Evaluation of Lightweight TPMs for Automotive Software Updates over the Air

Richard Petri[1], Markus Springer[1], Daniel Zelle[1],
Ira McDonald[2], Andreas Fuchs[1], and Christoph Krauß[1]

[1] Fraunhofer SIT
Rheinstr. 75
64295 Darmstadt, Germany
{richard.petri|markus.springer|daniel.zelle}@sit.fraunhofer.de
{andreas.fuchs|christoph.krauss}@sit.fraunhofer.de
[2] High North Inc
PO Box 221 / E21761 Ridge Rd
Grand Marais, MI 49839, USA
blueroofmusic@gmail.com

Abstract Given the growing importance of Information Technology in today's vehicles with their ever increasing connectivity and its safety relevance, it is obvious that security technologies need to be employed and improved. Besides secure coding efforts, many of the attack classes and scenarios demand the embodiment of Hardware Security Modules such as the Trusted Computing Group's Trusted Platform Module. This paper discusses the use cases and benefits of TPM usage in automotive ECUs. We further show and evaluate how the Automotive Thin Profile released by the Trusted Computing Group can be used to secure Software Over-The-Air updates. We show in detail, how this use case can be addressed by a combination of different levels of TPM implementations – namely full TPMs and Automotive Thin TPMs. In order to give an estimation of the necessary implementation cost, the first ever measurements of required code and RAM sizes for different TPMs and an overview of TPM implementations are provided.

Keywords: TPM, HSM, Automotive, SOTA, Update, Resources-Constraint

1 Introduction

Modern cars have an increasing number of electronic components due to the development towards autonomous vehicles. Up to 100 Electronic Control Units (ECUs) realize different vehicular functions in hardware and software. ECUs communicate with each other via different bus systems such as CAN, LIN, MOST, FlexRay and Automotive Ethernet. In addition to this, cars become more connected to the outside world, for example using Vehicle to Vehicle communication (V2V) as well as connections to the Internet or the smartphone. Vehicles are also gaining Internet connectivity to provide the driver with the

possibility to stream music, receive e-mails, and receive Facebook messages. It is also possible to unlock the car via smartphone, start or stop the engine, and turn the light on or off.

A primary tool for securing modern vehicles is the use of Hardware Security Modules (HSMs). An HSM provides secure storage for cryptographic keys and a secure execution environment for (cryptographic) operations similarly to classical Smart Cards. In addition to these features, HSMs typically ensure the integrity of the platform. Automotive HSMs have been investigated in the past by different research projects (e.g., [1,2]) and standardization groups (e.g., HIS, SAE Vehicle Electrical Hardware Security Task Force, AUTOSAR HSM WorkGroup, Trusted Computing Group (TCG) - Embedded Systems Work Group - Vehicle Services Subgroup).

In this paper we show how TPM 2.0 technology can help to solve automotive security related problems and give an estimation about the resource consumption of different TPM profiles. After reviewing the state of the art in automotive security research in Section 2 we introduce the scenario that we analyzed in Section 3. In this paper we focus on software over the air updates of ECUs, because remote updates are necessary to fix vulnerabilities of already deployed ECUs, are fast and cost efficient, and they have strong security implications.

We continue exploring TPM 2.0 and its profiles for embedded systems which are designed to enable TPM functionalities with limited resources in Section 4. Furthermore, we show how TPM 2.0 technologies can protect remote updates in a Software Over-The-Air (SOTA) update scenario, where we point out which approaches are needed to establish a secure update mechanism. In Section 5 we present our approach to measuring the resource consumption of our TPM 2.0 implementation and a comparison of these results. Finally we show how the security features of a TPM can increase the security of remote software updates in Section 6. The paper concludes with a summary of our work and an outlook for our research plans on the topic of automotive security in Section 7.

2 Related Work

In recent years, the security of vehicles has been intensively investigated. Researchers have shown the insecurity of Remote Keyless Entry (RKE) systems as well as immobilizers in [3,4]. Because security vulnerabilities have been shown, RKE and immobilizers are subject of newly proposed security measures like key fob air gestures and GPS fencing [5]. Rouf et al. presented an attack on a vehicle via the wireless Tire Pressure Monitoring System in [6]. Different attack vectors and vulnerabilities have been identified by Koscher et al. [7] and by Checkoway et al. [8]. How vehicles can be attacked via their remote interfaces has been analyzed by Miller and Valasek in [9,10]. Their attacks include the update of an infotainment system with a self-written firmware. In [11] Becsi et al. provide another overview of the change of security requirements with the increased connectivity of vehicles and give an overview of identified security problems.

One of the first major attempts to solve some of the security related issues via Hardware Security Modules was introduced by the HIS – SHE specification [12]. The first requirements proposals for TPM-like HSM were then defined within the EVITA project [1], introducing several classes of Hardware Security Modules [13,14]. This work was continued within the PRESERVE project [2], which included flexible choice of HSMs between Evita and TPMs. Alternative approaches based on TPM were discussed widely [15] as well as general approaches of establishing trust in automotive networks [16]. More concretely, the Trusted Computing Group has released a whole new set of specifications suited for use in the automotive context [17,18,19,20,21,22].

In the past there have been several different kinds of proposals for secure software updates. In [23] Hossain and Mahmud investigate the secure upload of software updates to a vehicle over the wireless interface (such as cellular or Wi-Fi). In their proposal, vehicles authenticate themselves with a software distribution center (SDC) and establish a secure session. The SDC packages the software update together with its respective message digest and sends the update to the vehicle twice. Only if the message digest of both received updates can be verified and are equal will the receiving vehicle install the update. Nielsson and Larson proposed a secure firmware updates over the air (SFOTA) protocol in [24]. The protocol relies on preshared symmetric keys for encryption and public keys for the software update portal on the vehicle for data integrity and authenticity. The vehicle does not authenticate itself with the software portal. Mansor et al. analyze the firmware upgrade protocol of EVITA [1] in [25]. Their proposal considers the secure transfer of the flash driver to the ECU as well as fixing a potential problem with "bricking cars" during updates by introducing a rollback mechanism.

3 Software Over-the-Air Update Scenario

Fixing vehicles via software updates is crucial for both the manufacturer and the owner / driver of the vehicle. Recalling vehicles into repair shops in order to update their software and fix problems takes months to complete and is very costly to manufacturers. For that reason the industry is very interested in technologies that prevent and mitigate the need for recalls. One prominent technique to prevent these recalls is the Software Over-The-Air (SOTA) update mechanism. A software update can be installed without bringing a vehicle into the repair shop.

In general an update would be initiated by an OEM or a supplier that forwards the update to an OEM. This update is then forwarded to the vehicle using mobile networks such as GSM, UMTS, or LTE. Once an ECU receives the update it will store the update until there is an opportunity for the installation. This process poses several technical challenges: For one, because of the overall inhomogeneous environments in vehicles due to different models, software components, and configurations, there is a major need for standardized upload and download protocols, as well as the need to only partially update software. There is also a need for a way to recover from unsuccessful updates.

Along with these technical challenges, there exist many security related requirements. For example, the need to verify both the integrity of the software update itself as well as the authenticity of the update's origin. Updates must only be sent by authorized entities and installed after complete verification. Proof of the authenticity of the receiving vehicle is also important for the vendor to prevent the installation of the software updates on unauthorized vehicles. In addition, it should not be possible to downgrade the software version. An attacker may try to downgrade the software to a known vulnerable and exploitable version. Depending on the requirements of the OEM, the confidentiality of software updates will also need to be ensured in order to protect intellectual property. OEMs may also want to keep track of a vehicle's update status and thus require reliable acknowledgment upon successful update installation. This also requires some form of non-repudiation for the vehicle's acknowledgment.

In order to provide authenticity in a scalable fashion, the use of asymmetric cryptography is required. Authenticity of the data also requires the integrity of said data. In case the intellectual property associated with firmware should be protected, confidentiality is another requirement and can be met by the use of either symmetric or asymmetric cryptography. This makes the secure handling, management, and distribution of keys another very important task. Finally, with the deployment of remote firmware updates the aspects of liability and auditing become paramount. Once updates have occurred, the OEM needs to record in an audit log which software versions have been installed in order to provide proof of a successful recall to government agencies. Especially in the face of chip tuning, undesirable side effects can occur and OEMs need ways to defend against liability claims in such cases.

4 TPMs for Resource Constrained Systems

The threats against automotive ECUs in vehicles and automotive E/E systems are manifold. A few examples include the theft of intellectual property (encoded within firmware), theft of privacy-related data (e. g., navigation destinations, driving behavior, address books), manipulation of odometers, illegal feature activation (i. e., activating software and/or hardware features without payment), chip tuning, firmware manipulation (with safety critical impact [10]), firmware downgrade (as a possible intermediate step for an attack), product counterfeiting, and used part refurbishment.

Pure software-based security approaches are incapable of preventing these kinds of attacks, even at a fairly low attack potential. Furthermore, software-based approaches typically serve the purpose of access authentication and attack prevention. Since both software and firmware are never free of bugs, these preventions will fail at some point. The purpose of HSM-based solutions is to add additional lines of defense to protect sensitive data and enforce or report on platform integrity, which is often referred to as "Security in Depth". HSMs can provide the ability to create and securely store cryptographic materials such as keys and in some cases also support a secure execution environment for

cryptographic operations. This enables an application to encrypt, decrypt, or sign data without ever having to export the keys from the HSM.

Furthermore, HSM-based solutions provide the means to recover from attack scenarios by providing Roots of Trust of known integrity that can be used to verify the correctness of an ECUs configuration after it has been reset/reflashed. Many technologies for HSMs have been developed in the past, including different schemes aimed at the automotive scenarios described in Section 2. The main drawbacks of HSMs are their overall costs and resource requirements. Because of this, projects like EVITA have designed tailored HSMs to fit the needs of specific scenarios.

In the EVITA deliverable 3.2[14], the authors mention and compare different architecture alternatives on where to place an HSM. The HSM can be placed on a separate chip from the CPU. This solution is more expensive with respect to cost and required size on the system board. This solution also introduces security issues as it allows tampering and eavesdropping on the connection between CPU and HSM. Another possibility is to place HSM in the same chip as the CPU. This approach is more economical than the separate chip architecture. The HSM can be implemented using a finite state machine or using an additional programmable secure core. Using a finite state machine is less flexible as it does not allow response to possible future attacks on security. The finite state machine approach does not allow the integration of new cryptographic algorithms and is therefore only appropriate for applications with high security requirements but a relatively short lifetime. Another disadvantage is that the implementation of asymmetric cryptography requires a lot of memory space. Using a programmable secure core may be more expensive than a finite state machine but is still more economic than the separate chip architecture and also more flexible because it also allows the integration of new cryptographic algorithms. This is also the approach recommended by the EVITA project.

The SHE HSM allows implementation on the same chip as the CPU in as a finite state machine or as an additional CPU core [12]. The Trusted Platform Module (TPM) is another such kind of HSM that provides all of the features mentioned above as well as many more. The TPM can be implemented both on a dedicated chip as well as using firmware in a protected environment in a System-on-Chip (SoC). The architecture of TPMs is described in more detail in section 4.4. The following Section 4.1 introduces the TPM as well as its profiles aimed at different usage scenarios. Section 4.2 then focuses on the Automotive Thin Profile and the benefits of using such TPMs in vehicles.

4.1 TPM 2.0 and Profiles

The TPM is specified by the Trusted Computing Group (TCG)[3] and is currently available in version 2.0 [17]. TPM 2.0 is a complete rewrite of the previous TPM 1.2 [26]. Many of the new features are not compatible with the data types and functional layout of TPM 1.2. TPM 2.0 still provides the three Roots of Trust:

[3] http://www.trustedcomputinggroup.org/

Storage, Reporting, and Measurement. These allow the provisioning of reliable identities, secure storage of secrets, and reporting of software integrity.

One of the big improvements in TPM 2.0 over TPM 1.2 is the cryptographic agility. TPM 1.2 only supported RSA $\leq$ 2048 bit and SHA1. TPM 2.0 supports RSA 2048/3072, ECC, and SHA-2 up to SHA512. Unlike TPM 1.2, TPM 2.0 also supports symmetric ciphers such as AES and HMAC, which makes it possible to use encryption in a much more efficient manner. There is also support for all cryptographic algorithms from TCG's Algorithm Registry [27].

One of the most potent features in TPM 2.0 is the Enhanced Authorization. Instead of being restricted to using only very limited authorization mechanisms that allowed only single factor authorization, it is now possible to form policy statements using newly introduced policy commands. These allow for more flexibility: A policy may either require multiple factors or have several different paths allowed for policy fulfillment. The non-volatile memory capabilities were also enhanced. The NV memory may now also be used as (monotonic) counters, bitmaps, and extended PCRs. Challenger and Arthur thoroughly describe these changes as well as many more enhancements in [28].

As previously mentioned, the TCG has developed profiles that are based on the full TPM 2.0 library specification and are specifically designed for different scenarios. These profiles require only a minimum selection of functionality that must be supported. The Automotive Thin Profile [19] was specifically designed with a resource constrained automotive context in mind. The next section further describes the benefits of using a TPM 2.0 in vehicles.

4.2 Benefits of using TPM / TSS 2.0 in Vehicles

Using the new TPM 2.0 as an HSM in vehicles has several benefits. The TPM 2.0 specification is considered the most elaborated resource among HSM specifications. It is a widely accepted open standard and many vendors provide TPM / TSS solutions resulting in wide availability and competitive pricing. With TPM 2.0, the specification has become flexible enough to allow adaptation to different scenarios and is especially suitable for automotive applications through its cryptographic agility, enhanced authorization, and modular library nature.

In vehicles there are different types of ECUs and it is usually too costly (and often not possible) to integrate a high-end HSM into each ECU. The EVITA project for example defines three classes of ECUs, ranging from Head Units with smartphone-like computing power, through engine control units with moderate computing power, to sensors and simple actuators with scarce resources. For each of these classes different HSMs (full, medium, small) have been defined in EVITA [14]. The TCG already released a specification for the Automotive Thin TPM profile. Since the TCG only releases profiles based on the TPM 2.0 library specification, one standard TPM and Trusted Software Stack 2.0 (TSS) [21] can be used as a basis for all ECUs in a vehicle to simplify the development process and reducing costs.

Another important aspect of the flexibility is the cryptographic agility of the TPM and TSS 2.0. Since vehicle platforms have a long product life cycle,

usually at least 15 years, advances in cryptographic analysis can render certain cryptographic algorithms or key lengths insecure. Using TPM 2.0, insecure or outdated algorithms can be exchanged easily, without affecting the overall platform.

Furthermore, the TPM 2.0 provides several features required in the automotive context. This includes for example monotonic counters, which can be used to prevent firmware downgrades in remote firmware update protocols.

4.3 Automotive Thin Profile

The Automotive Thin Profile defines a subset of the TPM 2.0 library specification [17] that is suitable for the automotive scenario, while giving the design freedom to develop and deploy cost effective and simplified solutions. The TCG has described scenarios for proving the identity of an ECU, reporting used software, and remote software updates. The Automotive Thin Profile is designed to be deployable on all ECUs in a vehicle, with the main target being the smallest ECUs. These ECUs have scarce resources in terms of speed, ROM, RAM and Non-Volatile (NV) memory, as well as highly sophisticated power management. These ECUs do not typically run a conventional OS.

The TCG assumes that a vehicle and its TPMs have a life cycle of 20 years and that ECUSs are generally configured in three or more separated and isolated networks. In the center of these networks is a Security Security Gateway, that interconnects the separate networks and communicates via a Head Unit or other Remote Function Call interface with the OEM or government Remote Center. The Remote Center is usually some form of safety and maintenance center. The Security Gateway always serves as a relay between the actual ECU and the Remote Center.

By assuming the above described system model, the TCG proposes two types of TPMs for vehicles: An Automotive Rich TPM, installed in the Security Gateway as it usually is a more powerful ECU, and the Automotive Thin TPM, which is deployed in all other ECUs. Since there is not yet a specification for an Automotive Rich Profile, it is presumed to support similar commands to the PC Client Profile. The Rich TPM should be able to manage several Thin TPMs at once and support local certificate store and management services for ECUs with Thin TPMs in case of lost communications.

The main usage scenario of Thin TPMs is to support ECUs with integrity and attestation requirements for remote maintenance services. Thin TPMs enable storage of ECU firmware measurements, the creation of integrity digests, and the creation of signatures for said digests. Depending on the implementation of the Thin TPM, it may further support ECUs to verify signatures of received firmware updates and patches, and can be leveraged to acknowledge and confirm successful update installation to the Remote Center [19]. This enables OEMs to audit and prove the software state of vehicles in case of liability claims.

In comparison to the Automotive Thin Profile, SHE only supports very basic security functionality such as symmetric cryptography for encryption, decryption, and message authentication, secure storage of keys, and secure boot.

The Automotive Thin Profile also supports asymmetric cryptography. The EVITA HSMs aim at providing authenticity and integrity of the in-vehicle communication as well as the providing a secure execution environment. It should be noted that the EVITA light HSM is very similar to SHE. A detailed comparison on the supported algorithms of EVITA and SHE is given in [14].

In addition to a reduced set of commands, the Automotive Thin Profile also features a subset of algorithms. The profile only requires either RSA with 2048 bits or ECC P256, at least one symmetric algorithm like AES 128 in CFB mode, HMAC, and SHA256. All other algorithms supported by a full TPM are optional for this profile.

Because of the reduced set of commands, the Automotive Thin Profile does not support some of the functionality defined by the full TPM 2.0. For example, if Automotive Thin TPMs only support the minimum mandatory functionalities, then they lack the ability to use monotonic counters in the NV RAM, they are not able to verify signatures, and they are not able to use enhanced authorization, which is a prominent feature of the TPM 2.0. The mandatory commands do not include Policy or NV RAM commands.

4.4 TPM Architectures

In addition to new features, the TPM 2.0 standard is also a lot more flexible in terms of its implementation. It is a common misconception that a TPM must be implemented as a discrete chip. In [29] multiple other options are listed, including firmware based implementations in a discrete chip, to firmware or virtual implementations on a shared chip supported by a hardware isolated execution environment. In [29], all of these types of implementations are referred to as "TPM Mobile". Note for example the architecture shown in Figure 1, which shows a virtual implementation of a TPM using a hypervisor. This approach starts with the hardware verifying and booting a "secure boot" capable bootloader. This bootloader in turn loads and verifies a hypervisor, which hosts two (or more if desired) separate environments: a secure environment with a virtual TPM implementation (possibly hosted by a small secure OS), and the regular application OS.

There are several important aspects in a firmware or virtual based approach, which serve to ensure a chain of trust between the boot stages. At the bottom is the Boot ROM of the hardware, which must be immutable (e. g., masked ROM on the CPU chip). It initializes the required hardware to load and verify the next boot stage. The verification may for example be based on keying material stored in one-time programmable memory on the chip. The next stage is a secure bootloader, which may just be a richer and updatable version of the first stage. Due to this verification a transitive chain of trust is created from the boot ROM. This feature is commonly known as "secure boot" or "verified boot".

Another important aspect of a firmware or virtual TPM loaded during secure boot is an effective isolation between the firmware or virtual TPM and the application execution environment. Figure 1 illustrates an approach using a hypervisor, however other techniques may be used. Some platforms feature a

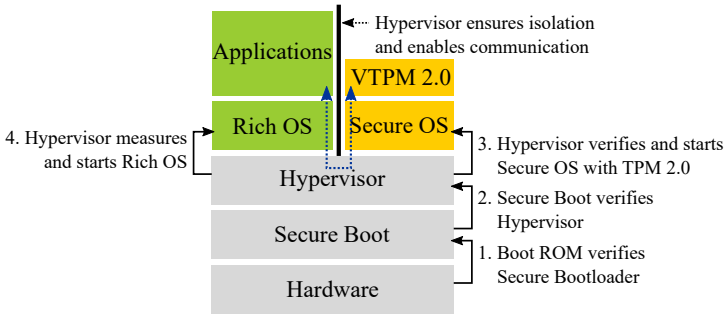

Figure 1. Architectural overview of a virtual TPM and its boot sequence

hardware based isolation mechanism, which is capable of isolating the address space of a firmware TPM from the regular applications. One example is the "TrustZone" technology by ARM [30]. In this case, a higher privileged "monitor mode" with its own set of registers is introduced. The CPU is booted in this mode and a lightweight OS is booted, which manages a small set of trusted applications. After this secure zone is set up, a regular rich OS can be booted in the normal mode. Hardware based isolation guarantees that memory pages reserved for the secure zone are not accessible to the regular rich OS. Some implementations may even support peripherals only accessible to the secure OS and its trusted applications. Another technology with similar features to TrustZone is Intel's Platform Trust Technology (PTT). In both cases a very cost-effective implementation approach for a TPM 2.0 can be offered.

In the context of automotive TPM implementations a commonly raised concern about TPMs and other HSM is the additional cost of implementations. Given the history of the SHE implementation [12], it is known that several SoC implementations exist. However, as noted above, a TPM can be implemented e.g. in a TrustZone version in scenarios where cost and size are a major concern. Note however that some TrustZone implementations do not provide all necessary functionality to implement TPMs, such as roll-back protection. Implementation considerations therefore need to be handled on a case-by-case basis considering the aspects of cost, size, execution speed, implemented TPM functions, and required protection level.

5 Resource Consumption Estimates

Although an automated analysis of the resource consumption of any program with perfect results it not possible due to Rice's Theorem, a limited analysis is possible under certain assumptions. In [31] an extension to the GCC compiler is described which aims to help static analysis of a program's stack consumption. The extension outputs a diagnostics file listing the stack consumption properties of each compiled function, including the stack frame size, which is flagged as either static, an upper bound, or dynamic. To determine an estimate of the

stack consumption for this paper, we combined this information with a callgraph generated with the CFlow [32] program. As all functions of the implementation have bounded stack frame sizes, and an acyclic callgraph, a reliable estimate for the maximum stack depth can be determined. Note that this measurement does not include the stack consumption of the software dependencies, which are cryptographic math libraries and the standard C library. In case of the standard C library, the figure would be highly platform dependent, the cryptographic math libraries' consumption varies depending on the used cryptographic primitives and hardware acceleration.

	Automotive Thin	Full TPM	Reduced
Stack	2140B	2256B	5,14%
Global State	5959B	9415B	36,71%
Codesize	119KB	190KB	37,37%

Table 1. Required memory resources of a simulated TPM 2.0 implementation

The remaining figures after the stack consumption are the code size and the size of the TPM's state, both of which can be easily measured with common development software. The results of the measurement performed on a TPM simulator implementation are shown in Table 1. The used platform is a Cortex-A7 chipset, a 32-Bit CPU by ARM. The performed stack consumption measurement places an estimate at over 2KB, with no significant difference between a full TPM implementation and a thin TPM. The bulk of the savings are measurable in the size of the code and global state, both of which are reduced by about 37%. Note that these figures don't include various buffers (e. g., transmission buffers) or the nonvolatile storage of the TPM, which may vary depending on the implementation. The reason behind these savings are not just the removed commands themselves, but also the removed transitive requirements, such as (un-)marshaling functions. The reduced set of mandatory supported cryptographic algorithms in the Automotive Thin Profile further reduces the amount of necessary code and definitions of static buffers.

A firmware or virtual implementation of a full TPM is already small enough, when the separation techniques mentioned in Section 4.4 are used, making the Automotive Thin Profile an unnecessary restriction for firmware or virtual TPMs. The impact of the Automotive Thin Profile on hardware implementations (e.g. a discrete chip or integrated into an SoC) is hard to predict, but similar savings in size are likely. Note that the measured TPM simulator is not tailored to a specific platform and as such is not designed to be particularly small.Hence, the Automotive Thin Profile may still be an effective cost saving solution for budget sensitive hardware implementations. The impact on the resource consumption for the application side is not necessarily affected by a thin TPM, aside from the reduced feature set. The functions common between a full TPM and a thin TPM

remain the same and thus exhibit the same memory consumption. The code and data size of TSS 2.0 libraries is noticeably reduced by about 15%, which is however only relevant for dynamically linked binaries, as a static linking would automatically exclude any unused functionality. Aside from this, the TSS 2.0 adds at most 336 bytes of stack consumption to an application.

6 Solving SOTA

We already described the limitations of the Automotive Thin Profile in Section 4 with its minimal required set of functions. These do not support monotonic counters nor are they capable of verifying a signature [19].

Thus the Automotive Thin Profile cannot validate a signature and also does not support recognition of downgrade attacks with the use of counters and policies. Note however, that any TPM implementation can also support functions beyond those required by a particular profile.

The specification of Automotive Thin Profile already introduced the idea of using enhanced TPMs in a small number of ECUs that provide the necessary functions to establish a secure connection between the Remote Center server (here also referred to as backend) and the car. This architecture is visualized in a simplified manner in Figure 2. The figure shows a Security Gateway with a TPM that features functions beyond the Automotive Thin Profile to communicate with the backend via GPRS, UMTS, or LTE. This enhanced TPM is called rich TPM. The Security Gateway ECU also communicates with other ECUs in the car which only have thin TPMs providing the mandatory functions of the Automotive Thin Profile.

6.1 Authenticated Update

Based on this architectural idea an authenticated software update can be performed. Figure 3 illustrates a process which ensures authenticity of a firmware update in a system with intact integrity. [19] describes how the integrity of automotive systems can be ensured with the TPM quote function.

After the backend ensures the integrity of the vehicle it transfers a new firmware update to the vehicle. The following steps are described by Figure 3. First the update is received by the Security Gateway ECU that has a rich TPM. The Security Gateway ECU validates the authenticity of the update with the use of TPM2_Hash and TPM2_VerifySignature. If this firmware update is valid the Security Gateway ECU can proceed, otherwise it waits for a new update attempt. In case the firmware is meant for the Security Gateway ECU itself, it performs the update.

Otherwise the update needs to be transfered to the target ECU. The minimal requirements of a thin TPM allow the function TPM2_StartAuthSession. Both the Security Gateway and target ECUs use TPM2_StartAuthSession to establish an authenticated channel between each other over which the update is transfered. Then the target ECU performs the update.

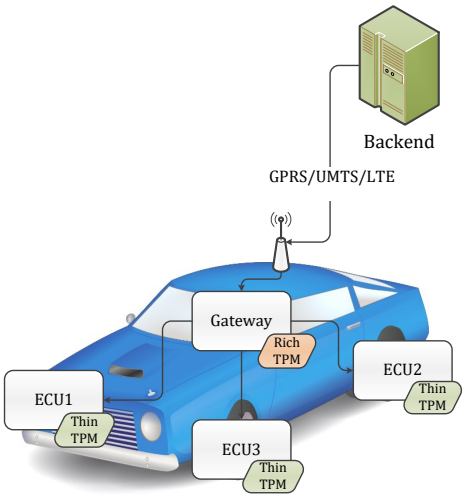

Figure 2. Actors in Vehicle Over the Air Update

After every update the system integrity is controlled by the Security Gateway and the backend. In case the system integrity is proven, all updates were successful. Otherwise, recovering an old stable state could be an option as well as investigating the potential attack.

6.2 Downgrade Protection

The procedure described in Section 6.1 does not protect against downgrade attacks. An attacker which monitors updates could present an old update, for example in order to revert a security fix.

The TPM 2.0 Library specification introduces the scheme of Enhanced Authorization. It allows the arbitrary combination of different policy elements to logical expressions via “and” and “or”. From these statements, TPM2_PolicyNV can be used to compare a TPM-internal monotonic counter value against a reference value, preventing a downgrade of firmwares to an older version. This would be important for example when older updates with known vulnerabilities could be replayed as first step in an attack vector.

6.3 Confidentiality

A developer can ensure confidentiality on the air gap by introducing a Transport Layer Security(TLS) [33] like protocol. The rich TPM can support a key exchange and support client authentication to the backend server as well as

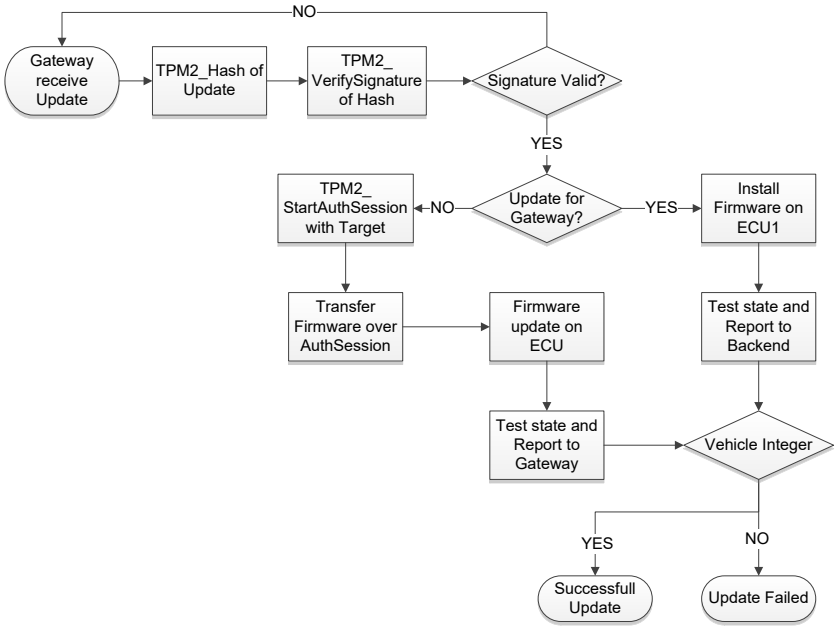

Figure 3. Authenticated Update Process

validating the server certificate. A TPM can support a key-exchange using TPM2_ECDH_KeyGen for example.

6.4 Auditing and Liability

The functionality of Software Over-The-Air updates has new requirements in the area of legal liability. An automotive OEM must have the ability to proof that an update has been installed and what version was running on which ECU after a vehicle completes a SOTA update. A TPM-based process can achieve this property by performing a remote attestation of the vehicle's integrity after an update process has been performed. Depending on the detailed requirements, this can either mean an attestation of a Security Gateway ECU inside the vehicle that in turn collects attestations of all ECUs or by performing end-to-end attestations all the way between the backend and target ECUs.

7 Conclusion and Outlook

The security of automotive systems has been gaining major publicity in the recent past. Regarding automotive requirements, we have shown the necessity

of Hardware Security Modules. Implementations based on Trusted Platform Module specifications as defined by the Trusted Computing Group for Desktops, Mobile Devices, and Automotive systems provide a good way to satisfy these requirements.

This paper presents the first work on code and RAM-size consumptions for TPM-based HSM-realizations. The savings between a full-blown TPM and specialized automotive TPMs in this regard range between 25% and 50%, depending on the choice of algorithms, functions, and leveraging of consecutive optimization potentials.

This work will be continued to implement complete end-to-end scenarios in the future. This will allow more concrete estimations using realistic deployment scenario hardware and functionality.

References

1. "EVITA project." http://www.evita-project.org/.
2. "PRESERVE project." http://www.preserve-project.eu/.
3. T. Eisenbarth, T. Kasper, A. Moradi, C. Paar, M. Salmasizadeh, and M. T. M. Shalmani, "On the power of power analysis in the real world: A complete break of the keeloq code hopping scheme," in *Advances in Cryptology – CRYPTO 2008*, pp. 203 – 220, Springer, 2008.
4. R. Verdult, F. D. Garcia, and J. Balasch, "Gone in 360 Seconds: Hijacking with Hitag2," in *21st USENIX Security Symposium*, pp. 237 – 252, USENIX Association, 2012.
5. R. Bavyar and R. Mohanamurali, "Next generation auto theft prevention and tracking system for land vehicles," in *2014 International Conference on Information Communication and Embedded Systems (ICICES)*, IEEE, 2014.
6. I. Rouf, R. Miller, H. Mustafa, T. Taylor, S. Oh, W. Xu, M. Gruteser, W. Trappe, and I. Seskar, "Security and Privacy Vulnerabilities of In-car Wireless Networks: A Tire Pressure Monitoring System Case Study," in *19th USENIX Security Symposium*, USENIX Association, 2010.
7. K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental Security Analysis of a Modern Automobile," *IEEE Symposium on Security and Privacy*, pp. 447 – 462, 2010.
8. S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive Experimental Analyses of Automotive Attack Surfaces," in *20th USENIX Security Symposium*, USENIX Association, 2011.
9. C. Miller and C. Valasek, "A Survey of Remote Automotive Attack Surfaces," in *Blackhat*, 2014. https://www.blackhat.com/us-14/briefings.html#a-survey-of-remote-automotive-attack-surfaces.
10. A. Greenberg, C. Miller, and C. Valasek, "Hackers Remotely Kill a Jeep on the Highway—With Me in It," *Wired*, 2015. http://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/.
11. T. Becsi, S. Aradi, and P. Gaspar, "Security issues and vulnerabilities in connected car systems," in *2015 International Conference on Models and Technologies for Intelligent Transportation Systems (MT-ITS)*, pp. 477 – 482, IEEE, 2015.

12. Hersteller Initiative Software (HIS) AK Security, *SHE–Secure Hardware Extension*, version 1.1 ed., 2009.
13. O. Henniger, L. Apvrille, A. Fuchs, Y. Roudier, A. Ruddle, and B. Weyl, "Security requirements for automotive on-board networks," in *9th International Conference on Intelligent Transport Systems Telecommunications (ITST 2009)*, pp. 641–646, IEEE, 2009.
14. B. Weyl, M. Wolf, F. Zweers, T. Gendrullis, M. S. Idrees, Y. Roudier, H. Schweppe, H. Platzdasch, R. El Khayari, O. Henniger, *et al.*, "Secure on-board architecture specification," *Evita Deliverable D3.2*, vol. 3, p. 2, 2010.
15. M. Klimke, "Benefits and Values of the Trusted Platform Module," *escar*, 2014.
16. D. Bhattacharya, S. S. Neelakantan, J. Shokrollahi, and H. Löhr, "Bootstrapping trust in automotive networks with different types of ECUs," in *escar*, 2014.
17. Trusted Computing Group, *Trusted Platform Module Library Specification*, Family 2.0, Level 00, Revision 01.16 ed., October 2014.
18. Trusted Computing Group, "Trusted Computing Group TPM 2.0 Library Specification Approved as an ISO/IEC International Standard," June 2015.
19. Trusted Computing Group, *TCG TPM 2.0 Library Profile for Automotive-Thin*, Version 1.0 ed., March 2015.
20. Trusted Computing Group, *TCG TPM 2.0 Mobile Common Profile*, Family 2.0, Revision 00.31 ed., December 2015.
21. Trusted Computing Group, *TSS System Level API and TPM Command Transmission Interface Specification*, Family 2.0, Revision 01.00 ed., January 2015.
22. Trusted Computing Group, *TCG Guidance for Securing IoT*, Version 1.0, Revision 14 ed., April 2015.
23. I. Hossain and S. M. Mahmud, "Analysis of a secure software upload technique in advanced vehicles using wireless links," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 1010 – 1015, IEEE, 2007.
24. D. K. Nilsson and U. E. Larson, "Secure firmware updates over the air in intelligent vehicles," in *IEEE International Conference on Communications Workshops (ICC)*, pp. 380 – 384, IEEE, 2008.
25. H. Mansor, K. Markantonakis, R. N. Akram, and K. Mayes, "Don't Brick Your Car: Firmware Confidentiality and Rollback for Vehicles," in *10th International Conference on Availability, Reliability and Security (ARES)*, pp. 139 – 148, IEEE, 2015.
26. Trusted Computing Group, *TPM Main Specification*, Level 2 Version 1.2, Revision 116 ed., March 2011.
27. Trusted Computing Group, *TCG Algorithm Registry*, revision 01.22 ed., February 2015.
28. David Challener and Will Arthur, *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. Apress, 2015.
29. Trusted Computing Group, *TPM 2.0 Mobile Reference Architecture*, family 2.0, level 00, revision 142 ed., December 2014.
30. ARM, "Trustzone." `http://www.arm.com/products/processors/technologies/trustzone/`, 2015.
31. E. Botcazou, C. Comar, and O. Hainque, "Compile-time stack requirements analysis with GCC," in *GCC Developers Summit*, p. 93, 2005.
32. S. Poznyakoff, "GNU cflow." `https://www.gnu.org/software/cflow/`, 2011.
33. T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," 2008. `https://www.ietf.org/rfc/rfc5246.txt`.

All links were last followed on May 13, 2016.