SSOV: A Single Sign-on Protocol for Accessing Vehicular Application Services with the Support of Secret Credential Management System

You-Wei Chang, Po-Ching Lin
Department of Computer Science and Information Engineering
National Chung Cheng University
Chiayi, Taiwan
jack85072@gmail.com, pclin@cs.ccu.edu.tw

2020 International Computer Symposium (ICS) | 978-1-7281-9255-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICS51289.2020.00092

***Abstract*—With the rapid development of 5G networks, vehicle-to-everything (V2X) applications will become an essential part of vehicular functions. However, existing login methods for mobile applications such as typing passwords or scanning QR codes are unsafe and inconvenient for moving vehicles. The identity privacy from being tracked is also desired for vehicular applications. In this work, we propose a single sign-on (SSO) protocol, SSOV, for accessing vehicular application services. This protocol is built upon the security credential management system (SCMS) defined in the IEEE 1609.2 standard, and requires minor changes to the original system. This protocol allows the vehicles to access the application services via the pseudonym certificates in the SCMS, and the vehicles can remain anonymous during the login process. This design shares some similarities with OAuth or OpenID Connect, but also bears two unique features. First, SSOV is specifically designed for vehicles and does not require any password for the login process. Second, unlike the other two SSO standards, SSOV does not require multiple identity providers, which will introduce extra difficulties in verification and also increase security risks. We analyze the security of SSOV in terms of the confidentiality of the key parameters, the impact of a specific compromised role on the system, and how we will prevent it in the threat model.**

***Index Terms*—Single sign-on, SCMS, V2X communication, anonymity, authentication**

I. Introduction

Owning to the rapid development of 5G networks and the Internet [1], vehicular networking will be increasingly mature in the near future [2]. Various vehicular application services such as infotainment and maps on the head unit are getting popular in modern cars. However, most of them are independently developed and use their own authentication methods. Without a standardized identity authentication method, the vehicle anonymity may be uncontrollable and undermined. As long as one of the application services gives rise to a privacy risk, the anonymity of a vehicle will be broken.

To address the above problem in this work, we present a method, namely Single-Sign-On-for-Vehicle (SSOV), to preserve the anonymity of vehicles when they access the vehicular applications. This method leverages the Security Credential Management System (SCMS) [3] designed for vehicular communication. The SCMS has been adopted in the IEEE 1609.2 standard [4], and can provide changing pseudonymous certificates for the vehicle privacy. The certificates are generated from the butterfly key, and are updated without keeping sending the public keys to the SCMS, thereby increasing the scalability in the certificate management significantly. With the support of the SCMS, the vehicles with the pseudonymous certificates can ensure anonymity while allowing other entities (e.g., vehicles, the components in the infrastructure) to verify the signatures of the messages. The vehicular applications can also leverage the well established SCMS for vehicle authentication without the vehicles to register the applications beforehand. As a result, this method can realize the single-sign-on (SSO) function for the vehicular applications while preserving the identity privacy of vehicles.

This SSO function in this work can relieve the inconvenience for vehicles to register individual accounts and passwords in different application services. Maintaining multiple accounts may also raise potential security concerns if the same account names or even passwords are used across multiple application services. Thus, the identity privacy of vehicles can be also inherently protected due to the support of privacy preservation in the SCMS. The application services also do not need to maintain the database of stored accounts and passwords, thereby reducing the risk of personal data leakage if the database is compromised. They can also leverage the certificate revocation list established in the SCMS to prevent misbehaving attackers.

To support the SSO function in this work, we add an authorization server (AS) to the original SCMS. An application service just needs to redirect the messages from a vehicle to the AS in the SCMS, which is responsible for the authentication of the vehicle. The AS will encrypt the linkage value and send to the application services via the vehicle; thus, in a certain period of time, the vehicle can log into the application services again without the intervention from the AS.

The remainder of this work is organized as follows. We will introduce some background knowledge about the SCMS and present its role in SSOV in Section II. We will also review the SSO-related standards such as OAuth, OpenID Connect in that section. Section III will describe in detail the design of

978-1-7281-9255-0/20/$31.00 ©2020 IEEE
DOI 10.1109/ICS51289.2020.00092

SSOV and discuss its security. In Section IV, we will show the security verification of SSOV and assess its efficiency. Finally, we will conclude and future work in Section V.

II. Related Work

Before introducing the system, we need to explain the basics of the SCMS and review typical SSO methods such as OAuth and OpenID Connect.

A. SCMS

The SCMS is designed to manage the identities and credentials of vehicles in V2X communication, while preserving their privacy. It supports four main functions: registration, certificate issuance, misbehavior reporting and revocation. The functions are scattered in several components to prevent a single component from knowing too much about the vehicle data. The SCMS features the ability to manage and issue a large number of pseudonym certificates [5] efficiently. The vehicles can use different pseudonym certificates over different time periods for communication without being tracked. The SCMS uses the butterfly key expansion to significantly reduce the number of requests for a large number of pseudonym certificates. For efficient certificate revocation, the SCMS inserts a linkage value into each pseudonym certificate. This scheme allows the SCMS to revoke all the pseudonym certificates of the malicious vehicles after the revocation time.

In the SCMS, butterfly key expansion enables the generation of a large number of pseudonym certificates, public and private keys efficiently. A vehicle sends just one request to the registration authority (RA) with one public key seed and two expansion functions. This key expansion is based on elliptic curve encryption, but can be replaced with any algorithm based on the difficulty of discrete logarithmic problems. Suppose G is an agreed base point of some order l. The caterpillar key pair of the butterfly key is an integer a, and $A = aG$. A and the expansion function $f_k(\iota)$, where $\iota = (i, j)$ denotes the jth pseudonym certificate in the ith time period (e.g., a week), are sent to the RA when the vehicle requests for a pseudonym certificate. When the RA receives the request, the corresponding cocoon public keys $B_\iota = A + f_k(\iota) * G$ will be generated according to the time period, the number of certificates required in a time period and the expansion functions. The vehicle can calculate the corresponding private key $b_\iota = a + f_k(\iota) * G$. The pseudonym certificate authority (PCA) also introduces a random factor into the cocoon public key, so that even the RA cannot track the vehicle identities. Readers are referred to SCMS [4] for more details.

A linkage value is inserted into each pseudonym certificate for efficient certificate revocation [6]. The SCMS assumes two linkage authorities (LAs) in the system. Either LA has its own 32-bit identifier la_id_1 or la_id_2. Consider the first linkage authority LA_1. The first step of generating the pre-linkage value is to take the 128-bit randomly generated initial linkage seed at the time of registration as $ls_1(0)$. The second is to calculate the linkage seed for each time period $i > 0$, $ls_1(i) \leftarrow H_u(la\_id_1 \| ls_1(i-1))$. $H_u(m)$ denotes the output m of SHA-256 with a maximum of u bytes. The pre-linkage value of the jth pseudonym certificate in the ith time period is derived from a function of the linkage seed, LA identifiers and (i, j). The linkage value is the xor output of the pre-linkage values from both LAs. In this way, if the certificates of a misbehaving vehicle is to be revoked, the SCMS can get the linkage seed through the linkage value in the certificate, and then calculate all the linkage values after the time period. Thus, the subsequent pseudonym certificates with these linkage values can be revoked.

B. Single sign-on

Single sign-on (SSO) [7] allows the users to register only once and access multiple systems with a single identity. In contrary, on many websites and application services, the users have to register a unique identity for authentication before accessing them. Requesting the users to set a unique identity and a secure password on every website is troublesome. If a user uses the same password on multiple websites, the password may be broken once it is broken on one of the websites. Using SSO can reduce password fatigue from different username and password combinations and the time to re-enter passwords for the same identity. Moreover, for smaller sites without the capability to securely maintain user accounts. SSO provides a convenient way to log in, but it also brings security problems if the login method of SSO is compromised. Many studies have examined the SSO security. Wang et al. presented an attack to SSO provided by Facebook [8]. They leverage an SSO flaw by which the attacker's authorization can be used to log into the user's account.

C. OAuth and OpenID Connect

Open Authorization (OAuth) [9] is an open standard that allows user-authorized third-party applications to access a user's private resources on a website, but without providing a username or password to the third-party application. When a user authorizes a third-party application for a specific resource, that application obtains an access token from the authorization server. An access token is a specific string that defines the scope and expiration of the token for a third-party application and includes a random number to avoid replay attacks. The entire process of the OAuth authorization steps may require user intervention. The situation depends on what method the identity provider (IdP) uses to authenticate the user. We do not adopt OAuth in this work because its main goal is to provide third-party application resources licensing and use, and there are multiple steps to exchange information with the resource owner. However, the SCMS needs only identification authentication without those overheads.

OpenID Connect (OIDC) [10] is widely used on various websites to authenticate users. Its major difference from OAuth is that its purpose is to authenticate users, whereas OAuth is to license resources to third-party applications. OIDC is an open identity authentication protocol that enables application services to be logged into with a third-party identity provider, which can be created by any organization or

individual. Because OIDC is based on OAuth 2.0 on the modified supplement, it supplements OAuth with respect to many deficiencies. First, pseudo-authorization is necessary to implement SSO in OAuth. In that way, the user authorizes a third-party application to a non-existent resource, so that the application can use OAuth to authenticate the user identity. Second, the IdP provides an ID token that contains the user information required for authentication. This token is different from the OAuth access token, but the specification states that the IdP provides the ID token with the user information and the digital signature of the certification organization [11].

The OAuth and OpenID Connect standards are highly related; thus their security problems will be discussed together. Fett et al. [12] conducted a systematic analysis of their security problems. Summarily speaking, the attacks are both related to fake IdPs. The first is a man-in-the-middle attack that tricks users into using AIdP (a fake IdP controlled by the attacker) for authentication and sending the collected data to the honest IdP to obtain authorization from the user. The second is that the attacker first obtains an authorization code from the honest IdP. When the user logs into the application service using AIdP, the authorization code is attached to the redirected URL of the application service to spoof this service. However, both standards provide third parties with the ability to build their own IdP, making it difficult to identify forgeries [13].

III. Research Method

In this section, we will present the SSOV protocol that leverages the SCMS to provide anonymous identity authentication for users to access the third-party services.

A. System Overview

The original SCMS framework does not support the authorization methods for accessing the third-party applications. If the application developers design their own authentication methods, the vehicles may be likely to lose their anonymity because their identities are not pseudonyms provided by the SCMS, but ordinary accounts in those applications. To solve this problem without overhauling the original SCMS architecture, we introduce an authorization server (AS) into it. The AS can authorize the vehicles to access the third-party services by interacting with the components in the original SCMS.

The message flows in the SSOV protocol are illustrated in Fig. 1. First, the vehicle initiates a request to log into the application service (step 1), which only needs to verify the pseudonym certificate from the vehicle, but leaves the check whether the vehicle is legal to the SCMS by redirecting the vehicle's request to the AS (steps 2-3). The SCMS maintains the legality of the vehicles, and may revoke a vehicle's legality due to its misbehavior. In the following steps, if the vehicle is illegal, the application service will not get a correct authorization code to obtain the linkage value of the vehicle from the SCMS; otherwise, the AS will deliver an authorization code to the vehicle (step 4), which then passes the code to access the application service (step 5). The application service can obtain the encrypted linkage value from the AS (steps 6-7). Thus, when the vehicle accesses the application service again, the linkage value can be derived from the vehicle's certificate validation step for login verification, which does not require the SCMS intervention for a certain period of time. The roles in this system will introduced in Section III-B.

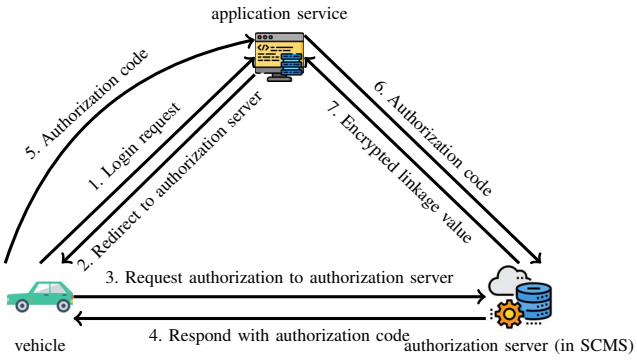

Fig. 1. Message flows in the SSOV protocol.

B. Roles

The roles in this system and their functions are introduced in the following, and the differences between our proposed system and the original SCMS will be clarified.

- Vehicles: We assume the vehicles support V2X communication, and use the pseudonym certificates provided by the SCMS. The on-board unit (OBU) on the vehicles can be installed with several third-party applications to access the V2X services that provide convenient features to the vehicles.
- Application Services: An application service requires registration with the SCMS to obtain a client identifier (CID) and client secret for identifying itself in the subsequent verification. When receiving the vehicle login request, the service verifies the vehicle's certificate, then attaches its CID, and redirects the request to the AS, which will verify the vehicle successfully. The application service will receive the authorization code from the AS through the vehicle, and it can use this code to obtain the encrypted linkage value of the vehicle from the AS.
- Authorization Server (AS): It is part of the revised SCMS. When receiving the request redirected from the application service via the vehicle, the AS will check whether the vehicle has been revoked or not. If not, the AS will return the authorization code to be redirected to the application service's callback. When the application service sends the authorization code, its own CID and client secret to the AS, the AS will validate the application service, and if successful, returns the vehicle's encrypted linkage value in the current time period and its unique identifier (UID), a hash value of both linkage chain identifiers (lci_1, lci_2) [3] and client secret, to the application service.

C. Details of the protocol

For clarification, we will discuss the operations in two parts: The first consists the protocol messages between a vehicle, an application service, and an AS. The second is the protocol messages between the components inside the revised SCMS.

1) Messages outside the SCMS: The transmission of messages outside the SCMS is illustrated in Fig. 2. We describe the detail in each step as follows.

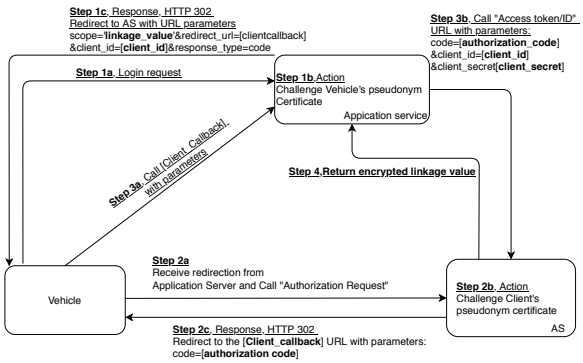

Fig. 2. Protocol messages outside the SCMS.

1) When the vehicle sends a login request to the application service, it will attach its own pseudonym certificate for verification and perform the Diffie-Hellman (DH) key exchange for encrypting subsequent data transmission (steps 1a-1b). After ensuring the transmission security and the validity of the certificate, the application service returns a response to redirect the request to the AS (step 1c). The data in it include the callback and the CID of the application service, and the vehicle's linkage value.
2) When the AS receives the redirected request via the vehicle from the application service, it will verify the pseudonym certificate of the vehicle and perform DH key exchange with the vehicle (steps 2a-2b). If the verification is valid, it will send an authorization code to the application service's callback via the vehicle, and store the linkage value, the verification expiration time and the authorization code in the database (step 2c). The verification expiration time limits the effective time of the authorization code. If the verification fails, a failure message is sent back to the application service.
3) The vehicle sends back the authorization code to the application service's callback. The application service will perform DH key exchange with the AS, and then the application service will use the authorization code and attach the CID and client secret to request for the information from the AS (step 3a). The data includes the encrypted linkage values $AES(hash(linkage\_value)||client\_secret)$ and the UID $hash(lci_1||lci_2||client\_secret)$ (steps 3b-4).

2) Messages within the SCMS: The messages communicated within the SCMS are illustrated in Fig. 3. The steps to protect the anonymity and untraceability of the vehicles will adhere to the SCMS standards and are described as follows.

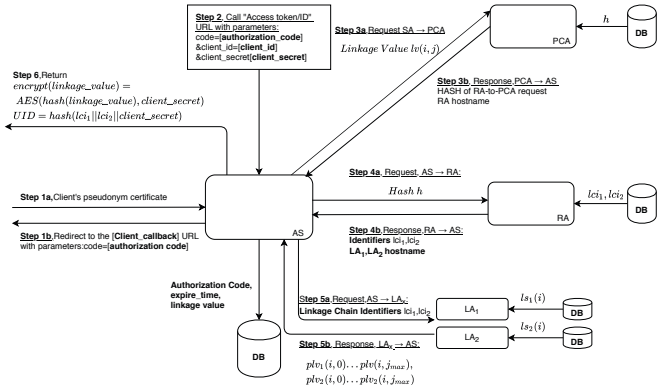

Fig. 3. Protocol messages within the SCMS.

1) When receiving a redirected request from a vehicle, the AS first verifies the vehicle's pseudonym certificate, performs DH key exchange with the vehicle after successful verification to secure the subsequent transmission, stores the authorization code, expiration time and linkage value into the database (step 1a), and finally redirects the authorization code from the vehicle to the application service callback (step 1b).
2) Upon receiving the authorization code, the application service verifies the certificate with the AS, performs the DH key exchange with the AS, and finally uses CID, client secret and authorization code to request for the encrypted linkage value from the AS (step 2).
3) When the AS receives the request, it will first check whether the authorization code is correct. If it is correct, the AS will send the corresponding linkage value to the PCA, which will send back the hash of the RA-to-PCA certificate request at the time of registration and the RA's host name (steps 3a-3b).
4) The AS will send the hashed data derived from the PCA to the RA, which will return the corresponding linkage chain identifiers (lci_1, lci_2) with the host names of LA_1, LA_2 (steps 4a-4b).
5) The AS sends (lci_1, lci_2) to the LAs, which get the corresponding linkage seeds $(ls_1(i), ls_2(i))$ from the database to derive the pre-linkage values of the vehicle over the time periods and respond with them to the AS (steps 5a-5b).
6) The AS computes the linkage values of the vehicle from the pre-linkage values by the xor operation, hashes each linkage value with the client secret and also lci_1, lci_2 obtained in step 4 with the client secret to get the UID, and responds with the results back to the application service (step 6).

D. Possible Threats and Defense

There are several possible threats and we will explain how to defend them and reduce the damages to the vehicle

privacy. First, the DH key exchange must be completed for the vehicle, application services and AS to establish secure communications with each other. However, this scheme is known to be vulnerable to man-in-the-middle attacks, but this problem can be easily avoided by using the certificates of the participating roles.

Furthermore, if an attacker has compromised a vehicle, both the AS and the application service will still verify the vehicle's pseudonym certificate during the login phase; thus, there is no way to spoof the application service access because the attacker does not get the private key corresponding to the public key in another vehicle's pseudonym certificate. As a result, the attacker cannot impersonate other vehicles.

If an attacker has compromised an application service, in the encryption steps in Section III-C1, the attacker can only get the encrypted linkage values. The attacker cannot use these values to log in to other applications, nor can he/she infers the user's linkage value because a large number of pseudonymous certificates are given to the vehicle over a period of time.

IV. Security Proof and System Evaluation

We use ProVerif (prosecco.gforge.inria.fr/personal/bblanche/proverif) to reason the security properties of this system. We express the communication process and the encryption in the system into the syntax supported by ProVerif. The following steps are involved in the security proof: (1) to simplify and abstract the system process, (2) to implement the DH exchange, and (3) to declare the vehicle's public key and make the application services and the AS publicly accessible. The private information, which the attacker cannot access, includes the authorization code and the encryption key during the conversation. The other information not marked as private will be known to the attacker by default. ProVerif will dynamically analyze the imported processes, automatically supplement more rules according to them, and attempt to change the protocol flow or use a man-in-the-middle attack to try to get information marked as private.

In Listing 1, we first define the communication channels between the vehicles, the application services, and the AS in scripts. We assume the channels are public and anyone can capture packets transmitted over them. The scripts define the basic data types needed for encryption and decryption, such as public keys, private keys, and the base point of the elliptic curve. In lines 10–32, the algorithms of shared key encryption, DH encryption, and signature are expressed in ProVerif. In lines 34–39, we define the information to be transmitted throughout the system and its data type, and mark the confidential information as private to resist the attacker. In lines 40–42, we mark the target of the attack, and then hand it over to ProVerif, which will push the entire system flow to see whether the attacker has a chance to obtain it.

Listing 1. Basic Definition

```
1 free VtoServer: channel.
2 free VtoAServer: channel.
3 free StoAServer: channel.
4
5 type pkey.
6 type skey.
7 type G.
8 type exponent.
9
10 fun hash(bitstring): bitstring.
11 (* Signatures *)
12
13 fun sign(G, skey): bitstring.
14 fun pk(skey):pkey.
15 reduc forall m: G, k: skey; getmess(sign(m,k)) = m.
16 reduc forall m: G, k: skey; checksign(sign(m,k), pk(k)) =
       m.
17
18 (* Symmetric encryption *)
19
20 fun s_enc(bitstring,bitstring):bitstring.
21 reduc forall x: bitstring, y: bitstring;
       s_dec(s_enc(x,y),y) = x.
22
23 (* Shared key encryption *)
24
25 fun enc(bitstring, G): bitstring.
26 reduc forall x: bitstring, y: G; dec(enc(x,y),y) = x.
27
28 (* Diffie-Hellman *)
29
30 const g: G.
31 fun exp(G, exponent): G.
32 equation forall x: exponent, y: exponent; exp(exp(g, x),
       y) = exp(exp(g, y), x).
33
34 free linkage_value: bitstring.
35 free client_id: bitstring.
36
37 free client_secret: bitstring [private].
38 free encrypt_linkage_value: bitstring [private].
39 free auth_code: bitstring [private].
40 query attacker(client_secret).
41 query attacker(encrypt_linkage_value).
42 query attacker(auth_code).
```

In Listing 2, we present the implementation of the AS communication processes with ProVerif. In line 3, the AS receives the login requests from the application service transmitted via the vehicle. In lines 6–9, the AS verifies the vehicle's certificate, and if verification is correct, it performs a DH key exchange with the vehicle for a symmetric key. In line 12, the AS returns `auth_code` to the application service via the vehicle. When the application service sends a request with `auth_code`, it will verify the certificate of the application service first and then whether the `auth_code` is correct. The AS will send back the encrypted linkage value to the vehicle.

Listing 2. Authorization server

```
1 let AServer(skAS: skey, pkAS: pkey, pkV: pkey, pkS: pkey) =
2     in(VtoAServer,pkZ: pkey);
3     new n2:exponent;
4     out(VtoAServer,(sign(exp(g,n2),skAS),pkAS));
5     in(VtoAServer,m5:bitstring);
6     let x3 = checksign(m5,pkZ) in
7     let k1 = exp(x3,n2) in
8     if pkZ = pkV then
9        in(VtoAServer,(m7:bitstring,m8:bitstring));
10       let s_client_id = dec(m7,k1) in
11       let v_linkage_value = dec(m8,k1) in
12       out(VtoAServer,enc(auth_code,k1));
13       (*Sever to AServer*)
14       in(StoAServer,pkY:pkey);
15       new n3:exponent;
16       out(StoAServer,(sign(exp(g,n3),skAS),pkAS));
17       in(StoAServer,m12:bitstring);
18       let x4 = checksign(m12,pkY) in
19       let k2 = exp(x4,n3) in
20       if pkY = pkS then
21          in(StoAServer,m13:bitstring);
22          let check_code = dec(m13,k2) in
23          if check_code = auth_code then
24             out(StoAServer,enc(encrypt_linkage_value,k2)).
```

Listing 3 presents the implementation of application services. Lines 2 to 11 show the communication between the vehicle and the application service. The application service receives the certificate of the vehicle, verifies the certificate and obtains the vehicle's linkage value. Lines 12 to 13 represent the authorization code that the AS transmits to the application service from the vehicle. Lines 15 to 25 show the authorization code used by the application service to exchange the vehicle's encrypted linkage value with the AS. Lines 26 to 28 run a

hash of the vehicle's linkage value, and then do another hash with the client secret to get the encrypted linkage value, and compare the encrypted linkage value with the one from the AS. If both match, then the login is successful.

Listing 3. Application Service

```
let Server(skS: skey, pkS: pkey, pkV: pkey, pkAS: pkey) =
   in(VtoServer,pkX:pkey);
   new n0:exponent;
   out(VtoServer,(sign(exp(g,n0),skS),pkS));
   in(VtoServer,m1:bitstring);
   let x1 = checksign(m1,pkX) in
   let k = exp(x1,n0) in
   if pkX = pkV then
      in(VtoServer,m2:bitstring);
      let v_linkage_value = dec(m2,k) in
      out(VtoServer,enc(client_id,k));
      in(VtoServer,m10:bitstring);
      let as_auth_code = dec(m10,k) in
      (*Sever to AServer*)
      out(StoAServer,pkS);
      in(StoAServer,(m11:bitstring,pkY:pkey));
      let x2 = checksign(m11,pkAS) in
      new n4:exponent;
      let k1 = exp(x2,n4) in
      out(StoAServer,sign(exp(g,n4),skS));
      if pkY = pkAS then
         out(StoAServer,enc(as_auth_code,k1));
         in(StoAServer,m13:bitstring);
         let enc_linkage_value = dec(m13,k1) in
         let login_check =
               s_enc(hash(v_linkage_value),client_secret) in
         if login_check = enc_linkage_value then
            new Login_success:bitstring;
            out(VtoServer,enc(Login_success,k)).
```

The messages for vehicle are presented in Listing 4. From Lines 2 to 11, the vehicle sends the login request to the application service and performs certificate verification and key exchange. From line 12 to line 22, the vehicle message is redirected to the AS via the application service, and after AS authentication, it obtains the authorization code. Line 23 is to send the authorization code received from the AS to the application service. Line 24 is to receive the login result back from the application service.

Listing 4. Vehicle

```
let Vehicle(skV: skey, pkV: pkey, pkS: pkey, pkAS: pkey) =
   out(VtoServer,pkV);
   in(VtoServer,(m0:bitstring,pkY:pkey));
   let x0 = checksign(m0,pkS) in
   new n1: exponent;
   let k = exp(x0,n1) in
   out(VtoServer,sign(exp(g,n1),skV));
   if pkY = pkS then
      out(VtoServer,enc(linkage_value,k));
      in(VtoServer,m3:bitstring);
      let s_client_id = dec(m3,k) in
      out(VtoAServer,pkV);
      in(VtoAServer,(m4:bitstring,pkZ:pkey));
      let x2 = checksign(m4,pkAS) in
      new n3: exponent;
      let k1 = exp(x2,n3) in
      out(VtoAServer,sign(exp(g,n1),skV));
      if pkZ = pkAS then
         out(VtoAServer,(enc(client_id,k1),
               enc(linkage_value,k1)));
         in(VtoAServer,m9:bitstring);
         let as_auth_code = dec(m9,k1) in
         out(VtoServer,enc(as_auth_code,k));
         in(VtoServer,m10:bitstring).
```

We implemented SSOV to simulate the vehicle login application service on the website and compared it with the application using regular Google OAuth [14] authentication. The application service website and the AS are built using caddy as a reverse proxy (caddyserver.com) and the Python website framework Flask, and MySQL to store the necessary information. Because Google OAuth requires a user mouse click to agree for new users to log in, it does not reflect the actual execution time. On the SSOV protocol, it takes an average of 1.45 seconds for a new user to log in, excluding the time within. If a user has logged in before, Google OAuth takes 2.10s, while SSOV takes 0.44s. Thus, the time spent on SSOV has only minor effect on the user's login experience.

V. Conclusion and Future Work

In this work, we present SSOV as an SCMS-based protocol to provide vehicles to use anonymous SSO for V2X application services. Our solution adds an AS with only minor modification on the SCMS, and still maintains the anonymity of the vehicle to the SCMS. We use the pseudonym certificates generated by the SCMS to implement the AS for user authentication and legitimacy checks on vehicles. For V2X applications, SSOV provides an easy way for vehicles to log in and also eliminates malicious users. The users will have better experiences of accessing the application services because they no longer need to register an account for each application service. The users do not need to manually enter their account and password when accessing the application services. Also, the access with SSOV ensures that the vehicle is anonymous when authenticated with the application services.

In the future work, we will build on this mechanism to extend the functionality, for example, by providing vehicles with access to application services on the SCMS framework while allowing the application services to charge the vehicles and keep the vehicles anonymous.

References

[1] cuereport. Cellular vehicle-to-everything (c-v2x) - market size, growth forecast 2020 to 2025. Technical report, 2020.

[2] Global Info Research. United states automotive vehicle to everything (V2X) market by manufacturers, states, type and application, forecast to 2023, 2018.

[3] B. Brecht, D. Therriault, A. Weimerskirch, W. Whyte, V. Kumar, T. Hehn, and R. Goudy. A Security Credential Management System for V2X Communications. *IEEE Transactions on Intelligent Transportation Systems*, 19(12), pp. 3850–3871, Dec. 2018.

[4] IEEE standard for wireless access in vehicular environments–security services for applications and management messages - amendment 2–PDU functional types and encryption key management. *IEEE Std 1609.2b (Amendment to IEEE Std 1609.2-2016)*, pages 1–30, 2019.

[5] A. Pfitzmann and M. Köhntopp. *Anonymity, Unobservability, and Pseudonymity — A Proposal for Terminology*, pages 1–9. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[6] J. J. Haas, Y. Hu, and K. P. Laberteaux. Efficient certificate revocation list organization and distribution. *IEEE Journal on Selected Areas in Communications*, 29(3):595–604, 2011.

[7] J. De Clercq. Single sign-on architectures. In *International Conference on Infrastructure Security*, pages 40–58. Springer, 2002.

[8] R. Wang, S. Chen, and X. Wang. Signing me onto your accounts through facebook and google: A traffic-guided security study of commercially deployed single-sign-on web services. In *IEEE Symposium on Security and Privacy*, pages 365–379, 2012.

[9] (Ed.). D. Hardt. Rfc6749 - the oauth 2.0 authorization framework, 2012. https://tools.ietf.org/rfc/rfc6749.txt.

[10] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore. Openid connect core 1.0 incorporating errata set 1. openid foundation.

[11] M. Jones, J. Bradley, and N. Sakimura. JSON web token (jwt). *Tech. Rep.*, 2015.

[12] D. Fett, R. Küsters, and G. Schmitz. A comprehensive formal security analysis of oauth 2.0. In *ACM SIGSAC Conference on Computer and Communications Security*, CCS, 2016.

[13] C. Mainka, V. Mladenov, and J. Schwenk. Do not trust me: Using malicious IdPs for analyzing and attacking single sign-on. In *IEEE European Symposium on Security and Privacy* (EuroSP), 2016.

[14] Google oauth 2.0 overview. https://developers.google.com/identity/protocols/oauth2.