An OAuth2-based Protocol with Strong User Privacy Preservation for Smart City Mobile e-Health Apps

Victor Sucasas, Georgios Mantas, Ayman Radwan, and Jonathan Rodriguez
Instituto de Telecomunicações, Aveiro, Portugal

***Abstract*—In the context of the Smart City concept, mobile e-Health applications can play a pivotal role towards the improvement of citizens' quality of life, since they can enable citizens to access personalized e-Health services, without limitations on time and location. However, accessing personalized e-Health services through citizens' mobile e-Health applications, running on their mobile devices, raises many privacy issues in terms of citizens' identity and location. These privacy issues should be addressed so that citizens, concerned about privacy leakage, will embrace Smart City mobile e-Health applications and reap their benefits. Hence, in this paper we propose an OAuth2-based protocol with strong user privacy preservation that addresses these privacy issues. Our proposed protocol follows the OAuth2 protocol flow and integrates a pseudonym-based signature scheme and a delegation signature scheme into the user authentication phase of the OAuth2 protocol. The proposed protocol enables citizens authentication towards the servers providing personalized e-Health services, while preserving their privacy from malicious mobile applications and/or eavesdroppers. Moreover, the proposed protocol does not require to store sensitive information in the citizens' mobile devices.** [1]

I. Introduction

Smart City is an emerging concept whose main goal is to improve the quality of life of its citizens by leveraging Information and Communication Technologies (ICTs) as the key medium [1]. Towards this direction, Smart City mobile e-Health applications (apps) that are able to gain citizen's authorization, and access personalized e-Health services on behalf of the citizen, can play a pivotal role [2]. In the context of Smart City, personalized e-Health services, enhancing citizens' quality of life, can be developed and provided by Service Providers (SPs) utilizing emerging technologies and data available in Smart City. On the other hand, Smart City mobile e-Health apps can be created either by the SPs, providing the personalized e-Heath services, or by third-party developers making use of the Application Programming Interfaces (APIs) of the SPs [3]. It is worthwhile to mention that Smart City mobile apps developers can take advantage of the growing popularity of powerful mobile devices (e.g. smartphones, tablets) and the large variety of their connectivity options (e.g., 2G/3G/4G, IEEE 802.11, Bluetooth) to create mobile apps that enrich the citizen experience in terms of access to personalized e-Health services without limitations on time and location.

However, due to the fact that the citizens' personalized e-Health services incorporate sensitive information, derived from resources such as profile data, contextual or even vital data, the need for authentication and authorization solutions that enable secure access to them is of paramount importance [4]. Today, the most widely used solution for mobile apps to gain authenticated and authorized access to end-user's services on remote SPs is the OAuth protocol that has been adopted by major service providers such as Facebook, Google and Microsoft [5], [6]. OAuth was primarily designed to provide a process for end-users to authorize third-party websites to access their resources stored on a SP on their own behalf. Nonetheless, ever since OAuth was successfully adopted by the industry, major identity providers (e.g. Facebook, Google, Microsoft) have used it for user authentication as well. Furthermore, the developers' community has retargeted OAuth to mobile applications, in addition to web applications. Hence, OAuth has become a de-facto authentication and authorization protocol for mobile apps [5]. The latest version of OAuth protocol is the OAuth2 protocol that has obsoleted the OAuth1 protocol [7].

Although OAuth2 has already been supported widely for mobile apps, it has been vulnerable to mobile malware attacks targeting end-user's credentials [5], [8], [9]. Malicious mobile apps, made to look like innocent mobile apps, can be downloaded and installed on citizen's mobile device exposing him/her to credential theft attacks [10], [11]. These attacks can lead to citizen's identity and location privacy issues since they can allow attackers not only to steal citizen's real identity but also to track his/her physical location by linking citizen's real identity to location information that may be included in transmitted messages [4]. Nevertheless, these identity and location privacy issues will prevent citizens, concerned about privacy leakage, from embracing Smart City mobile e-Health apps and reaping their benefits. Therefore, in this paper, we propose an OAuth2-based protocol for Smart City mobile e-Health apps that addresses the citizen's identity and location privacy issues by providing strong user privacy preservation. Particularly, the proposed scheme adopts the OAuth2 protocol flow for mobile e-Health apps and enhances the end-user authentication process by integrating a pseudonym-based signature scheme and a delegation signature scheme in order to achieve user privacy preservation for Smart City mobile e-Health apps.

Following the introduction, this paper is organized as follows. In Section II, the OAuth2 protocol flow on mobile apps is presented. Furthermore, vulnerabilities of the OAuth2 protocol for mobile apps and possible attacks that exploit these vulnerabilities are discussed. In Section III we provide the system model. Section IV the proposed OAuth2-based protocol with strong user privacy preservation is presented. The security analysis of the proposed protocol is provided in section V. Finally, section VI concludes this paper.

[1] The research leading to these results has received funding from the Fundação para a Ciencia e Tecnologia and the ARTEMIS JU (ACCUS ARTEMIS-005-2012 / GA number 333020). Author Radwan also acknowledges the grant of the Fundação para a Ciencia e a Tecnologia (FCT - Portugal): SFRH/BPD/104585/2014.

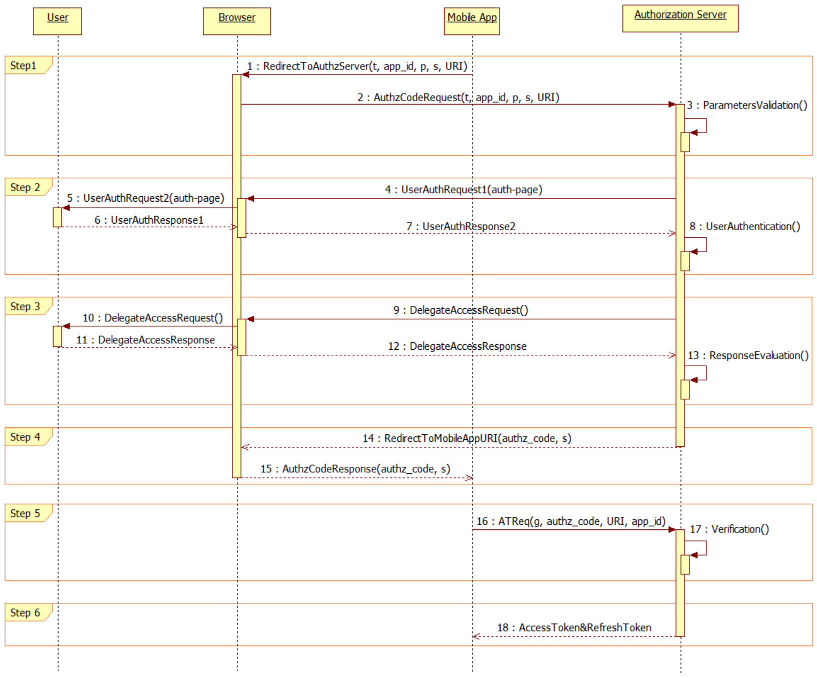
User
Browser
Mobile App
Authorization Server
Step1
1 : RedirectToAuthzServer(t, app_id, p, s, URI)
2 : AuthzCodeRequest(t, app_id, p, s, URI)
3 : ParametersValidation()
Step 2
4 : UserAuthRequest1(auth-page)
5 : UserAuthRequest2(auth-page)
6 : UserAuthResponse1
7 : UserAuthResponse2
8 : UserAuthentication()
Step 3
9 : DelegateAccessRequest()
10 : DelegateAccessRequest()
11 : DelegateAccessResponse
12 : DelegateAccessResponse
13 : ResponseEvaluation()
Step 4
14 : RedirectToMobileAppURI(authz_code, s)
15 : AuthzCodeResponse(authz_code, s)
Step 5
16 : ATReq(g, authz_code, URI, app_id)
17 : Verification()
Step 6
18 : AccessToken&RefreshToken

Fig. 1. OAuth2 protocol flow for mobile apps.

II. OAuth2 and Mobile Apps

According to [12], the OAuth2 protocol defines four different roles: a) the Resource Owner: an entity capable of granting access to its protected resources, b) the Resource Server: the server hosting the Resource Owner's protected resources, c) the Client: an application requesting access to the protected resources on behalf of the Resource Owner, and d) the Authorization Server (AS): the server that issues access tokens to the Client after successfully authenticating the Resource Owner and obtaining authorization.

In the implementations of the OAuth2 protocol for mobile apps, the Resource Owner is the user that holds the protected resources and the Client is the mobile app, installed and executed on the user's mobile device, requesting access to the users protected resources. In addition, the user-agent can be implemented by an embedded web browser or a system native browser. An embedded web browser is a User Interface (UI) view component that is embedded in the mobile app in order to display online contents within the hosting mobile app. On the other hand, the system native browser is a separate app, running on the mobile device as well, and is not embedded in the mobile app. However, both of them provide similar functionality in the OAuth2 protocol flow. For this reason, we will refer to them as browser [5], [8].

The most common implementations of the OAuth2 protocol for mobile apps are based on browser redirection. The mobile app redirects the user's browser to the AS, which interacts with the user, and then it redirects the user's browser back to the mobile app. The AS authenticates the user, identifies the mobile app to the user, and then requests the user to decide whether he/she will grant or deny authorization to the mobile app to access protected resources on behalf of him/her. Assuming that the user grants access, the AS redirects the user's browser back to the mobile app with an authorization code. Once the mobile app holds the authorization code, it can request the AS for an access token, which will grant the mobile app access to the requested user's resources hosted in the Resource Server [5], [8]. As it is illustrated in Fig. 1, the OAuth2 protocol flow for mobile apps consists of the following steps [5], [6], [8], [13], [12]:

Step 1: The Mobile App initiates the flow by redirecting the Browser to the Authorization Server in order to request the Authorization Code (msg 1 & 2). Particularly, the Mobile App sends to the Authorization Server via the Browser the following: a) the response type (t=code) b) the Mobile App identifier app_id assigned during the registration process with the Authorization Server, c) the requested permission scope p, d) an optional state parameter s used to maintain the state between the request and response, and e) a redirection URI to which the Authorization Server will redirect the Browser back once access is granted (or denied). Upon receipt of the AuthzCodeRequest(t, app_id, p, s, URI) (msg 2), the Authorization Server validates the request to ensure that all required parameters are present and valid against its own local storage, where these parameters have been defined during the registration process (msg 3). If the request is invalid, the protocol stops. Otherwise, the protocol continues to the next step.

Step 2: The Authorization Server sends to the Browser an authentication page (msg 4), where the user is prompted (msg 5) to provide his own credentials (msg 6 & 7) in order to authenticate with the Authorization Server. The Authorization Server verifies the received User's credentials (msg 8). If the verification process fails, then the OAuth2 protocol flow is stopped. Otherwise, the User is authenticated to the Authorization Server and the protocol continues to the next step.

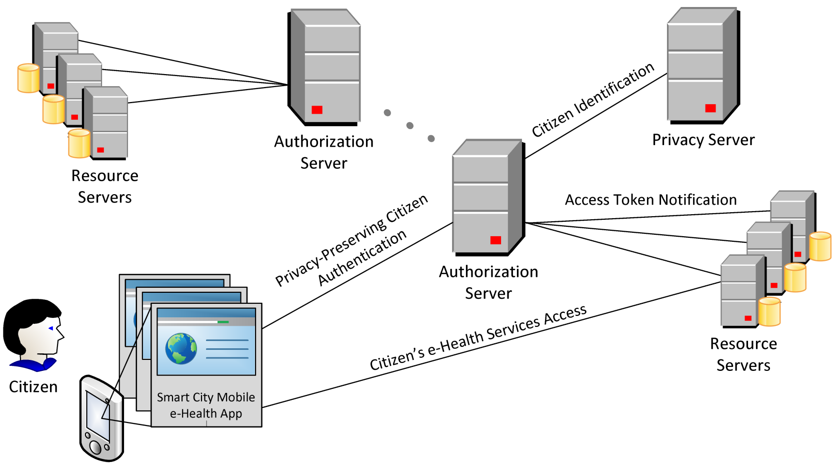

Fig. 2. System model of the proposed privacy-preserving OAuth2-based protocol for Smart City mobile e-Health apps.

Step 3: The Authorization Server requests the User to decide whether or not to grant specific permissions to the Mobile App (msg 9 & 10). The User replies to indicate his decision (msg 11 & 12). Upon receipt of the User's decision, the Authorization Server evaluates it (msg 13). If the User denies to grant the requested permissions, the protocol stops. On the other hand, if the User agrees to grant the requested permissions, the protocol continues to the next step.

Step 4: The Authorization Server redirects the Browser back to the Mobile App using the redirection URI including the Authorization Code authz_code and the state parameter s provided by the Mobile App in Step 1 (msg 14 & 15).

Step 5: The Mobile App requests an access token from the Authorization Server by sending the grant type (g=authorization_code), the Authorization Code authz_code, the redirection URI and the Mobile App identifier app_id (msg 16). Upon receipt of the ATReq(g, authz_code, URI, app_id) (msg 16), the Authorization Server authenticates the Mobile App, validates the Authorization Code, and ensures that the received redirection URI matches the URI used by the Authorization Server to redirect the Browser back to the Mobile App in Step 4. If all validations completed successfully (msg 17), the protocol continues to the next step.

Step 6: The Authorization Server responds back to the Mobile App with an access token and, optionally, a refresh token (msg 18).

However, an attacker can exploit a malicious mobile app, running on the user's mobile device, to steal user's credentials and compromise user's privacy in terms of identity and location. This threat is possible either an embedded web browser or a system native browser is used by the mobile app in the user authentication process. A malicious mobile app can steal the user's credentials from its embedded web browser by injecting JavaScript code that is able to retrieve them upon clicking the submit button and then by using a JavaScript interface to receive them [8]. On the other hand, in the case of a system native browser, a malicious mobile app can phish the user's credentials by presenting its own UI instead of allowing the system browser to render the user authentication UI [9].

III. System Model

The proposed privacy-preserving OAuth2-based protocol is designed for Smart City mobile e-Health apps to gain authenticated and authorized access to citizens' personalized e-Health services hosted on remote resource servers (i.e. SPs) located in the Smart City. In this sense, the system where our proposed protocol is applied consists of the following entities, as it is also shown in the Fig. 2:

- Citizen: A person subscribed to personalized e-Health services hosted on one or more resource servers. Citizen achieves privacy-preserving authentication, through the proposed protocol, towards the AS. Citizen is referred to as user in the rest of the paper.
- Smart City mobile e-Health app: An e-Health app running on a citizen's mobile providing the citizen with access to his/her personalized e-Health services. It gains citizen's authorization and access personalized services on behalf of the citizen.
- Resource servers: The servers hosting the protected citizen's e-Health services.
- Authorization servers (ASs): The servers providing authenticated and authorized access to the protected citizen's e-Health services hosted on the Resource Servers.
- Privacy server (PS): The server that stores the citizens' real identities and that provides the appropriate cryptographic elements so that the citizens can privately authenticate towards the ASs, which can contact the PS to obtain his/her real identity.

The proposed privacy-preserving OAuth2-based protocol, described in sec. IV, follows the OAuth2 protocol flow, described in sec. II, in which we integrate a pseudonym-based signature scheme and a delegation signature scheme. This modification only affects the information transmitted in the Step 2 of the OAuth2 protocol.

IV. Proposed OAuth2-based protocol with strong user privacy preservation

In this section we propose a novel privacy-preserving OAuth2-based protocol that achieves user authentication towards an AS without the disclosure of the user credentials (nickname and password), hence without disclosing any private information towards the browser and mobile app that could be exploited by an attacker. The proposed protocol is based on a pseudonym-based signature scheme and a delegation signature scheme in order to provide privacy-preserving user authentication between the user and the AS. These two schemes are integrated in the Step 2 of the OAuth2. However, our signature scheme does not affect the information exchanged in the messages of the other Steps of the OAuth2 protocol, as they are described in Fig. 1. Hence, our proposed OAuth2-based protocol only modifies the Step2 of the original version of OAuth2. The rest of the Steps remain as they are described in the section II.

The proposed signature scheme is an adaptation of the scheme presented in [14] (preliminaries on Elliptic Curve Cryptography can be found in this work) and our work in [15], hence it relays on the same security proof. With the proposed protocol, a user can use different mobile apps, and authenticate himself towards an AS using a different pseudonym per application, hence avoiding that any eavesdropper can link the activities of different applications to the same user. Moreover, the proposed protocol does not require the user to store any sensitive information in the mobile device.

A. Protocol initialization

The proposed protocol requires users and ASs to register in the Smarty City PS. During the registration in the PS, the user u selects nickname, $nick_u$, and a password, psw_u, which can be used to access his user account. This account includes the user real identity. For the ASs, the registration in the PS requires the AS to provide the list of users profiled in the Resource Servers hosting the personalized e-Health services. Such list of users is mapped to the users registered in the PS.

The PS is in charge of generating the public parameters. The protocol initialization is composed of three steps: i) parameter generation; ii) anonymous credential generation; and iii) static pseudonym generation.

1) Parameter generation: For protocol parameters generation the PS performs the following steps:

- The PS picks randomly a prime number p of K bits, the value of K defines the security level of the system. Then select two cyclic groups of order p containing points on the selected elliptic curve, G_1 and G_2, in which the Elliptic Curve Discrete Logarithm Problem (ECDLP) is hard [16].
- The PS selects a point in the curve P as a generator of G_1.
- The PS selects a bilinear map e such that $e : G_1 \times G_2 \rightarrow G_2$.
- The PS selects two cryptographic hash functions $H_1, H_2 : \{0,1\}^* \rightarrow G_1$ and a function $H_1 : \{0,1\}^* \rightarrow Z_p$. [2]
- The PS selects a hash function $H_3 : \{0,1\}^* \rightarrow Z_p$.
- The PS chooses at random a secret and public key pair: secret key $s \in_R Z_p$, and public key $W = sP$.
- The PS chooses a public value $Q_s \in_R G_1$ and compute a restriction key $W_s = sQ_s$.

The calculated parameters are stored in the PS and each AS, as well as a in a Privacy Application, P-App, that we consider in our proposed OAuth2-based protocol. P-App can be downloaded from the PS after user registration and installed in the users' mobile devices. The P-App is in charge of performing the cryptographic operations described in the proposed privacy-preserving OAuth2-based protocol. It is important to mention that the P-App does not store any user's private or sensitive information and does not require to be tamper resistant.

2) Anonymous credential generation: The PS, holding the public and secret values, generates an anonymous credential for each AS. To generate such credential for the AS v the PS performs the following steps:

- The PS selects a random value $\mu_v \in_R Z_p$.
- The PS computes the AS v secret key $S_v = P\frac{1}{(s+\mu_v)}$.
- The PS sends to the AS v, over a secure channel, the credential $Cre_v = (\mu_v, S_v)$. The AS v can verify the correctness of the credential by checking if $e(\mu_v P + W, S_v) = e(P, P)$ holds.

3) Static pseudonym generation: Each AS v, provided with a credential Cre_v, computes a pseudonym by using the secret value μ_v and the public value Q_s, $pseu_v = \mu_v Q_s$. This pseudonym is published by the PS, and identifies the AS as a valid server providing a privacy-preserving authentication mechanism. The user u also computes an static pseudonym constructed with the public value Q_s and his nickname and password: $pseu_u = \mu_u Q_s$, where $\mu_u = H_3(nick_u || psw_u)$. Note that the operation $||$ denotes concatenation. The user's static pseudonym is stored in the P-App, and although this pseudonym is never transmitted for user authentication, its leakage does not provide any advantage to perform user impersonation.

B. Privacy-preserving user authentication

In the proposed privacy-preserving OAuth2-based protocol, the Step 2 maintains the same interactive procedure, as it is described in the sec. II, but integrates information related to the pseudonym-based signature scheme in the exchanged messages. As a result the exchanged messages of the Step 2 in our protocol are structured with the following way:

UserAuthRequest1 The AS v sends an authentication request code to the user u, through the browser, composed of a pseudonym and a signature. To construct this message the AS v, with a public pseudonym $pseu_v$, and an anonymous credential (μ_v, S_v), performs the following steps:

- The AS v converts the app_id into an integer value and calculates the specific client application value $Q_i = H_1(app\_id)$.

[2] As it is explained in [16], it is sufficient with a function $H : \{0,1\}^* \rightarrow A$, for a given set A, and an admissible encoding function $L : A \rightarrow G_1$

- The AS v generates an app-specific pseudonym $pseu_{v,i} = \mu_v Q_i$ and a nonce rn.
- The AS v signs the pseudonym $pseu_{v,i}$ and the nonce rn with its public pseudonym $pseu_v$, by performing the message signing algorithm.
- The AS v sends an authentication request code composed of the message $M = rn||pseu_{v,i}$ and a signature $sig_{pseu_v}(M)$, to the user u.

For the AS v to sign the message $M = rn||pseu_{v,i}$ (a concatenation of the nonce and the app-specific pseudonym) with its public pseudonym $pseu_v$ and obtain $sig_{pseu_v}(M)$, the AS performs the message signing algorithm which has the following steps:

- The AS v picks randomly α, r, $r' \in_R Z_p$.
- The AS v computes $T = \alpha S_v$, $R_{G_1} = rQ_i$ and $R = e(Q_i, P)^{r'}$.
- The AS v computes $c = H_2(M||T||R_{G_1}||R||pseu_v)$, where the operator $||$ represents concatenation.
- The AS v calculates $z_1 = c\alpha + r'$ and $z_2 = c\mu_v + r$.
- The signature of the value M with the pseudonym $pseu_v$ is composed by the tuple $sig_{pseu_v}(M) = (T, c, z_1, z_2)$.

UserAuthRequest2 The browser receives the authentication request code, opens the P-App, and passes the code to the P-App. The P-App validates the signature $sig_{pseu_v} = (M)$ by using the signature verification algorithm:

- The P-App computes $R'_{G_1} = z_2 Q_s - c pseu_v$ and $R' = e(Q_s, P)^{z_1} / e(pseu_v + W_s, T)^c$.
- The P-App calculates $c' = H_2(M||T||R'_{G_1}||R'||pseu_v)$.
- The P-App validates the signature if the equality $c' = c$ holds.

Note that the verification algorithm requires the public values Q_s and W_s, which are stored in the P-App. After validation, the P-App prompts a user interface for the user to insert his nickname and password.

UserAuthReponse1() The user u types the nickname, $nick_u$ and password, psw_u. The P-App generates the value $pseu_u = \mu_u Q_s$, where $\mu_u = H_3(nick_u||psw_u)$, and checks that it matches with the stored user static pseudonym $pseu_u$. If the P-App gets a positive match, then the P-App is certain that the user u introduced a valid nickname and password and performs the following steps:

- The P-App computes the app-specific value $Q_i = H_1(app\_id)$.
- The P-App generates an app-specific pseudonym $pseu_{u,i} = \mu_u Q_i$. The pseudonym is app-specific because the user u will generate different pseudonyms for other apps, but it will always generate the same pseudonym for the same app. It is also important to mention that this pseudonym will be different than that of other users of the same app.
- The P-App uses the $pseu_{u,i}$ to sign a warrant w_u with a delegation signing algorithm by performing:
    - The P-App creates a warrant, w_u, which includes the AS v app-specific pseudonym $pseu_{v,i}$, the nonce rn, and the right delegation scope, i.e. time of validity.
    - The P-App signs the warrant by computing $sig'_{pseu_{u,i}}(w_u) = \mu_u H_1(w_u)$.
    - The P-App constructs the tuple $Del_{u,v} = (pseu_{u,i}, w_u, sig'_{pseu_{u,i}}(w_u))$.
- The P-App passes the value $Del_{u,v}$ to the browser as an authentication response code. This code is forwarded through the browser to the AS v. Note that, the browser or the app cannot learn any private information from the response code. Moreover, the response code cannot be reused in future occasions.

UserAuthResponse2() The AS v receives the authentication response code, $Del_{u,v}$, and makes a delegated signature by performing the following steps.

- The AS v creates a request for the PS, a string R, asking for the user u real identity.
- The AS v, holding the tuple $Del_{u,v}$ from the user u, signs the request R with his pseudonym $pseu_{v,i}$, in combination with the signature received from u, by computing:

$$\begin{gathered} Delsig'_{pseu_v, pseu_u}(R, w_u) = \\ sig'_{pseu_u}(w_u) + \mu_v H_2(R||w_u). \end{gathered} \tag{1}$$

UserAuthentication() In this step, the AS v contacts the PS in order to identify the user u, i.e. to obtain its real identity. The AS v transmits a message with the request, the combined signature, and the server v signature $sig_{pseu_v}(M)$ to the PS. The message includes the values: R, w_u, $Delsig'_{pseu_{v,i}, pseu_{u,i}}(R, w_u)$, $pseu_{u,i}$, $pseu_{v,i}$, $pseu_v$, $sig_{pseu_v}(M)$, app_id, rn.

The PS receives the message from the AS v. and performs the following steps:

- The PS checks whether the pseudonym $pseu_v$ is included in its list of valid servers and validates the signature $sig_{pseu_v}(M)$, where $M = rn||pseu_{v,i}$, using the signature verification algorithm described above.
- The PS computes the app-specific value $Q_i = H_1(app\_id)$
- The PS validates the delegated signature, $Delsig'_{pseu_{v,i}, pseu_{u,i}}(R, w_u)$, using the Q_i, with the delegation verification algorithm, which consists of checking that the following equation holds:

$$\begin{gathered} e(Delsig'_{pseu_{v,i}, pseu_{u,i}}(R, w_u), Q_i) \\ = e(H_1(w_u), pseu_{u,i}) e(H_2(R||w_u), pseu_{v,i}) \end{gathered} \tag{2}$$

- The PS takes the value Q_i and and computes the app-specific pseudonyms for the list of users registered with the AS v, i.e. it computes $pseu_{=}\mu_u Q_i$ for all users in the list of users of the AS v. Then, the PS looks up the user with pseudonym $pseu_{u,i}$ and retrieves his real identity. Note that the PS can store the app-specific pseudonyms, in the AS v users' profile, for future pseudonym searches.
- The PS sends back, to the AS v, the real identity of the users if this does not violate any of the constraints specified in the signed user's warrant w_u.

V. Security analysis

In this section we provide the correctness proof of the pseudonym-based signature scheme and the delegation scheme. We also provide an analysis of the proposed protocol in terms of location privacy and conditional privacy.

A. Correctness of the signature and delegation schemes

It is worth mentioning that the correctness of the signature scheme is provided in [14], hence we do not include it here due to space constraints. However, we provide the correctness of the delegation signature scheme, in which the user u and AS v compute a combined signature, which is validated by the AS and the PS. This delegation signature scheme was first presented in [17]. Such signature, $Delsig'_{pseu_v,pseu_u}(R, w_u)$, is validated if the following equation holds:

$$\begin{aligned} &e(Delsig'_{pseu_{v,i},pseu_{u,i}}(R, w_u), Q_i) \\ &= e(H_1(w_u), pseu_{u,i})e(H_2(R||w_u), pseu_{v,i}) \end{aligned} \tag{3}$$

Note that according the properties of the Bilinear map e, [16], we can state that:

$$\begin{aligned} e(H_1(w_u), pseu_{u,i})e(H_2(R||w_u), pseu_{v,i}) &= \\ e(H_1(w_u), Q_i)^{\mu_u} e(H_2(R||w_u), Q_i)^{\mu_v} &= \\ e(H_1(w_u)\mu_u, Q_i)e(H_2(R||w_u)\mu_v, Q_i) &= \\ e(H_1(w_u)\mu_u + H_2(R||w_u)\mu_v, Q_i) &= \\ e(Delsig'_{pseu_{v,i},pseu_{u,i}}(R, w_u), Q_i) & \end{aligned} \tag{4}$$

B. Identity privacy

To achieve identity privacy, only the PS should be able to recover the real identity linked to a given pseudonym, whereas any other entity, apart from the citizen holding the pseudonym and the PS, should not be able to obtain such relationship. According to the proposed protocol, the citizen's pseudonyms are computed as $pseu_{u,i} = \mu_u Q_i$ and $pseu_u = \mu_u Q_s$. Only the PS and the citizen hold the secret value μ_u, constructed with the user's nickname and password. Hence, knowing or calculating the public values Q_i and Q_s, the PS can always compute the pseudonym of the citizen and retrieve his/her real identity. On the other hand, any third-party, without the knowledge of the μ_u value, is not able to compute the pseudonym of a citizen. Moreover, the knowledge of the pseudonym $pseu_u$ or $pseu_{u,i}$, by an attacker, does not provide any advantage to obtain the μ_u value, under the assumption that the ECDLP is hard.

C. Location privacy

As indicated in [18], location privacy can be achieved by granting a large number of pseudonyms and providing effective mechanisms to change the active pseudonym. In the proposed scheme a citizen is given a set of pseudonyms, one app-specific pseudonym per mobile e-Health app. In addition, these app-specific pseudonyms, $pseu_{u,i}$, cannot be linked to each other since they are seen as random sequence strings by eavesdroppers. This prevents eavesdroppers from linking different mobile e-Health apps running on a device to the location of this specific device, which would facilitate the the citizen tracking, and hence allowing eavesdroppers to collect valuable information to infer the citizen's identity.

VI. conclusion

In this paper we have proposed an OAuth2-based protocol for Smart City mobile e-Health apps. The proposed protocol addresses the citizen's identity and location privacy issues of citizens accessing personalized e-Health services through their mobile e-Health apps running on their mobile devices. Our proposed protocol follows the OAuth2 protocol flow, but it provides privacy-preserving user authentication by integrating a pseudonym-based signature scheme and a delegation signature scheme. This modification only affects the information transmitted in the Step 2 of the OAuth2 protocol. Moreover, the proposed protocol does not require the storage of any sensitive information, e.g. secret keys or credentials, in the citizen's mobile device, hence avoiding attackers to steal information that could be used to reveal the citizen's real identity.

References

[1] ITU. An overview of smart sustainable cities and the role of information and communication technologies. 2014.

[2] Solanas et al. Smart health: A context-aware health paradigm within smart cities. *Communications Magazine, IEEE*, 52(8):74–81, Aug 2014.

[3] M. Fengou, G. Mantas, D. Lymberopoulos, N. Komninos, S. Fengos, and N. Lazarou. A new framework architecture for next generation e-health services. *Biomedical and Health Informatics, IEEE Journal of*, 17(1):9–18, Jan 2013.

[4] Antoni Martinez-Balleste, Pablo Perez-Martinez, and Agusti Solanas. The pursuit of citizens' privacy: A privacy-aware smart city is possible. *IEEE Communications Magazine*, 51:136–141, 2013.

[5] Eric Y. Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. Oauth demystified for mobile application developers. In *2014 ACM SIGSAC*, CCS '14. ACM, 2014.

[6] San-Tsai Sun and Konstantin Beznosov. The devil is in the (implementation) details: An empirical analysis of oauth sso systems. Aug 2012.

[7] E. Hammer-Lahav. The oauth 1.0 protocol. In *IETF*, 2010.

[8] M. Shehab and F. Mohsen. Towards enhancing the security of oauth implementations in smart phones. In *MS, 2014 IEEE*, June.

[9] M. M. T. Lodderstedt and P. Hunt. Oauth 2.0 threat model and security considerations. In *IETF2013*, 2013.

[10] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A survey on security for mobile devices. *IEEE Communications Surveys and Tutorials*, 15(1):446–471, 2013.

[11] Mantas G, Komninos N, Rodriguez J, Logota E, and Marques H. Security for 5g communications. In *Fundamentals of 5G Mobile Networks*. John Wiley & Sons, 2015.

[12] D. Hardt. The oauth 2.0 authorization framework. In *IETF2012*, 2012.

[13] G. Developers. Using oauth 2.0 for installed applications. In *Available: https://developers.google.com/identity/protocols/OAuth2InstalledApp*, 2015.

[14] Yong Zhang and Jun-Liang Chen. A delegation solution for universal identity management in soa. *Services Computing, IEEE Transactions on*, 4(1):70–81, Jan 2011.

[15] V. Sucasas, F. Saghezchi, A. Radwan, H. Marques, J. Rodriguez, S. Vahid, and R. Tafazolli. Efficient privacy preserving security protocol for vanets with sparse infrastructure deployment. In *ICC, IEEE*, 2015.

[16] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. *SIAM J. of Computing*, 32(3):586–615, 2003. extended abstract in Crypto'01.

[17] Huang et al. A short proxy signature scheme: Efficient authentication in the ubiquitous world. In Tomoya Enokido, editor, *EUC Workshops*, volume 3823, pages 480–489. Springer, 2005.

[18] Rongxing et al. A lightweight conditional privacy-preservation protocol for vehicular traffic-monitoring systems. *Intelligent Systems, IEEE*, 28(3):62–65, May 2013.