-1][31:24];
 endcase
 return Ke;
endfunction

//__________AddRoundKey_____________________________________________
function automatic bit[BYTE-1:0]AddRoundKey;
 input bit [BYTE-1:0]SE;
 input bit [BYTE-1:0]SEkey;
 bit     [BYTE-1:0]SE1;
 SE1= SE^SEkey;
          return SE1;
endfunction

//__________SubBytes_____________________________________________
function automatic bit[BYTE-1:0]SubBytes;
 input bit [BYTE-1:0]SE;
 bit     [BYTE-1:0]SE1;
 SE1 = S_box[SE[7:4]][SE[3:0]];
 return SE1;
endfunction

//__________ISubBytes_____________________________________________
function automatic bit[BYTE-1:0]ISubBytes;
 input bit [BYTE-1:0]SE;
 bit     [BYTE-1:0]SE1;
 SE1 = IS_box[SE[7:4]][SE[3:0]];
 return SE1;
endfunction

//__________ShiftRows_____________________________________________
function automatic  bit[BYTE-1:0]ShiftRows;
 input bit [BYTE-1:0]SMA[NIBBLE-1:0][NIBBLE-1:0];
 input integer i,j;
 bit     [BYTE-1:0]SE1[NIBBLE-1:0][NIBBLE:0];
 case(i)
  0:SE1[i][j] = SMA[i][j];
```

```
   1:SE1[i][j] = SMA[i][(j+1)%4];
   2:SE1[i][j] = SMA[i][(j+2)%4];
   3:SE1[i][j] = SMA[i][(j+3)%4];
  endcase
  return SE1[i][j];
endfunction

//__________IShiftRows_____________________________________________
function automatic  bit[BYTE-1:0]IShiftRows;
  input bit [BYTE-1:0]SMA[NIBBLE-1:0][NIBBLE-1:0];
  input integer i,j;
  bit      [BYTE-1:0]SE1[NIBBLE-1:0][NIBBLE:0];
  case(i)
    0:SE1[i][j] = SMA[i][j];
    1:SE1[i][j] = SMA[i][(j+3)%4];
    2:SE1[i][j] = SMA[i][(j+2)%4];
    3:SE1[i][j] = SMA[i][(j+1)%4];
  endcase
  return SE1[i][j];
endfunction

//__________MixColumns_____________________________________________
function automatic  bit[BYTE-1:0]MixColumns;
  input bit [BYTE-1:0]SMA[NIBBLE-1:0][NIBBLE-1:0];
  bit [BYTE-1:0]SMA1[NIBBLE-1:0][NIBBLE-1:0]=SMA;
  input integer i,j;
  bit      [BYTE-1:0]SE1;
  case(i)
    0:SE1 = (xtime(SMA[0][j]))^(xtime(SMA[1][j])^SMA1[1][j])^SMA[2][j]^SMA[3][j];
    1:SE1 = SMA[0][j]^xtime(SMA[1][j])^(xtime(SMA[2][j])^SMA1[2][j])^SMA[3][j];
    2:SE1 = SMA[0][j]^SMA[1][j]^xtime(SMA[2][j])^(xtime(SMA[3][j])^SMA1[3][j]);
    3:SE1 = (xtime(SMA[0][j])^SMA1[0][j])^SMA[1][j]^SMA[2][j]^xtime(SMA[3][j]);
  endcase
  return SE1;
endfunction

//__________IMixColumns_____________________________________________
function automatic  bit[BYTE-1:0]IMixColumns;
  input bit [BYTE-1:0]SM[NIBBLE-1:0][NIBBLE-1:0];
  input integer i,j;
```

```
 bit     [BYTE-1:0]SE1;
 case(i)
  0:SE1                                                                                              =
X0E[SM[0][j][7:4]][SM[0][j][3:0]]^X0B[SM[1][j][7:4]][SM[1][j][3:0]]^X0D[SM[2][j][7:4]][SM[2][j][3:0]]^X09[SM
[3][j][7:4]][SM[3][j][3:0]];
  1:SE1                                                                                              =
X09[SM[0][j][7:4]][SM[0][j][3:0]]^X0E[SM[1][j][7:4]][SM[1][j][3:0]]^X0B[SM[2][j][7:4]][SM[2][j][3:0]]^X0D[SM
[3][j][7:4]][SM[3][j][3:0]];
  2:SE1                                                                                              =
X0D[SM[0][j][7:4]][SM[0][j][3:0]]^X09[SM[1][j][7:4]][SM[1][j][3:0]]^X0E[SM[2][j][7:4]][SM[2][j][3:0]]^X0B[SM
[3][j][7:4]][SM[3][j][3:0]];
                3:SE1                                                                                =
X0B[SM[0][j][7:4]][SM[0][j][3:0]]^X0D[SM[1][j][7:4]][SM[1][j][3:0]]^X09[SM[2][j][7:4]][SM[2][j][3:0]]^X0E[SM
[3][j][7:4]][SM[3][j][3:0]];
 endcase
 return SE1;
endfunction

//__________xtime_____________________________________________________
function automatic  bit[BYTE-1:0]xtime;
 input bit [BYTE-1:0]SE;
 bit     [BYTE-1:0]SE1;
 SE1 = (SE&8'h80)?SE<<1^8'h1B:SE<<1;
 return SE1;
endfunction

//__________printSM_____________________________________________________
function void  printSM;
 input bit [BYTE-1:0]SE;
 input integer i,j;
 if (j == 0)$fwrite (mcd , "%h\n", SE);
 else if (j == 0 && i == 0)$fwrite (mcd, "\n\n");
 else $fwrite (mcd , "%h\t", SE);
endfunction

//__________CORE_____________________________________________________
always@(posedge clk) begin
  if(reset == 1)begin
   count = 0;
   i = 3;
```

```
    nr= 0;
    endop = 0;
  end
  if (count<7 && go == 0)count++;
  else if (go == 0 && ed == 0 && count < 26 && count >= 7)count++;
  else if (go == 0 && ed == 1 && count < 46 && count >= 27)count++;
  mcd = $fopen("logModel.txt");
  W[0] = mkey[127:96];
  W[1] = mkey[95:64];
  W[2] = mkey[63:32];
  W[3] = mkey[31:0];
  case(count)
    0:;
//////////////////////////KEY SCHEDULE\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    1:begin
       if (i < Nb*(Nr+1))i++;
       else begin
        if (ed == 0 && go == 0)count = 6;
        else if (ed == 1 && go == 0)count = 27;
        foreach (SM[i,j])SM[i][j] = MsgtoState(msgin,i,j);
        if (i != 2*Nb*(Nr+1))i = 0;
       end
     end
    2:temp = W[i-1];
    3:if (i % Nk == 0 && i < 44)temp = rotword(temp);
    4:if (i % Nk == 0 && i < 44)temp = subword(temp);
    5:if (i % Nk == 0 && i < 44)temp[31:24] = temp[31:24]^RC[(i/Nk)-1];
    6:begin
     W[i] = W[i-Nk] ^ temp;
     count = 0;
     end
    7:;
//////////////////////////ENCRYPTION\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    8:foreach  (StatematrikE[n,i,j])StatematrikE[n][i][j] = KeytoStateE(W,n,i,j);
    9:begin
       $fwrite (mcd , "SM encryption %d\n\n", nr);
       foreach  (SM[i,j])printSM(SM[i][j],i,j);
      end
    10:foreach (SM[i,j])SM[i][j]    = AddRoundKey(SM[i][j],StatematrikE[nr][i][j]);
    11:foreach (SM[i,j])SM[i][j]    = SubBytes(SM[i][j]);
```

```
12:foreach (SM[i,j])SMA[i][j]   = ShiftRows(SM,i,j);
13:foreach (SM[i,j])SM[i][j]    = SMA[i][j];
14:if(nr < 9)foreach (SM[i,j])SMA[i][j]  = MixColumns(SM,i,j);
15:begin
   foreach (SM[i,j])SM[i][j]   = SMA[i][j];
   nr++;
  end
16:if(nr != 10)count = 8;
17:begin
   $fwrite (mcd , "SM %d\n\n", nr);
   foreach (SM[i,j])begin
                if (j == 0)$fwrite (mcd , "%h\n", SM[3-i][3-j]);
                else if (j == 0 && i == 0)$fwrite (mcd, "\n");
                else $fwrite (mcd , "%h\t", SM[3-i][3-j]);
               end
  end
18:foreach  (SM[i,j])SM[i][j]   = AddRoundKey(SM[i][j],StatematrikE[10][i][j]);
19:begin
   $fwrite (mcd , "output%d\n\n", nr);
   foreach (SM[i,j])begin
                if (j == 0)$fwrite (mcd , "%h\n", SM[3-i][3-j]);
                else if (j == 0 && i == 0)$fwrite (mcd, "\n");
                else $fwrite (mcd , "%h\t", SM[3-i][3-j]);
               end
  end
20:begin
   msgout = StatetoMsg(SM);
   endop = 1;
  end
21:;
22:;
23:;
24:;
25:;
26:begin
   if(go == 1)begin
    i = Nb*(Nr+1);
    count = 1;
    nr = 0;
    endop = 0;
```

```
        foreach (SM[i,j])SM[i][j] = 0;
        foreach (SMA[i,j])SMA[i][j] = 0;
       end
      end
    27:;
//////////////////////////DECRYPTION\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
    28:foreach  (StatematrikD[n,i,j])StatematrikD[n][i][j] = KeytoStateD(W,n,i,j);
    29:begin
       $fwrite (mcd , "SM Decryption %d\n\n", nr);
       foreach  (SM[i,j])printSM(SM[3-i][3-j],i,j);//print
       $display ("%h\t%h\t%h\n",SM[0][0],StatematrikD[nr][0][0],SM[0][0]^StatematrikD[nr][0][0]);
      end
    30:foreach  (SM[i,j])SM[i][j]   = AddRoundKey(SM[i][j],StatematrikD[nr][i][j]);

    31:if(nr != 0)foreach (SMA[i,j])SMA[i][j]  = IMixColumns(SM,i,j);
    32:if(nr != 0)foreach (SM[i,j])SM[i][j]   = SMA[i][j];
    33:foreach (SM[i,j])SMA[i][j]   = IShiftRows(SM,i,j);
    34:foreach (SM[i,j])SM[i][j]    = SMA[i][j];
    35:begin
       foreach (SM[i,j])SM[i][j]   = ISubBytes(SM[i][j]);
       nr++;
      end
    36:if(nr != 10)count = 28;
    37:foreach  (SM[i,j])SM[i][j]   = AddRoundKey(SM[i][j],StatematrikD[10][i][j]);
    38:;
    39:begin
       $fwrite (mcd , "output%d\nn", nr);
       foreach  (SM[i,j])printSM(SM[3-i][3-j],i,j);//print
      end
    40:begin
       msgout = StatetoMsg(SM);
       endop = 1;
      end
    41:;
    42:;
    43:;
    44:;
    45:;
    46:begin
       if(go == 1)begin
```

```
          i = 2*Nb*(Nr+1);
          count = 1;
          nr = 0;
          foreach (SM[i,j])SM[i][j] = 0;
          foreach (SMA[i,j])SMA[i][j] = 0;
          endop = 0;
        end
       end
     47:;
   endcase
 end

endmodule
```

System Verilog test bench

```
module tbsv();

 parameter KEYSIZE = 128;

 bit ed;
 bit go;
 bit [KEYSIZE-1:0]msginM;
 bit [KEYSIZE-1:0]msgoutM;
 bit endopM;
        bit clk,reset_n,en_n,enmE_n,enmD_n,resetM;
        bit [KEYSIZE-1:0]mkey;
        bit [KEYSIZE-1:0]msginE;
        bit [KEYSIZE-1:0]msginD;
        bit [KEYSIZE-1:0]masterk;

        wire ack_k,ack_a,ack_aD;
        wire [KEYSIZE-1:0]msgoutE;
        wire [KEYSIZE-1:0]msgoutD;

 Enc_Dec_tl                                                                    E0
(clk,reset_n,en_n,enmE_n,enmD_n,mkey,msginE,msginD,msgoutE,msgoutD,ack_k,ack_a,ack_aD);
 AESE     M0 (ed,resetM,msginM,mkey,go,msgoutM,endopM);

 class pack;
   rand bit [KEYSIZE-1:0]plaint;
   rand bit [KEYSIZE-1:0]encrypt;
   rand integer operationmode;  //1:encrypt   2:decrypt   3:encrypt/decrypt
   constraint c {operationmode >0;
            operationmode <4;}
 endclass

 integer round,iter;
 integer mcd;
 integer prev;
 integer count = 0;
```

```
  pack p;


// _______Generazione clock_____________________________
        always
                begin
                        #2 clk = !clk;
                end
// _______Generazione random____________________________
        initial
          begin
            p = new();
                         assert(p.randomize());
                         masterk = $urandom();
                         masterk =masterk*112'hFCFBFA27B2E3812FDBFFCDF42971;
          end
// ___condizioni iniziali_____________________________
  initial
    begin
    iter = 4;
    go = 1;
    clk = 0;
                        reset_n = 0;
                        en_n = 1;
                        enmE_n = 1;
                        enmD_n = 1;
                        mkey = 128'h0;
                        msginD =128'h0;
                        round = 0;
                        mcd = $fopen("logtb.txt");
                        $fwrite (mcd , "Test bench AES 128 \n");
                        $fwrite (mcd , "Master Key %h\n", masterk,"t =",$time,"\n");
    end

        always @(posedge clk)
                        begin
                          case(count)
```

```
                    0:begin
                      #4 reset_n = 1;
                      if (iter != 0)en_n = 0;
                      if (iter != 0)go = 0;
                      mkey = masterk;
                      prev = p.operationmode;
                      if (iter == 0)begin
                       resetM = 1;
                       reset_n = 0;
                       masterk =$urandom();
                                                masterk
=masterk*112'hFCFBFA27B2E3812FDBFFCDF42971;
                       iter = 4;
                       $fwrite (mcd , "Cambio della Master Key \n","t =",$time,"\n");
                       #10count = 6;
                      end
                      if (endopM == 1 && iter != 0)$fwrite (mcd , "Test n:  %d\t", round);
                      if(p.operationmode == 1)begin
                       ed = 0;
                       msginM = p.plaint;
                       msginD = p.plaint;
                       if (endopM == 1)$fwrite (mcd , "Encryption\n");
                       if (endopM == 1)$fwrite (mcd , "Messaggio da cifrare : %h\n", p.plaint);
                      end
                      else if(p.operationmode == 2 && iter != 0)begin
                       ed = 1;
                       msginM = p.encrypt;
                       msginE = p.encrypt;
                       if (endopM == 1)$fwrite (mcd , "Decryption\n");
                       if (endopM == 1)$fwrite (mcd , "Messaggio da decifrare : %h\n",
p.encrypt);
                      end
                      if (endopM == 1 && iter != 0)begin
                       $fwrite (mcd , "Modello completato \n");
                       count = 1;
                      end
```

```
 end
6:begin
   $fwrite (mcd , "Master Key %h\n", masterk);
   resetM = 0;
   reset_n = 1;
   count = 0;
   mkey = masterk;
 end
1:begin
   if(ack_k == 0 && ed == 0)begin
    enmE_n = 0;
    #10 count = 3;
   end
   else if(ack_k == 0 && ed == 1)begin
    enmD_n = 0;
    #10 count = 3;
   end
 end

3:begin
   enmE_n = 1;
   enmD_n = 1;
   if (ack_a == 1 || ack_aD == 1)begin
    go = 1;
    #5 count = 4;
   end
 end
4:begin
    round ++;
    iter --;
    $display("msgoutM := %h\n", msgoutM);
    $display("msgoutD := %h\n", msgoutD);
    $display("msgoutE := %h\n", msgoutE);
    if (p.operationmode == 1)begin
     assert(p.randomize());
     if (msgoutE == msgoutM)begin
      $fwrite (mcd , "cifratura avvenuta correttamente \n");
```

```
                $fwrite (mcd , "Messaggio cifrato : %h\n", msgoutE);
               end
               else $fwrite (mcd , "cifratura non avvenuta correttamente \n");
              end
              else begin
               assert(p.randomize());
               if (msgoutD == msgoutM)begin
                $fwrite (mcd , "decifratura avvenuta correttamente \n");
                $fwrite (mcd , "Messaggio decifrato : %h\n", msgoutD);
               end
               else $fwrite (mcd , "decifratura non avvenuta correttamente \n");
              end
              #5 count = 0;
             end


        endcase
    end

endmodule
```

Verilog module (version 1 encryption only)

```
//------------------------------------------------------------------------------
// key_schedule ultimate
//------------------------------------------------------------------------------

module Key_schedule (//inputs
                        clk,
                        reset_n,
                        en_n,
                        keyin,
                     //outputs
                        key,
                        key_ready
                    );

    function integer clogb2;//log base 2
        input [31:0] value;
        integer     i;
        begin
            clogb2 = 0;
            for(i = 0; 2**i < value; i = i + 1)
                clogb2 = i + 1;
        end
    endfunction

    parameter WORD = 32;
    parameter HALFWORD = 16;
    parameter BYTE = 8;
    parameter NIBBLE = 4;
    parameter REGLENGHT = 10;
    parameter REGKEYLENGHT = 44;
    parameter KEYSIZE = 128;
    parameter ROUND = 11;


    input   [KEYSIZE-1:0]keyin;
    input   clk,reset_n,en_n;
    output  [KEYSIZE-1:0]key;
    output  key_ready;

    reg [KEYSIZE-1:0] key;
    reg key_ready;
    reg [BYTE-1:0] S_box [HALFWORD-1:0][HALFWORD-1:0];
    reg [BYTE-1:0] RC [REGLENGHT-1:0];
    reg [clogb2(HALFWORD)-1:0] i,m,state,nextstate;
    reg [WORD-1:0] wordap1, wordapr1,rotates;
    reg [WORD-1:0] W [REGKEYLENGHT-1:0];
```

```
    parameter IDLE  =  4'h0;
    parameter M_KEY =  4'h1;
    parameter KEY_S0 = 4'h2;
    parameter KEY_S1 = 4'h3;
    parameter KEY_S2 = 4'h4;
    parameter KEY_S3 = 4'h5;
    parameter KEY_S4 = 4'h6;
    parameter KEY_S5 = 4'h7;
    parameter KEY_S6 = 4'h8;
    parameter KEY_S7 = 4'h9;
    parameter KEY_S8 = 4'ha;
    parameter KEYT  =  4'hb;

//____________________status register_____________________________________
    always@(posedge clk)
        begin
            if(reset_n == 0)
                begin
                    state <= IDLE;
                end
            else
                begin
                    state <= nextstate;
                end
        end
//____________________state transition_____________________________________
    always@(*)
        begin
            case(state)

                IDLE:   begin
                            if (en_n == 0) nextstate = M_KEY;
                            else nextstate = IDLE;
                        end
                M_KEY:  nextstate = KEY_S0;
                KEY_S0: nextstate = KEY_S1;
                KEY_S1: nextstate = KEY_S2;
                KEY_S2: nextstate = KEY_S3;
                KEY_S3: nextstate = KEY_S4;
                KEY_S4: nextstate = KEY_S5;
                KEY_S5: nextstate = KEY_S6;
                KEY_S6: nextstate = KEY_S7;
                KEY_S7: begin
                            if (i == ROUND -1 ) nextstate = KEY_S8;
                            else nextstate = KEY_S0;
                        end

                KEY_S8: nextstate = KEYT;
                KEYT  :;
```

```
98
99                  default: nextstate = IDLE;
100
101             endcase
102         end
103
104     initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/sbox.txt",S_box);
105     initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/RC.txt",RC);
106
107 //____________________core_____________________________________________________
108     always@(*)
109         begin
110             key_ready = 1;
111             key = 128'h0;
112             case(state)
113
114                 IDLE:;
115                 M_KEY:;
116 //____________________keys generation (start)_____________________________________
117                 KEY_S0:;
118                 KEY_S1:;
119                 KEY_S2:;
120                 KEY_S3:;
121                 KEY_S4:;
122                 KEY_S5:;
123                 KEY_S6:;
124                 KEY_S7:;
125                 KEY_S8:key_ready = 0;
126 //____________________keys generation (end)_____________________________________
127 //____________________keys sending_____________________________________
128                 KEYT:   begin
129                             if(en_n == 0)
130                                 begin
131                                 if(m <= ROUND-1)key = {W[0+4*m],W[1+4*m],W[2+4*m],W[3+4*m]};
132                                     else key = 128'h0;
133                                     key_ready = 0;
134                                 end
135                             else
136                                 begin
137                                     key_ready = 1;
138                                     key = 128'h0;
139                                 end
140                         end
141
142                 default: begin
143                           key_ready = 1;
144                           key = 128'h0;
145                         end
146             endcase
```

```
147         end
148
149     always@(posedge clk)
150         begin
151  //_____________________IDLE state_____________________________________________
152             if(state == IDLE)
153                 begin
154                     i <= 4'h1;
155                     m <= 4'h0;
156                 end
157  //_____________________acqusition master key_____________________________________
158             if (state == M_KEY)
159                 begin
160                     W[0] <= keyin[(4*WORD)-1:3*WORD];
161                     W[1] <= keyin[(3*WORD)-1:2*WORD];
162                     W[2] <= keyin[(2*WORD)-1:WORD];
163                     W[3] <= keyin[WORD-1:0];
164                 end
165  //_____________________keys generation (start)_____________________________________
166             if(state == KEY_S0) rotates <= W[(4*i)-1];
167             if(state == KEY_S1)
168                 begin
169                 wordapr1[BYTE-1:0]  <= rotates[(BYTE*4)-1:BYTE*3];
170                 wordapr1[(BYTE*2)-1:BYTE]   <= rotates[BYTE-1:0];
171                 wordapr1[(BYTE*3)-1:BYTE*2] <= rotates[(BYTE*2)-1:BYTE];
172                 wordapr1[(BYTE*4)-1:BYTE*3] <= rotates[(BYTE*3)-1:BYTE*2];
173                 end
174             if(state == KEY_S2) rotates <= wordapr1;
175             if(state == KEY_S3)
176                 begin
177             wordap1[(BYTE*4)-1:BYTE*3]<= S_box[rotates[(NIBBLE*8)-1:NIBBLE*7]]
178             [rotates[(NIBBLE*7)-1:NIBBLE*6]]^ RC[i-1];
179             wordap1[(BYTE*3)-1:BYTE*2]<= S_box[rotates[(NIBBLE*6)-1:NIBBLE*5]]
180             [rotates[(NIBBLE*5)-1:NIBBLE*4]];
181             wordap1[(BYTE*2)-1:BYTE]  <= S_box[rotates[(NIBBLE*4)-1:NIBBLE*3]]
182             [rotates[(NIBBLE*3)-1:NIBBLE*2]];
183             wordap1[BYTE-1:0]         <= S_box[rotates[(NIBBLE*2)-1:NIBBLE]]
184             [rotates[NIBBLE-1:0]];
185                 end
186             if(state == KEY_S4) W[4*i] <= W[4*(i-1)] ^ wordap1;
187             if(state == KEY_S5) W[4*i + 1] <=  W[4*i + 0] ^ W[4*(i-1) + 1];
188             if(state == KEY_S6) W[4*i + 2] <=  W[4*i + 1] ^ W[4*(i-1) + 2];
189             if(state == KEY_S7)
190                 begin
191                     if (i == ROUND -1) begin
192                         wordapr1 <= 32'h0;
193                         wordap1  <= 32'h0;
194                         rotates  <= 32'h0;
195                     end
196                     W[4*i + 3] <=  W[4*i + 2] ^ W[4*(i-1) + 3];
```

```
186             if(state == KEY_S4) W[4*i] <= W[4*(i-1)] ^ wordap1;
187             if(state == KEY_S5) W[4*i + 1] <=  W[4*i + 0] ^ W[4*(i-1) + 1];
188             if(state == KEY_S6) W[4*i + 2] <=  W[4*i + 1] ^ W[4*(i-1) + 2];
189             if(state == KEY_S7)
190                 begin
191                     if (i == ROUND -1) begin
192                         wordapr1 <= 32'h0;
193                         wordap1  <= 32'h0;
194                         rotates  <= 32'h0;
195                     end
196                     W[4*i + 3] <=  W[4*i + 2] ^ W[4*(i-1) + 3];
197                     i <= i + 1;
198 
199                 end
200 //____________________keys generation (end)_____________________________________
201 //____________________keys sending_____________________________________
202             if (state == KEYT && en_n == 0)
203                 begin
204                     if( m <= ROUND) m <= m + 4'h1;
205                     else m <= ROUND + 1;
206                 end
207 
208         end
```

```
//------------------------------------------------------------------------------
// AES128coreE ultimate
//------------------------------------------------------------------------------

module AES128coreE (//inputs
                    clk,
                    reset_n,
                    ena_n,
                    enm_n,
                    msgin,
                    keyin,
                    //outputs
                    msgout,
                    ack
                    );

    function integer clogb2;//log base 2
        input [31:0] value;
        integer     i;
        begin
            clogb2 = 0;
            for(i = 0; 2**i < value; i = i + 1)
                clogb2 = i + 1;
        end
    endfunction

    parameter HALFWORD = 16;
    parameter BYTE = 8;
    parameter KEYSIZE = 128;
    parameter ROUND = 11;

    input [KEYSIZE-1:0] msgin,keyin;
    input ena_n,enm_n,clk,reset_n;

    output [KEYSIZE-1:0] msgout;
    output ack;

    reg [KEYSIZE-1:0] msgout;
    reg  ack;
    reg [BYTE-1:0] S_box[HALFWORD-1:0][HALFWORD-1:0];
    reg [BYTE-1:0] SM    [clogb2(HALFWORD)-1:0][clogb2(HALFWORD)-1:0];
    reg [BYTE-1:0] Statematrik [ROUND-1:0][clogb2(HALFWORD)-1:0][clogb2(HALFWORD)-1:0];
    reg [BYTE-1:0] mask8;
    reg [HALFWORD-1:0]M7;
    reg [clogb2(HALFWORD)-1:0] m,nr,state,nextstate;

    parameter IDLE  =   4'h0;
    parameter KEYR  =   4'h1; //key acquisition
    parameter MSG   =   4'h2; //acquisition message in
    parameter KAL   =   4'h3; //key addition layer
```

```
52      parameter BSL   =   4'h4; //byte substitution layer
53      parameter SRL   =   4'h5; //shift row layer
54      parameter MCL   =   4'h6; //mix column layer
55      parameter MSGO  =   4'h7; //msg out
56
57      always@(posedge clk)
58          begin
59              if(reset_n == 0)
60                  begin
61                      state <= IDLE;
62                  end
63              else
64                  begin
65                      state <= nextstate;
66                  end
67          end
68
69      always@(*)
70          begin
71              case(state)
72
73                  IDLE:   if (ena_n == 0) nextstate = KEYR;
74                          else nextstate = IDLE;
75                  // key acquisition
76                  KEYR:   if (m == ROUND-1 && ena_n == 0) nextstate = MSG;
77                          else nextstate = KEYR;
78                  //acquisition message in
79                  MSG:    if (ena_n == 0 && enm_n == 0)nextstate = KAL;
80                          else nextstate = MSG;
81                  //key addition layer
82                  KAL   : begin
83                              if(ena_n == 0)
84                                  begin
85                                      if ( nr == ROUND-1)nextstate = MSGO;
86                                      else nextstate = BSL;
87                                  end
88                              else nextstate = KAL;
89                          end
90                  //byte substitution layer
91                  BSL   : if(ena_n == 0)nextstate = SRL;
92                          else nextstate = BSL;
93                  //shift row layer
94                  SRL   : begin
95                              if(ena_n == 0)
96                                  begin
97                                      if (nr < ROUND-1) nextstate = MCL;
98                                      else if (nr == ROUND-1) nextstate = KAL;
99                                      else nextstate = SRL;
100                                 end
```

```
                        else nextstate = SRL;
                    end
                //mix column layer
                MCL   : if(ena_n == 0)nextstate = KAL;
                        else nextstate = MCL;
                //msg out
                MSGO  : if (enm_n == 1 && ena_n == 0) nextstate = MSG;
                        else nextstate = MSGO;
            endcase
        end

initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/sbox.txt",S_box);

    always@(*)
        begin
            if (ena_n == 0)
                begin
                    ack = 0;
                    M7 = 16'h0;
                    case(state)
                        IDLE:;
                        KEYR:if(m == ROUND-1) ack = 1;
                        MSG:;
                        KAL:;
                        BSL:;
                        SRL:;
                        MCL:begin
                                M7[0] = |((SM[0][0] & mask8)>>7);
                                M7[1] = |((SM[1][0] & mask8)>>7);
                                M7[2] = |((SM[2][0] & mask8)>>7);
                                M7[3] = |((SM[3][0] & mask8)>>7);
                                M7[4] = |((SM[0][1] & mask8)>>7);
                                M7[5] = |((SM[1][1] & mask8)>>7);
                                M7[6] = |((SM[2][1] & mask8)>>7);
                                M7[7] = |((SM[3][1] & mask8)>>7);
                                M7[8] = |((SM[0][2] & mask8)>>7);
                                M7[9] = |((SM[1][2] & mask8)>>7);
                                M7[10] = |((SM[2][2] & mask8)>>7);
                                M7[11] = |((SM[3][2] & mask8)>>7);
                                M7[12] = |((SM[0][3] & mask8)>>7);
                                M7[13] = |((SM[1][3] & mask8)>>7);
                                M7[14] = |((SM[2][3] & mask8)>>7);
                                M7[15] = |((SM[3][3] & mask8)>>7);
                            end
                        MSGO:begin
                                ack = 1;
                                M7 = 16'h0;
                            end

```

```
150                         default: begin
151                                     ack = 0;
152                                     M7 = 16'h0;
153                                 end
154                     endcase
155             end
156             else
157                 begin
158                     ack = 0;
159                     M7 = 16'h0;
160                 end
161         end
162
163
164     always@(posedge clk)
165         begin
166             if(ena_n==0)
167                 begin
168 //_________________________IDLE state_____________________________________________
169                     if(state == IDLE)
170                         begin
171                             msgout  <= 128'h0;
172                             m       <= 4'h0;
173                             nr      <= 4'h0;
174                             mask8   <= KEYSIZE;
175                         end
176
177 //_________________________acqusition KEY_____________________________________________
178                     if (state == KEYR)
179                         begin
180                             if(m < ROUND-1)m <= m + 4'h1;
181                             Statematrik [m][3][3]   <= keyin[7:0];
182                             Statematrik [m][2][3]   <= keyin[15:8];
183                             Statematrik [m][1][3]   <= keyin[23:16];
184                             Statematrik [m][0][3]   <= keyin[31:24];
185                             Statematrik [m][3][2]   <= keyin[39:32];
186                             Statematrik [m][2][2]   <= keyin[47:40];
187                             Statematrik [m][1][2]   <= keyin[55:48];
188                             Statematrik [m][0][2]   <= keyin[63:56];
189                             Statematrik [m][3][1]   <= keyin[71:64];
190                             Statematrik [m][2][1]   <= keyin[79:72];
191                             Statematrik [m][1][1]   <= keyin[87:80];
192                             Statematrik [m][0][1]   <= keyin[95:88];
193                             Statematrik [m][3][0]   <= keyin[103:96];
194                             Statematrik [m][2][0]   <= keyin[111:104];
195                             Statematrik [m][1][0]   <= keyin[119:112];
196                             Statematrik [m][0][0]   <= keyin[127:120];
197                         end
198 //_____________________acquisition MSG_____________________________________________
199                     if (state == MSG && enm_n == 0)
```

```
                    begin
                        m <= 4'h0;
                        SM [3][3]   <= msgin[7:0];
                        SM [2][3]   <= msgin[15:8];
                        SM [1][3]   <= msgin[23:16];
                        SM [0][3]   <= msgin[31:24];
                        SM [3][2]   <= msgin[39:32];
                        SM [2][2]   <= msgin[47:40];
                        SM [1][2]   <= msgin[55:48];
                        SM [0][2]   <= msgin[63:56];
                        SM [3][1]   <= msgin[71:64];
                        SM [2][1]   <= msgin[79:72];
                        SM [1][1]   <= msgin[87:80];
                        SM [0][1]   <= msgin[95:88];
                        SM [3][0]   <= msgin[103:96];
                        SM [2][0]   <= msgin[111:104];
                        SM [1][0]   <= msgin[119:112];
                        SM [0][0]   <= msgin[127:120];
                    end
//__________________key addition layer ____________________________________________
                if (state == KAL)
                    begin
                        if (nr < ROUND-1) nr <= nr + 4'h1;
                        SM[0][0] <= SM[0][0] ^ Statematrik[nr][0][0];
                        SM[1][0] <= SM[1][0] ^ Statematrik[nr][1][0];
                        SM[2][0] <= SM[2][0] ^ Statematrik[nr][2][0];
                        SM[3][0] <= SM[3][0] ^ Statematrik[nr][3][0];
                        SM[0][1] <= SM[0][1] ^ Statematrik[nr][0][1];
                        SM[1][1] <= SM[1][1] ^ Statematrik[nr][1][1];
                        SM[2][1] <= SM[2][1] ^ Statematrik[nr][2][1];
                        SM[3][1] <= SM[3][1] ^ Statematrik[nr][3][1];
                        SM[0][2] <= SM[0][2] ^ Statematrik[nr][0][2];
                        SM[1][2] <= SM[1][2] ^ Statematrik[nr][1][2];
                        SM[2][2] <= SM[2][2] ^ Statematrik[nr][2][2];
                        SM[3][2] <= SM[3][2] ^ Statematrik[nr][3][2];
                        SM[0][3] <= SM[0][3] ^ Statematrik[nr][0][3];
                        SM[1][3] <= SM[1][3] ^ Statematrik[nr][1][3];
                        SM[2][3] <= SM[2][3] ^ Statematrik[nr][2][3];
                        SM[3][3] <= SM[3][3] ^ Statematrik[nr][3][3];
                    end
//__________________byte substitution layer ____________________________________
                if (state == BSL)
                    begin
                        SM[0][0]  <= S_box[SM[0][0][7:4]][SM[0][0][3:0]];
                        SM[1][0]  <= S_box[SM[1][0][7:4]][SM[1][0][3:0]];
                        SM[2][0]  <= S_box[SM[2][0][7:4]][SM[2][0][3:0]];
                        SM[3][0]  <= S_box[SM[3][0][7:4]][SM[3][0][3:0]];
                        SM[0][1]  <= S_box[SM[0][1][7:4]][SM[0][1][3:0]];
                        SM[1][1]  <= S_box[SM[1][1][7:4]][SM[1][1][3:0]];
                        SM[2][1]  <= S_box[SM[2][1][7:4]][SM[2][1][3:0]];
```

```
                        SM[3][1]  <= S_box[SM[3][1][7:4]][SM[3][1][3:0]];
                        SM[0][2]  <= S_box[SM[0][2][7:4]][SM[0][2][3:0]];
                        SM[1][2]  <= S_box[SM[1][2][7:4]][SM[1][2][3:0]];
                        SM[2][2]  <= S_box[SM[2][2][7:4]][SM[2][2][3:0]];
                        SM[3][2]  <= S_box[SM[3][2][7:4]][SM[3][2][3:0]];
                        SM[0][3]  <= S_box[SM[0][3][7:4]][SM[0][3][3:0]];
                        SM[1][3]  <= S_box[SM[1][3][7:4]][SM[1][3][3:0]];
                        SM[2][3]  <= S_box[SM[2][3][7:4]][SM[2][3][3:0]];
                        SM[3][3]  <= S_box[SM[3][3][7:4]][SM[3][3][3:0]];
                    end
//__________________shift row layer ____________________________________________
                if (state == SRL)
                    begin
                        SM[1][0]  <= SM[1][1];
                        SM[2][0]  <= SM[2][2];
                        SM[3][0]  <= SM[3][3];
                        SM[1][1]  <= SM[1][2];
                        SM[2][1]  <= SM[2][3];
                        SM[3][1]  <= SM[3][0];
                        SM[1][2]  <= SM[1][3];
                        SM[2][2]  <= SM[2][0];
                        SM[3][2]  <= SM[3][1];
                        SM[1][3]  <= SM[1][0];
                        SM[2][3]  <= SM[2][1];
                        SM[3][3]  <= SM[3][2];
    end
___mix column layer ____________________________________________________________
  if (state == MCL)
    begin
        SM[0][0]  <= (M7[0] ?((SM[0][0]<<1)^8'h1b):SM[0][0]<<1) ^
        (SM[1][0] ^ (M7[1]?((SM[1][0]<<1)^8'h1b):SM[1][0]<<1)) ^ SM[2][0] ^ SM[3][0];
        SM[1][0]  <= SM[0][0] ^ (M7[1] ?((SM[1][0]<<1)^8'h1b):SM[1][0]<<1) ^
        (SM[2][0] ^ (M7[2]?((SM[2][0]<<1)^8'h1b):SM[2][0]<<1)) ^ SM[3][0];
        SM[2][0]  <= SM[0][0] ^ SM[1][0] ^ (M7[2] ?((SM[2][0]<<1)^8'h1b):SM[2][0]<<1)
        ^(SM[3][0] ^ (M7[3]?((SM[3][0]<<1)^8'h1b):SM[3][0]<<1));
        SM[3][0]  <= (SM[0][0] ^ (M7[0]?((SM[0][0]<<1)^8'h1b):SM[0][0]<<1)) ^ SM[1][0]
        ^SM[2][0] ^ (M7[3] ?((SM[3][0]<<1)^8'h1b):SM[3][0]<<1);
        SM[0][1]  <= (M7[4] ?((SM[0][1]<<1)^8'h1b):SM[0][1]<<1) ^
        (SM[1][1] ^ (M7[5]?((SM[1][1]<<1)^8'h1b):SM[1][1]<<1)) ^ SM[2][1] ^ SM[3][1];
        SM[1][1]  <= SM[0][1] ^ (M7[5] ?((SM[1][1]<<1)^8'h1b):SM[1][1]<<1) ^
        (SM[2][1] ^ (M7[6]?((SM[2][1]<<1)^8'h1b):SM[2][1]<<1)) ^ SM[3][1];
        SM[2][1]  <= SM[0][1] ^ SM[1][1] ^ (M7[6] ?((SM[2][1]<<1)^8'h1b):SM[2][1]<<1)
        ^(SM[3][1] ^ (M7[7]?((SM[3][1]<<1)^8'h1b):SM[3][1]<<1));
        SM[3][1]  <= (SM[0][1] ^ (M7[4]?((SM[0][1]<<1)^8'h1b):SM[0][1]<<1)) ^
        SM[1][1] ^ SM[2][1] ^ (M7[7] ?((SM[3][1]<<1)^8'h1b):SM[3][1]<<1);
        SM[0][2]  <= (M7[8] ?((SM[0][2]<<1)^8'h1b):SM[0][2]<<1) ^
        (SM[1][2] ^ (M7[9]?((SM[1][2]<<1)^8'h1b):SM[1][2]<<1)) ^ SM[2][2] ^ SM[3][2];
        SM[1][2]  <= SM[0][2] ^ (M7[9] ?((SM[1][2]<<1)^8'h1b):SM[1][2]<<1) ^ (SM[2][2]
        ^(M7[10]?((SM[2][2]<<1)^8'h1b):SM[2][2]<<1)) ^ SM[3][2];
        SM[2][2]  <= SM[0][2] ^ SM[1][2] ^ (M7[10] ?((SM[2][2]<<1)^8'h1b):SM[2][2]<<1)
```

```
        ^(SM[3][2] ^ (M7[11]?((SM[3][2]<<1)^8'h1b):SM[3][2]<<1));
        SM[3][2]  <= (SM[0][2] ^ (M7[8]?((SM[0][2]<<1)^8'h1b):SM[0][2]<<1)) ^
        SM[1][2] ^ SM[2][2] ^ (M7[11] ?((SM[3][2]<<1)^8'h1b):SM[3][2]<<1);
        SM[0][3]  <= (M7[12] ?((SM[0][3]<<1)^8'h1b):SM[0][3]<<1) ^ (SM[1][3] ^
        (M7[13]?((SM[1][3]<<1)^8'h1b):SM[1][3]<<1)) ^ SM[2][3] ^ SM[3][3];
        SM[1][3]  <= SM[0][3] ^ (M7[13] ?((SM[1][3]<<1)^8'h1b):SM[1][3]<<1) ^
        (SM[2][3] ^( M7[14]?((SM[2][3]<<1)^8'h1b):SM[2][3]<<1)) ^ SM[3][3];
        SM[2][3]  <= SM[0][3] ^ SM[1][3] ^ (M7[14] ?((SM[2][3]<<1)^8'h1b):SM[2][3]<<1)
        ^(SM[3][3] ^ (M7[15]?((SM[3][3]<<1)^8'h1b):SM[3][3]<<1));
        SM[3][3]  <= (SM[0][3] ^ (M7[12]?((SM[0][3]<<1)^8'h1b):SM[0][3]<<1)) ^
        SM[1][3] ^ SM[2][3] ^ (M7[15] ?((SM[3][3]<<1)^8'h1b):SM[3][3]<<1);
                        end
//_____________________message out ____________________________________________
                    if (state == MSGO) // first sending the most significant
                        begin
                            nr <= 4'h0;
                            msgout[7:0]     <= SM[0][0];
                            msgout[15:8]    <= SM[1][0];
                            msgout[23:16]   <= SM[2][0];
                            msgout[31:24]   <= SM[3][0];
                            msgout[39:32]   <= SM[0][1];
                            msgout[47:40]   <= SM[1][1];
                            msgout[55:48]   <= SM[2][1];
                            msgout[63:56]   <= SM[3][1];
                            msgout[71:64]   <= SM[0][2];
                            msgout[79:72]   <= SM[1][2];
                            msgout[87:80]   <= SM[2][2];
                            msgout[95:88]   <= SM[3][2];
                            msgout[103:96]  <= SM[0][3];
                            msgout[111:104] <= SM[1][3];
                            msgout[119:112] <= SM[2][3];
                            msgout[127:120] <= SM[3][3];
                        end
                end
        end

endmodule

```

Verilog module Top level (version 1)

```
2  //------------------------------------------------------------------------------
3  //Security module top level
4  //------------------------------------------------------------------------------
5
6  module Enc_Dec_tl (//inputs
7                         clk,
8                         reset_n,
9                         en_n,
10                        enmE_n,
11                        enmD_n,
12                        mkey,
13                        msginE,
14                        msginD,
15                      //outputs
16                        msgoutE,
17                        msgoutD,
18                        key_ready,
19                        ack_aE,
20                        ack_aD
21                    );
22
23     input clk,reset_n,en_n,enmE_n,enmD_n;
24     input [127:0]mkey,msginE,msginD;
25
29     wire [127:0]key;
30     wire key_ready;
31
32     Key_schedule K0 (clk,reset_n,en_n,mkey,key,key_ready);
33     AES128coreE AE (clk,reset_n,key_ready,enmE_n,msginD,key,msgoutE,ack_aE);
34     AES128coreD AD (clk,reset_n,key_ready,enmD_n,msginE,key,msgoutD,ack_aD);
35
36 endmodule
```

Verilog module (version 2 encryption only)

```
//------------------------------------------------------------------------------
// BlackV ultimate
//------------------------------------------------------------------------------

module BlackV (//inputs
                        clk,
                        reset_n,
                        en_n,
                        msgPinc,
                        keyin,
                        msginP,
                      //outputs
                        msgoutE,
                        key_ready,
                        ackE
                      );

    function integer clogb2;//log base 2
        input [31:0] value;
        integer      i;
        begin
            clogb2 = 0;
            for(i = 0; 2**i < value; i = i + 1)
                clogb2 = i + 1;
        end
    endfunction

    parameter WORD = 32;
    parameter HALFWORD = 16;
    parameter BYTE = 8;
    parameter NIBBLE = 4;
    parameter REGLENGHT = 10;
    parameter REGKEYLENGHT = 44;
    parameter KEYSIZE = 128;
    parameter ROUND = 11;

//KEY_SCHEDULER#####################################
    input   [KEYSIZE-1:0]keyin;
    input   clk,reset_n,en_n;
    output  key_ready;
//ENCRYPTION#######################################
    input   msgPinc;
    input   [KEYSIZE-1:0]msginP;
    output  [KEYSIZE-1:0]msgoutE;
    output  ackE;


    reg [4:0] state,nextstate;
```

```
50  //KEY_SCHEDULER###################################
51      reg key_ready;
52      reg [BYTE-1:0] S_box [HALFWORD-1:0][HALFWORD-1:0];
53      reg [BYTE-1:0] RC [REGLENGHT-1:0];
54      reg [clogb2(HALFWORD)-1:0] i,m;
55      reg [WORD-1:0] wordap1, wordapr1,rotates;
56      reg [WORD-1:0] W [REGKEYLENGHT-1:0];
57  //ENCRYPTION######################################
58      reg [KEYSIZE-1:0]msgoutE;
59      reg ackE;
60      //reg [BYTE-1:0] S_box[HALFWORD-1:0][HALFWORD-1:0];
61      reg [BYTE-1:0] SME   [clogb2(HALFWORD)-1:0][clogb2(HALFWORD)-1:0];
62      reg [BYTE-1:0] StatematrikE [ROUND-1:0][clogb2(HALFWORD)-1:0][clogb2(HALFWORD)-1:0];
63      reg [BYTE-1:0] mask8;
64      reg [HALFWORD-1:0]M7;
65      reg [clogb2(HALFWORD)-1:0] mE,nrE;
66
67
68      parameter IDLE  =  5'h00;
69      parameter M_KEY =  5'h01;
70      parameter KEY_S0 = 5'h02;
71      parameter KEY_S1 = 5'h03;
72      parameter KEY_S2 = 5'h04;
73      parameter KEY_S3 = 5'h05;
74      parameter KEY_S4 = 5'h06;
75      parameter KEY_S5 = 5'h07;
76      parameter KEY_S6 = 5'h08;
77      parameter KEY_S7 = 5'h09;
78      parameter KEY_S8 = 5'h0a;
79      parameter KEYT  =  5'h0b;
80      parameter WAIT  = 5'h0c;
81
82      parameter MSGP  = 5'h11; //acquisition message in
83      parameter KAL   = 5'h12; //key addition layer
84      parameter BSL   = 5'h13; //byte substitution layer
85      parameter SRL   = 5'h14; //shift row layer
86      parameter MCL   = 5'h15; //mix column layer
87      parameter MSGO  = 5'h16; //msg out
88
89  //____________________status register_____________________________________
90      always@(posedge clk)
91          begin
92              if(reset_n == 0)
93                  begin
94                      state <= IDLE;
95                  end
96              else
97                  begin
98                      state <= nextstate;
99                  end
```

```
100         end
101  //____________________state transition_____________________________________
102     always@(*)
103         begin
104             if (en_n == 0)begin
105             case(state)
106                 IDLE:nextstate = M_KEY;
107                 M_KEY:  nextstate = KEY_S0;
108                 KEY_S0: nextstate = KEY_S1;
109                 KEY_S1: nextstate = KEY_S2;
110                 KEY_S2: nextstate = KEY_S3;
111                 KEY_S3: nextstate = KEY_S4;
112                 KEY_S4: nextstate = KEY_S5;
113                 KEY_S5: nextstate = KEY_S6;
114                 KEY_S6: nextstate = KEY_S7;
115                 KEY_S7: begin
116                             if (i == ROUND -1 ) nextstate = KEY_S8;
117                             else nextstate = KEY_S0;
118                         end
119 
120                 KEY_S8: nextstate = KEYT;
121                 KEYT  : begin
122                             if(m == ROUND) nextstate = WAIT;
123                             else nextstate = KEYT;
124                         end
125                 WAIT:   begin
126                             if(msgPinc == 1) nextstate = MSGP;
127                             else nextstate = WAIT;
128                         end
129                 //acquisition message in
130                 MSGP:   if ( msgPinc == 1)nextstate = KAL;
131                         else nextstate = MSGP;
132                 //key addition layer
133                 KAL   : begin
134                             if ( nrE == ROUND-1)nextstate = MSGO;
135                             else nextstate = BSL;
136                         end
137                 //byte substitution layer
138                 BSL   : nextstate = SRL;
139                 //shift row layer
140                 SRL   : begin
141                             if (nrE < ROUND-1) nextstate = MCL;
142                             else if (nrE == ROUND-1) nextstate = KAL;
143                             else nextstate = SRL;
144                         end
145                 //mix column layer
146                 MCL   : nextstate = KAL;
147 
148                 //msg out
149                 MSGO  : if (en_n == 1 && msgPinc == 0) nextstate = MSGP;
```

```
                        else nextstate = MSGO;

                default: nextstate = IDLE;

            endcase
            end
            else nextstate = nextstate;
        end

initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/sbox.txt",S_box);
initial $readmemh ("/Users/giorgio/Desktop/Projects/RENESAS/memory_txt/RC.txt",RC);


//_____________________core_____________________________________________________
    always@(*)
        begin
            key_ready = 0;
            ackE = 0;
            M7 = 16'h0;
            case(state)

                IDLE:;
                M_KEY:;
//_____________________keys generation (start)_____________________________________
                KEY_S0:;
                KEY_S1:;
                KEY_S2:;
                KEY_S3:;
                KEY_S4:;
                KEY_S5:;
                KEY_S6:;
                KEY_S7:;
                KEY_S8:;
//_____________________keys generation (end)_____________________________________
//_____________________keys sending_____________________________________
                KEYT:;
                WAIT:key_ready = 1;



//_____________________ENCRYPTION_____________________________________
                MSGP:;
                KAL:;
                BSL:;
                SRL:;
                MCL:begin
                        M7[0] = |((SME[0][0] & mask8)>>7);
                        M7[1] = |((SME[1][0] & mask8)>>7);
                        M7[2] = |((SME[2][0] & mask8)>>7);
                        M7[3] = |((SME[3][0] & mask8)>>7);
```

```
                        M7[4] = |((SME[0][1] & mask8)>>7);
                        M7[5] = |((SME[1][1] & mask8)>>7);
                        M7[6] = |((SME[2][1] & mask8)>>7);
                        M7[7] = |((SME[3][1] & mask8)>>7);
                        M7[8] = |((SME[0][2] & mask8)>>7);
                        M7[9] = |((SME[1][2] & mask8)>>7);
                        M7[10] = |((SME[2][2] & mask8)>>7);
                        M7[11] = |((SME[3][2] & mask8)>>7);
                        M7[12] = |((SME[0][3] & mask8)>>7);
                        M7[13] = |((SME[1][3] & mask8)>>7);
                        M7[14] = |((SME[2][3] & mask8)>>7);
                        M7[15] = |((SME[3][3] & mask8)>>7);
                    end
                MSGO:begin
                        ackE = 1;
                        M7 = 16'h0;
                    end
                default:begin
                        ackE = 0;
                        M7 = 16'h0;
                        key_ready = 0;
                    end
            endcase
        end

    always@(posedge clk)
        begin
//_____________________IDLE state_____________________________________________
            if(state == IDLE)
                begin
                    i  <= 4'h1;
                    m  <= 4'h0;
                    mE <= 4'h0;
                    nrE     <= 4'h0;
                    mask8   <= KEYSIZE;
                end
//_____________________acqusition master key_____________________________________
            if (state == M_KEY)
                begin
                    W[0] <= keyin[(4*WORD)-1:3*WORD];
                    W[1] <= keyin[(3*WORD)-1:2*WORD];
                    W[2] <= keyin[(2*WORD)-1:WORD];
                    W[3] <= keyin[WORD-1:0];
                end
//_____________________keys generation (start)_____________________________________
            if(state == KEY_S0) rotates <= W[(4*i)-1];
            if(state == KEY_S1)
                begin
                    wordapr1[BYTE-1:0]  <= rotates[(BYTE*4)-1:BYTE*3];
```

```
                    wordapr1[(BYTE*2)-1:BYTE]   <= rotates[BYTE-1:0];
                    wordapr1[(BYTE*3)-1:BYTE*2] <= rotates[(BYTE*2)-1:BYTE];
                    wordapr1[(BYTE*4)-1:BYTE*3] <= rotates[(BYTE*3)-1:BYTE*2];
                end
            if(state == KEY_S2) rotates <= wordapr1;
            if(state == KEY_S3)
                begin
                wordap1[(BYTE*4)-1:BYTE*3]  <= S_box[rotates[(NIBBLE*8)-1:NIBBLE*7]]
                [rotates[(NIBBLE*7)-1:NIBBLE*6]]^ RC[i-1];
                wordap1[(BYTE*3)-1:BYTE*2]  <= S_box[rotates[(NIBBLE*6)-1:NIBBLE*5]]
                [rotates[(NIBBLE*5)-1:NIBBLE*4]];
                wordap1[(BYTE*2)-1:BYTE]    <= S_box[rotates[(NIBBLE*4)-1:NIBBLE*3]]
                [rotates[(NIBBLE*3)-1:NIBBLE*2]];
                wordap1[BYTE-1:0]           <= S_box[rotates[(NIBBLE*2)-1:NIBBLE]]
                [rotates[NIBBLE-1:0]];
                end
            if(state == KEY_S4) W[4*i] <= W[4*(i-1)] ^ wordap1;
            if(state == KEY_S5) W[4*i + 1] <=  W[4*i + 0] ^ W[4*(i-1) + 1];
            if(state == KEY_S6) W[4*i + 2] <=  W[4*i + 1] ^ W[4*(i-1) + 2];
            if(state == KEY_S7)
                begin
                    if (i == ROUND -1) begin
                        wordapr1 <= 32'h0;
                        wordap1  <= 32'h0;
                        rotates  <= 32'h0;
                    end
                    W[4*i + 3] <=  W[4*i + 2] ^ W[4*(i-1) + 3];
                    i <= i + 1;

                end
//____________________keys generation (end)_____________________________________
//____________________keys to matrik_____________________________________
            if (state == KEYT && en_n == 0)
                begin
                    if(m < ROUND)
                        begin
                            StatematrikE [m][3][3]  <= W[(m+1)*4 - 1][7:0];
                            StatematrikE [m][2][3]  <= W[(m+1)*4 - 1][15:8];
                            StatematrikE [m][1][3]  <= W[(m+1)*4 - 1][23:16];
                            StatematrikE [m][0][3]  <= W[(m+1)*4 - 1][31:24];
                            StatematrikE [m][3][2]  <= W[(m+1)*4 - 2][7:0];
                            StatematrikE [m][2][2]  <= W[(m+1)*4 - 2][15:8];
                            StatematrikE [m][1][2]  <= W[(m+1)*4 - 2][23:16];
                            StatematrikE [m][0][2]  <= W[(m+1)*4 - 2][31:24];
                            StatematrikE [m][3][1]  <= W[(m+1)*4 - 3][7:0];
                            StatematrikE [m][2][1]  <= W[(m+1)*4 - 3][15:8];
                            StatematrikE [m][1][1]  <= W[(m+1)*4 - 3][23:16];
                            StatematrikE [m][0][1]  <= W[(m+1)*4 - 3][31:24];
                            StatematrikE [m][3][0]  <= W[m*4][7:0];
                            StatematrikE [m][2][0]  <= W[m*4][15:8];
```

```
                        StatematrikE [m][1][0]  <= W[m*4][23:16];
                        StatematrikE [m][0][0]  <= W[m*4][31:24];
                    end
                if( m < ROUND) m <= m + 4'h1;
            end


//##############################################ENCRYPTION#################################

        if(en_n==0)
            begin

//____________________acquisition MSGP_____________________________________________________
                if (state == MSGP && msgPinc == 1)
                    begin
                        mE <= 4'h0;
                        SME [3][3]  <= msginP[7:0];
                        SME [2][3]  <= msginP[15:8];
                        SME [1][3]  <= msginP[23:16];
                        SME [0][3]  <= msginP[31:24];
                        SME [3][2]  <= msginP[39:32];
                        SME [2][2]  <= msginP[47:40];
                        SME [1][2]  <= msginP[55:48];
                        SME [0][2]  <= msginP[63:56];
                        SME [3][1]  <= msginP[71:64];
                        SME [2][1]  <= msginP[79:72];
                        SME [1][1]  <= msginP[87:80];
                        SME [0][1]  <= msginP[95:88];
                        SME [3][0]  <= msginP[103:96];
                        SME [2][0]  <= msginP[111:104];
                        SME [1][0]  <= msginP[119:112];
                        SME [0][0]  <= msginP[127:120];
                    end
//____________________key addition layer _____________________________________________________
                if (state == KAL)
                    begin
                        if (nrE < ROUND-1) nrE <= nrE + 4'h1;
                        SME[0][0] <= SME[0][0] ^ StatematrikE[nrE][0][0];
                        SME[1][0] <= SME[1][0] ^ StatematrikE[nrE][1][0];
                        SME[2][0] <= SME[2][0] ^ StatematrikE[nrE][2][0];
                        SME[3][0] <= SME[3][0] ^ StatematrikE[nrE][3][0];
                        SME[0][1] <= SME[0][1] ^ StatematrikE[nrE][0][1];
                        SME[1][1] <= SME[1][1] ^ StatematrikE[nrE][1][1];
                        SME[2][1] <= SME[2][1] ^ StatematrikE[nrE][2][1];
                        SME[3][1] <= SME[3][1] ^ StatematrikE[nrE][3][1];
                        SME[0][2] <= SME[0][2] ^ StatematrikE[nrE][0][2];
                        SME[1][2] <= SME[1][2] ^ StatematrikE[nrE][1][2];
                        SME[2][2] <= SME[2][2] ^ StatematrikE[nrE][2][2];
                        SME[3][2] <= SME[3][2] ^ StatematrikE[nrE][3][2];
                        SME[0][3] <= SME[0][3] ^ StatematrikE[nrE][0][3];
```

```
                        SME[1][3] <= SME[1][3] ^ StatematrikE[nrE][1][3];
                        SME[2][3] <= SME[2][3] ^ StatematrikE[nrE][2][3];
                        SME[3][3] <= SME[3][3] ^ StatematrikE[nrE][3][3];
                    end
//__________________byte substitution layer ____________________________________
                if (state == BSL)
                    begin
                        SME[0][0]  <= S_box[SME[0][0][7:4]][SME[0][0][3:0]];
                        SME[1][0]  <= S_box[SME[1][0][7:4]][SME[1][0][3:0]];
                        SME[2][0]  <= S_box[SME[2][0][7:4]][SME[2][0][3:0]];
                        SME[3][0]  <= S_box[SME[3][0][7:4]][SME[3][0][3:0]];
                        SME[0][1]  <= S_box[SME[0][1][7:4]][SME[0][1][3:0]];
                        SME[1][1]  <= S_box[SME[1][1][7:4]][SME[1][1][3:0]];
                        SME[2][1]  <= S_box[SME[2][1][7:4]][SME[2][1][3:0]];
                        SME[3][1]  <= S_box[SME[3][1][7:4]][SME[3][1][3:0]];
                        SME[0][2]  <= S_box[SME[0][2][7:4]][SME[0][2][3:0]];
                        SME[1][2]  <= S_box[SME[1][2][7:4]][SME[1][2][3:0]];
                        SME[2][2]  <= S_box[SME[2][2][7:4]][SME[2][2][3:0]];
                        SME[3][2]  <= S_box[SME[3][2][7:4]][SME[3][2][3:0]];
                        SME[0][3]  <= S_box[SME[0][3][7:4]][SME[0][3][3:0]];
                        SME[1][3]  <= S_box[SME[1][3][7:4]][SME[1][3][3:0]];
                        SME[2][3]  <= S_box[SME[2][3][7:4]][SME[2][3][3:0]];
                        SME[3][3]  <= S_box[SME[3][3][7:4]][SME[3][3][3:0]];
                    end
//__________________shift row layer ____________________________________________
                if (state == SRL)
                    begin
                        SME[1][0]  <= SME[1][1];
                        SME[2][0]  <= SME[2][2];
                        SME[3][0]  <= SME[3][3];
                        SME[1][1]  <= SME[1][2];
                        SME[2][1]  <= SME[2][3];
                        SME[3][1]  <= SME[3][0];
                        SME[1][2]  <= SME[1][3];
                        SME[2][2]  <= SME[2][0];
                        SME[3][2]  <= SME[3][1];
                        SME[1][3]  <= SME[1][0];
                        SME[2][3]  <= SME[2][1];
                        SME[3][3]  <= SME[3][2];
                    end
//__________________mix column layer ____________________________________________
if (state == MCL)
    begin
    SME[0][0]  <= (M7[0] ?((SME[0][0]<<1)^8'h1b):SME[0][0]<<1) ^
    (SME[1][0] ^ (M7[1]?((SME[1][0]<<1)^8'h1b):SME[1][0]<<1)) ^ SME[2][0] ^ SME[3][0];
    SME[1][0]  <= SME[0][0] ^ (M7[1] ?((SME[1][0]<<1)^8'h1b):SME[1][0]<<1) ^
    (SME[2][0] ^ (M7[2]?((SME[2][0]<<1)^8'h1b):SME[2][0]<<1)) ^ SME[3][0];
    SME[2][0]  <= SME[0][0] ^ SME[1][0] ^ (M7[2] ?((SME[2][0]<<1)^8'h1b):SME[2][0]<<1)
    ^(SME[3][0] ^ (M7[3]?((SME[3][0]<<1)^8'h1b):SME[3][0]<<1));
    SME[3][0]  <= (SME[0][0] ^ (M7[0]?((SME[0][0]<<1)^8'h1b):SME[0][0]<<1)) ^ SME[1][0]
    ^SME[2][0] ^ (M7[3] ?((SME[3][0]<<1)^8'h1b):SME[3][0]<<1);
```

```
400     SME[0][1]  <= (M7[4] ?((SME[0][1]<<1)^8'h1b):SME[0][1]<<1) ^ (SME[1][1]
401     ^ (M7[5]?((SME[1][1]<<1)^8'h1b):SME[1][1]<<1)) ^ SME[2][1] ^ SME[3][1];
402     SME[1][1]  <= SME[0][1] ^ (M7[5] ?((SME[1][1]<<1)^8'h1b):SME[1][1]<<1) ^ (SME[2][1]
403     ^(M7[6]?((SME[2][1]<<1)^8'h1b):SME[2][1]<<1)) ^ SME[3][1];
404     SME[2][1]  <= SME[0][1] ^ SME[1][1] ^ (M7[6] ?((SME[2][1]<<1)^8'h1b):SME[2][1]<<1)
405     ^(SME[3][1] ^ (M7[7]?((SME[3][1]<<1)^8'h1b):SME[3][1]<<1));
406     SME[3][1]  <= (SME[0][1] ^ (M7[4]?((SME[0][1]<<1)^8'h1b):SME[0][1]<<1)) ^ SME[1][1]
407     ^SME[2][1] ^ (M7[7] ?((SME[3][1]<<1)^8'h1b):SME[3][1]<<1);
408     SME[0][2]  <= (M7[8] ?((SME[0][2]<<1)^8'h1b):SME[0][2]<<1) ^ (SME[1][2] ^
409     (M7[9]?((SME[1][2]<<1)^8'h1b):SME[1][2]<<1)) ^ SME[2][2] ^ SME[3][2];
410     SME[1][2]  <= SME[0][2] ^ (M7[9] ?((SME[1][2]<<1)^8'h1b):SME[1][2]<<1) ^ (SME[2][2]
411     ^(M7[10]?((SME[2][2]<<1)^8'h1b):SME[2][2]<<1)) ^ SME[3][2];
412     SME[2][2]  <= SME[0][2] ^ SME[1][2] ^ (M7[10] ?((SME[2][2]<<1)^8'h1b):SME[2][2]<<1)
413     ^(SME[3][2] ^ (M7[11]?((SME[3][2]<<1)^8'h1b):SME[3][2]<<1));
414     SME[3][2]  <= (SME[0][2] ^ (M7[8]?((SME[0][2]<<1)^8'h1b):SME[0][2]<<1)) ^ SME[1][2]
415     ^SME[2][2] ^ (M7[11] ?((SME[3][2]<<1)^8'h1b):SME[3][2]<<1);
416     SME[0][3]  <= (M7[12] ?((SME[0][3]<<1)^8'h1b):SME[0][3]<<1) ^ (SME[1][3] ^
417     (M7[13]?((SME[1][3]<<1)^8'h1b):SME[1][3]<<1)) ^ SME[2][3] ^ SME[3][3];
418     SME[1][3]  <= SME[0][3] ^ (M7[13] ?((SME[1][3]<<1)^8'h1b):SME[1][3]<<1) ^ (SME[2][3]
419     ^(M7[14]?((SME[2][3]<<1)^8'h1b):SME[2][3]<<1)) ^ SME[3][3];
420     SME[2][3]  <= SME[0][3] ^ SME[1][3] ^ (M7[14] ?((SME[2][3]<<1)^8'h1b):SME[2][3]<<1)
421     ^(SME[3][3] ^ (M7[15]?((SME[3][3]<<1)^8'h1b):SME[3][3]<<1));
422     SME[3][3]  <= (SME[0][3] ^ (M7[12]?((SME[0][3]<<1)^8'h1b):SME[0][3]<<1)) ^ SME[1][3]
423     ^SME[2][3] ^ (M7[15] ?((SME[3][3]<<1)^8'h1b):SME[3][3]<<1);
424     end
425 //_____________________message out ____________________________________________
426                     if (state == MSGO) // first sending the most significant
427                         begin
428                             nrE <= 4'h0;
429                             msgoutE[7:0]    <= SME[0][0];
430                             msgoutE[15:8]   <= SME[1][0];
431                             msgoutE[23:16]  <= SME[2][0];
432                             msgoutE[31:24]  <= SME[3][0];
433                             msgoutE[39:32]  <= SME[0][1];
434                             msgoutE[47:40]  <= SME[1][1];
435                             msgoutE[55:48]  <= SME[2][1];
436                             msgoutE[63:56]  <= SME[3][1];
437                             msgoutE[71:64]  <= SME[0][2];
438                             msgoutE[79:72]  <= SME[1][2];
439                             msgoutE[87:80]  <= SME[2][2];
440                             msgoutE[95:88]  <= SME[3][2];
441                             msgoutE[103:96] <= SME[0][3];
442                             msgoutE[111:104] <= SME[1][3];
443                             msgoutE[119:112] <= SME[2][3];
444                             msgoutE[127:120] <= SME[3][3];
445                         end
446                 end
447         end
448
449 endmodule
```

Verilog simple direct test bench

```
//--------------------------------------------------------------------------------
//test bench security module
//--------------------------------------------------------------------------------

module Enc_Dec_tb ();

    parameter KEYSIZE = 128;

    reg clk,reset_n,en_n,enmE_n,enmD_n;
    reg [KEYSIZE-1:0]mkey;
    reg [KEYSIZE-1:0]msginE;
    reg [KEYSIZE-1:0]msginD;

    wire ack_k,ack_a,ack_aD;
    wire [KEYSIZE-1:0]msgoutE;
    wire [KEYSIZE-1:0]msgoutD;

Enc_Dec_tl E0 (clk,reset_n,en_n,enmE_n,enmD_n,mkey,msginE,
                msginD,msgoutE,msgoutD,ack_k,ack_a,ack_aD);

//_________Generazione clock_____________________________________________________
    always
        begin
            #3 clk = !clk;
        end

initial
        begin
//_____condizioni iniziali_____________________________________________________
            clk = 0;
            reset_n = 0;
            en_n = 1;
            enmE_n = 1;
            enmD_n = 1;
            mkey = 128'h0;
            msginD =128'h0;
//_____generazione chiavi______________________________________________________
            #5 reset_n = 1;
            #5 en_n = 0;
            mkey = 128'h2b7e151628aed2a6abf7158809cf4f3c;

//######disturbo (rimosso)_____________________________________________________
            //#560 en_n = 1;
            #560 en_n = 0;
            #10 en_n = 0;

//_____Primo test cifratura e decifratura stesso messaggio in serie____________
            #70 msginD = 128'h7072696d6f2074657374202020202020;
            enmE_n = 0;
            #10 enmE_n = 1;

            #260 msginE = 128'h663b34250d62402d9c090293ceb02dab;
            enmD_n = 0;
            #10 enmD_n = 1;
```

```
56
57  //_____Secondo test ciratura e decifratura di due messaggi in parallelo_____________________
58
59              #500 msginD = 128'h7365636f6e646f202074657374202020;
60              enmE_n = 0;
61              msginE = 128'ha04dbef22907946f5c40f6d285e1b4c6;
62              enmD_n = 0;
63
64              #20 enmE_n = 1;
65              enmD_n = 1;
66  //######disturbo (rimosso)_____________________________________________________________________
67              //#60 en_n = 1;
68              #60 en_n = 0;
69              #10 en_n = 0;
70
71
72          end
73
74  endmodule
```

Bibliography

1. Christof Paar and Jan Pelzl, "***Understanding Cryptography***".
2. Kerstin Lemke, Christof Paar and MarkoWolf (Eds.), "***Embedded Security in Cars***".
3. Agnes Hui Chan Virgil Gligor (Eds.), "***Information Security***".
4. Nicolas Navet and Françoise Simonot-Lion, "***Automotive Embedded Systems Handbook***"
5. Aamer Nadeem, Dr M. Younus Javed, "***A Performance Comparison of Data Encryption Algorithms***".
6. Federal Information Processing Standards Publication 197 (November 26, 2001) "***ADVANCED ENCRYPTION STANDARD (AES)***".
7. Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage,"***Comprehensive Experimental Analyses of Automotive Attack Surfaces***".
8. Karl Koscher, Alexei Czeskis, Franziska Roesner, Shwetak Patel, Tadayoshi Kohno, Stephen Checkoway, Damon McCoy, Brian Kantor, Danny Anderson, Hovav Shacham, and Stefan Savage, "***Experimental Security Analysis of a Modern Automobile***".
9. Pierre Kleberger, Tomas Olovsson, and Erland Jonsson, "***Security Aspects of the In-Vehicle Network in the Connected Car***".
10. Dennis K. Nilsson, Phu H. Phung, and Ulf E. Larson, "***Vehicle ECU Classification based on Safety-Security characteristics***".
11. Miguel León Chávez, Carlos Hernández Rosete, and Francisco Rodríguez Henríquez, "***Achieving Confidentiality Security Service for CAN***".
12. Hendrik Schweppe, Yves Roudier (EURECOM), "***Security and Privacy for In-Vehicle Networks***".
13. Ivan Studnia, Vincent Nicomette, Eric Alata, Yves Deswarte, Mohamed Kaâniche, Youssef Laarouchi, Renault S.A.S, CNRS, LAAS, Univ. Toulouse, "***Security of embedded automotive networks: state of the art and a research proposal***".
14. BOSCH, "**CAN Specification Version 2.0**"
15. Audi, BMW, Daimler, Porsche, Volkswagen (HIS memebers), "***SHE Version 1.8.2***"
16. EVITA, "***Deliverables 2.1, 2.3, 2.5.1, 2.5.2, 2.6, 3.1, 3.2***".
17. Chris Spear, "***SystemVerilog for Verification: A Guide to Learning the Testbench Language Features***".