On Formal Security Analysis of Automotive Systems

Sigrid Gürgens and Norman Lahr and Daniel Zelle
Fraunhofer Institute for Secure Information Technology
Darmstadt, Germany
Email: {sigrid.guergens, norman.lahr, daniel.zelle}@sit.fraunhofer.de

***Abstract*—Today's cars use an increasing number of electronic control units (ECUs) due to the development towards a more and more autonomous behavior. Further, they get exceedingly connected to the Internet to provide numerous services to the driver, such as streaming music, receiving e-mails or remotely locking and unlocking the car's door. These new features can result in security risks. Researchers have shown security problems in multiple cases regarding e.g. the OBD-II port of a car, the mobile phone connection via Bluetooth, and the cellular connection of a car. The currently used approaches to avoid such failures are restricted to applying risk analysis, attack trees, penetration tests etc. Numerous approaches based on formal models exist, yet to the best of our knowledge none addresses the specifics of automotive systems. This paper introduces the extension of an approach previously used for cryptographic protocol analysis, explains its tool support, and applies it to a German manufacturer's system to remotely unlock car doors that was recently found to be flawed. Our analysis confirms these findings and identifies an additional security issue. The paper further introduces a fieldbus communication model, thus enabling to include in-car network communication in the formal analysis.**

I. Introduction

We are surrounded by cooperating embedded systems in our everyday life that control multiple aspects from coffee machines to medical equipment. One of the most complex embedded systems we have to cope with are vehicles. Modern vehicles use an increasing number of electronic control units (ECUs) due to the development towards a more and more autonomous behavior. In addition, cars get more and more connected to the outside world, for example by vehicle to vehicle communication and by connections to smartphones or to the Internet. The latter are used to provide to the driver the possibility to stream music, or receive e-mails or online messages. It is also possible to unlock the car's door via smart phone, start or stop the engine, and turn the light on or off.

These new features can result in security risks. Researchers have shown security problems in multiple cases regarding the OBD-II port of a car, the CD-player, the mobile phone connection via Bluetooth, and the cellular connection of a car [1]. The latter has recently been found to allow unlocking [2] and steering of cars [3].

Security problems are often a result of a fragile security design in which security mechanisms are used wrongly. Security requirements are addressed by applying risk analysis, attack trees, penetration tests etc. The SAE suggests this approach for the automotive industry [4] as standard procedure. However, these methods have drawbacks. First, they always depend on the knowledge, skill, and creativity of the people applying the procedure. Moreover, as the systems become more and more complex in functionality and connectivity, it becomes harder for a tester to cover all possible attack vectors of a system. Further, it is preferable to detect issues in an early state of development that allows easier and thus less costly adaptation of the design.

In the past, formal analysis has proven to be a useful approach for the verification of cryptographic protocols (see Section VII). In this paper we present extensions of our work on cryptoprotocol analysis (see e.g. [5]): (i) a generic way of modeling cryptographic mechanisms on different abstraction levels that allows to focus on different aspects of the system to be analysed, and (ii) a formal model of fieldbus communication, thus enabling the formal security analysis of automotive systems including in-car networks. Our approach is supported by the Simple Homomorphism Verification Tool (SHVT).

We use the real world automotive scenario that in [2] was reported to be flawed to show the applicability and usefulness of our approach. In the next section, we construct an abstract automotive system architecture and an attack model. From these we derive a formal model based on Asynchronous Product Automata (APA) in Section III. Section IV explains the tool support. Then, Section V introduces the real-world example, the concrete attack scenarios we used, and the respective formal models. It then presents the results of our SHVT based analyses. Our extension regarding fieldbus communication is introduced and discussed in Section VI. Our paper is summed up with the related work and a final discussion of our results and future work.

II. An Abstract Model of a Connected Car

Before a formal model of a system can be determined, it is required to create an abstract system model derived from the real system that is evaluated to outline the attack vectors. In this section, we define an abstract vehicle communication

system model that provides telemetric functions and on-board communication. This is a subset of all possible communication architectures in a modern vehicle and represents a common communication system for a connected car. On the vehicle side the telemetric function is implemented by a Telemetric ECU, and logically connected components that are required for its operation. Furthermore, we describe the common interfaces and techniques that are utilized to connect the individual components. The created model is the basis for the construction of a formal model in Section III. In addition, we specify a proper attacker model for the derived system model.

In [6], Miller et al. surveyed several communication network and bus topologies of car models from different manufacturers. Furthermore, the analysis of an automotive telemetric platform introduced in [2] reveals some information about network and bus structure. Based on this information we consider a network infrastructure limited to Internet Protocol-based and GSM (Global System for Mobile Communications) communication networks (point-to-point) of a so called *connected car* that offers several services to passengers. A connected car is commonly defined as a car that is equipped with access to the Internet. Additionally, we consider a fieldbus topology, e.g. a CAN, for the in-car network. This enables various use cases, e.g. sharing the Internet connection or offering comfort functions to the driver, like unlocking the doors, starting the engine, or enabling pre-heating from remote. Moreover, the manufacturer or other service providers can offer remote services like car tracking or car sharing. Most comfort functions need direct access to the safety-critical in-car network to submit commands to corresponding ECUs. Therefore car manufacturers utilize a gateway unit to translate the control messages from the outer to the inner car network. Also the Internet-uplink is handled by this unit. Furthermore, a central server in the back-end or Internet, respectively, is employed to manage remote services.

Hence, we include four categories of components in our system model: (1) An *In-Car Network* that covers a fieldbus which is connected to the Telemetric ECU and additional sensor and actuator ECUs. (2) A *Telemetric ECU* as an Internet gateway and entry point to the car's internal network and so to the cyber-physical interface (sensors and actors) of the car. The connection to the Internet is established by a cellular network connection (GPRS (General Packet Radio Service) over GSM). (3) A *Backend-Server*, connected to the Internet via an arbitrary wide area network (WAN) that offers services or forwards service requests to the Telemetric ECU. (4) *Terminals* that represent clients, like smart phones, using the offered connected car services via a cellular network. Figure 1 illustrates the derived system model.

A. Attacker Model

In this paper, we consider the wireless communication interfaces GSM/GPRS as the main attack vectors for a remote attacker and the fieldbus interface for a local attacker. Hence

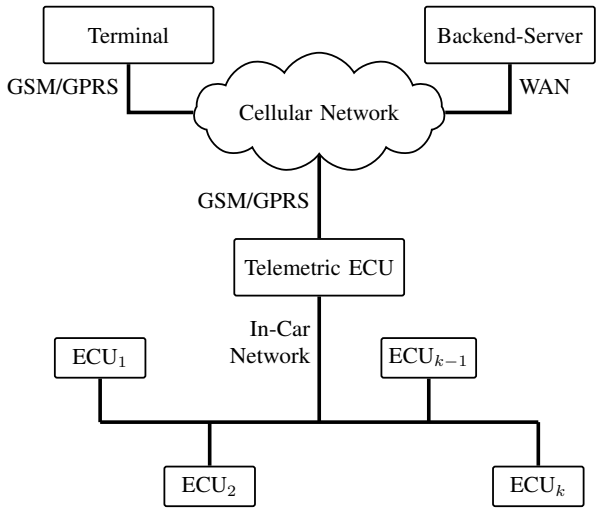

Fig. 1. System Model of a Connected Car.

the skill of an attacker includes the ability to read and write from and to the GSM/GPRS communication channel. This can be achieved for example by using a software-defined radio (SDR) and a proper software to run a base transceiver station (BTS) at moderate cost as demonstrated in [7]. For the fieldbus there are several adapters available on the market, e.g. usbtin[1] for CAN, with a simple serial communication interface that can be used by crafted scripts.

So, for a general attacker model an attacker follows the Dolev-Yao attacker model on considered communication technologies. So she can eavesdrop each message on a channel, inject custom messages, replace messages, and suppress messages. Further, we assume that an attacker is not able to break secure cryptographic primitives to reveal keys just by knowing messages.

III. The Formal Model of a Connected Car

In this section we introduce our approach for formally modeling the system introduced in Section II. We use Asynchronous Product Automata (APA), a universal and very flexible operational description concept for cooperating systems [8]. It "naturally" emerges from formal language theory [9]. APA are supported by the Simple Homomorphism Verification Tool (SHVT) that provides components for the complete cycle from formal specification to exhaustive analysis and verification [8].

We have used our APA based approach and the SHVT successfully in the past to model and analyze the security of cryptographic protocols (see for example [5], [10]) and scenarios based on Trusted Computing [11]. This section gives a short summary of the concepts used in our earlier work, introduces new concepts, and explains how our approach can

[1] http://www.fischl.de/usbtin/

be used for security analysis in general. Extensions of our approach to capture specifics of the automotive domain will be introduced in Section VI.

A. APA Specification of the Connected Car Model

As stated in [10], an APA can be seen as a family of elementary automata. The set of all possible states of the whole APA is structured as a product set; each state is divided into state components. In the following the set of all possible states is called state set. The state sets of elementary automata consist of components of the state set of the APA. Different elementary automata are "glued" by shared components of their state sets. Elementary automata can "communicate" by changing the content of shared state components.

Each entity (agent) P acting in the system is modeled by an appropriate number of elementary automata that perform the entity's actions, accompanied by an adequate number of state components. What is appropriate depends on the environment and the attack model that shall be modeled. In order to reduce the state space it is often convenient to use one elementary automaton per agent.

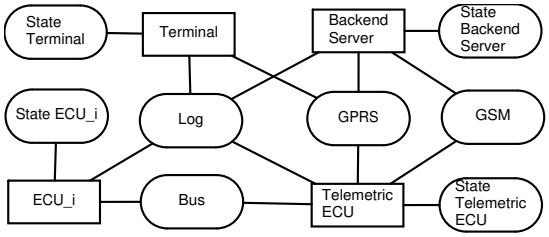

Fig. 2. APA model derived from the connected car model of Figure 1.

Figure 2 graphically represents an APA formally modeling that part of the Connected Car model introduced in the previous section not yet including specifics of the automotive domain. The rectangles of an APA depict the respective elementary automata and the ellipses represent their state components. The neighborhood relation N (graphically represented by an edge) indicates which state components are included in the state of an elementary automaton and may be changed by a state transition of this automaton. Communication of the automata is achieved by adding to and removing data from shared state components.

Our APA model contains the following elementary automata: $Terminal$ (e.g. a mobile phone), *T-ECU* acting as a Telemetric ECU entity, $Backend$ representing the Backend-Server, and ECU_i representing other ECUs of the system. The Cellular network is modeled by two state components GSM and $GPRS$. Automaton $Terminal$ may change $GPRS$ but cannot read or change the state of GSM. This reflects that a mobile phone usually communicates with the Backend-Server via GPRS. A system using GSM communication between $Terminal$ and $Backend$ can easily be covered by including GSM in the neighborhood relation of $Terminal$. $Backend$ and *T-ECU* communicate via the GSM component when sending and receiving an SMS and via the GPRS component for requesting and receiving IP-based messages. *T-ECU* is connected to other ECUs throught the Bus state component. The global state component Log holds a sequence of actions performed by the elementary automata that are relevant for the security property to be analyzed.

Different scenarios may be modeled with a different set of elementary automata and state components. For example, *T-ECU* may be split into two automata modeling a Telemetric ECU consisting of a GSM modem with integrated SIM card and a microcontroller.

The full specification of the APA includes the state sets (i.e. data types), the state transition relations of the elementary automata, and the initial state, which we will explain in the following sections.

1) State Sets of Components: For the definition of the concrete state sets of components of the APA depicted in Figure 2 we use as basis a set $Agents$ of agents, a set $nat0$ of natural numbers to model e.g. random numbers, and numerous sets of constants for specifying types of communication channels, key identifiers, IP addresses, etc. The union of all these sets represents the set of *atomic data*, based on which we construct a set $\mathcal{D}$ of data by concatenation and Cartesian products that is used to define the components' state sets. While the state sets of most components can be freely defined, the shared ones must comply to a specific syntax.

GSM and GPRS state sets Elements of the components GSM and $GPRS$ have the following form:

$((sender, recipient), message)$ with $message = mpart_1, \ldots, mpart_k$ $(k \in \mathbb{N})$ and $mpart_i = (header_i, channel_i, content_i)$ (for all $i \in \{1, \ldots, k\}$).

The parameter $header$ is used to specify protocol specific details such as the cryptographic mechanisms to be used to extract $content$. $content$ in turn is either again of the form $message$, i.e. consists of one or more parts, or is an element of $\mathcal{D}$ that is neither header or channel nor contains reserved terms like `plain,noConst` etc. (see below). Thus a message for example sent from $Backend$ to *T-ECU* via the $GPRS$ component can be constructed as $((IP(Backend), IP(\text{T-ECU})), (header_1, ch_1, content_1))$ with $content_1 = (header_2, ch_2, data)$, and $data$ denoting the actual data.

2) Security Mechanisms: We model security mechanisms by way of abstract communication channels on various different abstraction levels. For channel specification we use the following data type sets:

- $ChType$ contains all types of channels, `aesencCh`

e.g. denotes the type of an AES encryption channel. We distinguish between concealing and not concealing channels (a channel cannot be both at the same time). The specific channel type `plain` denotes a channel that is not cryptographically secured.

- $Const$ contains all constituents of channels, i.e. essential data needed in order to establish a channel, and the constant `noConst`. Constituents depend on the channel type. For example, the constituents of a channel that is established simply by using a cryptographic algorithm and key contain just this key. A channel containing the constant `noConst` is called a basic channel. While any cryptographic channel is established by using specific constituents, we take a basic channel's security properties for granted, thus its constituents are irrelevant for an analysis.
- Rnd denotes the set of all random numbers and also contains the constant `noRnd`.
- $Keyowners := (Agents \times Agents) \cup Agents$ contains the names of regular agent pairs sharing a symmetric key and of agents being the owner of an asymmetric private key.

Constituents are again specified on different abstraction levels. A basic cryptographic key for example contains among others the parameter *basic*, while a level 1 key contains the basic channel used to establish it. Analogously certificates can be specified on basic or higher abstraction levels. These different abstraction levels of constituents allow to model different levels of channels: Level 1 channels use basic constituents, level 2 channels use level 1 constituents, etc. An example of a channel is $(\texttt{aesencCh}, \texttt{noConst}, (Backend, \textbf{\textit{T-ECU}}), \texttt{noRnd})$.

We use algebraic functions to determine e.g. the type of a channel, its constituents, its inverse, the type and inverse of a key, etc.

We assume cryptographic algorithsm to be unbreakable, meaning that e.g. a specific unguessable key is necessary to decrypt a ciphertext. The resulting properties of security mechanisms are captured by axioms. Examples are

A1 An agent can only extract $content$ from a tuple $(header, ch, content)$ if ch is not a concealing channel or if it owns its inverse.
The inverse of the example channel above for instance is identical except for the first parameter being `aesdecCh`.

A2 An agent can generate a channel if it owns the channel's constituents (i.e. if they are elements of its state components).
Hence higher level channels (actually containing constituents) allow analysis e.g. of channel establishment. Since basic channels only contain the constant `noConst`, consequently they cannot be generated by an agent in the course of the system running. An agent, in particular an adversary, either owns and can use such a channel or it does not, thus allowing to focus on structural issues the system to be analysed might have.

transition name	
$(sender, recipient, message)$	local variables
$(sender, recipient, message) \in GPRS$	conditions
$\overset{automaton}{\rightarrow}$	
$(sender, recipient, message) \hookleftarrow GPRS$	actions
$(\ldots) \hookrightarrow GPRS$	

TABLE I
STATE TRANSITION RELATION

The idea of modeling communication by using abstract channels is not new (see Section VII). However, the different levels of channels provide the means to focus on different aspects of the analysis without changing the way security mechanisms are modeled which makes our model very flexible. Our abstract channels and their constituents together with the axioms constitute a system of formal cryptographic primitives that is on the one hand abstract but on the other hand captures all elements that are relevant in reality. An APA specification compliant to our model must identify the concealing channels and respect the axioms.

3) State Transition Patterns and Initial State: To specify the agents' actions we use so-called *state transition patterns* describing state transitions of elementary automata. Table 1 shows its structure. The lines above the symbol $\overset{\textbf{automaton}}{\rightarrow}$ indicate the necessary conditions for the automaton to transform a state transition. Examples for conditions are that $GPRS$ contains a message tuple $(sender, recipient, message)$, that $recipient$ is equal to the automaton's name, that the message has a specific structure, that $automaton$ owns the inverse of a channel used within the message, etc. The lines below $\overset{\textbf{automaton}}{\rightarrow}$ specify the changes of the state performed by $automaton$. $\hookrightarrow$ and $\hookleftarrow$ denote that it adds/removes data to/from a state component. $automaton$ does not perform any other changes within this state transition.

Finally, the initial state has to be specified. It contains in particular the channels owned by the automata. The syntax and semantics of state transition patterns for APA are explained in more detail in [10].

4) Roles: In the above state transition pattern as well as in the protocol description to be introduced in Section V, the acting entities are roles rather than agents, any smartphone acting as $Terminal$ will behave in the way specified by the respective patterns. For the analysis of a concrete scenario, roles are then instantiated with concrete agents (see Section V).

5) Introducing Dishonest Behavior: Our model includes the explicit specification of dishonest behavior reflecting the abilities of a Dolev-Yao attacker as explained in Section II. Let A denote the elementary automaton of an attacker, C_{sh} a state

component shared with other elementary automata, and C_A any state component in its neighborhood relation not shared. Then A can

1) remove tuples $((sender, recipient), message)$ from C_{sh} or leave them there and add them to C_A,
2) decompose tuples $((sender, recipient), message) \in C_A$ and
   a) write $sender, recipient$ and $message$ to C_A,
   b) decompose $m = mpart_1, \ldots, mpart_k$ and write all $mpart_i$ to C_A,
   c) for $mpart_i = (header_i, ch_i, content_i)$, write $header_i$ and, if ch_i is not a concealing channel or if A owns its inverse channel, $content_i$ to C_A,
   d) for $header_i = (h_1, \ldots, h_r)$, write all $h_j\,(j \in \{1, \ldots, r\})$ to C_A,
   e) for $content = (d_1, \ldots, d_i) \in C_A$with all d_i neither header nor channel, decompose $content$ and write all d_i to C_A,
   f) for $D_1, \ldots, D_r \subseteq \mathcal{D}$ and $d_i \in D_i\,(i = 1, \ldots, r)$ being elements of any C_A, construct $(d_1, \ldots, d_r) \in D_1 \times \ldots \times D_r$ and add it to C_{sh}.

Note that A cannot extract a channel ch_i from any $mpart_i$.

6) Security Properties: A security property holds for a system specified by an APA if it holds in every state of this APA, or, in other words, if the APA does not contain a state in which the security property is violated. In the past we addressed confidentiality [11] and fair non-repudiation [5]. In this paper we focus on a specific authenticity and freshness property: Each time automaton a_1 performs a specific state transition t, automaton a_2 must have performed a corresponding state transition t' before. In order to evaluate this property we use a global state component Log in which all relevant actions are recorded in the order of occurence.

IV. Tool Support

While simple protocols and systems can be analyzed by hand, for more complex ones tool support is inevitable. We use the Simple Homomorphism Verification Tool (SHVT) [9] developed by Fraunhofer SIT that provides support for finite state based verification of APA.

A. The Simple Homomorphism Verification Tool

The SHVT provides components for the complete cycle from formal specification to exhaustive validation as well as visualization and inspection of computed reachability graphs. The tool manages the components of the model, allows to select alternative parts of the specification (e.g. alternative attack scenarios), and automatically "glues" together the selected components to generate a combined APA model. After an initial state is selected, it automatically computes the reachability graph. It can also be used for simulation, i.e. user-guided computation of paths in the reachability graph. The SHVT stops if either no further state transition is possible or the reached state satisfies a property that is specified as *break condition*. It then can compute paths in the reachability graph from this state to the initial state, thus showing ways this state can be reached. The following figure shows the first steps of a reachability graph. The nodes M-1, M-2 etc. represent the states of the system. A context menu of a node allows e.g. to show the content of all state components of the respective state. The edges show the names of the respective state transition relation from one state to the next. Other SHVT functions allow to show a list of all local variables used in a specific state transition relation and their values.

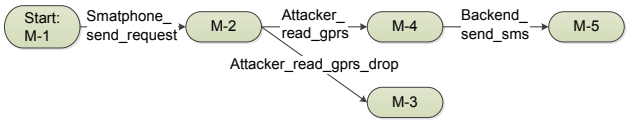

Fig. 3. Example of a reachability graph

B. Restricting Attack Behavior

As with any other model checking tool, we need to restrict the number of reachable states in order for the SHVT to terminate. First of all, the legitimate agents' behavior should be limited so as to obtain a finite state set for the APA without attacker. This can be achieved for example by requesting a specific condition to hold (e.g. knowledge of an IP address) for a specific automaton in order to perform a state transition and disabling the condition (by removing used IP addresses). The number of protocol runs can hereby be restricted to the number of IP addresses in the initial state. However, including attack behavior as generic as described in Section III-A5 would lead to infinitely many states (A could for example construct infinitely many wrappings $(header_i, ch_i, (header_{i+1}, ch_{i+1}, \ldots))$). So we need to restrict this behavior. On the other hand, any restrictions should not result into missing possible attacks. In the following, we describe the restrictions applied so far.

Message construction templates We restrict the tuples that A constructs and adds to the shared components by defining categories corresponding to the types of atomic data (e.g. a category $numbers$ with data values $1, 2, \ldots$). These are then used by A to identify the underlying structure of a tuple read from a shared component in terms of categories. The *message construction template* for a message sent from $Backend$ to *T-ECU* via GPRS for example can be (2,IP(Backend),IP(*T-ECU*), [*cat1,cat2*]). The first element specifies the number of data values that have been generalized to categories, the following two elements denote sender and recipient (taken from the read $GPRS$ tuple), and the fourth

element contains the actual data structure (here containing two categories $cat1$ and $cat2$). The template is then stored in one of A's state components. If in this state A finds an element to belong to more than one category (e.g. the number 1 that belongs to both $nat0$ and Rnd), it constructs templates with all possible combinations. An element belonging to no category is kept as is. When constructing tuples to be added to a shared component, A uses the respective templates and analogous header and channel construction templates (currently being added manually) and non-deterministically instantiates the categories by the respective values. See Section V for a concrete example.

By assigning equivalence classes to categories we can relax the restrictions posed by categories. Whenever A encounters a data value belonging to a specific category, it adds this category and all those that are equivalent to the message construction template. This enables to analyze a wider range of attacks that deploys an agent's disability to recognize the type of message items (being a random number, a date, etc).

Optimizing order of actions Assume a tuple in a shared component (e.g. GSM) can be processed by an automaton not equal to A. If A can add another tuple to the component, the SHVT non-deterministically processes the possible state transitions and thus doubles the subsequent state space. In order to avoid this, we allow A to add tuples to shared components only if the respective component is empty. Another even more important consequence is that all paths in which A adds a tuple to a shared component that none of the legitimate agents can process are not continued.

V. Analysis of a Real-World Example

In this section we demonstrate the usefulness of our approach by applying it to a protocol previously used by a car manufacturer which implements the use case of unlocking a car's doors by a smart phone.

A. The Example

As stated in Section I, the example protocol was already found to be flawed [2]. Yet it still serves as a proof of concept as it represents a real word example.

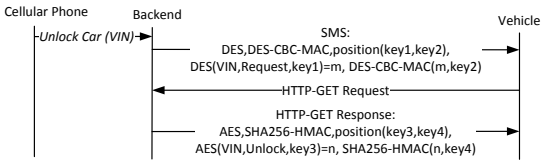

Fig. 4. Protocol presented in [2]

Figure 4 illustrates the protocol as it was presented in [2]. First, the user, by using his/her mobile phone, requests the manufacturer's backend server to open his/her car's door by sending the command *Unlock Car (VIN)* (where VIN denotes the car's Vehicle Identification Number). Next, the backend server sends a wake up message including the VIN to the car's telemetric ECU via Short Message Service (SMS). This message is encrypted with DES and authenticated with an DES based message authentication code (DES-CBC-MAC). The message further contains information indicating to the recipient which algorithms and keys to use for decryption and MAC verification. Once having checked the MAC and that the VIN is correct, the telemetric ECU activates more powerful computation parts. It then sends an HTTP-GET request to the backend server in plaintext which answers with an HTTP-GET response including again information on the cryptoalgorithms and keys to use. This message further contains the *Unlock* command and the VIN, encrypted with AES and authenticated with an HMAC based on SHA256. If this message is accepted the doors will be unlocked.

For this scenario we distinguish between three different attacker models which differ in the individual foreknowledge:

- Sc1 The weakest attacker does not know any keys. All she can do is to record messages and to replay them into the communication channel.
- Sc2 The second attacker knows the DES keys. This assumption is reasonable since a DES key can be revealed within a day by a brute-force attack [12].
- Sc3 The most powerful attacker (resembling the author of [2]) owns a car that she is able to analyze in depth. Therewith she learns all secret keys that are required for the protocol.

B. The Scenario APA Model and its Instantiations

We use the APA introduced in Figure 2. In order to include dishonest behavior, we extend it by one elementary automaton A with state components for storing addresses, message headers, values of atomic data, message parts, messages, etc. A in particular owns a component Ch_A that contains all channels A knows. A has full access to GSM and $GPRS$ but cannot access Bus.

As explained in Section III-A, the resulting APA and the corresponding state transition relations model roles rather than concrete entities. By instantiating these roles we model the attack scenarios described above. The scenarios have in common one instance of the role Backend, its state components initially contain all data needed in order to process a legitimate protocol run.

We then distinguish the three scenarios as follows: Sc1 with the weakest attacker which does not own any keys is

modeled by further assigning one instance each to the roles $Terminal$, *T-ECU* and ECU_i, respectively, the latter acting as the device unlocking the car's door. In the initial state, Ch_A does not contain any channels except `plain`. Sc2 is modeled by also using only one entity each for $Terminal$, *T-ECU* and ECU_i. However, by adding the DES encryption and DES-CBC-MAC generation channels to the initial Ch_A, we model the fact that the attacker knows the DES keys. For Sc3 we instantiate the roles $Terminal$, *T-ECU* and ECU_i each with two entities, thus modeling two cars, distinguished by $VIN1$ and $VIN2$, respectively. A owns one of them (*T-ECU*$_1$ and ECU_{i1}) and knows all its keys, modeled by adding the respective channels to the initial Ch_A. The fact that all cars use the same keys as reported in [2] is modeled by adding the channels also to *T-ECU*$_2$'s channel component. Since we do not focus on channel establishment we use only basic channels.

The security property we focus on in this paper and that we will show not to hold for the example scenario is as follows: Each time ECU_i processes a car unlock command, it must have been initiated by $Terminal$.

While in reality a message can get lost, in our model all messages are either processed by a legitimate agent or dropped by the attacker. Hence a state violating the above property can be characterized by the Log component containing more open car actions processed by ECU_i than open car requests by the respective $Terminal$ for a given car. The APA model satisfies the above property if its state set does not contain such a state.

C. SHVT supported analysis

In order to evaluate the above property, we use a function f that checks whether the number of `openCar` actions by ECU_i is bigger than the respective number of requests by $Terminal$, and set as break condition $f(Log) = \texttt{true}$. In our current implementation, $Terminal$ can only start one run (it knows only one IP address), and the attacker only listens to this run and learns messages. Hence in each state of this run, $f(Log)$ evaluates to `false`. We then start a second run in which $Terminal$ cannot perform any state transition (it does not know any IP addresses), but A can construct messages and add them to GSM and $GPRS$, using the knowledge gained in the first run. If there is then a state in which $f(Log) = \texttt{true}$, the car unlocking must have been initiated by the attacker, hence an attack is found. We can easily generalize this implementation by logging the attacker's dropping actions as well and specifying $f(Log) = \texttt{true}$ if the number of openCar actions is bigger than the number of $Terminal$'s initiation actions less the number of the attacker's dropping actions.

Replaying Messages In Sc1, A can only read from and write to GSM and $GPRS$. Having learned all tuples sent by $Backend$ in one run, it adds the SMS tuple that had been used in the first run again to GSM and extracts *T-ECU*'s IP address from the reply. It then uses this IP address to construct a new header (according to the header template currently defined manually) and uses this (instead of the first run's $GPRS$ tuple header) in the GPRS tuple. *T-ECU* accepts this tuple (except for the IP address it is identical to all such tuples received in unlock car processes) and induces ECU_i to unlock the door. Hence the break condition is fulfilled and the SHVT stops.

Unlocking Someone Else's Car This attack corresponds to Sc3. In order to prohibit message replay, we enhanced the protocol by adding a sequence number to the messages (we forgo the description of the technical details). Then, in the first run involving only $Terminal_1$, since in Sc3 A knows all channels used by *T-ECU*$_1$, it learns all headers and the actual data of the GSM and GPRS tuples, respectively. This actually models the state of the author of [2] after having hacked his own car. A then listens to an unlock car process initiated by $Terminal_2$ thereby learning the SMS address of *T-ECU*$_2$ and the car's VIN2.

We then start another analysis run with all A has learned in the previous ones added to its initial knowledge components, and allowing it to construct messages. Using the message and channel construction templates for the SMS message, A constructs and adds tuples to GSM. While some of these tuples will not be processed by the recipient because they are correct with respect to the message format but contain the wrong header or data, one of these tuples contains *T-ECU*$_2$'s SMS address and $VIN2$. Since *T-ECU*$_2$ uses the same channels as *T-ECU*$_1$, it accepts this message and answers with the HTTP_GET request. A then uses the message and channel construction templates and the known channels to construct GPRS tuples. Again, some of them are not accepted because of containing the wrong data or header. However, one contains *T-ECU*$_2$'s current IP address, the correct data and uses the correct channels. Thus *T-ECU*$_2$ accepts this tuple and induces ECU_{i2} to unlock the car's door. This is the only action recorded in the Log component, hence the break condition is again fulfilled and the SHVT stops.

The SHVT also found this attack in one run involving only $Terminal_1$ by adding *T-ECU*$_2$'s SMS and $VIN2$ to its initial knowledge (according to [2], A can easily learn them).

Downgrading of Security Mechanism Let us now consider Sc2 in which A only owns the DES channels. Having just listened to one round of communication and having extracted all possible items, it knows both headers and the content of the GSM tuple but only the outer header of the GPRS tuple. Having extracted the VIN from the GSM tuple, in a second run it can construct the GSM tuple and send it to *T-ECU* which will respond with an HTTP-GET request. For constructing the GPRS tuple that actually induces the door to be unlocked, instead of using the AES and SHA256HMAC header and channels (which A cannot use since it does not own the channels), it downgrades the channels to be used by *T-ECU* by using the DES and DES-CBC-MAC headers and channels. In our model, *T-ECU* does not distinguish between tuples

received via GSM and $GPRS$, hence it uses the channels indicated by the headers, verifies the MAC to be correct and induces ECU_i to unlock the door. Again, the break condition holds and the SHVT stops.

The analysis presented in [2] does not reveal whether or not the telemetric ECU would accept this message. A good implementation would in the first place not use an outdated algorithm such as DES. It would further not allow downgrading of mechanisms or at least apply further measures to prohibit the attack (e.g. allowing DES channels only to be used for messages received via GSM). Yet, using the same keys for all cars as was the case in the analyzed system is not considered good practice either. The above described attack, if possible, would constitute a major weakness of the scheme: The purpose of the MAC, namely to prove the origin of a message, is not achieved, and the only barrier that is left in order to successfully perform an attack is to get to know the actual message that must be sent. What is worse, if downgrading of the used cryptographic algorithm is possible, the two car scenario becomes insecure even if each car uses its own keys, as A can discover the structure of all messages sent to its own car.

While the first two attacks described above have been reported in [2] and depend on weaknesses that are well-known, to the best of our knowledge the last one has not been mentioned yet.

VI. Formal Model of a Bus System

So far we considered traditional security protocols that are developed for point to point connections. These unicast communication channels have only two communication partners, so every message has a single sender and a single receiver.

In contrast, automotive networks primarily are built on bus systems like CAN, CAN FD or Flexray. Senders in these networks broadcast messages to every entity in the network. The intended recipients process the message, the other entities simply ignore it. For example the brake signal is broadcast and processed by brakes but also by the anti-lock braking system and perhaps by electric pretensioners.

This communication model common in automotive systems is not covered in formal analysis so far. However, new protocols are implemented for secure communication in vehicle networks. One prominent example is the Secure Onboard Communication (SecOC) from AUTOSAR [31]. It requires to implement protocols for key distribution as well as synchronization of freshness values. Formal analysis can be used to investigate these protocols and prove their security features.

A. Modeling a Fieldbus

A proper model of a fieldbus system needs to deliver a message from one sender to multiple recipients. Moreover each recipient must have the opportunity to process the message before it gets altered or deleted by another recipient. Additionally the model must allow an attacker to perform the same types of attacks that are possible on a real bus, i.e. the attacker must be able to read, alter and create messages in the communication channel.

Our APA model must satisfy these requirements. If we model fieldbus communication by using a shared state component equivalent to GSM and $GPRS$ where each ECU can read from and write to, we encounter a problem as we would need to determine which ECU eventually should remove the message. Any artificial order of ECU read actions that could be used to assign the "last" read action to a specific automaton that can then delete the message would not reflect the real behavior of the system.

In order to solve this issue we split the state component representing the fieldbus into two component types: Each automaton ECU_i can write to *Write-Bus*, only automaton ECU_i has read access to its own component *Read-Bus*$_i$. Furthermore we introduce a global automaton $Fieldbus$ that manages message distribution, i.e. reads a message from *Write-Bus* and adds it to all *Read-Bus* components. The Attacker automaton can both read from and write to *Write-Bus* but has no connection to any *Read-Bus* component. This allows the attacker to arbitrarily manipulate messages sent on the fieldbus and add new ones but also enforces messages that are processed by more than one recipient to be identical. This structure is illustrated in figure 5, read and write capacities are indicated by arrows.

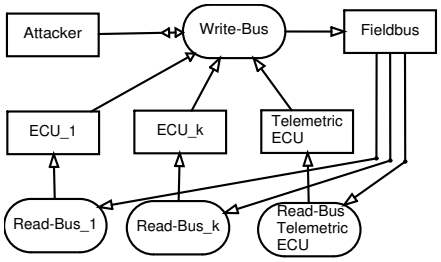

Fig. 5. APA model of an In-Car network

The state set of *Write-Bus* is slightly different to the ones of GSM and $GPRS$ and contains triples $(sender, messageID, message)$ with the structure of $message$ as explained in Section III. $messageID$ needs to be transformed into the actual recipients by $Fieldbus$, hence the state sets of the *Read-Bus* components are identical to those of GSM and $GPRS$.

B. Optimization for APA

We can optimize the presented model by implementing the global $Fieldbus$ automaton with knowledge about which message will be processed by which ECU. $Fieldbus$ will

then forward the message only to these recipients which avoids all state transitions where ECU_i removes a message from *Read-Bus*_i without processing it. This change drastically reduces the state space and thus allows to cover more cases in an analysis.

VII. Related Work

Formal verification of protocols is a well-established research area. Mostly, it is based on an abstract (term-based) representation of cryptographic primitives that can be automatically verified using model checking and theorem proving tools. Early work can be traced back to [13] and [14], see [15] for a survey. Newer approaches include AVISPA [16] which provides the High Level Protocol Specification Language (HLPSL [17]) and four different analysis tools. AVISPA has been successfully used for example to identify a security flaw in the SAML-based Single-Sign-On protocol for Google Applications [18]. An extension of AVISPA, called AVANTSSAR [19], is able to reason about the dynamic composition of services.

More recently, Blanchet introduced an approach using Horn clauses to represent a protocol and to specify implications between facts (predicates on messages) and thus to prove security properties [20]. The approach is automated by ProVerif [21] and has already been successfully applied for the analysis of numerous protocols. ProVerif either finds an attack (which in some cases may not work in reality as Horn clauses represent a protocol abstraction), or a proof of a specific security property holding, or it returns "cannot be proved" in cases where neither the one nor the other can be achieved.

Another approach recently introduced whose underlying formal semantics is very similar to ours is Tamarin [22]. A protocol is modeled as a labeled transition system using a set of (labeled) multiset rewriting rules. Security properties are expressed as trace properties, the adversary and the protocol interact by updating network messages and freshness information. So far it is focused on protocols that use DH exponentiation, but it can also deal with other, user-defined cryptographic primitives. Tamarin can handle an unbounded number of protocol runs and fresh data. If it terminates (which is not guaranteed), it returns either a proof of correctness or a counterexample.

An analysis of the web based authentication protocol OAuth2 [23] introduced last year [24] focusses on web standards and specifications and is not supported by a tool.

These approaches (and others not mentioned here) certainly have their merits. They have been shown to be very useful for the analysis of cryptographic protocols. However, they all model unicast communication and are thus not able to analyse automotive systems that utilize other types of communication, namely fieldbus.

Approaches focusing particularly on automotive systems so far analyze the attack surfaces of vehicles ([25], [6]) and identify attack points like the OBD-II port, media/CD player, Bluetooth, tire pressure monitoring systems (TPMS), remote keyless entry, etc.

As explained in Section I, testing revealed multiple problems in the telemetric systems as well (see e.g. [2], [26], [3]). However, these findings required a large technical knowhow and special skills and resulted in single vulnerabilities. Covering the whole complexity of the system however would require much more time. Further, the testing approach can ascertain security weaknesses of the tested system but lacks the ability to make statements about the provided security properties. For that reason we see the necessity of a formal approach.

Another aspect of our approach is the notion of abstract channels which has been used in many approaches before. Most of them aim at proving the security of vertical protocol composition (see for example [27], [28]): A protocol specification P uses an abstract channel that is assumed to provide specific security properties, another protocol Q specifies channel establishment. Security of the composition is then proven by independently proving the security of P and Q, and using specific conditions on the refinement. In [29], Rudolph uses channels on two different abstraction levels for the top-down design of cryptographic protocols and proves specific security properties on the basis of APA specifications.

In contrast, our channel model allows to specify channels not only on two but on arbitrarily many abstraction levels without changing the way security mechanisms are modeled. This makes our model very flexible.

VIII. Conclusions and future work

In [10] we introduced an approach for protocol analysis based on Asynchronous Product Automata and supported by the Simple Homomorphism Verification Tool. In this paper we provide an extension of our previous work that firstly concerns a very flexible model of cryptographic primitives that allows to either focus on the correct usage of mechanisms such as TLS, HTTPS etc. or assumes them to work correctly and under this assumption investigates whether the system achieves the desired security properties. Secondly, we showed how to model fieldbus communication that until today is still the basis of any in-car network. To the best of our knowledge, our approach is the first one capable of modeling automotive systems that go beyond classical unicast communication channels.

To show the adequateness of our approach we applied it to a specific telemetric system deployed by a German car manufacturer and described and analyzed in [2]. Our analysis revealed the same security weaknesses already reported in [2]. Furthermore, we were able to detect another possible weakness not yet reported which concerns the ability of downgrading one

of the security mechanisms used under specific assumptions and herewith of violating data origin authenticity of the car unlock command.

Our approach is useful to find structural weaknesses such as the ones presented in this paper. However, it cannot address implementation failures such as buffer overflow, side channel attacks etc. These go beyond its capabilities. In this sense it complements other analysis approaches, for example penetration tests that can help to verify the correctness of the implementation and configuration.

Our decision to keep using the SHVT given other available tools is motivated by its flexibility which we effectively showed by including fieldbus communication in our APA model. In the automotive domain this constitutes an important improvement as it allows to cover in-car communication in security analysis.

Future work on the tool includes to generalize the generation and usage of message construction templates to enable the automatic handling of arbitrary messages. This entails to investigate the limits of such generalization as too complex messages may cause explosion of the state space. The SHVT is currently not freely available (but can be purchased from Fraunhofer SIT), yet it has been made available to partners of various research projects.

As already pointed out in Section I, several examples of automotive systems exist that have failed to provide necessary security properties. Moreover, current activities by for example ISO and AUTOSAR work on including appropriate use of security mechanisms in the development of automotive systems ([30], [31]). The SecOC document by AUTOSAR for example specifies the synchronisation of freshness counters used by ECUs to prohibit message replay. Our fieldbus APA model enables the analysis of this and other protocols that are currently being developed. In fact, these will be topics of future work that additionally includes an improvement of our sequence number model. Another interesting open question that we plan to work on is to which extend it can be formally proven that the restricted attack behavior does not hide possible attacks.

REFERENCES

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, and S. Savage, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*, May 2010, pp. 447–462.

[2] D. Spaar, "Beemer, Open Thyself! - Security vulnerabilities in BMW's ConnectedDrive," *c't 05/2015*, 2015, http://heise.de/-2540957.

[3] C. Valasek and C. Miller, "Remote exploitation of an unaltered passenger vehicle," *IOActive, Seattle, USA. Available: http://www. ioactive. com/pdfs/IOActive_Remote_Car_Hackin g. pdf. Last visited*, vol. 18, 2015.

[4] "Cybersecurity guidebook for cyber-physical vehicle systems," SAE INTERNATIONAL, East Lansing, Michigan, Tech. Rep. J3061, January 2016.

[5] S. Gürgens and C. Rudolph, "Security Analysis of (Un-) Fair Non-repudiation Protocols," *Formal aspects of computing*, 2004.

[6] C. Miller and C. Valasek, "A survey of remote automotive attack surfaces," *black hat USA*, 2014.

[7] A. Dabrowski, N. Pianta, T. Klepp, M. Mulazzani, and E. Weippl, "Imsi-catch me if you can: Imsi-catcher-catchers," in *Proceedings of the 30th annual computer security applications Conference*. ACM, 2014, pp. 246–255.

[8] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche, "The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems," *Formal Aspects of Computing, The Int. Journal of Formal Methods*, vol. 11, pp. 1–24, 1999.

[9] P. Ochsenschläger, J. Repp, and R. Rieke, "Abstraction and composition – a verification method for co-operating systems," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 12, pp. 447–459, 2000.

[10] S. Gürgens, P. Ochsenschläger, and C. Rudolph, "Role based specification and security analysis of cryptographic protocols using asynchronous product automata," in *DEXA 2002 International Workshop on Trust and Privacy in Digital Business*. IEEE, 2002.

[11] S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga, "Security evaluation of scenarios based on the TCG's TPM specification," in *Computer Security - ESORICS 2007*, ser. Lecture Notes in Computer Science, J. Biskup and J. Lopez, Eds., vol. 4734. Springer Verlag, 2007.

[12] SciEngines GmbH, "Break DES in less than a single day." 2009, http://www.sciengines.com/index.php?option=com_content&id=79.

[13] G. Lowe, "Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR," in *Second International Workshop, TACAS '96*, ser. LNCS, vol. 1055. SV, 1996, pp. 147–166.

[14] L. C. Paulson, "Proving properties of security protocols by induction," in *Computer Security Foundations Workshop, 1997. Proceedings., 10th*, Jun 1997, pp. 70–83.

[15] B. Roscoe, P. Ryan, S. Schneider, M. Goldsmith, and G. Lowe, *The modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

[16] AVISPA, "Research project IST-2001-39252," http://www.avispa-project.org/.

[17] Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, J. Mantovani, S. Mödersheim, and L. Vigneron, "A high level protocol specification language for industrial security-sensitive protocols," in *Workshop on Specification and Automated Processing of Security Requirements-SAPS'2004*. Austrian Computer Society, 2004, pp. 13–p.

[18] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra, "Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for google apps," in *Proceedings of the 6th ACM workshop on Formal methods in security engineering*. ACM New York, NY, USA, 2008, pp. 1–10.

[19] AVANTSSAR, "Research project IST-2001-39252," http://www.avantssar.eu/.

[20] B. Blanchet *et al.*, "Modeling and Verifying Security Protocols with the Applied Pi Calculus and ProVerif," *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.

[21] B. Blanchet, B. Smyth, and V. Cheval, "ProVerif 1.90: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial," *URL: http://prosecco. gforge. inria. fr/personal/bblanche/proverif/manual. pdf*, 2015.

[22] S. Meier, B. Schmidt, C. Cremers, and D. Basin, "The TAMARIN prover for the symbolic analysis of security protocols," in *International Conference on Computer Aided Verification*. Springer, 2013, pp. 696–701.

[23] E. D. Hardt, "The oauth 2.0 authorization framework," Internet Engineering Task Force (IETF), Tech. Rep., 2012.

[24] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of oauth 2.0 (2016) 1–75," *URL http://arxiv. org/abs/1601.01229*.

[25] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces," in *Proceedings of the 20th USENIX Conference on Security*, ser. SEC'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 6–6. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028067.2028073

[26] I. Foster, A. Prudhomme, K. Koscher, and S. Savage, "Fast and vulnerable: a story of telematic failures," in *Proceedings of the 9th USENIX Conference on Offensive Technologies*. USENIX Association, 2015, pp. 15–15.

[27] V. Cheval, V. Cortier, and E. Le Morvan, "Secure refinements of communication channels," Ph.D. dissertation, LORIA, UMR 7503, Université de Lorraine, CNRS, Vandoeuvre-lès-Nancy, 2015.

[28] S. Mödersheim and L. Viganò, "Sufficient conditions for vertical composition of security protocols," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*. ACM, 2014, pp. 435–446.

[29] C. Rudolph, "A model for secure protocols and its application to systematic design of cryptographic protocols," Ph.D. dissertation, Queensland University of Technology, 2001.

[30] ISO, "Road vehicles - cybersecurity engineering," http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=70918, 2016.

[31] AUTOSAR, "Specification of module secure onboard communication," https://www.autosar.org/fileadmin/files/standards/classic/4-3/software-architecture/safety-and-security/standard/AUTOSAR_SWS_SecureOnboardCommunication.pdf, 2016.