Design, Formal Specification and Analysis of Multi-Factor Authentication Solutions with a Single Sign-On Experience

Giada Sciarretta[1,2](✉), Roberto Carbone[1], Silvio Ranise[1], and Luca Viganò[3]

[1] Security & Trust, FBK, Trento, Italy
{giada.sciarretta,carbone,ranise}@fbk.eu
[2] University of Trento, Trento, Italy
[3] King's College London, London, UK
luca.vigano@kcl.ac.uk

Abstract. Over the last few years, there has been an almost exponential increase of the number of mobile applications that deal with sensitive data, such as applications for e-commerce or health. When dealing with sensitive data, classical authentication solutions based on username-password pairs are not enough, and multi-factor authentication solutions that combine two or more authentication elements of different categories are required. Many different such solutions are available, but they usually cover the scenario of a user accessing web applications on their laptops, whereas in this paper we focus on native mobile applications. This changes the exploitable attack surface and thus requires a specific analysis. In this paper, we present the design, the formal specification and the security analysis of a solution that allows users to access different mobile applications through a multi-factor authentication solution providing a Single Sign-On experience. The formal and automated analysis that we performed validates the security goals of the solution we propose.

1 Introduction

Context and Motivations. Over the last few years, there has been an almost exponential increase of the number of *mobile applications* (or *apps*, for short) that deal with sensitive data, ranging from apps for e-commerce, banking and finance to apps for well-being and health. One of the main reasons behind such a success is that mobile apps considerably increase the portability and efficiency of online services. Banking apps allow users not only to check their account balances but also to move money and pay bills or friends [1]. Mobile health apps range from personal health records (PHR) to personal digital assistants using connected devices such as smartwatches and other body-worn devices or implants. As reported in [2], there are nowadays more than 100,000 mobile health apps on the market, a number that is increasing on a weekly basis.

However, also the reports on security and privacy issues in mobile apps are increasing on a weekly basis, bearing concrete witness to the fact that the

L. Bauer and R. Küsters (Eds.): POST 2018, LNCS 10804, pp. 188–213, 2018.
https://doi.org/10.1007/978-3-319-89722-6_8

management of sensitive data is often not properly taken into account by the developers of the apps. For example, the studies performed by He et al. [3] on free mobile health apps available on the Google Play store show that the majority of these apps send sensitive data in clear text and store it on third party servers that do not support the required confidentiality measures.

When dealing with sensitive data, classical authentication solutions based on username-password pairs are not enough. The "General Data Protection Regulation" [4] mandates that specific security measures must be implemented, including *multi-factor authentication*, a strong(er) authentication solution that combines two or more authentication elements of different categories (e.g., a password combined with a pin sent to a mobile device, or some biometric data). There are many alternative solutions on the market for providing multi-factor authentication. Examples are FIDO (Fast IDentity Online, https://fidoalliance.org), which enables mobile devices to act as U2F (Universal 2nd Factor) authentication devices over Bluetooth or NFC, and Mobile Connect (https://mobileconnect.io), which identifies users through their mobile phone numbers.

In addition to the establishment of high-level security for authentication solutions for mobile apps, it is essential to take the usability aspect into consideration. Monitoring apps often require a daily or even hourly use, but understandably users cannot be bothered by a long and complex authentication procedure each time they want to read or update their data, especially on mobile devices where the keyboard is small and sometimes uncomfortable to use. A better usability can be provided by supporting a *Single Sign-On (SSO) experience*, which allows users to access different, federated apps by performing a single login carried out with a selected identity provider (e.g., Facebook or Google). While the authentication session is valid, users can directly access all the apps in the federation, without having to enter their credentials again and again.

Contributions. In this paper, we present the design, the formal specification and the security analysis of a solution that allows users to access different mobile apps through a multi-factor authentication solution providing a SSO experience.

We focus on multi-factor authentication solutions that use *One Time Passwords (OTPs)*, which are passwords that are valid for a short time and can only be used once. We have selected OTP-generation approaches as they are commonly used to provide strong authentication and many alternative solutions (from physical to software tools) are available on the market. For instance, Google Authenticator is a mobile app that generates OTPs [5]. Like Google Authenticator, many of the OTP-generation solutions on the market are applicable only for web solutions and use mobile devices as an additional factor.

However, in the scenario considered in this paper, users are not accessing web apps on their laptops or desktop computers, but instead they are accessing native mobile apps. In relation to SSO and multi-factor authentication, web and mobile environments and channels guarantee different security properties, e.g., in web scenarios identity providers can authenticate service provider apps using shared secrets, but this is not possible for native mobile apps that are unable to keep values secret. This changes the exploitable attack surface and thus requires a specific analysis. To the best of our knowledge, the definition of a multi-factor authentication solution for native apps is still not well specified. Even if there are some solutions currently used, their security analyses have been performed informally or semi-formally at best, and without following a standardized formal procedure. This makes a comparison between the different solutions both complex and potentially misleading.

For the security assumptions and the design of a native SSO solution, our work is based on [6,7]. In this previous work, we presented a solution for native SSO and performed a semi-formal security analysis. In this work, we extend these studies by providing a multi-factor authentication solution and a formal analysis of the identified security goals.

Summarizing, our contributions are four-fold as we have

1. designed a multi-factor authentication solution that uses OTPs as an authenticator factor and provides a SSO experience for native apps;
2. provided a description of the proposed solution detailing the security and trust assumptions;
3. formally defined the security goals of our multi-factor authentication solution;
4. formally analyzed our solution by modeling the flow, assumptions and goals using a formal language (ASLan++) and model-checking the identified security goals with the SATMC tool.

The results of our analysis show that our solution behaves as expected.

Organization. Section 2 provides background on strong authentication solutions and SSO for native mobile apps, and on ASLan++ and SATMC. Section 3 describes the design of the proposed multi-factor authentication solution, discusses the peculiarities of a multi-factor authentication solution compared to a basic username-password authentication, and identifies the corresponding security assumptions and security goals. For concreteness, Sect. 4 describes our solution in the context of mHealth apps, and the solution is then formally analyzed using SATMC. Section 5 discusses related work and Sect. 6 draws conclusions.

2 Background

This section provides the basic notions required to understand the proposed design for a multi-factor authentication solution that supports a SSO experience and its security assessment. In Sect. 2.1, we describe the entities involved in a multi-factor authentication and SSO solution, discuss the different OTP-generation approaches, and identify the functional requirements of a native SSO solution. In Sect. 2.2, we provide useful background for our formal analysis.

2.1 Multi-factor Authentication and Native SSO

The entities involved in a multi-factor native SSO solution are: a *User (User)* that wants to access a native *Service Provider app (SP_C)*; an *Identity Provider* *server (IdP$_S$)* that manages the digital identities of the users and provides the multi-factor process; a *User Agent (UA)*, which could be a browser or a native app used to perform the multi-factor process between the SP_C and IdP_S. Optionally, the SP_C app could have a backend server (SP_S).

A multi-factor authentication solution augments the security of the basic username-password authentication by exploiting two or more authentication factors. In [8], it is defined as:

"a procedure based on the use of two or more of the following elements — categorised as knowledge, ownership and inherence: i) something only the user knows, e.g., static password, code, personal identification number; ii) something only the user possesses, e.g., token, smart card, mobile phone; iii) something the user is, e.g. biometric characteristic, such as a fingerprint. In addition, the elements selected must be mutually independent [...] at least one of the elements should be non-reusable and non-replicable".

The more factors are used during the authentication process, the more confidence a service has that the user is correctly identified.

There are many multi-factor techniques on the market. In this paper, we focus on a well-accepted solution that combines a PIN code ("something only the user knows") with the generation of an OTP using a software OTP generator ("something only the user possesses"). When an OTP-generation approach is used, a different password is generated for each authentication request and is valid only once, providing a fresh authentication property. Thus, compromising an old OTP does not have security consequences in the authentication process.

There exist many algorithms for generating OTPs and we can classify them into three main OTP-generation approaches:

- *Time synchronization:* the OTP is generated starting from a shared secret key (called *seed*) and the current time of the operation. IdP_S must validate this value: only OTPs that fall into a short temporal range are accepted.
- *Lamport's algorithm* [9]: the first OTP is generated from a seed value and each successor OTP value is based on the value of its predecessor. For example, if s is a seed value and $F(x)$ is a one-way function, we have the following OTPs: $o_1 = s, o_2 = F(o_1), o_3 = F(o_2), \dots o_n = F(o_{n-1})$. The last OTP, o_n, is stored on IdP_S. When a *User* wants to login, she sends o_{n-1} to the server, and the server applies the function F and checks that the result corresponds to the stored value. If the two values correspond, IdP_S authenticates *User* and updates the stored value with o_{n-1}. In the next login, *User* will use o_{n-2} and so on. After n logins, *User* has to change the seed value and calculate new OTP values.
- *Challenge/Response:* in the execution of this approach, IdP_S presents a "challenge" (e.g., a random number) and *User* answers with a valid "response", which is an OTP value calculated using a mathematical algorithm starting from the challenge.

Although our solution is parametric in the OTP-generation approach, in Sect. 4, we will detail and analyze the time synchronization approach in the context of a real-world scenario.

Native SSO protocols allow users to access multiple SP_C apps through a single authentication performed with an IdP_S. As identified in [6], the two requirements that we expect for a native SSO solution are: *(i)* the IdP user credentials can be used to gain access to several SP_C apps—this implies that a *User* does not need to have credentials with a SP_C to access it; *(ii)* if a *User* has already a login session with an IdP_S, then she can access new SP_C apps without re-entering her IdP credentials—only the *User* consent is required.

2.2 Formal Analysis: ASLan++ and SATMC

The use of formal languages and automatic tools for analyzing security protocols has allowed researchers to uncover a large number of vulnerabilities in protocols that had been thought to be, or even informally proved to be, secure. Famous examples range from protocols such as the Needham-Schroeder Public Key protocol to Kerberos or TLS (see [10] for details). These examples underline how the design of a protocol that requires specific security goals is not a simple task, as its security depends on several assumptions on trust and communication channels (e.g., the federation between the involved parties, and the transport protocol used in the message exchange). Several formal languages have been developed, all sharing the idea to extract from the protocol message flow a description of the entities involved, the exchanged messages and the channel assumptions. Formal protocol specifications are then given in input to automated tools that check the desired security goals of the protocol against realistic threat models.

In this paper, we use ASLan++ [11], the input specification language of the AVANTSSAR Platform [12]. ASLan++ is a high-level formal language that formalizes the interactions between the different protocol roles, where a role represents a sequence of operations (e.g., sending and receiving messages) that must be executed by the entity that plays that role. ASLan++ supports the specification of different channel assumptions and security goals, most notably different variants of authentication and confidentiality. In our analysis, we use SATMC [13], which is one of the model checkers of the AVANTSSAR platform. SATMC uses state-of-the-art SAT Solvers and allows for the specification of security goals written using the Linear Temporal Logic.

3 Description of Our *mID(OTP)* Solution

In this section, we present a mobile identity management solution that augments the security of the native SSO solution proposed in [6] by adding a multi-factor authentication based on the generation of OTPs. We called it *mID(OTP)* to highlight the dual goal that our solution pursued: *(i)* to establish a multi-factor authentication and *(ii)* to manage identities for native mobile apps, e.g., providing a SSO service. As we will describe, *mID(OTP)* is parametric on the OTP generation (i.e., it supports different OTP-generation approaches).

In the mobile context, two possible design choices are available: a *UA* could be played either by a browser (external or embedded in the SP_C app) or by a native app. In the design of *mID(OTP)*, we have preferred the latter choice, as a native app can be (easily) extended to support the generation of an authentication factor (e.g., by adding the code for a OTP generator or a library to process the user's fingerprint). In addition, as the *UA* is involved in the authentication phase with the IdP_S, it must be trusted in knowing the user's IdP credentials. Thus, we assume that this native app, called *IDOTP*, is released directly by the IdP_S.

mID(OTP) consists of three phases: *registration*, *activation* and *exploitation*, which we describe in the following subsections.

3.1 Registration and Activation Phases of *mID(OTP)*

The registration phase of *mID(OTP)* is performed by the SP_C developers and corresponds to the exchange of some information about SP_C, such as the package name and logo, together with its certificate fingerprint *key_hash* (i.e., the hash of the certificate of the app). Note that *key_hash* depends on the private key of the SP_C developer and is thus different for apps by different developers. The registration phase can be performed in different ways, e.g., entering the data into an online dashboard or via an email exchange. As a trust relationship between SP_C and IdP_S is established as result of the registration phase, it is important that the IdP_S validates the SP_C data and in some cases (e.g., when user personal or sensitive data are involved) a service-level agreement could be required as well.

The activation phase of *mID(OTP)* is performed by the *User* to configure the native app *IDOTP* on her smartphone. In addition to the procedure described in [6]—user login and release of a token (*token_IdP*) used (from here on) to identify the user session in place of the user credentials—at the end of the activation phase the *IDOTP* is configured to generate OTPs, usually requiring the creation of a PIN code for the future interactions.

Also the activation phase can be performed in different ways. As a multi-factor authentication is configured during this phase, it is essential to provide the *User* with an activation code—exchanged using a secure channel (e.g., after an in-person identification)—that she has to enter during the process.

3.2 Exploitation Phase of *mID(OTP)*

The exploitation phase of *mID(OTP)*, which is shown in Fig. 1, is performed every time the *User* accesses a SP_C that requires the multi-factor authentication and SSO experience offered by *IDOTP*. In Step S1, *User* opens the SP_C app that sends a request to SP_S including a session token *token_sync* (Step S2). SP_S checks the validity of *token_sync*. If *token_sync* has expired, SP_S sends an error message asking for a login to SP_C (Step S3), otherwise Step S7 is executed. If a login form is presented to *User*, she clicks the login button (Step A1) and SP_C sends a login request to *IDOTP* (Step A2). As a consequence, in Step A3 *IDOTP* reads the *key_hash* value of SP_C and in Step A4 sends a request to IdP_S asking the SP_C data. The received *key_hash* is used by IdP_S to validate

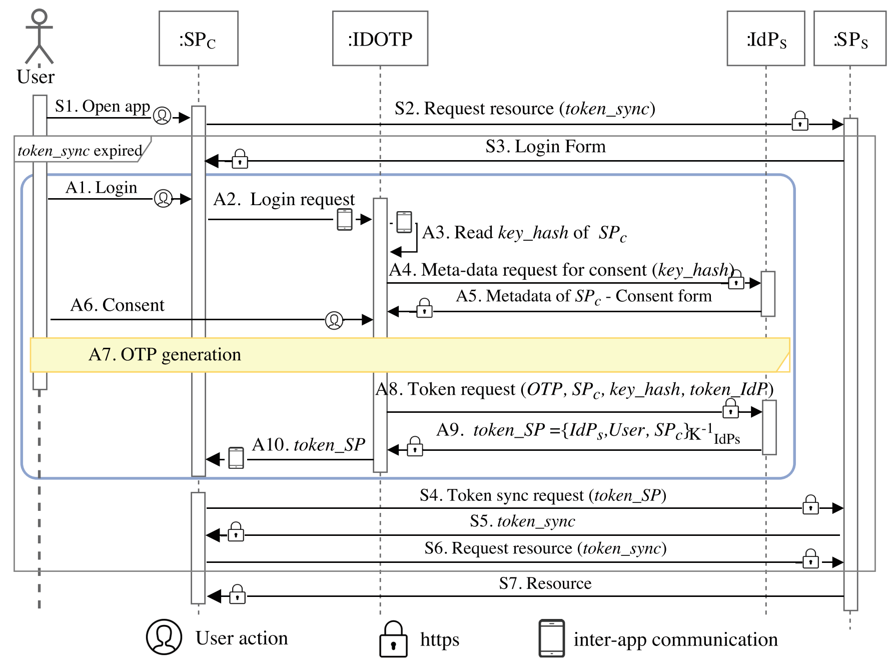

User
:SP_C
:IDOTP
:IdP_S
:SP_S
S1. Open app
S2. Request resource (token_sync)
token_sync expired
S3. Login Form
A1. Login
A2. Login request
A3. Read key_hash of SP_c
A4. Meta-data request for consent (key_hash)
A5. Metadata of SP_c - Consent form
A6. Consent
A7. OTP generation
A8. Token request (OTP, SP_c, key_hash, token_IdP)
A9. token_SP ={IdP_s,User, SP_c}K^-1_IdPs
A10. token_SP
S4. Token sync request (token_SP)
S5. token_sync
S6. Request resource (token_sync)
S7. Resource
User action
https
inter-app communication

Fig. 1. Exploitation phase of *mID(OTP)*.

the SP_C identity. If SP_C is valid, IdP_S returns to $IDOTP$ a consent containing the meta-data of SP_C (Step A5). In Step A6, $User$ checks whether SP_C is the app that she wants to access and decides whether to give her consent or not. If $User$ agrees, the OTP is generated following one of the approaches described in Sect. 2.1 (Step A7). Then, in Step A8, $IDOTP$ sends a token request to IdP_S including the OTP value, key_hash and $token\_IdP$, which corresponds to the user credentials entered during the activation phase. IdP_S checks the validity of OTP, key_hash and $token\_IdP$. If they are valid, a token ($token\_SP$) for the SP app is returned (Step A9). $token\_SP$ contains the identity of $User$, IdP_S and SP, and is digitally signed with $K^{-1}_{IdP_S}$, the private key of IdP_S. In Step A10, $IDOTP$ returns $token\_SP$ to SP_C as result of Step A2. To finalize the authentication, SP_C sends a token request to SP_S with $token\_SP$ (Step S4). SP_S checks the validity of $token\_SP$, and if it is valid, creates and sends to SP_C a token $token\_sync$ (Step S5). This token will be used by SP_C to synchronize user data in the future interactions, until its expiration. When SP_C needs to synchronize data, sends a request to the SP_S including $token\_sync$ (Step S6), and SP_S returns the requested resource to SP_C (Step S7).

We have labeled the steps with “S” and “A”. The S steps are related to the *SP* (but note that our representation is only an example and each *SP* could support different solutions). The A steps represent the steps related to the authentication solution. As the S steps can vary depending on the choices of the SP developers, in our analysis, we will focus on the A steps. Compared to the protocol flow proposed in [6], we have enhanced its security by adding the generation, exchange and validation of OTPs. For example, the OTP extension protects mainly against a stolen smartphone. Indeed, even if the user's smartphone is stolen, the intruder cannot login as the victim without generating the expected OTP.

3.3 Towards a Formal Specification of Multi-factor Authentication

We now discuss the peculiarities of a multi-factor authentication solution compared to a basic username-password authentication; in doing so, we introduce some concepts that will be the key for the formal analysis.

In a basic username-password authentication, the expected security goal is:

($\mathbf{G1}_A$) *SP* authenticates *User*

Here, *User* is required to provide an authentication factor: either credentials (something only she knows) or a session token (e.g., a cookie stored in her browser) in order to properly complete the authentication process. If this is the case, it is possible to specify a minimum set of security assumptions (e.g., on the behavior of *User* or on the communication channels) that are necessary to guarantee $\mathrm{G1}_A$. For example, if the channel used for the login is not https, then an intruder can eavesdrop the *User*'s password and impersonate her in the future. We call these assumptions *strong assumptions* (to distinguish them from the *weak assumptions* that we define later).

A multi-factor authentication solution augments the security of the basic username-password authentication by exploiting two or more authentication factors. By the definition given in Sect. 2.1, we infer that *mID(OTP)* is a two-factor authentication solution using knowledge and ownership elements (factors). We do not consider inherence factors. In addition, instead of considering the independent factors, we introduce the concept of instance-factors.

We call *instance-factor* (*IFactor*) every specific instance of either an ownership factor ($IFactor_o$) or a knowledge factor ($IFactor_k$). The multi-factor authentication solution *mID(OTP)* that we propose contains three instance-factors:

– the $IFactor_o$ *token_IdP* that is stored in *IDOTP* and in IdP_S as a result of the activation phase (used as a session token in place of the user credentials to provide a SSO experience);
– another $IFactor_k$ that can vary according to the specific OTP generator used, e.g., a PIN known by the user (used to protect the OTP generator);
– an $IFactor_o$ that is stored in *IDOTP* (and possibly shared with IdP_S), according to the OTP-generator approach used (e.g., a seed value or a private key).

Note that the $IFactor_o$ *token_IdP* is present in all instances of our solution, whereas the other two factors may differ depending on the specific solution (and this is the reason why we cannot name them explicitly a priori).

Compared to classic notion of authentication factors, instance-factors can have a dependency. For example, the two $IFactor_o$ are stored in *IDOTP*. Thus, by breaching the *IDOTP* app both of them are compromised. However, it is important to note that different mitigations can be implemented for the different instance-factors. For example, in our solution, if a *User* realizes that the *IDOTP* has been compromised (e.g., if her smartphone has been stolen), she can invalidate *token_IdP*, thus blocking possible attacks.

We are not aware of any formal definition of the multi-factor authentication property apart from [14]. In [14] they analyzed a two-factor and two-channel authentication solution that combines a classic single-factor solution with the exchange of a second factor using the GSM/3G/4G communication infrastructure of the user's mobile phone. By generalizing the definition in [14] by considering a solution involving n instance-factors, we can define the following security goal:

($\mathbf{G1}_{MFA}$) Goal $G1_A$ (i.e., *SP* authenticates *User*) holds even if an intruder knows up to $n-1$ instance-factors.

Thus, the addition of instance-factors ensures some "redundancy", meaning that even if one of them is compromised there are no attacks.

We call *weak assumption* (wa) an assumption that, whenever it is not valid or not implemented properly, causes the disclosure of a non-empty set of instance-factors of the same type, i.e., either $IFactor_o$ or $IFactor_k$. We refer to this set as the *set of instance-factors associated with wa* and denote it by writing $IF(wa)$.[1] For example, if a weak assumption $wa1$ states that the intruder cannot read the values typed by *User*, and in the authentication process *User* has to enter her *password* and *PIN*, then $IF(wa1) = \{password, PIN\}$. This definition can be easily extended to a set of weak assumptions WA' as follows: $IF(WA') = \bigcup_{wa_i \in WA'} IF(wa_i)$. We write WA to denote the *set of all the weak assumptions*.

Defining Security Goals. The notions that we just introduced allow us to rephrase the definition of the security goal $G1_{MFA}$ of a multi-factor authentication solution in the following way:

($\mathbf{G1}_{MFA}$) Goal $G1_A$ holds under the strong assumptions and under chosen subsets of weak assumptions (WA') such that the set of instance factors associated to $WA \setminus WA'$ does not include all the instance-factors. That is, $|IF(WA \setminus WA')| < n$.

A main characteristic of *mID(OTP)* is the use of OTPs. In $G1_{MFA}$, we considered (among others) the instance-factors linked to the OTP generation. In addition, as reported in Sect. 2, an OTP "should be non-reusable and non-replicable." Indeed, if the OTP is not fresh, then the knowledge of an OTP leads to the same attacks possible when knowing the instance-factors linked to

[1] To compromise all instance-factors, at least two weak assumptions must be not valid.

its generation. Thus, it is crucial that the following security goal about the OTP is satisfied:

(G2) The OTP must prove its origin (meaning that IdP_S authenticates *IDOTP*, as *IDOTP* is the only app that possesses a secret value shared with IdP_S or a private key), and it is non-reusable (i.e., IdP_S accepts only one OTP for a specific operation so as to avoid replay attacks).

3.4 Assumptions

Our solution is based on different security assumptions, which we have classified as *strong* or *weak* assumptions.

Strong Assumptions. We have identified the following assumptions and checked them to be strong assumptions (see Sect. 4.5): *Trust Assumption* that clarifies the trust relationships between the different entities, *Communication Assumptions* that specify the concrete implementation of the communication channels required in *mID(OTP)*, and *Activation Assumption* that identifies the assumptions related to the activation phase of *mID(OTP)*.

Trust Assumption. mID(OTP) is based on the following trust relationship:

(TA) IdP_S is trusted by SP_C.

Communication Assumptions. Communications between the parties are subject to the following assumptions:

(ComA1) The communication between SP_C and *IDOTP* is carried over an inter-app communication implemented using `StartActivity ForResult()`. This Android method—which allows an app to open another app and get a result back—guarantees that the SP_C app that sends a request to *IDOTP* at Step A2 in Fig. 1 is the same app that receives the result back from *IDOTP* at Step A10.

(ComA2) To read the *key_hash* value (Step A3 of Fig. 1), *IDOTP* uses the Android method `getPackageInfo(client packageName, PackageManager. GET SIGNATURES)`, which extracts the information about the certificate fingerprint included in the package of SP_C.

(ComA3) The communication between *IDOTP* and IdP_S occurs over a unilateral SSL or TLS channel (henceforth SSL/TLS), established through the exchange of a valid certificate (from IdP_S to *IDOTP*).

Note that even if these assumptions refer to a concrete implementation of the communication channels, in Sect. 4.3 we will provide the formal counterpart abstracting away the implementation details. By doing so, any implementation satisfying the abstract assumptions can be used in place of the implementation mentioned above (e.g., considering a similar solution in the case of iOS), and the results of our security analysis still hold. For example, the main reason to have ComA1 is to avoid the eavesdropping of the identity assertion (*token_SP*) by a malicious app, as in this way an intruder can use it to impersonate the user on another smartphone. An alternative implementation of ComA1 could be obtained by requiring SP_C to insert a fresh value in the token request. In this way, SP_C will accept only the *token_SP* that includes the expected fresh value. Regardless of the design choice, it is crucial that SP_C (and SP_S if it is involved) only accepts tokens that are released for itself for a particular operation.

Activation Assumption. Phishing attacks (e.g., a malicious app that creates a fake login form and steals the user's credentials) are one of the most common types of attack and usually are beyond the scope of an authentication protocol. In our analysis, together with a secure communication, we assume that no phishing is possible during the activation phase:

(ActivA) The activation phase is correctly performed by *User*. That is, *User* downloads the correct *IDOTP* (it is not a fake app) and correctly follows the process, and the communication channels used are secure.

Weak Assumptions. We have identified two categories for weak assumptions: *Background Assumptions* that specify the assumptions on the environment (user's smartphone), and *User Behavior Assumptions* that specify which user behaviors are allowed in our model.

Background Assumptions. The environment is subject to these assumptions:

(BA1) Integrity and confidentiality of data stored in the device.
(BA2) There is no surveillance software (e.g., keylogger) installed on the user's device capable of reading the values that *User* types.

User Behavior Assumptions. To enforce a correct execution of the flow and to investigate the security consequences of a stolen smartphone, in our analysis we take into account the following behavioral rules:

(UBA1) *User* enters her $IFactor_k$ only in the correct *IDOTP* app being careful not to be seen by other people.
(UBA2) *User* is the only person using the *IDOTP* app that stores the $IFactor_p$ associated to her identity.

4 Formal Specification and Analysis of the mID(OTP) Solution: The mHealth Use-Case

In this section, we describe how the semi-formal description of the *mID(OTP)* solution can be translated into a formal model (in this case, specified in ASLan++). *mID(OTP)* provides a general solution for several application contexts. Instead of presenting at first the general model and then the formalization of a use-case, for brevity and concreteness, here we describe directly the formalization of a real use-case scenario that involves mHealth (mobile health) apps. All the concepts presented apply in general to every solution based on *mID(OTP)* (apart from a trivial renaming of the entities). Only the steps and instance-factors related to the particular OTP generator used are specific for this use-case.

In Sect. 4.1, we describe the entities and the steps of the OTP-generator approach for this use-case. In Sects. 4.2 and 4.3, we detail the mapping between the assumptions and their formal specification. In Sect. 4.4, we give the formalization of the security goals. In Sect. 4.5, we present the results of our security analysis.

4.1 Description of the TreC Scenario

TreC is an acronym for "Cartella Clinica del Cittadino", i.e., "Citizens' Clinical Record". TreC is a platform developed in the Trentino region (Italy) for managing personal health records (PHRs).[2] In addition to the web platform, which is routinely used by around 80,000 users, TreC is currently designing and implementing a number of native Android applications to support self-management and remote monitoring of chronic conditions. These applications are used in a "living lab" by voluntary chronic patients according to their hospital physicians. Examples are:

– "TreC-Lab: Diario Diabete", a mobile diary that allows patients to record health data, such as the blood glucose level and physical activity, and
– "TreC: Referti", which permits patients to consult their personal health data and medical prescriptions from the smartphone.

In the traditional web scenario, patients access services using their local healthcare system credentials (leveraging a SAML-based SSO [15] solution), but a solution for native SSO was missing. The solution we have proposed will allow patients to access different TreC e-health native mobile apps (and possibly other third-party e-health apps) through a single authentication act. An implementation of the proposed model is currently being tested by TreC users.

In the following, we instantiate the entities described in Sect. 3 with the entities involved in TreC: *Patient* plays the role of *User* who wants to access her PHR on her smartphone. *ADC* ("Autenticazione del Cittadino") is the IdP of the local health care system and plays the role of IdP_S. *OTP-PAT* plays the role of *IDOTP* and manages the generation of OTPs and the SSO experience for the apps installed on the phone that are part of the federation. $TreC_C$ (TreC client) plays the role of SP_C and is one of the apps that are part of the *ADC* federation and it is used by *Patient* to read her PHR. $TreC_S$ (TreC server) plays the role of SP_S and manages user health data.

Figure 2 shows the A-steps of the exploitation phase of *mID(OTP)* for this use-case. Compared to Fig. 1, we have detailed the OTP generation box (steps A7 a–c), and graphically shown the channel properties, which we will explain

[2] More information is available at https://trec.trentinosalute.net/.

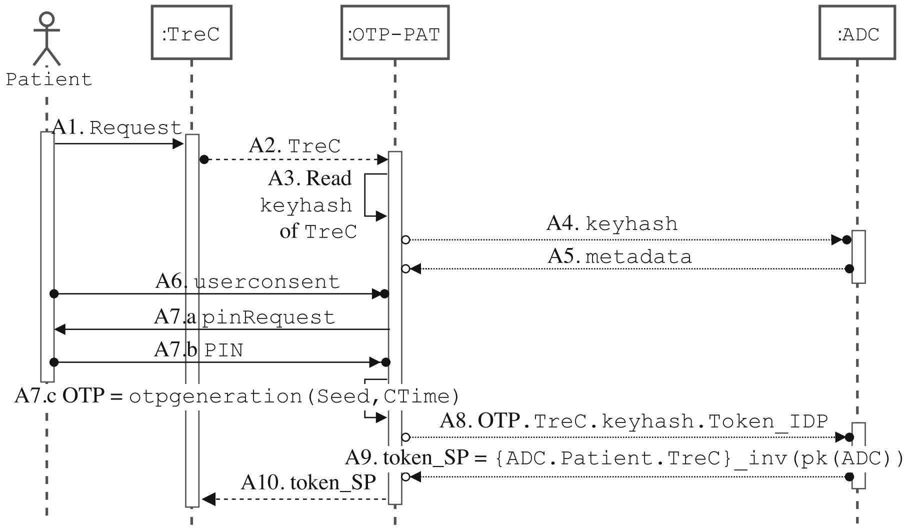


Fig. 2. MSC of the exploitation phase of the TreC scenario.

in Sect. 4.3. Given that *TreC*$_S$ is not involved in the A-steps, for the sake of brevity, in the rest of the section we refer to *TreC*$_C$ simply with *TreC*. Steps A7 (a–c) model the behavior of a Time-OTP (TOTP) algorithm [16], which is a time synchronization algorithm that generates OTPs as a function of the time of the execution and a seed (i.e., a shared secret). In general, the TOTP algorithm requires that "the prover and verifier must either share the same secret or the knowledge of a secret transformation to generate a shared secret" [16], without specifying when and how to exchange this secret. In the analyzed use-case, *OTP-PAT* obtains the seed value as part of the activation phase, and then stores it encrypted with the PIN code ($\{|seed|\}\_PIN$) selected by *Patient*. Thus, the OTP generation box depicted in Fig. 1 is replaced here with a PIN request (Steps A7.a), the entering of the PIN (Steps A7.b) and the generation of the OTP as a function of the seed—extracted using the PIN as decryption key—and of time (Steps A7.c).

The TreC scenario corresponds to a multi-factor authentication with 3 instance-factors: *token_IdP* and $\{|seed|\}\_PIN$ are $IFactor_o$, and *PIN* is an $IFactor_k$.

In the rest of this section, we present the formalism that we have used to specify this use-case, detailing the initial state and the behavior of the entities, the channels and the security goals. We also describe how we have formalized the assumptions presented in Sect. 3.4. In Table 1, we show each assumption and the corresponding formal specification. In addition, we model what in Sect. 3.3 is indicated as *an assumption not valid or not implemented properly* by removing it from the formal model, as shown in the last column of Table 1.

4.2 Formal Specification of the Initial State and of the Behavior of Entities

Initial States. The initial state of a protocol defines the initial knowledge of the intruder, who is indicated with the letter `i`, and of all the honest entities that participate in the protocol session, where a protocol session is a particular run of the protocol, played by specific entities, using specific instances of the communication channels and optionally, additional parameters that must be passed as initial knowledge to the different entities. To model the TA assumption, as shown in Table 1, in our analysis we have not considered sessions with `i` playing the role of *OTP-PAT* and *ADC*.

Regarding the registration phase, we have modeled the data provided by the *TreC* developer as initial knowledge of *ADC*. In general, after the registration phase, IdP_S creates two databases: `trustedSPs`, containing the relation between the SP_C identities and their *key_hash* values, and `metadataDB`, containing the relation between the *key_hash* and the information (e.g., name and logo) provided by the SP developers. As shown in Table 1 by the ActivA assumption, we have modeled the data obtained as result of the activation phase (*token_IdP* and data required for generating OTPs) as initial knowledge of *User*, *IDOTP*

Table 1. Mapping between assumptions (Asm(s) for short) and formal specification.

Asm	Formal specification	
	Specification of Asm	**Removal of Asm**
TA	We do not consider sessions with `i` playing the role of *ADC*	add sessions with `i` playing the role of *ADC*
ComA1	`link(T2O,O2T);`	delete `link(T2O,O2T);`
ComA2	`authentic_on(T2O,TreC);` and DB Keyhash	delete `authentic_on(T2O,TreC);`
ComA3	`confidential_to(O2A,ADC);` `weakly_authentic(O2A);` `weakly_confidential(A2O);` `authentic_on(A2O,ADC);` `link(O2A,A2O);`	delete `confidential_to(O2A,ADC);` `weakly_authentic(O2A);` `weakly_confidential(A2O);` `authentic_on(A2O,ADC);` `link(O2A,A2O);`
ActivA	Data obtained during the activation phase are `nonpublic` values shared as parameters between *Patient*, *OTP-PAT* and *ADC*	add `iknows(pinUser);` `iknows(token_IDP);` `iknows({|seed|}_pinUser);` in general add all the `iknows(`$IFactor$`)`; obtained during the activation phase
BA1	"Built-in": `i` cannot read the internal state of the other entities	add `iknows(token_IDP);` and `iknows({|seed|}_pinUser);` in general add all the `iknows(`$IFactor_p$`);`
BA2	"Built-in": `i` cannot read the internal state of the other entities	add `iknows(pinUser);` in general add all the `iknows(`$IFactor_k$`);`
UBA1	`confidential_to(P2O,OTP-PAT);`	delete `confidential_to(P2O,OTP-PAT);`
UBA2	`authentic_on(P2O,Patient);`	delete `authentic_on(P2O,Patient);`

and *IdP*$_S$. In particular, for the use-case, as result of the activation phase: a *Patient* knows her PIN value (`pinUser`), *OTP-PAT* knows `token_IDP` and `{|seed|}_pinUser`, and *ADC* creates a DB (`usersDB`) with `Patient`, `token_IDP` and `seed` as entry.

To specify that the intruder knows a message m, we use the ASLan++ predicate `iknows(`m`)`. As shown in Table 1 for ActivA, BA1 and BA2, the removal of an assumption (which we will do to consider different scenarios of the analysis) boils down to adding some `iknows` facts to the initial knowledge of the intruder.

Behavior of Entities. The behavior of the honest entities is specified by the evolution of the system, which consists of a sequence of operations performed by each role. For simplicity, Fig. 3 shows the evolution of the protocol using a process view, which describes the messages exchanged in Fig. 2 for each entity as a set of actions (e.g., receive or send a message and DB access). This formal representation can be translated into various role-based formal languages and input to different state-of-the-art security protocol analyzers. In our analysis, we use ASLan++ and SATMC (see [11] for more details on language and tool).

The translation of the process view into ASLan++ is quite straightforward. The complete ASLan++ specification can be found at https://st.fbk.eu/publications/POST-2018. Here, for lack of space, we provide only an example by considering Steps 1 and 2 of Fig. 2, which involve the entities *Patient*, *TreC* and *OTP-PAT*. Focusing on *TreC*, this exchange of messages in ASLan++ corresponds to

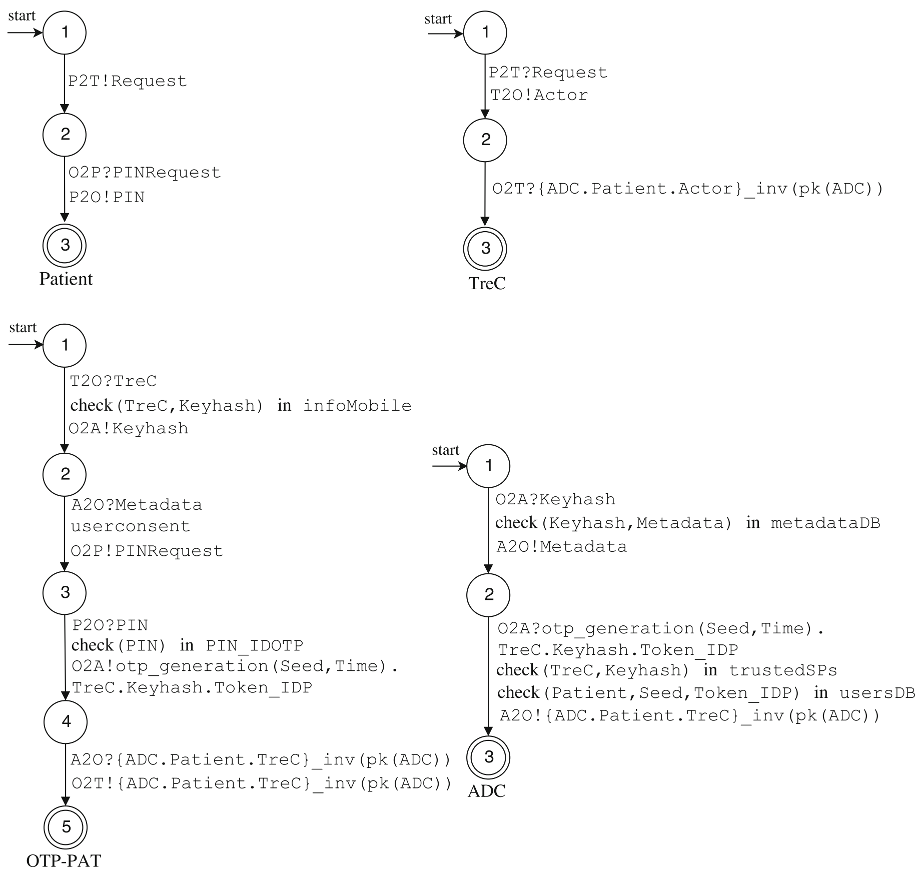
```
Patient -Ch_P2T-> Actor: Request; % Step 1
Actor -Ch_T2O-> OTP-PAT: Actor;   % Step 2
```

where `Actor` is the keyword used in ASLan++ to represent the entity taken into consideration, in our example *TreC*.

In our analysis, we have considered the behavior of a Dolev-Yao intruder [17], who can overhear and modify messages using his initial knowledge and the knowledge obtained from the traffic—this behavior is built-in in the SATMC tool. An operation that is not allowed to `i` is the reading of the internal state of another entity, where an internal state is a list of expressions known by the corresponding entity. Thus, as highlighted in Table 1, BA1 and BA2 are built-in in the tool.

4.3 Formal Specification of Channels

For a detailed definition of the properties of channels between two protocol entities A and B we point the reader to [18,19]. In a nutshell, consider a message M sent on a channel *A2B* from A to B. *A2B* is *authentic* if B can rely on the fact that only A could have sent M. *A2B* is *confidential* if A can rely on the fact that only B can receive M. *A2B* is *weakly authentic* if the channel input is exclusively accessible to a single, but yet unknown, sender, and *A2B* is *weakly confidential* if the channel output is exclusively accessible to a single, yet unknown, receiver. A *link* between two channels *A2B* and *B2A* means that the entity sending messages


start
1
P2T!Request
2
O2P?PINRequest
P2O!PIN
3
Patient
start
1
P2T?Request
T2O!Actor
2
O2T?{ADC.Patient.Actor}_inv(pk(ADC))
3
TreC
start
1
T2O?TreC
check(TreC,Keyhash) in infoMobile
O2A!Keyhash
2
A2O?Metadata
userconsent
O2P!PINRequest
3
P2O?PIN
check(PIN) in PIN_IDOTP
O2A!otp_generation(Seed,Time).
TreC.Keyhash.Token_IDP
4
A2O?{ADC.Patient.TreC}_inv(pk(ADC))
O2T!{ADC.Patient.TreC}_inv(pk(ADC))
5
OTP-PAT
start
1
O2A?Keyhash
check(Keyhash,Metadata) in metadataDB
A2O!Metadata
2
O2A?otp_generation(Seed,Time).
TreC.Keyhash.Token_IDP
check(TreC,Keyhash) in trustedSPs
check(Patient,Seed,Token_IDP) in usersDB
A2O!{ADC.Patient.TreC}_inv(pk(ADC))
3
ADC

Legend:

- P, T, O, and *A* stands for Patient, TreC, OTP-PAT, and ADC respectively, and P2T, P2O, O2P, O2T, O2A, T2O, A2O are their unidirectional channels.
- *Ch*!*M* means that message *M* is sent over channel *Ch*.
- *Ch*?*M* means that a message, says *M*, is received over channel *Ch* and a variable *X* is set to *M*.
- *M1* . *M2* is the concatenation of messages *M1* and *M2*.
- *check(X,Y,...,Z)* in *DB* means that *(X,Y,...,Z)* must be in *DB*, otherwise the protocol stops.
- *M*_inv(pk(ADC)) means that message *M* is digitally signed with the private key of ADC.

Fig. 3. Protocol view.

over the *A2B* is the same entity that receives messages from *B2A*. We have represented these properties graphically in Fig. 2 as follows: $A \bullet\!\!\rightarrow B$, $A \circ\!\!\rightarrow B$, $A \rightarrow\!\!\bullet B$, $A \rightarrow\!\!\circ B$ mean authentic, weak authentic, confidential and weak confidential channel, respectively; moreover, we indicate a link property between two channels with the same trace for the corresponding arrows.

As shown in Table 1, we have modeled as channel properties the tree communication assumptions (ComA1, ComA2 and ComA3) and the two user behavior assumptions (UBA1 and UBA2). The modeling of these assumption is far from a trivial mapping and requires an explanation.

ComA1 is related to the inter-app communication in the mobile. The property expected by the `StartActivityForResult` method can be modeled by a link property between the two channels used in the mobile: the app that has sent a request is the same app that will receive the result.

ComA3 is modeled with five channel properties (see Table 1) that all together model a TLS/SSL unilateral channel.

Regarding ComA2, we have modeled an Android method, which extracts the *key_hash* value included in the package of an app, using an authentic channel (used by *TreC* to send its identity to *OTP-PAT*) and a DB containing the relations between the SP_C identities and their *key_hash*, used by *OTP-PAT* to read the correct *key_hash* value. This is due to the fact that this method—executed by the Android OS—guarantees the authenticity of its output.

We have modeled UBA1 and UBA2 as properties of the channel from *Patient* to *OTP-PAT* (`P2O`). UBA1 is necessary to prevent leakage of the *PIN*—entered in a malicious app or watched by an intruder during the typing—thus, we have modeled `P2O` as a confidential channel. UBA2 guarantees the possession of the *OTP-PAT* app installed in the user's smartphone. Having this assumption, only the valid *Patient* can communicate with that particular installation of *OTP-PAT*, thus we have modeled `P2O` as an authentic channel.

4.4 Formal Specification of Security Goals

As described in Sect. 3.3, we have defined $G1_{MFA}$ in terms of a traditional authentication goal and the strong and weak assumptions. This means that, in the formal model, we consider the traditional authentication goal $G1_A$ and we check whether it holds under the strong assumptions and different (sub)sets of weak-assumptions. The property must hold if the intruder is not able to compromise all the instance-factors. $G1_A$ requires that a message is transmitted in an authenticated and fresh manner, thus allowing *TreC* to authenticate *Patient* and offering replay-protection at the same time. For the definition of authentication we refer to [20]: whenever the entity B completes a run of the protocol apparently with the entity A, then A has previously been running the protocol apparently with B, and the two entities agree on a message M. In ASLan++, this corresponds to specifying the goal

($G1_A$) `SP_authn_U_on_Request: (_)` *Patient* `*->>` *TreC*;

where `*->>` indicates authenticity, directedness (i.e., the only (honest) receiver of a message is the intended one [11]) and freshness. In addition, following the definition in [20], associated goal labels are used to specify which values of M the goal is referring to, namely, the `Request` value in State 1 of the *Patient* process (in Fig. 3) and the corresponding value in the last state of the *TreC* process (State 3 in Fig. 3).

Similarly, the OTP properties are checked by means of the goal

(G2) `IDP_authn_UA_on_OTP: (_)` $\mathit{OTP}\text{-}\mathit{PAT}$ `*->>` ADC;

with the associated goal labels specifying for M the values `otp_generation (Seed,Time)` in States 3 of both the $\mathit{OTP}\text{-}\mathit{PAT}$ and the ADC processes in Fig. 3, where we have modeled `Seed` as a constant value shared between $\mathit{OTP}\text{-}\mathit{PAT}$ and ADC, and `Time` as a session parameter (cf. [16]) shared between $\mathit{OTP}\text{-}\mathit{PAT}$ and ADC. Thus, ADC will accept only one OTP value for each session, enforcing the property (informally described in Sect. 3.3) that OTP is non-reusable.

4.5 Results of the Security Analysis

We are now ready to discuss the results of the security assessment that we have performed on the mHealth use-case. Our focus is determining whether the concurrent execution of a finite number of protocol sessions enjoys the expected security goals in spite of the intruder. To this aim, we have mechanically analyzed the formal model of our use-case using SATMC, a state-of-the-art model checker for security protocols. SATMC carries out an iterative deepening strategy on k. Initially k is set to 0, and then it is incremented till an attack is found (if any) or k_{max} is reached. If this is the case, no attack traces of length up to k_{max} exist, where the length of the trace is computed by taking into account the parallel execution of non-conflicting actions (actions executed in parallel are considered as a single step). The trace includes the actions performed by attacker and honest participants, where most of the actions of the attacker are executed in parallel (and counted as a single step) with the ones of honest participants. We set k_{max} to 1.5 times the length of the longest trace of the protocol when only honest entities participate. As a rule of thumb, with this choice we are reasonably confident that no attack is possible with greater values of k_{max}. In our analysis, the length of the longest trace of the protocol when only honest entities participate is 19, and thus we have set $k_{max} = 30$. We have considered several scenarios including (at most) three parallel sessions in which the intruder either does not play any role or plays the role of SP_C (the TreC app in the use-case). In each session, we used different instances of the channels. The complete set of specifications can be found at the companion website.

In Sect. 3.4, in relation to the security goal $\mathrm{G1}_{MFA}$ (and consequently to $\mathrm{G1}_A$), we have described a list of strong and weak assumptions that we have

Table 2. Analyses performed for $\mathrm{G1}_A$.

Analysis	**Strong Asm(s)**	**Weak Asm(s)**	**Atk**
1	all -1	all	Yes
2	all	all -1	No
3	all	all $-m$ $(1 < m \leq 4)$	*

added to the model to constrain the intruder's abilities. Table 2 summarizes the security analyses that we have performed to check this goal.

Regarding the strong assumptions (TA, ComA1, ComA2, ComA3 and ActivA), we have performed the following analyses:

Analysis 1: We have checked that by removing only one of the five strong assumptions from the model we have a violation of $G1_A$ (i.e., there is an attack). For this analysis, we have thus performed 5 executions of SATMC removing one strong assumption at a time. To provide an example of an attack, Fig. 4 shows the attack trace deriving from removing ComA2. In this attack, `i` can impersonate `trec` simply because the channel used to exchange its identity is not authentic; thus, `i` can pretend to be another app. Note that, for the sake of clarity, this figure (and, similarly, the other figures shown in this section) represents only the significant steps of the attack traces found by the SATMC tool.[3]

Regarding the weak assumptions (BA1, BA2, UBA1, and UBA2), we have performed the following analyses that are detailed in Table 3:

Analysis 2: We have checked that by removing only one of the four weak assumptions from the model, SATMC does not find any attack on the solution (i.e., the intruder is not able to impersonate the user). Indeed, as shown in Table 3, by removing only one weak assumption, the intruder obtains only 1 or 2 instance-factors.

Analysis 3: We have checked that by removing specific subsets of weak assumptions it is possible to compromise all the instance-factors, causing a violation of $G1_A$. In Table 2, the star (*) denotes that the result

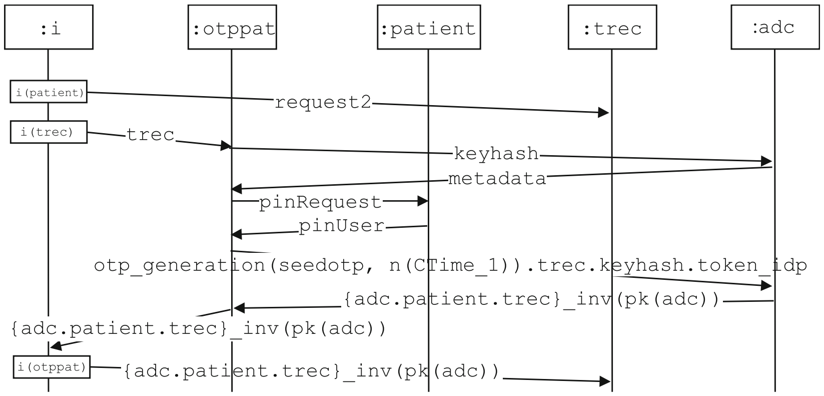

:i
:otppat
:patient
:trec
:adc
i(patient)
request2
i(trec)
trec
keyhash
metadata
pinRequest
pinUser
otp_generation(seedotp, n(CTime_1)).trec.keyhash.token_idp
{adc.patient.trec}_inv(pk(adc))
{adc.patient.trec}_inv(pk(adc))
i(otppat)
{adc.patient.trec}_inv(pk(adc))

Fig. 4. Attack trace without the strong assumption ComA2.

[3] The original charts can be examined on the companion website https://st.fbk.eu/publications/POST-2018.

Table 3. Results for G1$_A$ (Analyses 2 and 3).

Removed weak Asm(s)	**Compromised factors**			**Atk**
	PIN	**{seed}_PIN**	**token_IdP**	
BA1	×	✓	✓	No
BA2	✓	×	×	No
UBA1	✓	×	×	No
UBA2	×	✓	✓	No
(UBA1 ∨ BA2) ∧ BA1	✓	✓	✓	Yes
(UBA1 ∨ BA2) ∧ UBA2	✓	✓	✓	Yes

can be "yes" or "no" depending on the chosen subset of weak assumptions. The subsets shown in Table 3 violate G1$_A$ and result in different attack traces. Figure 5 shows the attack trace deriving from removing UBA1 and UBA2 (e.g., a proximity intruder that watches the *PIN* entered by *Patient* and then steals the smartphone). In the attack, `i` initiates a session of the protocol with `trec` pretending to be `patient` (indicated as `i(patient)`). By entering the PIN code (`pinUser`) when requested by `otppat`, `i` is able to impersonate the patient and obtaining the requested resource (`resources1`). Figure 6 shows the attack trace deriving from removing both BA1 and BA2 (e.g., a hacker that steals the *PIN* typed by *Patient* using a keylogger and reads *token_IdP* and {|*seed*|}_*PIN* exploiting a malware installed on the smartphone). In this case, `i` is able to generate an OTP and sends a token request to `adc`.

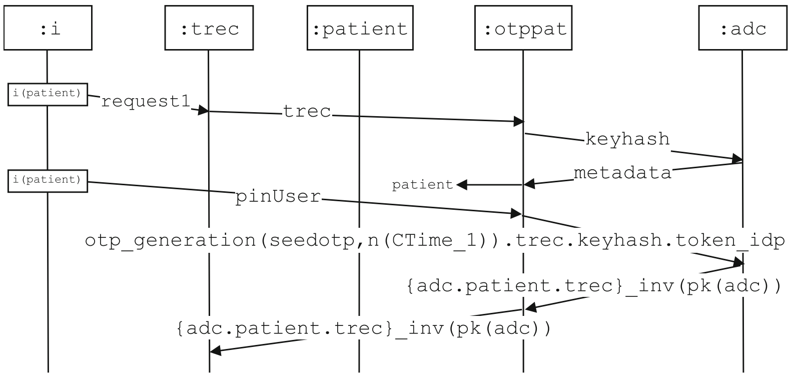

:i
:trec
:patient
:otppat
:adc
i(patient)
request1
trec
keyhash
metadata
patient
i(patient)
pinUser
otp_generation(seedotp,n(CTime_1)).trec.keyhash.token_idp
{adc.patient.trec}_inv(pk(adc))
{adc.patient.trec}_inv(pk(adc))

Fig. 5. Attack trace obtained removing UBA1 and UBA2.

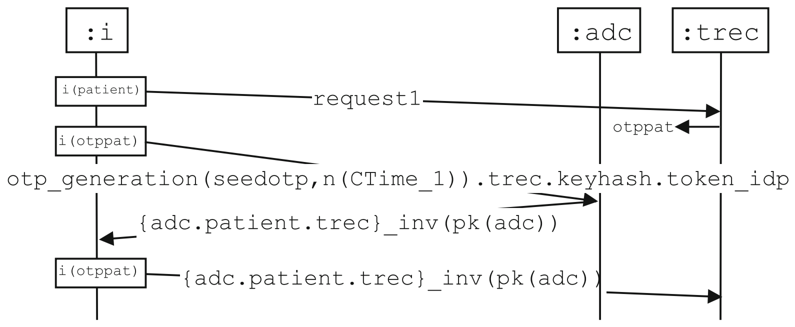

:i
:adc
:trec
i(patient)
request1
otppat
i(otppat)
otp_generation(seedotp,n(CTime_1)).trec.keyhash.token_idp
{adc.patient.trec}_inv(pk(adc))
i(otppat)
{adc.patient.trec}_inv(pk(adc))

Fig. 6. Attack trace obtained removing BA1 and BA2.

As expected, when checking the solution w.r.t. the security goal G2—which embodies the OTP properties—under all the (weak and strong) assumptions, SATMC does not find any attack.

5 Related Work

OAuth 2.0 [21] and OpenID Connect [22] have been designed for light-RESTful API services, and are considered the de-facto standards for managing authentication and authorization. These protocols are well-accepted in the web scenario, but they provide only partial support for mobile apps (frequent use of the expression "out of scope"). This could lead to the implementation of insecure solutions. An in-depth analysis of OAuth in the mobile environment—underlining possible security problems and vulnerabilities—is available in [23,24].

Given the lack of specifications, the OAuth Working Group has released in 2017 a best practice with the title "OAuth 2.0 for Native Apps" [25]. The specification of [25] has two main differences with respect to our solution: the choice of *UA* (browser vs native app) and the activation phase. The authors of [25] do not described any security issues in using native apps as *UA*; they discourage this because of the overhead on users to install a dedicated app. Nevertheless, in some scenarios, we consider this to be an advantage rather than a drawback because it allows for easily integrating new security mechanisms (e.g., access control and a wider range of MFA solutions). Concerning the activation phase of our solution, it allows for better mitigation of phishing as users directly interact with our app. Instead, [25] requires a redirection from a (possible malicious) SP_C to a browser, thus users can be cheated by a fake browser invoked by SP_C. We want to underline that, as described in [7], our solution is not designed from scratch but on top of Facebook; and the formalization that we have presented in this work can be easily extended to also analyze the OAuth solution of [25].

Much research has been carried out to discover vulnerabilities in different implementations of OAuth 2.0 and OpenID Connect in web and mobile scenarios. For instance, Sun et al. [26] analyzed hundreds of OAuth apps focusing on classical web attacks such as Cross-Site Scripting (XSS) and Cross-Site Request Forgery (CSRF). Other studies, such as [27,28], analyzed the implementations of multi-party web apps via browser-related messages. In the context of mobile apps, a similar work is described in [29], where Yang et al. discovered an incorrect use of OAuth that permits an intruder to login as a victim without the victim's awareness. To evaluate the impact of this attack, they have shown that more than 40% of 600 top-ranked apps were vulnerable.

Although these techniques are useful for the analysis of a specific implementation (as they are able to discover serious security flaws), it is important to perform a comprehensive security analysis of the standard itself. In the context of web apps, Fett et al. [30] performed a formal analysis of the OAuth protocol using an expressive web model (defined in [31]) that describes the interaction between browsers and servers in a real-world set-up. This formal analysis revealed two unknown attacks on OAuth that violate the authorization and authentication properties. A similar analysis is performed for OpenID Connect in [32]. Two other examples of formalizations of OAuth are [33], where the different OAuth flows are modeled in the Applied Pi calculus and verified using ProVerif extended with WebSpi (a library that models web users, apps and intruders), and [34], where OAuth is modeled in Alloy.

In our analysis (cf. Sect. 4) we used ASLan++ and SATMC. In the past, SATMC has revealed severe security flaws in the SAML 2.0 protocol [15] and in the variant implemented by Google [18]; by exploiting these flaws a dishonest service provider could impersonate a user at another service provider. Moreover, Yan et al. [35] used ASLan++ and SATMC to analyze four security properties of OAuth: confidentiality, authentication, authorization, and consistency.

The aforementioned formal analyses, however, focus on the web app scenario, whereas in this paper we deal with native apps. In [36], Ye et al. used Proverif to analyze the security of a SSO implementation for Android. They applied their approach to the implementation of the Facebook Login and identified a vulnerability that exploits super user (SU) permissions. In contrast, our analysis assumes that the user smartphone cannot be rooted. Indeed, if a malicious app is able to obtain a SU permission, then it can set for itself the permission to access all the data stored in the smartphone, compromising all the user data and the tokens of the other apps installed on the rooted smartphone.

YubiKey NEO [37] is one of the most attractive mobile identity management products on the market. It is a token device that supports OTPs and the FIDO Alliance Universal 2nd Factor (U2F) protocol, and, by integrating an NFC (Near Field Communication) technology, it can be used to provide a second-factor also in the mobile context. Compared to this product, our solution provides a multi-factor authentication solution for native mobile apps without requiring an additional device.

6 Conclusions

We have presented the design of *mID(OTP)*, a multi-factor authentication solution for native mobile apps that includes an OTP exchange and provides a

SSO experience. In addition to the protocol flow, we have detailed the security assumptions and defined two security goals: $G1_{MFA}$ related to a multi-factor authentication solution and G2 that identifies the properties of a OTP. To perform a security analysis of *mID(OTP)*, we have detailed the OTP-generation approach in the context of a real use-case scenario (TreC). We have formally modeled the flow, assumptions and goals of TreC using a formal language (ASLan++) and checked the identified security goals using a model-checker (SATMC).

The solution we have presented, as well as the formal specification and analysis that we have given, can be generalized quite straightforwardly to other use-cases, which we are currently doing. As future work, we also plan to extend the analysis to other authentication factors, such as biometric traits. In addition, we started exploring an alternative formalization of multi-factor authentication protocols that decomposes the protocol and models the authentication property as a composition of two goals: one related to basic authentication (involving *User*, *UA*, SP_C and IdP_S) and one related only to the generation and validation of the OTP (without involving SP_C). In this way, a proper separation is kept between the multi-factor authentication performed with IdP_S and the basic authentication plus SSO experience offered to SP_C. As a preliminary analysis, we can affirm that the two different definitions of goals lead to similar attack traces.

Acknowledgements. This work has partially been supported by the Health & Wellbeing high-impact initiative (HII-H&WB) of FBK, the TreC (Cartella del Cittadino) project of the Azienda Provinciale per i Servizi Sanitari, and the activity "API Assistant" of the action line Digital Infrastructure of the EIT Digital.

References

1. BBA – British Bankers' Association: An app-etite for banking (2017). https://www.bba.org.uk/wp-content/uploads/2017/06/WWBN-IV.pdf
2. European Commission: mHealth (2016). https://ec.europa.eu/digital-single-market/en/mhealth
3. He, D., Naveed, M., Gunter, C.A., Nahrstedt, K.: Security Concerns in Android mHealth App (2014). https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4419898/
4. European Commission: Regulation EU 2016/679 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). http://www.eugdpr.org
5. Google: Google Authenticator. https://support.google.com/accounts/answer/1066447?hl=en
6. Sciarretta, G., Carbone, R., Ranise, S., Armando, A.: Anatomy of the Facebook solution for mobile single sign-on: security assessment and improvements. J. Comput. Secur. (COSE 2017) **F71**, 71–86 (2017). https://doi.org/10.1016/j.cose.2017.04.011
7. Sciarretta, G., Armando, A., Carbone, R., Ranise, S.: Security of mobile single sign-on: a rational reconstruction of Facebook login solution. In: Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016). SECRYPT, vol. 4, pp. 147–158 (2016)

8. ECB - European Central Bank: Final guidelines on the security of Internet payments (2014). https://www.eba.europa.eu/documents/10180/934179/EBA-GL-2014-12+%28Guidelines+on+the+security+of+internet+payments%29.pdf/f27bf266-580a-4ad0-aaec-59ce52286af0
9. Lamport, L.: Password authentication with insecure communication communications. Commun. ACM **24**(11), 770–772 (1981)
10. Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems: Concepts and Design, 4th edn. Addison-Wesley, Boston (2005)
11. AVANTSSAR Project: Deliverable D2.3 (update) ASLan++ specification and tutorial. http://www.avantssar.eu/pdf/deliverables/avantssar-d2-3_update.pdf (2008)
12. Armando, A., et al.: The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 267–282. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_19
13. Armando, A., Carbone, R., Compagna, L.: SATMC: a SAT-based model checker for security protocols, business processes, and security APIs. STTT **18**(2), 187–204 (2016)
14. Armando, A., Carbone, R., Zanetti, L.: Formal modeling and automatic security analysis of two-factor and two-channel authentication protocols. In: Lopez, J., Huang, X., Sandhu, R. (eds.) NSS 2013. LNCS, vol. 7873, pp. 728–734. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38631-2_63
15. OASIS: SAML V2.0 technical overview (2005). https://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf
16. IETF: TOTP: Time-Based One-Time Password Algorithm (2011). https://tools.ietf.org/html/rfc6238
17. Dolev, D., Yao, A.: On the security of public-key protocols. IEEE Trans. Inf. Theor. **29**(2), 198–208 (1983)
18. Armando, A., Carbone, R., Compagna, L., Cuéllar, J., Tobarra, L.: Formal analysis of SAML 2.0 web browser single sign-on: breaking the SAML-based single sign-on for Google Apps. In: Proceedings of the 6th ACM Workshop on Formal Methods in Security Engineering (FMSE), pp. 1–10 (2008)
19. Mödersheim, S., Viganò, L.: Secure pseudonymous channels. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 337–354. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_21
20. Lowe, G.: A hierarchy of authentication specifications. In: Proceedings of the 10th IEEE Workshop on Computer Security Foundations (1997)
21. IETF: The OAuth 2.0 Authorization Framework (2012). http://tools.ietf.org/html/rfc6749
22. OIDF: OpenID Connect Core 1.0 (2014). http://openid.net/specs/openid-connect-core-1_0.html
23. Chen, E., Pei, Y., Chen, S., Tian, Y., Kotcher, R., Tague, P.: OAuth demystified for mobile application developers. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS) (2014)
24. Shehab, M., Mohsen, F.: Towards enhancing the security of OAuth implementations in smart phones. In: IEEE International Conference on Mobile Services (MS), pp. 39–46 (2014)
25. OAuth Working Group: OAuth 2.0 for Native Apps (2016). https://tools.ietf.org/html/rfc8252

26. Sun, S., Beznosov, K.: The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems. In: Proceedings of the ACM Conference on Computer and Communications Security (CCS 2012) (2012)
27. Wang, R., Chen, S., Wang, X.: Signing me onto your accounts through Facebook and Google: a traffic-guided security study of commercially deployed single-sign-on web services. In: Proceedings of the IEEE Symposium on Security and Privacy (S&P), pp. 365–379 (2012)
28. Sudhodanan, A., Armando, A., Carbone, R., Compagna, L.: Attack patterns for black-box security testing of multi-party web applications. In: Proceedings of the 23rd Annual Network and Distributed System Security Symposium (NDSS) (2016)
29. Yang, R., Lau, W.C., Liu, T.: Signing into one billion mobile app accounts effortlessly with OAuth2.0. In: Black Hat Europe (2016)
30. Fett, D., Küsters, R., Schmitz, G.: A comprehensive formal security analysis of OAuth 2.0. In: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1204–1215. ACM (2016)
31. Fett, D., Küsters, R., Schmitz, G.: An expressive model for the web infrastructure: definition and application to the BrowserID SSO system. In: Proceedings of the 35th IEEE Symposium on Security and Privacy (S&P), pp. 673–688. IEEE Computer Society (2014)
32. Fett, D., Küsters, R., Schmitz, G.: The web SSO standard OpenID connect: in-depth formal security analysis and security guidelines. In: Proceedings of the 30th Computer Security Foundations Symposium (CSF). IEEE Computer Society (2017)
33. Bansal, C., Bhargavan, K., Maffeis, S.: Discovering concrete attacks on website authorization by formal analysis. In: Proceedings of 25th IEEE Computer Security Foundations Symposium (CSF 2012), pp. 247–262 (2012)
34. Pai, S., Sharma, Y., Kumar, S., Pai, R.M., Singh, S.: Formal verification of OAuth 2.0 using alloy framework. In: Proceedings of the IEEE International Conference on Communication Systems and Network Technologies (CSNT), pp. 655–659 (2011)
35. Yan, H., Fang, H., Kuka, C., Zhu, H.: Verification for OAuth using ASLan++. In: Proceedings of 16th IEEE International Symposium on High Assurance Systems Engineering HASE, pp. 76–84 (2015)
36. Ye, Q., Bai, G., Wang, K., Dong, J.S.: Formal analysis of a single sign-on protocol implementation for Android. In: Proceedings of the 20th ICECCS, pp. 90–99 (2015)
37. Yubico: YubiKey NEO. https://www.yubico.com/products/yubikey-hardware/yubikey-neo