Design and Implementation of an Architecture for Vehicular Software Protection

Michael Scheibel
Sirrix AG
Bochum, Germany
m.scheibel@sirrix.com

Christian Stüble
Sirrix AG
Bochum, Germany
c.stueble@sirrix.com

Marko Wolf
escrypt GmbH
Bochum, Germany
mwolf@escrypt.com

Abstract

Embedded software is becoming the most innovative and valuable part of current and future automotive vehicles. Software updates after delivery of the vehicle to patch or enhance existing software, or delivery of digital content for in-car infotainment has become a common issue also within the automotive area. In order to prevent misuse that could lead to real manufacturer's loss or damage or even affect the operational safety of a vehicle, embedded software and digital content need to be protected. However, most deployed software distribution mechanisms rely on trusted networks despite their inherent weaknesses in flexibility and reliability and their enormous organizational complexity, which in spite of everything, provide only low resistance against attacks of skilled adversaries.

In this paper we present the design of a security architecture for software protection that is capable of binding arbitrary digital content to a certain vehicular hard- and software configuration. The underlying protocol in particular ensures that the content can never be accessed unauthorized even though it is being distributed using a fully untrusted infrastructure. We show how to implement this architecture efficiently by means of virtualization technology, an (open source) security kernel, trusted computing functionality, and a legacy operating system (currently Linux).

Keywords: automotive, software protection, trusted computing, authenticated booting, digital rights management.

1 Motivation

Embedded software is becoming the most innovative and valuable part of current and future automotive vehicles. Due to the ubiquitous use of flashable Electronic Control Units (ECUs)[1], software updates take place also after delivery of the vehicle. Reasons are warranty-based updates in recalls, allowing to correct defective software, but also value-adding updates, such as software-based features sold in the after-market [16]. In addition to executable program code, delivery of digital content to vehicles has become important; for example, routing and location based information for navigation systems as well as multimedia content for in-car entertainment.

Intellectual property and in particular manufacturer's know-how implied in software and digital content needs to be protected in order to prevent misuse such as unauthorized feature activation, unauthorized updates or modifications, illegal copies, and know-how theft. Such attacks could lead to real loss or damage such as undermined business models, false warranty claims, and damage the manufacturer's public image. Since software already controls very critical components such as airbags, anti-lock brakes, or the engine control, a wireless attacker or an (intentional or unintentional) unauthorized software update may even affect the operational safety of a vehicle. While considering malicious attacker scenarios so far has been mostly only a subject of matter for a few single components such as immobilizer or tachograph [20], information security today becomes a crucial design and implementation issue within automotive electronic development.

So far, most deployed software distribution mechanisms rely on trusted networks despite the inherent weaknesses in flexibility and reliability and their enormous organizational complexity. Therefore, actual vehicle ECUs provide only low resistance to attacks of skilled adversaries. With only minimal and easily obtainable equipment, an adversary can easily read out and in-

[1] A flashable ECU is a microcontroller capable of reprogramming its memory for application programs and data based on so-called flash memory technology [10].

stall any software at will. Considering this, the awareness of the need for efficient software protection mechanisms grows [1, 10], and implementations start to appear in the form of digitally signed software updates [8]. As already known from the PC area, solely software based mechanisms cannot provide reliable protection. An adversary can for instance read cryptographic secrets from unprotected memory components, thus being able to break most software based protection measures. However, software protection based on tamper-resistant components, and cryptographic techniques are still uncommon in automotive applications. Since the design of fully secure ECUs is considered to be too costly or even infeasible in practice [5], it is highly desirable to design security architectures in which only few components are secure and fully trusted, while others can be implemented with common off-the-shelf components.

1.1 Our Contribution & Outline

In this paper we present the design and implementation of a security architecture for software protection that is capable of binding arbitrary digital content to a certain vehicular hard- and software configuration. The underlying protocol in particular ensures that the content can never be accessed unauthorized even though it is being distributed using a fully untrusted infrastructure.

Therefore we employ Trusted Computing (TC) technology, more precisely a Trusted Platform Module (TPM)[2], and our Turaya security kernel. Moreover, we improve security *and* safety of the vehicle, by employing virtualization technology (VT), i.e., running all security software components on a small security kernel in parallel to - but strongly isolated from - legacy operating system(s) that allow the reuse of existing applications.

Our paper is organized as follows. After analyzing previous work in the area of automotive software protection in Section 1.2, we summarize functionalities, application scenarios and provide security properties of our approach in Section 2. Section 3 introduces our Turaya security kernel that implements elementary security properties followed by Section 4 that shortly illustrates the idea of Trusted Computing. In Section 5 we give a detailed description of our protocol design and provide some annotations to our first prototype implementation in Section 6 based on our Turaya security kernel, virtualization technology and Trusted Computing.

[2]A Trusted Platform Module is a cheap ("one dollar") smartcard-like security chip that strengthens a computing platform against typical attacks.

1.2 Related Work

Various authors have identified the need for information security and software protection in vehicles [6, 9, 24, 27]. Ehlers presents an approach for vehicle system integrity by binding software IDs based on digital signatures or hashes to fixed hardware IDs of the ECUs, but does not address malicious actions at all [11]. Seshadri et al. [28] introduce a software-based attestation mechanism for verifying embedded software at run-time, but cannot prevent subsequent manipulations and exclude all types of hardware attacks. Adelsbach et al. introduce a requirement model and a secure installation protocol for software installation in embedded systems [1], whereas Fibikova describes several (access control) software countermeasures to face current automotive threats [13]. However, both do not address hardware requirements. First virtualization approaches [12, 17] for automotive operating systems based on microkernels[3] mainly have safety in mind and for instance do not address how to protect cryptographic secrets.

Nevertheless, the need for hardware-based security measures in embedded systems design has already been identified [18, 25, 32]. Moreover, there exist a few hardware-based approaches to include strong security also in embedded systems [4, 7, 22]. However, all are proprietary solutions and none of them addresses the particular requirements and constraints within the automotive context such as the high cost pressure on automotive ECUs, their low computing power, limited storage, and sporadic network connectivity.

2 The Goal

The proposed protocol allows a content provider to "bind" arbitrary software and digital content to a "certain" vehicle hard- and software configuration. In other words, a content provider can reliably ensure that his bound and encrypted content can be decrypted and accessed only by a previously authorized vehicle configuration while being distributed using a fully untrusted infrastructure. In particular, bound content can be decrypted and accessed only by a previously autho-

[3]A microkernel is a minimalized operating system kernel that provides only essential services such as logical address spaces and inter-process communication (IPC). Processes on top of the microkernel run in their own address space and are therefore strongly isolated from each other.

rized hard- and software configuration of a vehicle.

The given functionality can be used to limit the usage of a software update or digital content to a certain vehicle brand, a certain vehicle type or even to a single car, thus preventing for instance unauthorized feature activation, unauthorized updates or unauthorized modifications and preventing unauthorized copies. Moreover, since decryption of bound content is possible only on previously defined hard- and software configurations, a content provider can enforce the usage of trustworthy platform configurations that implement know-how theft prevention and/or Digital Rights Management (DRM) capabilities so that decrypted content cannot leak into untrusted software.

Our software protection prototype is realized on top of the Turaya security kernel suited for embedded devices based for instance on ARM or MIPS architectures. We assume the security kernel to be running at least on the central gateway or the so-called head unit. As the current development moves from many small, separate ECUs to a few larger ECUs that combine the functionality of several ECUs, we suggest to also run the security kernel on these larger ECUs.

The security kernel basically uses two important mechanisms, namely virtualization technology (VT) and Trusted Computing (TC). Virtualization technology – well known in server environments – employs a small OS kernel (hypervisor) which allows to run multiple, full-fledged operating systems on one host computer at the same time. Thus, VT enables the efficient reuse of existing applications, while running even different OS versions strongly isolated on a single computing platform in parallel. Moreover, VT prevents that the state of one OS or isolated application could affect the state of another, thus security vulnerabilities and bugs of, e.g., a legacy operating system, cannot affect security-critical components.

The Turaya security kernel actually uses a lightweight virtualization technique called paravirtualization that is able to outperform even native software configurations [14, 23]. TC technology on the other hand, enables hardware-based software protection relying on a standardized, tamper-resistant security chip tightly bound to the corresponding computing platform. Thus, TC as a solid root of trust enables several security mechanisms such as protected key storage, protected cryptographic operations, remote attestation or authenticated boot (cf. Section 4). However, the Turaya security kernel is needed to run at the client platform only. There it has to prevent circumvention of existing security policies by (malicious) software. In particular, our proposed software protection protocol requires no changes of the existing server environment at the (trusted) content provider side.

3 Turaya Security Kernel

As depicted in Figure 1, the Turaya security kernel is a small software layer that provides an abstract interface to the hardware resources, guarantees strong isolation of applications and implements elementary security properties built on a hardware layer that is enhanced by Trusted Computing technology. Existing operating systems and applications on top of the Turaya security kernel are running in parallel to - but strongly isolated from - security-critical applications. When focussing DRM systems, the isolation feature of the security kernel can be used to prevent unauthorized access to (decrypted) content. For safety purposes, the isolation feature prevents that the state of one legacy OS or isolated application can affect the state of another. The security kernel is capable of integrating virtualization software such as microkernels [21, 19], and is prepared to take use of emerging hardware virtualization technologies [2, 15]. Additionally, it integrates security-critical services such as a secure user interface, persistent storage, and authenticated booting.

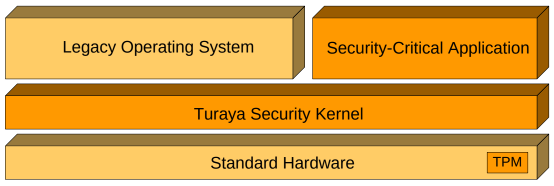


Figure 1: Turaya architecture overview.

The security kernel can be logically divided into a *hypervisor layer* and a *trusted software layer* (cf. Figure 2).

The main task of the hypervisor layer is to provide an abstract interface of the underlying hardware resources like interrupts, memory and hardware devices[4]. Moreover, this layer allows to share these resources and realizes access control enforcement on the object types known to this layer. Currently we are using a microkernel as the base of the hypervisor layer.

The trusted software layer builds on the hypervisor layer and realizes security-critical services

[4]Device drivers that are able to directly access the main memory (DMA-enabled drivers) must be outsourced from the legacy operating system for security reasons.

needed to build secure computing platforms. It consists of a set of isolated security components that - due to its modular concept - can be accomplished exactly according to the designated task. In the following we briefly describe some elementary components of the trusted software layer.

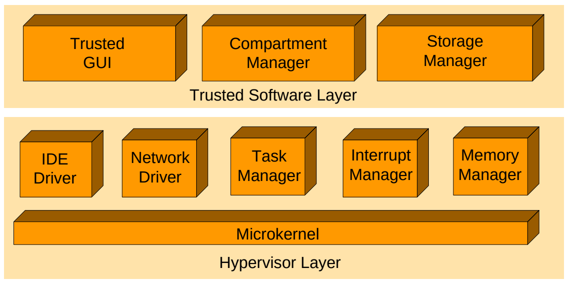


Figure 2: Turaya security kernel.

The *Trusted GUI* controls the graphic adapter and the corresponding input devices to establish a trusted path between the user and an application. The Trusted GUI labels application windows with unique application names. Moreover, the Trusted GUI enforces a strong isolation between applications on the GUI level. Unauthorized applications cannot, for instance, access the graphical output of other applications or fake their interface to look like the usual authentication dialog.

The *Compartment Manager* loads compartments[5] and measures the integrity of compartments. These integrity measurements can then be reported to existing local compartments as well as to remote compartments. In cooperation with Trusted Computing hardware, this functionality constitutes the basis for elaborate Digital Rights Management applications. The advantage of this approach in contrast to other integrity measurement architectures (e.g., [26]) lies in enhanced end-user privacy protection and improved manageability, e.g., of software updates.

Finally, the *Storage Manager* enables other applications to persistently store their local states. It preserves the integrity, confidentiality, and freshness of the managed data such that only the application or the user having produced the data may later re-access it.

4 Trusted Computing

This section introduces those Trusted Computing functionalities which are significant for the design of our software protection protocol, namely the authenticated boot process and some cryptographic building blocks.

4.1 Authenticated Boot

During an authenticated boot process, each part of code which is executed is “measured” before execution, e.g., by calculating a cryptographic hash of the code. Trusted Computing hardware is responsible for the secure storage and provision of the measurement results. Upon completion of an authenticated boot process, these measurement results reflect the configuration of the currently running hardware and software environment. Trusted Computing technology however remains passive and does explicitly not prevent a certain computing environment from being compromised during runtime. However, integrated cryptographic mechanisms enable the platform to verifiably report their actual state to local and to external parties.

4.2 Secured Cryptography

Trusted Computing hardware implements a set of cryptographic operations to ensure that malicious software cannot compromise cryptographic keys. Usually, key generation and decryption operations are done “on-chip”. Private and secret keys never leave the chip without being encrypted. To perform a decryption operation with a specific key, several types of authorization are possible. A distinctive feature of Trusted Computing hardware is the ability to not only use passwords as authorization but also integrity measurements. That is, only a platform running previously defined software or hardware components is authorized to use a certain key. Moreover, the property that a certain key is “bound” to a platform configuration can be certified by Trusted Computing hardware. This certification includes the integrity measurements which authorize a platform to employ the key. A remote party can verify the certificate and validate the embedded integrity measurement against “known good” configurations before encrypting data with the certified key[6].

4.3 Trusted Platform Module

The base of Trusted Computing technology is the Trusted Platform Module (TPM) that is considered to be a tamper-resistant hardware device similar to

[5] An application or operating system that is logically or even physically isolated from other software components.

[6] Mostly, the data to be encrypted is a secret symmetric key used for the encryption of larger amounts of data.

a smart-card and is assumed to be securely bound to the computing platform. The TPM is primarily used as a root of trust for integrity measurement and reporting (cf. 4.1) and to secure all critical cryptographic operations (cf. 4.2). Current TPMs base on the specification version 1.2 [30] published by the Trusted Computing Group (TCG), successor of the Trusted Computing Platform Alliance (TCPA), an initiative led by AMD, HP, IBM, Infineon, Intel, Lenovo, Microsoft, and Sun. TPMs are available, e.g., from Atmel, Broadcom, Infineon, Sinosun, STMicroelectronics, and Winbond. As depicted in Figure 3, a TPM basically consists of an asymmetric cryptographic engine (RSA), a cryptographic hash function (SHA-1), a true random number generator, a few bytes of volatile memory, some kilobytes non-volatile memory and sensors for tampering detection.

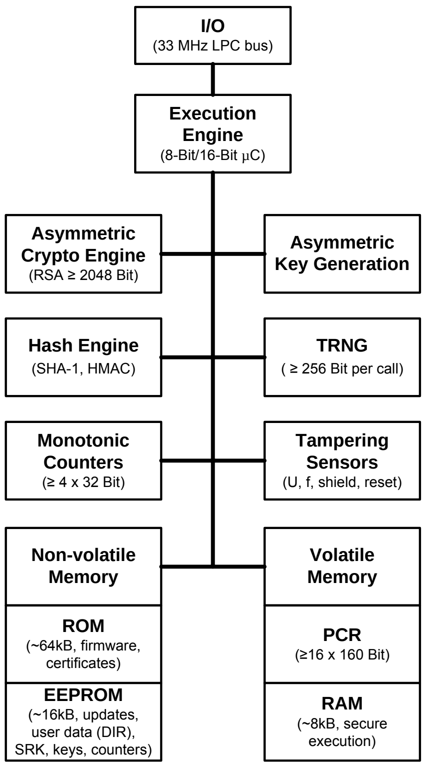


Figure 3: TPM chip version 1.2.

5 Design

This section describes the software protection protocol in detail. This protocol relies on the Trusted Computing functionalities described in the previous section.

5.1 Security Objectives & Assumptions

This section defines the overall security objectives and considers the required assumptions.

Security Objectives: Unauthorized vehicle hardware and/or software configurations must not be able to access content (*content confidentiality*). Moreover, content cannot be modified during transfer process without being at least detected (*content integrity*).

Assumptions: The underlying hardware (e.g., CPU, devices, TPM) is non-malicious and behaves as specified. The content server, i.e., the server that holds the original content and provides the binding of the content to the vehicle certificate, is fully trusted. The vehicle has an TPM chip version 1.1b or higher integrated.

5.2 Protocol Overview

The two parties involved in our protocol are the *user* and the *content provider*.

The *content provider* distributes digital content (ECU software, navigation data, media files, etc.) that will be employed by the *user*. For simplicity, we summarize original equipment manufacturers, authorized maintenance service providers, and infotainment data vendors in the single party that provide digital content. The *user* refers to the person that currently uses the vehicle. As owner and user often coincide and a distinction between the two does not affect our proposed solution, we summarize them with a single party.

As illustrated in Figure 4, the protocol basically consists of four steps. Firstly, the TPM in the car's head unit generates an asymmetric key pair and a certificate stating that the private key is bound to the specific TPM and the car's head unit[7] configuration. Secondly, the certificate and the public key are transferred to the content provider who validates the certificate. This validation implies verifying the signature and checking the configuration against "known good" reference values. Thirdly, on successful validation, the content provider uses the public key to establish a "trusted" channel to the car. This means that content previously encrypted using this public key can be decrypted only if the vehicle provides the "trusted" configuration as stated in the successfully validated certificate. A trusted configuration in this context means that the Turaya security kernel is installed

[7]The head unit integrates all computing subsystems in a car.

and running on the head unit, and that all components (both hardware and software) processing the decrypted content are trusted by the content provider. The decrypted and bound content is finally sent to the vehicle for decryption and usage if (and only if) it provides the stated trusted configuration. Note that steps (1) and (2) can be done already at the end of the manufacturing process by the OEM, thus reducing initial complexity. A new key pair and a corresponding certificate is thereafter only necessary, if one of the crucial and therefore measured hard- or software components has been modified due to a software update or hardware change.

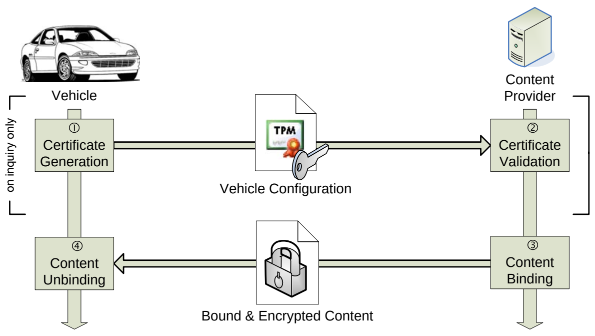


Figure 4: Certificate-based content binding.

5.3 Server Architecture

As mentioned in Section 2, the content provider side requires no changes of their existing server environment (and does not need Trusted Computing technology either) when employing our proposed software protection protocol. Thus they can employ their common computing architecture to accomplish certificate validation and content binding.

5.4 Client Architecture

The software components involved in certificate creation and content unbinding on the client side are depicted in Figure 5. The two components that need to be outsourced from the untrusted legacy OS are the *Trusted Viewer* which decrypts and renders the content and the *Trust Manager* which provides an abstraction of Trusted Computing technology. The Trust Manager offers an interface for key and certificate generation and for decryption (unbinding) with a provided key. Note that only the Trusted Viewer may use this interface and thus access TPM functionality. The isolation feature of the Turaya security kernel, e.g., as provided by a secure GUI, prevents unauthorized accesses from the legacy OS to plain content or the TPM.

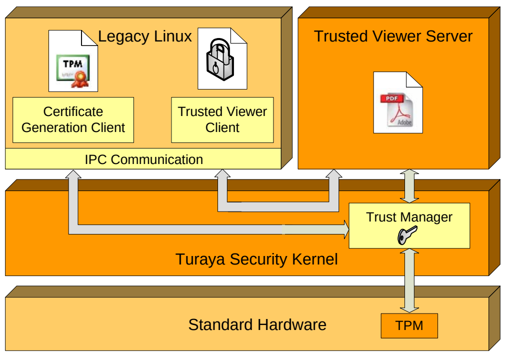


Figure 5: Client architecture.

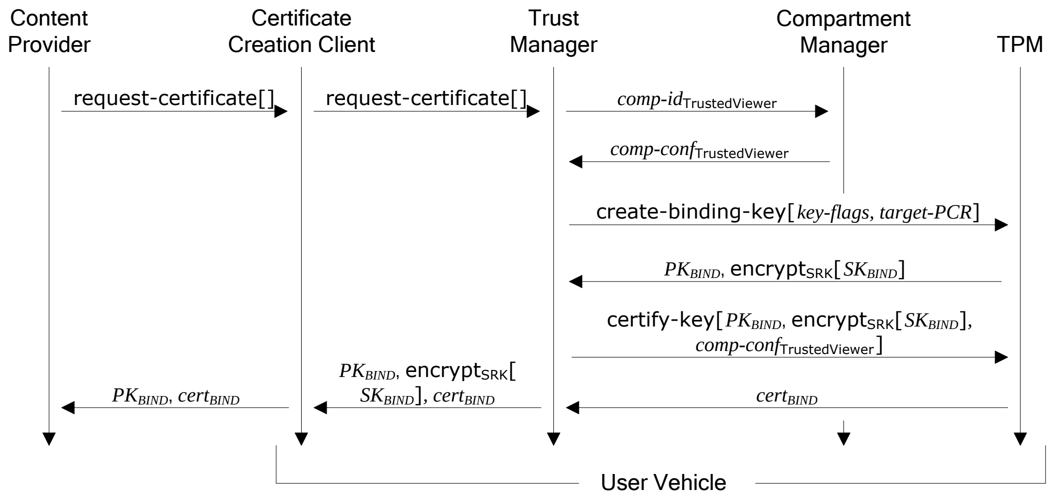
5.5 Protocol Details

The following section gives further details on the protocol steps introduced in Section 5.2.

Certificate Generation: The legacy OS provides a client application (*Certificate Generation Client*) to invoke the Trust Manager to create an asymmetric key pair and a corresponding certificate stating that the private key is bound to the actual vehicle configuration. The actual vehicle configuration has been made available for the Trust Manager by the authenticated booting capability of the head unit (cf. Section 4.1) that measures all crucial computing subsystems in a car. The involved protocol steps are depicted in Figure 6. Finally, the certificate is transferred to the content provider. Since the certificate size is only around 1 kB, even small-bandwidth connections such as GSM can be employed. Note again that a new key pair and thus a new certificate is only necessary if a security-critical hard- or software component has been changed or modified.

Certificate Validation: On the arrival of the certificate, the content provider has to validate the stated values with the given TPM signature. The content provider therefore has the TPM's signature verification key available or has to establish an additional protocol to receive the TPM's signature verification key.[8] As depicted in Table 5.5, the certificate contains the public RSA binding key PK_{BIND} and its cryptographic hash H_{PK}, the value C_{Vehicle} that represents the measured configuration of the Trusted Viewer compartment and

[8]This can be done also anonymously using an anonymous TPM attestation identity key (AIK).



Content Provider
Certificate Creation Client
Trust Manager
Compartment Manager
TPM
request-certificate[]
request-certificate[]
comp-id_TrustedViewer
comp-conf_TrustedViewer
create-binding-key[key-flags, target-PCR]
PK_BIND, encrypt_SRK[SK_BIND]
certify-key[PK_BIND, encrypt_SRK[SK_BIND], comp-conf_TrustedViewer]
PK_BIND, encrypt_SRK[SK_BIND], cert_BIND
PK_BIND, cert_BIND
cert_BIND
User Vehicle

Figure 6: Vehicle certificate generation protocol.

the underlying hardware and software environment as well as the TPM signature S over H_{PK} and C_{Vehicle}. On successful integrity verification, the content provider checks the given vehicle configuration against "known good" and thus trusted reference values.

Public binding key PK_{BIND}
Hash value H_{PK} = SHA-1(PK_{BIND})
Vehicle configuration C_{Vehicle} = ($comp\text{-}conf_{\text{TrustedViewer}}$, $target\text{-}PCR$)
Signature S = $sign_{TPM}$ (H_{PK}, C_{Vehicle})

Table 1: Structure of the vehicle TPM certificate.

Content Binding: On successful certificate validation, the content provider uses the given public key to encrypt and bind contents to the corresponding vehicle configuration. Virtually, the content provider encrypts and binds a secret symmetric key $symm - key$ used for the encryption of the actual content. The bound content now can be transferred to the user using a capable data link, i.e., a GSM connection or offline transfer such as by USB sticks or DVDs.

Content Unbinding: The vehicle's head unit is able to decrypt and access the content if it provides the trusted configuration as stated during certificate generation. As depicted in Figure 7, the legacy OS provides a client application (*Trusted Viewer Client*) to invoke the Trusted Viewer application that runs in parallel to, but isolated from the legacy OS. The Trusted Viewer, in turn invokes the Trust Manager to unbind the secret symmetric key. Since the unbinding function is a protected TPM hardware function, the unbinding will succeed only if the actual configuration matches the one at certificate generation. On success, the Trusted Viewer uses the secret symmetric key for decryption and rendering of the actual content. Note that since the symmetric decryption key resides only within the isolated Trusted Viewer, it cannot be compromised by a malicious legacy OS application or legacy OS leakage.

6 Implementation

Our prototype runs on Intel/AMD IA32 processors, but the underlying L4 microkernel [31] is known to run on embedded processors such as ARM [3] as well. The prototype is securely booted with a modified GRUB bootloader [29]. After startup it shows two *compartments* (cf. Figure 8). The first one contains a standard Linux distribution running on a virtualized Linux kernel. The other compartment contains the Trusted Viewer which currently supports AES-encrypted PDF files. The AES key is bound to the software configuration and has to be unbound by the Trust Manager before use. The Trust Manager builds on an open-source Trusted Software Stack (TSS). All components solely communicate through IPC calls. The GUI for initiating the certificate creation and the rendering of encrypted content is part of the untrusted compartment.

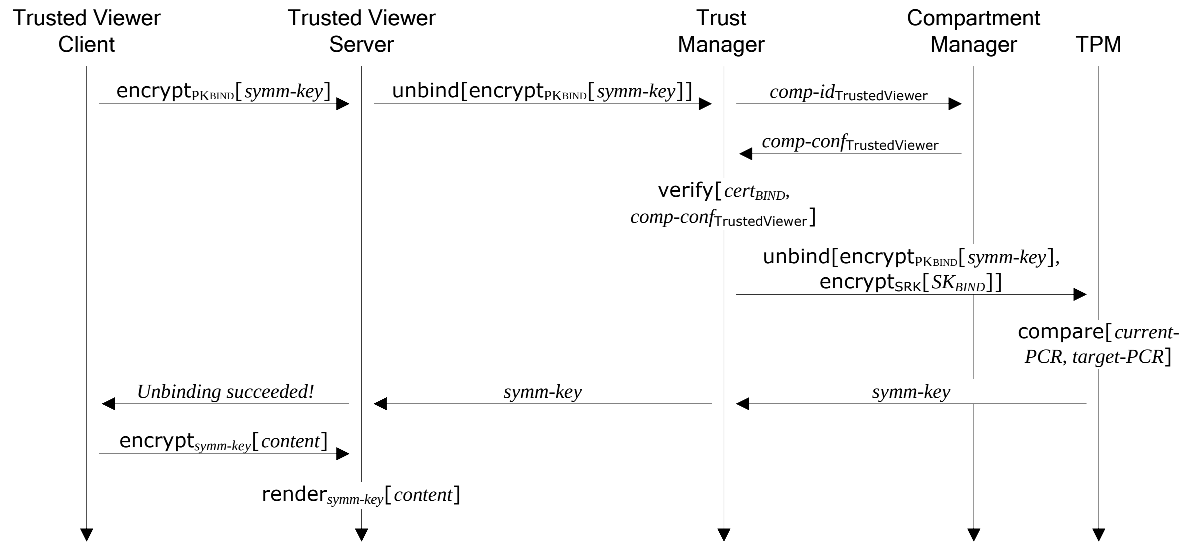

Trusted Viewer Client
Trusted Viewer Server
Trust Manager
Compartment Manager
TPM
encrypt$_{PK_{BIND}}$[*symm-key*]
unbind[encrypt$_{PK_{BIND}}$[*symm-key*]]
comp-id$_{\text{TrustedViewer}}$
comp-conf$_{\text{TrustedViewer}}$
verify[*cert*$_{BIND}$, *comp-conf*$_{\text{TrustedViewer}}$]
unbind[encrypt$_{PK_{BIND}}$[*symm-key*], encrypt$_{SRK}$[SK_{BIND}]]
compare[*current-PCR*, *target-PCR*]
Unbinding succeeded!
symm-key
symm-key
encrypt$_{symm\text{-}key}$[*content*]
render$_{symm\text{-}key}$[*content*]

Figure 7: Unbind protocol.

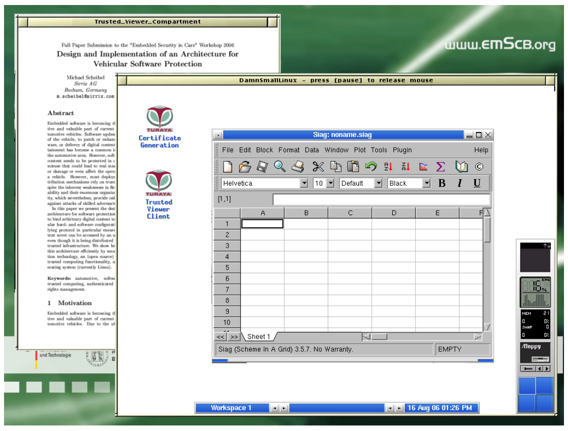

Trusted_Viewer_Compartment
www.emSCB.org
DamnSmallLinux - press [pause] to release mouse
Certificate Generation
Trusted Viewer Client
Slag: noname.siag
File Edit Block Format Data Window Plot Tools Plugin Help
Helvetica
Sheet 1
Siag (Scheme In A Grid) 3.5.7. No Warranty.
EMPTY
Workspace 1
16 Aug 06 01:26 PM

Figure 8: Screenshot showing the legacy OS (foreground) and the Trusted Viewer (background)

7 Summary and Outlook

In this paper we presented the design of a security architecture for software protection that is capable to bind arbitrary digital content to a certain vehicular hard- and software configuration. We explained how the underlying protocol ensures that the content can never be accessed unauthorized even though it is being distributed using a fully untrusted infrastructure. We showed how to implement this architecture efficiently by means of virtualization technology, an (open source) security kernel, Trusted Computing functionality, and a legacy operating system (currently Linux).

Even though TC technology has not yet reached the automotive area, we believe it could solve several critical security issues and enable various innovative automotive business models while requiring only minimal technical and financial resources.

8 Acknowledgments

Our thanks to Rainer Landfermann and André Weimerskirch for their insightful editing remarks and commentary.

References

[1] Adelsbach, A., Huber, U., and Sadeghi, A.-R. Secure software delivery and installation in embedded systems. In *First Information Security Practice and Experience Conference—ISPEC 2005*, vol. 3439 of *Lecture Notes in Computer Science*, pp. 255–267.

[2] Advanced Micro Devices, Inc. AMD virtualization solutions. URL: `enterprise.amd.com/Solutions/Consolidation/virtualization.aspx`.

[3] Advanced RISC Machines Limited (ARM). URL: `www.arm.com`.

[4] Advanced RISC Machines Limited (ARM). Trustzone technology overview. URL: `www.arm.com/products/esd/trustzone_home.html`.

[5] Anderson, R., and Kuhn, M. Tamper Resistance - a Cautionary Note. In *Proceedings of*

the Second Usenix Workshop on Electronic Commerce (1996), pp. 1–11.

[6] ANTHONY BELLISSIMO, JOHN BURGESS, K. F. Secure software updates: Disappointments and new challenges. In *USENIX Hot Topics in Security Workshop—(HotSec 2006).*

[7] ATMEL CORPORATION. Secure microcontrollers. URL: `www.atmel.com`.

[8] BAUERSACHS, G., CHODURA, H., HUBER, M., KOBER, H., KUHLS, B., MIEHLING, T., AND VONDRACEK, P. "Gehirn-Wäsche" fürs Steuergerät. *Elektronik Automotive*, 4 (2005), 53–57.

[9] BROY, M. Sichere Software im Automobil – Potenziale, Herausforderungen,Trends. In *Second Workshop on Embedded Security in Cars—ESCAR 2004, Bochum, Germany, November 10–11, 2004.*

[10] DAIMLERCHRYSLER AG. Functional specification of a flash driver version 1.3. Tech. rep., Herstellerinitiative Software, 2002. URL: `www.automotive-his.de/download/HIS%20flash%20driver%20v130.pdf`.

[11] EHLERS, T. Systemintegrität von vernetzter Fahrzeugelektronik. In *First Workshop on Embedded Security in Cars—ESCAR 2003, Cologne, Germany, November 18–19, 2003.*

[12] ELPHINSTONE, K., HEISER, G., HUUCK, R., PETTERS, S. M., AND RUOCCO, S. L4cars. In *Third Workshop on Embedded Security in Cars—ESCAR 2005, Cologne, Germany, November 29–30, 2005.*

[13] FIBIKOVA, L. On building trusted services in automotive systems. In *Second Workshop on Embedded Security in Cars—ESCAR 2004, Bochum, Germany, November 10–11, 2004.*

[14] GOLEM.DE. Virtuelles linux schneller als natives? URL: `www.golem.de/0606/45659.html`.

[15] INTEL CORPORATION. Intel virtualization technology. URL: `www.intel.com/technology/computing/vptech/`.

[16] KIFMANN, A. Software als Option: BMW prescht vor. *Automobil-Elektronik*, 4 (2005), 38–39.

[17] KLEIDERMACHER, D. Systems software security requirements for in-car digital infotainment. In *Third Workshop on Embedded Security in Cars—ESCAR 2005, Cologne, Germany, November 29–30, 2005.*

[18] KOCHER, P., LEE, R. B., MCGRAW, G., RAGHUNATHAN, A., AND RAVI, S. Security as a new dimension in embedded system design. In *Fortyfirst Design Automation Conference—DAC 2004*, pp. 753–760.

[19] L4HQ. Home of the L4 community. URL: `www.l4hq.org`.

[20] LEMKE, K., SADEGHI, A.-R., AND STÜBLE, C. An open approach for designing secure electronic immobilizers. In *Information Security Practice and Experience—ISPEC 2005, Singapore, April 11-14, 2005*, pp. 230–242.

[21] LIEDTKE, J. Toward real microkernels. *Communications of the ACM 39*, 9 (1996), 70–77.

[22] MILENKOVIC, M., MILENKOVIC, A., AND JOVANOV, E. Hardware support for code integrity in embedded processors. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems—CASES 2005*, pp. 55–65.

[23] NATIONAL ICT AUSTRALIA. L4 performance results. URL: `ertos.nicta.com.au/research/l4/performance.pml`.

[24] PAAR, C. Eingebettete Sicherheit im Automobil. In *First Workshop on Embedded Security in Cars—ESCAR 2003, Cologne, Germany, November 18–19, 2003.*

[25] RAVI, SRIVATHSAND RAGHUNATHAN, A., KOCHER, P., AND HATTANGADY, S. Security in embedded systems: Design challenges. *ACM Transactions on Embedded Computing Systems 3*, 3 (2004), 461–491.

[26] SAILER, R., ZHANG, X., JAEGER, T., AND VAN DOORN, L. Design and implementation of a TCG-based integrity measurement architecture. *13th USENIX Security Symposium, San Diego, California* (August 2004).

[27] SCHEIDEMANN, K. Sicherheitsaspekte nachladbarer Dienste im Automobil. In *Second Workshop on Embedded Security in Cars—ESCAR 2004, Bochum, Germany, November 10–11, 2004.*

[28] SESHADRI, A., PERRIG, A., DOORN, L. V., AND KHOSLA, P. Using software-based attestation for verifying embedded software in cars. In *Second Workshop on Embedded Security in Cars—ESCAR 2004, Bochum, Germany, November 10–11, 2004.*

[29] STUEBLE, C., AND SELHORST, M. Trusted GRUB. URL: `www.prosec.rub.de/trusted_grub.html`.

[30] TCG WORK GROUP. TCG TPM Specification Version 1.2 Revision 94, 2006.

[31] THE FIASCO μ-KERNEL. URL: `www.tudos.org/fiasco/`.

[32] VAN BATTUM, G., AND CALUCCIO, D. Physical security for automotive applications: What can we learn from other industries? In *Third Workshop on Embedded Security in Cars—ESCAR 2005, Cologne, Germany, November 29–30, 2005*.