On Authentication in a Connected Vehicle: Secure Integration of Mobile Devices with Vehicular Networks

Kyusuk Han, Swapna Divya Potluri, and Kang G. Shin

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109-2121
{kyusuk,swapnap,kgshin}@umich.edu

ABSTRACT

Recent advances in in-vehicle technologies have paved way to a new era of connectivity. Vehicle manufacturers have already deployed various technologies for driving assistance, anti-theft, and infotainment. They are now developing ways to interface mobile devices with vehicles and provide the customer's smartphone or tablet the ability to send/receive information to/from the car. However, such an integration introduces severe security risks to the vehicle. The in-vehicle network was originally designed to operate in a closed environment and thus, security was not of concern. It has now become an important issue due to an increasing number of external interfaces to the in-vehicle network. Several studies have already shown that an in-vehicle network can be easily compromised just by connecting cheap commercial devices and doing reverse engineering. Although research efforts have been made to secure in-vehicle networks, most of them focused on defining security requirements, or presenting attack scenarios without providing any feasible solution. Also, to the best of our knowledge, there hasn't been any study with a specific focus on understanding and analyzing the security aspects of integrating mobile devices with cars. In this paper, we define the integration model, present the attack scenarios, define the security objectives, and then propose a 3-step verification mechanism that meets our objectives.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Real-time and embedded systems; J.7 [**Computers in Other Systems**]: Command and control; D.4.6 [**Security and Protection**]: Authentication

Keywords

Vehicular networks; security; mobile device integration; authentication; Controller Area Network (CAN)

ICCPS'13 April 8-11, 2013, Philadelphia, PA, USA.

1. INTRODUCTION

Cars are undoubtedly the most sophisticated devices we use in our daily lives. Currently, most cars are equipped with an average of 70 ECUs (Electronic Control Units) that provide advanced functionalities inside the vehicle. These ECUs are internally connected via serial buses and communicate using a standard protocol called the *Controller Area Network* (CAN). CAN is not only used by ECUs, but also by devices that have a user interface, such as the dashboard. Other user interfaces include infotainment and navigation systems, in-vehicle WiFi, voice recognition modules, etc. Recent innovations in this area include GM's Onstar, Ford's Sync, and so on.

Most of these systems are manufacturer-specific. They are often built into the car and hence not easily customizable and/or upgradable. A futuristic model would be to have manufacturer-independent interfaces that would let users customize according to their preferences and requirements (i.e., enabling user-driven, not manufacturer-driven, experience inside a vehicle). This would allow mobile devices, like smartphones and tablets, to replace in-vehicle entertainment systems, dashboards or diagnostic systems. One such example would be Ford's OpenXC platform [7] that enables a one-way communication from the car to the mobile device. It provides a uniform open interface to which the users can simply attach their mobile devices. The users can then use applications that can process the data from the in-vehicle network.

However, such an integration is governed by two major restrictions. First, the car manufacturers usually want to keep their in-vehicle communications hidden from the customers to preserve their product secrets. Second, integration of external devices with the in-vehicle network will have acute security/safety implications. For example, the authors of [3] analyzed the vulnerabilities of vehicular communications, and Vandenbrink [14] showed that the in-vehicle network could be sniffed using a cheap commercial embedded device.

To address these concerns, Ford introduced a conversion / translation module that acts as a gateway and provides a layer of abstraction for the internal network. However, this setup does not guarantee a secure integration for all the communication scenarios. Although several researchers have already proposed security mechanisms for integrating external

devices with in-vehicle networks, they are not applicable to this interface model as they only consider direct connection between an external device and the in-vehicle network, and do not account for the first restriction mentioned above.

Considering the above-mentioned restrictions, we adopt the gateway model implemented by Ford [7] and propose a communication and security architecture to securely integrate external devices with the in-vehicle network. Our architecture is composed of three main components: A user device (UD), a gateway (GW) and ECUs on an in-vehicle network. We perform a detailed analysis of all relevant attackers to create a corresponding attacker model. We then establish our security objectives and design a three-step authentication protocol that allows only authorized devices to access the in-vehicle network without exposing confidential vehicle information. Finally, we show that our design meets these objectives. Our contribution in this paper is three-fold:

- Mutual authentication between the UD and the GW;
- Authentication of the GW by the in-vehicle network;
- Verification of the UD's requests.

The remainder of this paper is organized as follows. Section 2 briefly reviews CAN, a representative in-vehicle network, and the security issues thereof. In Section 3 we discuss the security model of a connected vehicle and enumerate all the attack scenarios and assumptions based on the model. Our three-step authentication protocol is presented in Section 4 and evaluated via a detailed security analysis in Section 5. Finally, we conclude the paper in Section 6.

2. REVIEW OF IN-VEHICLE NETWORKS

For completeness, we briefly review the various components of an in-vehicle network. As touched upon in Section 1, the in-vehicle network connects, and enables communication between, several ECUs. CAN is the most widely-used protocol for this communication (others being FlexRay and LIN) and will hence be our primary focus.

2.1 Controller Area Network

CAN is a multi-master broadcast serial bus that connects the ECUs inside a vehicle. There are four types of CAN frames: *data* frame that contains the data to be transmitted; *remote* frame that requests transmissions from a specific node/identifier; *error* frame that is used when an error is detected in the received frame; and *overload* frame to inject a delay between frames. A CAN data frame can contain a data field of up to 8 bytes as shown in Fig. 1. The base frame format allows an 11-bit identifier (ID), while the extended frame format allows a 29-bit ID. Since it is a multi-master protocol, the order/priority of transmission is determined through bus contention, called *arbitration*: a process of broadcasting one bit at a time and comparing it with the bits broadcast by other ECUs. The frame with the smallest ID wins the arbitration and gets transmitted first. A 16-bit CRC field (with a 1-bit CRC delimiter) is provided to check the integrity of each received frame. For more information of CAN frame formats, see ISO 11898-1:2003 or other related CAN standards.

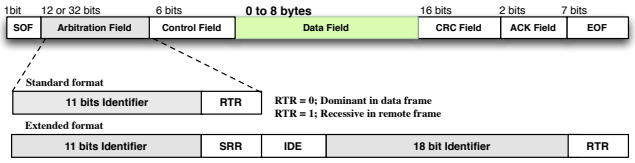

Figure 1: CAN frame format

The frames are identified using their frame IDs. After winning the bus contention via arbitration, the ECU writes the frame, one bit at a time, onto the bus. The CAN frame does not contain the address of the transmitter or the receiver node; a frame is broadcast over the bus, and each node checks the unique ID in the frame, and determines whether to accept or ignore it. For completeness, a brief example of CAN communication architecture is shown in Fig. 2. ECU 2 broadcasts frame 1, all ECUs connected to the CAN bus receive it. ECUs 1 and 4 accept it, while ECU 3 discards it.

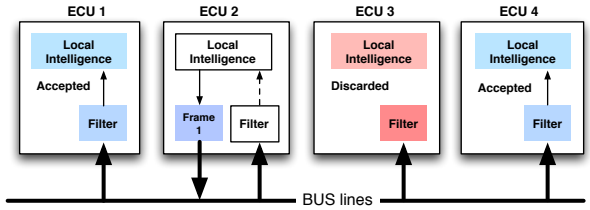

Figure 2: Example of frame filtering: when ECU 2 sends Frame 1, ECUs 1 and 4 receive it, while ECU 3 discards it.

2.2 Security Issues of CAN

Several researchers have already reported the weak security support of CAN. The authors of [3] demonstrated vulnerabilities in the current automotive networks by presenting various attack scenarios. Recent reports, such as [11], have revealed the weakness of the CAN when it is open to the external world. The authors of [9] argue that CAN is insecure and vulnerable to DoS (Denial of Service) attacks.

All of the above-reported issues can be attributed to the following major drawbacks of the CAN architecture:

- A CAN frame has no authentication field.
- The payload field in a CAN frame allows only up to 8 bytes, making it difficult to deploy strong security primitives.
- ECUs do not have the computational power to support cryptographic functions.
- No authentication of the sender and the receiver of a frame.

It is, however, important to note that CAN was designed to be simple and to work in isolation. It is difficult to overhaul the entire design to support security mechanisms since the automotive manufacturers are reluctant to adopt such a

wholesale change due to the associated cost. Note that other protocols, such as FlexRay, have been introduced without addressing all the security issues.

2.2.1 Encryption

As long as the communication in an in-vehicle network remains closed, frames need not be encrypted as stated in [13]. However, opening up the in-vehicle network to the outside world requires protection of vehicle privacy. The authors of [13, 15] state that encryption of all vehicular data transmissions will improve the security of automotive bus communications, however this will increase the overall overhead significantly.

As mentioned earlier, CAN's small data field makes it difficult to implement encryption. Hence, while block and stream cipher algorithms are commonly used for data encryption in regular networks, such mechanisms are not practical in vehicular networks. An interesting effort reported in [2] is to deploy RC4, a lightweight stream cipher encryption, for CAN data frames. No two messages must be encrypted with the same key under RC4. To meet this requirement, they recommended the use of public key cryptosystem (PKC) for refreshing the keys. Although several researchers [1, 6] proposed the use of PKC and several key management schemes, they haven't yet been practically implemented.

2.2.2 Authentication

Cryptographic message authentication achieves strong protection against forgery, but incurs high overhead, thus degrading the overall throughput. The small number of bits in each frame makes it difficult to provide frame-level authentication. Most existing studies focus on the reduction of authentication delay while providing strong security. The authors of [13] proposed a truncated MAC model, and showed that 32 bits are good enough for cryptographic strength, although public recommendations, such as [4], state the need for at least 64 bits. The authors of [8] proposed a broadcast authentication methodology using MD5, SHA-1 and SHA-256, which needs about 1ms to hash a 14-byte string with SHA1 at a 40MHz clock speed. The authors of [10] proposed an authentication protocol for CAN by substituting the CRC field in CBC-MAC with the KASUMI algorithm [5]. Since the KASUMI algorithm supports 64-bit data blocks, it requires four CAN frame transmissions which would increase the bus load.

3. SECURITY MODEL OF A CONNECTED VEHICLE

3.1 Overview of a Connected Vehicle

A vehicular network is considered "connected" if it has a wired or a wireless interface/communication with a device that is not a part of the vehicle. This connectivity can be broadly classified as follows.

Connection with a trusted entity. Given below are some application scenarios of this type of connectivity:

(i) Anti-theft: Starting 2010, GM introduced a stolen vehicle slowdown service using its flagship device, OnStar. This feature allows OnStar (a trusted entity), to remotely (wireless) slow down a stolen vehicle.

(ii) Remote firmware update: Tesla Motors recently deployed an over-the-air firmware update feature: a module in the vehicle can receive an update, without visiting service centers (trusted entities).

(iii) Remote diagnostics: Technicians (trusted entities) can connect remotely with the diagnostics port on the vehicle using WiFi communication.

(iv) User devices: Using smartphones to lock/unlock vehicles. The request to the car is, however, redirected through a secure server (trusted entity).

Connection with an untrusted entity. The connection between a user device (untrusted entity) and the in-vehicle network (for one/two-way communications) is made through a gateway. User devices, such as smartphones and tablets, are directly connected to the OBD port through a gateway, and not through a secure server.

In this paper, we primarily focus on the connection between an untrusted user devices and a vehicle. This kind of integration is governed by two major restrictions. First, the manufacturers usually want to keep their in-vehicle communications proprietary. Second, integration of an external device into the in-vehicle network as in Fig. 3(a) will introduce additional security concerns. To address these concerns, Ford, in their OpenXC model, introduced a conversion/translation module that acts as a gateway and provides a layer of abstraction for the internal network as in Fig. 3(b). We adopt Ford's model and enhance it with our security protocol.

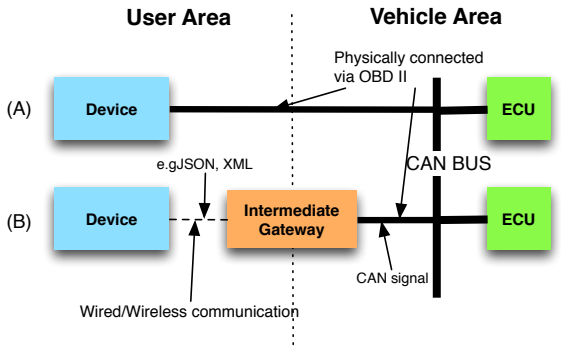

Figure 3: (a) User device directly receives CAN signals. (b) The gateway relays the vehicle information to the mobile device by converting the CAN signals to a device friendly format

3.2 System Model

By employing Ford's model as in Fig. 3(b), we define three entities:

User device (UD), which interacts with the vehicle. It is capable of heavy computations and has a standard communication network architecture. Examples are smartphones and tablets.

Gateway (GW), which interacts with both the UD and CAN. Interaction with the UD is done over a generic network, while interaction with the ECUs is done over a vehicle-specific network, i.e., CAN.

ECU, which interacts with the gateway through CAN. All of its communications are limited by the capability of CAN. We will thus use the terms ECU and CAN bus interchangeably.

Fig. 4 shows an example system setup using Ford's openXC platform. The complete setup includes the UD and the vehicle. The GW can be built with off-the-shelf components.

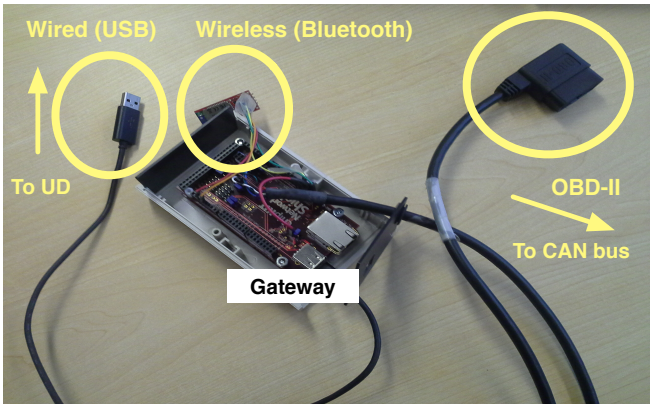

Figure 4: Example gateway: Ford's OpenXC platform

3.3 Assumptions

Below are the assumptions we make to substantiate the model under consideration.

- Link between the GW and the ECUs is secure.
- The ECUs are in a secure space. Some ECUs can be physically damaged, but they don't affect the others on the same bus.
- Each ECU has its own unique seed key provided by an external trusted party. Compromising the key doesn't affect the other ECUs on the same bus.
- Once the GW is attached and remains attached to the vehicle, we consider it physically secure.

3.4 Attack Scenarios

We focus on the below attack scenarios:

Case 1. Compromised UD: Attackers may attach their own devices or compromise a valid UD.

Case 2. Compromised GW: Attackers may attach their own device. We do not consider modifying a genuine GW using remote access or physical attachment.

Case 3. Both UD and GW are compromised: This scenario is a subset of Case 2.

We do not consider the case in which the ECUs are compromised. Such cases, including physically damaging the ECUs, are very rare and beyond the scope of this paper. The attack scenarios can be further classified as follows.

A1: Key exposure due to compromised entities: After compromising an entity (UD or GW), the attacker may try to extract the secret information from it.

A2: Invalid key: An attacker may attach devices to communicate with the ECUs in the vehicle. They may also try to use invalid keys.

A3: Impersonating a UD: An attacker's device may try to impersonate a UD.

A4: Impersonating a GW: An attacker's device may try to impersonate a GW.

A5: Fraudulent requests from a compromised UD: An attacker may compromise a UD and then send invalid requests to the ECUs.

A6: Fraudulent requests from a malicious GW: An attacker may try to compromise the GW through wired / wireless communications. He/ she may then send malicious commands or codes to the GW to read unauthorized vehicle information or to write control commands to the CAN. However, we do not consider the case in which the attacker transmits jamming signals to incapacitate the GW.

Table 1 shows all of these possible attack scenarios.

Table 1: Attack scenarios

	UD	GW	A1	A2	A3	A4	A5	A6
Case 1	Compromised	Not Compromised	Y	Y	Y	N	Y	N
Case 2	Compromised	Compromised	Y	Y	Y	Y	Y	Y
Case 3	Not Compromised	Compromised	Y	Y	Y	Y	N	Y

3.5 Security Requirements

We define the security requirements for each entity as follows.

UD (User Device):

- Only a valid UD should be able to send/receive messages to/from the ECU through the GW.

- A UD should be able to check the validity of the GW.

GW (Gateway):

- Only a valid GW should be able to send/receive messages to/from the ECU and the UD.

- The GW should be able to check the validity of the UD.

- The GW should be able to check the validity of messages from the ECUs and the UD.

- Without a request from the UD, the GW should not send or receive any vehicle information from an ECU.

ECU:

- An ECU should be able to check the validity of the GW.

- An ECU should sends/receives information to/from only a valid UD through a valid GW.

- Only a valid UD should be able to communicate with the ECUs.

- The frame format in the CAN bus should not be revealed to the entities which are not physically and directly attached.

Common:

- Exposing the secret information in one entity should not affect other entities.

4. THE PROPOSED PROTOCOL

In this section, we propose a three-step authentication protocol that provides secure communication between the UD and the ECU, preventing any malicious device from extracting confidential information from CAN.

4.1 Overview of the Protocol

The proposed protocol consists of the following three phases:

P1: Authenticating the GW;

P2: Mutual authentication between the UD and the GW;

P3: Authenticating UD's data request.

Our model consists of two different kinds of communication paths:

N1: Communication over the in-vehicle network, e.g., CAN, Flexray, etc.

N2: Communication over a wired or wireless network, e.g., USB, WiFi, Bluetooth, etc.

We assume that the communication between the GW and the ECUs is over **N1** and the communication between the UD and the GW is over **N2**. Thus, **P1** runs only on **N1**, **P2** runs only on **N2**, and **P3** on both **N1** and **N2**.

Table 2 shows the notations used in the protocol.

4.2 Initial Key Distribution

The initial key distribution is different for the three entities in our model. We use longevity and operating environment of each entity as the deciding factors to determine the appropriate key. The ECUs have the greatest longevity as they are not usually replaced after deployment in a car. Hence, we can safely assume that the seed key would be a long-term key and that it is built into the ECU at the time of car assembly.

Table 2: Notations

Notation	Description
LK	Long term seed key, shared between ECU and Trusted third party (e.g. vehicle manufacturers).
MK	Midterm key
SK	Short term key
TS	Timestamp. $TS = Year\|\|Month\|\|Date\|\|Time\|\|Expiration\_date$
RN_i	Random number i, randomly selected by each entity
$h(m)$	Hash of an arbitrary message m
f_j	$j = 1$: generate $Cert_{ID}$; $j = 2, 3$: derive keys; $j = 4$: Auth from the UD; $j = 5$: Auth from the GW. The same algorithms can also be used for all functions.

The GW, on the other hand, has a shorter lifetime than the ECUs as they are usually deployed at the time of car assembly or installed at a later time. Thus, the GW is associated with a mid-term key. The key is built into the GW and activated at the time of its purchase.

UDs have the shortest lifetime in our model because users often replace them with new models as and when they become available. For example, mobile devices like smartphones are usually replaced often and every user has his/her own device. We therefore associate UDs with a short-term key. The key is distributed to the users at the time of registration with the car manufacturer.

Table 3 compares the lifetimes of these entities and their associated keys.

Table 3: Comparison of different entities

Device Type	Lifetime	Description	Keys	Source
User Device (UD)	Short term (Months - 2 year)	Users frequently upgrade or replace. Used in external environments and connected to the gateway	SK_1, SK_2	On registration
Gateway (GW)	Mid term (Years)	Users rarely replace. Connected to the user device and the CAN bus only	MK_1, MK_2	During purchase
ECU	Long term (As same as a car's lifetime	Only replaced when broken, connected to the CAN bus only	LK	Built-in

Next, we need to establish a mechanism for each ECU to be able to verify the GW and the UD. One way to accomplish this is to use certificates. The certificate $Cert$ can be generated by using the seed key LK, timestamp TS, and

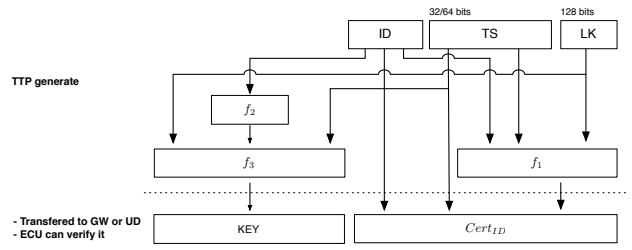

Figure 5: Key and certificate generation for P1 and P3. For the certificate in P2, LK is substituted by MK_2

the device ID, where $Cert_{ID} = ID||TS||f_1(ID||TS||LK)$. f_1, f_2 and f_3 are one-way cryptographic hash functions and used only for generating SKs or MKs and $Cert$.

The UD receives two keys and two certificates for **N1** and **N2** as in Fig. 5, and the GW receives two keys for **N1** and **N2**, with one certificate for **N1**.

4.3 Three-Step Authentication Protocol

4.3.1 P1: Authentication of the gateway by the ECUs

Suppose GW has MK_1 as the key for **N1**.

When the GW with valid $Cert_{GW}$ is connected to the ECU, it generates a request message REQ and $Auth_0^{P1}$ to initiate **P1**, where $Auth_0^{P1} = f_4(MK_1||MSG)$ and $MSG = REQ||Cert_{GW}$. Then, the GW sends MSG and $Auth_0^{P1}$ to the ECU.

After receiving requests from the GW, the ECU first checks if $Cert_{GW}$ is valid. If it is, then the ECU generates MK^* and $Auth_*^{P1}$, where $MK^* = f_3(LK||f_2(ID_{GW})||TS)$ and $Auth_*^{P1} = f_4(MK^*||MSG)$. If $Auth_*^{P1} \equiv Auth_0^{P1}$, the ECU stores MK^* as the key that is shared with the GW. This remains true as long as the GW stays connected and TS is not expired. Algorithm 1 shows the pseudocode for **P1**.

```
Algorithm 1: Authenticated Key Establishment
-------------------------------------------------
GW:
    MSG = REQ||CERT_GW
    AUTH_0_P1 = f4(MK_1||MSG)
    Send MSG||AUTH_0 to ECU
ECU:
    If( CERT is valid?)
        Generate MK* = f3(LK||f2(ID_GW)||TS)
        AUTH* = h(MK*||MSG)
        If (AUTH* == AUTH_0)
            If Previous key exists
                Delete old key
            Store MK* , TS
        else
            Discard MSG
    else
        Discard MSG

-------------------------------------------------
```

4.3.2 P2: Mutual authentication between the UD and the GW

When the UD is connected to the GW via **N2** for the first time, they need to be mutually authenticated. As discussed in Section 3.5, the UD should be connected only to a valid GW and the GW should only allow an authorized UD to connect to it. The valid GW has two keys: MK_1 and MK_2. MK_1 is used in **N1**, and MK_2 in **N2** (see Table 3). Similarly, a UD has two keys: SK_1 and SK_2, where SK_1 is used for **P3** and SK_2 for **P2**. Since the processes of **P2** are done over **N2**, two entities are assumed to use function h.

The UD sends MSG and $Auth_0^{P2}$, where $MSG = REQ||Cert_{UD}^{P2}$ and $Auth_0^{P2} = h(SK_2||MSG)$. $Cert_{UD}^{P2}$ is generated by trusted third party, where $Cert_{UD}^{P2} = ID_{UD}||TS||h(ID_{UD}||TS||MK_2)$.

The GW checks if $Cert_{UD}^{P2}$ from the UD is valid, and then verifies whether $Auth_0^{P2}$ is valid by generating SK_*^{P2} and $Auth_0^*$, where $SK_*^{P2} = h(TS||f_2(ID_{UD})||MK_2)$ and $Auth_0^* = h(SK_*^{P2}||MSG)$. If $Auth_0^* \equiv Auth_0^{P2}$, the GW selects RN_0 and generates $Auth_1^{P2} = h(SK_*^{P2}||RES||RN_0)$ where RES is the response message of an arbitrary size. The GW then responds by sending RES, RN_0 and $Auth_1^{P2}$ to the UD.

The UD verifies $Auth_1^{P2}$ by comparing $Auth_1^*$ generated by the UD with $Auth_1^{P2}$, where $Auth_1^* = h(SK_2||RES||RN_0)$. If $Auth_1^{P2} = Auth_1^*$, the UD regards the GW authenticated, and sends CON, RN_1, $Auth_2^{P2}$, where CON is a confirmation message, RN_1 is a randomly selected string by the UD, and $Auth_2^{P2} = h(SK_2||CON||RN_1||RN_0)$.

The GW generates $Auth_2^* = h(SK_*^{P2}||CON||RN_1||RN_0)$ and compares $Auth_2^*$ and $Auth_2^{P2}$. If they are identical, the GW sets SK_*^{P2} as the shared key with the UD, which remains valid until TS expires. Algorithm 2 provides the pseudocode for **P2**.

```
Algorithm 2: Secure Channel Setup between User Device
and CAN Translator
-------------------------------------------------
UD:
    MSG = REQ||CERT_UD_P2
    AUTH_0 = h(SK_2||MSG)
    Send MSG||AUTH_0 to GW

GW:
    If( CERT_UD_P2 is valid?)
        Generate SK* = h(MK_2||h(ID_UD)||TS)
        AUTH_0* = h(SK*||MSG)
        If (AUTH_0* == AUTH_0)
             Selects RN_0
             // Response from GW to UD
             Generates AUTH_1 = h(SK*||RES||RN_0)
             Generates MSG_2 = RES||RN_0||AUTH_1
             Send MSG_2 to UD
        else
             Discard request
    else
        Discard request

UD:
    Generates AUTH_1* = h(SK_2||RES||RN_0)
    If (AUTH_1* == AUTH_1)
        GW is authenticated
        Selects RN_1
        Generates AUTH_2 = h(SK_2||CON||RN_1||RN_0)
        // Confirmation from UD to GW
```

```
        Generates MSG_3 = CON||RN_1||AUTH_2
        Sends MSG_3 to GW
    else
        Stop

GW:
    Generates AUTH_2* = h(SK*||CON||RN_1||RN_0)
    If (AUTH_2* == AUTH_2)
        If Previous key exists
             Delete old key
        Store SK*, TS
    else
        Discard request

-------------------------------------------------
```

4.3.3 P3: Authentication of data request in ECU

When the UD wants to interact with an ECU via the GW, the UD first generates $HR_1 = h(SK_2||RN_1)$ and sends RN_1 and HR_1 to the GW. We assume that this transmission is done over a secure channel set up via **P2**.

The GW checks if HR_1 is valid, and then responds by sending RN_2 and HR_2, where RN_2 is a randomly selected string and $HR_2 = h(SK_*^{P2}||\ RN_2||RN_1)$.

After checking the validity of HR_2, the UD sends $Auth_1^{P3}$ to the GW, where $Auth_1^{P3} = f_4(SK_1||RN_1||\ RN_2)$ and f_4 is also a cryptographic hash function. If it is the first time, the UD also sends the certificate to the GW. The entire process is done over **N2**.

The GW generates $Auth_2^{P3}$ and K_{TEM}, where $Auth_2^{P3} = f_5(MK_1||\ Auth_1^{P3}||RN_2))$, $K_{TEM} = h(MK_1||RN_2||RN_1)$, and f_5 is also a cryptographic hash function. The same algorithms can be used for f_4 and f_5. The GW then sends RN_1, RN_2 and $Auth_2^{P3}$ to the ECUs over **N1**. If it is the first time, the GW also sends the certificate of the UD to the ECUs.

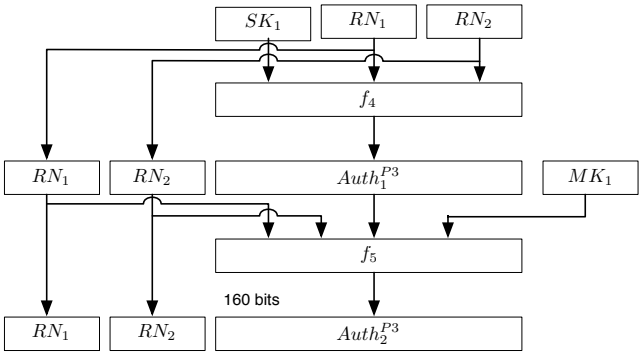

Figure 6: The ECU verifies the request from a user device

If the ECUs had previous interactions with the UD, they store SK_1. Otherwise, ECUs verify $Cert_{UD}^{P3}$ and derive SK_1. The ECUs generate $Auth_1^*$ and $Auth_2^*$ as shown in Fig. 6, and then compare $Auth_2^*$ and $Auth_2^{P3}$. If they are identical, the ECUs generate K_{TEM} which is the same as the GW's. Algorithm 3 provides the pseudocode for **P3**.

```
Algorithm 3: Authenticated Request to ECU
-------------------------------------------------
UD:
    Selects 64/128 bits RN_1
    Generates HR_1 =h(SK_2||RN_1)
    Send Request with RN_1, HR_1 to GW

GW:

    If HR_1 is valid
        selects 64/128 bits RN_2
        generates HR_2 = h(SK_2||RN_2||RN_1)
        sends response with RN_2, HR_2 to UD

UD:
    If HR_2 is valid
        generates AUTH_1 = f4(SK_1||RN_1||RN_2)
        sends frame request with AUTH_1 to GW
        If it is the initial request
            sends CERT_U_1 to GW

GW:
    Generates AUTH_2= f5(MK_1||AUTH_1||RN_2))
    Generates K_TEM = h(MK_1||RN_2||RN_1)
    If UD firstly requests
        sends RN_1||RN_2||AUTH_2||CERT_U_1 to ECU
    else
        sends RN_1||RN_2||AUTH_2 to ECU

ECU:
    If TS_1, TS_2 are valid
    //by checking CERT_U_1 and CERT_U_2
    //simultaneously
        Generates AUTH_1* = f4(SK_1||RN_1||RN_2)
        Generates AUTH_2* = f5(MK_1||AUTH_1*||RN_2)
    If AUTH_2 == AUTH_2*
        Generates K_TEM = h(MK_1||RN_2||RN_1)
    else
        Filter request

-------------------------------------------------
```

After verifying the request from the GW, the GW and the ECUs can establish a secure channel by using K_{TEM}. The GW now only receives authenticated CAN messages from the ECUs over **N1**, and vice versa. Several existing protocols designed for CAN such as [4, 8, 10, 13] can be used for authenticated frame transmission.

The entire process of our protocol is shown in Fig. 7.

5. EVALUATION

In this section, we present a detailed security analysis of the proposed protocol and calculate the associated performance overheads.

5.1 Security Analysis

Security against key exposure from compromised entities. Suppose key $K_{\mathcal{C}}$ is derived from another key $K_{\mathcal{M}}$, and the attacker $\mathcal{A}$ is now the owner of $K_{\mathcal{C}}$. $\mathcal{A}$ will try to retrieve the key $K_{\mathcal{M}}$ of an *attached* device D. If the probability that $\mathcal{A}$ extracts $K_{\mathcal{M}}$ from $K_{\mathcal{C}}$ is not higher than the probability that $\mathcal{A}$ extracts $K_{\mathcal{C}}$ without it, then we can conclude that the protocol is secure. $K_{\mathcal{C}}$ is derived from $K_{\mathcal{M}}$ using the cryptographic hash function h. The probability that $\mathcal{A}$ derives $K_{\mathcal{M}}$ from $K_{\mathcal{C}}$ is the same as the probability that $\mathcal{A}$

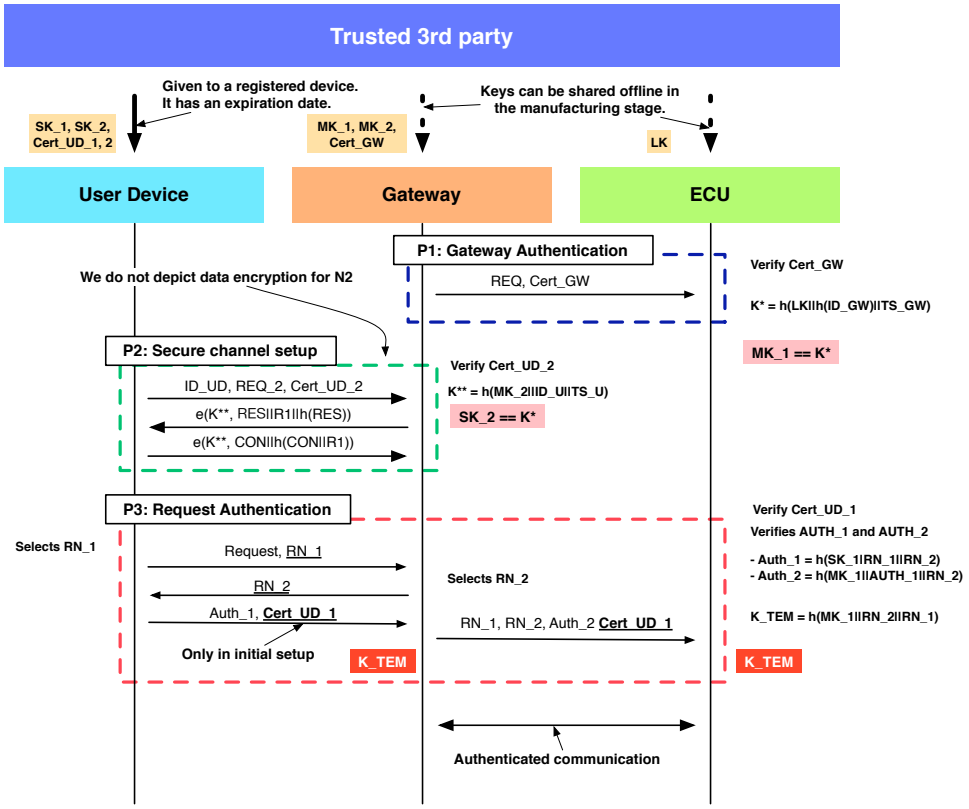

Figure 7: The full 3-step authentication process

finds a message m (a pre-image) from $h(m)$.

$$Pr[\mathcal{F}(MK_1) \equiv LK] \cup Pr[\mathcal{F}(SK_1) \equiv LK] \equiv Pr[\mathcal{F}(h(m)) \equiv m],$$

where $\mathcal{F}$ is a function that finds the pre-image of the input. Thus, it is difficult to derive LK from MK_1 or SK_1, and MK_2 from SK_2 in the protocol.

Security against invalid key usage. If a certificate $Cert_{ID}$ is generated by using timestamp TS, the sender's ID and the receiver's key K, a malicious entity without a valid certificate $Cert_{\mathcal{A}}$ will try to get authenticated by the receiver. The probability that the attacker generates a valid $Cert_{\mathcal{A}}$, such that $Cert_{\mathcal{A}} = Cert_{ID}$ is the same as the probability that the attacker finds a collision in the hash function.

$$Pr[Cert_{\mathcal{A}} \equiv Cert_{ID}] \equiv Pr[h(m) \equiv h(m')],$$

where $m \neq m'$.

Security against impersonation of a user device. In this scenario, a malicious entity $\mathcal{A}$ tries to impersonate a valid UD. $\mathcal{A}$ impersonates a valid UD by sending a fraudulent proof $Auth_{\mathcal{A}}$. The probability of $Auth_{\mathcal{A}} = Auth_0^{P2}$ is the same as the probability that the attacker succeeds in finding a collision in the cryptographic hash function. The difficulty in finding a collision depends on the underlying cryptographic hash function as below.

$$Pr[Auth_{\mathcal{A}} \equiv Auth_0^{P2}] \equiv Pr[h(m') \equiv h(m)],$$

where $m' \neq m$.

Security against Impersonation of GW. Suppose a malicious entity $\mathcal{A}$ tries to impersonate a valid GW that is attached to CAN and communicates with the ECU without a valid key from the GW. $\mathcal{A}$ impersonates a valid GW by sending a fraudulent proof $Auth_{\mathcal{A}}$. The probability of $Auth_{\mathcal{A}} = Auth_*^{P1}$ is the probability that any attacker succeeds in finding a collision in the cryptographic hash functions as below.

$$Pr[Auth_{\mathcal{A}} \equiv Auth_*^{P1}] \equiv Pr[h(m') \equiv h(m)],$$

where $m' \neq m$.

Security against fraudulent requests from a user device. In order for an attacker $\mathcal{A}$ to succeed in receiving vehicle information by sending a fraudulent request, $\mathcal{A}$ should be able to impersonate both the GW and the ECUs. We showed the probability of success in impersonating a valid device in the previous section. Even after a successful impersonation, $\mathcal{A}$ should be able to generate $Auth_{\mathcal{A}}^*$ for a fraudulent request in P3. The probability that an attacker $\mathcal{A}$ who has compromised the GW, also succeeds in sending a fraudulent request to the ECUs, is not higher than the probability that $\mathcal{A}$ succeeds in finding two collisions when the same cryptographic hash function is used in **P2** and **P3**.

Security against fraudulent requests from a malicious GW. Since we assume that the GW authenticated in **P1** is secure, we only need to consider the malicious GW that has not yet been authenticated. Suppose a compromised gateway $GW_{\mathcal{A}}$ attempts to send fraudulent requests with fraudproof $Auth_2^{GW_{\mathcal{A}}}$ to the ECUs. The probability of $GW_{\mathcal{A}}$ succeeding in sending a fraudulent message to an ECU without valid MK_1 is the same as the probability that the attacker finds $Cert_{GW_{\mathcal{A}}}^{P1}$, $Auth_2^{GW_{\mathcal{A}}}$ and $Cert_{GW_{\mathcal{A}}}^{P3}$, where $Cert_{GW_{\mathcal{A}}}^{P1} \equiv Cert_{GW}^{P1}$, $Auth_2^{GW_{\mathcal{A}}} \equiv Auth_2^*$, and $Cert_{GW_{\mathcal{A}}}^{P3} \equiv Cert_{UD}^{P3}$.

5.2 Implementation Overhead

5.2.1 Implementation Setup

For the evaluation of our model, we set up a testbed as shown in Fig. 8. We used Ford's OpenXC library [7] running on Digilent chipKIT Max32 which has a 32-bit processor with 40 MHz processing capability, and a Digilent chipKIT Network Shield. An emulator software running on the Digilent chipKIT Max32 (32-bit, 40 MHz) simulates the CAN bus environment. We used a translation software which converts the raw data sent from the car into a JSON format which is then parsed by the Android device. We used a Toshiba Thrive Android tablet as our user device platform.

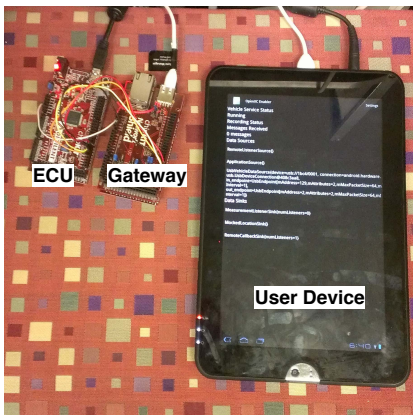

Figure 8: Testbed

The ECU(emulator) performs authentication and verification of the GW when the GW is attached to it. We then attach the UD to the GW. The GW performs Step 2 authentication to verify the user device. The GW will not translate the simulated information if the unauthenticated device is attached to it. The user device only receives data after the 3-step verification is completed. The user can only request preregistered messages.

5.2.2 Implementation Results

We now provide the details of our data frame setup along with the specifications used for the evaluation of the overhead incurred in the protocol. The communications in **P2**, which are done over **N2**, do not increase any perceivable additional overhead, so we only focus on the overheads due to data transmissions during **P1** and **P3**. We assume use of SHA-1 that produces a 160-bit hash output.

First, in case of **P1**, the GW needs to send $Cert_{GW}$ and $Auth_0^{P1}$. Let us use the following data fields: 128-bit **ID**, Universally Unique IDentifier (UUID)[1]; 32/64-bits timestamp (**TS**), 32 (64) bits cover 136 (293 billion) years; 160 bits **Proof** of ID and time using message authentication code.

Six 8-byte CAN frames are required to transmit $Cert_{GW}$:

$$\lceil \frac{Size(ID) + Size(TS) + Size(Proof)}{8} \rceil = 6.$$

If $Auth_0^{P1}$ and **Proof** are set to use 160 bits with SHA-1, we need three frames for transmission. Thus, for **P1**, a total of 9 frames should be transmitted. However, this happens only at the time of initial setup, and such transmissions do not increase the overhead significantly.

For **P3**, the GW sends the following data: RN_1 and RN_2 use a 64-bit random string. $Auth_2^{P3}$ uses 160 bits with SHA-1. Five 8-byte CAN frames are required to transmit the above data.

$$\lceil \frac{(Size(RN_1) + Size(RN_2) + Siz(Auth_2^{P3}))}{8} \rceil = 5.$$

When we set up an authenticated connection using **P3** for the first time, we need to transmit eleven 8-byte CAN frames, including $Cert$ of a UD, to the CAN. The number and size of frames can be reduced using several existing authentication protocols designed for CAN such as [13, 4, 8, 10]. The actual transmission time in the vehicle depends on the CAN arbitration time from the priority and the data transmission intervals. The authors of [12] showed an SAE CAN message period to range from 5 to 1000ms.

The hash computation requires each ECU to perform 4 hash operations, generating $Auth_1^{P3}$, $Auth_2^{P3}$, K_{TEM}, and $Cert_U$ for the initial setup. The hash computation is also known to take a small amount of time. For example, the authors of [8] showed the computation of cryptographic hash functions to take less than 1.2ms when SHA-1 is executed at an 80Mhz clock speed 16-bit microprocessor. When we use 32-bit processors with 40Mhz computation capability, the delay incurred by cryptographic computations is less than 1ms. This is in tune with the SAE standards for safety-critical control applications.[2] Each SHA-1 computation takes 200 μs or 0.2 ms. For the initial establishment, we need a total of 5 SHA-1 computations, which amounts to a 1 ms delay. Thus, the protocol does not incur any significant overhead.

6. CONCLUSION

Connecting in-vehicle networks to the external world introduces serious security risks due to their inherent design for operation in an isolated environment. Moreover, it is difficult to deploy the security mechanisms intended for generic computer networks to vehicular networks due to the significant differences in their intended operational environments. By accounting for these facts and trends, we defined security models for "connected" vehicles, proposed and analyzed

[1]RFC 4122 A UUID URN Namespace, July 2005
[2]SAE Class C Application Requirement Considerations, SAE J2056/1.

a three-step authentication protocol that secures the communication between the in-vehicle network and an external network(mobile device) while meeting the vehicle manufacturers' requirements. In the future, we would like to design a security model against attackers who physically attach a compromised device and directly interact with the in-vehicle network.

Acknowledgement

The work reported in this paper was supported in part by the US Army Research Office under Grant W911NF-12-1-530 and the US Air Force Office of Scientific Research under Grant FA9550-10-1-0393.

7. REFERENCES

[1] AUTOSAR. Automotive Open System Architecture.

[2] M. Chavez, C. Rosete, and F. Henriquez. Achieving Confidentiality Security Service for CAN. In *Electronics, Communications and Computers, 2005. CONIELECOMP 2005. Proceedings. 15th International Conference on*, pages 166–170, 2005.

[3] 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 *SEC'11: Proceedings of the 20th USENIX conference on Security*, pages 1–16. USENIX Association, Aug. 2011.

[4] M. Dworkin. NIST SP 800-38B, Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication. Technical report, NIST, May 2005.

[5] A. Elouafiq. Authentication and Encryption in GSM and 3GUMTS: An Emphasis on Protocols and Algorithms. *arXiv.org*, cs.CR, Apr. 2012.

[6] EVITA. E-safety vehicle intrusion protected applications. EVITA Consortium.

[7] Ford Motors. Inc. The openxc platform. *http://openxcplatform*, 2013.

[8] B. Groza and P.-S. Murvay. Broadcast Authentication in a Low Speed Controller Area Network. *revised postproceedings version of the paper presented at SECRYPT'11, to appear in e-Business and Telecommunications, Springer CCIS*, pages 1–16, Feb. 2012.

[9] 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. *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462, 2010.

[10] D. Nilsson, U. Larson, and E. Jonsson. Efficient In-Vehicle Delayed Data Authentication Based on Compound Message Authentication Codes. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5, 2008.

[11] D. K. Nilsson, U. E. Larson, F. Picasso, and E. Jonsson. A First Simulation of Attacks in the Automotive Network Communications Protocol FlexRay. *Proceedings of the International Workshop on Computational Intelligence in Security for Information Systems CISIS'08*, pages 1–8, Sept. 2008.

[12] T. Nolte, H. Hansson, and C. Norstrom. Probabilistic worst-case response-time analysis for the controller area network. In *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*, pages 200–207. IEEE Computer Society, 2003.

[13] H. Schweppe, Y. Roudier, B. Weyl, L. Apvrille, and D. Scheuermann. Car2X Communication: Securing the Last Meter. *WIVEC 2011, 4th IEEE International Symposium on Wireless Vehicular Communications, 5-6 September 2011, San Francisco, CA, United States*, pages 1–5, June 2011.

[14] R. VandenBrink. Dude, your car is pwned! *SANSFIRE 2012, Washington, DC*, Jul 6th - 15th 2012.

[15] M. Wolf, A. Weimerskirch, and C. Paar. Security in Automotive Bus Systems. *in: Proceedings of the Workshop on Embedded Security in Cars (escar)'04*, pages 1–13, 2004.