Modular Security Analysis of OAuth 2.0 in the Three-Party Setting

Xinyu Li[1,2,3], Jing Xu[1,3], Zhenfeng Zhang[1,3], Xiao Lan[4], Yuchen Wang[1,3]
[1]*TCA Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing, China*
[2]*State Key Laboratory of Cryptology, Beijing, China*
[3]*University of Chinese Academy of Sciences, Beijing, China*
[4]*Cybersecurity Research Institute, Sichuan University, Chengdu, China*
Email: xinyuli1920@gmail.com, xujing@iscas.ac.cn, {zfzhang, lanxiao, wangyuchen}@tca.iscas.ac.cn

***Abstract*—OAuth 2.0 is one of the most widely used Internet protocols for authorization/single sign-on (SSO) and is also the foundation of the new SSO protocol OpenID Connect. Due to its complexity and its flexibility, it is difficult to comprehensively analyze the security of the OAuth 2.0 standard, yet it is critical to obtain practical security guarantees for OAuth 2.0.**

In this paper, we present the first computationally sound security analysis of OAuth 2.0. First, we introduce a new primitive, the three-party authenticated secret distribution (3P-ASD for short) protocol, which plays the role of issuing the secret and captures the token issue process of OAuth 2.0. As far as we know, this is the first attempt to formally abstract the authorization technology into a general primitive and then define its security. Then, we present a sufficiently rich three-party security model for OAuth protocols, covering all kinds of authorization flows, providing reasonably strong security guarantees and moreover capturing various web features. To confirm the soundness of our model, we also identify the known attacks against OAuth 2.0 in the model. Furthermore, we prove that two main modes of OAuth 2.0 can achieve our desired security by abstracting the token issue process into a 3P-ASD protocol. Our analysis is not only modular which can reflect the compositional nature of OAuth 2.0, but also fine-grained which can evaluate how the intermediate parameters affect the final security of OAuth 2.0.

***Index Terms*—cryptographic protocols, provable security, security model, OAuth, authorization**

1. Introduction

OAuth 2.0 protocol [1] is commonly used as a way for Internet users to grant websites or applications access to their information on other websites without revealing the users' access credentials such as passwords. For instance, the user of a photo-sharing website supporting OAuth can grant a third-party photo printing service access to her permitted photographs, without sharing her long-term credential at the photo-sharing site with the printing site. In practice, OAuth is also used for authentication and serves as the foundation for the new single sign-on (SSO) standard OpenID Connect [2]. Specifically, a user can log in at a third-party website utilizing her identity managed by a service provider. Currently, OAuth 2.0 is being adopted by numerous major companies such as Google, Facebook, Microsoft, Twitter and LinkedIn, and moreover has become the *de facto* authorization and authentication protocol in mobile applications [3].

Due to its wide usage for authorization and authentication in practice, numerous studies assessing the security of OAuth 2.0 have been published and several practical attacks have been discovered. Li et al. [4] revealed two critical impersonation vulnerabilities in many OAuth 2.0 implementations. Chen et al. [3] performed practical evaluations on the security of OAuth implementations of mobile applications and found 59.7% of those applications were vulnerable. Shernan et al. [5] found 25% of websites using OAuth 2.0 in their evaluations vulnerable to the Cross-site Request Forgery (CSRF) attacks [6]. In addition, several unknown attacks [7] [8] on OAuth 2.0 were demonstrated using formal analysis methods. Fortunately, the improved security recommendations for OAuth 2.0 have been proposed and listed in the standard [1] or security considerations [9]. These attacks stress the need for provable security analysis of OAuth protocols.

So far, almost all analysis efforts [7] [8] [10] [11] [12] regarding the security of OAuth were the formal analysis in specific implementations based on the Dolev-Yao model [13] in an automated or even manual way. However, computationally sound security analysis is another attractive option both in practice and theory, and is gradually being required by the standards bodies, e.g., the analysis on TLS [14]–[16], EMV [17], QUIC [18] [19] and EAP [20]. By the game-based approach in the computational model, we can explore the security relations between the target protocol and the employed primitives, and a modular analysis can help to explore the exact security properties that sub-protocols should satisfy to guarantee the final security goals. Therefore, carrying out a modular security analysis for OAuth 2.0 in the game based model makes analysis easier and less error-prone, which is valuable and important for protocol analysts and designers.

1.1. Our Contributions

In this work, we aim to provide a *systematic* and *computationally sound* security analysis in the computational setting on the different modes of OAuth 2.0. However, this is a very complex and challenging task. First, due to the complexity of the three-party interactions and trust

Jing Xu is the corresponding author.

DOI 10.1109/EuroSP48549.2020.00025

relationships in OAuth, it is not easy to model the security for three-party protocols especially for each party with different roles. Second, OAuth is not a "standalone" protocol in isolation, but rather depends on other protocols such as TLS [21], and thus it is much harder to evaluate the exact security of OAuth. Finally, the OAuth 2.0 standard [1] only defines a general framework and leaves many details unspecified. For instance, the authentication mode used for authenticating the client by the server as well as the channel for secure transmission is not specified. That means that solely using the security claims from the OAuth 2.0 standard document [1] is not sufficient to analyze OAuth accurately, and we will have to make some assumptions on these specifics. More specifically, our technical contributions are threefold.

Three-party authenticated secret distribution protocols. We introduce a new primitive, referred to as three-party authenticated secret distribution (3P-ASD for short) protocol, which serves as a basis for OAuth protocols and plays the role of "issuing the secret". In a 3P-ASD protocol, a client can obtain a fresh secret from the server only with the permission of an honest user, which means the user authorizes the server to issue the secret for the client. Although the authorization technology is widely used in practice especially in the cloud computing environment, surprisingly, as far as we know our work is the first attempt to formally abstract the technology into a general primitive and then define its security.

Stimulated by the work [22] on the compositional security of two separated parts—key exchange protocol and the protocol requiring symmetrically distributed keys, in the 3P-ASD primitive, we only concentrate on how to issue a secret securely under the permission mechanism, without imposing any restriction on the usage of the resulting secret. Regarding how to use it may be the main task of the upper-layer protocols considering the 3P-ASD protocol as a core subroutine. For instance, in an OAuth protocol (an intuitive application of the 3P-ASD primitive), it is by the use of the secret that the client can get access to the user resource stored at the trusted server, and moreover the secret has some additional restrictions on the duration and access scope. We believe that the separation of "how to obtain" from "how to use" plays an important role not only in modular security analysis but also in protocol design. As a new primitive, 3P-ASD may be of independent value and can have many other applications.

We then present the basic security definitional framework for 3P-ASD protocols, particularly, achieving secure entity authentication, confidentiality of the fresh secret, and session integrity are the main security goals. Note that here the entity authentication goal captures the notion of "permission", that is, the client cannot obtain the secret of the user at the server without the user's permission since the adversary cannot impersonate an honest user to the server.

Notice that our model can capture the so-called "web attack" as in [8], which is one of the main attack vectors against OAuth-like web based protocols where the adversary can exploit features of the web platform to interfere with messages transmitted through the web browser. To achieve this goal, the adversary is additionally provided with two abilities in the model. Specifically, the adversary can maliciously send messages via an honest browser to another website, and can also obtain some private information from the HTTP referrer header contained in the request of the user's browser.

Security model for OAuth protocols in three-party settings. Our goal is to define a sufficiently rich three-party security model for OAuth protocols, covering all kinds of authorization flows (different modes), providing reasonably strong security guarantees, and capturing various web features. Different from 3P-ASD protocols whose main goal is how to issue secrets securely, the security definition for OAuth protocols should capture how to use secrets securely.

It is difficult to propose a comprehensive security model for OAuth 2.0. The main reason is that many details are unspecified in the standard document [1]. Therefore, in our security model, the general execution environment and adversarial capabilities are considered instead of the local protocol details, which makes it feasible to accomplish the overall analysis of OAuth 2.0. It is also difficult to define the goal of authentication and authorization because of each party with different roles. Our solution is that an entry is kept by the server and the client recording each issued secret and its associated party. By this way, more specifically, by checking whether the secret is issued to the specified party, the security property can be modelled.

Our model for OAuth can capture various web features, which is more suitable for OAuth-like web-based protocols and makes the positive security results more meaningful. Moreover, several attacks against OAuth 2.0, including 307 redirect attack, IdP mix-up attack and state leak attack [8], can be identified in our model. Particularly, we explain with our model why OAuth without countermeasures is vulnerable, which furthermore confirms the soundness of our model.

In addition, by the game-based approach, we can define general models for a large class of protocols instead of an individual OAuth instance. The Bellare-Rogaway model [23] lays the foundation of the analysis of various (authenticated) key exchange protocols. Similarly, the general OAuth security model we introduce in this paper can cover all main modes of OAuth 2.0 and a class of web-based protocols like OAuth 1.0 [24] and OpenID Connect [2].

Modular provable security analysis of OAuth 2.0 protocols. Stimulated by the modular security analysis work [20] [22] [25], we abstract the core module of the OAuth 2.0 protocols, the process of token issue, which can be regarded as an instantiation of the 3P-ASD protocol (See Figure 2 and Figure 3). Each mode of OAuth 2.0 is then fully defined via a specific instantiation of the 3P-ASD protocol. Since any of the intermediate or final secrets generated during token issue can affect the final security of OAuth 2.0, we have to be very careful during the instantiation process.

We also prove the security of OAuth 2.0 protocols with countermeasures via a modular approach. In particular, we show if the token issue process of OAuth 2.0 is 3P-ASD secure and the channel establishment protocol is ACCE secure, then the composed OAuth 2.0 protocol is secure. Furthermore, we prove the token issue process of both authorization code grant and implicit grant of OAuth 2.0

are 3P-ASD secure, and thus both modes of OAuth 2.0 satisfy the security goals we define.

Note that the resource owner password credentials grant and client credential grant in OAuth 2.0 are not considered in this paper, not only due to their rare use in practice as mentioned in [3], but also the fact that both modes can be considered as special two-party cases and have relatively simple proof.

In order to provide more options in practice, the OAuth 2.0 standard [1] only define a general framework, and some details such as authentication modes and channel establishment protocols are unspecified. In our analysis of OAuth 2.0 protocols, both the user authentication and the client authentication (to the server) are abstracted into a universal authentication scheme, which means that any authentication method satisfying the security requirements is supported, such as password based authentication, public key based authentication, or two-factor authentication. Similarly, any secure protocol for channel establishment is also generalized into an authenticated and confidential channel establishment (ACCE) protocol. Therefore, our analysis captures the generality and scalability property of the OAuth 2.0 framework.

Different from all existing analyses which regard OAuth as a whole and only judge whether OAuth satisfies the pre-defined goals, our approach is fine-grained and can exactly evaluate how specific parameters (generated during 3P-ASD sub-protocol) affect the final security, by separating the process "how to issue secrets securely" from the process "how to use secrets securely". Rather than just for a modular analysis, the treatment itself in this paper would undoubtedly be helpful to comprehensively understand the security of OAuth and also instructive to the design and analysis of other OAuth-like authorization protocols.

It is also worth emphasizing that the extendibility is one of the significant advantages of the 3P-ASD primitive, although OAuth is just an intuitive application of the 3P-ASD primitive where the usage of the issued secrets is very straightforward. Recall that the composability result in [22] can apply to not only the straightforward scenarios (e.g., using the symmetric keys to build a secure channel), but also more complicated applications (e.g., using the symmetric keys as the basis for a new protocol like TLS session resumption [25]). Similarly, the 3P-ASD primitive, which does not make any restriction on the usage of the issued secret, can also be applied to other more elaborated applications.

1.2. Related Work

Security analysis of OAuth 2.0. In view of the wide application of OAuth 2.0 in practice, formal analysis and comprehensive evaluation are necessary for guaranteeing its security, however, as mentioned above many analysis efforts such as [3]–[5], [7] regarding the security of OAuth 2.0 were targeted towards finding errors in specific implementations.

Pai et al. [10] analyzed the security of OAuth 2.0 based on the Alloy model checker and found a previously known security weakness in [9]. In [7], Bansal et al. analyzed the security of OAuth 2.0 using the WebSpi library and the ProVerif analysis tool, and discovered two unknown vulnerabilities (i.e., Covert Redirect and Social CSRF attack). Later, Fett et al. [26] proposed a manually driven and more expressive model (also called FKS model) for the web infrastructure. In 2018, Jayasri et al. [12] used the formal tool UPPAAL to carry out a formal verification of safety, liveness and absence of deadlock properties of the authentication code mode in OAuth 2.0.

Based on the slightly extended version of the FKS model [26], Fett et al. [8] carried out the first extensive formal analysis of the OAuth 2.0 standard, revealed four unknown attacks, and furthermore proved the security of OAuth 2.0 fixed with the countermeasures in [8]. Although based on the Dolev-Yao model for automated analysis, their analysis is manually driven since the underlying FKS model is not directly suitable for automation [26]. In addition, their analysis cannot be directly applied to the analysis of OpenID Connect, since OpenID Connect adds specific details on top of OAuth. Later, Fett et al. [27] carried out the formal analysis of OpenID Connect under the FKS model from scratch. Recently, Fett et al. [28] performed an extensive formal security analysis of the OpenID Financial-grade API under the same model. Indeed, our modular approach makes the security analysis of OpenID Connect easier. Even though separate analysis of OpenID Connect is still needed, it is just enough to prove the token issue process satisfying 3P-ASD security, due to the fact that the security goals of OpenID Connect is consistent with OAuth. Specifically, the token issue process of OpenID Connect corresponds to that of OAuth 2.0, however, OpenID Connect introduces many new concepts and defines an additional mode. In addition, both OpenID Connect and OAuth 2.0 are requested to guarantee the properties of (integrity for) authorization and (integrity for) authentication according to [27]. Then according to Theorem 1, if we prove the token issue process is 3P-ASD secure, the security properties of OpenID Connect can be achieved. Different from symbolic analysis in [8] and [27], our analysis of OAuth and application to OpenID Connect is also valuable.

All the above security analyses of OAuth and OpenID Connect are based on the Dolev-Yao symbolic model [13]. In practice, both the symbolic approach and the computational approach are widely used for the security analysis of cryptographic protocols. In the symbolic model, cryptographic primitives are represented by symbolic values and often idealized, the adversary is only restricted to use the pre-defined primitives for computation, and the proof is based on logical or algebraic reasoning. While in the computational model, the cryptographic primitives are functions applied to bitstrings, the adversary is any probabilistic Turing machine, and the proof is a reduction from the protocol to the underlying primitives. Therefore, the symbolic model is more suitable for automated tools of proofs to reduce the risk of errors in manual proofs, whereas the computational model considers more realistic adversaries and reduction-based proofs can help to choose security parameters and explore the security relation between the target protocol and the underlying primitives. More discussions can be referred to [29] [30] [31]. In addition, the modular computational approach in this work further helps us to deeply understand the security strength and design subtlety of OAuth 2.0 and other related protocols.

As far as we know, the only attempt to use computational reduction based provable security in the context of securing OAuth has, up until now, been done by Chari et al. [32] in the universal composability (UC) security framework [33] using a simulation based argument. Nevertheless, their analysis is not enough to ensure the practical security of OAuth. Specifically, the ideal functionalities are strictly weaker than the security guarantees we expect from OAuth in practice and only the security of the access token in OAuth is modelled, which means that some real attacks cannot be covered. For example, in the Mix-up attack [8] against OAuth 2.0, the adversary can compromise the intermediate parameter (namely the *code* we will introduce later) during the token issue process and then break the authentication security of OAuth 2.0 even without the access token. Therefore, the OAuth protocol proven secure in their security model is still vulnerable to the Mix-up attack. Moreover, their analysis only concentrates on the access token issue process of the OAuth 2.0 in the authorization code grant type, not considering the full OAuth 2.0 or other grant types, which is incomplete. Both UC and the game-based approaches are widely used in practice for provable security, however, whether the analysis of OAuth 2.0 by UC framework can also achieve the above advantages is unclear or at least not very intuitive, which may be another interesting work in the future.

Computational analysis of other three-party protocols. In 2019, Schwenk et al. [29] proposed a rich three-party authentication model for Kerberos-like authentication protocols based on a general match predicate instantiated with the matching conversations, and gave the first computational analysis of the unmodified Kerberos protocol. Other work like [20] [34] employed different ways to define partners according to protocol specifics. As an extension, in the three-party AKE protocols an additional session key is established and thus key indistinguishability or ACCE security is required as specified in [20] [29] [34]. By extending the ACCE model, Bhargavan et al. [35] proposed the first security definition for proxied TLS, where the client and server intend to agree on the read/write privileges of the proxy during the TLS process to avoid uncontrolled interceptions. Inspired by but different from these existing models for three-party authentication or authenticated keys exchange (AKE) protocols, our model is designed for authorization-based protocols including 3P-ASD and OAuth. In other words, their functionalities and goals are different, and thus it is almost impossible to use the existing models to analyze OAuth-like protocols.

2. Preliminaries and Definitions

In this section, we briefly recall some primitives and definitions that our analysis employs.

2.1. Matching Conversations

The concept of matching conversation [16] [23] is used for entity authentication in AKE protocols by ensuring honest parties cannot be forged and messages indeed come from the party it claims to be, and we would use it for the security definition of ACCE and 3P-ASD protocols. Here we employ the modified version in [16] taking into account which party sends the last message.

Definition 1. (Matching Conversation [16]) Let transcripts $T_{A,s}$ and $T_{B,t}$ be two sequences of messages sent and received in chronological order, by session oracles π_A^s and π_B^t, respectively. We say that π_A^s has a matching conversations to π_B^t if (1) $T_{B,t}$ is a prefix of $T_{A,s}$ and π_A^s has sent the last message, or (2) $T_{A,s} = T_{B,t}$ and π_B^t has sent the last message.

2.2. ACCE Protocols

Authenticated and confidential channel establishment (ACCE) protocol [16] is a protocol executed between two parties, which combines an ordinary two-party AKE protocol with a stateful authenticated encryption (sAE) scheme $\mathsf{StE} = (\mathsf{StE.Init}, \mathsf{StE.Enc}, \mathsf{StE.Dec})$ (following [16]). Specifically, an AKE protocol is firstly executed to achieve mutual authentication and establish a session key K, and then the transmitted data can be encrypted and authenticated using an sAE scheme which is keyed by the session key K. TLS 1.2 is one of the prime examples for ACCE protocols.

Recall that in AKE protocols key indistinguishability is required, which however cannot be provided by protocols like TLS 1.2 due to the interleaving of the key exchange phase and the message encryption phase [16]. A secure ACCE protocol aims at establishing a secure communication channel in the sense of sAE, which requires that (i) entity authentication, any session reaches acceptance only when it has a unique matching conversation; and (ii) channel security, all data is transmitted over an authenticated and confidential channel. The channel security captures both integrity and privacy, and to distinguish these two properties in the proof, we define them separately as in [20]. The security requirements for ACCE protocols capture exactly the properties expected from protocols like TLS 1.2 in practice.

Similar to the security model of AKE protocols, the adversary in ACCE protocols can interact with the session oracle π_U^i by issuing the Send, Reveal and Corrupt queries to forward messages, learn the session keys and learn the long-term secret key respectively, where π_U^i models party U executing the i-the session of an ACCE instance. More details can be referred to [16]. In addition, queries Encrypt (for encryption) and Decrypt (for decryption) are added to allow the adversary to interact with the established channel as depicted in Figure 1.

The goal of the adversary is to break the entity authentication, channel integrity or privacy of a fresh session oracle π_U^i, where π_U^i is considered to be fresh if U and its peer are not corrupted, and moreover π_U^i and its matching oracle π_V^j (if exists) are not revealed.

Definition 2. (Entity Authentication) An adversary $\mathcal{A}$ breaks the entity authentication security with the advantage $\mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}$ if there exists an fresh oracle π_U^i that accepts without a matching conversation.

An ACCE protocol CHAN provides entity authentication, if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}$ is negligible.

Definition 3. (Channel Integrity) An adversary $\mathcal{A}$ breaks the channel integrity security with the advantage

Encrypt($\pi_U^i, m_0, m_1, len, H$):	Decrypt(π_U^i, C, H):
(1) If π_U^i has not reached acceptance or $\|m_0\| \neq \|m_1\|$, return $\perp$	(1) If π_U^i has not reached acceptance, return $\perp$
(2) $u = u+1$	(2) $v = v+1$
(3) $(C^0, st_e^0) \leftarrow$ StE.Enc(K, len, H, m_0, st_e)	(3) $(m, st_d) \leftarrow$ StE.Dec(K, H, C, st_d)
(4) $(C^1, st_e^1) \leftarrow$ StE.Enc(K, len, H, m_1, st_e)	(4) If $v > u$ or $C \neq \vec{C}[v]$, then $flag = 1$
(5) $(\vec{C}[u], st_e) = (C^b, st_e^b)$	(5) If $flag = 1$, return m
(6) Return $\vec{C}[u]$	(6) Return $\perp$

Figure 1: Encrypt and Decrypt oracles. m_0 and m_1 are two messages, len is the length parameter and H is the header data. $b \leftarrow \{0,1\}$ is randomly sampled by π_U^i. The list $\vec{C}$ is initialized to $\emptyset$. The counters u and v as well as the variable $flag$ are initialized to 0. st_e and st_d denote the states for encryption and decryption respectively, and are initialized by the initialization algorithm StE.Init of StE. StE.Enc and StE.Dec denote the encryption and decryption algorithm of StE respectively.

$\mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}$ if $\mathcal{A}$ makes a Decrypt query for a fresh session π_U^i and obtains something other than $\perp$.

An ACCE protocol CHAN provides channel integrity, if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}$ is negligible.

Definition 4. (Channel Privacy) An adversary $\mathcal{A}$ breaks the channel privacy security with the advantage $\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}$ if it terminates with the output (π_U^i, b'), such that π_U^i is fresh and $\pi_U^i.b = b'$, where $\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}} = |\Pr[\pi_U^i.b = b'] - \frac{1}{2}|$.

An ACCE protocol CHAN provides channel privacy, if for all PPT adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}$ is negligible.

Definition 5. (ACCE Security) We say that a protocol Π is ACCE-secure, if Π satisfies entity authentication, channel integrity, and channel privacy.

Note that, if only the server S is authenticated by default in many ACCE protocols like TLS, then only the ACCE security at the side of client C can be guaranteed since the adversary can trivially impersonate the unauthenticated C. Specifically, C only accepts with unique matching conversation, and the channel at C's side satisfies integrity and privacy properties.

2.3. Universal Authentication Schemes

A universal authentication scheme auth consists of three PPT algorithms (Gen, Auth, Ver) such that:

(1) The key-generation algorithm Gen takes as input the security parameter 1^λ, and outputs an authentication/verification key pair (sk, vk).
(2) The authentication credential generation algorithm Auth takes as input the authentication key sk and a message $m \in \{0,1\}^*$, and outputs a credential $cre \leftarrow \mathsf{Auth}_{sk}(m)$.
(3) The deterministic verification algorithm Ver takes as input a verification key vk, a message m, and a credential cre. It outputs a bit b, with $b = 1$ meaning valid and $b = 0$ invalid. We write this as $b = \mathsf{Ver}_{vk}(m, cre)$.

It is required that for any key k output by $\mathsf{Gen}(1^\lambda)$, and any $m \in \{0,1\}^*$, it holds that $\mathsf{Ver}_k(m, \mathsf{Auth}_{sk}(m)) = 1$.

To define the security of a universal authentication scheme auth = (Gen, Auth, Ver), we consider the following security experiment $\mathsf{Exp}^{\mathsf{SUF-CMA}}_{\mathsf{auth},\mathcal{A}}$ between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

(1) $\mathcal{C}$ runs $\mathsf{Gen}(1^\lambda)$ to generate a key pair (sk, vk).
(2) The adversary $\mathcal{A}$ is given an oracle access to $\mathsf{Auth}_{sk}(\cdot)$.
(3) Eventually $\mathcal{A}$ outputs a pair (m^*, cre).
(4) $\mathcal{A}$ succeeds and the experiment outputs 1 if and only if $\mathsf{Ver}_{vk}(m^*, cre) = 1$ and cre was not outputted by $\mathsf{Auth}_{sk}(m^*)$ before.

Definition 6. (SUF-CMA) We say that a universal authentication scheme auth is secure against *strong forgeries under adaptive chosen-message attacks* (SUF-CMA), if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth},\mathcal{A}} = \Pr[\mathsf{Exp}^{\mathsf{SUF-CMA}}_{\mathsf{auth},\mathcal{A}} = 1]$ is negligible.

Note that the above definition covers both symmetric-key primitive (e.g. MAC schemes) by letting $sk = vk$ and asymmetric-key primitive (e.g., digital signature schemes). Thus the Auth oracle queried by the adversary can be instantiated for specific authentication schemes.

3. OAuth 2.0 Protocols

The OAuth 2.0 framework consists of three entities: the trusted server, the user (or resource owner) and the client. It provides two primary functionalities: authorization, which allows a user to grant a client access to her resources stored at the server; and authentication[1], which makes a user log in at a client utilizing her identity managed by a server.

Before a client can interact with a trusted server, the client needs to be registered at the trusted server. The details of the registration process are out of the scope of the OAuth protocols, and this process is usually a manual task with the trusted server assigning credentials to the client: a public OAuth *client_id* and (optionally) a *client_secret*. Also, a client may register one or more redirection endpoints URIs at a trusted server. As we will see below, the trusted server redirects the user's browser to one of these redirection URIs specified by the client in each run of the OAuth protocol.

Just as mentioned in Section 1, the OAuth 2.0 framework supports four modes (grant types), and only authorization code mode and implicit mode are used in practice.

OAuth 2.0 Authorization Code Mode. As illustrated in Figure 2, the steps of authorization code mode are as follows.

1) The user starts the OAuth flow by visiting a client (e.g., clicking on a button to select a server[2]).

2)–3) The client requests authorization from the user by redirecting the user's browser to the authorization endpoint URI at the server, with the client's redirection endpoint URIs *redirect_uri*, the client's identifier *client_id*, and a randomly generated *state* (a token to prevent CSRF attacks) as parameters.

4) The response of the authorization request prompts the user to provide his authentication credentials to authenticate himself to the server.

1. Although the authentication use-case is unspecified in the OAuth standard, it is widely used in practice.

2. To know which server the user wants to use for authorization, as discussed in [8], the client utilizes explicit user intention tracking to store the user intention in this step and use it later in step 8). Another solution using different redirection URIs to distinguish different servers would lead to naïve RP session integrity attack [8] and thus would not be considered here as in [8].

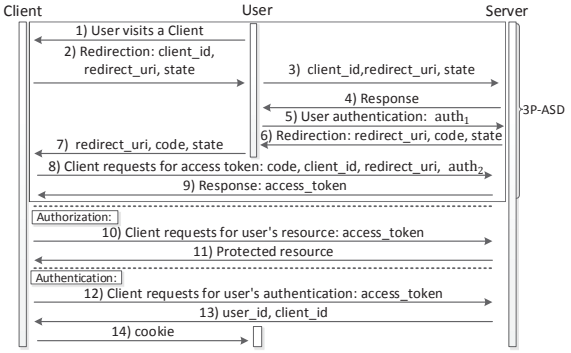

Figure 2: OAuth 2.0 authorization code mode

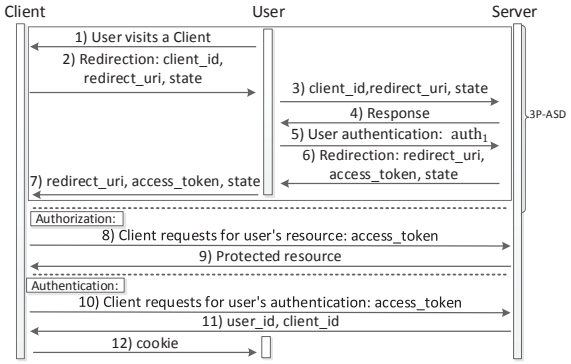

Figure 3: OAuth 2.0 implicit mode

5) The user authenticates himself with any suitable HTTP authentication scheme $auth_1$ matching its security requirements (e.g., by password authentication).

6)–7) If the authentication is valid by the verification algorithm of $auth_1$, the server generates a random authorization code *code* and redirects[3] the user's browser to the client's redirection endpoint URIs *redirect_uri*, with *code* and *state* as parameters.

8) If the *state* is the same as above, the client requests for access token from the server by providing *code*, *client_id*, *redirect_uri*, and identity authentication information by $auth_2$ which is an authentication scheme similar to $auth_1$.

9) The server checks whether the *code* is issued for the client identified by *client_id*, the authentication information is correct for *client_id*, the *redirect_uri* coincides with the one in the step 2), and the *code* has not been redeemed before. If all these checks are successful, the server issues an access token *access_token* as the response.

When OAuth is used for *authorization*:

10) The client sends the user resource request to the server by the use of *access_token*.

11) If the *access_token* check is successful, the server sends the protected resources to the client.

When OAuth is used for *authentication*:

12) The client sends the authentication request to the server by the use of *access_token*.

13) If the *access_token* check is successful, the server sends the user's account ID *user_id* with the *client_id* to the client.

14) The client then issues a session *cookie* to the user or user's browser. The *cookie* consists of a nonce, the user's identifier, and the domain of the server.

OAuth 2.0 Implicit Mode. The implicit mode is a simplified version of authorization code mode. Instead of providing an authorization code, the server directly delivers an access token to the client via the user. Specifically, steps 1)-5) are the same in both types, while steps 6)-9) in authorization code mode (see Figure 2) are replaced with steps 6) and 7) in implicit mode (see Figure 3). In addition, as a special requirement in the implicit mode, the *client_id* in 11) should be checked by the client when authenticating the user, to prevent re-use of access tokens across clients as explained in [36]. The implicit mode is useful in cases where the cryptographic primitives are too heavy to be implemented or executed by the client.

Remark 1. All the transcripts that contain confidential information in OAuth 2.0 are transmitted through the secure channel (e.g., TLS), which is abstracted into the ACCE channel with server-only authentication in this paper (the client authentication is optional in many ACCE protocols such as TLS and SSH). Specifically, the *state*, *code* and *access_token*, the user authentication and client authentication (to the server) as well as the final resources and cookie in both protocols should be protected through the secure channel, while there is no mandatory requirement for other message flows that do not contain any confidential information.

4. Authenticated Secret Distribution Protocols

In this section, we introduce the concept of three-party authenticated secret distribution (3P-ASD) protocols to model the authorization technology and define the security model of 3P-ASD protocols.

A 3P-ASD protocol is generally carried out among three parties: the trusted server S, the user (also, the resource owner) U, and the client C. An honest execution of a 3P-ASD protocol should result in a client C obtaining a fresh secret *sec* from the server S with the permission of an honest user U, where the secret *sec* is stored at S associated with U for a pre-defined time period. In particular, U first authenticates himself to S and at the same time expresses his agreement on issuing the secret *sec* to C, then S (optionally) authenticates C and finally issues the corresponding *sec* to C securely. Note that it should be impossible for any other client without the permission of U to obtain the secret *sec*, even with the permission of another user $U' \neq U$. Therefore, achieving secure entity authentication, the confidentiality, and the integrity are the main security goals of 3P-ASD protocols.

4.1. Execution Environment

A protocol is carried out by a set of parties $P \in \mathcal{P}$, and each party P is a (potential) participant in the system. $\mathcal{P}$ is partitioned into three disjoint sets $\mathcal{U}$, $\mathcal{C}$ and $\mathcal{S}$, consisting of the users (or the user's browser by which

3. During the redirection, a designated HTTP status code 30X is used.

the user can interact with the servers and clients), clients and servers, respectively[4]. Note that, any suitable authentication scheme matching its security requirements is supported in 3P-ASD protocols, and thus the key setting is not specified. That is, the long-term key associated with each party are either based on asymmetric keys or pre-shared secret keys (PSK), for example, if the password based authentication is supported, a password should be pre-shared between two parties during the deployment.

We use an administrative label π_P^s ($s \in [1, n_\pi]$) to refer to the s-th session of a 3P-ASD protocol instance Π at the party P. In addition, Π is logically built out of n sequentially running sub-sessions between two parties (i.e., Π_1,Π_2,...,Π_n) according to the protocol specifications, and $\pi_P^s.\Pi_j$ denotes the execution of π_P^s in its sub-session Π_j for $j \in [1, n]$. Each sub-session begins when the current sub-session ends. Note that the secret *sec* would not be used in any sub-session according to the definition of 3P-ASD protocols.

Associated to each session π_P^s, there is a set of variables to reflect the local state of π_P^s during run of the protocol:

- *role* $\in \{\mathcal{U}, \mathcal{C}, \mathcal{S}\}$: the session owner P's role in this session.
- *peers*: a list of the identities of the intended communication peers of π_P^s, in particular, $peers_i$ indicates the intended peer in the sub-session $\pi_P^s.\Pi_i$.
- $\{sk, pk\}$: the (possibly empty) long-term secret/public key of P.
- $PSK_{PP'}$: the (possibly empty) pre-shared secret key shared between party P and its peer P', where $P' \in \pi_P^s.peers$.
- $\overrightarrow{\alpha} = \{\alpha_1, ..., \alpha_n\}$: a vector of accept states $\alpha_i \in \{\mathsf{accepted}, \mathsf{rejected}, \mathsf{running}\}$, where α_i represents the acceptance state of the intermediate sub-session $\pi_P^s.\Pi_i$ and α_n represents the acceptance state of the full protocol π_P^s. The sub-session $\pi_P^s.\Pi_i$ can run if and only if $\pi_P^s.\Pi_{i-1}$ has been accepted.
- $insec = \{insec_1,...,insec_{m_{in}}\} \cup \{\perp\}$: the (possibly empty) list of m_{in} ($m_{in} \in \mathbb{N}$) intermediate secrets distributed in π_P^s in chronological order, where $insec_i \in \{0,1\}^*$. The leakage of intermediate secrets may affect the security of final secrets.
- $sec = \{sec_1,...,sec_{m_s}\} \cup \{\perp\}$: the (possibly empty) list of m_s ($m_s \in \mathbb{N}$) final secrets distributed in π_P^s in chronological order, where $sec_i \in \{0,1\}^*$.
- $mod = \{mod_1,...,mod_n\}$: the vector of the authentication modes, where $mod_i \in \{\mathsf{auth}, \mathsf{unauth}\}$ indicates the authentication mode of $\Pi_P^s.\Pi_i$, and auth indicates $\pi_P^s.\mathsf{peer}_i$ is authenticated to P, otherwise unauthenticated.
- $K = \{K_1,...,K_n\}$: the (possibly empty) vector of the session keys, where $K_i \in \{0,1\}^* \cup \{\perp\}$ indicates the session key established in the sub-session $\pi_P^s.\Pi_i$.
- $T = \{T_1,...,T_n\}$: the vector of transcripts, where $T_i \in \{0,1\}^*$ indicates the transcripts of messages sent or received during the running of sub-session $\pi_P^s.\Pi_i$.

In the above definition, the parameters n, m_{in} and m_s are instantiated according to specifications, while n_π is used to bound the security of 3P-ASD in the proof.

4. In reality one party may play different roles. To distinguish different roles, we assume parties are partitioned into disjoint sets in one instance.

To model the impact of the leakage of the intermediate parameter on the security, we use sec to denote the final secret (e.g., the access token in OAuth), and use *insec* to denote the intermediate secrets generated during the issue of *sec* in 3P-ASD protocols (e.g., the code in OAuth).

As sessions are unique, we write as a shorthand, e.g., $\pi_P^s.peers$ for the element *peers* contained in the variables of π_P^s, and analogously for other variables.

During the running of 3P-ASD protocol, the honest server S keeps an entry (S, U, *insec*, *sec*) for each user U.

4.2. Adversarial Interaction

We consider a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ who has full control over the communications between all parties, enabling interception, injection and dropping of messages. The adversary $\mathcal{A}$ may interact with the protocol via the following oracle queries.

- NewSession $(P, [V, W])$: This query creates a new session π_P^s at the party P, optionally specifying its intended communication peers (possibly empty) V and W. It is also required that the roles of P, V and W are different.
  The variable $\overrightarrow{\alpha}$ is updated as $\pi_P^s.\alpha_1 = \mathsf{running}$, and the variables $\{role, pk, sk, peers, PSK, insec, sec, mod, K, T\}$ (for P, V and W) are initialized according to the specific 3P-ASD protocol.
  Besides, if π_P^s plays the role of protocol initiator who sends the first flow, π_P^s also produces and returns its first message m according to the specification of protocol Π.
- Send (π_P^s, m): This query sends a message m to π_P^s. If $\pi_P^s.\alpha_n \neq \mathsf{running}$, returns $\perp$. Otherwise, π_P^s runs the protocol with the input m on behalf of P, and returns the response m^* and the updated acceptance state $\pi_P^s.\alpha$ according to the protocol specification. Note that for web-based authorization protocols like OAuth 2.0, some distinguished control messages have special behaviours:
  - If $P \in U$ and $m = (\mathsf{'Redirectcode'}, URI)$, then the user P's browser would be redirected to the endpoint URI URI[5] at some client or server (possibly with some parameters appended to the URI) according to the HTTP redirect code Redirectcode = 30X [37].
  - If $P \in U$ and $m = (\mathsf{'MalTransfer'}, URI, para)$, the user $P \in U$ (i.e., P's browser) is sent to an URI URI that linked to another party P', and then π_P^s visits the URI and transfers $para$[6] to the current session of P'.
  - If $P \in U$ and $m = (\mathsf{'OriginLeak'}, URI)$, the user $P \in U$ (i.e., P's browser) is sent to an URI URI that linked to another party P', and then π_P^s visits the URI and sends a request including the HTTP referrer header field[7] to the current session of P'.

5. It's called the authorization endpoint URI at the server side and the redirection endpoint URI at the client side as depicted in Figure 2.

6. In practice, the URI is often be used for parameter delivery when using the GET HTTP request method, where $para$ is just appended to the tail end of the URI and the resulting string can also be considered as a new URI. In addition, the parameter may also be transferred in the POST bodies when using POST HTTP request method.

7. The referrer header, as a special message format and an optional HTTP header field during message transmission, contains certain information that indicates the last address of the user's browser, and thus by checking the referrer, the new webpage at P' can see where the request originates.

For convenience, we use MalTransfer and OriginLeak to denote the last two cases respectively.

- KeyReveal (π_P^s, i): This query returns $\pi_P^s.K_i$ (if it exists), the session key in sub-session $\pi_P^s.\Pi_i$ to the adversary. If $\pi_P^s.\alpha_i \neq$accepted, $\pi_P^s.K_i = \perp$.
- Corrupt $(P, [V])$: This query returns a certain long-term secret key of party P according to the second input entry. Specifically,
  - Corrupt (P, V): the pre-shared key $PSK_{P,V}$ (if it exists) is returned.
  - Corrupt (P): the long-term secret key sk of P is returned.
- InsecReveal (π_P^s): This query provides the intermediate secret *insec* (if exists) of some user U at some server S obtained by the party P from π_P^s.
- SecReveal (π_P^s): This query provides the final secret *sec* (if exists) of some user U at some server S obtained by the party P from π_P^s. If $P = U$ or $P = S$, all sessions in which P participates are considered to be issued SecReveal.

In our model, we are working in the post-specified peers model introduced in [38], where the identities of a session's peer can be learned during the protocol running rather than at the beginning of the protocol run.

In addition, the queries MalTransfer and OriginLeak are indispensable for modeling the flexibility of the web browser, which often becomes the source of web-based attacks like CSRF against OAuth-like authorization protocols. Specifically, by MalTransfer query the adversary can maliciously force an honest browser to send some message to another honest website, and by OriginLeak query, the adversary can obtain some leaked information on the last page of the user's browser. In practice, due to the lack of vigilance, the user may click the link built by the adversary elaborately just as presented in the MalTransfer and OriginLeak queries, which leads to practical attacks such as the state leak attack against OAuth protocols (refer to Section 5.3). Particularly, the policy of the user's browser behavior in regard to the referrer header determines how much private information will be leaked in the OriginLeak query. For example, to resist the state leak attack where the one-time *state* may be leaked through the referrer header, the recently introduced referrer policy [39] is adopted in [8] to specify that referrer header cannot contain anything except for the origin of the respective page.

4.3. Security Definition

Freshness. Firstly we need to exclude trivial attacks in which the adversary can break the security of the session by trivial means. For example, an adversary can obtain the secret itself immediately by the query SecReveal, or reveal the session key by the query KeyReveal and furthermore obtain the secret. In order to exclude trivial attacks we present the freshness predicate $\mathsf{Fresh}(\pi_A^s.\Pi_i)$ during the running of the sub-session Π_i. This predicate is evaluated at the end of the sub-session $\pi_A^s.\Pi_i$ and yields true if and only if the following conditions hold:

1) $\pi_A^s.\alpha_i =$ accepted, i.e., π_A^s has accepted at sub-session Π_i,
2) The adversary has not queried KeyReveal (π_A^s, i), InsecReveal (π_A^s) or SecReveal (π_A^s),
3) The adversary has not queried Corrupt (A, B), Corrupt (A) or Corrupt (B), where $\pi_A^s.peers_i = B$,
4) The adversary has not queried KeyReveal (π_B^t, i), InsecReveal (π_B^t) or SecReveal (π_B^t), where $\pi_A^s.\Pi_i$ has a matching conversation to $\pi_B^t.\Pi_i$ (if it exists).

Definition 7. (Freshness) A sub-session $\pi_A^s.\Pi_i$ is fresh if $\mathsf{Fresh}(\pi_A^s.\Pi_i) = \mathsf{true}$. Furthermore, a session π_A^s is fresh if for all $i \in [1, n]$ we have $\mathsf{Fresh}(\pi_A^s.\Pi_i) = \mathsf{true}$.

In the above definition, matching conversation between two sessions implies the matching of session keys. Note that no Corrupt query is allowed for a fresh session π_A^s, since the secret issuing is actually an identity authentication process, thus the leakage of the long-term secret key would lead to impersonation and in turn the leakage of the secret to be issued in current and even future sessions. Additionally, the leakage of the intermediate secrets is not allowed either, however, if the intermediate secrets can only be used once like the *code* in OAuth 2.0, we may relax the requirement by allowing the InsecReveal query only after π_A^s accepted.

Security Definition. The security of a 3P-ASD protocol is defined by requiring that (i) the protocol provides entity authentication, ensuring honest parties cannot be forged and messages indeed come from the party it claims to be by requiring that any fresh session accepts only when there exists a unique matching conversation in each sub-session; (ii) the protocol provides confidentiality, ensuring that any PPT adversary cannot obtain the secret from a fresh session with non-negligible probability; and (iii) the protocol provides integrity to resist CSRF attacks (refer to the state leak attack against OAuth in Section 5.3) for web-based protocols, ensuring that the client cannot obtain the secret from one user (maybe the adversary) but wrongly believes it from another honest user.

Each security notion is formally defined by a game played between a PPT adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, denoted by $\mathrm{G}^{\mathsf{EA}}_{\Pi,\mathcal{A}}$, $\mathrm{G}^{\mathsf{Con}}_{\Pi,\mathcal{A}}$ and $\mathrm{G}^{\mathsf{Int}}_{\Pi,\mathcal{A}}$ respectively, with the same overall setup but different winning conditions. In each game, $\mathcal{C}$ generates the long-term public/secret key pair and pre-shared key for each participant. The adversary $\mathcal{A}$ receives all parties' public keys as input and issues NewSession, Send, KeyReveal, Corrupt, InsecReveal, SecReveal, MalTransfer and OriginLeak queries.

Definition 8. (Entity Authentication) In the security game $\mathrm{G}^{\mathsf{EA}}_{\Pi,\mathcal{A}}$ for entity authentication, suppose the PPT adversary $\mathcal{A}$ interacts with a 3P-ASD protocol Π in the above execution environment and at some point $\mathcal{A}$ stops with no output.

We say that $\mathcal{A}$ wins the entity authentication game, denoted by $\mathrm{G}^{\mathsf{EA}}_{\Pi,\mathcal{A}} = 1$, if there exists a session π_A^s and its sub-session $\pi_A^s.\Pi_i$ such that

1) $\pi_A^s.\alpha_i =$ accepted and $\pi_A^s.peers_i = B$,
2) $\pi_A^s.mod_i =$ auth,
3) $\mathcal{A}$ did not issue Corrupt (A, B), Corrupt (A) or Corrupt (B) before $\pi_A^s.\Pi_i$ accepted,
4) $\mathcal{A}$ did not issue KeyReveal to oracle π_B^t such that π_B^t accepted in sub-session Π_i while having a matching

conversation to π_A^s (if such an oracle exists) in Π_i[8],
5) There does not exist an oracle π_B^t such that π_A^s has a matching conversation to π_B^t in Π_i.

If an oracle π_A^s accepts in the above sense, then we say that π_A^s accepts maliciously in sub-session Π_i.

Π satisfies entity authentication, if for all PPT adversaries $\mathcal{A}$ the advantage $\mathrm{Adv}^{\mathsf{EA}}_{\Pi,\mathcal{A}} = \Pr[\mathrm{G}^{\mathsf{EA}}_{\Pi,\mathcal{A}} = 1]$ is negligible.

Definition 9. (Confidentiality) In the security game $\mathrm{G}^{\mathsf{Con}}_{\Pi,\mathcal{A}}$ for confidentiality, suppose the PPT adversary $\mathcal{A}$ interacts with a 3P-ASD protocol Π in the above execution environment and at some point $\mathcal{A}$ stops and outputs a secret sec_i.

We say that $\mathcal{A}$ wins the confidentiality game, denoted by $\mathrm{G}^{\mathsf{Con}}_{\Pi,\mathcal{A}} = 1$, if the following conditions hold:
1) There exists an entry (S, U, sec^*) stored by S, where $S \in \mathcal{S}$, $U \in \mathcal{U}$, and $sec_i \in sec^*$,
2) The server oracle π_S^t has issued sec^* to the client oracle π_C^s in a specific instance initiated[9] by the user oracle π_U^r for some $r, s, t \in [1, n_\pi]$,
3) π_S^t, π_C^s and π_U^r are all fresh.

Π satisfies confidentiality, if for all PPT adversaries $\mathcal{A}$ the advantage $\mathrm{Adv}^{\mathsf{Con}}_{\Pi,\mathcal{A}} = \Pr[\mathrm{G}^{\mathsf{Con}}_{\Pi,\mathcal{A}} = 1]$ is negligible.

Remark 2. In our definition, the adversary $\mathcal{A}$ wins when he outputs just one element in the sec^* entry, thus confidentiality ensures that each element of the sec^* entry in the fresh session should be confidential to the adversary.

Definition 10. (Integrity) In the security game $\mathrm{G}^{\mathsf{Int}}_{\Pi,\mathcal{A}}$ for integrity, suppose the PPT adversary $\mathcal{A}$ interacts with a 3P-ASD protocol Π in the above execution environment and at some point $\mathcal{A}$ stops with no output.

We say that $\mathcal{A}$ wins the integrity game, denoted by $\mathrm{G}^{\mathsf{Int}}_{\Pi,\mathcal{A}} = 1$, if there exists one user oracle π_U^r has initiated and completed one session with the client oracle π_C^s and server oracle π_S^t for $r, s, t \in [1, n_\pi]$ such that the following conditions hold:
1) π_U^r, π_C^s and π_S^t are all fresh,
2) Eventually C obtains a secret sec^* from its oracle π_C^s,
3) There exists an entry (S, U', sec^*) stored by S, however $U' \neq U$.

Π satisfies integrity, if for all PPT adversaries $\mathcal{A}$ the advantage $\mathrm{Adv}^{\mathsf{Int}}_{\Pi,\mathcal{A}} = \Pr[\mathrm{G}^{\mathsf{Int}}_{\Pi,\mathcal{A}} = 1]$ is negligible.

Remark 3. Note that U' in the definition may be an adversary, which captures the attack where the client obtains a secret from the adversary however wrongly believes the secret from the honest user U.

Definition 11. (3P-ASD Security) We say a 3P-ASD protocol Π is secure if Π satisfies entity authentication, confidentiality and integrity.

4.4. Instantiation of 3P-ASD in OAuth 2.0

According to the specification of OAuth, the core module of the OAuth protocol, the process of access token issue, can be regarded as an instantiation of the 3P-ASD protocol. Specifically, in OAuth 2.0, *insec* = {*state, code*} and *sec* = {*code, access_token*}, where *code* is empty for implicit mode of OAuth 2.0.

When we instantiate a 3P-ASD protocol in OAuth, the essential parts included in *insec* and *sec* should be identified carefully. Sometimes the only inclusion of the final secret in *sec* is insufficient, since the leakage of some other information may result in the security definition for 3P-ASD being too weak to conclude the entire security of OAuth. For example, in OAuth 2.0, *access_token* is the final secret to be issued with a predetermined validity period used by the client, while the intermediate secret *code* is generated uniformly and can be used only once. Intuitively, it is enough to require the confidentiality of the *access_token* to guarantee secure authorization and authentication in OAuth. Unfortunately, the leakage of *code* before its first use may break the authentication goal of OAuth 2.0 even though the final *access_token* is confidential, which can be confirmed in the mix-up attack against OAuth 2.0 (refer to Section 5.3). Therefore, in the instantiation of OAuth 2.0, it is indispensable for containing *code* in the *sec* entry.

Note that according to the definition of intermediate secrets *insec* in Section 4, the leakage of parameters in *insec* may affect the security of 3P-ASD protocols and thus cannot be obtained trivially by the InsecReveal query. For example, in OAuth 2.0, the leaked *state* would lead to CSRF attacks breaking the integrity goal, and the leaked *code* would lead to leakage of the final *access_token*.

The above analysis indicates that each parameter generated in OAuth can affect the final security, and furthermore confirms that the abstraction of 3P-ASD primitive from OAuth is not only for a modular proof, but for a fine-grained and comprehensive understanding of OAuth security.

5. Security Model of OAuth 2.0 Protocols

In this section, we define a general three-party security model for OAuth protocols, and also identify several known attacks against OAuth 2.0 with our security model.

5.1. Overview

As specified in Section 3, OAuth can be regarded as an application layer protocol combining the "secret issue" process and "secret usage" process entirely. Therefore, the security of an OAuth protocol is defined by requiring that the protocol provides authorization, authentication, integrity for authorization and integrity for authentication.

Recall that the OAuth 2.0 standard [1] only explicitly purpose itself for authorization rather than user authentication, and thus achieving secure authorization is the most fundamental requirement for OAuth 2.0. On the other hand, OAuth 2.0 authorization protocol flow have been significantly re-purposed for user authentication by most major identity providers like Google, Facebook, Microsoft and Twitter. As a result, the subsequent document RFC 6819 [9] for threat model and security considerations of OAuth 2.0 have discussed the threats against the so-called "OAuth login" scenarios where OAuth 2.0 is used for

8. With this condition, a kind of trivial attack can be excluded, where π_B^t sends the last message and thus reaches accepted earlier than π_A^s. If $\mathcal{A}$ is allowed to query KeyReveal to oracle π_B^t, he can re-encrypt the last messages to π_A^s, which leads to the acceptance of π_A^s without a matching conversation. More details can be referred to [16].

9. We say a party "initiates" the protocol if he starts the protocol by sending the first message flow.

user authentication, and a series of work on the analysis of OAuth 2.0 such as [3] [7] [8] also consider the authentication property to capture practical application scenarios. Therefore, in this paper OAuth 2.0 should be analyzed for not only the original design (i.e., secure authorization) but also the practical application (i.e., secure authentication). In addition, we define the integrity for authorization/authentication properties to model the CSRF attacks against the client and server as specified in [1] [9].

The execution environment is very similar to that for 3P-ASD protocols in Section 4.1 and would be omitted to avoid needless repetition, except for a few differences as follows:

1) The intermediate secret variable associated with the session π_P^s is redefined as $insec' = insec \cup sec$, where *insec* and *sec* are parameters of the 3P-ASD part of OAuth (and can be specified for OAuth 2.0 as in Section 4.4).
2) The final secret variable associated with the session π_P^s is redefined as $sec' = \{m, cookie\}$, where *cookie* denotes the cookie information, consisting of a nonce, the domain of the server S and the identifier U_{id} of user U, and m denotes the resource of U stored at S. Note that m is empty for the authentication functionality, whereas *cookie* is empty for the authorization functionality.
3) The server S keeps an entry (S, U, m) for each user U.
4) The honest client C keeps an entry $(C, S, U, cookie)$ for the user U and the server S.

In addition, the definition of Freshness in Definition 7 remains unchanged, and the final and intermediate secrets are instantiated with the above $insec'$ and sec' for OAuth instances.

5.2. Security Definition

The security of an OAuth protocol is defined to satisfy the security requirements for OAuth in practice as in [8]. Specifically, (i) the protocol provides authorization security, ensuring that the adversary $\mathcal{A}$ cannot obtain the resource stored in the server S by the user U from one fresh OAuth session; (ii) the protocol provides authentication security, ensuring that the adversary $\mathcal{A}$ cannot obtain the valid cookie from one fresh OAuth session to log in a client C; (iii) the protocol provides the integrity for authorization security, ensuring that that a client can obtain a user's resource only when the user has initiated an OAuth instance with him, and moreover the client cannot obtain the resource of other users from the instance, and (iv) the protocol provides integrity for authentication security, ensuring that a client can accept the user's authentication only when the user has initiated an OAuth instance with him, and moreover the client cannot accept the authentication of other users from that instance.

The security properties for OAuth protocols are formally defined by a game played between a PPT adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, denoted by $G^{Author}_{OAuth,\mathcal{A}}$, $G^{Authen}_{OAuth,\mathcal{A}}$ [10], $G^{IntAuthor}_{OAuth,\mathcal{A}}$ and $G^{IntAuthen}_{OAuth,\mathcal{A}}$, respectively. In each game, the challenger $\mathcal{C}$ generates the long-term public/secret key pair and pre-shared key for each participant. The adversary $\mathcal{A}$ receives all parties' public keys as input and issues NewSession, Send, KeyReveal, Corrupt, InsecReveal, SecReveal, MalTransfer and OriginLeak queries as defined in Section 4.2.

Definition 12. (Authorization) In the security game $G^{Author}_{OAuth,\mathcal{A}}$ for authorization, suppose the PPT adversary $\mathcal{A}$ interacts with an OAuth protocol in the above execution environment and at some point $\mathcal{A}$ stops and outputs the resource m^*.

We say that $\mathcal{A}$ wins the authorization game, denoted by $G^{Author}_{OAuth,\mathcal{A}} = 1$, if the following conditions hold:

1) There exists an entry (S, U, m^*) stored by S for some $S \in \mathcal{S}$ and $U \in \mathcal{U}$,
2) The server oracle π_S^t has issued m^* to the client oracle π_C^s in a specific instance initiated by the user oracle π_U^r for some $r, s, t \in [1, n_\pi]$,
3) π_S^t, π_C^s and π_U^r are all fresh.

An OAuth protocol provides authorization security goal, if for all PPT adversaries $\mathcal{A}$ the advantage $Adv^{Author}_{OAuth,\mathcal{A}} = Pr\,[G^{Author}_{OAuth,\mathcal{A}} = 1]$ is negligible.

Definition 13. (Authentication) In the security game $G^{Authen}_{OAuth,\mathcal{A}}$ for authentication, suppose the PPT adversary $\mathcal{A}$ interacts with an OAuth protocol in the above execution environment and at some point $\mathcal{A}$ stops and outputs the cookie $cookie^*$.

We say that $\mathcal{A}$ wins the authentication game, denoted by $G^{Authen}_{OAuth,\mathcal{A}} = 1$, if the following conditions hold:

1) There exists an entry $(C, S, U, cookie^*)$ stored by C for some $C \in \mathcal{C}$, $S \in \mathcal{S}$ and $U \in \mathcal{U}$,
2) The client oracle π_C^s has issued the $cookie^*$ to the user oracle π_U^r using the server oracle π_S^t [11], where $U \in \mathcal{U}$, $C \in \mathcal{C}$, $S \in \mathcal{S}$ and $r, s, t \in [1, n_\pi]$,
3) π_S^t, π_C^s and π_U^r are all fresh.

An OAuth protocol provides authentication security goal, if for all PPT adversaries $\mathcal{A}$ the advantage $Adv^{Authen}_{OAuth,\mathcal{A}} = Pr\,[G^{Authen}_{OAuth,\mathcal{A}} = 1]$ is negligible.

Definition 14. (Integrity for authorization) In the security game $G^{IntAuthor}_{OAuth,\mathcal{A}}$, suppose that the PPT adversary $\mathcal{A}$ interacts with an OAuth protocol in the above execution environment and at some point $\mathcal{A}$ stops with no output.

We say that $\mathcal{A}$ wins the integrity for authorization game, denoted by $G^{IntAuthor}_{OAuth,\mathcal{A}} = 1$, if at least one of the following two conditions holds:

1) When $\mathcal{A}$ terminates, there exists a client oracle π_C^s obtaining m^* for $C \in \mathcal{C}$ and $s \in [1, n_\pi]$ such that
   - There exists an entry (S, U, m^*) stored by S for some $S \in \mathcal{S}$ and $U \in \mathcal{U}$,
   - U has never initiated an OAuth instance with the client oracle π_C^s and a server oracle π_S^t for some $t \in [1, n_\pi]$,
   - $\mathcal{A}$ has not issued Corrupt query for U, C or S,
   - π_C^s is fresh.
2) When $\mathcal{A}$ terminates, the user oracle π_U^r has initiated and completed an OAuth instance with the client oracle π_C^s and server oracle π_S^t for some $r, s, t \in [1, n_\pi]$ such that
   - π_U^r, π_C^s and π_S^t are all fresh,

10. Authen refers to the single sign-on functionality in OAuth.

11. The description "using the server" denotes that the user utilizes its identity managed by the server.

- C obtains m^* from its oracle π_C^s,
- There exists some entry (S, U', m^*) stored by S for some $U' \in \mathcal{U}$, however $U' \neq U$.

An OAuth protocol provides integrity for authorization goal, if for all PPT adversaries $\mathcal{A}$ the advantage $\mathrm{Adv}^{\mathsf{IntAuthor}}_{\mathsf{OAuth},\mathcal{A}} = \Pr[\mathrm{G}^{\mathsf{IntAuthor}}_{\mathsf{OAuth},\mathcal{A}} = 1]$ is negligible.

Definition 15. (Integrity for authentication) In the security game $\mathrm{G}^{\mathsf{IntAuthen}}_{\mathsf{OAuth},\mathcal{A}}$, suppose that the PPT adversary $\mathcal{A}$ interacts with an OAuth protocol in the above execution environment and at some point $\mathcal{A}$ stops with no output.

We say that $\mathcal{A}$ wins the integrity for authentication game, denoted by $\mathrm{G}^{\mathsf{IntAuthen}}_{\Pi,\mathcal{A}} = 1$, if at least one of the following two conditions holds:

1) When $\mathcal{A}$ terminates, there exists a client oracle π_C^s ($C \in \mathcal{C}$ and $s \in [1, n_\pi]$) which accepts a user's login authentication with identity U_{id} using the server S such that
   - U_{id} uniquely identifies the user U at the server S,
   - U has never initiated an OAuth instance with the client oracle π_C^s and a server oracle π_S^t for some $t \in [1, n_\pi]$,
   - $\mathcal{A}$ has not issued Corrupt query for U, C or S,
   - π_C^s is fresh.
2) When $\mathcal{A}$ terminates, the user oracle π_U^r has initiated and completed an OAuth instance with the client oracle π_C^s and server oracle π_S^t for some $r, s, t \in [1, n_\pi]$ such that
   - π_U^r, π_C^s and π_S^t are all fresh,
   - π_C^s accepts the user's login authentication with U'_{id},
   - U'_{id} uniquely identifies the user U' at the server S, however $U' \neq U$.

An OAuth protocol provides integrity for authentication security goal, if for all PPT adversaries $\mathcal{A}$ the advantage $\mathrm{Adv}^{\mathsf{IntAuthen}}_{\mathsf{OAuth},\mathcal{A}} = \Pr[\mathrm{G}^{\mathsf{IntAuthen}}_{\mathsf{OAuth},\mathcal{A}} = 1]$ is negligible.

Definition 16. (OAuth Security) We say an OAuth protocol is secure if it satisfies authorization, authentication, integrity for authorization and integrity for authentication.

Remark 4. In order to prevent a trivial attack, in the condition 1) of Definition 14 (resp. Definition 15), π_C^s is required to be fresh. Specifically, if an adversary can issue a KeyReveal query to π_C^s, with U's resource m (resp. U's identifier U_{id}) possibly obtained from his own instance with U, he can encrypt m (resp. U_{id}) to π_C^s, which trivially breaks the integrity for authorization (resp. authentication).

Remark 5. Note that U' in the Definition 14 and 15 may be an adversary, which captures the attacks where the client obtains the resource from the adversary however wrongly believes the resource from the honest user U, or the client accepts the authentication of U under the adversary's name.

Remark 6. Note that the condition 1) of Definition 14 and Definition 15 is not included in the definition of integrity for 3P-ASD protocols (Definition 10), since it has been captured by the entity authentication property in Definition 8 as mentioned above, namely if the client can obtain U's secret at S without U's permission, then the entity authentication goal will be broken by impersonating U to S.

Remark 7. It is worth mentioning that rather than being tailored to OAuth 2.0, our model hardly involves any protocol specifications except for the 3P-ASD instantiation, and thus can generally apply to a family of OAuth-like protocols such as OpenID Connect, OAuth 1.0 and other protocols for access delegation, with a specified instantiation of 3P-ASD respectively like in Section 4.4.

5.3. Capturing Known Attacks in Our Model

In this subsection, we recall some known attacks against OAuth 2.0 and explain with our model why OAuth 2.0 protocols without countermeasures are vulnerable.

307 redirect attack. The 307 redirect attack against the authorization code mode and the implicit mode of OAuth 2.0 was proposed by Fett et al. in 2016 [8] if the server (used for login) chooses the 307 HTTP status code in step 6) of Figure 2. Specifically, when a user U wants to log in at a client C managed by the adversary, then he is redirected to the server S in steps 3)-5) to enter his credentials in a HTTP POST request. S receives and checks these credentials, and redirects U's browser to C's redirection endpoint. Since the 307 HTTP status code is used for this redirection, U's browser would send a POST request to C with all data (certainly including the user's credentials) from the previous request. As a result, the adversary running C can impersonate U.

Formally, this attack is captured by our model as follows. The adversary $\mathcal{A}$ can query Corrupt oracle to control a client C and obtain U's credentials, and finally $\mathcal{A}$ can not only impersonate U and obtain U's resource but also use U's resource at S to log in at the honest clients without U's participation. Therefore, both the authorization security and the authentication security of OAuth are broken. To resist this attack, Fett et al. [8] suggests to use 303 HTTP status code instead of 307 HTTP status code, since in the HTTP standard [37] only 303 redirect is defined unambiguously to drop the body of the previous HTTP POST request (i.e., the user's credentials in this attack).

IdP mix-up attack. The IdP mix-up attack against OAuth 2.0 (the authorization code mode and the implicit mode) was proposed by Fett et al. in 2016 [8]. In this paper, IdP means the server. Let us take the authorization code mode as an example to recall the mix-up attack. First, the adversary $\mathcal{A}$ intercepts the first message 1) from U to C and replaces S in the message 1) with S' managed by himself. Second, $\mathcal{A}$ intercepts the response message 2) of C (containing a redirect to S') and modifies a redirect to S. Third, by the normal running 3)-7) of protocol, C obtains the *code* issued by S, however he wrongly believes that the *code* was issued by the malicious S', and thus leaks the *code* to S' (i.e., the adversary) in 8). Finally, using the *code*, $\mathcal{A}$ can obtain the *access_token* and in turn the protected resource stored at S (resp. obtain the *cookie* from C by starting a new login process at C with that *code*, and furthermore impersonate U to log in at C with the *cookie* though in this case the *access_token* is unknown). Note that this attack can be valid only in the case that the server S does not need to provide the authentication to C, and the communication channel in 1) and 2) is public.

Formally, this attack is captured by our model as follows. The adversary $\mathcal{A}$ can query Send oracle to replace

the original messages 1) and 2), and finally receive the protected resource m stored at S and the cookie including (U_{id}, S) from C, where U_{id} is the unique identifier of U at S. Explicitly, the sessions among S, C and U are fresh, and thus the authorization security and the authentication security of OAuth are broken respectively. Fett et al. [8] suggests that the identity of the server should be included in the redirect URI in some form that cannot be influenced by the adversary to resist this attack, correspondingly, each server should add such a parameter (e.g., in 6) and 7) of Figure 2 and Figure 3), which has been adopted by RFC draft [40].

State leak attack. The state leak attack against the authorization code mode of OAuth 2.0 was also proposed in [8], where an adversary $\mathcal{A}$ can force a user to be logged in under the $\mathcal{A}$'s name at the client C, or force C to use the resource of $\mathcal{A}$ instead of the user. Specifically, if a user U wants to log in at a client C, U needs to finish the authentication to S, and then U is redirected to C (the step 7) of Figure 2). However, the response to this request (the step 14) of Figure 2) can be a page containing a link to the $\mathcal{A}$'s website (via the OriginLeak query). When U clicks the link, U's browser sends a request to $\mathcal{A}$. The Referrer header in this request contains the full URI of the user's previous page (i.e., including *state* and *code*). Then, $\mathcal{A}$ can use the leaked state to perform a CSRF attack against the victim U. For example, as a user, $\mathcal{A}$ visits C by S and then obtains the *code*$'$ issued by S to C, then he can redirect U's browser via the MalTransfer query to C again with the leaked *state* as well as the *code*$'$. Accordingly, the user U will log in at C as $\mathcal{A}$, and moreover C will obtain the resources from $\mathcal{A}$ but C wrongly believes them from U.

Formally, this attack is captured by our model as follows. At first, U wishes to log in at C with the aid of S, and the user oracle π_U^r initiates an instance with client oracle π_C^s and server oracle π_S^t. However, due to the leaked *state*, π_C^s accepts with U_{id*}, which uniquely identifies the user U^* at S ($U^* \neq U$). In this instance the sessions among U, C and S are all fresh, and thus the integrity for authentication is broken. Similarly, the integrity for authorization is also broken.

Intuitively, the countermeasures against the state leak attack are limiting the *state* to a single use and avoiding leakage of the *state* by network [39] (e.g., the Referrer header can be blocked entirely), and thus the OriginLeak query is not useful for the adversary.

For other attacks. Besides the latest attacks as described above, our model can also capture other known attacks against OAuth 2.0 in the protocol-level, which is also the common advantage of the game-based model in the computational approach. However, if the adversary owns stronger abilities, e.g., the adversary may access some private secrets of the victim due to lack of protection, or if the attacks are based on the flaws in the sense of the specific implementation rather than the protocol itself, the model would not be applicable any more.

6. Security Analysis of OAuth Protocols

In this section we will perform a computationally sound security analysis of the revised OAuth protocols by abstracting the main part as a 3P-ASD protocol, achieving not only a modular analysis but more important a deep understanding of the security and design of OAuth 2.0.

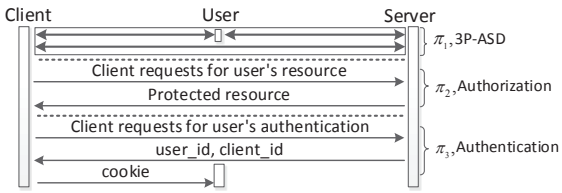

Figure 4: The proof framework of OAuth 2.0

For simplicity, in the following parts of this paper, n_π, n_c, n_u, and n_s denote the maximum number of sessions, clients, users and servers, respectively. In addition, we use CHAN to denote channel establishment protocols which are actually ACCE protocols for secure message transmission.

6.1. Security Analysis of OAuth 2.0

In this subsection, we prove both the authorization mode and the implicit mode of OAuth 2.0 with the above countermeasures satisfy the security goals defined in Section 4.2.

First, we abstract the construction framework of OAuth shown in Figure 4, and prove that the composed OAuth 2.0 protocol is secure if the following conditions hold: 1) the protocol π_1 in OAuth 2.0 is a secure 3P-ASD protocol with *insec* = {*state, code*} and *sec* = {*code, access_token*}, where code is empty for the implicit mode; and 2) the channel establishment protocol is a secure ACCE protocol.

Theorem 1. If the protocol π_1 is 3P-ASD secure, and the channel establishment protocol CHAN in π_2 and π_3 is ACCE secure, then OAuth 2.0 is a secure OAuth protocol for authorization, authentication, integrity for authorization and integrity for authentication. Formally, for any PPT adversary $\mathcal{A}$, we have

$\mathsf{Adv}^{\mathsf{Author}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + \mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}} + n_c n_\pi.\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}$,

$\mathsf{Adv}^{\mathsf{Authen}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + \mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}} + n_u n_\pi.\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}$,

$\mathsf{Adv}^{\mathsf{IntAuthor}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + 2n_c n_\pi.\mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}$,

$\mathsf{Adv}^{\mathsf{IntAuthen}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + 2n_c n_\pi.\mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}$.

Proof. The proof of this theorem is given in Appendix A.

Then we prove the security of OAuth 2.0 authorization code mode (Theorem 2). Firstly we prove the π_1 part in OAuth 2.0 authorization code is a secure 3P-ASD protocol (Lemma 1).

Lemma 1. The protocol π_1 in OAuth 2.0 authorization code mode is a secure 3P-ASD protocol. Formally, for any PPT adversary $\mathcal{A}$, we have

$\mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} \leq 5\mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth}_1,\mathcal{A}} + n_c \cdot \mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth}_2,\mathcal{A}}$,

$\mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} + n_u n_\pi \cdot (\mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}} + 4\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}})$,

$\mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} + n_u n_\pi \cdot \mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}$.

Proof. The proof of this lemma is given in Appendix B.

Theorem 2. If the channel establishment protocol CHAN is ACCE secure, and authentication schemes $auth_1$ and $auth_2$ are SUF-CMA secure, then OAuth 2.0 authorization code mode is a secure OAuth protocol for authorization, authentication, integrity for authorization and integrity for authentication.

Proof. Since π_1 part in OAuth 2.0 authorization code mode is 3P-ASD secure (Lemma 1), according to Theorem 1, OAuth 2.0 authorization code mode is a secure OAuth protocol. ■

Then we prove the security of OAuth 2.0 implicit mode (Theorem 3). Firstly we prove the π_1 part in OAuth 2.0 implicit code is a secure 3P-ASD protocol (Lemma 2).

Lemma 2. The protocol π_1 in OAuth 2.0 implicit mode is a secure 3P-ASD protocol. Formally, for any PPT adversary $\mathcal{A}$, we have

$\mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} \leq 3\mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth}_1,\mathcal{A}}$,

$\mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} + n_u n_\pi \cdot (\mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}} + 2\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}})$,

$\mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} + n_u n_\pi \cdot \mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}$.

Proof. The proof of this lemma is given in Appendix C.

Theorem 3. If the channel establishment protocol CHAN is ACCE secure, and the authentication scheme $auth_1$ is SUF-CMA secure, then OAuth 2.0 implicit mode is a secure OAuth protocol for authorization, authentication, integrity for authorization and integrity for authentication.

Proof. Since π_1 part in OAuth 2.0 implicit mode is 3P-ASD secure (Lemma 2), according to Theorem 1, OAuth 2.0 implicit mode is a secure OAuth protocol. ■

7. Conclusion

In this paper, we carried out the first computationally sound security analysis of OAuth 2.0. Specifically, we developed a sufficiently rich three-party security model for OAuth protocols, covering all kinds of authorization flows, providing reasonably strong security guarantees and moreover capturing various web features. We proved the security of two main modes of OAuth 2.0 via modular approach, which further confirmed the sound design of OAuth protocols. We also identified several known attacks against both original OAuth 2.0 in our security model. In addition, we introduced a new primitive, the three-party authenticated secret distribution protocol, which serves as a basis for OAuth protocols and would be of independent value.

Future work includes studying the generic composability of the 3P-ASD primitive with arbitrary tasks that use the issued secrets. We also hope our modular approach can shed light on practical issues in and facilitate the design of relatively complex security protocols.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grants 61572485, 61802021, 61802270 and 61802376.

References

[1] D. Hardt, "The OAuth 2.0 authorization framework," 2012, https://tools.ietf.org/html/rfc6749.

[2] N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "Openid connect core 1.0 incorporating errata set 1," *The OpenID Foundation, specification*, 2014.

[3] E. Y. Chen, Y. Pei, S. Chen, Y. Tian, R. Kotcher, and P. Tague, "OAuth demystified for mobile application developers," in *ACM CCS*. ACM, 2014, pp. 892–903.

[4] W. Li and C. J. Mitchell, "Security issues in oauth 2.0 sso implementations," in *ISC'14*. Springer, 2014, pp. 529–541.

[5] E. Shernan, H. Carter, D. Tian, P. Traynor, and K. Butler, "More guidelines than rules: Csrf vulnerabilities from noncompliant oauth 2.0 implementations," in *DIMVA*. Springer, 2015, pp. 239–260.

[6] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *ACM CCS*. ACM, 2008, pp. 75–88.

[7] C. Bansal, K. Bhargavan, A. Delignat-Lavaud, and S. Maffeis, "Discovering concrete attacks on website authorization by formal analysis," *J. Comput. Secur.*, vol. 22, no. 4, pp. 601–657, 2014.

[8] D. Fett, R. Küsters, and G. Schmitz, "A comprehensive formal security analysis of oauth 2.0," in *ACM CCS*. ACM, 2016, pp. 1204–1215.

[9] M. McGloin and P. Hunt, "Oauth 2.0 threat model and security considerations," 2013, https://tools.ietf.org/html/rfc6819.

[10] S. Pai, Y. Sharma, S. Kumar, R. M. Pai, and S. Singh, "Formal verification of oauth 2.0 using alloy framework," in *CSNT*. IEEE, 2011, pp. 655–659.

[11] A. Kumar, "Using automated model analysis for reasoning about security of web protocols," in *ACSAC*. ACM, 2012, pp. 289–298.

[12] K. S. Jayasri, K. P. Jevitha, and B. Jayaraman, "Verification of oauth 2.0 using uppaal," in *Social Transformation – Digital Way, CSI 2018*. Springer, 2018, pp. 58–67.

[13] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inform. Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[14] C. Brzuska, H. Jacobsen, and D. Stebila, "Safely exporting keys from secure channels," in *EUROCRYPT*. Springer, 2016, pp. 670–698.

[15] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu, "Multiple handshakes security of TLS 1.3 candidates," in *S&P*. IEEE, 2016, pp. 486–505.

[16] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "On the security of TLS-DHE in the standard model," in *CRYPTO*. Springer, 2012, pp. 273–293.

[17] C. Brzuska, N. P. Smart, B. Warinschi, and G. J. Watson, "An analysis of the EMV channel establishment protocol," in *ACM CCS*. ACM, 2013, pp. 373–386.

[18] M. Fischlin and F. Günther, "Multi-Stage Key Exchange and the Case of Google's QUIC Protocol," in *ACM CCS*. ACM, 2014, pp. 1193–1204.

[19] R. Lychev, S. Jero, A. Boldyreva, and C. Nita-Rotaru, "How secure and quick is QUIC? Provable security and performance analyses," in *S&P*. IEEE, 2015, pp. 214–231.

[20] C. Brzuska and H. Jacobsen, "A modular security analysis of EAP and IEEE 802.11," in *PKC*. Springer, 2017, pp. 335–365.

[21] T. Dierks, "The Transport Layer Security (TLS) protocol version 1.2," 2008, https://tools.ietf.org/html/rfc5246.

[22] C. Brzuska, M. Fischlin, B. Warinschi, and S. C. Williams, "Composability of Bellare-Rogaway key exchange protocols," in *ACM CCS*. ACM, 2011, pp. 51–62.

[23] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *CRYPTO*. Springer, 1993, pp. 232–249.

[24] E. Hammer-Lahav, "The OAuth 1.0 protocol," 2010, https://tools.ietf.org/html/rfc5849.

[25] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, "A cryptographic analysis of the TLS 1.3 handshake protocol candidates," in *ACM CCS*. ACM, 2015, pp. 1197–1210.

[26] D. Fett, R. Küsters, and G. Schmitz, "An expressive model for the web infrastructure: Definition and application to the browserid sso system," in *S&P*. IEEE, 2014, pp. 673–688.

[27] ——, "The web sso standard openid connect: In-depth formal security analysis and security guidelines," in *IEEE Computer Security Foundations Symposium (CSF)*. IEEE, 2017, pp. 189–202.

[28] D. Fett, P. Hosseyni, and R. Küsters, "An extensive formal security analysis of the openid financial-grade api," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 453–471.

[29] J. Schwenk and D. Stebila, "A reduction-based proof for authentication and session key security in 3-party kerberos," 2019, http://eprint.iacr.org/2019/777.

[30] J. Herzog, "A computational interpretation of dolev-yao adversaries," *Theoretical Computer Science*, vol. 340, no. 1, pp. 57–81, 2005.

[31] B. Blanchet, "Security protocol verification: Symbolic and computational models," in *International Conference on Principles of Security and Trust(POST)*. Springer, 2012, pp. 3–29.

[32] S. Chari, C. S. Jutla, and A. Roy, "Universally composable security analysis of oauth v2.0." 2011, http://eprint.iacr.org/2011/526.

[33] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *FOCS*. IEEE, 2001, pp. 136–145.

[34] M. Bellare and P. Rogaway, "Provably secure session key distribution: the three party case," in *STOC*. ACM, 1995, pp. 57–66.

[35] K. Bhargavan, I. Boureanu, A. Delignat-Lavaud, P.-A. Fouque, and C. Onete, "A formal treatment of accountable proxying over tls," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 799–816.

[36] R. Wang, Y. Zhou, S. Chen, S. Qadeer, D. Evans, and Y. Gurevich, "Explicating sdks: Uncovering assumptions underlying secure authentication and authorization," in *USENIX Security*. USENIX, 2013, pp. 399–414.

[37] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content," Tech. Rep., 2014.

[38] R. Canetti and H. Krawczyk, "Security analysis of ike's signature-based key-exchange protocol," in *CRYPTO*. Springer, 2002, pp. 143–161.

[39] J. Eisinger and E. Stark, "Referrer policy–editors draft, 28 march 2016. w3c." https://w3c.github.io/webappsec-referrer-policy/.

[40] M. Jones, J. Bradley, and N. Sakimura, "Oauth 2.0 mix-up mitigation–draft-ietf-oauth-mix-up-mitigation-01. ietf. jul. 2016," https:tools.ietf.org/html/draft-ietf-oauth-mix-up-mitigation-01.

Appendix A. Security Proof of Theorem 1

In this proof, the common game-hopping techniques are used to bound the adversary's advantage. We accomplish the proof by bounding $\mathsf{Adv}^{\mathsf{Author}}_{\mathsf{OAuth},\mathcal{A}}$, $\mathsf{Adv}^{\mathsf{Authen}}_{\mathsf{OAuth},\mathcal{A}}$, $\mathsf{Adv}^{\mathsf{IntAuthor}}_{\mathsf{OAuth},\mathcal{A}}$ and $\mathsf{Adv}^{\mathsf{IntAuthen}}_{\mathsf{OAuth},\mathcal{A}}$ in Game A, B, C and D, respectively.

Game A: Authorization security of OAuth 2.0

Game A.0. This is the real authorization game. Hence:

$$\mathsf{Adv}^{\mathsf{Author}}_{\mathsf{OAuth},\mathcal{A}} = \mathsf{Adv}^{\mathsf{G_{A.0}}}_{\mathsf{OAuth},\mathcal{A}}.$$

Game A.1. The challenger in this game proceeds as before, but it aborts if there exists any client oracle π^s_C accepts maliciously in π_2. Note that π_2 is established by the ACCE protocol CHAN, and we can construct an adversary $\mathcal{B}$ in the ACCE security experiment of CHAN to simulate Game A.0 for $\mathcal{A}$. Particularly, $\mathcal{B}$ simulates π_1 for $\mathcal{A}$ itself, forwards all messages to its ACCE oracle of CHAN and uses the responses to simulate π_2 for $\mathcal{A}$, and then uses $\mathcal{A}$'s advantage to break the ACCE–auth security of CHAN in π_2. We observe that $\mathcal{B}$ can provide a perfect simulation for $\mathcal{A}$ by its own ACCE oracle. If the client oracle π^s_C accepts maliciously in π_2 (also in CHAN), B will win in its own experiment. Hence,

$$\mathsf{Adv}^{\mathsf{G_{A.0}}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{A.1}}}_{\mathsf{OAuth},\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Game A.2. In this game, the challenger aborts if $\mathcal{A}$ successfully outputs the resource m^* satisfying Definition 12, where the server oracle π^t_S has issued m^* to the client oracle π^s_C in the OAuth instance initiated by the user oracle π^r_U, and we use Pr[Success] to denote the success probability. Hence,

$$\mathsf{Adv}^{\mathsf{G_{A.2}}}_{\mathsf{OAuth},\mathcal{A}} = 0.$$

We use Q to denote the event $\mathcal{A}$ has broken the confidentiality security of π_1 as depicted in Figure 4, and $\overline{Q}$ to denote the complementary event. Then we have

$$\Pr[\text{Success}] = \Pr[\text{Success}/Q]\cdot\Pr[Q] + \Pr[\text{Success}/\overline{Q}]\cdot\Pr[\overline{Q}].$$

Then there exists an algorithm can distinguish these two games (i.e., Game A.1 and Game A.2) if the challenger aborts in this game, and the probability of abort event is bounded by Pr[Success].

In fact, note that due to the security of π_1, the adversary $\mathcal{A}$ cannot obtain and output the correct secrets from π_1 for the later user in π_2, namely $\Pr[Q]$ is bounded by $\mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}}$ and thus $\Pr[\text{Success}/Q]\cdot\Pr[Q] \leq \mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}}$.

As a result, we argue that $\mathcal{A}$ can only succeed when $\overline{Q}$ occurs, which implies only π^s_C can obtain the correct $\{$*code, access_token*$\}$ and generate the ciphertext in π_2. We will prove that if $\mathcal{A}$ successfully outputs m^*, we can construct an adversary $\mathcal{B}$ to break the channel privacy of CHAN in π_2.

First, $\mathcal{B}$ guesses indices $<C, s>$ for $C \in \mathcal{C}$ and $s \in [1, n_\pi]$, where m^* is issued to π^s_C. If such indices do not exist, $\mathcal{B}$ aborts. According to Game A.1, there must exist one unique server oracle π^t_S to which π^s_C has a matching conversation in π_2. Then $\mathcal{B}$ simulates Game A.1 as the challenger for $\mathcal{A}$. In particular, it simulate the π_1 part of OAuth by creating messages itself since $\mathcal{A}$ cannot employ π_1 to win, while in π_2 it responds to the queries to π^t_S and π^s_C using its own ACCE experiment. Specifically, $\mathcal{B}$ makes the query $\mathsf{Encrypt}(\pi^t_S, m_0, m_1)$ to its ACCE experiment, where m_0 and m_1 are two random resource values, and the returned ciphertext would be used as the messages 11) in Figure 2. If $m^* = m_0$ then $\mathcal{B}$ outputs $b' = 0$; else if $m^* = m_1$ then $\mathcal{B}$ outputs $b' = 1$; otherwise $\mathcal{B}$ randomly outputs a bit b'. Apparently, if $\mathcal{A}$ can output the correct m^*, $\mathcal{B}$ can win in its ACCE-priv experiment of CHAN. A similar simulation process can be referred to the proof of Lemma 1 in [20]. Hence:

$$\Pr[\text{Success}/\overline{Q}]\cdot\Pr[\overline{Q}] \leq n_c n_\pi.\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}.$$

We observe that $\mathcal{B}$ can provide a perfect simulation for $\mathcal{A}$ by its own ACCE oracle, and thus $\mathcal{A}$ cannot distinguish these two games as soon as the abort does not occur. Thus we have

$$\begin{aligned}\mathsf{Adv}^{\mathsf{G_{A.1}}}_{\mathsf{OAuth},\mathcal{A}} &\leq \mathsf{Adv}^{\mathsf{G_{A.2}}}_{\mathsf{OAuth},\mathcal{A}} + \Pr[\text{Success}] \\ &\leq \mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}} + n_c n_\pi.\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}.\end{aligned}$$

Combing the above probabilities yields the stated bound. ■

Game B: Authentication security of OAuth 2.0

Game B.0. This is the real authentication game. Hence:

$$\mathsf{Adv}^{\mathsf{Authen}}_{\mathsf{OAuth},\mathcal{A}} = \mathsf{Adv}^{\mathsf{G_{B.0}}}_{\mathsf{OAuth},\mathcal{A}}.$$

Game B.1. The challenger in this game proceeds as before, but it will abort if there exists an oracle that accepts maliciously in π_1 in the sense of Definition 8,

which means the entity authentication property of π_1 is broken. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{B}.0}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G}_{\mathsf{B}.1}}_{\mathsf{OAuth},\mathcal{A}} + \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}}.$$

Recall that entity authentication ensures messages indeed come from the party as it claims to be (c.f. Definition 8), and thus the honest party cannot be forged. Then entity authentication security of π_1 guarantees that the adversary cannot forge the honest U to make C obtain the secret of U without the permission of U and in turn obtain the correct *cookie** from C.

Game B.2. The challenger in this game proceeds as before, but it aborts if there exists any user oracle accepts maliciously in π_3, and we can construct another algorithm to break the entity authentication security of CHAN in π_3 as discussed in the above Game A.1. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{B}.1}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G}_{\mathsf{B}.2}}_{\mathsf{OAuth},\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Game B.3. In this game, we add an abort rule. The challenger will abort if the adversary $\mathcal{A}$ successfully outputs the cookie information *cookie** satisfying Definition 13, where the client oracle π^s_C has issued *cookie** to the user oracle π^r_U using the server oracle π^t_S in the OAuth instance. We use Pr[Success] to denote the success probability. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{B}.3}}_{\mathsf{OAuth},\mathcal{A}} = 0.$$

Similarly, we use Q to denote the event $\mathcal{A}$ has broken the confidentiality security of π_1 as depicted in Figure 4 and $\overline{Q}$ to denote the complementary event. Then we have Pr[Success] = Pr[Success/Q]·Pr[Q] + Pr[Success/$\overline{Q}$]·Pr[$\overline{Q}$].

Then there exists an algorithm can distinguish these two games if the challenger aborts in this game, and the probability of abort event is bounded by Pr[Success].

In fact, due to the security of π_1, the adversary $\mathcal{A}$ cannot obtain and output the correct secrets from π_1 for the later use in π_3, namely Pr[Q] is bounded by $\mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}}$ and thus Pr[Success/Q]· Pr[Q] $\leq \mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}}$.

As a result, $\mathcal{A}$ can only succeed when $\overline{Q}$ occurs, which implies only π^s_C can obtain the correct {*code, access_token*} and generate the ciphertext in π_2. We will prove that if $\mathcal{A}$ successfully outputs *cookie**, we can construct an adversary $\mathcal{B}$ to break the channel privacy of CHAN in π_3. The reduction process is similar to that of the above Game A.2, namely $\mathcal{B}$ provides a perfect simulation for $\mathcal{A}$ by its own ACCE oracle, except that $\mathcal{B}$ firstly guesses indices $<U, r>$ for $U \in \mathcal{U}$ and $r \in [1, n_\pi]$ which has a matching conversation to one unique client oracle π^s_C for $C \in \mathcal{C}$ and $s \in [1, n_\pi]$. Hence,

$$\text{Pr[Success/}\overline{Q}\text{]·Pr[}\overline{Q}\text{]} \leq n_u n_\pi.\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}.$$

And now we can conclude that

$$\begin{aligned}\mathsf{Adv}^{\mathsf{G}_{\mathsf{B}.2}}_{\mathsf{OAuth},\mathcal{A}} &\leq \mathsf{Adv}^{\mathsf{G}_{\mathsf{B}.3}}_{\mathsf{OAuth},\mathcal{A}} + \text{Pr[Success]} \\ &\leq \mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}} + n_u n_\pi.\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}.\end{aligned}$$

Combing the above probabilities yields the stated bound. ■

Game C: Integrity for authorization of OAuth 2.0

Game C.0. This is the real integrity for authorization game:

$$\mathsf{Adv}^{\mathsf{IntAuthor}}_{\mathsf{OAuth},\mathcal{A}} = \mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.0}}_{\mathsf{OAuth},\mathcal{A}}.$$

Game C.1. The challenger in this game proceeds as before, but it will abort if there exists an oracle accepting maliciously in π_1 in the sense of Definition 8, which means the entity authentication property of π_1 is broken. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.0}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.1}}_{\mathsf{OAuth},\mathcal{A}} + \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}}.$$

Similar to the above Game B.1, entity authentication security of π_1 guarantees that the adversary cannot forge the honest uses and thus the client C cannot obtain the secret of U without the permission of U.

Game C.2. In this game, the challenger will abort if a user U has initiated one OAuth instance with C and S, however, C obtains the secret of another user U' in π_1 in the sense of Definition 10, which means the integrity property of π_1 is broken. Thus we have,

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.1}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.2}}_{\mathsf{OAuth},\mathcal{A}} + \mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}}.$$

Game C.3. In this game, the challenger will abort if there exists any client oracle accepts maliciously in π_2, and we can construct another algorithm to break the entity authentication security of CHAN in π_2 as discussed in the above Game A.1. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.2}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.3}}_{\mathsf{OAuth},\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Game C.4. In this game, the challenger will abort if the adversary breaks the integrity security by satisfying condition 1) (resp. condition 2)) of Definition 14. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.4}}_{\mathsf{OAuth},\mathcal{A}} = 0.$$

Note that now the only remaining ways to succeed for $\mathcal{A}$ is to impersonate π^s_C and forge a valid ciphertext for *access_token* of U (resp. another user U') in 10) or to impersonate π^t_S and forge a valid ciphertext for the protected resource of U (resp. another user U') in 11) of Figure 2.

Now we can construct an adversary $\mathcal{B}$ in the ACCE–int security experiment of CHAN, which simulates the π_1 for $\mathcal{A}$ itself while simulates π_2 by querying its Encrypt and Decrypt oracle as in the above Game A.2, and then use $\mathcal{A}$'s advantage to break the ACCE–int security of CHAN in π_2. First, $\mathcal{B}$ guesses indices $<C, s>$ for $C \in \mathcal{C}$ and $s \in [1, n_\pi]$ such that $\mathcal{A}$ can win the game using the oracle π^s_C, and aborts if such indices do not exist. According to Game C.3, there must be one unique server oracle π^t_S to which the client oracle π^s_C has a matching conversation. If $\mathcal{A}$ outputs a valid ciphertext c, $\mathcal{B}$ will forge a valid ciphertext c and win in its own experiment.

We observe that $\mathcal{B}$ can provide a perfect simulation for $\mathcal{A}$ by its own ACCE oracle, and thus $\mathcal{A}$ cannot distinguish these two games as soon as the abort does not occur. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.3}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G}_{\mathsf{C}.4}}_{\mathsf{OAuth},\mathcal{A}} + 2n_c n_\pi.\mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}$$

Combing the above probabilities yields the stated bound. ■

Game D: Integrity for authentication of OAuth 2.0.

The proof is similar to Game C. In brief, the adversary can win the integrity for authorization game by one of three ways, i.e., breaking the entity authentication goal of π_1, breaking the integrity goal of π_1, or breaking the ACCE-int goal of CHAN between S and C in π_3. The first four games, Game D.0, Game D.1, Game D.2, and Game D.3 are similar to the above Game C.0, Game C.1, Game C.2 and Game C.3, respectively. To avoid unnecessary repetition, we omit the details and just specify the last game Game D.4 as follows.

Game D.4. In this game, the challenger will abort if the adversary breaks the integrity security by satisfying condition 1) (resp. condition 2)) of Definition 15. Thus we have

$$\mathsf{Adv}^{\mathsf{G}_{\mathsf{D}.4}}_{\mathsf{OAuth},\mathcal{A}} = 0.$$

Note that now the only remaining ways to succeed for $\mathcal{A}$ is to impersonate π_C^s and forge a valid ciphertext for *access_token* of U (resp. another user U') in 12) or to impersonate π_S^t and forge a valid ciphertext for $user\_id$ of U (resp. another user U') in 13) of Figure 2.

Now we can construct an adversary $\mathcal{B}$ in the ACCE–int security experiment of CHAN, which simulates the π_1 for $\mathcal{A}$ itself while simulates π_3 by querying its Encrypt and Decrypt oracle as in the above Game A.2, and then use $\mathcal{A}$'s advantage to break the ACCE–int security of CHAN in π_3. Particularly, first, $\mathcal{B}$ guesses indices $<C, s>$ for $C \in \mathcal{C}$ and $s \in [1, n_\pi]$ such that $\mathcal{A}$ can win the game using the oracle π_C^s, and aborts if such indices do not exist. According to Game D.1, there must be one unique server oracle π_S^t to which the client oracle π_C^s has a matching conversation. If $\mathcal{A}$ outputs a valid ciphertext c, $\mathcal{B}$ will forge a valid ciphertext c and win in its own experiment.

We observe that $\mathcal{B}$ can provide a perfect simulation for $\mathcal{A}$ by its own ACCE oracle, and thus $\mathcal{A}$ cannot distinguish these two games as soon as the abort does not occur. Thus we have

$$\mathsf{Adv}^{\mathsf{G_{D.3}}}_{\mathsf{OAuth},\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{D.4}}}_{\mathsf{OAuth},\mathcal{A}} + 2n_c n_\pi . \mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}.$$

Combing the above probabilities yields the stated bound. ■

Theorem 1 now follows immediately from Games A-D.■

Appendix B. Security Proof of Lemma 1

We accomplish the proof by bounding $\mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}}$, $\mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}}$ and $\mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}}$ in Game A, Game B, and Game C, respectively.

Game A: Entity authentication of π_1

Game A.0. This is the real entity authentication game:

$$\mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}} = \mathsf{Adv}^{\mathsf{G_{A.0}}}_{\pi_1,\mathcal{A}}.$$

Game A.1. In this game, the challenger aborts if there exists any user oracle accepting with intended server partner but without a matching conversation at the server oracle, and we can construct another algorithm to break the entity authentication security of CHAN between them (referred to steps 3)-6) in Figure 2) as discussed in Game A.1 of Theorem 1. Hence

$$\mathsf{Adv}^{\mathsf{G_{A.0}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{A.1}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Game A.2. In this game, the challenger aborts if there exists any user oracle π_U^r accepting with intended client partner C but without a matching conversation at the client oracle.

Similar to Game A.1, the probability of the abort event can be bounded by the ACCE-auth security of CHAN (referred to the step 1), 2) and 7) in Figure 2). Hence:

$$\mathsf{Adv}^{\mathsf{G_{A.1}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{A.2}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Game A.3. In this game, the challenger aborts if there exists any client oracle π_C^s accepting with intended server partner S but without a matching conversation at the server oracle.

Similar to Game A.1, the probability of the abort event can be bounded by the ACCE-auth security of CHAN (referred to the step 8) and 9) in Figure 2). Hence:

$$\mathsf{Adv}^{\mathsf{G_{A.2}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{A.3}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Game A.4. In this game, the challenger will abort if there exists any server oracle π_S^t accepting with intended user partner U but without a matching conversation at the user oracle. It is split into two sub-games as follows.

Game A.4.1. In this sub-case, the challenger aborts if π_S^t accepts U's authentication (referred to the step 5) of Figure 2), however, the authentication information has not been generated by any honest sessions of U. We can bound the probability of the aborting event denoted by $\Pr[\mathsf{abort}_1]$, utilizing the advantage of another adversary $\mathcal{B}$ against the SUF-CMA security of the authentication scheme *auth*$_1$. First, $\mathcal{B}$ guesses a user U whose authentication information is forged by $\mathcal{A}$. If such a user does not exist, $\mathcal{B}$ aborts. Then $\mathcal{B}$ simulates Game A.3 as the challenger, except that $\mathcal{B}$ queries its own Auth oracle for generating the authentication information for U. Finally, $\mathcal{B}$ outputs what $\mathcal{A}$ outputs and will win if $\mathcal{A}$ wins. Hence

$$\Pr[\mathsf{abort}_1] \leq n_u \cdot \mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth}_1,\mathcal{A}}.$$

Game A.4.2. If the challenger does not abort in the Game A.4.1, there must be an honest user U as the partner of π_S^t, and moreover, the same session key is shared between π_U^r and π_S^t. In this sub-case, we can bound the probability of the aborting event denoted by $\Pr[\mathsf{abort}_2]$, i.e., the probability of π_S^t accepting without matching conversations, by the ACCE-auth security of CHAN (referred to the step 3)–6) in Figure 2) as discussed in the above Game A.1 of Theorem 1. Hence:

$$\Pr[\mathsf{abort}_2] \leq \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Collecting bounds from the above two sub-games yields:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{G_{A.3}}}_{\pi_1,\mathcal{A}} &\leq \mathsf{Adv}^{\mathsf{G_{A.4}}}_{\pi_1,\mathcal{A}} + \Pr[\mathsf{abort}_1] + \Pr[\mathsf{abort}_2] \\ &\leq \mathsf{Adv}^{\mathsf{G_{A.4}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth}_1,\mathcal{A}}.\end{aligned}$$

Game A.5. In this game, the challenger aborts if there exists any server oracle π_S^t accepting with intended client partner C but without a matching conversation at the client oracle. It is also split into two sub-games as the above Game A.4.

Game A.5.1 In this sub-case, the challenger aborts if π_S^t accepts C's authentication (referred to the step 8) of Figure 2), however, the authentication information has not been generated by any honest sessions of C. We can bound the probability of the aborting event denoted by $\Pr[\mathsf{abort}_1]$, utilizing the advantage of another adversary $\mathcal{B}$ against the SUF-CMA security of the authentication scheme *auth*$_2$. The reduction process is the same as that of the above Game A.4.1 and thus would be omitted here. Hence:

$$\Pr[\mathsf{abort}_1] \leq n_u \cdot \mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth}_2,\mathcal{A}}.$$

Game A.5.2. If the challenger does not abort in the Game A.5.1, there must be an honest client C as the partner of π_S^t, and moreover, the same session key is shared between π_C^s and π_S^t. In this sub-case, we can bound the probability of the aborting event denoted by $\Pr[\mathsf{abort}_2]$, i.e., the probability of π_S^t accepting without matching conversations, by the ACCE-auth security of CHAN (referred to the step 8)–9) in Figure 2) as discussed in the above Game A.1 of Theorem 1. Hence:

$$\Pr[\mathsf{abort}_2] \leq \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}}.$$

Collecting bounds of the above two sub-cases yields:

$$\begin{aligned}\mathsf{Adv}^{\mathsf{G_{A.4}}}_{\pi_1,\mathcal{A}} &\leq \mathsf{Adv}^{\mathsf{G_{A.5}}}_{\pi_1,\mathcal{A}} + \Pr[\mathsf{abort}_1] + \Pr[\mathsf{abort}_2] \\ &\leq \mathsf{Adv}^{\mathsf{G_{A.4}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-ea}}_{\mathsf{CHAN},\mathcal{A}} + n_u \cdot \mathsf{Adv}^{\mathsf{SUF-CMA}}_{\mathsf{auth}_2,\mathcal{A}}.\end{aligned}$$

If the challenger does not abort in Game A.5, we have

$$\mathsf{Adv}^{\mathsf{G_{A.5}}}_{\pi_1,\mathcal{A}} = 0.$$

Combing the above probabilities yields the stated bound. ■

Game B: Confidentiality of π_1

We will bound the advantage that the adversary obtains at least one of elements in the *sec** entry from a fresh session.

Game B.0. This is the real confidentiality game of π_1, hence

$$\mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}} = \mathsf{Adv}^{\mathsf{G_{B.0}}}_{\pi_1,\mathcal{A}}.$$

Game B.1. The challenger in this game proceeds as before, but it aborts if there exists an oracle accepting maliciously in π_1 in the sense of Definition 8, which means the entity authentication property of π_1 is broken. Thus we have

$$\mathsf{Adv}^{\mathsf{G_{B.0}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{B.1}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}}.$$

Game B.2. The challenger guesses indices $<U, r>$ for $U \in \mathcal{U}$ and $r \in [1, n_\pi]$ such that *sec** is issued in an instance initiated by π^r_U, and aborts if such indices do not exist. Hence:

$$\mathsf{Adv}^{\mathsf{G_{B.1}}}_{\pi_1,\mathcal{A}} \leq n_u n_\pi \cdot \mathsf{Adv}^{\mathsf{G_{B.2}}}_{\pi_1,\mathcal{A}}.$$

According to the Game B.1, there must exist a unique client oracle π^s_C and a unique server oracle π^t_S to which π^r_U has a matching conversation in the respective sub-session.

Game B.3. In this game, the challenger will abort if the adversary outputs the correct *sec** satisfying Definition 9. According to the above games, the secret *sec** is issued by π^t_S to π^s_C under the permission of π^r_U. Moreover, recall that the entity authentication of π_1 guarantees the honest parties cannot be forged, and thus

(1) the adversary cannot impersonate U to S and then obtain the *code* sent to π^r_U in the step 6) of Figure 2.
(2) the adversary cannot impersonate C to U and then obtain the *code* sent to π^s_C in the step 7) of Figure 2.

First, we bound the adversary's advantage to obtain *code* from the ciphertext 6), 7) or 8) in Figure 2. by the ACCE-priv security of CHAN in 6), 7) or 8). Particularly, if the adversary $\mathcal{A}$ outputs the correct *code* then we can construct another algorithm to break the ACCE-priv security of CHAN in 6), 7) or 8), where the reduction process is similar to that of the Game A.2 in Theorem 1 and would be omitted here.

Second, we bound the adversary's advantage to receive *code* from π^s_C in 8) by the ACCE-int security of CHAN protocol between π^r_U and π^s_C (recall that the mix-up attack occurs in this case). Specifically, since only π^r_U receives the *code* in step 6), then it must be π^r_U that establishes the CHAN channel with π^s_C for 1), 2) and 7) in Figure 2. Thus if the server S's identity information protected in 7) is tampered by $\mathcal{A}$ with an identity controlled by itself, then $\mathcal{A}$ forges a valid ciphertext and thus the ACCE-int security of CHAN between π^r_U and π^s_C is broken. The reduction details are similar to that of Game C.4 of Theorem 1 and omitted here.

Finally, we bound the adversary's advantage to obtain the correct *access_token*. Without *code*, the adversary cannot receive the correct *access_token* from 9) unless he can decrypt the ciphertext in 9), by which we can construct another algorithm to break the ACCE-priv security of CHAN between π^s_C and π^t_S as proved in Game A.2 of Theorem 1 and the proof details are omitted here. Hence:

$$\mathsf{Adv}^{\mathsf{G_{B.2}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{B.3}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}} + 4\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}.$$

Up to now, all the ways for the adversary to obtain *sec** have been considered. If the challenger does not abort, then

$$\mathsf{Adv}^{\mathsf{G_{B.3}}}_{\pi_1,\mathcal{A}} = 0.$$

Combing the above probabilities yields the stated bound. ■

Game C: Integrity of π_1

We will bound the advantage that the adversary breaks the integrity security.

Game C.0. This is the real integrity game of π_1, hence

$$\mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}} = \mathsf{Adv}^{\mathsf{G_{C.0}}}_{\pi_1,\mathcal{A}}.$$

Game C.1. The challenger in this game proceeds as before, but it aborts if there exists an oracle accepting maliciously in π_1 in the sense of Definition 8, which means the entity authentication property of π_1 is broken. Thus we have

$$\mathsf{Adv}^{\mathsf{G_{C.0}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{C.1}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{EA}}_{\pi_1,\mathcal{A}}.$$

Game C.2. The challenger guesses indices $<U, r>$ for $U \in \mathcal{U}$ and $r \in [1, n_\pi]$ such that the client obtains the secret of another user U' from the instance initiated by π^r_U, and aborts if such indices do not exist. Hence:

$$\mathsf{Adv}^{\mathsf{G_{C.1}}}_{\pi_1,\mathcal{A}} \leq n_u n_\pi . \mathsf{Adv}^{\mathsf{G_{C.2}}}_{\pi_1,\mathcal{A}}.$$

According to the Game C.1, there must exist a unique client oracle π^s_C and a unique server oracle π^t_S to which π^r_U has a matching conversation in the respective sub-session.

Game C.3. In this game, the challenger aborts if π^s_C obtains the secret *sec* of another user U' from the instance with π^r_U and π^t_S. Hence:

$$\mathsf{Adv}^{\mathsf{G_{C.3}}}_{\pi_1,\mathcal{A}} = 0.$$

Now we bound the probability of the abort event. We first stress that the adversary cannot employ the OriginLeak and MalTransfer queries to mount the state leak attack. Specifically, as discussed in Section 5.3, the countermeasure [39] is to block the referrer header entirely or strip it down to the origin of the respective page. As a result, the OriginLeak query cannot leak any private information including the state, and the adversary cannot use the MalTransfer to send a correct *state* to the client. Moreover, the correct *state* can only be used once in the message 7).

Also note that 6), 8) and 9) in Figure 2 cannot be forged owing to the entity authentication security of π_1, then the only remaining way to win for $\mathcal{A}$ is to modify the ciphertext 7) into a valid ciphertext of *code*$'$ for U'. In this case the ACCE-int security of CHAN protocol between π^s_C and π^r_U is broken since $\mathcal{A}$ has forged a valid ciphertext, the reduction details of which are similar to the Game C.4 of Theorem 1 and thus omitted here. Thus we have

$$\mathsf{Adv}^{\mathsf{G_{C.2}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{C.3}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}.$$

Combing the above probabilities yields the stated bound. ■

Lemma 1 now follows immediately from Games A-C. ■

Appendix C. Security Proof of Lemma 2

We accomplish the proof by bounding $\mathsf{Adv}^{\mathsf{Match}}_{\pi_1,\mathcal{A}}$, $\mathsf{Adv}^{\mathsf{Con}}_{\pi_1,\mathcal{A}}$ and $\mathsf{Adv}^{\mathsf{Int}}_{\pi_1,\mathcal{A}}$ in Game A, Game B and Game C, respectively.

Game A: Entity authentication of π_1

The proof is quite same to that of Game A for Lemma 1, except that the Game A.3 and Game A.5 are removed due to no interaction between C and S in the implicit

mode. That is, we use Game A.0 to denote the real game, then we just need to consider the events that a user oracle accepts the server maliciously in sub-session between them (i.e., 3)-6) in Figure 3) in Game A.1, a user oracle accepts the client maliciously in sub-session between them (i.e.,1), 2) and 7) in Figure 3) in Game A.2 and the server oracle accepts the user maliciously in sub-session between them (i.e., 3)-6) in Figure 3) in Game A.3, and the corresponding reduction process and reduction bound are the same to that of Game A.0, Game A.1, Game A.2 and Game A.4 of Lemma 2 respectively. To avoid unnecessary repetitions, we omit the details here.

Game B: Confidentiality of π_1

The proof is similar to that of Game B for Lemma 1, except that we only need to consider the leakage of *access_token* rather than both of *access_token* and *code*. Thus **Game B.0.–Game B.2.** are the same to that of Game B for Lemma 1 and are omitted here, while the **Game B.3.** is as follows:

Game B.3. In this game, the challenger will abort if the adversary outputs the correct sec^* satisfying the condition of Definition 9. According to the above games, the secret sec^* is issued by π_S^t to π_C^s under the permission of π_U^r. Moreover, recall that the entity authentication guarantees the honest parties cannot be forged, thus the entity authentication security of π_1 guarantees that

(1) the adversary cannot impersonate U to S and obtain the *access_token* sent to π_U^r in the step 6) of Figure 3.
(2) the adversary cannot impersonate C to U and obtain the *access_token* sent to π_C^s in the step 7) of Figure 3.

Then if the adversary $\mathcal{A}$ can outputs the correct *access_token*, then we can construct another adversary $\mathcal{B}$ that simulates for $\mathcal{A}$ by its ACCE oracle to break the ACCE-priv security of CHAN in the step 6) or 7) in Figure 3, the reduction details of which are similar to that of Game A.2 in Theorem 1 and omitted here. Hence:

$$\mathsf{Adv}^{\mathsf{G_{B.2}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{B.3}}}_{\pi_1,\mathcal{A}} + 2\mathsf{Adv}^{\mathsf{ACCE-priv}}_{\mathsf{CHAN},\mathcal{A}}.$$

Up to now, all the ways for the adversary to obtain sec^* have been considered. If the challenger does not abort, then

$$\mathsf{Adv}^{\mathsf{G_{B.3}}}_{\pi_1,\mathcal{A}} = 0.$$

Combing the above probabilities yields the stated bound. ■

Game C: Integrity of π_1

The proof of Game C is similar to that of Game C for Lemma 1, except that *code* entry is empty. Thus **Game C.0.–Game C.2.** are the same to that of Game C for Lemma 1 and are omitted here, while the **Game C.3.** is as follows:

Game C.3. In this game, the challenger aborts if π_C^s obtains the secret *sec* of another user U' from the instance with π_U^r and π_S^t. Hence:

$$\mathsf{Adv}^{\mathsf{G_{C.3}}}_{\pi_1,\mathcal{A}} = 0.$$

Now we bound the probability of the abort event. We first stress the adversary cannot employ the OriginLeak and MalTransfer queries to mount the state leak attack, due to the countermeasure [39] as discussed in Game C.3 of Lemma 1.

Also note that 6) in Figure 3 cannot be forged owing to the entity authentication security of π_1, then the only remaining way to win for $\mathcal{A}$ is to modify the ciphertext 7) into a valid ciphertext of *access_token*$'$ for U'. In this case, the ACCE-int security of CHAN protocol between π_C^s and π_U^r since $\mathcal{A}$ has forged a valid ciphertext, the reduction retails of which are similar to the Game C.4 of Theorem 1 and thus omitted here. Thus we have

$$\mathsf{Adv}^{\mathsf{G_{C.2}}}_{\pi_1,\mathcal{A}} \leq \mathsf{Adv}^{\mathsf{G_{C.3}}}_{\pi_1,\mathcal{A}} + \mathsf{Adv}^{\mathsf{ACCE-int}}_{\mathsf{CHAN},\mathcal{A}}.$$

Combing the above probabilities yields the stated bound. ■

Lemma 2 now follows immediately from Games A-C. ■