Security Requirements for a Modern CCU

Version: 1.1

Date: 26. März 2015

Executive Summary

This document defines fundamental security requirements for a modern Connectivity Control Unit (CCU). It does not provide specific design or implementation guidelines, but an overview of the requirements which must be fulfilled by a CCU in order to guarantee a specific level of security.

History

Version	*Author*	*Comment*	*Date*
0.1	CSA	Creation of document	2015-02-06
0.2	CSA	Finalization of first draft	2015-02-10
0.3	FST	Review and extensions	2015-02-10
0.4	CSA	Integrate comments from review	2015-02-11
1.0	FST	Review and final approval	2015-02-12
1.1	FST	Integrated comments regarding safety	2015-03-26

Table of Contents

1 Introduction

Nowadays modern cars are connected to various systems outside the core automotive network like infotainment units, vehicle-to-vehicle (V2V) networks, or also to the Internet and the manufacturer backend respectively. Usually, these connections are controlled by a Connectivity Control Unit (CCU) which serves as an information gateway in the vehicle, where it is connected to both the wireless link via a GSM module or even more sophisticated cellular communication technologies (UMTS, LTE) and to the internal busses via the OBD II or possibly different interfaces.

However, automotive security covers more aspects than only securing a CCU. It covers protection of electronic control units (ECU), in-vehicle communication, and external interfaces against malicious encroachments by an attacker. By providing an additional external communication interface to the vehicle data busses, the attack surface of the respective vehicle is fundamentally enlarged.

To guarantee a sufficient security level for such a CCU, some fundamental requirements will be defined in Section 2.

2 Security Requirements

In the following, we present some overall security requirements that need to be fulfilled for providing secure CCUs. Note that the following requirements do not replace the need for specifying an overall security concept. However, they can be understood as guidelines for developing a security concept.

R1 Secure Communication between CCU and the Backend

To secure the communication between the CCU and the external interfaces, here called backend, six requirements are defined in the following.

In general, possible attacks on the communication can be conducted on two different layers:

• Communication between sensors, other ECUs and the CCU.

• Communication between CCU and backend.

Since the vehicle bus and therefore the sensors and the ECUs are standardized and well established, the definition of security requirements for the communication with the internal vehicle bus is out of scope in this report. It is obvious and well-known that similar security risks are existent in the automotive communication ecosystem which are already addressed by several manufacturers and publicly founded projects [1] [2].

R1.1 Ensure Authenticity and Integrity of Transmitted Data

To prevent modification of data sent from the backend to the CCU or to prevent that faked data is received by the backend and processed, it is important that the integrity of the message is ensured and at the same time only an authorized person could have send the message. There are two main methods to provide integrity and authentication of a message:

• Message authentication code (MAC)

• Digital signature

Both methods have in common that it is hard to generate a valid MAC or signature for a given message without knowing the secret token. Also, it is hard to find a so-called collision, i.e., a second message that has the same MAC or signature as given message with a valid MAC or signature.

A MAC is generated symmetrically, that means that the generation and the verification are both done using the same key. For constructing a MAC, either a hash-based (HMAC) or a cipher-based (CMAC) approach can be used. Since many modern automotive-qualified controller are equipped with an AES hardware accelerator, the AES-CMAC should be the first choice to secure messages.

Digital signatures are based on asymmetric cryptography, i.e., a private key is used to generate the signature, while a public key which is publicly available is used to verify the signature. The advantage over a MAC is the fact that the public key must not be stored secretly which results in an easier key management. It is common to use signatures together with private-key certificates based on the X.509 standard.

R1.2 Ensure Confidentiality of Data

To ensure confidentiality of secret data sent to or from the backend, like geo-information data or personal data, the data stream must be encrypted with a secure encryption algorithm like the Advanced Encryption Standard (AES). Furthermore, a suitable mode of operation like CBC together with a MAC so ensure the integrity or an Authenticated Encryption like the GCM or CCM must be used.

R1.3 Mutual Authentication of Entities

For a secure communication line between the CCU and the backend, a mutual authentication must be done prior the data transmission to ensure all other security requirements. That means that both communication partners, like the CCU and the backend is authenticating each other at the same time. This is usually done with a challenge-response algorithm based on asymmetric cryptography, with the help of a client and a server certificate like done in the *Transport Layer Security* (TLS) protocol.

If the communication parties share a common secret, the mutual authentication can also be done via a symmetric algorithm like secure variants based on the Needham-Schroeder protocol.

R1.4 Ensure Freshness of Data Messages

An easy accessible attack vector is a replay attack which means that a message is recorded and played back later by an attacker.

To avoid replay attacks on the communication between the CCU and the backend, it is necessary to add sequence numbers, timestamps or another variable component like nonces to the messages. Note that these sequence numbers or nonces need to be integrity-protected together with the message via a MAC or a digital signature, described in Requirement R1.1.

The generation of the timestamps or nonces must also be done in a secure manner. That means that timestamps must be calculated via a Real Time Clock (RTC) which cannot be tampered and must be synchronized with the backend server. Nonces must be generated by a cryptographic secure (pseudo) random number generator.

R1.5 Ensure Perfect Forward Secrecy

To ensure perfect forward secrecy of all communication, the system must generate session keys with a non-deterministic algorithm from long-term key material for each new session. After a session is closed, the used session key must be destroyed in order to prevent a decryption of previous recorded messages if the system is broken later and keys are extracted.

Through this requirement, it is ensured that an attacker cannot decrypt messages which were recorded in the past, even if he is able to break into the system and can extract key material. By using a non-deterministic key generation algorithm, e.g., by using randomness for each key generation, an attacker cannot compute future session keys from the key material, leading to a system with the Perfect Forward Secrecy (PFS) property.

R1.6 Usage of a Suitable Protocol for the Secure Communication

To secure the network communication and to comply with the above requirements, the CCU needs to make use of existing network protocols. Two good options are the *Internet Protocol Security (IPsec)* protocol and the *Transport Layer Security (TLS)* protocol.

IPsec can be used to implement an authenticated and encrypted communication between two network hosts. IPsec itself is, strictly spoken, no protocol, but a protocol suite, as it compasses several protocols for particular purposes: for the key exchange, the *Internet Key Exchange (IKE)* protocol is used. If the communication between hosts only needs to be authenticated, the *Authentication Header (AH)* protocol can be employed. In case that content also needs to be encrypted, it is recommended to use the *Encapsulating Security Payload (ESP)* protocol that allows for both encrypted and authenticated communication. In fact, the AH protocol should be avoided due to its limited use. Additionally, it is important to use the ESP protocol never in the so-called encryption-only mode, but always with authentication being enabled. Otherwise, the protocol lacks defensive measures against message manipulation and attackers can attempt to re-route messages or manipulate targeted option or content fields of the transmitted messages.

The details on the secure configuration of IPsec exceed the scope of this document. The reader is therefore referred to the technical guideline TR-02102-3 of the *Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI)* [3].

TLS provides security not between network hosts, but *Transport Control Protocol (TCP)* ports, i.e., between applications. The protocol has been developed over several versions and the latest version is 1.2 (as of 2015/02). It is recommended to always use the latest protocol version exclusively, because as for all protocols, a large number of practical attacks against the TLS protocol in earlier versions are known. To address several risks through the choice of an insecure cipher suite or parameters, the BSI has published a technical guideline for the secure configuration of the TLS, namely TR-02102-2 [4]. Furthermore, the draft for TLS 1.3 [5] shows a large step towards a simpler protocol with only few but secure algorithms and parameters. For example, the data stream compression functionality which can lead to so-

called entropy attacks will be removed for TLS 1.3. Also, support for all non-AEAD cipher suites, which were responsible for several attacks on TLS in the past, was removed.

For the CCU, TLS is more relevant than IPsec, because for the use cases of a CCU, the communication on application layer needs to be secured rather than on the network layer. Therefore, Table 1 gives a recommendation for basic TLS 1.2 configuration parameters, whereby Table 2 gives recommendation for extended TLS 1.2 parameters. These recommendations are considered safer due to a doubled symmetric AES key length and longer SHA-2 hash values. The choices of the parameters are mainly related to the used controller, the supported hardware accelerators and the targeted performance.

	Key Agreement		Encryption	Mode	Hash	Use Until
TLS_	ECDHE_ECDSA_	WITH_	AES_128_	CBC_ GCM_	SHA256	2020+
	ECDHE_RSA_	WITH_	AES_128_	CBC_ GCM_	SHA256	2020+
	DHE_DSS_	WITH_	AES_128_	CBC_ GCM_	SHA256	2020+
	DHE_RSA_	WITH_	AES_128_	CBC_ GCM_	SHA256	2020+

Table 1: Recommended Basic TLS configuration parameters

	Key Agreement		Encryption	Mode	Hash	Use Until
TLS_	ECDHE_ECDSA_	WITH_	AES_256_	CBC_ GCM_	SHA384	2020+
	ECDHE_RSA_	WITH_	AES_256_	CBC_ GCM_	SHA384	2020+
	DHE_DSS_	WITH_	AES_256_	CBC_ GCM_	SHA384	2020+
	DHE_RSA_	WITH_	AES_256_	CBC_ GCM_	SHA384	2020+

Table 2: Recommended Extended TLS configuration parameters

R2 Unique Cryptographic Identities

A key aspect of a secure CCU is the injection of unique identities which are required for most of the security requirements. The usual way for this is the employment of certificates based on strong asymmetric cryptography like RSA or ECC. The certificates bind a public/private key

pair to a specific ID which belongs to one unique device. In this way, every CCU has its own key pair and is unambiguously identifiable.

On the other hand, every CCU stores the public key of the backend server in order to perform an authentication protocol prior the communication.

R3 Hardware-based Security

Since the CCU is built into the car and not isolated in a secure environment, an attacker has physical access to the device. This special aspect has to be addressed by the security design and lead to several fundamental requirements towards the used hardware.

R3.1 Key Storage

There are various ways to store key material securely on the CCU. In this report, four different methods are proposed:

- On-chip One-Time-Programmable (OTP) space
- On-chip SRAM storage spaces
- Encrypted storage of private keys
- Usage of Hardware Security Modules to protect cryptographic keys (Compare Requirement R3.3)

A distinction must be drawn between public and private keys. Whereby a public key must be stored tamperproof or at least tamper evident, a private key must be stored in a way that an attacker cannot extract the key.

As a result, the public key of the backend must be securely stored by the device. Since the key is publicly known and is not confidential, a possible attacker can extract the key without any harm to the overall security of the system. Nevertheless, the public key must be stored read-only, which means that the key must be secured against unauthorized modifications, which could give the attacker the possibility to inject his own public key and lead to a successful impersonation attack.

To fulfill this requirement often some fuse technology is used where the fuse can be burned with the public key and is automatically locked after the write process. Another way can be the usage of a SRAM-based key store whereby the key can also be updated if the update process is started with the correct credentials. Since a SRAM chip is erased at a power loss, the memory should be buffered by an additional battery.

A higher security level is required for the secure storage of the CCU's private key. Since this key is used for all other cryptographic operations like the authentication or for the encryption of the data stream the key must be stored confidential. One way to protect the key is to encrypt the updateable private key with a master key which is generated at production and then made unreadable by blowing some specific fuses. Another technique is to have some integrated memory which is hardened against unauthorized access and physical attacks respectively, and where it is difficult to extract the key. Such techniques are offered by some chip manufacturers to additionally secure keying material from unauthorized extraction.

However, the generation - especially of asymmetric key material - is a complex task and is likely to take a significant amount of time at the end-of-line test of a production phase, which might be unacceptable for the CCU manufacturer. Therefore, a possible alternative is that the cryptographic keys are be generated by an external unit, for example a Local Registration Authority (LRA) that resides at end-of-line of production and can communicate with the CCU. After generating key material by the LRA, the LRA injects the cryptographic material in the CCU, where it is stored. The private keys have to be permanently removed from the LRA to avoid a leakage of confidential data.

R3.2 Hardware Support/Acceleration for Cryptographic Algorithms

For nearly all security related operations some cryptographic algorithms are used. Since these algorithms are costly in terms of memory and time, many automotive-qualified controller support some of the most common algorithms like the AES.

Due to the fact that many cryptographic protocols require a secure random number generator, the hardware should have some built-in true random number generator (TRNG). Since most of the protocols only require a deterministic random bit generator (DRBG) which can be built efficiently in software, the TRNG, which usually has a low throughput, must be used to seed the DRBG with a real random value in order to get pseudo-random numbers with sufficient entropy.

R3.3 Usage of Hardware Security Modules

The most secure approach to protect cryptographic keys is to employ a dedicated security controller (Hardware Security Module) which provides protections against physical extraction of cryptographic keys. HSMs are integrated circuits specifically developed and designed for security use-cases. Typically, implementations range from smart cards used for identification and authentication purposes, such as, national identification cards, to Trusted Platform Modules which are Hardware Security Modules for personal computers. HSMs typically consist of a CPU core, different types of data storage (e.g., RAM, ROM, Flash), a memory protection unit, a memory encryption unit, sensors, cryptographic accelerators, and further peripheral components. Most HSMs employ sophisticated countermeasures against physical attacks, such as active sensors to detect fault and glitch attacks, and also employ cryptographic implementations which are hardened against side channel attacks.

R4 Use Strong Cryptography with Sufficient Key Lengths

To implement the security mechanisms, it is necessary to use only strong and well-standardized algorithms with common recommended key lengths. One widely accepted recommendation of algorithms and key lengths is the technical guideline TR-02102-3 of the *Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik*, *BSI* [6]. This guideline defines methods and parameters which have to be used for a specific security level and specific lifetime of a device or system respectively. For example, the encryption algorithm AES is the only recommended block cipher and should be used with the Galois-Counter Mode (GCM), Cipher-Block Chaining (CBC), or Counter Mode (CTR). For asymmetric cryptographic, RSA with at least 2048 bits or ECC with the brainpool curve P224r1 or larger is recommended.

For the development of a new CCU the recommendation for algorithms and key sizes must be used in order to design a secure system.

R5 Harden CCU Operating System

The operating system (OS), usually a Linux-based OS, used on the CCU should be hardened against typical vulnerabilities. Two recommended kernel modules which introduce some advanced security policies like Mandatory Access Control (MAC) are SELinux, and Apparmor. If possible, an additional hypervisor-based approach should be implemented to isolate security operations from other task on the controller.

R6 Employ a Security Engineering Process during CCU Development

During all phases of the design and development of the CCU, security aspects have to be considered. In detail, a security engineering process has to be introduced in order to ensure that all security critical architecture decisions are well evaluated and are correctly implemented. This includes regular meetings and discussions with representatives of all in the development involved departments and the security department as well as regular reviews of the architecture and the implementation itself.

R7 Perform Penetration Tests of the CCU

Even though a well-documented design process with focus on the security and mandatory reviews are applied during the development of the CCU, some critical flaws and weak spots can appear in the specific device. Especially in the production phase of the device, which is mostly done from an external manufacturer or supplier, vulnerabilities like open debug ports or insecure debug routines can appear. To close the gap between the designed security architecture and the manufactured device, a rigorous penetration test of the CCU must be performed. Ideally, this test should be done by independent and experienced security experts.

R8 Restrict Communication to the Vehicle Bus

One main threat caused by integrating and attaching the CCU to the on-board diagnostic board of a vehicle is that the CCU gets direct access to the vehicle board-network and may inject or manipulate safety-related messages. As a result, the CCU may inflict critical damage to the whole vehicle board network or to the different safety-related ECUs. Therefore, the process of generating messages for the vehicle board network must be tightly controlled to prevent that arbitrary safety-critical messages can be generated by the CCU leading to safety-critical behavior of the vehicle.

To inspect and control messages generated by the CCU for the vehicle board network, every message which is generated needs to be controlled and validated, i.e., basic firewalling mechanisms should be employed. This includes message inspection based on a whitelist which allows only messages which are explicit listed in this list. The remaining messages are dropped and not transmitted to the vehicle bus. A more sophisticated approach is the implementation of a Stateful Package Inspection (SPI) which also evaluates the time and state aspect of the messages and will allow or drop messages based on the current protocol state.

To this end, before a message is passed to the device driver which delivers the message to the OBD-bus, it needs to be inspected by a dedicated security component. This security component could be realized as an own kernel component or module or a dedicated security application which is able to inspect every message delivered to the OBD interface.

However, a more secure solution is the realization of the firewall component on a separate compliant microprocessor which is isolated from the remaining application processor where the Linux-OS is executed. This means that the different networks, namely the in-vehicle bus like CAN, FlexRay or Ethernet are connected to the separate microprocessor which communicates with the application processor via inter node communication. As a result, the in-vehicle network is clearly isolated and communication to the external network like WiFi, GSM or LTE is realized via dedicated firewall components.

R9 Secure Boot

To ensure a trustworthy system, the boot process of the firmware must be secured. Usually, secure boot is ensured by a chain of trust whereby the integrity of each component is validated by the previous component. The validation of the firmware is done by a cryptographic hash value of the code which is compared to pre-configured stored value. The initial component works as a core root of trust for measurement and must be stored tamperproof inside the chip, ideally inside a specialized HSM core of the controller. This root of trust is triggering the bootloader and initiating the boot process. Through this measure, the actual firmware and application is validated and run in a trustworthy state.

In order to enable a flexible secure boot (e.g. for security updates), a secure reference update mechanism (e.g., based on a shared secret or a public key scheme) is required, which allows firmware updates in the field.

R10 Secure Flash

To enable the possibility of updates of the CCU's firmware later in the field, a secure flash process must be implemented. Therefore, the integrity and authenticity of a new firmware image must be validated by strong cryptographic mechanisms like a digital signature based on RSA or ECC. Furthermore, the secure boot process must be updated which means that the stored hash value of the firmware and application code must be updated inside the protected memory area.

Particular cautions must be taken if the new image is not validated successful during the update process. In this case, the system must boot the last secure firmware, which means that the new firmware image has to be stored in an empty flash area on the chip and if and only if validated successfully the bootloader must be updated with the address of the new image. That means that the chip must be equipped with enough memory for storing two firmwares concurrently.

R11 Secure Confidential and Private Data on the CCU

To ensure the privacy of the driver and secure confidential data, data must be stored encrypted on the CCU. This can be some geo-information data, cellphone identification data or also some buffered application data like messages from the infotainment domain. Furthermore, unused temporary data should be erased as soon as possible.

R12 Secure Physical Access to the CCU

Many publicly known attacks on embedded devices were done or at least with the help of open debug ports like JTAG or UART [7] or poorly protected external interfaces and

memories. Therefore, these attack vectors should be carefully considered and must be evaluated by a mandatory penetration test described in Requirement R7.

R12.1 Secure Debug

Embedded processors in general and in particular CCUs have several accesses and debug ports and routines. These are essential during the development and production phase where the correct chip behavior and functionality has to be evaluated. But at the same time, these debug ports can give an attacker confidential information or in the worst case full access to the chip and the running application. Therefore, all interfaces which are not required after the production, e.g., JTAG, SPI or UART, must be deactivated and the corresponding pins must be unwired.

If such a debug port is required, for example if a controller has to be replaced and the error must be examined by the manufacturer, this interface must be protected by strong cryptography.

R12.2 Authentication of External Interfaces

All accesses from external interfaces like over the GSM, UMTS or LTE should be authenticated by a mutual authentication scheme between the CCU and the backend as described in Requirement R1.3.

R12.3 Securing the Memory

To restrict the reading of the flash memory, the whole flash should be encrypted with a symmetric block cipher which is usually supported by current automotive-qualified controllers. Since the flash memory is a permanent memory, the attacker can use various methods to extract the data on the flash chip like invasive attacks which destroy the chip. In contrast, the RAM is automatically erased at a power-off. Therefore, it can be sufficient to deactivate all debug accesses like JTAG to protect the RAM, since the attacker can use only attack vectors which leave the chip and the power connection intact. Nevertheless, if a RAM encryption is supported by the used controller, this feature should be used to prevent more sophisticated attacks like the cold boot attack [8].

R13 Employ Mechanisms in the Backend to Isolate Malicious CCUs

Beside the CCU itself, the backend must be protected against potential attacks as well. Since this report only defines requirements for the CCU, the security requirements in of the backend side are out of scope.

Nevertheless, one requirement for the backend must be the isolation and the secure handling of a malicious CCU. In detail, this should include a logging of any potential attack on the network and backend as well as active measures to force the malicious CCU into a fail-safe state like a forced firmware update, a CCU reset or a deactivation of the CCU with a message to the driver or infotainment system respectively that the CCU was compromised and should be examined in a garage. At least, the CCU should be isolated from taking part in the communication by revoking potentially used certificates.

R14 Establish a Private Communication Network

A private communication network should be established by assigning private IP addresses to the CCU. This means that the CCUs are not directly reachable from the Internet but rather only connected to the isolated manufacturer backend system. This should be done by using Network Address Translation (NAT) and by providing a private Access Point Name (APN) which is used by the integrated SIM.

In contrast to a public APN of a mobile communication provider, which usually allows direct access to the Internet, the private APN allocate the CCU's SIM to a private network which is separated by the Internet. By using NAT, the backend maintainer assigns only private IP addresses for the CCUs which are non-routable in the global Internet. Additionally, the router which is connected to the CCU must be configured in a way that the particular CCUs are isolated and cannot send packets directly to each other. Thereby, a CCU has just one single logical connection directly to the backend server which can allow further connections like a link to a music streaming service for the infotainment system or a link to the navigation service provider in order to allow map updates.

3 Bibliography

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham und S. Savage, „Experimental Security Analysis of a Modern Automobile," in *IEEE Symposium on Security and Privacy*, Oakland, 2010.

[2] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner und T. Kohno, „Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *SEC'11 Proceedings of the 20th USENIX conference on Security*, Berkeley, 2011.

[3] BSI, „TR-02102-3 Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Teil 3 - Verwendung von Internet Protocol Security (IPsec) und Internet Key Exchange (IKEv2)," 2014.

[4] BSI, „TR-02102-2 Kryptographische Verfahren: Empfehlungen und Schlüssellängen. Teil 2 - Verwendung von Transport Layer Security (TLS)," 2014.

[5] E. R. T. Dierks, „The Transport Layer Security (TLS) Protocol Version 1.3," 2015.

[6] BSI, „TR-02102-1 Kryptografische Verfahren: Empfehlungen und Schlüssellängen," 2014.

[7] B. Jack, „Exploting Embedded Systems," in *Blackhat Amsterdam 2006*, Amsterdam, 2006.

[8] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum und E. W. Felten, „Lest We Remember: Cold Boot Attacks on Encryption Keys," in *Proceedings 17th USENIX Security Symposium (Sec '08)*, San Jose, 2008.