Formal Verification of an Authorization Protocol for Remote Vehicle Diagnostics

CHALMERS

Formal Verification of an Authorization Protocol for Remote Vehicle Diagnostics

PIERRE KLEBERGER
GUILHEM MOULIN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2013

TECHNICAL REPORT

Formal Verification of an Authorization Protocol for Remote Vehicle Diagnostics

PIERRE KLEBERGER
GUILHEM MOULIN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2013

Formal Verification of an Authorization Protocol for Remote Vehicle Diagnostics
PIERRE KLEBERGER
GUILHEM MOULIN

Technical report 2013:09
ISSN 1652-926X
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Gothenburg, Sweden 2013

Formal Verification of an Authorization Protocol for Remote Vehicle Diagnostics
Technical report
PIERRE KLEBERGER
GUILHEM MOULIN
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Remote diagnostics protocols have generally only considered correct authentication to be enough to grant access to vehicles. However, as diagnostics equipment or their keys can be stolen or copied, these devices can not be trusted. Thus, authentication alone is not enough to prevent unauthorized access to vehicles. In previous work, we proposed an authorization protocol to prevent unauthorized access to vehicles. In the automotive industry where lives are at risk and a certain liability is exacted on the manufacturer, their vehicles and its software, it is critical that such a protocol has no flaws. Thus, using formal methods to prove the correctness of protocol designs is an important step. In this paper, we formally prove that the proposed authorization protocol provides mutual authentication between the diagnostics equipment and the vehicle, and that it guarantees both secrecy of the distributed session key and freshness of the distributed authorization information. Our formal analysis is conducted using both the Burrows-Abadi-Needham (BAN) Logic and the ProVerif automated verification tool. To the authors' best knowledge, this is the first formally verified authorization protocol for remote vehicular diagnostics.

Keywords: remote diagnostics; formal verification; authorization protocol; connected car.

1 Introduction

Vehicles have over the last decades become increasingly dependent on software. A modern car has somewhere between 50–100 embedded computers, *i.e.*, Electronic Control Units (ECUs), depending on brand and model. These ECUs handle different tasks, such as engine control, anti-spin system, or mirror adjustment. So far, software updates and vehicle diagnostics are performed by bringing the vehicles to the repair shop, but this is currently starting to change. As wireless vehicular communications is introduced, such maintenance can be performed remotely. However, remote diagnostics and software download comes with great security challenges: At no time should unauthorized access to vehicles be given. Since the vehicle is safety-critical, such unauthorized access can compromise the safety requirements of the vehicle and open up a new domain of cyber-physical attacks. Several attack vectors against the car have already been identified and the impact of getting access to the vehicle has already been demonstrated [1, 2]. For example have researchers already demonstrated how brakes were disabled during a journey [1].

Several protocols for performing remote diagnostics and firmware updates have been proposed during the last decade, *e.g.*, [3–8]. Most of them have been designed with security in mind, but not all of them (*e.g.*, Diagnostics over IP [7, 8]). Unfortunately, security protocols are very hard to design and as have been seen many times before, vulnerabilities in them are sometimes found years later. Thus, in the automotive industry where lives are at risk and a certain liability is exacted on the manufacturer, their vehicles and its software, using formal methods to prove the correctness of protocol designs is an important step.

Today's diagnostics protocols use authentication to validate the identity of the diagnostics equipment. Diagnostics equipment holds authentication keys to access vehicles, however, since these keys can be stolen or copied, authentication alone is not enough to protect vehicles from unauthorized access [9]; indeed stolen equipment could then be used to access any vehicle accepting the authentication keys. In an attempt to tackle these issues, an authorization protocol, transparent to the diagnostics and software download protocols being used, was proposed by Kleberger and Olovsson [10]. An informal argument of the correctness of the protocol was also given with respect to the security properties: Mutual authentication, secrecy of the distributed session key, and freshness of the distributed authorization information. Even though the informal analysis looked promising, it is only from a formal analysis possible design flaws can be uncovered and the correctness of the protocol and its security properties can be guaranteed. Several techniques, mostly complementing each others, can be used to formally verify security properties; notable approaches including various logic and typing systems, theorem provers, and model checking. In this paper we present a formal analysis of the above properties, first using a belief logic, namely the Burrows-Abadi-Needham (BAN) Logic [11], then using ProVerif, an automated verification tool by Bruno Blanchet based on a subset of the π-calculus [12, 13]. This report is the extended version of the short paper published in [14].

The rest of this paper is organized as follows. In next section, the authorization protocol is introduced and the desired security properties are discussed. Section 3 and 4 present the analysis performed using the BAN Logic and ProVerif tool. The paper concludes by a presentation and discussion of our results in Section 5, a review of related work in Section 6, and finally, our conclusion in Section 7.

2 An Authorization Protocol for Remote Diagnostics

The vehicle *must never* allow any diagnostics access from outside, unless such access also has been granted. To grant access to vehicles, we have proposed an approach consisting of three steps: First, a Trusted Third Party (TTP) that governs the authorization was introduced. The TTP issues authorization tickets to the vehicle and diagnostics equipment in accordance to its stored security policy. Second, we proposed an authorization protocol to securely distribute these tickets to the vehicle and diagnostics equipment. Finally, we proposed to implement a fine-grained access control mechanism in the network stack of the gateway ECU[1] in the vehicle based on the authorization information provided in the distributed ticket. Only a brief overview of the protocol is given here. For a detailed description of all steps, see [10].

2.1 Assumptions and Limitations

The protocol is designed to support an unlimited number of TTPs. Thus, depending on region, situation, and organization, authorization information can be placed at different locations as needed. For example, one TTP can be acknowledged for usage by the repair shops of an automotive company to authorize both remote diagnostics and software download to the vehicles, while another TTP can be acknowledged to authorize the vehicle inspection authority to only acquire the vehicles' health information. To decide which TTP to be used in an authorization request, the identity of the TTP is either specified in the initialization of the protocol or looked up as part of the protocol (see [10]). We believe that for most authorization requests the identity of the TTP is already known, *i.e.*, it is already preconfigured in the diagnostics equipment which TTP to submit the authorization request to.[2] Therefore, without loss of critical functionality, in this paper we describe and prove the authorization protocol when the identity of the TTP is initially known to the diagnostics equipment.

2.2 Authorization Architecture

Three entities are involved in the protocol (as shown in Figure 2.1):

- The *Diagnostics Equipment (DE)* that wants to establish a diagnostics session to the vehicle.
- The *Vehicle (V)* to be diagnosed.
- The *Trusted Third Party (TTP)* that governs the access to vehicles in accordance to its stored security policy. The security policy describes (1) a timespan for which the authorization is valid, (t_{begin}, t_{end}), (2) a set of authorized diagnostics equipment, and (3) an optional policy, $\mathrm{V}_{\mathrm{policy}}$, that describes the diagnostics messages authorized to be performed against the vehicle.

[1]The ECU that connects the in-vehicle network to external networks, *e.g.*, the Communications Control Unit (CCU).

[2]The TTP must also accept such a request from the specific diagnostics equipment.

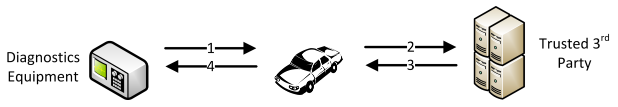

Figure 2.1: *Messages exchanged during authorization*

Table 2.1: Authorization Protocol

(a) Protocol

(1)	**DE** $\rightarrow$ **V**	$\mathrm{Cert}_{\mathrm{DE}} \parallel \mathrm{Cert}_{\mathrm{TTP}}$
(2)	**V** $\rightarrow$ **TTP**	$\mathrm{Cert}_{\mathrm{DE}} \parallel \mathrm{Enc}_{\mathrm{TTP}}(\mathrm{V}_{\mathrm{ID}} \parallel \mathit{nonce})$
(3)	**TTP** $\rightarrow$ **V**	$\mathrm{Enc}_{V}(\mathit{Ticket}_{\mathrm{V}} \parallel \mathit{nonce}) \parallel \mathrm{Enc}_{\mathrm{DE}}(\mathit{Ticket}_{\mathrm{DE}})$
		$\mathit{Ticket}_{\mathrm{V}} = \mathrm{Sign}_{\mathrm{TTP}}(K_{\mathrm{V,DE}} \parallel \Delta t \parallel \mathrm{V}_{\mathrm{policy}})$
		$\mathit{Ticket}_{\mathrm{DE}} = \mathrm{Sign}_{\mathrm{TTP}}(K_{\mathrm{V,DE}} \parallel \Delta t \parallel \mathrm{V}_{\mathrm{ID}})$
(4)	**V** $\rightarrow$ **DE**	$\mathrm{Enc}_{\mathrm{DE}}(\mathit{Ticket}_{\mathrm{DE}})$

(b) Notations

DE	Diagnostics Equipment
V	Vehicle
TTP	Trusted Third Party
$\mathrm{Cert}_{\mathrm{X}}$	Certificate of device X
$\mathrm{Sign}_{\mathrm{X}}$	Signature created by device X, using its private key
$\mathrm{Enc}_{\mathrm{X}}$	Encryption created for device X, using X's public key (also includes a Message Integrity Code)

Furthermore, each device is associated with a certificate and is in possession of the corresponding private key. These certificates have been issued by a Certificate Authority (CA) within the automotive company (or its appointed contractor) and all involved devices trust this CA directly or indirectly as a root of trust.

2.3 The Protocol

An illustration of the authorization protocol is shown in Figure 2.1. The protocol, together with its notation, is given in Table 2.1. After successful completion of the last step of the protocol, the authorization policy is known by the vehicle and both the vehicle and the diagnostics equipment share a common cryptographic session key. The protocol consists of the following messages:

1. **Initiate.** DE sends its certificate and the certificate of the TTP (which holds the authorization information) to V to request authorization of a diagnostics session.

2. **Authorization Request.** V sends its ID and the certificate of DE to TTP. (The address of TTP is found in the Common Name (CN) of TTP's certificate.) V records the time of the request, t_{req}.

3. **Authorization Response.** The TTP uses the vehicle's ID, V_{ID}, to find the vehicle's certificate and security policy within its database. A symmetric session key, $K_{\mathrm{V,DE}}$, to be

used by DE and V is generated and the duration of the authorization is calculated from the authorized timespan: $\Delta t = t_{end} - t_{current}$. Two authorization tickets are generated and sent back to V. $\mathrm{V}_{\mathrm{policy}}$ is optional and may contain what commands DE is allowed to execute.

4. **Protocol Completion.** The security policy and the session key are updated in V and the expiration time of the authorization is calculated: $t_{exp} = t_{req} + \Delta t$. Since the vehicle knows that $Ticket_{\mathrm{V}}$ is fresh (which will be formally proved in following sections) and the time of the request, the clocks in V and the TTP do not need to be synchronized. Furthermore, the authorization ticket is sent encrypted to DE to complete the authorization.

2.4 Desired Security Properties

We want to prove that the following security properties hold:

$\mathbf{R_1}$: Mutual Authentication. The secret session key generated by the TTP is only distributed to DE and V, thus providing mutual authentication between them.

$\mathbf{R_2}$: Secrecy of session key, $K_{\mathrm{V,DE}}$. An adversary cannot obtain the clear text of encrypted messages sent between DE and V.

$\mathbf{R_3}$: Freshness of Δt. The authorization tickets are not delayed or replayed, such that it can affect the calculation of the duration of the authorized session.

3 Proof using BAN Logic

The normalized version of the protocol is given in Table 3.1. In general, all certificates are represented by the corresponding public key, signed by a CA. Furthermore, it is assumed that encrypted messages can not be modified without detection. A Message Integrity Code (MIC) is therefore added to encrypted messages.

Table 3.1: Normalized Authorization Protocol

(1)	**DE → V**	$\{K_{\mathrm{DE}}\}_{K_{\mathrm{CA}}^{-1}}, \{K_{\mathrm{TTP}}\}_{K_{\mathrm{CA}}^{-1}}$
(2)	**V → TTP**	$\{K_{\mathrm{DE}}\}_{K_{\mathrm{CA}}^{-1}}$
(3)	**TTP → V**	$\{Ticket\}_{K_{\mathrm{V}}}, \{Ticket\}_{K_{\mathrm{DE}}}$
		$Ticket = \{K_{\mathrm{V,DE}}, \Delta t, \mathrm{V}\}_{K_{\mathrm{TTP}}^{-1}}$
(4)	**V → DE**	$\{Ticket\}_{K_{\mathrm{DE}}}$

Note that the nonce is omitted in the idealized form, indeed its role is merely to avoid replay attacks, but these cannot be expressed in a logic such as BAN's which does not have a notion of time. Also, while V's policy is irrelevant in the idealized form, we keep its identity in the ticket to avoid Man-In-The-Middle attacks.

3.1 Inference rules

In this section, we recall the BAN Logic Inference Rules, as found in [11].

$$\textsc{SSign}\quad \frac{P \textbf{ believes } K_{Q,P} \qquad P \textbf{ sees } \{X\}_K}{P \textbf{ believes } Q \textbf{ said } X} \qquad\qquad \textsc{SEnc}\quad \frac{P \textbf{ believes } K_{Q,P} \qquad P \textbf{ sees } \{X\}_K}{P \textbf{ sees } X}$$

$$\textsc{ASign}\quad \frac{P \textbf{ believes } K_Q \qquad P \textbf{ sees } \{X\}_{K_Q^{-1}}}{P \textbf{ believes } Q \textbf{ said } X} \qquad\qquad \textsc{GetMess}\quad \frac{P \textbf{ believes } K_Q \qquad P \textbf{ sees } \{X\}_{K_Q^{-1}}}{P \textbf{ sees } X}$$

$$\textsc{AEnc}\quad \frac{P \textbf{ believes } K_P \qquad P \textbf{ sees } \{X\}_{K_P}}{P \textbf{ sees } X} \qquad\qquad \textsc{Nonce}\quad \frac{P \textbf{ believes fresh } (X) \qquad P \textbf{ believes } Q \textbf{ said } X}{P \textbf{ believes } Q \textbf{ believes } X}$$

$$\textsc{Jurisdiction}\quad \frac{P \textbf{ believes } Q \textbf{ controls } X \qquad P \textbf{ believes } Q \textbf{ believes } X}{P \textbf{ believes } X}$$

$$\textsc{Proj}_1\quad \frac{P \textbf{ sees } (X, Y)}{P \textbf{ sees } X} \qquad\qquad \textsc{Proj}_2\quad \frac{P \textbf{ sees } (X, Y)}{P \textbf{ sees } Y}$$

$$\textsc{Fresh}_1\quad \frac{P \textbf{ believes fresh } (X)}{P \textbf{ believes fresh } (X, Y)} \qquad\qquad \textsc{Fresh}_2\quad \frac{P \textbf{ believes fresh } (Y)}{P \textbf{ believes fresh } (X, Y)}$$

3.2 Assumptions

We make the following assumptions:

- The CA has issued all certificates and all principals trust the CA, thus believes the content of them.

$$\text{CA } \textbf{controls } K_{\mathcal{P}r}, \quad \mathcal{P}r \in \{\text{CA}, \text{DE}, \text{V}, \text{TTP}\}$$

$$\frac{\mathcal{P}r \textbf{ sees } \{X\}_{K_{\text{CA}}^{-1}}}{\mathcal{P}r \textbf{ believes } X}, \quad \mathcal{P}r \in \{\text{DE}, \text{V}, \text{TTP}\}$$

$$\mathcal{P}r \textbf{ believes } \{X\}_{K_{\text{CA}}^{-1}}, \quad \mathcal{P}r \in \{\text{DE}, \text{V}, \text{TTP}\}$$

- The symmetric keys are generated by the TTP. V and DE therefore believes that the key is under the control of the TTP. Furthermore, generated keys are assumed to be unique for each request. Indeed the key-space should be big enough and keys should have enough entropy so that we can use them as nonces.

$$\text{V } \textbf{believes} \text{ TTP } \textbf{controls } K_{\text{V, DE}}$$

$$\text{DE } \textbf{believes} \text{ TTP } \textbf{controls } K_{\text{V, DE}}$$

$$\text{V } \textbf{believes fresh } (K_{\text{V, DE}}) \qquad \text{DE } \textbf{believes fresh } (K_{\text{V, DE}})$$

- TTP generates the time for which a ticket is considered valid: Δt. We can therefore assume:

$$\text{V } \textbf{believes} \text{ TTP } \textbf{controls } \Delta t$$

3.3 Proofs

We prove authentication (first order beliefs only; we discuss second order beliefs at the end of this section), *i.e.*, the key exchange between the two parties V and DE:

$$\text{V } \textbf{believes } K_{\text{V, DE}} \qquad \text{DE } \textbf{believes } K_{\text{V, DE}}$$

We also prove that Δt, issued by TTP, is trustworthy from V's perspective, meaning V can rely on it to grant or deny access to DE.

$$\text{V } \textbf{believes } \Delta t$$

We first prove that DE believes TTP issues the session key $K_{\text{V, DE}}$, a statement we then make use of to show that DE believes that $K_{\text{V, DE}}$ is a good key:

$$\dfrac{\text{Mesg. \#1 and assumption}}{\text{DE } \textbf{believes } K_{\text{TTP}}}$$

$$\textsc{Proj}\ \dfrac{\textsc{ASign}\ \dfrac{\dfrac{\dfrac{}{\text{DE } \textbf{believes } K_{\text{DE}}} \quad \dfrac{\text{Mesg. \#4}}{\text{DE } \textbf{sees } \{\textit{Ticket}\}_{K_{\text{DE}}}}}{\text{DE } \textbf{sees } \textit{Ticket}}\ \textsc{AEnc}}{\text{DE } \textbf{believes} \text{ TTP } \textbf{said } (K_{\text{V, DE}}, \Delta t, \text{V})}}{\text{DE } \textbf{believes} \text{ TTP } \textbf{said } K_{\text{V, DE}}}$$

$$\dfrac{\dfrac{}{\text{DE } \textbf{believes} \text{ TTP } \textbf{controls } K_{\text{V, DE}}} \quad \dfrac{\dfrac{}{\text{DE } \textbf{believes fresh } (K_{\text{V, DE}})}}{\text{DE } \textbf{believes} \text{ TTP } \textbf{believes } K_{\text{V, DE}}}\ \textsc{Nonce}}{\text{DE } \textbf{believes } K_{\text{V, DE}}}\ \textsc{Jur.}$$

The corresponding belief for V goes in a similar way. The only arguable difference is that, since V is passed back its (MIC'ed) nonce in Mesg. #3, we could have used N instead of $K_{\text{V, DE}}$ in the Nonce Verification rule.

$$\dfrac{\dfrac{}{\text{V } \mathbf{believes}\ \mathbf{fresh}\ (K_{\text{V, DE}})} \qquad \dfrac{\text{ASign}\ \dfrac{\text{Proj}_1\ \dfrac{\text{AEnc}\ \dfrac{\dfrac{}{\text{V } \mathbf{believes}\ K_{\text{V}}} \quad \dfrac{\text{Mesg. \#3 (after split)}}{\text{V } \mathbf{sees}\ \{\mathit{Ticket}, N\}_{K_{\text{V}}}}}{\text{V } \mathbf{sees}\ (\mathit{Ticket}, N)}}{\text{V } \mathbf{sees}\ \mathit{Ticket}} \quad \dfrac{\text{Mesg. \#1 and assumption}}{\text{V } \mathbf{believes}\ K_{\text{TTP}}}}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{said}\ (K_{\text{V, DE}}, \Delta t, \text{V})}}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{said}\ K_{\text{V, DE}}}\ \text{Proj}}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{believes}\ K_{\text{V, DE}}}\ \text{Nonce}$$

$$\dfrac{\dfrac{}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{controls}\ K_{\text{V, DE}}} \qquad \vdots}{\text{V } \mathbf{believes}\ K_{\text{V, DE}}}\ \text{Jur.}$$

As for the authorization part, V needs to believe in Δt. Here again we use the freshness of $K_{\text{V, DE}}$ in the Nonce Verification rule; thanks to the MIC in Mesg. #3, that of N could be used instead.

$$\text{Proj}_1\ \dfrac{\dfrac{\vdots}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{said}\ (K_{\text{V, DE}}, \Delta t, \text{V})}}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{said}\ (K_{\text{V, DE}}, \Delta t)}$$

$$\dfrac{\dfrac{}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{controls}\ \Delta t} \qquad \dfrac{\text{Nonce}\ \dfrac{\dfrac{\dfrac{}{\text{V } \mathbf{believes}\ \mathbf{fresh}\ (K_{\text{V, DE}})}}{\text{V } \mathbf{believes}\ \mathbf{fresh}\ (K_{\text{V, DE}}, \Delta t)}\ \text{Fresh}_1 \quad \vdots}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{believes}\ (K_{\text{V, DE}}, \Delta t)}}{\text{V } \mathbf{believes}\ \text{TTP } \mathbf{believes}\ \Delta t}\ \text{Proj}_2}{\text{V } \mathbf{believes}\ \Delta t}\ \text{Jur.}$$

Note that we only prove first order believes, hence mutual authentication is not established at this point. Second order beliefs such as V **believes** DE **believes** $K_{\text{V, DE}}$ cannot be proved given our assumptions, but this can easily be fixed by extending the protocol with a further handshake (encrypted with the shared key $K_{\text{V, DE}}$) between the two parties V and DE, in the fashion of the Needham-Schroeder Shared-Key protocol [11, 15]:

$$\begin{array}{lll} (5) & \mathbf{DE} \to \mathbf{V} & \{N'\}_{K_{\text{V, DE}}} \\ (6) & \mathbf{V} \to \mathbf{DE} & \{N' - 1\}_{K_{\text{V, DE}}} \\ \hline \end{array}$$

For instance using these two new messages, together with the freshness of the new nonce N', the second order belief V **believes** DE **believes** $K_{\text{V, DE}}$ stems from the Nonce Verification rule. We also make use of this technique in the ProVerif model below.

4 Model Using ProVerif

An implementation of the protocol has been written in ProVerif and an explanation of the implementation and all its details follow in this section. The complete listing of the ProVerif code is found in Listing A.1, Appendix A.

4.1 Assumptions and Prerequisites

A public communication channel, denoted `c`, is used for transmitting messages between the principals DE, V, and TTP. The channel is under control of the adversary, thus any message transmitted through the channel can be read by, or have been sent/modified by, the adversary. We define the channel by:

```
free c: channel .
```

We use the approach to model symmetric encryption, asymmetric encryption, and signing, as given in the ProVerif manual [13] without any further explanation. Furthermore, to handle certificates, a dedicated type is introduced and we define new functions to model the creation and verification of them.

```
type host .
type cert .
fun mkcert(host, pkey, spkey, sskey): cert.
reduc forall h: host, p: pkey, s: spkey, ssk: sskey;
    getcert(mkcert(h, p, s, ssk)) = (h, p, s).
reduc forall h: host, p: pkey, s: spkey, ssk: sskey;
    checkcert(mkcert(h, p, s, ssk), spk(ssk)) = (h, p, s).
```

Additional types to represent a nonce, the duration of an authorized session (Δt), a vehicle's ID (V_{ID}), and a security policy, are also introduced. Furthermore, a database (*i.e.*, table) with authorization information used by the TTP is defined. The table holds a vehicle's certificate and access policy for the corresponding vehicle ID. (Note that tables are not accessible to adversaries.)

```
type nonce .
type duration .
type vid .
type policy .
table vids(vid, cert, policy).
```

Finally, we assume that there exists at least one principal of DE, V, and TTP, that are trustworthy. These principals are represented by trusted hosts.

```
free DE, V, TTP: host.
```

All certificates are considered public, but the corresponding private keys must be kept secret. To ensure that none of the private keys are accessible by the adversary, we define the following secrecy assumptions:

```
not attacker(new skDE).
not attacker(new skV).
not attacker(new skTTP).
not attacker(new sskCA).
not attacker(new sskDE).
not attacker(new sskV).
not attacker(new sskTTP).
```

We define four events to record the state of important protocol execution points. The collected state information is used to evaluate the required security properties. The recorded events represents: the point at which V sends the ticket request (Mesg. #2), the point at which the session key is generated by the TTP, and finally, the point at which the key is accepted by V and DE, respectively.

```
23 event VsendRequest(cert, cert, cert, vid, nonce).
24 event createKey(key, vid, duration, policy, nonce).
25 event VacceptsKey(key, vid, duration, policy, nonce).
26 event DEacceptsKey(key, vid, duration).
```

After the protocol has finished, encrypted data can be exchanged between DE and V. To represent the data to transmit, we introduce a new private bitstring, denoted `s`. If this data, after encryption and transmission, can be obtained and recovered by the adversary, the protocol is broken and the encryption key compromised. To verify that only DE and V are in possession of the distributed session key, we introduce two tables in which DE and V records their encrypted communication and the corresponding keys used (as suggested by [13]).

```
27 free s: bitstring [private].
28 table DEmess(host, host, key, bitstring).
29 table Vmess(host, host, key, bitstring).
```

4.2 Protocol Implementation

The three principals, DE, V, and TTP, are modeled as separate processes. Each process exchanges messages through the public channel, `c`. A main process is used to initiate the three principals. The details of the implementation follow.

The main process initiates the public/private key-pairs, both for signing and encryption, and creates the certificates for the CA, DE, V, and TTP. All public keys and certificates are made available to the adversary on the communication channel `c`. Note that the vehicle ID, `vid`, is not considered secret and is also made available to the adversary (Line #48).[1] Finally, an unlimited number of parallel process of DE, V, and TTP, are initiated.

```
30 process
31   (* Create CA keys *)
32   new sskCA: sskey;
33   let spkCA = spk(sskCA) in out(c, spkCA);
34   (* Create DE keys *)
35   new sskDE: sskey;
36   let spkDE = spk(sskDE) in out(c, spkDE);
37   new skDE: skey;
38   let pkDE = pk(skDE) in out(c, pkDE);
39   let certDE = mkcert(DE, pkDE, spkDE, sskCA) in
40     out(c, certDE);
41   (* Create V keys *)
42   new sskV: sskey;
43   let spkV = spk(sskV) in out(c, spkV);
44   new skV: skey;
45   let pkV = pk(skV) in out(c, pkV);
```

[1]The vehicle ID is generally readable through the front window of a car.

```
46   let certV = mkcert(V, pkV, spkV, sskCA) in
47     out(c, certV);
48   new v: vid; out(c, v);
49   new p: policy; insert vids(v, certV, p);
50   (* Create TTP keys *)
51   new sskTTP: sskey;
52   let spkTTP = spk(sskTTP) in out(c, spkTTP);
53   new skTTP: skey;
54   let pkTTP = pk(skTTP) in out(c, pkTTP);
55   let certTTP = mkcert(TTP, pkTTP, spkTTP, sskCA) in
56     out(c, certTTP);
57   (* Start processes *)
58   ( (!processDE(skDE, spkCA, certDE, certTTP, spkTTP)) |
59     (!processV(skV, pkV, spkCA, certV, v)) |
60     (!processTTP(skTTP, sskTTP, spkCA, certTTP)) |
61     (processQ)
62   )
```

DE sends the certificates of DE and TTP to V (Mesg. #1). If the received ticket can be recovered and the signature is valid, this is recorded by the event (Line #72). Furthermore, to verify that the key is unique for this session alone (thus, providing mutual authentication), the private bitstring `s` is encrypted using the received key and the result is stored in the table `DEmess` (Line #75).

```
63 let processDE(skDE: skey, spkCA: spkey,
64     certDE: cert, certTTP: cert, spkTTP: spkey) =
65   (* Message 1 *)
66   out(c, (certDE, certTTP));
67   (* Message 4 *)
68   in(c, x: bitstring);
69   let TicketDE = adec(x, skDE) in
70   let (k: key, dt: duration, v: vid) =
71     checksign(TicketDE, spkTTP) in
72   event DEacceptsKey(k, v, dt);
73   (* To check that k is the same at DE and V *)
74   let m: bitstring = senc(s, k) in
75   insert DEmess(DE, V, k, m).
```

V waits for a new request from DE (Mesg. #1). If the certificates in the request are valid, a new authorization request to the TTP is prepared. A nonce is generated and we record the new TTP request as an event (Line #83) before Mesg. #2 is sent. The new authorization ticket is received from the TTP (Mesg. #3) and if the ticket signature is valid, the session key, authorization duration, and authorization policy can be recovered and recorded by the event (Line #90). Furthermore, to verify that the session key is unique, the private bitstring `s` is encrypted in the same way as done in `processDE` above. The encrypted bitstring is also made available to the adversary, so that the confidentiality property of the encrypted communication can be verified.

```
76 let processV(skV: skey, pkV: pkey, spkCA: spkey, certV: cert, v: vid) =
77   (* Message 1 *)
78   in(c, (certDEX: cert, certTTPX: cert));
```

```
79   let (hx: host, pkDE: pkey, spkDE: spkey) = checkcert(certDEX, spkCA) in
80   let (hy: host, pkTTP: pkey, spkTTP: spkey) = checkcert(certTTPX, spkCA) in
81   (* Message 2 *)
82   new N: nonce;
83   event VsendRequest(certDEX, certV, certTTPX, v, N);
84   out(c, (certDEX, aenc((v, N), pkTTP)));
85   (* Message 3 *)
86   in(c, (x: bitstring, TicketDE: bitstring));
87   let (TicketV: bitstring, =N) = adec(x, skV) in
88   let (k: key, dt: duration, p: policy) =
89     checksign(TicketV, spkTTP) in
90   event VacceptsKey(k, v, dt, p, N);
91   (* Message 4 *)
92   out(c, TicketDE);
93   (* To check that k is the same at DE and V *)
94   let m: bitstring = senc(s, k) in
95   insert Vmess(DE, V, k, m);
96   (* Send some encrypted data to adversary *)
97   out(c, m).
```

TTP waits for new authorization requests from V (Mesg. #2). If the request is from a "trusted" DE with a valid certificate, and a "trusted" V, the information regarding this session is looked up in TTP's policy database (Line #105). A new session key, `k`, is generated and the duration of the authorization is calculated (represented by a new `dt`). We record the generation of authorization information as a new event (Line #110). Finally, the tickets are assembled and sent to V.

```
98  let processTTP(skTTP: skey, sskTTP: sskey,
99      spkCA: spkey, certTTP: cert) =
100   (* Message 2 *)
101   in(c, (certDEX: cert, m: bitstring));
102   let (=DE, pkDEX: pkey, spkDEX: spkey) =
103     checkcert(certDEX, spkCA) in
104   let (v: vid, N: nonce) = adec(m, skTTP) in
105   get vids(=v, certV: cert, p: policy) in
106   let (=V, pkV: pkey, spkV: spkey) = getcert(certV) in
107   (* Message 3 *)
108   new k: key;
109   new dt: duration;
110   event createKey(k, v, dt, p, N);
111   let TicketV = sign((k, dt, p), sskTTP) in
112   let TicketDE = sign((k, dt, v), sskTTP) in
113   out(c, (aenc((TicketV, N), pkV), aenc(TicketDE, pkDEX))).
```

4.3 Queries of Security Properties

Several queries have been stated to test the desired security properties as shown by the results in Table 5.1.

First, mutual authentication (R_1), holds if DE and V both believes that the session key (issued by the TTP) only is available to itself and the other party. Thus, if V and DE accepts the key, the key should have been created by the TTP:

```
query k: key, v: vid, d: duration, p: policy, N: nonce;
     event(VacceptsKey(k, v, d, p, N))
  && event(DEacceptsKey(k, v, d))
  ==> event(createKey(k, v, d, p, N)).
```

However, this query cannot be written directly as above into ProVerif, but needs to be rewritten using tables and a helper process (as described in the manual [13]):

```
event termProto(host, host, key, bitstring, host, host, key, bitstring).
query h: host, k: key, k': key, m: bitstring;
    event(termProto(h, V, k, m, h, V, k', m)) ==> k = k'.
let processQ =
  ! get DEmess(hDE, hDE', kDE, mDE) in
    get Vmess(hV, hV', kV, mV) in
  event termProto(hDE, hDE', kDE, mDE, hV, hV', kV, mV).
```

Secondly, the secrecy of the session key, $K_{\mathrm{DE,V}}$, (R_2), holds if the adversary cannot recover the encrypted bitstring, `s`, made available in the end of `processDE`.

```
query attacker(s) .
```

Finally, freshness of Δt, (R_3), holds if: for each time a ticket is accepted by DE and V, it has also been recently created by the TTP. Thus, a one-to-one correspondence between number of accepted and created tickets is required:

```
query k: key, v: vid, d: duration, p: policy, N: nonce;
  inj-event(DEacceptsKey(k, v, d)) ==>
  inj-event(createKey(k, v, d, p, N)).
query k: key, v: vid, d: duration, p: policy, N: nonce;
  inj-event(VacceptsKey(k, v, d, p, N)) ==>
  inj-event(createKey(k, v, d, p, N)).
```

5 Results and Discussion

Using the BAN framework, we were only able to prove authentication up to first order beliefs. However we argued that the protocol can easily be extended if second order beliefs such as V **believes** DE **believes** $K_{\mathrm{V,DE}}$ are desired.

The result of our ProVerif analysis is presented in Table 5.1. As for the desired security properties, we find that:

- Mutual Authentication (R_1): From query 1 (in Table 5.1) we find that if the protocol terminates, the session key that DE and V hold is the same. Thus, mutual authentication is guaranteed. Furthermore, to ensure that there is an authenticated key exchange, as described in [13], query 5 is also evaluated.

- Secrecy of session key $K_{\mathrm{V,DE}}$ (R_2): We conclude from query 2 that the session key is secret and has not been leaked.

Table 5.1: Results of ProVerif Analysis

Nr.	Query	Result
1	`event(termProto(h_122,V[],k_123,m_124,h_122,V[],k',m_124)) ==> k_123 = k'`	true
2	`not attacker_bitstring(s[])`	true
3a	`inj-event(DEacceptsKey(k_11147,v_11148,d_11149)) ==> inj-event(createKey(k_11147,v_11148,d_11149,p_11150,N_11151))`	false
3b	`(but event(DEacceptsKey(k_13303,v_13304,d_13305)) ==> event(createKey(k_13303,v_13304,d_13305,p_13306,N_13307))`	(true)
4	`inj-event(VacceptsKey(k_19372,v_19373,d_19374,p_19375,N_19376)) ==> inj-event(createKey(k_19372,v_19373,d_19374,p_19375,N_19376))`	true
5	`event(termProto(x_122,y,k_123,m_124,x',y',k_123,m')) ==> (x_122 = x' && y = y')`	true
6	`inj-event(VacceptsKey(k_10741,v_10742,d_10743,p_10744,N_10745)) ==> (event(createKey(k_10741,v_10742,d_10743,p_10744,N_10745)) ==> event(VsendRequest(cde_10738,cv_10739,cttp_10740,v_10742,N_10745)))`	true

- Freshness of Δt (R_3): From query 4, we find that each ticket received by V has previously been generated by the TTP and that a one-to-one correspondence between requests and answers holds. The query can even be extended to include the request from V, as shown in 6; whenever V receives a ticket, it was preceded by a request from V itself. We conclude that as 4 (and 6) hold, the ticket is fresh, hence so is Δt. This, however, is not true for a ticket received by DE. As shown by query 3, freshness cannot be guaranteed (*i.e.*, a one-to-one correspondence is not found, as expressed by 3a), rather it can only be proved that the ticket was created by the TTP at some earlier point in time (as expressed by 3b). Thus, replay and delay of tickets to DE is possible. Although, this is not a surprise, as it was already pointed out in our informal analysis [10].

To conclude, we find all the desired security properties to hold for the authorization protocol. Even though replay and delay of $\text{Ticket}_{\text{DE}}$ is possible, such activity does not prevent V from enforcing its access policy in accordance to the acquired and fresh security policy (in Ticket_{V}). Thus, the vehicle is still kept protected against unauthorized access.

6 Related Work

Formal verification using the ProVerif tool has already been used to analyze security properties in different vehicular protocols. In [16], Nilsson et al. propose a re-keying protocol for hierarchical vehicular networks and proves its security properties using ProVerif. An attestation-based security architecture where only genuine ECUs in the in-vehicle network can communicate is proposed by Oguma et al. [17]. The protocol is further discussed by Lee et al. [18], where a formal analysis is performed. Moreover, Pedroza et al. [19] convert a SysML model of the diagnosis and firmware download protocol described in [6] into ProVerif and prove the security correctness. Nevertheless, to the authors' best knowledge, we are the first to formally verify an authorization protocol for remote vehicular diagnostics.

7 Conclusion

Many proposed security protocols have been found to be broken after some time. By using formal methods to reason about protocols' security properties, design flaws can be identified in early stages. In this paper, we have formally analyzed an authorization protocol for vehicle diagnostics access using two different analysis methods. Both methods come to the same result. We conclude that the protocol provides mutual authentication between the connecting diagnostics equipment and the vehicle, that the distributed session key cannot be obtained by an attacker, and that the authorization information used by vehicles to enforce access control is fresh. Thus, a correct implementation of the protocol prevents unauthorized access to vehicles.

Acknowledgments

We would like to express our gratitude to Tomas Olovsson and Andrei Sabelfeld for their kind input and comments to this paper.

References

[1] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, et al. "Experimental Security Analysis of a Modern Automobile". In: *2010 IEEE Symposium on Security and Privacy (SP).* 2010, pp. 447–462.

[2] 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 Security Symposium.* Aug. 2011, pp. 77–92.

[3] S. M. Mahmud, S. Shanker, and I. Hossain. "Secure Software Upload in an Intelligent Vehicle via Wireless Communication Links". In: *IEEE Intelligent Vehicles Symposium, 2005. Proceedings.* 2005, pp. 588–593.

[4] I. Hossain and S. M. Mahmud. "Secure Multicast Protocol for Remote Software Upload in Intelligent Vehicles". In: *Proc. of the 5th Ann. Intel. Vehicle Systems Symp. of National Defense Industries Association (NDIA).* June 2005, pp. 145–155.

[5] D. K. Nilsson and U. E. Larson. "Secure Firmware Updates over the Air in Intelligent Vehicles". In: *IEEE International Conference on Communications Workshops, 2008. ICC Workshops '08.* May 2008, pp. 380–384.

[6] M. S. Idrees, H. Schweppe, Y. Roudier, M. Wolf, D. Scheuermann, and O. Henniger. "Secure Automotive On-Board Protocols: A Case of Over-the-Air Firmware Updates". In: *Communication Technologies for Vehicles.* Lecture Notes in Computer Science 6596. Jan. 2011, pp. 224–238. URL: `http://link.springer.com/chapter/10.1007/978-3-642-19786-4_20` (visited on 09/19/2013).

[7] *ISO 13400-1:2011: Road vehicles — Diagnostic communication over Internet Protocol (DoIP) — Part 1: General information and use case definition.* ISO, 2011.

[8] *ISO 13400-2:2012: Road vehicles — Diagnostic communication over Internet Protocol (DoIP) — Part 2: Transport protocol and network layer services.* ISO, 2012.

[9] P. Kleberger, T. Olovsson, and E. Jonsson. "An In-Depth Analysis of the Security of the Connected Repair Shop". In: *Proceedings of the Seventh International Conference on Systems and Networks Communications (ICSNC 2012).* Nov. 2012, pp. 99–107. URL: `http://www.thinkmind.org/index.php?view=article&articleid=icsnc_2012_5_10_20167`.

[10] P. Kleberger and T. Olovsson. "Protecting Vehicles Against Unauthorised Diagnostics Sessions Using Trusted Third Parties". In: *Computer Safety, Reliability, and Security.* Lecture Notes in Computer Science 8153. Jan. 2013, pp. 70–81. URL: `http://link.springer.com/chapter/10.1007/978-3-642-40793-2_7` (visited on 08/30/2013).

[11] M. Burrows, M. Abadi, and R. M. Needham. “A Logic of Authentication”. In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 426.1871 (1989), pp. 233–271. URL: `http://rspa.royalsocietypublishing.org/content/426/1871/233`.

[12] B. Blanchet. “Automatic verification of correspondences for security protocols”. In: *Journal of Computer Security* 17.4 (Jan. 2009), pp. 363–434. URL: `http://dx.doi.org/10.3233/JCS-2009-0339` (visited on 10/02/2013).

[13] B. Blanchet, B. Smyth, and V. Cheval. *ProVerif 1.87: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.* 2013.

[14] P. Kleberger and G. Moulin. “Short Paper: Formal Verification of an Authorization Protocol for Remote Vehicle Diagnostics”. In: *IEEE Vehicular Network Conference (VNC)*. Dec. 2013.

[15] P. Syverson and I. Cervesato. “The Logic of Authentication Protocols”. In: *Foundations of Security Analysis and Design.* Lecture Notes in Computer Science 2171. Jan. 2001, pp. 63–137. URL: `http://link.springer.com/chapter/10.1007/3-540-45608-2_2` (visited on 11/25/2013).

[16] D. K. Nilsson, U. E. Larson, and E. Jonsson. “Low-Cost Key Management in Hierarchical Wireless Vehicle Networks”. In: *Proc. IEEE Intelligent Vehicles Symposium.* June 2008.

[17] H. Oguma, A. Yoshioka, M. Nishikawa, R. Shigetomi, A. Otsuka, and H. Imai. “New Attestation-Based Security Architecture for In-Vehicle Communication”. In: *IEEE Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008.* 2008, pp. 1–6.

[18] G. Lee, H. Oguma, A. Yoshioka, R. Shigetomi, A. Otsuka, and H. Imai. “Formally Verifiable Features in Embedded Vehicular Security Systems”. In: *2009 IEEE Vehicular Networking Conference (VNC).* Oct. 2009, pp. 1–7.

[19] G. Pedroza, M. Idrees, L. Apvrille, and Y. Roudier. “A Formal Methodology Applied to Secure Over-the-Air Automotive Applications”. In: *2011 IEEE Vehicular Technology Conference (VTC Fall).* 2011, pp. 1–5.

A ProVerif Implementation

Listing A.1: ProVerif Code

```
set ignoreTypes = false .

free c: channel .

(* Symmetric encryption *)
type key .
fun senc(bitstring, key): bitstring .
reduc forall m: bitstring, k: key;
        sdec(senc(m, k), k) = m .

(* Assymetric encryption *)
type pkey .
type skey .

fun pk(skey): pkey .
fun aenc(bitstring, pkey): bitstring .
reduc forall m: bitstring, sk: skey;
        adec(aenc(m, pk(sk)), sk) = m .

(* Signtures *)
type spkey .
type sskey .
```

```

fun spk(sskey): spkey .
fun sign(bitstring, sskey): bitstring .
reduc forall m: bitstring, ssk:sskey;
        getmess(sign(m, ssk)) = m .
reduc forall m: bitstring, ssk:sskey;
        checksign(sign(m, ssk), spk(ssk)) = m .

(* Certificates *)
type host .
type cert .
fun mkcert(host, pkey, spkey, sskey): cert.
reduc forall h: host, p: pkey, s: spkey, ssk: sskey;
    getcert(mkcert(h, p, s, ssk)) = (h, p, s).
reduc forall h: host, p: pkey, s: spkey, ssk: sskey;
    checkcert(mkcert(h, p, s, ssk), spk(ssk)) = (h, p, s).

(* Additional Types *)
type nonce .
type duration .
type vid .
type policy .
table vids(vid, cert, policy).

(* Trusted Hosts *)
free DE, V, TTP: host.

(* Secrecy Assumptions *)
not attacker(new skDE).
not attacker(new skV).
not attacker(new skTTP).
not attacker(new sskCA).
not attacker(new sskDE).
not attacker(new sskV).
not attacker(new sskTTP).


(* Secrecy and Authentication *)
free s: bitstring [private].
table DEmess(host, host, key, bitstring).
table Vmess(host, host, key, bitstring).

(* Secrecy Queries *)
query attacker(s) .

(* Key Events *)
event VsendRequest(cert, cert, cert, vid, nonce).
event createKey(key, vid, duration, policy, nonce).
event VacceptsKey(key, vid, duration, policy, nonce).
event DEacceptsKey(key, vid, duration).

(* Key Queries *)
query k: key, v: vid, d: duration, p: policy, N: nonce;
  inj-event(VacceptsKey(k, v, d, p, N)) ==>
  inj-event(createKey(k, v, d, p, N)).
query k: key, v: vid, d: duration, p: policy, N: nonce;
  inj-event(DEacceptsKey(k, v, d)) ==>
  inj-event(createKey(k, v, d, p, N)).

query cde: cert, cv: cert, cttp: cert,
    k: key, v: vid, d: duration, p: policy, N: nonce;
  inj-event(VacceptsKey(k, v, d, p, N)) ==> (
    event(createKey(k, v, d, p, N)) ==>
    event(VsendRequest(cde, cv, cttp, v, N))
  ).

query cde: cert, cv: cert, cttp: cert,
    k: key, v: vid, d: duration, p: policy, N: nonce;
  event(DEacceptsKey(k, v, d)) ==> (
    event(createKey(k, v, d, p, N)) ==>
    event(VsendRequest(cde, cv, cttp, v, N))
  ).
query cde: cert, cv: cert, cttp: cert,
    k: key, v: vid, d: duration, p: policy, N: nonce;
  event(DEacceptsKey(k, v, d)) ==> (
    event(VsendRequest(cde, cv, cttp, v, N)) &&
    event(createKey(k, v, d, p, N))
  ).


(* Query SameKey *)
event termProto(host, host, key, bitstring, host, host, key, bitstring).
query h: host, k: key, k': key, m: bitstring;
    event(termProto(h, V, k, m, h, V, k', m)) ==> k = k'.
let processQ =
  ! get DEmess(hDE, hDE', kDE, mDE) in
    get Vmess(hV, hV', kV, mV) in
  event termProto(hDE, hDE', kDE, mDE, hV, hV', kV, mV).

(* End Queries *)

(* *)
query x: host, y: host, k: key, m: bitstring, x': host, y': host, m': bitstring;
    event(termProto(x, y, k, m, x', y', k, m')) ==>
    x = x' && y = y'.


(* Message Ordering *)
```

```
event beginDE(cert, cert) . (* certDE, certTTP *)
event termDE(cert, cert, key, vid, duration) . (* certDE, certTTP *)
event termV(cert, cert, cert, key, vid, duration, policy, nonce) . (* certDE,
    certV, certTTP *)
event TTPacceptsRequest(cert, cert, cert, vid, nonce) . (* certDE, certV, certTTP
     *)
event termTTP(cert, cert, cert, key, vid, duration, policy, nonce) . (* certDE,
    certV, certTTP *)


(*
query cde: cert, cv: cert, cttp: cert, k: key, v: vid, d: duration, p: policy, N:
     nonce;
  inj-event(DEacceptsKey(k, v, d)) ==> event(termTTP(cde, cv, cttp,k, v, d, p, N)
    ).
*)

(* Processes for DE, V, and TTP *)

(* Process DE *)
let processDE(skDE: skey, spkCA: spkey,
    certDE: cert, certTTP: cert, spkTTP: spkey) =
  (* Message 1 *)
  out(c, (certDE, certTTP));
  (* Message 4 *)
  in(c, x: bitstring);
  let TicketDE = adec(x, skDE) in
  let (k: key, dt: duration, v: vid) =
    checksign(TicketDE, spkTTP) in
  event DEacceptsKey(k, v, dt);
  (* To check that k is the same at DE and V *)
  let m: bitstring = senc(s, k) in
  insert DEmess(DE, V, k, m).

(* Process V *)
let processV(skV: skey, pkV: pkey, spkCA: spkey, certV: cert, v: vid) =
  (* Message 1 *)
  in(c, (certDEX: cert, certTTPX: cert));
  let (hx: host, pkDE: pkey, spkDE: spkey) = checkcert(certDEX, spkCA) in
  let (hy: host, pkTTP: pkey, spkTTP: spkey) = checkcert(certTTPX, spkCA) in
  (* Message 2 *)
  new N: nonce;
  event VsendRequest(certDEX, certV, certTTPX, v, N);
  out(c, (certDEX, aenc((v, N), pkTTP)));
  (* Message 3 *)
  in(c, (x: bitstring, TicketDE: bitstring));
  let (TicketV: bitstring, =N) = adec(x, skV) in
  let (k: key, dt: duration, p: policy) =
    checksign(TicketV, spkTTP) in
  event VacceptsKey(k, v, dt, p, N);
  (* Message 4 *)
  out(c, TicketDE);
  (* To check that k is the same at DE and V *)
  let m: bitstring = senc(s, k) in
  insert Vmess(DE, V, k, m);
  (* Send some encrypted data to adversary *)
  out(c, m).

(* Process TTP *)
let processTTP(skTTP: skey, sskTTP: sskey,
    spkCA: spkey, certTTP: cert) =
  (* Message 2 *)
  in(c, (certDEX: cert, m: bitstring));
  let (=DE, pkDEX: pkey, spkDEX: spkey) =
    checkcert(certDEX, spkCA) in
  let (v: vid, N: nonce) = adec(m, skTTP) in
  get vids(=v, certV: cert, p: policy) in
  let (=V, pkV: pkey, spkV: spkey) = getcert(certV) in
  (* Message 3 *)
  new k: key;
  new dt: duration;
  event createKey(k, v, dt, p, N);
  let TicketV = sign((k, dt, p), sskTTP) in
  let TicketDE = sign((k, dt, v), sskTTP) in
  out(c, (aenc((TicketV, N), pkV), aenc(TicketDE, pkDEX))).

(* Main *)
process
  (* Create CA keys *)
  new sskCA: sskey;
  let spkCA = spk(sskCA) in out(c, spkCA);
  (* Create DE keys *)
  new sskDE: sskey;
  let spkDE = spk(sskDE) in out(c, spkDE);
  new skDE: skey;
  let pkDE = pk(skDE) in out(c, pkDE);
  let certDE = mkcert(DE, pkDE, spkDE, sskCA) in
    out(c, certDE);
  (* Create V keys *)
  new sskV: sskey;
  let spkV = spk(sskV) in out(c, spkV);
  new skV: skey;
  let pkV = pk(skV) in out(c, pkV);
  let certV = mkcert(V, pkV, spkV, sskCA) in
    out(c, certV);
  new v: vid; out(c, v);
  new p: policy; insert vids(v, certV, p);
  (* Create TTP keys *)
  new sskTTP: sskey;
  let spkTTP = spk(sskTTP) in out(c, spkTTP);
  new skTTP: skey;
  let pkTTP = pk(skTTP) in out(c, pkTTP);
  let certTTP = mkcert(TTP, pkTTP, spkTTP, sskCA) in
    out(c, certTTP);
  (* Start processes *)
  ( (!processDE(skDE, spkCA, certDE, certTTP, spkTTP)) |
    (!processV(skV, pkV, spkCA, certV, v)) |
    (!processTTP(skTTP, sskTTP, spkCA, certTTP)) |
    (processQ)
  )

(* End *)
```