Secure Software Delivery and Installation in Embedded Systems[1]

André Adelsbach Ulrich Huber Ahmad-Reza Sadeghi
Andre.Adelsbach@nds.rub.de huber@crypto.rub.de sadeghi@crypto.rub.de

Horst Görtz Institute for IT Security
Ruhr-Universität Bochum
Universitätsstraße 150
44780 Bochum
Germany

February 2, 2006

[1] An extended abstract of this technical report appeared in the Proceedings of the First Information Security Practice and Experience Conference (ISPEC 2005) [AHS05].

Abstract

Increasingly, software (SW) in embedded systems can be updated due to the rising share of flashable electronic control units (ECUs). However, current SW installation procedures are insecure: An adversary can install SW in a given ECU without any sender authentication or compatibility assessment. In addition, SW is installed on an all-or-nothing basis: With the installation, the user acquires full access rights to any functionality. Concepts for solving individual deficiencies of current procedures have been proposed, but no unified solution has been published so far.

In this technical report we propose a method for secure SW delivery and installation in embedded systems. The automotive industry serves as a case example leading to complex trust relations and illustrates typically involved parties and their demands. Our solution combines several cryptographic techniques. For example, public key broadcast encryption enables secure SW distribution from any provider to all relevant embedded systems. Trusted computing allows to bind the distributed SW to a trustworthy configuration of the embedded system, which then fulfills a variety of security requirements. Finally, we outline the management of flexible access rights to individual functionalities of the installed SW, thus enabling new business models.

Keywords: secure software installation, broadcast encryption, trusted computing, property-based attestation, rights enforcement

Contents

Chapter 1

Introduction

Control unit hardware (HW) and software (SW) in embedded systems used to be tied together as one single product and rarely changed once the system had been shipped. Nowadays, HW and SW in an electronic control unit (ECU) have become separate products. SW can be updated or upgraded after shipment and add customer value due to the ubiquitous use of flashable[1] ECUs. Examples are the ECUs in a modern car where updates can increase the engine performance and reduce emission levels. Other examples are upgrades of the car navigation system and updates of the road information data.

Current procedures for installing SW in an embedded ECU are insecure—details about the deficiencies will be given in Section 2. Historically, these deficiencies didn't matter because SW installation was focused on warranty-based replacement of defective SW. The system owner was informed of costly recalls and received the SW updates free of charge, e.g., when safety-relevant subsystems like airbags or the ESP[2] contained SW bugs. Recently, a paradigm shift has taken place: Value-added SW components can be distributed to interested owners and new business models allow the extraction of revenues even after shipment, e.g., when car owners pay annual fees for updates of the navigation system data.

The secure delivery of SW to embedded systems and the management of the corresponding digital rights differ from any existing Digital Rights Management (DRM) system known to the authors. First, the distribution currently necessitates a skilled intermediary between SW provider[3] and user because the installation process relies on system-specific equipment which is only available to maintenance personnel. For example, an SW update in a vehicle ECU is usually installed via a manufacturer-specific diagnostic tester that is only intended for maintenance providers.[4] Second, different classes of such intermediaries exist: Depending on their equipment and capabilities, maintenance providers usually have different installation rights. In the automotive example, an uncertified garage might not be granted the right to install SW for safety-relevant ECUs such as the airbag ECU. Third, a newly developed SW component is not necessarily compatible with any target ECU and the SW of all other ECUs

[1] A flashable ECU is a microcontroller capable of reprogramming its memory for application programs and data based on so-called flash memory technology [Dai02].

[2] The Electronic Stability Program (ESP) helps to control a vehicle when it approaches the limits of stability.

[3] By SW provider we mean any party that develops SW for the embedded system, e.g., the original manufacturer of the system and his suppliers, but also independent SW developers.

[4] In the automotive example, maintenance providers such as dealers, garages and road service teams typically carry out the SW installation procedure as the car owner lacks the necessary equipment and skill set [HS04]. Although diagnostic testers are reported to have been cloned or stolen in some cases, the vast majority of SW updates is still carried out by maintenance providers.

in the embedded system. For example, an average compact-class vehicle contains 40 ECUs, while high-end and luxury class vehicles can have up to 70 ECUs.[5] Last, new business models for embedded systems will induce new requirements. Due to the high value of the embedded system and the potential consequences of system failure, non-repudiation will be an important requirement. For example, if an honest car owner has an accident due to defective SW, his dealer and the SW provider may not be able to deny the installation.

We propose a procedure for secure SW delivery and installation in embedded systems. We combine a variety of different cryptographic techniques to build such a secure procedure. The main contribution of our proposal is the secure installation procedure itself based on Public Key Broadcast Encryption (PKBE) and trusted computing. Another contribution will be a requirement model for all parties that participate in a typical distribution and installation setting. To the authors' knowledge, neither a suitable procedure nor a general requirement model has been previously published although several individual requirements have been proposed [BMW02, HS04, Stö03].

The use of the PKBE scheme proposed in [DF03] has several advantages in this particular setting.[6] First, it enables *efficient* one-way communication from SW providers to a potentially large, but select set of embedded systems, even though they have to be considered *stateless* receivers.[7] Specifically, the length of the message header does not grow with the number of intended receivers[8] as in the case of a standard Public Key Infrastructure (PKI).[9] Second, the proposed PKBE scheme allows the revocation of an unbounded number of receivers. Even if a large number of receivers has been compromised or is to be excluded, messages can still be broadcast to the remaining receivers. Last, it gives non-discriminatory access to the broadcast channel. The public key property allows any (not necessarily trusted) party to broadcast to any chosen set of receivers. Specifically, the manufacturer of the embedded system cannot exclude any SW provider from the broadcast channel or otherwise prevent competition.[10]

Trusted Computing is the enabling technology for an embedded system to become a trusted receiver of broadcast messages. Based on minimum additional hardware and cryptographic techniques such as attestation and sealing, an embedded system can be trusted to be in a particular configuration. The assessment of the compatibility of a particular SW component with the embedded system can be based on this configuration. In order to avoid discrimination of certain SW providers, we suggest the use of property-based attestation as introduced in [SS05].[11]

Section 2 briefly summarizes the work of other authors and illustrates deficiencies of the

[5] The Volkswagen Phaeton has 61 ECUs [HMF+03]. In addition, each OEM usually has different car models with differing ECU configurations. The ECU configuration of a particular model changes during the production life cycle due to an update of HW or SW components [HMF+03, Sch03]. The compatibility of an SW component does not only depend on the target ECU hardware, but also on other ECUs in the vehicle [AJ03, HWM03, Oef04].

[6] Broadcast encryption was introduced in [FN94] based on symmetric encryption schemes. Several improvements were proposed, e.g., in [NNL01, HS02, JHC+05]. We refer to PKBE in [DF03].

[7] Stateless receivers contain a set of secret keys which can't be updated throughout the system's lifetime.

[8] Intended receivers are all embedded systems to which the SW provider wishes to distribute a specific SW.

[9] If standard PKI was used on a broadcast channel, the message header length would be $O(|\mathcal{U} \setminus \mathcal{R}_{\mathcal{U}}|)$, where $\mathcal{U}$ is the set of receivers and $\mathcal{R}_{\mathcal{U}}$ the set of revoked receivers with $|\mathcal{R}_{\mathcal{U}}| \ll |\mathcal{U}|$. In the PKBE scheme from [DF03], the message header length is only $O(|\mathcal{R}_{\mathcal{U}}|)$.

[10] Non-discrimination is also important on the receiving end: For instance, the European Commission Regulation 1400/2002 prevents discrimination of independent maintenance providers. The manufacturer must give them access to necessary material and technical information, e.g., spare parts and diagnostic equipment.

[11] Similar methods are "semantic attestation" and "property attestation" [HCF04, PSHW04].

current SW installation practice. Our overall system model is explained in Section 3. The security requirement model follows in Section 4, while the proposed solution is discussed in Section 5. We conclude and highlight open questions in Section 6.

Chapter 2

Related Work

Several types of embedded systems exist and specific literature on each type is available. However, we consider a modern vehicle to be the most challenging example, namely due to the specific qualities of SW distribution and installation as outlined in Section 1. In particular, the high number of ECUs and their variants leads to a complex assessment of compatibility. Therefore, we focus on automotive literature and add an example from the field of IT security.

2.1 Current Practice and Deficiencies

A typical procedure for installing SW in an automotive ECU is described in [Dai02]. It is performed by a so-called flashloader, a standard SW environment that allows for in-system re-programming of ECUs. After initialization of the installation mode, the flashloader erases the programmable memory of the ECU. Then it writes the new SW into the programmable memory. Finally, the procedure ends with the deinitialization of the installation mode.

Current installation procedures rarely apply any cryptographic techniques [Dai02, DS04, HWM03]. The use of signatures has been proposed, but not yet implemented [HS04, HWM03, Mül04]. However, the only signature mentioned in the proposals is that of the manufacturer.[1] If the manufacturer must sign every SW component prior to installation, he is capable of discriminating individual SW providers. In addition, we illustrate several other deficiencies with some examples. First, the intellectual property contained in the SW is not protected with cryptographic techniques such as encryption, opening the door for reverse-engineering attacks. Second, the installation rights of the maintenance providers are not verified in the course of an installation. Hence anyone with the necessary equipment—including an adversary—can install any (potentially malicious) SW component. Third, the owner cannot prove that he has legally[2] acquired an SW component that has been installed in his embedded system. Even if the manufacturer applies a signature, the owner can still be accused of having acquired the SW illegally, e.g., without payment of license fees. Fourth, the embedded system does not verify compatibility. Even if signatures are used, they only prove the source of the SW, not compatibility. SW might be erroneously accepted by an incompatible embedded system due to the manufacturer's signature. Last, no rights management is currently applied. Techniques

[1] The proposals generally do not specify whether they refer to the manufacturer of the embedded system or that of the relevant ECU.

[2] By legal acquisition we mean the installation of a compatible SW component from a maintenance provider with the necessary installation rights including payment of all license fees to the SW provider.

such as expiry dates or usage counters are not yet implemented and prevent the introduction of more flexible business models. In the automotive example, those techniques would allow to sell additional horsepower or country-specific navigation data for a limited time frame or number of usages.

2.2 Other proposed solutions

A framework for international automotive SW installation standards is introduced in [DS04]. However, it doesn't consider any DRM or security aspects. An infrastructure for installing SW from any external interface is proposed in [HS04]. Although compatibility is ensured by checking if the hash values of all involved SW components form a valid SW release,[3] further security aspects are not covered. Requirements such as confidentiality, integrity, non-rejection and authenticity are mentioned, but not considered in the proposed architecture and left open to the specific implementation of each vehicle manufacturer. Several other papers introduce the concept of distributing SW to embedded systems in the field [BMW02, Stö03], but even if security requirements are mentioned, no specific proposal to fulfill them is mentioned.

A proposal for "end-to-end security" of SW installation in vehicles is made in [Mül04]. However, the signing of the SW component by "an authorized party" is the only protective measure, which provides only a partial solution[4] to the requirements that we will introduce in Section 4. Another proposal for secure SW installation is made in [HWM03]: It contains an authentication phase, in which the diagnostic tester is authenticated, as well as an installation routine, which verifies checksums or signatures of the SW provider. Again, only some of the requirements are fulfilled.[5]

Relevant IT security literature focuses on enforcement of access control policies for downloaded executable content by a secure operating system (OS) or by a secure SW environment which encapsulates the content [JPLI99]. However, embedded ECUs often do not have a standardized operating system. When SW is installed, the whole program memory can be erased and replaced with a new SW component. Therefore, we cannot assume a secure OS or SW environment in every ECU; the content thus needs to be analyzed *prior* to installation.

[3] An SW release is an SW configuration which has been released by the vehicle manufacturer. An SW configuration is a valid and operational set of SW components and corresponding coding parameters which can be programmed in the ECUs of a vehicle [HS04].

[4] For example, it does not prevent discrimination of independent SW providers as the vehicle manufacturer is assumed to take over the role of the authorized party.

[5] For example, signatures on the receiving end are omitted. Therefore the proposal does not prevent repudiation of a successful installation by the vehicle owner.

Chapter 3

Model

3.1 Introduction of Roles

The following roles (see Figure 3.1) will be used throughout the remainder of this report:

(O) **OEM:** The Overall Equipment Manufacturer (OEM) develops, assembles and delivers the embedded system to the users. In order to do so, O cooperates with suppliers that develop and/or manufacture components for the embedded system. The initial SW components at shipment time may be either from O or from his suppliers. Automotive examples are car manufacturers such as Daimler Chrysler, Ford, GM or Toyota.

(S) **SAP:** SW Application Programmers (SAPs) develop SW components for the embedded system. They may either be (i) suppliers that participate in developing and/or assembling the embedded system or (ii) independent programmers that develop SW components (updates and/or upgrades) and distribute them after shipment. Automotive examples are suppliers such as Bosch, Delphi, Denso, Siemens and Visteon.

Henceforth, we use the term *SW provider* as a synonym for "OEM or any SAP".

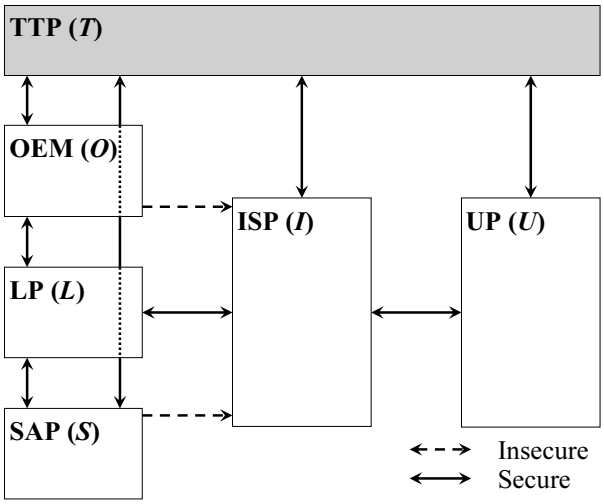
(I) **ISP:** The Installation Service Providers (ISPs) maintain the embedded system, i.e., mechanical parts, ECU HW and SW. As part of their maintenance services, they install updates and/or upgrades of SW components. They have equipment that is necessary for the installation procedure and capabilities that allow them to correctly install SW components. Automotive examples are car dealers, garages and road service teams.

We model the installation rights of I as clearance levels (see Definition 1).[1]

(L) **LP:** The License Provider (LP) distributes licenses for SW components that the SW providers O and S have developed. Prior to distribution of a license, L needs to establish terms and conditions with the SW providers in which the model for sharing license revenues is detailed.[2] To the authors' knowledge, automotive examples don't exist yet, but might be established as joint-ventures of OEMs and SAPs.

[1] Other models for installation rights can easily be integrated into our proposal. For the purpose of this report, clearance levels serve as an example.

[2] A discussion of licensing models, e.g., pay-per-use, pay-per-installation, is beyond the scope of this report.



Fi

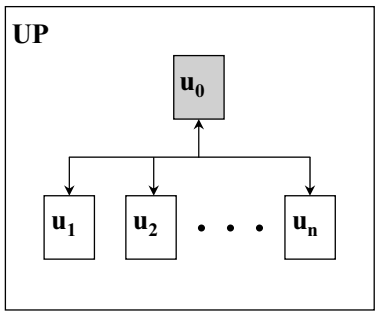


Figure 3.2: Internal structure of the user platform

(U) UP: The User Platform (UP) is manufactured by O and purchased by the user. The user is interested in SW for U and willing to pay for it if it offers a perceivable value-added. We define U's configuration as the collective information on each SW (and implicitly HW) component that is installed in U. The obvious automotive example for U is a car.

We assume U to have the internal structure depicted in Figure 3.2: $u_0, u_1, \ldots, u_n$ are components of U. In the implementation of an embedded system, they correspond to ECUs. u_0 is assumed to be the major part of the trusted computing base (see Section 5.2.1) and provides a central installation and license service. u_0 is the only component capable of distributing new SW to the other components u_i, $1 \leq i \leq n$. Due to cost constraints, we cannot assume the u_i to be high-performance components, i.e., their computational resources are limited, especially related to cryptographic techniques. The SW distribution from u_0 to the u_i is performed over an internal communication network to which all components are connected.[3]

[3]If several communication networks coexist, we assume that they are interconnected via gateways and form a coherent network. In the automotive example, this holds true for communication networks—so-called data busses—such as CAN, LIN and MOST.

(T) **TTP:** The Trusted Third Party (TTP) has two different certification tasks: First, T creates SW certificates for SW providers. These certificates confirm the properties of a newly developed SW component. By SW properties we mean characteristic features of SW such as functionality, interfaces, memory and processor requirements or supported protocols. Second, T creates clearance level certificates which certify I's right to install specific SW components.

In the automotive example, the role of the TTP is currently taken over by O. This implies a trust model in which each S_d must trust O. However, an independent T becomes necessary if O is not fully trusted and discrimination of S_d should be avoided. An independent T might evolve out of safety standards authorities such as the NHTSA[4] in the USA or the Euro NCAP[5] in Europe.

3.2 Model Formalization

After having introduced the model informally in Section 3.1, we now continue with a formal definition of roles, objects, technical terms, basic protocols and cryptographic primitives.

3.2.1 Formalization of Roles

OEM There is a single OEM O in our model. Nevertheless, extending the model to several OEMs is straightforward.

SAPs The set $\mathcal{S}$ of SAPs is defined as $\mathcal{S} := \{S_1, S_2, \ldots\}$. We use index d for SAP S_d.

SW providers OEM and SAPs have very similar roles in our model as they both offer SW components. In order to simplify the notation, we summarize them with P in the following way: $\mathcal{P}$ is the set of all SW providers, which is defined as $\mathcal{P} := \{P_0, P_1, P_2, \ldots\}$. We match $\mathcal{P}$ with O and $\mathcal{S}$ as follows, using index d throughout the report:

$$P_0 := O, \quad P_d := S_d \ \forall\, d \geq 1$$

ISPs The set $\mathcal{I}$ of ISPs is defined as $\mathcal{I} := \{I_1, I_2, \ldots\}$. We use index j for ISP I_j.

LP There is a single license provider L in our model. Nevertheless, extending the model to several license providers is straightforward.

UPs The set $\mathcal{U}$ of user platforms is defined as $\mathcal{U} := \{U_1, U_2, \ldots\}$. We use index k for user platform U_k, which has an internal structure with $n_k + 1$ components as depicted in Figure 3.2. The components of U_k are denoted $u_{k,i}$, with $1 \leq k \leq |\mathcal{U}|$ and $0 \leq i \leq n_k$.

TTP There is a single trusted third party T in our model.

[4] National Highway Traffic Safety Administration, `http://www.nhtsa.dot.gov/`
[5] `http://www.euroncap.com/`

3.2.2 Formalization of Objects

There are two types of objects in our model: *messages* and *SW components*. The former will be introduced in Section 3.2.4, the latter are the objects generated by $\mathcal{P}$, certified by T, licensed by L, installed by $\mathcal{I}$ and used by $\mathcal{U}$:

SW components Each SW component s_m is an element of the language $\mathcal{SW} := \{s_1, s_2, \ldots\} \subseteq \{0,1\}^*$ of all SW components. We use the index m for SW component s_m.

3.2.3 Preliminaries

We have already informally used several terms that need formal definition in order to facilitate requirement definition and security analysis. Before explaining the basic protocol steps, we define these terms. However, the reader may skip the definitions and trust in the fact that they are intuitive and self-explanatory. If questions arise in later sections, we recommend to return to the definitions:

Definition 1 A **clearance level** $c \in \mathcal{C}$ with $\mathcal{C} \subset \mathbb{N}$ is a right of an ISP I_j to install a well-defined set of SW components $\mathcal{SW}_c \subseteq \mathcal{SW}$. The **minimum clearance level** $c^{\mathsf{min}} \in \mathcal{C}$ of an SW component s defines the smallest clearance level that an ISP may have when installing s. It is determined with an implementation-specific function $\mathsf{MinClearance} : \mathcal{SW} \rightarrow \mathcal{C}$, which assigns an unambiguous minimum clearance level c^{min} to every SW component s. With c_m^{min} we always refer to the correct minimum clearance level of s_m defined by $c_m^{\mathsf{min}} \leftarrow \mathsf{MinClearance}(s_m)$. $\mathsf{MinClearance}$ is used to define $\mathcal{SW}_c$:

$$\mathcal{SW}_c := \{s \in \mathcal{SW} \mid c \geq \mathsf{MinClearance}(s)\}$$

An ISP I_j is said to be **compliant with a clearance level** c if he fulfills all requirements of this level, e.g., regarding equipment and skills. We denote this fact with $I_j \models c$. □

Many models for defining the requirements of clearance levels are possible; their discussion is beyond the scope of this report. In general, the definition is the output of a multi-party protocol between T, O, $\mathcal{S}$ and $\mathcal{I}$.

Definition 2 The **terms and conditions** $\mathcal{T}\&\mathcal{C}_m$ are a set of legally binding rules of conduct between SW provider P_d and license provider L for SW component s_m. □

For example, $\mathcal{T}\&\mathcal{C}_m$ specifies what P_d delivers to L and what L pays as remuneration to P_d. As $\mathcal{T}\&\mathcal{C}_m$ is a complex legal concept, we cannot fully model it mathematically. However, we focus on one aspect: $\mathcal{T}\&\mathcal{C}_m$ specifies the set of all allowed usage rights $\mathcal{R}_m^{\mathsf{total}}$ as follows.

Definition 3 A **usage right** r is a permission for a user platform U_k to perform specific operations on an SW component s_m. An **allowed usage right** is a usage right that L may grant to U_k without violating $\mathcal{T}\&\mathcal{C}_m$. This defines the set $\mathcal{R}_m^{\mathsf{total}}$ of all allowed usage rights:

$$\mathcal{R}_m^{\mathsf{total}} := \{r \mid r \text{ is an allowed usage right for } s_m \text{ under } \mathcal{T}\&\mathcal{C}_m\}$$

An **allowed rights set** $\mathcal{R}_m$ is a set of allowed usage rights $\mathcal{R}_m \subseteq \mathcal{R}_m^{\mathsf{total}}$. □

Definition 4 A **license** σ^{lic} unambiguously defines the target user platform U_k, the corresponding SW component s_m and the rights set $\mathcal{R}_m$ that is granted. □

Definition 5 Let a usage parameter p of SW s_m be a variable that influences the mode of execution of s_m. A **correct usage parameter** of s_m is a usage parameter that influences s_m in such a way that it complies with the corresponding license, i.e., the user platform U_k can only perform the operations defined by the rights set $\mathcal{R}_m$. □

Definition 6 Let the specified functionality of an SW component s_m be its input-output behavior as described by the honest SW provider P_d of s_m. A software s_m and a platform U_k are **compatible** iff, after an installation of s_m on U_k, s_m provides the specified functionality. Otherwise, s_m and U_k are **incompatible**. □

Definition 7 A **SW property** q is a specific quality of an SW component, where the set of all SW properties is denoted with $\mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}$. A **platform property** Q is a specific quality of a user platform U_k, where the set of all platform properties is denoted with $\mathcal{Q}_{\mathsf{UP}}^{\mathsf{total}}$. □

There is an unlimited number of properties. However, only a subset of these is relevant for our installation procedure. We therefore need to define this subset based on a statement regarding compatibility.

Definition 8 The **compatibility test function** compares SW and platform properties:[6] $\mathsf{CompatTest} : \mathcal{P}(\mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}) \times \mathcal{P}(\mathcal{Q}_{\mathsf{UP}}^{\mathsf{total}}) \rightarrow \{\mathtt{true}, \mathtt{false}\}$ outputs `true` on input $(\mathcal{Q}_m, \mathcal{Q}_k)$ iff an SW component s_m with properties $\mathcal{Q}_m \subseteq \mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}$ and a user platform U_k with properties $\mathcal{Q}_k \subseteq \mathcal{Q}_{\mathsf{UP}}^{\mathsf{total}}$ are compatible. Otherwise, it outputs `false`. □

We suppose that the trusted component $u_{k,0}$ of U_k is capable of computing the compatibility test function $\mathsf{CompatTest}$ based on the properties $\mathcal{Q}_m$ of s_m and the properties $\mathcal{Q}_k$ of U_k. For this purpose, $u_{k,0}$ interprets $\mathcal{Q}_m$ and derives required properties for U_k such as interfaces, minimum memory and minimum processing power. If U_k has all of the required properties, $\mathsf{CompatTest}$ returns `true`.

Definition 9 An SW property q is **compatibility-relevant** iff adding $q \in \mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}$ changes the output of $\mathsf{CompatTest}$ for at least one set of SW and platform properties, i.e., iff the following condition holds:

$$\begin{aligned} &\exists\ \mathcal{Q}_m \subseteq \mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}} \setminus \{q\}; \quad \exists\ \mathcal{Q}_k \subseteq \mathcal{Q}_{\mathsf{UP}}^{\mathsf{total}}; \quad \exists ind \in \{\mathtt{true}, \mathtt{false}\} : \\ &\mathsf{CompatTest}(\mathcal{Q}_m, \mathcal{Q}_k) = ind \quad \wedge \quad \mathsf{CompatTest}(\mathcal{Q}_m \cup \{q\}, \mathcal{Q}_k) = \neg ind \end{aligned}$$

Henceforth we assume $\mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}$ only contains compatibility-relevant SW properties.[7] □

Definition 10 With $\mathcal{Q}_m^{\mathsf{total}}$ we denote the **complete property set of SW component** s_m:

$$\mathcal{Q}_m^{\mathsf{total}} := \{q \in \mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}} \mid s_m \text{ has property } q\} \subseteq \mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}$$

It is determined with an implementation-specific function $\mathsf{DetermProps} : \mathcal{SW} \rightarrow \mathcal{P}(\mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}})$, which determines the complete property set $\mathcal{Q}_m^{\mathsf{total}}$ of an SW component s_m. With $\mathcal{Q}_m^{\mathsf{total}}$ we always refer to the correct and complete property set of s_m defined by $\mathcal{Q}_m^{\mathsf{total}} \leftarrow \mathsf{DetermProps}(s_m)$. We define a **property set of SW component** s_m as $\mathcal{Q}_m \subseteq \mathcal{Q}_m^{\mathsf{total}}$. □

[6] With $\mathcal{P}(\mathcal{X})$ we denote the power set of the set $\mathcal{X}$, i.e., the set of all subsets of $\mathcal{X}$.

[7] Unfortunately in practice, the set of compatibility-relevant SW properties is unknown because finding them would necessitate a series of tests of all possible SW properties with all possible platform properties. As both property sets are unlimited, the tests would last indefinitely. In the automotive example, standardization as well as a reasonable test depth and breadth ensure that the SW provider identifies most of these properties.

Definition 11 The **target component** for an SW s_m is the component $u_{k,i}, i \in \{1, \ldots, n_k\}$ of U_k that the trusted component $u_{k,0}$ chooses for installing s_m. The implementation-specific function $\mathsf{Target} : \mathcal{P}(\mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}) \times \mathcal{P}(\mathcal{Q}_{\mathsf{UP}}^{\mathsf{total}}) \rightarrow \{1, \ldots, n_k\}$ takes as input the SW properties $\mathcal{Q}_m \subseteq \mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}$ of s_m and the platform properties $\mathcal{Q}_k \subseteq \mathcal{Q}_{\mathsf{UP}}^{\mathsf{total}}$ of U_k and outputs the index i of the target component $u_{k,i}$. □

For example, Target might consider criteria such as available memory space, processor speed, location and proximity of all components.

3.2.4 Formalization of Basic Protocols

An SW installation procedure consists of several basic protocols. In this section, we define them independently of each other. However, we leave the actual implementation of these basic protocols to Section 5.3.

Prior to the actual protocols, we introduce our protocol notation. We are concerned with protocols between several parties $X_1, \ldots, X_n$ that exchange messages. We model each party as an interactive probabilistic algorithm. A common model for such algorithms is an Interactive Turing Machine (ITM) (for an example, see [GMR89]). An ITM is a deterministic multi-tape Turing machine consisting of the following components: (i) a local read-and-write tape, (ii) a local read-only random tape filled with uniformly distributed random bits (before the start of computation), and (iii) a read-only receiving tape as well as a write-only sending tape for communication with other machines (see [Gol01] for more details).
We denote a protocol Protocol between the interactive algorithms $X_1, \ldots, X_n$ as follows:

$$(X_1 : out_{X_1}; \ldots; X_n : out_{X_n}) \leftarrow \mathsf{Protocol}(X_1 : in_{X_1}; \ldots; X_n : in_{X_n}; * : in)$$

Each party X_i provides its input values in_{X_i} and may obtain output values(s) out_{X_i} after the end of protocol Protocol. In our notation, we separate parties with semicolons, whereas we separate the input and output values of a party with commas. We denote the common input, i.e., input values of all parties, with the placeholder $*$. If a party isn't required to provide any input value to the protocol, or if it doesn't obtain any output value, then we indicate this by the symbol $-$. This concludes the protocol notation, and the basic protocols follow.

The SystemSetup protocol is the first basic protocol to be executed. In this protocol, all public parameters and all secrets (for example, private and shared cryptographic keys) are generated and properly distributed to the appropriate parties. In all subsequent protocols, we do not explicitly mention public parameters and secrets. Instead, we assume that each interactive algorithm knows its secret(s) and the public parameters. For example, they may be encoded in the ITM's state transitions. In addition, each interactive algorithm is familiar with the relevant definitions of Section 3.2.3 and the cryptographic primitives of Section 3.2.5.

We illustrate the first two basic protocols in Figure 3.3. We start with the protocol for certification of the ISPs' clearance levels, followed by the protocol for SW certification:

Clearance Level Certification: This protocol occurs between an ISP I_j and the TTP T.
We assume that the index j of I_j has already been assigned in the SystemSetup protocol:

$$(I_j : \zeta_j^{\mathsf{clear}}, ind_j^{\mathsf{clear}}; T : req^{\mathsf{clear}}, ind_T^{\mathsf{clear}}) \leftarrow \mathsf{ClearCert}(I_j : c; T : -; * : j)$$

We explain the meaning of the input and output values. Starting with the input values, c is the requested clearance level and j is the index of I_j. The output values of I_j are

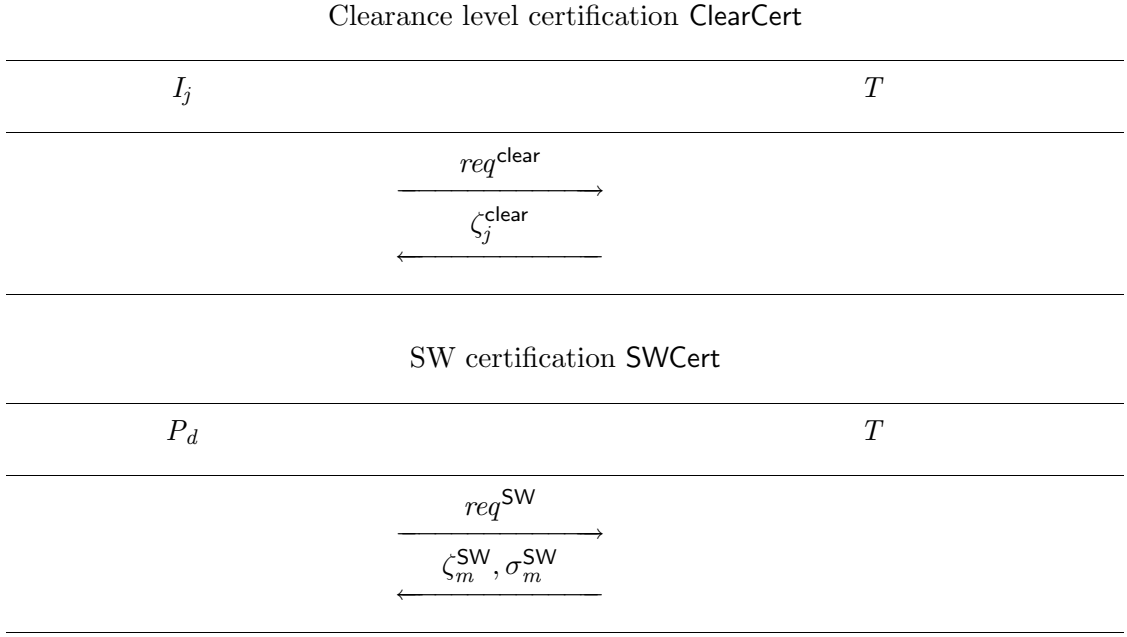


Figure 3.3: Illustration of two basic protocols

the clearance level certificate ζ_j^{clear} and an acceptance indicator $ind_j^{\mathsf{clear}} \in \{\texttt{true}, \texttt{false}\}$, indicating whether I_j accepts the output values of ClearCert as correct ($ind_j^{\mathsf{clear}} = \texttt{true}$). The output values of T are the certification request req^{clear} and an acceptance indicator ind_T^{clear}, indicating whether T accepts the output values of ClearCert.

SW Certification: This protocol occurs between a provider P_d and T. We assume that the index d of P_d has already been assigned in the SystemSetup protocol:[8]

$$(P_d : \zeta_m^{\mathsf{SW}}, \sigma_m^{\mathsf{SW}}, ind_d^{\mathsf{SW}}; T : req^{\mathsf{SW}}, m, ind_T^{\mathsf{SW}}) \leftarrow \mathsf{SWCert}(P_d : s; T : m; * : d)$$

The input values are the SW component s to be certified, the SW index m and the index d of SW provider P_d, where the SW index m is a counter for the number of SW components that T has certified. The output values of P_d are the SW certificate ζ_m^{SW}, the integrity proof σ_m^{SW} for s_m and an acceptance indicator ind_d^{SW}. The output values of T are the certification request req^{SW}, the updated SW index m and an acceptance indicator ind_T^{SW}.

Usage Rights Definition: This protocol occurs between an SW provider P_d and the license provider L:

$$(L : \mathcal{R}_m^{\mathsf{total}}; P_d : \mathcal{R}_m^{\mathsf{total}}) \leftarrow \mathsf{RightsDef}(L : in_L; P_d : in_{P_d}; * : m)$$

The input values are the generic values in_L and in_{P_d} as well as the index m of SW s_m. The output value of both parties is the set $\mathcal{R}_m^{\mathsf{total}}$ of all allowed usage rights.

[8] In a different trust model, O might be the party that certifies SW. This would significantly reduce the workload on T. However, it would require all $S \in \mathcal{S}$ to trust O or result in dispute if O denied fair evaluation.

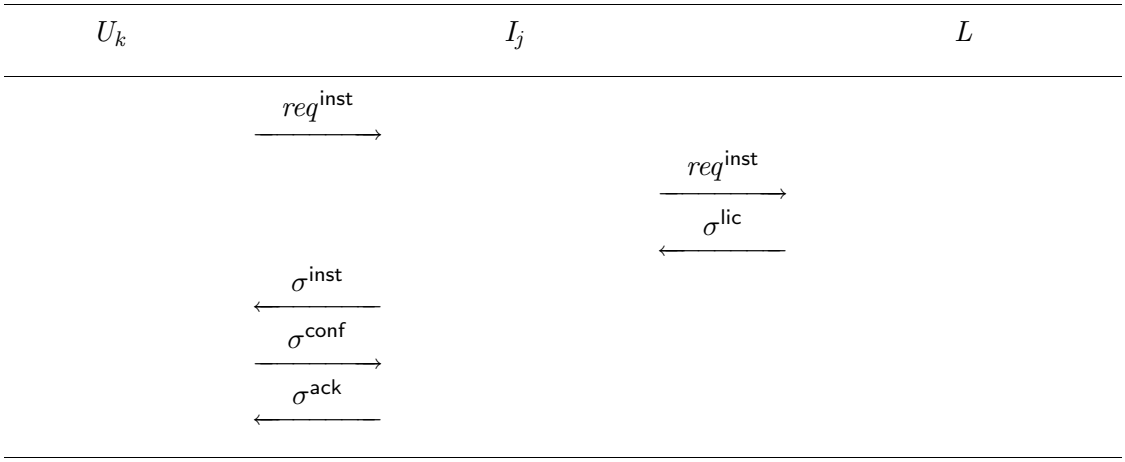


Figure 3.4: Illustration of the 3-party protocol SWInstallExternal

After SW certification, the SW provider needs to distribute the SW components to the ISPs in order to allow them to perform the installation procedure. The following protocol describes this SW distribution process that provides copy protection and authenticity of the distributed SW. Details about the cryptographic background will be given in Section 5.3.1.

SW distribution: This protocol occurs between a provider P_d and the set $\mathcal{I}$ of all ISPs. It is one-way from P_d to $\mathcal{I}$ in the sense that $\mathcal{I}$ only receives messages, but cannot send:[9]

$$(P_d : -; \mathcal{I} : s_m^{\mathsf{enc}}, \sigma_m^{\mathsf{prop}}, \zeta_m^{\mathsf{SW}}) \leftarrow \mathsf{SWDistrib}(P_d : \zeta_m^{\mathsf{SW}}, \sigma_m^{\mathsf{SW}}, \mathcal{R}_{\mathcal{U}}; \mathcal{I} : -; * : -)$$

The input values are the SW certificate ζ_m^{SW}, the integrity proof σ_m^{SW} and the subset $\mathcal{R}_{\mathcal{U}} = \{U_{k_1}, U_{k_2}, \ldots\} \subseteq \mathcal{U}$ of revoked[10] user platforms. P_d obtains no output value, whereas the output values of $\mathcal{I}$ are the protected SW component s_m^{enc}, a verifiable statement σ_m^{prop} of P_d with respect to s_m's properties and the SW certificate ζ_m^{SW}.

SW Installation External: The following protocol describes an actual SW installation. We show the flow of messages in Figure 3.4. This protocol occurs between a user platform U_k, an ISP I_j and the license provider L:

$$\begin{aligned}&(U_k : \sigma^{\mathsf{inst}}, \sigma^{\mathsf{ack}}, ind_k^{\mathsf{inst}}, \mathcal{Q}_k; I_j : req^{\mathsf{inst}}, \sigma^{\mathsf{lic}}, \sigma^{\mathsf{conf}}, ind_j^{\mathsf{inst}}; L : req^{\mathsf{inst}}, ind_L^{\mathsf{inst}})\\ &\leftarrow \mathsf{SWInstallExternal}(U_k : m, \mathcal{R}_m, \zeta_j^{\mathsf{clear}}, \zeta_m^{\mathsf{SW}}, \sigma_m^{\mathsf{prop}}, \mathcal{Q}_k; I_j : s_m^{\mathsf{enc}}; L : \mathcal{R}_m^{\mathsf{total}}; * : k, j)\end{aligned}$$

The input values of user platform U_k are the index m of s_m, the requested usage rights $\mathcal{R}_m$, I_j's clearance level certificate ζ_j^{clear}, the SW certificate ζ_m^{SW} of s_m, the property statement σ_m^{prop} of P_d with respect to s_m and the platform properties $\mathcal{Q}_k$ of U_k. The input value of ISP I_j is the protected SW component s_m^{enc}, whereas the license provider

[9] Specifically, the ITM P_d has no receiving tape and the ITMs $\mathcal{I}$ have no sending tape.

[10] Initially all user platforms are non-revoked. The SW providers consider a user platform revoked only if it has violated the protocol, e.g., published secret information or violated the granted usage rights.

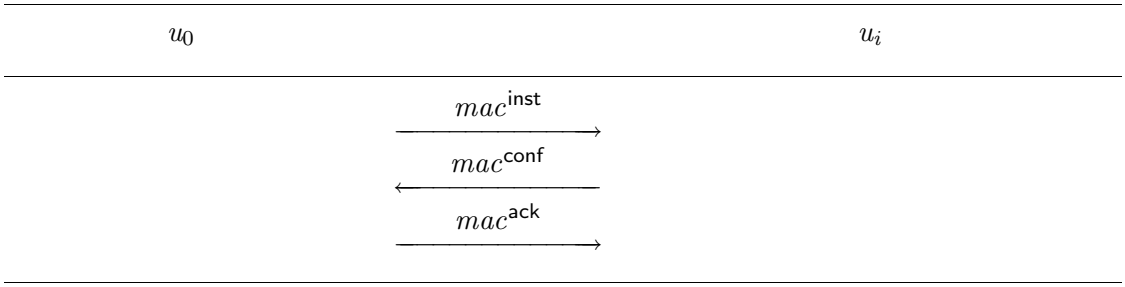


Figure 3.5: Illustration of the basic protocol SWInstallInternal

L's input value is the set $\mathcal{R}_m^{\mathsf{total}}$ of all allowed usage rights of s_m. The common input values are the indices k and j of U_k and I_j respectively.

The output values of user platform U_k are the external installation package σ^{inst}, containing SW component and license, the external installation acknowledgement σ^{ack} of I_j, an acceptance indicator ind_k^{inst} and the updated platform properties $\mathcal{Q}_k$. The output values of I_j are the SW installation request req^{inst} and the external installation confirmation σ^{conf} generated by U_k, the license σ^{lic} generated by L and the acceptance indicator ind_j^{inst}. The output values of L are the SW installation request req^{inst} of U_k and the acceptance indicator ind_L^{inst}.

SW Installation Internal: The following protocol (see Figure 3.5) is initiated by the trusted component $u_{k,0}$ of user platform U_k during a run of protocol SWInstallExternal. $u_{k,0}$ initiates the protocol only if it has already received a correct installation package σ^{inst}, where the conditions for correctness will be detailed in Section 5.3.2. The other protocol participant is the target component $u_{k,i}$ for s_m (see Definition 11). As the index k is unambiguous, we will simply refer to the components by u_0 and u_i:

$$(u_0 : mac^{\mathsf{conf}}, ind_0^{\mathsf{inst}}; u_i : mac^{\mathsf{inst}}, mac^{\mathsf{ack}}, s, p, ind_i^{\mathsf{inst}})$$
$$\leftarrow \mathsf{SWInstallInternal}(u_0 : i, m, s; u_i : -; * : -)$$

The input values of the trusted component u_0 are the index i of target component u_i, the index m of the SW component s_m to be installed and the SW component s. There is no input to u_i and no common input. The output values of u_0 are the internal installation confirmation mac^{conf} and the acceptance indicator ind_0^{inst}, whereas the output values of u_i are the internal installation package mac^{inst}, the internal installation acknowledgement mac^{ack}, the SW component s, the usage parameter p and the acceptance indicator ind_i^{inst} .

3.2.5 Formalization of Cryptographic Primitives

(GenKeySig, Sign, Verify) is a tuple that denotes the key generation, signing and verifying of a UF-CMA-secure[11] digital signature scheme which ensures authenticity and non-repudiation.

[11]For definitions of security notions, we refer to [BDPR98, ADR02].

$\sigma_X \leftarrow \mathsf{Sign}(K_X^{\mathsf{sign}}, M)$ means the signing of the message M with X's signing key K_X^{sign}, resulting in the signature tuple $\sigma_X := (M, sig_M)$. Note that σ_X contains both the message M and the actual signature sig_M. Two auxiliary functions split the signature tuple into its two components: $M \leftarrow \mathsf{Msg}(\sigma_X)$ and $sig_M \leftarrow \mathsf{Sig}(\sigma_X)$. $ind \leftarrow \mathsf{Verify}(K_X^{\mathsf{test}}, \mathsf{Msg}(\sigma_X), \mathsf{Sig}(\sigma_X))$ means the verification of σ_X with the test key K_X^{test}, which we sometimes abbreviate with $ind \leftarrow \mathsf{Verify}(K_X^{\mathsf{test}}, \sigma_X)$. The result is the Boolean value $ind \in \{\mathtt{true}, \mathtt{false}\}$.

(GenKeyPKBE, RegPKBE, EncPKBE, DecPKBE) is a tuple that denotes the key generation, user registration, encryption and decryption of an IND-CCA1-secure PKBE scheme (see Appendix B). T uses GenKeyPKBE to set up all the parameters of the scheme, e.g., the set of all public keys $\mathcal{K}^{\mathsf{enc}}$, which is available to any party. T uses RegPKBE to compute the set of secret decryption keys $\mathcal{K}_k^{\mathsf{dec}}$ to be delivered to a user U_k. $C \leftarrow \mathsf{EncPKBE}(\mathcal{K}^{\mathsf{enc}}, \mathcal{R}_{\mathcal{U}}, M)$ is used by a (not necessarily trusted) sender to encapsulate a message M with the set of public keys $\mathcal{K}^{\mathsf{enc}}$ in such a way that only the unrevoked users $\mathcal{U} \setminus \mathcal{R}_{\mathcal{U}}$ can recover it. $\mathsf{DecPKBE}(\mathcal{K}_k^{\mathsf{dec}}, C)$ is used by a non-revoked user U_k to decipher C with his private key set $\mathcal{K}_k^{\mathsf{dec}}$. DecPKBE returns M iff the user is non-revoked, i.e., $U_k \in \mathcal{U} \setminus \mathcal{R}_{\mathcal{U}}$. Otherwise, DecPKBE returns $\perp$.[12]

(GenKeySE, EncSE, DecSE) is a tuple that denotes the key generation, encryption and decryption algorithms of an IND-CPA-secure symmetric encryption scheme.

$\mathsf{MAC}(K_{X,Y}, M)$ is a function that calculates the strongly unforgeable Message Authentication Code (MAC) of message M under the shared key $K_{X,Y}$ of X and Y. The result $mac_M \leftarrow \mathsf{MAC}(K_{X,Y}, M)$ is a MAC tuple $mac_M := (M, \mu_M)$ that contains both the message M and the actual MAC μ_M. Two auxiliary functions split the MAC tuple into its two components: $M \leftarrow \mathsf{Msg}(mac_M)$ and $\mu_M \leftarrow \mathsf{Code}(mac_M)$. We denote the verification of mac_M with the key $K_{X,Y}$ with $ind \leftarrow \mathsf{Verify}(K_{X,Y}, \mathsf{Msg}(mac_M), \mathsf{Code}(mac_M))$.

[12] In this report, the message is an SW component s. For efficiency reasons, the sender does not encrypt s, but instead encrypts a symmetric session key under which s is encrypted. However, to simplify the notation we do not explicitly introduce the session key in our notation.

Chapter 4

Security Requirements

We consider the security requirements of each role separately. The following terms will be used in this section: When the installation result is *success*, we mean a complete run of the protocol SWInstallExternal with acceptance indicators $ind^{\mathsf{inst}} = \texttt{true}$ for each of the three protocol participants. This includes installation of a legal SW component and delivery of a legal license. A *legal SW component* is an SW component with a valid SW certificate (output $ind_T^{\mathsf{SW}} = \texttt{true}$ in protocol SWCert) and a valid property statement (correct σ_m^{prop} in protocol SWDistrib). A *legal license* is a license that was generated by L and acquired by U_k, such that the output of both U_k and L in protocol SWInstallExternal is $ind^{\mathsf{inst}} = \texttt{true}$. By *failure* we mean that no SW is installed, i.e., U_k's configuration does not change and SWInstallExternal leads to $ind^{\mathsf{inst}} = \texttt{false}$ for at least one protocol participant. A *legal ISP I_j for a specific SW s_m* has a correct clearance level certificate (output $ind_T^{\mathsf{clear}} = \texttt{true}$ in protocol ClearCert) and a clearance level c matching the minimum clearance level c_m^{min} of s_m, i.e., $c \geq c_m^{\mathsf{min}}$.

4.1 Common Requirements

(COR) Correctness: If all involved parties behave correctly and follow the specified protocols, then the installation result must be success.

We detail what it means for a party to behave correctly: First, all correct parties create correct signatures and MACs. Second, a correct user platform U_k only requests allowed usage rights for legal and compatible SW components. Third, a correct U_k only requests SW from an ISP I_j that is legal for s_m.

4.2 OEM Requirements

(OPE) Policy Enforcement: O requires enforcement of the following policies:

- **(OPE1) Rights Enforcement.** The terms and conditions of O, which define the set of all allowed usage rights $\mathcal{R}_m^{\mathsf{total}}$, should not be circumvented. Formally this means: The output p of protocol SWInstallInternal to the target component u_i must be a correct usage parameter (see Definition 5).
- **(OPE2) Compatibility Enforcement.** If an SW component s_m and a user platform U_k are incompatible (see Definition 6), then any installation attempt of

s_m in U_k must fail. Formally this means that for incompatible s_m and U_k the output ind_k^{inst} of protocol SWInstallExternal to U_k must be $ind_k^{\mathsf{inst}} = \texttt{false}$.

- **(OPE3) ISP Clearance Enforcement.** If an ISP has an insufficient clearance level for an SW component, then any attempt of this ISP to install the SW component in a user platform must fail. For example, this protects the OEM from warranty claims of the user when the user platforms fails after a faulty SW installation carried out by an illegal ISP. Formally this means that the output ind_k^{inst} of protocol SWInstallExternal to user platform U_k may be $ind_k^{\mathsf{inst}} = \texttt{true}$ only if ISP I_j is legal for the installed SW component s_m.

(OCF) Confidentiality: No party except O and the trusted component u_0 of U_k may be capable of reading SW developed by O in cleartext *prior to* installation. This is meant to protect the intellectual property contained in the SW. For example, S may not be capable of copying an SW component of O and subsequently certifying it as its own product. Formally this means: The SW installation packages σ^{inst} and mac^{inst}, which contain the SW s_m in protected form, must at least provide IND-CCA1 security.[1]

However, we only consider conditional access to the SW. Complementary measures, e.g., fingerprinting [BS95, CKLS97, Her03, KK04], are beyond the scope of this report. Note that requirement OCF also excludes I_j from reading the cleartext. However, I_j will still be necessary in most installation procedures because it possesses assets such as the necessary skills, installation equipment, maintenance area and spare parts.

(OI) Integrity: The installed SW component must be unmodified. Formally this means that no successful installation may simultaneously fulfill the following two conditions:
1. User platform U_k runs protocol SWInstallExternal with index m of SW s_m as input.
2. Protocol SWInstallInternal outputs a different SW s to u_i, that is $s \neq s_m$.

4.3 SAP Requirements

S shares all requirements with O; the role O and the letter "O" simply need to be replaced by S and "O" where necessary. However, S has one additional requirement:

(SND) Non-discrimination: The identity of an SAP S_d may neither influence S_d's ability to send over the broadcast channel nor the result of the installation procedure. For example, U_k may not be manipulated in such a way that it only accepts SW from specific SW providers. We formalize: Let s_m be an SW component that is legal, compatible with a user platform U_k and developed by SAP $S_d = P_d$. Further, let I_j be a legal ISP for s_m and the rights set $\mathcal{R}_m$ be allowed. Then if U_k, I_j and L behave correctly, the installation result of protocol SWInstallExternal must be success, no matter what the adversarial coalition $\mathcal{P} \setminus \{P_d\}$ does.

[1] The PKBE scheme of [DF03] only provides IND-CCA1 security, although the authors assume it to be IND-gCCA2-secure. We therefore restrict our ambitions to IND-CCA1 security. In addition, we cannot hope to achieve IND-CCA1 security *after* σ^{conf} is generated because the cleartext of any SW component s_m is contained in the target component u_i. In contrast to other applications, e.g., secure e-mail, s_m cannot be stored as ciphertext in u_i because u_i is too weak to perform decryption on the fly. However, the attacker at least needs to get access to a user platform U_k in which s_m is installed.

4.4 ISP Requirements

(INR) Non-repudiation: If the installation result is success, then the ISP must be able to prove origin and result of the installation to any honest party. We formalize: Let U_k, I_j and L run the protocol SWInstallExternal with result success. Then I_j must obtain proof of the following two statements:

1. U_k initiated the protocol run.
2. The installation result is success.

(ICE) Clearance Enforcement: This requirement is identical to OPE3. For example, this justifies an ISP's effort to obtain a clearance level certificate.

(IND) Non-discrimination: A legal I_j must be able to install any legal and compatible SW component that U_k requests. For example, the SW provider may not be able to separate ISPs with an identical clearance level into subgroups and exclude individual subgroups from the SW installation process. We formalize: Let s_m be an SW component that is legal and compatible with a user platform U_k. Further, let I_j be a legal ISP for s_m and $\mathcal{R}_m$ be an allowed rights set. Then if the parties U_k and L behave correctly, the installation result must be success.

(IFP) Frame-Proofness: If the installation result is failure, then the ISP may not be wrongly accused of treachery, e.g., of having installed SW. We assume the burden of proof is on the accuser. We formalize: Let the result of a run of protocol SWInstallExternal be failure. Then there may be no adversary $\mathcal{A}$ that proves to an honest verifier the installation result success for the same protocol run. The required proof is σ^{ack}.

4.5 License Provider Requirements

(LNR) Non-repudiation: The receivers of a legal license cannot repudiate its receipt. For example, U_k cannot receive a legal license and later refuse payment. We formalize: Let the output value ind_L^{inst} of the installation protocol SWInstallExternal to L be $ind_L^{\mathsf{inst}} = \texttt{true}$. Further, let the receiver(s), i.e., I_j or the owner of U_k or both, claim that the installation result was $ind^{\mathsf{inst}'} = \texttt{false}$. Finally, let the actual installation result be $ind^{\mathsf{inst}} \in \{\texttt{true}, \texttt{false}\}$. Then any honest verifier $\mathcal{V}$ must decide in favor of the actual installation result ind^{inst}.

4.6 User Requirements

(UNR) Non-repudiation: After the installation procedure, U_k must be able to prove the result, i.e., either success or failure, to any honest party. We formalize: Let the installation result be $ind^{\mathsf{inst}} \in \{\texttt{true}, \texttt{false}\}$, where `true` and `false` represent success and failure respectively. Then U_k must obtain the correct output value $ind_k^{\mathsf{inst}} = ind^{\mathsf{inst}}$, such that any honest verifier accepts the correctness of ind_k^{inst}.

(UIO) Installation Origin: No SW installation may be performed without request by U_k. For example, this prevents a malicious I_j from installing any SW without prior consent by U_k's owner. We formalize: Any installation attempt in a correct user platform U_k

must fail if U_k does not initiate the corresponding run of protocol SWInstallExternal with an SW installation request req^{inst}.

(UA) Authenticity: The installed SW component and the license must be authentic. We formalize both aspects of UA: Let a run of protocol SWInstallExternal with input value m of U_k and output value s to u_i result in success. Then s must equal s_m and the license σ^{lic} used in this protocol run must have originated from L.

Chapter 5

Proposed Solution

5.1 Overview

This section provides a summary of the proposed installation procedure (see Figure 5.1). The procedure consists of a setup period (Phases A–D) and the actual installation (Steps 1–6). The protocols for these two parts will be detailed in Section 5.3. Section 5.2 introduces the trust and communication channel assumptions on which the protocols rely. Finally, we formally analyze the security of our proposed solution in Section 5.4.

In the setup period, the system parameters, e.g., security parameters of the cryptographic schemes, are chosen; all cryptographic keys are generated and properly distributed. Then each I_j requests a clearance level certificate from T and obtains it if he is compliant with the requested clearance level. This certification is performed once per I_j and repeated only if existing certificates expire. In parallel, each SW provider requests an SW certificate from T for each of his SW components. After SW certification, the SW provider establishes terms and conditions with L. Both steps need to be done for each SW component.[1] Finally, each P_d distributes his SW components to the members of $\mathcal{I}$ via the broadcast channel.

The actual installation starts with the selection of an SW component by U_k. If U_k confirms the SW to be appropriate, then U_k sends an installation request to I_j. I_j obtains a license from L. After delivery of SW and license to U_k, U_k checks whether the license is legal. If so, the trusted component $u_{k,0}$ instructs the target component $u_{k,i}$ to install the SW. U_k then confirms the successful installation to I_j and awaits I_j's acknowledgement. After receiving the acknowledgement, $u_{k,0}$ instructs $u_{k,i}$ to use the SW.

5.2 Assumptions

Although we neither explicitly mention expiry dates nor random nonces, we assume they are used in any implementation of the system. Otherwise, privileges cannot be withdrawn and replay attacks become possible. For example, an adversary might reuse a signature from a previous protocol run, which the legitimate party wouldn't create in the current protocol run.

In addition, we omit the discussion of identity fraud which is not yet relevant in the idealized model. In any implementation, however, identity fraud has to be prevented with a verification by the receiving party that the sending party and her signing key match. For

[1] However, an SW provider and the license provider might establish more general terms and conditions that cover a whole set of SW components.

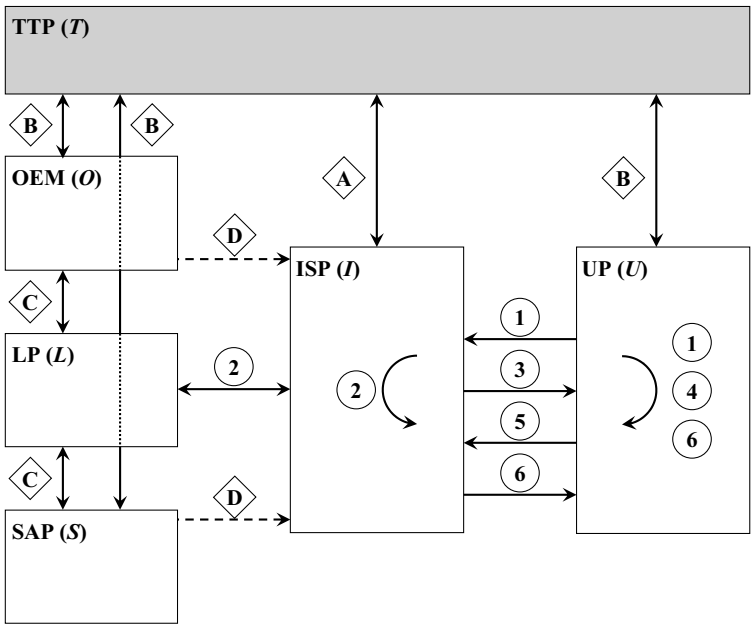


Figure 5.1: Installation procedure in six steps

example, a clearance level certificate of I_j might contain name and address of I_j, thus allowing the owner of U_k to verify the match between certificate and ISP.

5.2.1 Trust Relations

All honest parties are assumed to keep their secrets private, e.g., signature keys. There are no specific trust assumptions for the members of $\mathcal{P}$ and $\mathcal{I}$. For the other roles we assume:

L: All SW providers trust L to adhere to their terms and conditions.[2]

U: All parties trust the members of $\mathcal{U}$ to (i) keep their SW confidential, (ii) comply with the specified protocols and (iii) adhere to licenses. Due to the cost pressure that embedded systems have to experience we cannot assume each component of an arbitrary U_k to be fully trusted. Therefore, we distinguish between two types of components as introduced in Section 3.1: $u_{k,0}$ is fully trusted and the $u_{k,i}$, $1 \leq i \leq n_k$ are only partially trusted.

Every party trusts $u_{k,0}$, which securely stores U_k's private keys. $u_{k,0}$ verifies the trustworthiness of the $u_{k,i}$ via a shared secret $K_{0 \leftrightarrow i}$: $u_{k,0}$ only sends SW to a component $u_{k,i}$ that possesses a secret key $K_{0 \leftrightarrow i}$ embedded by $u_{k,i}$'s manufacturer. There are several ways for $u_{k,0}$ to obtain this key. For example, $u_{k,0}$ may receive it by means of a certificate from the manufacturer in which the symmetric key is encrypted with $u_{k,0}$'s public key $K_{k,0}^{\mathsf{pub}}$ in an asymmetric encryption scheme EncAE:

$$\mathsf{Sign}(K_{Manuf}^{\mathsf{sign}}, \mathsf{EncAE}(K_{k,0}^{\mathsf{pub}}, K_{0 \leftrightarrow i}))$$

[2] This implies a high level of trust in L. It might be reduced to a lower trust level by introducing techniques for tracing an L that violates the terms and conditions, e.g., double spender detection. However, due to the focus on secure SW delivery and installation, we omit any advanced licensing techniques.

We highlight further implementation aspects of $u_{k,0}$, e.g., sealing of the PKBE keys with the properties of a correct $u_{k,0}$, in Appendix C.

The components $u_{k,i}$, $1 \leq i \leq n$, are not fully trusted, but receive appropriate and cost-efficient protection measures. At production time, they are trusted and receive the shared secret $K_{0 \leftrightarrow i}$ from their manufacturer. We assume the $u_{k,i}$ to delete this key and their memory if an adversary tries to steal their data with an attack on HW or SW; after key deletion, they are no longer trusted. Depending on the commercial value of the SW they contain, the $u_{k,i}$ receive different protection measures, e.g., tamper-resistant memory for the most valuable components,[3] but only minimal protection for low-value components. Finally, we assume the $u_{k,i}$ to be reliable, i.e., complete an installation request in limited time. The Trusted Computing Base (TCB) of U_k thus comprises $u_{k,0}$ and the deletion mechanism of the $u_{k,i}$.

T: Every party trusts T. For example, this includes correct certification of SW properties and clearance levels as well as correct publishing of all public keys.

5.2.2 Communication channels

The communication channels are represented in Figures 3.1 and 3.2. All of them are assumed to preserve *integrity*, thus avoiding bit errors. In addition, all but two channels are assumed to be *secure*, i.e., authentic and confidential. The first exception is the one-way broadcast channel, which is neither authentic nor confidential. However, it is *non-discriminatory*: (i) all SW providers can send over the channel and (ii) the channel has global reach to all members of $\mathcal{I}$. The second exception is U_k's internal communication network. Due to cost constraints on U_k's components, it is only assumed to provide integrity and reliability. By *reliable* we mean that each message reaches its recipient after a limited amount of time.[4] Finally, the channels between L and I_j as well as between I_j and U_k are assumed to be reliable.

5.3 Protocols

5.3.1 Setup

The setup period starts with the SystemSetup protocol, in which T sets up the PKBE scheme, publishes the public keys and provides each U_k with his private key set. In addition, every party generates a private signing key and provides all other parties with the public test key, e.g., using T to certify and distribute the public keys. The setup period further consists of four phases A–D (see Figure 5.1):

Phase A: Each ISP I_j applies for certification of a particular clearance level c. We show the implementation of the certification protocol ClearCert in Figure 5.2, where the operator $\langle\rangle$ denotes the evaluation of a Boolean statement, e.g., $\langle 5 > 3 \rangle =$ `true`, and $\wedge$ denotes the Boolean `and` operator.

Phase B: Each SW provider P_d sends SW requests to T in order to obtain SW certificates and integrity proofs for his SW components. We show the implementation of the certification protocol SWCert in Figure 5.3.[5]

[3] In the automotive example, this might be the ECUs of airbags or the ESP.

[4] In a practical implementation, this amount should be in the order of hours or lower.

[5] Based on a comment of an ISPEC 2005 reviewer, we made a modification to the SW submission procedure

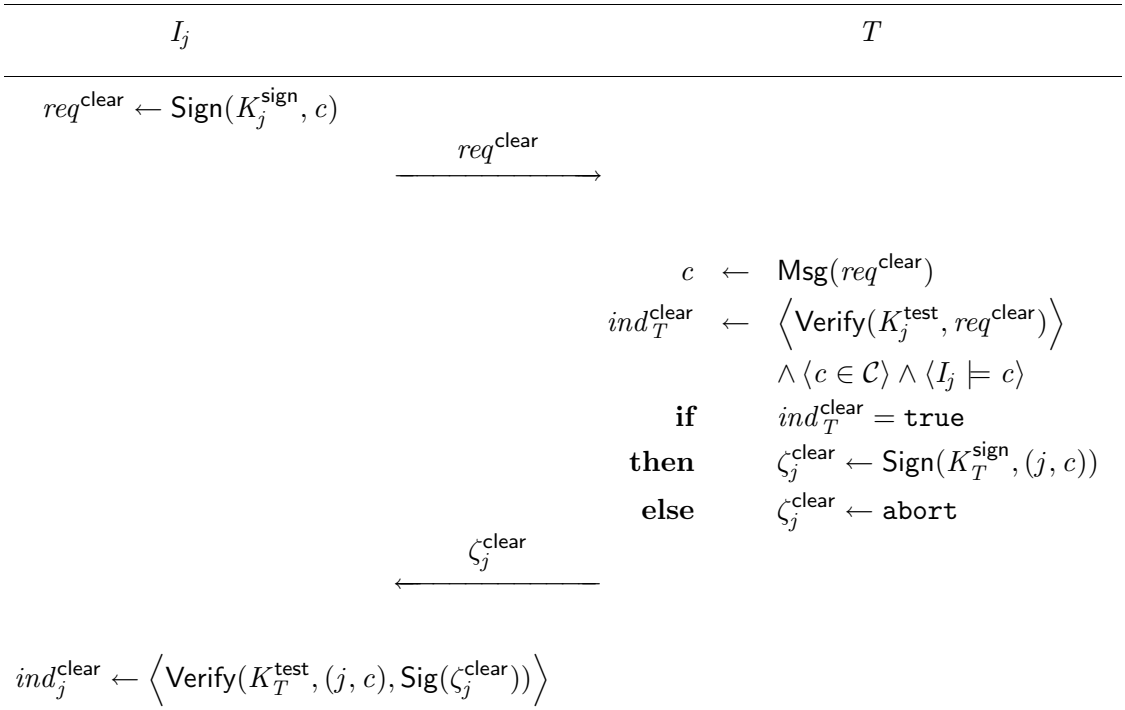


Figure 5.2: Implementation of protocol ClearCert

Phase C: Each SW providers P_d negotiates the terms and conditions for all his SW components with the license provider L. As the negotiation is implementation-specific, we omit any details on the input values and messages exchanged during the protocol RightsDef. Nevertheless, we state that each protocol run provides both participants with the set $\mathcal{R}_m^{\mathsf{total}}$ of all allowed usage rights for SW component s_m.

Phase D: Each SW providers P_d distributes his SW components to the ISPs $\mathcal{I}$. We show the implementation of the SW distribution protocol SWDistrib in Figure 5.4. P_d encrypts[6] the SW component s_m together with T's signature on (m, s_m). Adding the signature will allow the user platform to verify the authenticity of s_m, i.e., the fact that the deciphered SW component is equal to the SW component that T had certified. P_d then signs a statement on the properties of s_m. Finally, he broadcasts the encrypted SW component s_m^{enc} together with his property statement σ_m^{prop} and the SW certificate ζ_m^{SW}.

in order to eliminate a potential security vulnerability. The published paper [AHS05] included a hash-based authenticity check leading to a parallel encrypt-and-sign scheme. We have replaced it with a more secure sign-then-encrypt scheme based on [ADR02].

[6]We stress that the broadcast encryption scheme protects s_m between P_d and $\mathcal{U} \backslash \mathcal{R}_{\mathcal{U}}$, although the broadcast channel only leads from P_d to $\mathcal{I}$. The corresponding I_j executes the final delivery of the broadcast message to the respective U_k.

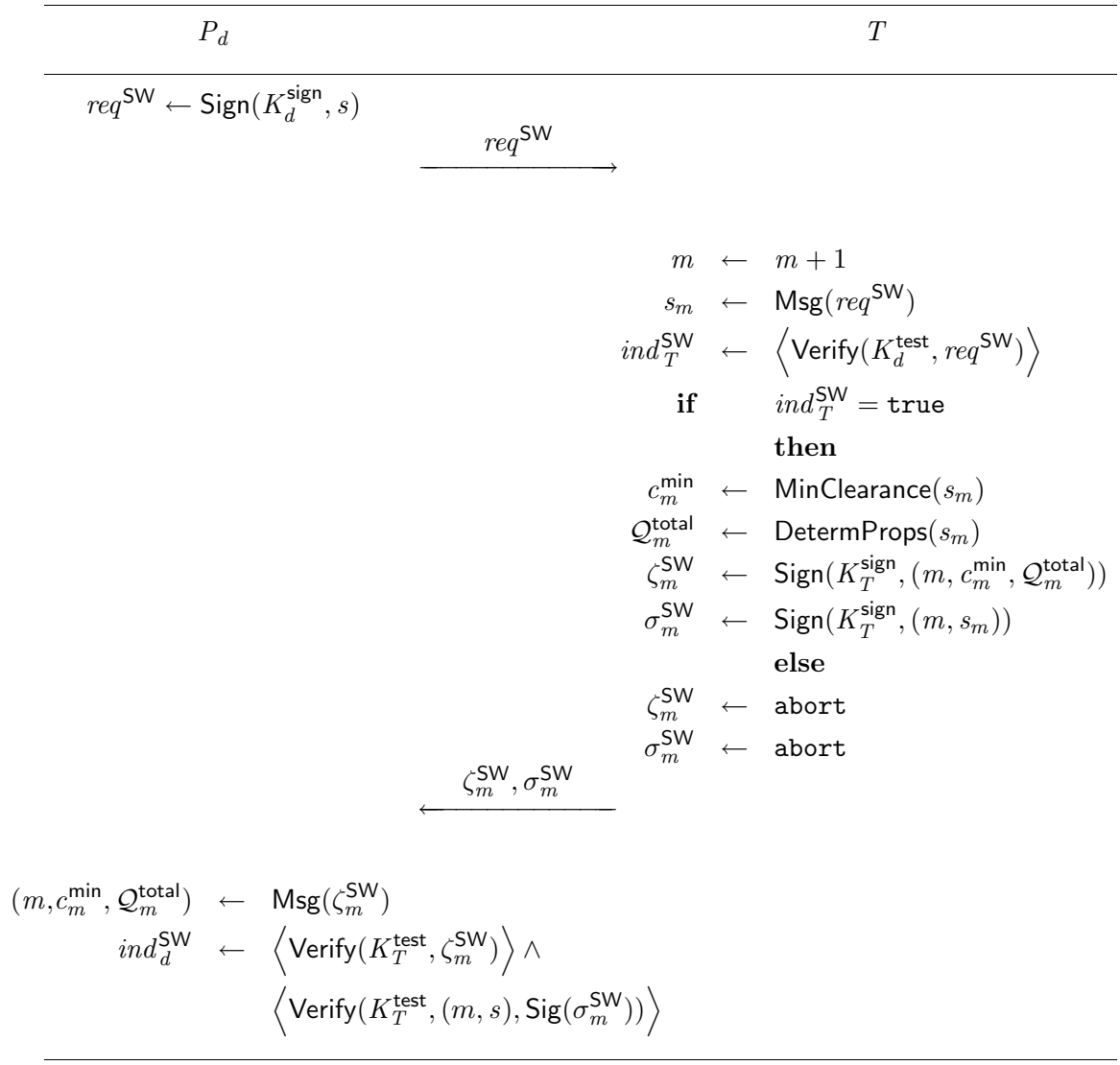


Figure 5.3: Implementation of protocol SWCert

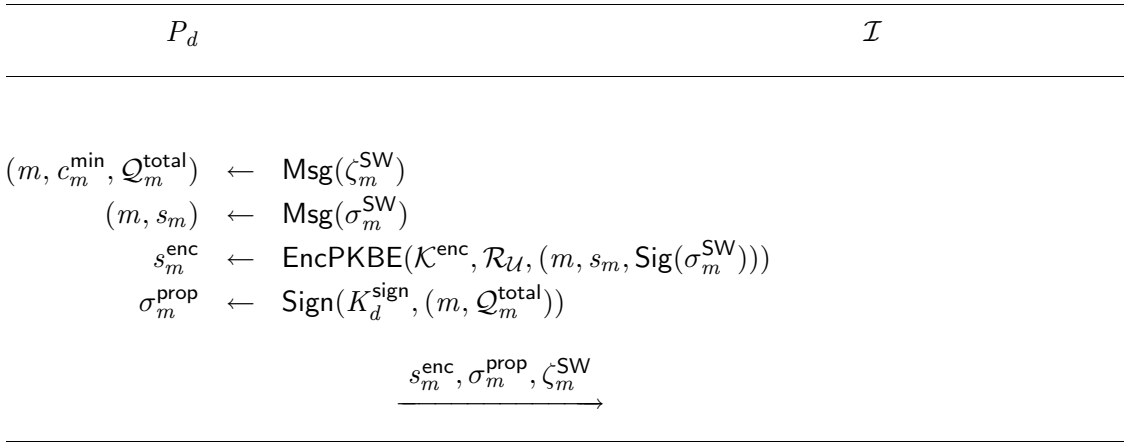


Figure 5.4: Implementation of protocol SWDistrib

5.3.2 Installation of an SW Component

After the setup phase, the installation procedure for a specific SW component can start. We show the implementation of protocol SWInstallExternal on pages 29 and 30. The following steps explain the protocol; their numbering corresponds with Figure 5.1. Throughout the protocol, we generically use U_k to indicate the target platform; however, unless otherwise specified, all calculations of U_k are actually performed by the trusted component u_0 of U_k:

1. The user platform U_k obtains as input the index m of the SW component s_m that is to be installed.[7] As additional input, U_k obtains the SW certificate and the property statement of s_m and the clearance certificate of I_j. Based on this input, U_k verifies that s_m is legal and compatible and that I_j is legal for s_m. If the verification succeeds, U_k sends an SW installation request req^{inst} to I_j.[8]

   Note that u_0 of U_k tracks the properties $\mathcal{Q}_k$ of platform U_k and updates them after each SW installation.

2. I_j passes the request on to L in order to obtain a license. L verifies that the requested rights are allowed. If so, L creates the license σ^{lic} and sends it to I_j. I_j signs the protected SW component s_m^{enc} as well as the license σ^{lic} to create the installation package σ^{inst}.

[7] How the SW component is selected is implementation-specific. In a practical implementation, either the owner of U_k or the ISP are likely to actually perform the selection in U_k's place. The owner may be interested in a particular s_m because he wants to enhance U_k's functionality. The ISP may recommend an installation of s_m because a defective SW component needs to be replaced. Several procedures for initiating the installation request can be implemented depending on the trust model. In a classical procedure, the owner might physically sign a paper-based installation request and trust I_j to initiate the correct installation request. In a more technical procedure, the owner might initiate the request himself after identifying himself to u_0 of U_k with the help of a smartcard.

[8] Compared to [AHS05], these verifications have moved from Step 4 to 1. The reason is that now U_k cannot issue an unnecessary SW request of an inappropriate SW component. This in turn saves I_j from requesting an unnecessary license and delivering an unnecessary installation package.

3. I_j sends the installation package σ^{inst} to U_k.

4. U_k verifies the validity of the SW installation package regarding correctness and authenticity. During verification, u_0 needs to decrypt the protected SW component s_m^{enc} with its private PKBE keys. If the verification succeeds, then U_k initiates the protocol SWInstallInternal. Finally, U_k generates a confirmation σ^{conf} of the installation result.

5. U_k sends the installation confirmation σ^{conf} to I_j.

6. I_j sends an acknowledgement back to U_k. The acknowledgement σ^{ack} is the indicator for U_k that I_j received the installation confirmation. Afterwards, U_k actually uses the SW component s_m. Specifically, u_0 of U_k instructs u_i to use s_m with a correct usage parameter p as part of protocol SWInstallInternal.

   We show the details of subprotocol SWInstallInternal on page 31. Note that this subprotocol only uses symmetric cryptography. Specifically, MACs and symmetric encryption replace signatures and asymmetric encryption respectively.

The trusted component u_0 stores all licenses and periodically checks if any of them has expired. When a license expires, u_0 recalculates the correct usage parameter p and tells u_i to execute the SW with this new parameter. For example, the new parameter might instruct u_i to completely stop using the SW or switch off some functionality.

5.4 Security Analysis for Proposed Solution

The security aspects of the proposed solution will be informally analyzed by verifying the requirements as defined in Section 4. For each requirement, the corresponding steps of the installation procedure will be indicated. Whenever we refer to u_0, we mean a correct and unrevoked $u_{k,0}$ of U_k. By u_i we mean $u_{k,i}$ of U_k.

5.4.1 Common

(COR) Correctness: Let all parties behave correctly and execute the specified protocols of the setup period. Further, let the participants of protocol SWInstallExternal behave correctly. We show that under these two assumptions, a run of protocol SWInstallExternal results in success, which is equivalent to all acceptance indicators ind^{inst} being equal to `true`. To simplify the argument, we note that all signature and MAC verifications of the form $\langle \mathsf{Verify}() \rangle$ must succeed because the signatures and MACs have been generated by correct parties and sent over communication channels that preserve integrity. Further, all index and value comparisons of the form $\langle x = y \rangle$ must succeed because the verifying parties use matching certificates and statements with correct index.

The acceptance indicator $ind_k^{\mathsf{inst}''}$ must equal `true` because s_m and U_k are compatible and I_j is legal for s_m. The acceptance indicator ind_L^{inst} must equal `true` because U_k only requests allowed usage rights. The acceptance indicator $ind_k^{\mathsf{inst}'}$ must equal `true` because $ind_k^{\mathsf{inst}''}$ equaled `true` and SWInstallInternal only contains MAC verifications prior to the generation of $ind_0^{\mathsf{inst}'}$. ind_j^{inst} must equal `true` because $ind_k^{\mathsf{inst}'}$ equaled `true`. Finally, ind_k^{inst} must equal `true` because $ind_k^{\mathsf{inst}'}$ equaled `true`.

Implementation of protocol SWInstallExternal

U_k	I_j	L

$$
\begin{aligned}
(m', \mathcal{Q}_m^{\mathsf{total}}) &\leftarrow \mathsf{Msg}(\sigma_m^{\mathsf{prop}}) \\
(m'', c_m^{\mathsf{min}}, \mathcal{Q}_m^{\mathsf{total}\prime}) &\leftarrow \mathsf{Msg}(\zeta_m^{\mathsf{SW}}) \\
(j', c) &\leftarrow \mathsf{Msg}(\zeta_j^{\mathsf{clear}}) \\
ind_k^{\mathsf{inst}\prime\prime} &\leftarrow \left\langle \mathsf{Verify}(K_d^{\mathsf{test}}, \sigma_m^{\mathsf{prop}}) \right\rangle \wedge \\
&\quad \left\langle \mathsf{Verify}(K_T^{\mathsf{test}}, \zeta_m^{\mathsf{SW}}) \right\rangle \wedge \\
&\quad \left\langle \mathsf{Verify}(K_T^{\mathsf{test}}, \zeta_j^{\mathsf{clear}}) \right\rangle \wedge \\
&\quad \left\langle m = m' \right\rangle \wedge \left\langle m = m'' \right\rangle \wedge \\
&\quad \left\langle \mathcal{Q}_m^{\mathsf{total}} = \mathcal{Q}_m^{\mathsf{total}\prime} \right\rangle \wedge \\
&\quad \left\langle \mathsf{CompatTest}(\mathcal{Q}_m^{\mathsf{total}}, \mathcal{Q}_k) \right\rangle \wedge \\
&\quad \left\langle j' = j \right\rangle \wedge \left\langle c \geq c_m^{\mathsf{min}} \right\rangle \\
\textbf{if} \quad & ind_k^{\mathsf{inst}\prime\prime} = \texttt{true} \\
\textbf{then} & \\
req^{\mathsf{inst}} &\leftarrow \mathsf{Sign}(K_k^{\mathsf{sign}}, (k, m, \mathcal{R}_m)) \\
\textbf{else} & \\
\textbf{abort} &
\end{aligned}
$$

$\xrightarrow{req^{\mathsf{inst}}}$

$\xrightarrow{req^{\mathsf{inst}}}$

$$
\begin{aligned}
(k, m, \mathcal{R}_m) &\leftarrow \mathsf{Msg}(req^{\mathsf{inst}}) \\
ind_L^{\mathsf{inst}} &\leftarrow \left\langle \mathsf{Verify}(K_k^{\mathsf{test}}, req^{\mathsf{inst}}) \right\rangle \\
&\quad \wedge \left\langle \mathcal{R}_m \subseteq \mathcal{R}_m^{\mathsf{total}} \right\rangle \\
\textbf{if} \quad & ind_L^{\mathsf{inst}} = \texttt{true} \\
\textbf{then} & \\
\sigma^{\mathsf{lic}} &\leftarrow \mathsf{Sign}\Big(K_L^{\mathsf{sign}}, \\
&\quad (k, m, \mathcal{R}_m)\Big) \\
\textbf{else} & \\
\sigma^{\mathsf{lic}} &\leftarrow \texttt{abort}
\end{aligned}
$$

$\xleftarrow{\sigma^{\mathsf{lic}}}$

Continuation of protocol SWInstallExternal

U_k	I_j	L

$$\xleftarrow{\sigma^{\mathsf{lic}}}$$

$$\sigma^{\mathsf{inst}} \leftarrow \mathsf{Sign}(K_j^{\mathsf{sign}}, (s_m^{\mathsf{enc}}, \sigma^{\mathsf{lic}}))$$

$$\xleftarrow{\sigma^{\mathsf{inst}}}$$

$$\begin{aligned}
(s_m^{\mathsf{enc}}, \sigma^{\mathsf{lic}}) &\leftarrow \mathsf{Msg}(\sigma^{\mathsf{inst}}) \\
(m''', s, sig_m^{\mathsf{SW}}) &\leftarrow \mathsf{DecPKBE}(\mathcal{K}_k^{\mathsf{dec}}, s_m^{\mathsf{enc}}) \\
i &\leftarrow \mathsf{Target}(\mathcal{Q}_m^{\mathsf{total}}, \mathcal{Q}_k) \\
ind_k^{\mathsf{inst}\prime} &\leftarrow ind_k^{\mathsf{inst}\prime\prime} \wedge \left\langle m''' = m \right\rangle \wedge \\
&\quad \left\langle \mathsf{Verify}(K_j^{\mathsf{test}}, \sigma^{\mathsf{inst}}) \right\rangle \wedge \\
&\quad \left\langle \mathsf{Verify}(K_L^{\mathsf{test}}, (k, m, \mathcal{R}_m), \mathsf{Sig}(\sigma^{\mathsf{lic}})) \right\rangle \wedge \\
&\quad \left\langle \mathsf{Verify}(K_T^{\mathsf{test}}, (m, s), sig_m^{\mathsf{SW}}) \right\rangle \wedge \\
&\quad \left\langle \mathsf{SWInstallInternal} : ind_0^{\mathsf{inst}\prime} = \mathtt{true} \right\rangle \\
\sigma^{\mathsf{conf}} &\leftarrow \mathsf{Sign}(K_k^{\mathsf{sign}}, (\sigma^{\mathsf{lic}}, ind_k^{\mathsf{inst}\prime}))
\end{aligned}$$

$$\xrightarrow{\sigma^{\mathsf{conf}}}$$

$$\begin{aligned}
(\sigma^{\mathsf{lic}\prime}, ind_k^{\mathsf{inst}\prime}) &\leftarrow \mathsf{Msg}(\sigma^{\mathsf{conf}}) \\
ind_j^{\mathsf{inst}} &\leftarrow ind_k^{\mathsf{inst}\prime} \wedge \\
&\quad \left\langle \mathsf{Verify}(K_k^{\mathsf{test}}, \sigma^{\mathsf{conf}}) \right\rangle \\
&\quad \textbf{if} \quad ind_j^{\mathsf{inst}} = \mathtt{true} \\
&\quad \textbf{then} \\
\sigma^{\mathsf{ack}} &\leftarrow \mathsf{Sign}(K_j^{\mathsf{sign}}, \sigma^{\mathsf{conf}}) \\
&\quad \textbf{else} \\
\sigma^{\mathsf{ack}} &\leftarrow \mathtt{abort}
\end{aligned}$$

$$\xleftarrow{\sigma^{\mathsf{ack}}}$$

$$\begin{aligned}
ind_k^{\mathsf{inst}} &\leftarrow ind_k^{\mathsf{inst}\prime} \wedge \left\langle \mathsf{Verify}(K_j^{\mathsf{test}}, \sigma^{\mathsf{conf}}, \mathsf{Sig}(\sigma^{\mathsf{ack}})) \right\rangle \\
\mathcal{Q}_k &\leftarrow \mathsf{UpdateProps}(\mathcal{Q}_m^{\mathsf{total}}, \mathcal{Q}_k)
\end{aligned}$$

Implementation of protocol SWInstallInternal

u_0			u_i		
s_i^{enc}	$\leftarrow$	$\mathsf{EncSE}(K_{0\leftrightarrow i}, s)$			
mac^{inst}	$\leftarrow$	$\mathsf{MAC}(K_{0\leftrightarrow i}, (i, m, s_i^{\mathsf{enc}}))$			
		$\xrightarrow{mac^{\mathsf{inst}}}$			
			$(i', m, s_i^{\mathsf{enc}})$	$\leftarrow$	$\mathsf{Msg}(mac^{\mathsf{inst}})$
			ind_i^{inst}	$\leftarrow$	$\langle \mathsf{Verify}(K_{0\leftrightarrow i}, (i, m, s_i^{\mathsf{enc}}),$
					$\mathsf{Code}(mac^{\mathsf{inst}}))\rangle$
			if		$ind_i^{\mathsf{inst}} = \mathtt{true}$
					then
			s	$\leftarrow$	$\mathsf{DecSE}(K_{0\leftrightarrow i}, s_i^{\mathsf{enc}})$
			mac^{conf}	$\leftarrow$	$mac(K_{0\leftrightarrow i}, (0, m))$
					else
			mac^{conf}	$\leftarrow$	$mac(K_{0\leftrightarrow i}, (0, \mathtt{abort}))$
		$\xleftarrow{mac^{\mathsf{conf}}}$			
$ind_0^{\mathsf{inst}\prime}$	$\leftarrow$	$\langle \mathsf{Verify}(K_{0\leftrightarrow i}, (0, m), \mathsf{Code}(mac^{\mathsf{conf}}))\rangle$			
ind_0^{inst}	$\leftarrow$	$ind_0^{\mathsf{inst}\prime} \wedge$			
		$\langle \mathsf{SWInstallExternal} : ind_k^{\mathsf{inst}} = \mathtt{true}\rangle$			
if		$ind_0^{\mathsf{inst}} = \mathtt{true}$			
		then			
		Calculate correct p			
mac^{ack}	$\leftarrow$	$mac(K_{0\leftrightarrow i}, (i, m, p))$			
		else			
mac^{ack}	$\leftarrow$	$mac(K_{0\leftrightarrow i}, (i, m, \mathtt{abort}))$			
		$\xrightarrow{mac^{\mathsf{ack}}}$			
			(i', m', p)	$\leftarrow$	$\mathsf{Msg}(mac^{\mathsf{ack}})$
			ind_i^{inst}	$\leftarrow$	$ind_i^{\mathsf{inst}} \wedge$
					$\langle \mathsf{Verify}(K_{0\leftrightarrow i}, (i, m, p),$
					$\mathsf{Code}(mac^{\mathsf{ack}}))\rangle$

5.4.2 OEM

(OPE1) Rights Enforcement: We assume to the contrary of OPE1 that there exists at least one run of protocol SWInstallExternal which leads to u_i using s_m with an incorrect usage parameter. To prove OPE1, we lead this assumption to a contradiction. The incorrect usage parameter has two possible explanations: Either u_i is compromised or it received an authentic usage instruction (i, m, p) with incorrect usage parameter p from u_0.[9] Due to the trust assumptions, u_i can't be compromised: Neither u_i nor u_0 disclose the shared key and therefore u_i can't be maliciously reprogrammed. Thus u_i must have received an incorrect usage parameter p from u_0. However, this can again only have two explanations: u_0 either incorrectly processed the usage rights during calculation of mac^{ack} or it correctly processed disallowed usage rights. Due to the trust assumptions, u_0 cannot process the usage rights incorrectly. Thus u_0 must have received disallowed usage rights. Due to the calculation of ind_L^{inst}, the disallowed usage rights must have been approved by L. However, due to the trust assumptions, L cannot approve a disallowed rights set. This concludes the proof by contradiction.

(OPE2) Compatibility Enforcement: We suppose to the contrary that the installation of at least one SW component s_m in U_k succeeds although s_m and U_k are incompatible (see Definition 6 on page 12). To prove OPE2, we lead this assumption to a contradiction. As the installation result is success, the output ind_k^{inst} of protocol SWInstallExternal to U_k must have equaled `true`. Due to the calculation of ind_k^{inst}, the intermediate results $ind_k^{\mathsf{inst}'}$ and $ind_k^{\mathsf{inst}''}$ must also have equaled `true`. Due to the calculation of $ind_k^{\mathsf{inst}''}$, the compatibility test $\mathsf{CompatTest}(\mathcal{Q}_m, \mathcal{Q}_k)$ must have generated output `true`. Due to the correctness of u_0, the compatibility test can only have generated the incorrect output `true` due to an incomplete property set $\mathcal{Q}_m \subset \mathcal{Q}_m^{\mathsf{total}}$ (see Definition 10 on page 12). Due to the authentication of $\mathcal{Q}_m$ as part of $ind_k^{\mathsf{inst}''}$, the incomplete property set must have originated from T. However, any property set from T must be complete due to the correctness of T. This concludes the contradiction.

(OPE3) ISP Clearance Enforcement: We suppose to the contrary of OPE3 that a dishonest ISP I_j succeeds in installing at least one SW component s_m in at least one user platform U_k although I_j is not legal for s_m. As argued in OPE2, the intermediate result $ind_k^{\mathsf{inst}''}$ of protocol SWInstallExternal must have been `true`. Due to the calculation of $ind_k^{\mathsf{inst}''}$, u_0 must have accepted $c \geq c_m^{\mathsf{min}}$. As u_0 is correct, it must have received incorrect clearance level information c or c_m^{min} (or both). Due to the authentication of these values in the calculation of $ind_k^{\mathsf{inst}''}$, the incorrect clearance level information must have originated from T. However, any clearance information from T must be correct due to the trust in T and the verifications in the two protocols ClearCert and SWCert. This concludes the proof by contradiction.

(OCF) Confidentiality: Intuitively, the privacy of an SW component is protected by the IND-CCA1 security of the PKBE scheme in protocol SWDistrib and the IND-CPA security of the symmetric encryption scheme in protocol SWInstallInternal. Only an unrevoked

[9] We stress the importance of the assumptions in Section 5.2 regarding replay attacks. If in contrast to the assumptions a MAC cannot be verified to be fresh, the adversary can replay an old MAC and thus activate usage with an expired usage parameters.

trusted component can decrypt the PKBE ciphertext; only trusted and target component can decrypt the symmetric ciphertext. Formally, we have to prove:

(i) The joint signature and encryption of SW component s_m, consisting of signature σ_m^{SW} from protocol SWCert and encryption s_m^{enc} from protocol SWDistrib, at least preserves the IND-CCA1 security of the PKBE encryption scheme.

(ii) The encrypt-then-mac method in protocol SWInstallInternal increases the IND-CPA security of the symmetric encryption scheme at least to IND-CCA1 security.

Following the notation in [ADR02], we prove (i) with the following theorem. The corresponding proof is detailed in Appendix D:

Theorem 1 *If $\mathcal{E}$ is an* IND-CCA1*-secure encryption scheme and $\mathcal{S}$ is a* UF-CMA*-secure signature scheme, then $\mathcal{S}t\mathcal{E}$ is* IND-CCA1*-secure in the Insider-security model.* □

The authors of [BN00] prove (ii). Specifically, Theorem 4.9 of [BN00] proves that an IND-CPA-secure encryption scheme together with a strongly unforgeable MAC scheme leads to an IND-CCA2-secure encrypt-then-MAC message authentication scheme.

(OI) Integrity: The proof is by contradiction. To the contrary of OI, let the installation result be success and the installed SW s in protocol SWInstallInternal be different from the SW s_m with index m that U_k requested in protocol SWInstallExternal. As s and s_m are different, u_i must either have been incorrect or correctly processed an incorrect s_i^{enc}. Due to the trust assumptions, u_i has either been correct or dysfunctional after key deletion. As the installation result was success ($ind_k^{\mathsf{inst}} = \texttt{true}$ and thus ind_0^{inst} as well as $ind_i^{\mathsf{inst}} = \texttt{true}$), u_i must have been correct and s_i^{enc} incorrect. Due to the calculation of ind_i^{inst} in protocol SWInstallInternal, s_i^{enc} must have originated from u_0. This gives again two possibilities: u_0 is incorrect or correctly processed an incorrect input s_m^{enc}. Due to the trust assumptions, we can exclude an incorrect u_0. Due to the calculation of $ind_k^{\mathsf{inst}'}$, u_0 rejects s_m^{enc} unless it is a valid ciphertext and contains a tuple (m, s_m) that has been witnessed and signed by T. If s_m^{enc} is incorrect, then T confirmed an incorrect (m, s_m) in the form of signature σ_m^{SW} in protocol SWCert. However, due to the trust assumptions we can exclude an incorrect T. We have contradicted all possible explanations for $s \neq s_m$.

5.4.3 SW Application Programmer

(SND) Non-discrimination: The proof is almost identical to the proof of COR. The only difference in the assumptions of both requirements is the (in)correctness of the adversarial coalition $\mathcal{P} \setminus \{P_d\}$. Yet the input values to the correctly behaving participants of protocol SWInstallExternal have been generated by the correctly behaving parties P_d and T. The adversarial coalition thus cannot influence the installation result, which together with the arguments for OCR proves SND.

Note that the channel assumptions (see Section 5.2.2) are crucial in this proof. If the channel wasn't public or if the broadcast message didn't reach all members of $\mathcal{I}$, then the proof would not hold. In addition, the public key property of the PKBE scheme complements the public access to the channel itself.

5.4.4 Installation Service Provider

(INR) Non-repudiation: We show how an honest ISP I_j proves both statements to any honest verifier $\mathcal{V}$:

1. Let the adversary $\mathcal{A}$ claim that U_k did not initiate the successful run of protocol SWInstallExternal. As the installation result is success, I_j must have received the SW installation request req^{inst} that U_k has generated. Due to the correctness of L and U_k and the calculation of ind_L^{inst} as well as $ind_k^{\mathsf{inst}\prime}$, $\mathcal{A}$ cannot forge the index k of U_k within req^{inst}. With req^{inst} as proof, I_j convinces any honest $\mathcal{V}$ that U_k signed the tuple $(k, m, \mathcal{R}_m)$ in order to initiate the protocol. This contradicts $\mathcal{A}$.
2. Let the adversary $\mathcal{A}$ claim that the installation result is failure although the actual result is success. As U_k is correct, the installation result is success only if U_k has sent the installation confirmation σ^{conf} and if I_j has acknowledged the receipt of σ^{conf}. I_j thus possesses a non-repudiable proof of the acceptance indicator $ind_k^{\mathsf{inst}\prime}$, which must equal `true` by the initial assumption. Any honest verifier will accept this proof for the installation result success, which contradicts $\mathcal{A}$.

   Note that both proofs strongly rely on the assumptions regarding freshness, non-expiration and identity verification (see Section 5.2). If these assumptions do not hold, then the proof fails as the adversary can combine messages from several protocol runs to deceive the verifier.

(ICE) Clearance Enforcement: The arguments for OPE3 also prove ICE.

(IND) Non-discrimination: The proof is almost identical to the proof of COR. The only difference in the assumptions of both requirements is the (in)correctness of the adversarial coalition $\mathcal{A} = \mathcal{P}$. $\mathcal{A}$ can only influence the result of protocol SWInstallExternal via three input values: ζ_m^{SW}, σ_m^{SW} and s_m^{enc}. The first two input values are output values of protocol SWCert, which ensures correctness of both values through the correctness of T and the restriction of $\mathcal{A}$'s input to the SW component s. The third input value s_m^{enc} is an output value of protocol SWDistrib, where $\mathcal{A}$ has full control over s_m^{enc}. However, $\mathcal{A}$ cannot tamper with any input value in this protocol due to the calculation of $ind_k^{\mathsf{inst}\prime}$, where U_k verifies the correctness of m and s_m using T's signature. $\mathcal{A}$ thus cannot forge any of the three input values, which together with the arguments for COR proves IND.

(IFP) Frame-Proofness: The proof is by contradiction. Let the adversary $\mathcal{A}$ deliver a valid proof σ^{ack} for the installation result success although the actual result was failure and I_j behaved correctly. Thus at least one of the acceptance indicators ind^{inst} must equal `false`. However, this leads to a contradiction for each indicator as long as the trust assumptions on U_k and L as well as the channel assumptions hold:

1. If ind_L^{inst} equals `false`, then the license equals $\sigma^{\mathsf{lic}} = \texttt{abort}$ and thus $ind_k^{\mathsf{inst}\prime} = \texttt{false}$, which implies $\sigma^{\mathsf{ack}} = \texttt{abort}$. The proof of $\mathcal{A}$ is therefore invalid.
2. If ind_j^{inst} equals `false`, then $\sigma^{\mathsf{ack}} = \texttt{abort}$ and $\mathcal{A}$'s proof is invalid.
3. If ind_k^{inst} equals `false`, then either the verification of σ^{ack} failed or $ind_k^{\mathsf{inst}\prime} = \texttt{false}$. Due to the correctness of I_j, the verification of σ^{ack} can only fail if $\sigma^{\mathsf{ack}} = \texttt{abort}$, which would contradict the validity of $\mathcal{A}$'s proof. Therefore $ind_k^{\mathsf{inst}\prime} = \texttt{false}$, which again implies $\sigma^{\mathsf{ack}} = \texttt{abort}$ and an invalid proof of $\mathcal{A}$.

This concludes the contradiction for all three indicators.

5.4.5 License Provider

(LNR) Non-repudiation: The verifier $\mathcal{V}$ is actually part of the implementation. We present it here for ease of reading: $\mathcal{V}$ first asks L for the SW installation request req^{inst}. If it is correct, then $\mathcal{V}$ defines success, i.e., $ind^{\mathsf{inst}} = \texttt{true}$, as the (intermediate) installation result. Only if the license receiver(s) still claim $ind^{\mathsf{inst}'} = \texttt{false}$, then $\mathcal{V}$ asks the receiver(s) for the installation confirmation σ^{conf}. After the extraction of $ind_k^{\mathsf{inst}'}$, $\mathcal{V}$ defines $ind^{\mathsf{inst}} = ind_k^{\mathsf{inst}'}$ as the installation result.

To prove the correctness of this verifier $\mathcal{V}$, we have to distinguish two cases that lead to dispute. L and the license receiver(s) call $\mathcal{V}$ if either

1. $ind^{\mathsf{inst}} = \texttt{false}$ and the output $ind_L^{\mathsf{inst}} = \texttt{true}$ to L was incorrect or
2. $ind^{\mathsf{inst}} = \texttt{true}$ and the output $ind_L^{\mathsf{inst}} = \texttt{true}$ to L was correct.

Note that the license receiver(s) are honest in case 1 and malicious in case 2.

1. L accepts the installation confirmation σ^{conf} and the installation result $ind^{\mathsf{inst}} = ind_k^{\mathsf{inst}'} = \texttt{false}$ due to the correctness of U_k.
2. The malicious receiver(s) cannot deceive $\mathcal{V}$ due to the correctness of U_k. As the installation result is success by the initial assumption, $ind_k^{\mathsf{inst}'} = \texttt{true}$ and the verifier agrees with L that $ind^{\mathsf{inst}} = \texttt{true}$.

We stress that the freshness of the license σ^{lic} is crucial in the above proof. If U_k accepts the same σ^{lic} twice in two different protocol runs although the freshness information is identical, then a malicious license receiver can easily deceive the verifier by combining messages from both protocol runs.

5.4.6 User

(UNR) Non-repudiation: We have to prove the correctness of ind_k^{inst} as well as the correctness of the verifier in case of dispute. We start with a proof by contradiction that ind_k^{inst} is correct. Let $ind_k^{\mathsf{inst}} = \neg ind^{\mathsf{inst}}$ be the incorrect output value of protocol SWInstallExternal to U_k. There are two alternatives:

1. False positive: $ind_k^{\mathsf{inst}} = \texttt{true}$ and $ind^{\mathsf{inst}} = \texttt{false}$
2. False negative: $ind_k^{\mathsf{inst}} = \texttt{false}$ and $ind^{\mathsf{inst}} = \texttt{true}$

We lead both alternatives to a contradiction:

1. As $ind^{\mathsf{inst}} = \texttt{false}$ and $ind_k^{\mathsf{inst}} = \texttt{true}$, by definition of ind^{inst} at least one of the output values ind_L^{inst} and ind_j^{inst} to L and I_j must equal $\texttt{false}$. Due to the correctness of L, ind_L^{inst} must equal $\texttt{true}$. Otherwise, L wouldn't have created a legal license and U_k would have calculated $ind_k^{\mathsf{inst}'} = \texttt{false}$ and thus $ind_k^{\mathsf{inst}} = \texttt{false}$. Therefore ind_j^{inst} must equal $\texttt{false}$. However, I_j delivered a correct σ^{ack} to U_k—otherwise ind_k^{inst} would have equaled $\texttt{false}$—and thus by definition of the protocol SWInstallExternal, ind_j^{inst} must have equaled $\texttt{true}$.

2. As $ind_k^{\mathsf{inst}} = \texttt{false}$, the installation result is $ind^{\mathsf{inst}} = \texttt{false}$ by definition and the contradiction is obvious. Note that U_k doesn't use s_m due to the calculation of ind_0^{inst} in protocol SWInstallInternal.

It remains to show the construction of a verifier $\mathcal{V}$ for the installation result and to prove its correctness. Although $\mathcal{V}$ is part of the implementation, we show it here for ease of reading. As ind_k^{inst} is correct, a dispute only arises if an adversary $\mathcal{A}$ maliciously claims $ind^{\mathsf{inst}'} = \neg(ind_k^{\mathsf{inst}})$ although the actual installation result is $ind^{\mathsf{inst}} = ind_k^{\mathsf{inst}}$. $\mathcal{V}$ asks $\mathcal{A}$ for the installation confirmation σ^{conf} and U_k for the installation acknowledgement σ^{ack}. If σ^{conf} is valid and indicates $ind_k^{\mathsf{inst}'} = \texttt{false}$, then $\mathcal{V}$ decides $ind^{\mathsf{inst}} = \texttt{false}$. If σ^{conf} indicates $ind_k^{\mathsf{inst}'} = \texttt{true}$, then $\mathcal{V}$ verifies $\left\langle \mathsf{Verify}(K_j^{\mathsf{test}}, \sigma^{\mathsf{conf}}, \mathsf{Sig}(\sigma^{\mathsf{ack}})) \right\rangle$. If the signature is valid, then $\mathcal{V}$ decides $ind^{\mathsf{inst}} = \texttt{true}$, otherwise $ind^{\mathsf{inst}} = \texttt{false}$. Note that in both cases $\mathcal{V}$ obtains the same result as a correct U_k.

(UIO) Installation Origin: To the contrary of UIO, we assume that the installation result is success although U_k did not generate req^{inst}. However, this implies that $ind_k^{\mathsf{inst}''} = \texttt{false}$ and thus $ind_k^{\mathsf{inst}} = \texttt{false}$. Therefore the installation result must be failure. This concludes the contradiction.

(UA) Authenticity: As the installation result is success, we have $ind_k^{\mathsf{inst}} = ind_k^{\mathsf{inst}'} = ind_k^{\mathsf{inst}''} = \texttt{true}$. Due to the calculation of $ind_k^{\mathsf{inst}''}$ and $ind_k^{\mathsf{inst}'}$ by a correct U_k, all SW indices match. Due to the calculation of $ind_k^{\mathsf{inst}'}$ based on input sig_m^{SW} from a correct T, the equality $s = s_m$ holds for the variable s of U_k in protocol SWInstallExternal. Due to the correctness of u_0 and u_i, the equality must also hold for the output s to u_i in protocol SWInstallInternal.

U_k ensures the authenticity of σ^{lic} in the calculation of $ind_k^{\mathsf{inst}'}$. Note that both authenticity proofs rely on the correctness of the public test keys from SystemSetup.

Chapter 6

Conclusion

In this report we have proposed a procedure for secure SW delivery and installation in embedded systems. It integrates installation service providers as intermediaries between SW provider and embedded system and categorizes them into separate clearance levels. Compatibility of the SW component and the target system is checked prior to installation. The fulfillment of a variety of requirements and the introduction of an elementary license system allows any SW provider to establish new business models that are currently not supported. The SW provider's intellectual property is protected and a variety of digital rights is supported. From the embedded system owner's point of view, the procedure prevents installation of illegal SW and supports warranty claims against the SW provider in case of defective SW with unambiguous evidence.

Public Key Broadcast Encryption (PKBE) enables efficient communication with embedded system on an insecure one-way channel. Even if the key material of delivered systems is not changed throughout the lifetime, an unbounded number of embedded systems can be revoked or excluded. The access to the broadcast channel is non-discriminatory, allowing any SW provider to distribute SW components after certification by a Trusted Third Party (TTP). The use of Trusted Computing (TC) concepts induces the necessary trust in the embedded system. Based on minimal TC hardware and a secure operating system kernel, the embedded system can be transformed into a trusted computing base (in an open environment). For example, this allows any SW provider to have trust in the confidentiality of his SW components. The use of property-based SW certification and sealing of the embedded system's configuration replaces the currently criticized attestation mechanisms which might be used for discrimination of individual SW providers.

Several opportunities for future work remain. For example, the need for a TTP should be reduced. One subject of investigation is the generation of the private key material by the embedded systems themselves and subsequent aggregation into a PKBE infrastructure. In addition, it would be interesting to consider proof-carrying code instead of SW certification by a TTP.

Bibliography

[ADR02] Jee Hea An, Yevgeniy Dodis, and Tal Rabin, *On the security of joint signature and encryption*, in Knudsen [Knu02], pp. 83–107.

[AHS05] André Adelsbach, Ulrich Huber, and Ahmad-Reza Sadeghi, *Secure software delivery and installation in embedded systems*, ISPEC 2005 (Robert H. Deng, ed.), Lecture Notes in Computer Science, vol. 3439, Springer, 2005, pp. 255–267.

[AHSS05] André Adelsbach, Ulrich Huber, Ahmad-Reza Sadeghi, and Christian Stüble, *Embedding trust into cars - secure software delivery and installation*, Third Workshop on Embedded Security in Cars—ESCAR 2005, Cologne, Germany, November 29–30, 2005, 2005.

[AJ03] H. Alminger and O. Josefsson, *Software handling during the vehicle lifecycle*, in VDI Society for Automotive and Traffic Systems Technology [VDI03], pp. 1047–1055.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh, *Hierarchical identity based encryption with constant size ciphertext*, in Cramer [Cra05], pp. 440–456.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway, *Relations among notions of security for public-key encryption schemes*, CRYPTO 1998 (Hugo Krawczyk, ed.), Lecture Notes in Computer Science, vol. 1462, Springer, 1998, pp. 26–45.

[BMW02] BMW Car IT, *Das Potenzial von Software im Fahrzeug*, Press report, BMW Group, July 22, 2002, URL `http://www.bmw-carit.de/pdf/plakate.pdf` - mailto: `info@bmw-carit.de` - file size: 2050 kB.

[BN00] Mihir Bellare and Chanathip Namprempre, *Authenticated encryption: Relations among notions and analysis of the generic composition paradigm*, ASIACRYPT 2000 (Tatsuaki Okamoto, ed.), Lecture Notes in Computer Science, vol. 1976, Springer, 2000, pp. 531–545.

[BS95] Dan Boneh and James Shaw, *Collusion-secure fingerprinting for digital data (extended abstract)*, CRYPTO 1995 (Don Coppersmith, ed.), Lecture Notes in Computer Science, vol. 963, Springer, 1995, pp. 452–465.

[CKLS97] Ingemar J. Cox, Joe Kilian, Tom Leighton, and Talal Shamoon, *Secure spread spectrum watermarking for multimedia*, IEEE Transactions on Image Processing **6** (1997), no. 12, 1673–1687.

[Cra05] Ronald Cramer (ed.), *Advances in Cryptology—EUROCRYPT 2005, Twenty-Fourth Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005*, Lecture Notes in Computer Science, vol. 3494, Springer, 2005.

[Dai02] Daimler Chrysler AG, *Functional specification of a flash driver version 1.3*, Specification, Herstellerinitiative Software, June 06, 2002, URL `http://www.automotive-his.de/download/HIS%20flash%20driver%20v130.pdf` - mailto: his@mbtech-services.net - file size: 224 kB.

[DF03] Yevgeniy Dodis and Nelly Fazio, *Public key broadcast encryption for stateless receivers*, Digital Rights Management Workshop (Joan Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 2696, Springer, 2003, pp. 61–80.

[DS04] Christoph Dallmayr and Oliver Schlüter, *ECU software development with diagnostics and flash down-loading according to international standards (SAE Technical Paper Series 2004-01-0273)*, in Society of Automotive Engineers (SAE) [Soc04], URL `http://www.sae.org/`.

[Eur04] *Jahrestagung Elektronik-Systeme im Automobil, Fachtag Design – Test – Diagnose elektronischer Systeme, Munich, Germany, February 12, 2004*, Euroforum, 2004.

[FN94] Amos Fiat and Moni Naor, *Broadcast encryption*, CRYPTO 1993 (Douglas R. Stinson, ed.), Lecture Notes in Computer Science, vol. 773, Springer, 1994, pp. 480–491.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff, *The knowledge complexity of interactive proof systems*, SIAM Journal on Computing **18** (1989), no. 1, 186–208.

[Gol01] Oded Goldreich, *Basic tools*, first ed., Foundations of Cryptography, vol. 1, Cambridge University Press, Cambridge, UK, 2001.

[GS02] Craig Gentry and Alice Silverberg, *Hierarchical ID-based cryptography*, ASIACRYPT 2002 (Yuliang Zheng, ed.), Lecture Notes in Computer Science, vol. 2501, Springer, 2002, pp. 548–566.

[HCF04] Vivek Haldar, Deepak Chandra, and Michael Franz, *Semantic remote attestation—a virtual machine directed approach to trusted computing*, Proceedings of the 3rd Virtual Machine Research and Technology Symposium (May 6–7, 2004, San Jose, CA, USA) (2004), 29–41, URL `http://www.usenix.org/events/vm04/tech/haldar/haldar.pdf`.

[Her03] Jürgen Herre, *Content based identification (fingerprinting)*, Digital Rights Management: Technological, Economic, Legal and Political Aspects (Eberhard Becker, Willms Buhse, Dirk Günnewig, and Niels Rump, eds.), Lecture Notes in Computer Science, vol. 2770, Springer, 2003, pp. 93–100.

[HF05] Vivek Haldar and Michael Franz, *Symmetric behavior-based trust: A new paradigm for Internet computing*, in NSPW 2004 [NSP05], 79–84.

[HL02] Jeremy Horwitz and Ben Lynn, *Toward hierarchical identity-based encryption*, in Knudsen [Knu02], pp. 466–481.

[HMF+03] A. Heinrich, K. Müller, J. Fehrling, A. Paggel, and I. Schneider, *Version management for transparency and process reliability in the ECU development*, in VDI Society for Automotive and Traffic Systems Technology [VDI03], pp. 219–230.

[HS02] Dani Halevy and Adi Shamir, *The LSD broadcast encryption scheme*, CRYPTO 2002 (Moti Yung, ed.), Lecture Notes in Computer Science, vol. 2442, Springer, 2002, pp. 47–60.

[HS04] Cornelia Heinisch and Martin Simons, *Loading flashware from external interfaces such as CD-ROM or W-LAN and programming ECUs by an on-board SW-component (SAE Technical Paper Series 2004-01-0678)*, in Society of Automotive Engineers (SAE) [Soc04], URL `http://www.sae.org/`.

[HWM03] M. Huber, T. Weber, and T. Miehling, *Standard software for in-vehicle flash reprogramming*, in VDI Society for Automotive and Traffic Systems Technology [VDI03], URL `http://www.automotive-his.de/download/presentation-baden-baden-2003-german.zip`, pp. 1011–1020.

[JHC+05] Nam-Su Jho, Jung Yeon Hwang, Jung Hee Cheon, Myung-Hwan Kim, Dong Hoon Lee, and Eun Sun Yoo, *One-way chain based broadcast encryption schemes*, in Cramer [Cra05], pp. 559–574.

[JPLI99] Trent Jaeger, Atul Prakash, Jochen Liedtke, and Nayeem Islam, *Flexible control of downloaded executable content*, ACM Transactions on Information and System Security **2** (1999), no. 2, 177–228.

[KK04] Deepa Kundur and Kannan Karthik, *Video fingerprinting and encryption principles for digital rights management*, Proceedings of the IEEE **92** (2004), no. 6, 918–932.

[Knu02] Lars R. Knudsen (ed.), *Advances in Cryptology—EUROCRYPT 2002, International Conference on the Theory and Applications of Cryptographic Techniques, Amsterdam, The Netherlands, April 28 – May 2, 2002*, Lecture Notes in Computer Science, vol. 2332, Springer, 2002.

[Mül04] Markus Müller, *IT-Security in Fahrzeugnetzen*, Elektronik Automotive (2004), no. 4, 54–59, ISSN: 1614-0125.

[NNL01] Dalit Naor, Moni Naor, and Jeff Lotspiech, *Revocation and tracing schemes for stateless receivers*, CRYPTO 2001 (Joe Kilian, ed.), Lecture Notes in Computer Science, vol. 2139, Springer, 2001, pp. 41–62.

[NSP05] *New Security Paradigms Workshop—NSPW 2004, Nova Scotia, Canada, September 20–23, 2004*, ACM Press, 2005.

[Oef04] Uwe Oeftiger, *Diagnose und Reparatur elektronisch unterstützter Fahrzeuge*, in Euroforum 2004 [Eur04].

[PSHW04] Jonathan Poritz, Matthias Schunter, Els Van Herreweghen, and Michael Waidner, *Property attestation—scalable and privacy-friendly security assessment of peer computers*, Research Report RZ 3548 (# 99559), IBM Research, Zurich Laboratory, May 10, 2004.

[Sch03] Martin Schmitt, *Software-update, configuration and programming of individual vehicles on the aftermarket with an intelligent data-configurator*, in VDI Society for Automotive and Traffic Systems Technology [VDI03], pp. 1021–1046.

[Soc04] Society of Automotive Engineers (SAE) (ed.), *SAE World Congress 2004, Vehicle Sensors, Actuators, and Diagnostics (SP-1853), Detroit, Michigan, USA, March 8–11, 2004*, 2004, URL `http://www.sae.org/`.

[SS03] Ahmad-Reza Sadeghi and Christian Stüble, *Taming "trusted computing" by operating system design*, WISA 2003 (Kijoon Chae and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 2908, Springer, 2003, pp. 286–302.

[SS05] ———, *Property-based attestation for computing platforms: Caring about properties, not mechanisms*, in NSPW 2004 [NSP05], pp. 67–77.

[Stö03] S. Stölzl, *Software products for vehicles*, in VDI Society for Automotive and Traffic Systems Technology [VDI03], pp. 1073–1088.

[Tru02] Trusted Computing Platform Alliance (TCPA), *Main Specification, Version 1.1b*, Technical specification, February 2002.

[Tru03a] Trusted Computing Group (TCG), *TCG Software Stack Specification, Version 1.1*, Technical specification, August 2003, URL `https://trustedcomputinggroup.org/`.

[Tru03b] ———, *TPM Main Specification, Version 1.2*, Technical specification, November 2003, URL `https://trustedcomputinggroup.org/`.

[VDI03] VDI Society for Automotive and Traffic Systems Technology (ed.), *Electronic Systems for Vehicles, VDI Berichte 1789, Congress*, Baden-Baden, Germany, VDI Verlag GmbH Düsseldorf, September 25–26, 2003.

Appendix A

Summary of Abbreviations and Variable Names

A.1 Abbreviations

CAN	Controller Area Network
ECU	Electronic Control Unit
ESP	Electronic Stability Program
Euro NCAP	European New Car Assessment Programme
DRM	Digital Rights Management
HW	Hardware
ISP	Installation Service Provider
ITM	Interactive Turing Machine
LIN	Local Interconnect Network
LP	License Provider
MAC	Message Authentication Code
MOST	Media Oriented System Transport
NHTSA	National Highway Traffic Safety Administration
OEM	Overall Equipment Manufacturer
OS	Operating System
PKBE	Public Key Broadcast Encryption
PKI	Public Key Infrastructure
SAP	Software Application Programmer
SW	Software
SWP	SW Provider
TC	Trusted Computing
TCB	Trusted Computing Base
TTP	Trusted Third Party
UP	User Platform
URL	Uniform Resource Locator

A.2 Variable Names

A.2.1 Roles

O	OEM
$\mathcal{S}$	Set of all SAPs
$S_d \in \mathcal{S}$	Specific SAP with index d
$\mathcal{P}$	Set of all SWPs
$P_d \in \mathcal{P}$	Specific SWP with index d
$\mathcal{I}$	Set of all ISPs
$I_j \in \mathcal{I}$	Specific ISP with index j
L	License provider
T	Trusted third party
$\mathcal{U}$	Set of all user platforms
$\mathcal{R}_\mathcal{U}$	Set of all revoked user platforms
$U_k \in \mathcal{U}$	Specific user platform with index k
$u_{k,i}$	Specific component in a specific user platform U_k
u_0	Trusted component of a user platform, shorthand for $u_{k,0}$
u_i	Target component within a user platform, shorthand for $u_{k,i}$

A.2.2 Variables Related to the Preliminaries

$\mathcal{SW}$	Set of all SW components
s_m	Specific SW component with index m
$\mathcal{C}$	Set of all clearance levels
c	Clearance level of an ISP
$\mathcal{SW}_c$	Set of all SW components installable with clearance level c
c^{min}	Minimum clearance level of an SW component
c_m^{min}	Minimum clearance level of a specific SW component s_m
$\mathcal{T}\&\mathcal{C}_m$	Terms and conditions of a specific SW component s_m
$\mathcal{R}_m$	Set of usage rights for a specific SW component s_m
$r \in \mathcal{R}_m$	Usage right for a specific SW component s_m
$\mathcal{R}_m^{\mathsf{total}}$	Set of all allowed usage rights of a specific SW component s_m
p	Usage parameter of an SW component
σ^{lic}	License for an SW component
$\mathcal{Q}_{\mathsf{SW}}^{\mathsf{total}}$	Set of all SW properties
$\mathcal{Q}_m^{\mathsf{total}}$	Set of all properties of a specific SW component s_m
$\mathcal{Q}_m$	Set of properties of a specific SW component s_m
q	Specific SW property
$\mathcal{Q}_{\mathsf{UP}}^{\mathsf{total}}$	Set of all user platform properties
$\mathcal{Q}_k$	Set of all properties of a specific user platform U_k
Q	Specific user platform property

A.2.3 Output Variables of the Basic Protocols

X	Party participating in a protocol
in_X	Input value of party X
out_X	Output value of party X
req^{clear}	Clearance level certification request from an ISP to the TTP
ind_T^{clear}	Acceptance indicator of the TTP T in protocol ClearCert
ζ_j^{clear}	Clearance level certificate of a specific ISP I_j
ind_j^{clear}	Acceptance indicator of ISP I_j in protocol ClearCert
req^{SW}	SW certification request from an SWP to the TTP
ind_T^{SW}	Acceptance indicator of the TTP T in protocol SWCert
ζ_m^{SW}	SW certificate for a specific SW component s_m
σ_m^{SW}	Integrity proof for a specific SW component s_m
$sig_m^{\mathsf{SW}} = \mathsf{Sig}(\sigma_m^{\mathsf{SW}})$	Integrity signature for a specific SW component s_m
ind_d^{SW}	Acceptance indicator of SWP P_d in protocol SWCert
s_m^{enc}	SW component s_m in protected form (signed and PKBE encrypted)
σ_m^{prop}	Property statement of SWP P_d with respect s_m
req^{inst}	SW installation request from U_k
σ^{inst}	SW installation package from I_j in protocol SWInstallExternal
σ^{conf}	Installation confirmation from U_k in protocol SWInstallExternal
σ^{ack}	Installation acknowledgement from I_j in protocol SWInstallExternal
ind_k^{inst}	Acceptance indicator of user platform U_k
ind_j^{inst}	Acceptance indicator of ISP I_j
ind_L^{inst}	Acceptance indicator of license provider L
s_i^{enc}	SW component s_m in protected form (symmetrically encrypted for u_i)
mac^{inst}	SW installation package from u_0 in protocol SWInstallInternal
mac^{conf}	Installation confirmation from u_i in protocol SWInstallInternal
mac^{ack}	Installation acknowledgement from u_0 in protocol SWInstallInternal
ind_0^{inst}	Acceptance indicator of trusted component u_0
ind_i^{inst}	Acceptance indicator of target component u_i

A.2.4 Cryptographic Keys

$\mathcal{K}^{\mathsf{enc}}$	Set of public keys of a PKBE scheme
$\mathcal{K}_k^{\mathsf{dec}}$	Set of private keys of U_k in a PKBE scheme
K_d^{sign}	Signature key of SW provider P_d
K_j^{sign}	Signature key of ISP I_j
K_L^{sign}	Signature key of the license provider L
K_k^{sign}	Signature key of user platform U_k
K_T^{sign}	Signature key of the TTP T
$K_{Manuf}^{\mathsf{sign}}$	Signature key of a manufacturer *Manuf*
K_d^{test}	Test key of SW provider P_d
K_j^{test}	Test key of ISP I_j
K_L^{test}	Test key of the license provider L
K_k^{test}	Test key of user platform U_k
K_T^{test}	Test key of the TTP T
$K_{0\leftrightarrow i}$	Key shared between trusted component u_0 and target component u_i
$K_{k,0}^{\mathsf{pub}}$	Public key of the trusted component $u_{k,0}$

Appendix B

Public Key Broadcast Encryption

In a PKBE scheme, any (not necessarily trusted) party can distribute an SW component s on the broadcast channel. Specifically, this holds true for each SW provider S, making the channel non-discriminatory. In the setup phase, T splits the set of all user platforms $\mathcal{U}$ into a well-chosen subset scheme in such a way that each U_k is part of several subsets. Two of these schemes were introduced in [NNL01] and extended to the public key property in [DF03]. T chooses the security parameters, e.g., key lengths, and generates a public key as well as a private key for each subset. All public keys are supposed to be known to any party while T gives the private key of each subset only to those U_k that are elements of the subset. T can generate the keys at any time after setup and provides U_k with its private keys when U_k is manufactured. For memory efficiency reasons, the user does not need to store each of his private keys.[1] In the automotive case example, the manufacturer of the trusted computing HW might take over the role of T. However, it might even be O if all SW providers trust O.

In the distribution phase, the SW provider first needs to select a set $\mathcal{U}$ of intended users. Then he computes a selection of subsets—called "cover" of $\mathcal{U}$—in such a way that only the members of $\mathcal{U}$ are contained in the subsets and that the number of subsets remains small.[2] Finally, he encrypts s with a session key and, in turn, the session key with the public keys of all subsets in the cover. On the receiving end, each U_k in the cover has the necessary private keys for decrypting the session key and subsequently s. Nobody else—specifically, neither the original SW provider nor any U_k outside the cover—can decrypt the session key, thus providing confidentiality. Although T can decrypt any session key based on the master key, the setup phase can be carried out in such a way that all potential users receive their key set and the master key is destroyed.

In our model, PKBE has one main advantage over a regular Public Key Infrastructure (PKI): PKBE is significantly more efficient regarding message header length, i.e., it needs far fewer encryptions of the session key when a message is sent over the one-way broadcast channel. In addition, PKBE in [DF03] even comprises a regular PKI. Each user is contained in a subset of size 1 to which only he holds the private key. Therefore a sender can distribute s even to a very small set of intended recipients by encrypting—in the worst case—the session key with the public subset key of each intended user.

The selection of the intended users might be based on two criteria. Firstly and most

[1] In [DF03], the authors present two alternatives with user storage requirements $O(\log |\mathcal{U}|)$ and $O(\log^2 |\mathcal{U}|)$ respectively where $\mathcal{U}$ is the set of user platforms and $|\mathcal{U}|$ its cardinality.

[2] [NNL01] presents an algorithm that finds a cover of size $O(|\mathcal{R}_\mathcal{U}|)$, where $\mathcal{R}_\mathcal{U}$ is the set of revoked users.

importantly, all revoked users are excluded, e.g., when a trusted component u_0 has been compromised and traced. Secondly, all potentially incompatible users might be excluded. For example, if a high-end SW component s can only be installed in a specific luxury class vehicle, the SW provider might exclude any compact class vehicle. However, U_k still performs a compatibility test. In the example, the compact class vehicle would refuse to install s anyway due to lacking compatibility. Therefore, the second selection step is unnecessary and even increases the message header length.

Note that [DF03] is based on a Hierarchical Identity-Based Encryption (HIBE) scheme as proposed in [HL02, GS02, BBG05]. The currently most efficient HIBE scheme is [BBG05] and was published after [DF03]. When implementing [DF03], the HIBE scheme [GS02] should thus be replaced with [BBG05].

Appendix C

Implementation

The proposed solution is relevant for an actual implementation. The cryptographic primitives, e.g., signatures, PKBE and symmetric encryption schemes, are readily available and their security has been proven. The roles that we have introduced either exist today or might be taken over by a party that can easily evolve out of existing players in the respective industries.

Trusted computing hardware is currently being developed by several industry groups and standards bodies such as the Trusted Computing Group.[1] An adaptation of the hardware, e.g., Trusted Platform Module or tamper-resistant memory, to an embedded environment seems feasible. In this scenario, the private key material of U_k is stored in a tamper-resistant memory and all other keys are stored in either tamper-resistant memory or encrypted form. All SW tasks are separated from each other by the operating system, preventing tasks from eavesdropping and modifying the physical memory or processor instructions. Secure operating systems can be based on secure microkernel architectures. For a discussion of these architectures, we refer to [SS03]. Due to the proposed installation procedure, only one component per embedded system needs to be a trusted computing base. This respects cost requirements of the respective industries that prevent the use of trusted hardware in every single component of the system.

Property-based sealing allows to bind the private PKBE keys to a correct configuration of u_0. We derive it from property-based attestation as introduced in [SS05] (for a similar method see [HCF04, HF05]). In contrast to attestation, which only proves U_k's platform configuration at a certain point of time, sealing allows to permanently bind secret information to a correct platform configuration. For this purpose, a trusted module of u_0 stores the private PKBE keys, but releases them only if u_0 is in a trustworthy configuration defined by u_0's properties. Each time that the task for decrypting PKBE ciphertext calls the trusted module and asks for the private keys, the module determines the current properties of the platform and checks if they match with the properties of a trustworthy configuration. The module releases the private keys only in case of a match. For further details on the implementation of u_0, we refer to [AHSS05].

[1] URL: `https://www.trustedcomputinggroup.org/`. For details, see [Tru02, Tru03a, Tru03b].

Appendix D

Proof of Theorem 1

We detail the proof sketch of [ADR02]. In Section (4) of the proof of Theorem 1 in [ADR02], the authors show that IND-gCCA2 security of the base encryption scheme paired with UF-CMA security of the base signature scheme implies IND-gCCA2 security of the resulting $\mathcal{S}t\mathcal{E}$ signcryption scheme. By signcryption we mean any composition of a signature and an encryption scheme that provides privacy and authenticity.

The basic idea of our proof is as follows. We assume to the contrary of our Theorem 1 that there exists an IND-CCA1 adversary $\mathcal{A}'$ which is capable of distinguishing the signcryptions of two messages M_0 and M_1 although the base schemes are IND-CCA1 and UF-CMA-secure. If $\mathcal{A}'$ existed, it would break the IND-CCA1 security of the signcryption scheme. However, with the help of this distinguisher $\mathcal{A}'$ we construct an adversary $\mathcal{A}$ which is capable of distinguishing the encryptions of two messages S_0 and S_1 in the base encryption scheme, thus breaking IND-CCA1 security of the base encryption scheme and contradicting the initial assumption.

Proof Let there be a distinguisher $\mathcal{A}'$ of the $\mathcal{S}t\mathcal{E}$ scheme whose success probability ϵ is non-negligible in the security parameter λ. Based on $\mathcal{A}'$ we construct a distinguisher $\mathcal{A}$ of the base scheme $\mathcal{E}$ that also has non-negligible success probability (see pages 50 and 51).

The challenger for the base encryption scheme runs the key generation algorithm and obtains the public encryption key as well as the secret decryption key of receiver R. The challenger keeps the decryption key DK_R secret and passes the encryption key EK_R on to $\mathcal{A}$.

$\mathcal{A}$ runs the key generation algorithms of $\mathcal{E}$ and $\mathcal{S}$ and combines the resulting keys with the public key EK_R obtained from his challenger. $\mathcal{A}$ passes the generated keys SDK_S, VEK_S and VEK_R on to $\mathcal{A}'$. Specifically, VEK_R contains the encryption key EK_R which the challenger of $\mathcal{A}$ chose. We note that $\mathcal{A}'$ also obtains the secret signing and decryption key SDK_S of S because the proof is in the Insider-security model.

Subsequently, $\mathcal{A}'$ issues adaptive chosen-ciphertext oracle queries. We denote the query stage of $\mathcal{A}'$ with `query` and the state information about the query history with *state*. As there is no de-signcryption oracle, $\mathcal{A}$ needs to impersonate this oracle and correctly answer the queries. When $\mathcal{A}'$ queries a signcryption C, then $\mathcal{A}$ sends C to his own decryption oracle (which does exist) and obtains the decryption S. $\mathcal{A}$ then verifies whether S is a valid signature and sends M to $\mathcal{A}'$ as the answer to the oracle query.

When $\mathcal{A}'$ decides that the query stage is over, $\mathcal{A}'$ starts the find stage, indicated with `find`, in order to find two message M_0 and M_1 on which it wishes to be challenged. $\mathcal{A}'$ sends both messages to $\mathcal{A}$, who signs both messages, thus creating two valid signcryptions, and forwards them to his challenger $\mathcal{C}$. $\mathcal{C}$ tosses a coin b in order to encrypt one of the two

Construction of adversary $\mathcal{A}$ against $\mathcal{E}$ from adversary $\mathcal{A}'$ against $\mathcal{S}t\mathcal{E}$

$\mathcal{C}$	$\mathcal{A}$	$\mathcal{A}'$

$(EK_R, DK_R) \leftarrow \mathsf{GenKeyEnc}(1^\lambda)$

$\xrightarrow{EK_R}$

$(EK_S, DK_S) \leftarrow \mathsf{GenKeyEnc}(1^\lambda)$
$(SK_S, VK_S) \leftarrow \mathsf{GenKeySig}(1^\lambda)$
$(SK_R, VK_R) \leftarrow \mathsf{GenKeySig}(1^\lambda)$
$SDK_S \leftarrow (SK_S, DK_S)$
$VEK_S \leftarrow (VK_S, EK_S)$
$VEK_R \leftarrow (VK_R, EK_R)$

$\xrightarrow{SDK_S,\ VEK_S,\ VEK_R}$

Repeat
$(C, state) \leftarrow \mathcal{A}'(SDK_S, VEK_S, VEK_R, state; \texttt{query})$

$\xleftarrow{C}$

$\xleftarrow{C}$

$S \leftarrow \mathsf{Dec}_{DK_R}(C)$

$\xrightarrow{S}$

$ind \leftarrow \mathsf{Verify}(VK_S, S)$
if $ind = \texttt{true}$
then
$M \leftarrow \mathsf{Msg}(S)$
else
$M \leftarrow \texttt{false}$

50 $\xrightarrow{M}$

Continuation of construction of adversary $\mathcal{A}$

$\mathcal{C}$	$\mathcal{A}$	$\mathcal{A}'$
		$(M_0, M_1, state) \leftarrow \mathcal{A}'(state; \texttt{find})$
	$\xleftarrow{M_0, M_1}$	
	$S_0 \leftarrow \mathsf{Sig}_{SK_S}(M_0)$ $S_1 \leftarrow \mathsf{Sig}_{SK_S}(M_1)$	
	$\xleftarrow{S_0, S_1}$	
$b \leftarrow \{0,1\}$ $C_b \leftarrow \mathsf{Enc}_{EK_R}(S_b)$		
	$\xrightarrow{C_b}$	
		$\xrightarrow{C_b}$
		$b' \leftarrow \mathcal{A}'(C_b, state; \texttt{guess})$
		$\xleftarrow{b'}$
	$\xleftarrow{b'}$	

messages selected at random. Then $\mathcal{C}$ sends the resulting challenge ciphertext C_b to $\mathcal{A}$, which in turn passes it on to $\mathcal{A}'$ without any modification.

$\mathcal{A}'$ starts the guess stage, indicated with `guess`, and outputs his guess b'. $\mathcal{A}'$ then sends b' to $\mathcal{A}$, who simply passes b' on to his challenger $\mathcal{C}$ as his own guess.

By the construction of $\mathcal{A}$, a correct guess b' of $\mathcal{A}'$ for $\mathcal{S}t\mathcal{E}$ always implies a correct guess of $\mathcal{A}$ for $\mathcal{E}$. Both adversaries thus have the same non-negligible success probability, contradicting the initial assumption that $\mathcal{E}$ is IND-CCA1-secure. We stress that $\mathcal{A}'$ cannot distinguish $\mathcal{A}$'s answers to his oracle queries from the answers of a real oracle. ■