Automotive Cyber Security Compendium

Infineon Microcontroller AURIX™

www.infineon.com
Disclaimer

This document provides an overview of cyber security best practices when using the AURIX™ family of automotive microcontrollers and their Hardware Security Module (HSM). The information presented hereafter does not guarantee a secure system, and does not constitute a warranty, express or implied. The authors recognize that perfect security does not exist and disclaim all liability for any damage or harm resulting from the use or misuse of the information contained in this document.

Abstract

How secure is an automotive system? How can automotive manufacturers and suppliers increase the level of protection using Infineon’s AURIX™ microcontrollers? A deep dive into the hardware and software implementation of the AURIX™ is needed to answer these questions.

The aim of this document is to list and describe the main cyber security mechanisms implemented in AURIX™ microcontrollers, and how to use them to increase security in Automotive Electronic Control Units (ECUs).

In addition, this document provides an overview of system-level cyber security measures that can further enhance vehicle security.
3. Overall system security 15
   3.1 System related measures 15
   3.2 Multiple layers of security 15
   3.3 Secured communication 15
      3.3.1 Key management system 16
      3.3.2 Intrusion detection and prevention system 16
      3.3.3 Reduced window of opportunity 16
   3.4 Software related measures 17
      3.4.1 Logical attacks 17
         3.4.1.1 Protocol based weaknesses 17
         3.4.1.2 Programming weaknesses 17
         3.4.1.3 Database 17
         3.4.1.4 Fault containment 18
      3.4.2 Timing attacks 19
      3.4.3 Power analysis attacks and electromagnetic analysis attacks 20
         3.4.3.1 Simple power analysis 20
         3.4.3.2 Differential and correlation power analysis 20
         3.4.3.3 Higher order attack and multivariate attacks 20
      3.4.3.4 Countermeasures 21
      3.4.4 Fault injection 22
         3.4.4.1 Example of attacks 22
         3.4.4.2 Countermeasures in software 22

Glossary 23

References 25
1. Introduction

The car ecosystem is growing fast and new connectivity technologies will move the car into the Internet of Things (IoT) domain, with all its benefits and threats.

Connectivity brings more services, new features and innovative revenue streams to road vehicles. However, as the number of Electronic Control Units (ECUs) per vehicle increases, and cars get connected to the internet, to users and to each other, their complexity and attack surface grow exponentially.

New and upcoming features such as remote diagnostic and software updates over the air, emergency call, internet services, in-car payment, mobile apps, as well as infotainment and traffic information all increase the attack surface of road vehicles.

The upcoming ISO 21434 cyber security standard, which is still under discussion, is expected to address vehicle cybersecurity management from different perspectives, including its relationship with safety, and will tackle for example how both can coexist without interferences or inconsistencies.

One of the most important security objectives is the protection of occupant safety and therefore all automotive safety-related components, communications, functions and interfaces. This includes protection against any attack on dedicated passive safety (e.g. belts, airbags, [near] accident detection, etc.) and active safety mechanisms (e.g. anti-lock braking system (ABS), electronic stability control (ESC), lane keep assistant, etc.), as well as all driving-related components and functions (e.g. vehicle steering, braking or gear shifting).

For each of these threats, system engineers and security architects must know how to best leverage ECU security features, and decide on the appropriate balance between security level and cost of implementation.

The present document is meant to help system designers implement the right level of security at all levels, using the security features included in Infineon’s AURIX™ family of microcontrollers.
2. AURIX™ Features

2.1 Hardware security

The hardware security module (HSM) inside AURIX™ provides a robust answer to cybersecurity requirements, access control, as well as data confidentiality, integrity and authenticity needs.

The HSM is designed to perform cryptographic operations using its internal and reserved resources such as a true random number generator (TRNG), AES, SHA and PKC hardware engines. In addition to secured boot and protecting access to the AURIX™ debug interface, the HSM’s main features are to provide a secured storage for cryptographic keys, to generate random numbers and to encrypt and decrypt data using symmetric and asymmetric cryptographic algorithms.

The CAN protocol, for example, was not designed with security in mind, but it can be hardened by using the AURIX™ HSM. Indeed, CAN messages are broadcast to the entire bus, and confidentiality, authenticity and integrity are not built into the protocol. Using the HSM to encrypt messages and generate message authentication codes can help to meet the most common security requirements of in-vehicle networks.

All cryptographic algorithms implemented in hardware reduce the latency of cryptographic operations and provide enough bandwidth to support CAN communications without excessive delays or resource overloading in the main CPU.

HSM domain structure

The AURIX™ HSM is an embedded microcontroller, a secured area logically separated from the rest of the AURIX™ chip, and has a dedicated internal core. It is composed of (see Figure 1):

› a 32 bit MCU (ARM™ core)
› a dedicated internal SRAM
› a boot ROM for the HSM firmware
› an AES 128 bit hardware accelerator
› two 16 bit timer modules
› a bridge module
› a true random number generator (TRNG)

Besides these modules which are included in the first AURIX™ generation, the second generation also includes a PKC module for asymmetric cryptography using elliptic curves, a SHA module and a secure watchdog.

The HSM is located on the system peripheral bus (SPB), which provides also its clock frequency.
2.1.1 CPU

HSM core is an ARM™ Cortex M3 32 bit core with a maximum clock frequency of 100 MHz. Having a dedicated core for the security operations, fully detached from the remaining part, provides a huge advantage because it allows to create a trusted execution environment with a separated domain.

2.1.2 Cache memory

The cache memory is a high-speed buffer located between the CPU and the (external) main memory. It holds a copy of a part of the memory content. Accessing the copy in the cache memory is considerably faster than retrieving the information from the main memory. In addition to its fast access speed, the cache also consumes less power than the main memory.

The speed and efficiency of cache systems is due to their physical proximity to the CPU.

Programs often utilize a specific section of the address space for their processing over a period of time. By including most or all of this code in the cache, system performance can be dramatically enhanced.

For more details please refer to the Infineon Application Tutorial AP32456.
2.1.3 True random number generator

The HSM’s true random number generator (TRNG) provides random data that can be used to generate cryptographic keys and in protocols (challenges, blinding values, padding bytes, etc.). It is compliant with the AIS 20/31 standard by German BSI [17].

The HSM TRNG module continuously generates true random bytes for the system operation once it is enabled.

2.1.4 General purpose timers

The HSM includes two 16-bit general purpose timers which can be started and stopped individually. They only count upwards and have their own pre-scaler and reload value.

The timer module offers 4 selectable clock sources per timer and interrupt capability.

2.1.5 Advanced encryption standard module

The advanced encryption standard (AES), also known by its original name Rijndael, is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001 [18].

The HSM includes an AES 128 hardware accelerator that supports the following modes: ECB, CBC, CTR, OFB and CFB as well GCM and XTS.

The AES algorithm performs data encryption and decryption in blocks of 128 bits of data.

For further information and details please refer to the Infineon AP32470, which describes hints how to optimize performances.

2.1.6 Bridge module

The main purpose of the bridge module is to connect the HSM subsystem to the host system and to enable communication between the two systems. All interaction between HSM and host runs through the bridge module.

The bridge module acts as a sort of “firewall”, protecting the HSM internal resources from access by other masters. This is fully applied only if the debug access to HSM has been restricted, as described in the AP32389 and AP32399.

The host system has only restricted access to the HSM’s resources. While the AURIX™ is not in HSM debug mode or memory testing, access to all internal HSM memories and peripherals is protected.

HSM core can access to the complete system memory map of the entire device via the Bridge module, which is a bus master on the SPB.

The bridge module provides a set of special function registers (SFRs) for exchanging data between host and HSM side. The SFRs are used to trigger interrupt in the HSM CPU and vice versa from HSM to host core.
2.1.7 Read-only memory

The boot read-only memory (ROM) of 4 KB contains code and read-only data that is used during the startup of the HSM.

A boot system (BOS) is provided for HSM. All mandatory BOS functions for internal testing, production usage and startup behavior are stored in the boot ROM of the HSM. No patch functionality is available for this code. The BOS is the only HSM software component provided by Infineon.

2.1.8 Random access memory

The internal and reserved amount of random access memory (RAM) capacity is variable from first to second AURIX™ generation.

In the first generation and specifically in the TC23xx devices the HSM RAM available is 24 KB, while in the TC27xx and TC29xx devices it is 40 KB in size.

In the second generation, for all the family of products the HSM contains 96 KB of RAM.

2.1.9 Public key cryptography module

This module is only available in AURIX™ 2nd generation TC3xx.

The public key cryptography (PKC) module is a hardware accelerator which supports fast signature generation and verification with ECDSA.

In particular, the PKC module enables modular and non-modular operations on integers and binary polynomials up to 256 bit length, like:

› multiplication
› modular addition and subtraction
› modular multiplication
› modular inversion and division

The PKC module also enables complex algorithms on all common elliptic curves of a length up to 256 bit:

› addition of two points in affine coordinates
› doubling of a point in affine coordinates
› scalar multiplication

The supported curves are all curves defined over finite fields of the types Fp and GF(2d) = F2[X]/f of bit length up to 256 bit.
2.1.10 HASH module SHA256

The HASH module is only available in AURIX™ 2nd generation TC3xx.

The HASH module is intended to be used for signature generation, verification and generic data integrity checks.

The HASH module can perform several operations (see Table 1):

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>HASH (output) size</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD-5</td>
<td>128 bits</td>
<td>Not recommended for use anymore</td>
</tr>
<tr>
<td>SHA-1</td>
<td>160 bits</td>
<td>Not recommended for use except in selected applications</td>
</tr>
<tr>
<td>SHA-2</td>
<td>224 bits</td>
<td>Standardized in 2001 by NIST. The AURIX™ HSM can perform SHA-2 up to its 256-bit variant in a hardware accelerator, while 384-bit and 512-bit variant can be performed only in software.</td>
</tr>
</tbody>
</table>

|     | 256 bits          |         |
|     | 384 bits          |         |
|     | 512 bits          |         |

Table 1 – HASH module operations

2.1.11 Secure watchdog

The secure watchdog is only available in AURIX™ 2nd generation TC3xx.

The secure watchdog is a 16-bit upcounting watchdog timer which allows to monitor system operation for possible timeouts as well as to check the correct order of operations acting as a checkpoint mismatch event generator.

2.1.12 Memory protection unit

The purpose of the memory protection unit (MPU) is to restrict the access and type of access of the CPU to configured memory areas.

MPUs are available for the application on every AURIX™ 1st (TC2xx) and 2nd generation (TC3xx) microcontroller TriCore™ and for security software on HSM CPU.

With the core MPU, the HSM implements an almost fully ARMv7-M compatible protected memory system architecture (PMSAv7).

In relation with security, the MPU supports spatial isolation of software partitions, tasks or functions and it is able to avoid faults and allows containment of possible software vulnerabilities.

The general rule to apply is to give a memory area the minimal access rights required to execute the application. For example a memory area which contains data should not have “execute” access right. Constant data should only have “read” access, while code area should not have “write” access right as well. For a code area located in volatile memory, the MPU can restrict the access once the memory area is initialized with the code.
2.2 Debugger protection mechanism

A first level of protection against physical attacks is the debugger protection mechanism. A good practice should be always to lock the debugger access via a 256 bit password in order to restrict the access to the internal resources of the device.

AURIX™ 1st and 2nd generation devices provide multiple ways to prevent the unauthorized access to the device.

For more details please refer to the Infineon AP32399 for AURIX™ 2nd generation (TC3xx) and AP32389 for AURIX™ 1st generation (TC2xx).

2.2.1 Destructive debug entry

Some OEM’s requirements strongly restrict the debug access in order to protect SW-IP and for tuning protection reasons. This requirements most of the time are in contradiction with the possibility to analyze hardware issues during field analysis returns. The full support of the debugging process can in this case be heavily limited and customers need to accept this restriction.

AURIX™ TC3xx offers, in addition to the 256 bit password protection, another solution called destructive debug entry.

With this feature, the possibility to open the debugger access set a permanent destructive action on the Flexray and CAN module, which become inactive and make the ECU completely useless in car environments.

2.3 Secured boot

Security is an essential component of any well-functioning system to ensure that each sub-component does what it is intended to do. In the automotive field, security may take multiple forms and affects the entire electrical/electronic (EE) architecture included in electronic control units (ECUs). One solution is to establish a “trusted” environment in an ECU microcontroller, by leveraging the isolation and security mechanisms in hardware and firmware. To do this, however, the code executed must be authentic, and has to be checked before execution; this is the goal and principle of secured boot.

Furthermore, the secured boot process can be leveraged to securely update ECU software or firmware in the field (over-the-air software updates). A device’s secured boot process checks the authenticity of the user software upon every boot (reset) cycle.

Early security innovations on microcontrollers focused on protection of the software and JTAG ports. These features began to materialize in the early to mid-2000s. Indeed, at the time, ECU production and assembly were increasingly moved or outsourced overseas to reduce costs. As such, it became necessary to protect the intellectual property contained in the software owned by OEM’s and Tier1’s.

Infineon has added security features to microcontrollers for well over a decade; one early specification was the implementation of the secure hardware extension (SHE) standard [10]. This standard includes AES128 encryption and key lock-out protections with a capacity for up to 10 keys. The SHE standard was released by HIS (a group of German OEMs and Tier 1 suppliers) in April 2009. Its functionality is most often implemented through a state machine (application-specific integrated circuit, ASIC) with dedicated security tasks. This includes a SHE secure boot that hashes (via an AES-128 CMAC) a fixed portion of code before handing operational control back to the primary application thread.

To activate the secured boot feature, a key has to be written into a dedicated slot. At the next reset cycle of the CPU, the SHE secure boot sequence will be enabled. There is some debate on the location of the memory for the boot sequence, since the standard requires that it belongs on the application side. This is a perfectly suitable solution assuming that secure boot ensures the application code is unaltered.
2.4 Flash protection mechanisms

Microcontroller flash memory is probably the most interesting point for an external attack, where firmware, application code and all the security configurations as passwords and cryptographic keys are preserved.

In order to protect the flash memory of the device, several mechanisms have been implemented inside the ECU from TriCore™ and HSM side respectively.

AURIX™ TC2xx devices include up to three PFLASH memory sectors dedicated for the HSM: S6 with 16 KB, S16 and S17 with 64 KB each, for a total PFLASH size of 144 KB.

AURIX™ TC3xx devices include up to 40 PFLASH memory sectors of 16 KB each dedicated for the HSM: from S0 to S39 for a total amount of 640 KB.

For more details please refer to Infineon AP32391.
2.4.1 TriCore™ side

AURIX™ TC2xx and TC3xx provide several protection mechanisms to restrict the read/write operation over NVMs:

- Write protection for program flash (PFLASH) with 256-bit password
  This protection can be activated per logical sector.
- Read protection with same 256-bit password
  This mechanism includes also the global write protection.
- Additionally two types of OTP protection per logical sector

OTP – One time programmable. This mechanism is used for program data and to lock UCB sectors. Once the protection is applied, the flash sector cannot be erased or rewritten anymore.

WOP – Write-once protected. Applicable to the PFLASH sectors as for the OTP, this mechanism protects the flash after first write attempt. Once the flash sector is programmed, it becomes read only.

2.4.2 HSM side

HSM provides a separate protection for its own program and data flash. HSM flash sectors are located in the first 40 logical sectors of PFLASH bank 0 for TC3xx device and only in the Sector 6, 16 and 17 in the TC2xx devices.

The program and data flash dedicated for the hardware security module are not physically separated from the rest of the flash device, but they can be logically segregated with two separate protections:

- OTP protection
- HSM exclusive protection

The exclusive protection mechanism allows that only HSM can have access rights to its own flash sectors. HSM can program or erase these sectors if it is not OTP protected. Without HSM exclusive protection normal program flash protection is effective.

2.4.3 Overview of flash protection

As described before, the following resources can be OTP protected by the customer:

- TriCore™ program flash sectors
- HSM program flash sectors
- HSM boot configuration which includes among the others the HSM boot enabled, SSW waits for secure boot, HSM exclusive protection
- UCBs for boot mode headers and for the SOTA to enable the swap mechanism

The following data are instead already OTP protected by Infineon:

- UCB_SSW: which contains trimming data applied by SSW during boot
- UCB_USER: which includes the UID (unique chip identifier), trimming and measurement data for use by application SW
- UCB_TEST: which includes test information
2.5 Noise generation

The aim of this mechanism is to obfuscate any operation executed during the HSM boot, for example the secure boot, to increase resistance against differential power analysis.

2.6 Clock glitch detection

Another sensible topic and crucial attack is the manipulation of the clock in order to create a glitch and modify the normal execution of the security application code.

The AURIX™ device is equipped with a clock control unit (CCU) which is supplied with a fast reacting clock monitor that supervises the SPB frequency clock against violations of its maximum design target of 100 MHz.

This monitor is configured and enabled by the SSW.

Once the over clocking event is detected by the sensor, this event triggers an interrupt which can cause a system or an application reset.

Another attack can be obtained manipulating the external clock which is providing the base frequency for the PLL (phase lock loop). So in case of an application reset, the execution of the SSW can be corrupted.

As countermeasure, AURIX™ provides an automatic disconnection from the external clock in case of a loss of lock of the PLL.

In case of a clock error, the CCU switches to the backup clock fBACK as the clock source.

A clock error is defined by the occurrence of at least one of the following conditions:

1) loss of lock event of the System PLL while selected as clock source for the CCU (CLKSEL = 01B)
2) loss of lock event of the Peripheral PLL while selected as clock source for the CCU (CLKSEL = 01B)

The clock control registers CCUCON3 and CCUCON4 are used to monitor the status of each PLL, backup clock, SPB clock and in order to set the upper and lower thresholds.
2.7 Voltage and temperature monitoring

The HSM provides the possibility to receive 10 inputs of sensors located in the host system. These sensors signal that the HSM is operating outside the range specified, detecting whether there is an attempt of a fault injection attack via supply voltages (5.0 V, 3.3 V or 1.2 V) or via temperature.

Figure 4 Example of PLL and clock distribution
3. Overall system security

3.1 System related measures

What does it mean that a car is secure? It means that the defined assets are protected against the defined attacks. With regard to the car system, the assets that are considered as the highest level are the application authenticity, integrity, confidentiality and availability.

To define the possible attacks, the cyber security concept of the car has to consider all possible interfaces to and interactions with the environment e.g. external communications, OEM or third-party servers, production line, garage, possible different owners and users. It is also of high importance to consider the life cycle of the car e.g. development, production, operation, maintenance, decommission (see 3.3.6 in GRVA-2019-02, [9]), which brings additional constraints and lead to additional cyber security feature needs and hence attack potential.

Moreover it is recommended that vehicle security architects follow general cyber security principles. One proposal for such principles comes from the “Proposal for a Recommendation on Cyber Security” of the Inland Transport Committee from the United Nations [9].

Therefore some of the key pre-requisites for a cyber-security protection concept are to:
- define a cyber-security policy management
- protect assets with multiple layers of security
- detect intrusion and react to possibly adapt the asset’s protection
- actively manage private or secret keys to reduce their value for an attacker

The following sub chapters give examples on how to reduce the motivation and the window of opportunity for an attacker, and increase the complexity of an attack.

3.2 Multiple layers of security

Defense in depth, also known as castle principle, is a well-known strategy to protect assets not only in the IT domain but also in general. Having such a strategy in place is also part of the 10 cyber security principles listed by the United Nations for “Automated/autonomous and connected vehicles” [9]. The goal of multiple layers is that if an attacker is compromising a single protection layer, still he cannot compromise the asset.

3.3 Secured communication

The protection of the assets transferred over communication channels between the car and an external server or between ECUs within the car is one of the first measures to implement in the system. Considering the automotive constraints, some of these assets also have hard real time requirements (e.g. a braking order) and therefore need the support of additional hardware [e.g. chapter 2.1.5] to reduce the processing latency.

Besides the latency of a single signal, it is also the amount of signals considered as assets to be processed in a period of time which leads also to the need for certain hardware acceleration.
3.3.1 Key management system

The motivation for an attacker to get a key value is related to the gain he will get out from knowing the secret key. Hence, the keys should be managed in such a way that the gain for an attacker is as minimal as possible. This is reducing some of the motivations for an attacker to attack an ECU. If such a gain is still quite high then a secure element has to be used (e.g. Infineon OPTIGA™ TPM).

A classical counter example is having one single symmetrical key used to protect a specific asset in thousands or millions of cars. Such a key is of high interest for an attacker who would invest time and money within an identification phase to get its value. This key could then be used within an exploitation phase to compromise thousands of cars. In contrast to this a key being exchanged, for example, at every key-on cycle is of almost no value for an attacker.

3.3.2 Intrusion detection and prevention system

In order to reach a better protection of the assets, the system can detect and adapt its behavior to the attack currently being executed. For the detection part, the system can measure and analyze key properties of the system. Either the analysis can take place within the car or on a backend having a higher computation power.

The advantage of having (part of) the analysis taking place on the backend also allows to consider information gathered from other cars at the same time or in the past. This leads to a more accurate evaluation if an attack is currently being executed.

Once the decision has been taken that an attack is taking place, the system has to adapt its behavior and apply protection measures. The protection measure can be diverse e.g.:

› reducing the available features
› disabling the usage of specific keys
› adapting Ethernet filters

The action can also consist of (only) logging or reporting the security related events to a car central ECU or to an OEM backend.

3.3.3 Reduced window of opportunity

Security related features can be made available only in specific car’s states where the feature is expected to be used, i.e. some features can be disabled or not be enabled under certain conditions.

During the transportation of a car from the production line to the garage, there is no reason to enable additional car features, or perform a pairing to a new ECU. Similarly, if an ECU is not able to communicate with the other ECUs it is paired with, it can reduce its nominal functionality and hence avoid using the related keys.

Having such principle in place is reducing the window of opportunity for an attacker to perform some attacks, hence increasing the security level.
3.4 Software related measures

Secure coding measures aim to detect an attack, avoid or limit its impact and also react to the detected attack. All existing attacks can be categorized in the following 4 types:
1. Logical attacks
2. Side channel attacks
3. Fault injection
4. Invasive attacks

For the fourth type of attack “Invasive attacks” the software may contribute less to the detection and protection of the assets. The software can implement countermeasures and contribute to the protection of the assets for the type of attacks listed in the sub chapters below.

3.4.1 Logical attacks

The software logical weaknesses are in general the ones having the highest risk since, depending on the location of the ECU in the car network, attacks may be performed remotely. The consequence is that the impact is not only on one but potentially on a huge number of cars.

3.4.1.1 Protocol based weaknesses

A communication protocol between two entities over an insecure channel can lead to a numerous type of attacks e.g. spoofing, man-in-the-middle, eavesdropping, replay, denial of service. Developing a custom secure protocol has a high chance of containing weaknesses and it is strongly recommended to use standardized protocols. Note that even standardized protocols may be vulnerable to attacks e.g. [KRACK] [6] and need updates.

3.4.1.2 Programming weaknesses

A major part of the real time automotive embedded software is written in C language. In order to write software which does what it is intended to, it is first of all important to consider the unspecified, undefined and implementation-defined behavior of C [C99] [16]. Moreover, for the development of safety related software, it is recommended to consider the best practice published by MISRA [2].

Additionally, considering an attacker who wants to misuse the implemented software and tries to compromise an asset by executing software out of its specified boundaries, it is required to further extend the coding guideline to e.g. check tainted values. For this purpose, MISRA published a “MISRA C:2012 Amendment 1” [3]. Another standard which can be considered is CERT C [1] published by the Computer Emergency Response Team of the Carnegie Mellon University [8].

3.4.1.3 Database

Besides the guidelines on how to write software, it is also recommended to check the implemented code against known weakness databases e.g. CVE [4] and CERT VU [5].
3.4.1.4 Fault containment

Having a strategy in place to only use standardized protocols, to check the known vulnerabilities databases and applying secure coding rule can unfortunately not ensure that the software is not containing weaknesses. It is therefore required, besides the weakness avoidance measures, to implement measures to limit the possible impact of a fault. The overall strategy for this is to restrict the resources a piece of software has access to. This is possible through:
› Different privilege levels
› Software partition isolation

3.4.1.4.1 CPU privilege level

A CPU can run in different hierarchically organized modes. Each of these modes gives more or less access to the microcontroller resources. The modes are built so that a code running with less access to resources cannot give himself more accesses. This principles build the hierarchical privilege levels.
The levels are used for example to run the operating system as supervisor mode and the application in user mode. This allows to limit the impact of software weaknesses in the application code since it is executed in a mode having less access rights.

3.4.1.4.2 Isolation

The isolation is a feature to ensure a spatial and temporal freedom from interference between application or software partitions. The temporal property can be ensured by an operating system using interrupts or traps.
The spatial isolation, commonly also handled by an OS, has to rely on a hardware property to restrict the address space accessible by a specific application, task or software partition. In the case of AURIX™ microcontroller this is done by the memory protection unit (MPU).

It is the task of the software integrator to setup this MPU and limit the memory address space, an application has access to, as much as possible. Moreover the type of access also has to be accordingly configured (see chapter 2.1.12).
3.4.2 Timing attacks

Timing attacks aim to derive knowledge of a confidential information by analyzing the timing behavior of the system. The time property or the variation in the time based on a confidential asset value may originate from the hardware or the software processing time. In case the hardware is not providing a processing time independent from the confidential information (e.g. a key value) then the software can implement a countermeasure to provide the information outside of its function or outside of the microcontroller based on the upper boundary value of the hardware processing time.

In case the software itself is the cause for this time variation, then either the same counter measure can be implemented or the software can be written in such a way that its execution time is independent from the confidential value.

A well-known example of a bad implementation is to exit a loop checking e.g. a MAC or a password as soon as one of the byte of the data is incorrect. This reduces the complexity from an exponential to a linear factor. The Code Listing 1 gives such an example where the function strcmp is returning as soon as one char is different. A better solution is to perform a loop over all elements and performing an OR over all element’s comparison as described in Code Listing 2.

**Code Listing 1 – Comparison – bad example**

```
001 int strcmp (const char * s1, const char * s2)
002 {
003 for(; *s1 == *s2; ++s1, ++s2)
004 if(*s1 == 0)
005 return 0;
006 return *( unsigned char *)s1 - *( unsigned char *)s2;
007 }
```

**Code Listing 2 – Comparison – better solution**

```
001 error |= *s1 ^ *s2
```
3.4.3 Power analysis attacks and electro-magnetic analysis attacks

The following chapters list the major classes of analysis and give a short overview.

3.4.3.1 Simple power analysis

Simple power analysis (SPA) consists of measuring the power or the EM radiation (SEMA) of one cryptographic operations and derive, based on the single measurement, the secret key value. As an example: the measurement of the power or of the electro-magnetic radiation of an RSA operation allows to distinguish square and multiply operation which are directly related to the secret key value. Figure 5 [13] gives an example of such an attack.

![Figure 5 Simple power analysis to derive key value][1]

3.4.3.2 Differential and correlation power analysis

Differential and correlation power analysis (DPA) was introduced by Kocher et. al. [11], [12]. It consists in performing statistical analysis over hundreds or thousands of measurements, a classification of these measurements in groups based on an assumed key value. The analysis leads to a positive result if the assumed key value is correct.

3.4.3.3 Higher order attack and multivariate attacks

The scientific community is constantly working on extending the options and possibilities of power or electromagnetic analysis. The DPA contest [7] is a good example for it. Instead of considering the assumed value of the crypto algorithm in a specific computation step, the attack is either considering the specificities of the possible implemented counter measures against SPA (HOA) or considering multiple parameters in different computation steps of the cryptographic algorithms.
3.4.3.4 Countermeasures

The countermeasures against power and electromagnetic analysis can be classified in following types:

1. Noise generation
2. Information leakage reduction
3. Masking of data and operation

Noise generation
Noise can be generated in the power consumption to reduce the signal to noise ratio. Such a measure is only useful for SPA. Even a simple mean over multiple power analysis traces would allow to perform a successful SPA.

Noise can also be generated in the timing behavior of the cryptographic function. This leads to higher effort and time required to analyze, understand and remove the effect of the jitter within the cryptographic computation.

Information leakage reduction
Information leakage reduction consists of performing the required operation in such a way that the measured power or EM radiation is independent from the secret value.

Example of measures:
Use data with a constant hamming weight e.g. “01” and “10” instead of 0 and 1.

This is possible in software where ever magic numbers can be used.

The operation on secret data and another operation leading to a constant power have to take place at the same time (for power and EM) and at a not distinguishable physical location (for EM).

The software cannot implement measures against EMA for this method since it cannot run two operations at the same location and at the same time. Note that, in general, the distance between two cores is bigger than the granularity of an EM sensor.

Masking of data and operation
The masking of data can be done for example by XORing with a random generated mask value in different places within the crypto algorithm. The goal is that the intermediate values within the cryptographic algorithm are never available unmasked.

The masking of operation can be done by performing the required operation within a sequence of dummy operation at a random position.

Note that having masked data would also lead to execution of operation no more related to the secret for RSA and ECC.

Many papers are available related to this topic, e.g. [14], [15].
3.4.4 Fault injection

3.4.4.1 Example of attacks

There is a wide variety of ways how to perform a fault injection:
› Voltage spike
› Clock spike
› Electromagnetic impulse
› Temperature out of specification
› Radiation
› Light impulse with different wavelength

Many papers are available related to this topic, describing how a fault can lead to compromise an asset. In a first step, the injection of a fault leads to a modification of the:
› executed instruction, e.g.
  – replacing one instruction with another one having no effect
› data value, e.g.
  – program counter to jump to another program location
  – data use in decision point (open debugger)
  – data within a cryptographic operation
  – data within a configuration register

3.4.4.2 Countermeasures in software

Countermeasures in software cannot avoid the occurrence of faults, but can limit the impact or detect the corruption.

The MPU helps to limit the impact of a fault by limiting the access and execution rights. This is already described in chapter 1.

The other measure deals with the detection of the corruption:
› Control flow, e.g.
  – generate a checksum over the executed path
  – use a watchdog

Check data integrity, e.g.
  – based on hardware features checking the integrity of the values (e.g. Error Correcting Code)
  – using magic number having a big hamming distance between valid values e.g. 0xA5 and 0x5A
  – values with all 0s and all 1s should never be used
  – Use checksums e.g. an XOR of a value and its inverted value should always generate “all 1s”.
› Execute some critical code section with redundant code and diverse variables e.g. Code Listing 3

Code Listing 3

```c
001 if (value == 0xA5) {
002   if (valueInverted != 0x5A) {
003     error();
004   } else { DoACtion();}
005 }
```
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Term</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>Advanced encryption standard, NIST FIPS 197</td>
</tr>
<tr>
<td>ASIC</td>
<td>Application-specific integrated circuit</td>
</tr>
<tr>
<td>AURIX™</td>
<td>Automotive real-time integrated architecture</td>
</tr>
<tr>
<td>BMHD</td>
<td>Boot mode header</td>
</tr>
<tr>
<td>BOS</td>
<td>Boot system</td>
</tr>
<tr>
<td>BSI</td>
<td>Bundesamt für Sicherheit in der Informationstechnik</td>
</tr>
<tr>
<td>CAN</td>
<td>Controller area network</td>
</tr>
<tr>
<td>CBC</td>
<td>Cipher block chaining</td>
</tr>
<tr>
<td>CCU</td>
<td>Clock control unit</td>
</tr>
<tr>
<td>CMAC</td>
<td>Cipher message authentication code</td>
</tr>
<tr>
<td>COM</td>
<td>Communication mode</td>
</tr>
<tr>
<td>CPU</td>
<td>Central processing unit</td>
</tr>
<tr>
<td>CTR</td>
<td>Counter mode</td>
</tr>
<tr>
<td>DFLASH</td>
<td>Data flash</td>
</tr>
<tr>
<td>DPA</td>
<td>Differential power analysis</td>
</tr>
<tr>
<td>ECB</td>
<td>Electronic code book</td>
</tr>
<tr>
<td>ECC</td>
<td>Error correcting code or elliptic curve cryptography</td>
</tr>
<tr>
<td>ECDSA</td>
<td>Elliptic curve digital signature algorithm</td>
</tr>
<tr>
<td>ECU</td>
<td>Electronic control unit</td>
</tr>
<tr>
<td>EE</td>
<td>Electrical/electronic</td>
</tr>
<tr>
<td>EMEM</td>
<td>Extended memory</td>
</tr>
<tr>
<td>FIPS</td>
<td>Federal information processing standard</td>
</tr>
<tr>
<td>GTM</td>
<td>General timer module</td>
</tr>
<tr>
<td>HIS</td>
<td>Hersteller Initiative Software</td>
</tr>
<tr>
<td>HOA</td>
<td>Higher order attack</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware security module</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of things</td>
</tr>
<tr>
<td>LFSR</td>
<td>Linear feedback shift register</td>
</tr>
<tr>
<td>LRU</td>
<td>Least recently used</td>
</tr>
<tr>
<td>MAC</td>
<td>Message authentication code</td>
</tr>
<tr>
<td>MBIST</td>
<td>Memory built-in self-test</td>
</tr>
<tr>
<td>MPU</td>
<td>Memory protection unit</td>
</tr>
<tr>
<td>NIST</td>
<td>U.S. National Institute of Standards and Technology</td>
</tr>
<tr>
<td>NVIC</td>
<td>Nested vector interrupt controller</td>
</tr>
<tr>
<td>OCDS</td>
<td>On-chip debug support</td>
</tr>
<tr>
<td>OEM</td>
<td>Original equipment manufacture</td>
</tr>
<tr>
<td>OFB</td>
<td>Output feedback</td>
</tr>
<tr>
<td>Acronym</td>
<td>Term</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>OPC</td>
<td>Operation code</td>
</tr>
<tr>
<td>OS</td>
<td>Operating system</td>
</tr>
<tr>
<td>OTP</td>
<td>One-time programmable</td>
</tr>
<tr>
<td>PFLASH</td>
<td>Program flash</td>
</tr>
<tr>
<td>PKC</td>
<td>Public key cryptography</td>
</tr>
<tr>
<td>PLL</td>
<td>Phase lock loop</td>
</tr>
<tr>
<td>PMSA</td>
<td>Protected memory system architecture</td>
</tr>
<tr>
<td>PSIS</td>
<td>Peripheral sensor interface 5</td>
</tr>
<tr>
<td>RAM</td>
<td>Random access memory</td>
</tr>
<tr>
<td>RCU</td>
<td>Reset control unit</td>
</tr>
<tr>
<td>ROM</td>
<td>Read-only memory</td>
</tr>
<tr>
<td>RSA</td>
<td>Rivest–Shamir–Adleman, U.S. Patent 4,405,829</td>
</tr>
<tr>
<td>RW</td>
<td>Read/write mode</td>
</tr>
<tr>
<td>SEAMA</td>
<td>Simple electro-magnetic radiation analysis</td>
</tr>
<tr>
<td>SFR</td>
<td>Special function register</td>
</tr>
<tr>
<td>SHA</td>
<td>Secure hash algorithms</td>
</tr>
<tr>
<td>SHE</td>
<td>Secure hardware extension</td>
</tr>
<tr>
<td>SoC</td>
<td>System on chip</td>
</tr>
<tr>
<td>SPA</td>
<td>Simple power analysis</td>
</tr>
<tr>
<td>SPB</td>
<td>Serial peripheral bus</td>
</tr>
<tr>
<td>SSG</td>
<td>Self-shrinking generator</td>
</tr>
<tr>
<td>SSW</td>
<td>Startup software</td>
</tr>
<tr>
<td>TPM</td>
<td>Trusted platform module</td>
</tr>
<tr>
<td>TRNG</td>
<td>True random number generator</td>
</tr>
<tr>
<td>V2X</td>
<td>Vehicle to everything</td>
</tr>
<tr>
<td>WOP</td>
<td>Write-once protected</td>
</tr>
</tbody>
</table>
References

[1] [CERT C] https://www.cert.org/secure-coding/

[2] [MISRA] https://www.misra.org.uk/

[3] [MISRA.am1] https://www.misra.org.uk/LinkClick.aspx?fileticket=V2wsZxtVGkE%3d&tabid=57

[4] [CVE] https://cve.mitre.org/

[5] [CERT VU] https://www.kb.cert.org/vuls/


[7] [DPA contest] http://www.dpacontest.org (DPA attacks on crypto algorithms)

[8] [CMU] https://www.cmu.edu/


[13] [BSA] Basics of Side-Channel Analysis, Marc Joye https://pdfs.semanticscholar.org/24b9/1a9bf187c8be70022c8cdc3aa29c7f36c5de.pdf

Multiplicative Masking and Power Analysis of AES

[15] [SAMM] Elena Trichina and Domenico De Seta and Lucia Germani
Simplified Adaptive Multiplicative Masking for AES

[16] [C99] ISO/IEC 9899:1999


Where to buy

Infineon distribution partners and sales offices:
www.infineon.com/WhereToBuy

Service hotline

Infineon offers its toll-free 0800/4001 service hotline as one central number, available 24/7 in English, Mandarin and German.

› Germany ................. 0800 951 951 951 (German/English)
› China, mainland ........ 4001 200 951 (Mandarin/English)
› India .................... 000 800 4402 951 (English)
› USA ........................ 1-866 951 9519 (English/German)
› Other countries ........ 00* 800 951 951 951 (English/German)
› Direct access ........... +49 89 234-0 (interconnection fee, German/English)

* Please note: Some countries may require you to dial a code other than “00” to access this international number.
Please visit www.infineon.com/service for your country!

Mobile product catalog
Mobile app for iOS and Android.