Journal of Computer Science

Original Research Paper

Two Factor Authentication for e-Government Services using Hardware-Like One Time Password Generators

[1]Giuseppe Della Penna, [2]Pietro Frasca and [2]Benedetto Intrigila

[1]Departement of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
[2]Dipartimento di Ingegneria dell'Impresa, University of Rome "Tor Vergata", Italy

Article history
Received: 17-10-2018
Revised: 12-12-2018
Accepted: 25-01-2019

Corresponding Author:
Giuseppe Della Penna
Departement of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy
Email: giuseppe.dellapenna@univaq.it

Abstract: A safe and accessible authentication technique is a prerequisite for any modern e-government application. Two-factor authentication is currently widely adopted, since it alleviates many vulnerabilities of password-based authentication. The majority of e-government systems currently make use of text messages to deliver the second authentication factor, but these messages do not constitute an adequate (secure and reliable) solution. In this paper we show how to use One-Time Passwords (OTP) generated by a per-user, ad-hoc built application installed on a smartphone to support a two-factor authentication scheme specifically targeted to e-government tasks. In particular, we develop a process for the request, generation and distribution of such an application that achieves the same security of OTP hardware devices but avoids the related distribution and management costs, requiring no dedicated hardware and relying on the pre-existing administrative infrastructure. The process is designed to be accessible by any citizen who is able to perform very basic operations on a smartphone.

Keywords: E-Government Services, Service Accessibility, Two-Factor Authentication, One-Time Password, Mobile Applications

Introduction

The increasing adoption of e-government systems is quickly and radically changing the way citizens interact with public institutions e.g., (Orgeron and Goodman, 2011; Santoso *et al*., 2016) while, on the other hand, the availability of such systems is becoming an index of social and political progress (Boyer-Wright and Kottemann, 2015; Yulistiawan *et al*., 2014). The quality of an e-government system depends on a set of very different factors that range from accessibility to security e.g., (Papadomichelaki *et al*., 2015). Here we focus on a pivotal security aspect: The authentication system.

Authentication and Digital Society

Authentication systems are at the basis of e-government systems e.g., (Bettacchi *et al*., 2017; National Institute of Standards and Technology, 2017), as well as of a wide range of applications belonging to the so called *Digital Society*. Therefore, an authentication technique that is both very accessible, to be used by any citizen and extremely safe, to access public services and personal data, is a prerequisite for any modern e-government application.

As it is well known, *static authentication systems*, i.e., based on static passwords, are vulnerable to many typologies of attacks. Such vulnerability can be substantially alleviated by the so called *multifactor authentication systems*, in particular two-factor authentication (Stanislav, 2015). In such authentication schemes, two techniques are joined: The first factor is still usually a static password, whereas the second factor can be provided in several ways, e.g., by *biometric characteristics* such as fingerprints e.g., (Velásquez *et al*., 2018; Kumar *et al*., 2017), by smart cards (Olabode, 2011) or by *One-Time Passwords (OTP)* (IETF, 1998).

In particular, the majority of e-government systems currently make use of text messages to provide the citizen with an OTP as second authentication factor. This is not surprising, since SMS is a basic service of mobile networks, available on every mobile phone, thus it is a quite *democratic* solution, which can be used by a very wide segment of the population.

However, as e-government services become more and more important in the citizen lives, the security and reliability constraints of such services must strengthen, making the use of text messages completely unacceptable.

Indeed, the public administration has little or no control on the internal *handling* of SMS messages, so there may be unpredictable delays in their delivery, possibly invalidating the citizen authentication session. Moreover, SMS phishing (Choudhary and Jain, 2018), which is a very common criminal activity, may be easily exploited in this scenario (Siadati *et al*., 2016; 2017). Finally, since in the GSM protocol only the airway traffic may be optionally encrypted and with a weak stream cipher, SMS messages may be relatively easy to intercept and read by some attacker (Barkan *et al*., 2008). For these reasons and also considering that SMS messages have a cost for the administration (and possibly for the citizens), it is clear that they will not be the preferred way to implement two-factor authentication in the near future (Meyer, 2016; NIST, 2017).

Our Ad-Hoc Solution

In this paper, we show how to use OTPs generated by a suitable application installed on a smartphone to support a two-factor authentication scheme specifically targeted to e-government tasks. In particular, *we develop a request, generation and distribution process* for such an application *that achieves the same security level of OTP hardware devices, which are currently a standard for highsecurity transactions* (indeed, they are widely adopted for e-banking).

This result is obtained by:

- Embedding in the executable code of the application *a substantial amount of information related to the specific user and his device*
- Performing an *ad-hoc compilation* of the application for each specific user
- *Distributing the application through a controlled, personalized channel* rather than through a public repository

To stress these important aspects, we call the application *Ad-Hoc OTP mobile app* or, shortly, *AH-OTP app*. While the current solutions of this kind differentiate the users only by the initial secret key manually entered in the app, as discussed below, the AH-OTP app will be substantially different for each user (this is the *ad-hoc* aspect).

Our solution does not make use of SMS or similar insecure channels to deliver OTPs, but at the same time avoids the distribution and management costs related to OTP hardware devices. Indeed, it does not use dedicated hardware and relies on administrative offices for the most critical identification phases. Moreover, it is designed to be accessible by a large number of citizens, which already own a smartphone and are able to perform very basic operations on it.

Clearly, we are still facing some digital divide, since a number of citizens, typically the most aged ones, will not be able to use our authentication system (Ebbers *et al*., 2016; Distel and Becker, 2017). However, this is an intrinsic issue in e-government (Baeuo *et al*., 2017; Distel and Becker, 2017), which we will not address here: Our purpose is to minimize such a number, while maintaining the highest security levels.

The paper is organized as follows. Section 2 compares the proposed approach with other two-factor authentication schemes. Section 3 introduces some preliminary notions about one-time passwords. Section 4 contains a detailed description of the proposed process and briefly discusses its accessibility. Section 5 focuses on the process and app security issues related to a number of common attacks and Section 6 formally prove the security of the proposed approach with respect to such attacks using model checking techniques. Finally, concluding remarks and future work are outlined in Section 7.

Related Work

Typically, the reference scenarios for smartphone-assisted authentication and authorization are the secure payments and the service access mechanisms provided, e.g., by Google or Microsoft.

Secure payments have been using two-factor authentication schemes for many years, but they usually employ dedicated hardware tokens to achieve higher security. Obviously such tokens have a cost both for the issuer and the user, must be replaced after a specific number of years and cannot be simply "stored in a safe place", but rather should be carried with the user everywhere authentication may be required. These constraints are acceptable for e-banking purposes, but clearly inapplicable to a diffuse e-government infrastructure. Only in the most recent years banks are also launching smartphone applications that may be used as a replacement for the dedicated token. Usually these applications are extensions of common e-banking applications and, to achieve the required authorization security level, use complex authentication procedures. In other words, such software-based authorization systems free the user from the extra hardware, but are more demanding with respect to both the smartphone hardware requirements and the user abilities.

On the other hand, smartphone-based payment systems such as Apple pay (Apple Inc, 2018) or Google pay (Google Inc, 2018) are being increasingly adopted, especially for micro payments. Both systems rely on traditional credit or debit cards, whose details are stored in an encrypted form in the smartphone itself. Using biometric or password-based security techniques, these systems are able to send via NFC to an enabled POS an authorization token which encodes the card data and, in this way, the card holder identity. It is clear that such authorization scheme is currently very payment-specific and hardware-eager, so it would be very difficult to extend to an e-government scenario.

Actually, most of the recent approaches to multi-factor authentication relying on electronic devices such as smartphones or personal computers also use biometrics e.g., Bailey *et al*. (2014), or a combination of biometrics with standard techniques such as one-time passwords e.g., Dasgupta *et al*. (2016). Again, the hardware requirements of such approaches make them unsuitable for a widely-available and accessible authentication scheme such as the one needed by the current e-government scenarios.

Two-factor authentication schemes employed by services such as Google or Microsoft make use of more general techniques, where the second factor can be generated by the user smartphone in different ways with increasing delay but decreasing requirements. As an example, Android users can simply answer to an operating system notification (if their phone is online) to access Google services. On the other hand, the most common way to access such services is to install (from the public stores) and configure an *authenticator app*, which essentially implement the TOTP (IETF, 2011) algorithm. Finally, these services also allow sending a server-generated OTP via text message as a backup procedure.

In the e-government context, a well-known example of application of two-factor authentication techniques is given by the United States approach. Such an approach relies on a centralized authentication system with a strong user-chosen password and an OTP sent via text Message (SMS) to the citizen phone (U.S. General Services Administration, 2018).

Also the recently introduced Italian Digital Identity Public System (SPID), (APLD, 2018) includes a two-factor authentication with OTP sent via SMS as the middle-level authentication, whereas the base one is a simple password-based mechanism and the strongest one is a two-factor with a hardware token.

Moreover, in Italy, the main third parties offering solutions for secure authentication to government portals as well as remote digital signature, such as InfoCert (2018) or Aruba (Aruba it, 2018), provide the users with OTP generators which are freely downloadable from the mobile stores and are configured using a secret key at the first run. Such an authenticator-based approach is the starting point of the process developed in this paper, too. However, with respect to all the solutions above we further reinforce the user identification process and the app code strength, making harder to stole the user identity or clone the app installed on his phone. This is a fundamental prerequisite for e-government services.

One-Time Passwords

One-Time Passwords (OTP) are passwords that can be used only to perform a single transaction and therefore are not affected by a number of issues associated with traditional passwords, since they are not vulnerable to replication attacks. Moreover, to further strengthen their security, OTPs, if not used, usually expire after a short interval of time. Clearly OTPs cannot be stored, but have to be generated on request and therefore they require some additional technology to be effectively exploited. OTP generation algorithms commonly make use of pseudorandom number generators and hash functions to make the prediction of the next password very difficult to achieve. Actually, there are several classes of OTP generation algorithms, but the most used in the practice are the *timesynchronized* ones. In this case, the OTP is based on the current timestamp, possibly merged with the previous password or, most commonly, with a shared user secret key. In the latter case we have the well-known *TOTP* (IETF, 2011) algorithm, which combines the current timestamp and the shared secret using a cryptographic hash function.

Time-synchronized OTP passwords are usually delivered through a dedicated hardware (*security token*) which contains an accurate clock (that must be synchronized with the clock on the authentication server). The security token generates and displays an OTP each time a button is pressed. This simple implementation has, however, a clear drawback in that such specific hardware must be carried along by the owner. In other cases, the OTP generator resides on the authentication server itself and the passwords are generated and then sent to the user through non-internet-based secure channels, like SMS text messages.

Nowadays, smartphones have all the computing power needed to implement an OTP algorithm and are always with us, so they are perfect candidates as security tokens. Indeed, many apps are available in the mobile stores, usually identified as *authenticators*, which can be configured to generate OTPs for specific services. However, in this case the OTP generation algorithm, being installed on a vulnerable device often connected to the internet and being downloaded from a public market, can be subject to many kinds of attack. As an example, an attacker can freely download the app from the store, analyse it, understand how the secret key is stored in the device's memory and, if the device is compromised, steal the secret key and clone the user's OTP generator. On the other hand, *this schema does not apply to the ad-hoc OTP application presented in this paper* since it is not public and is ad-hoc generated making use of obfuscated user-specific information.

The AH-OTP App and its Release Process

The core of any two-factor authentication scheme, where the first factor is usually a password, is the way of generating the second factor. As already discussed, in an e-government context we should try to achieve the best compromise between security and usability. The proposed solution relies on the *AH-OTP* app which is distributed through a very specific release process, as illustrated in Fig. 1.

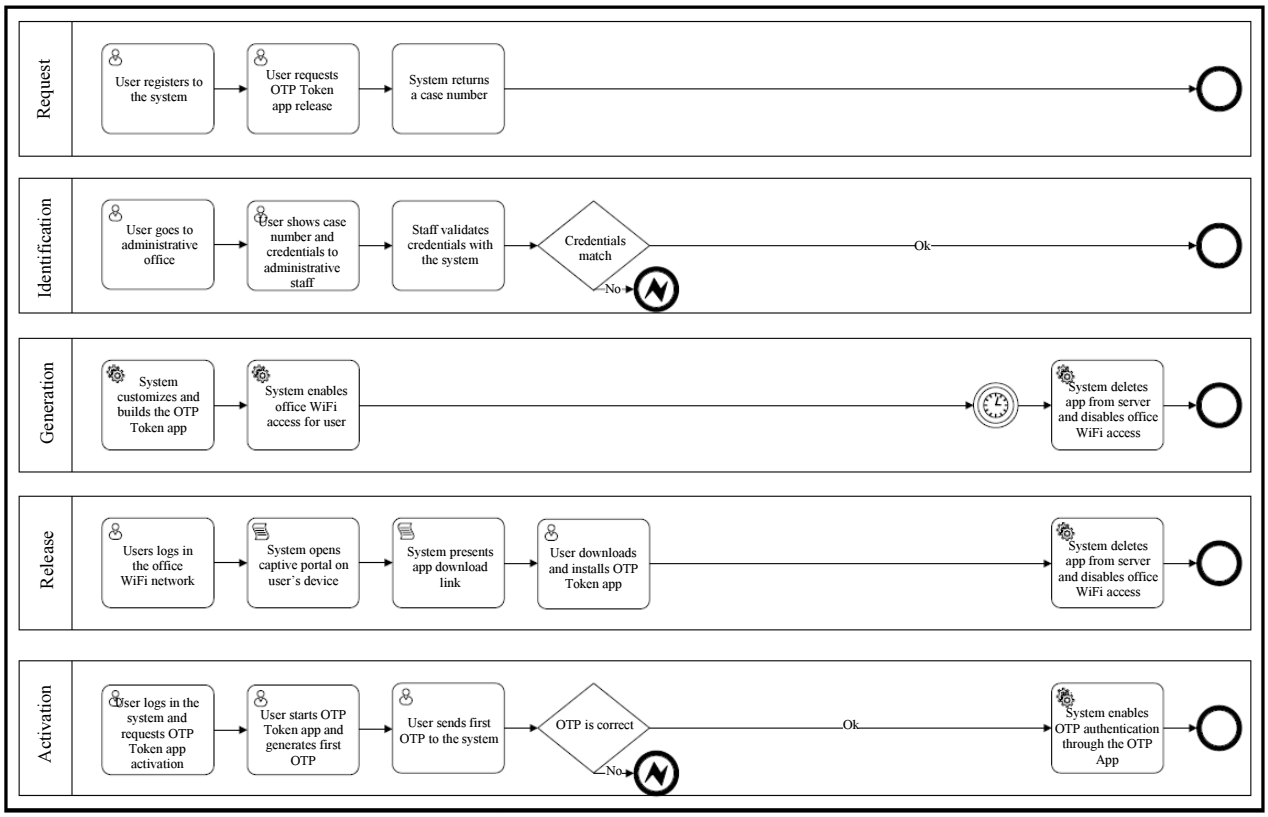

Fig. 1: The AH-OTP app release process

In particular, the AH-OTP app is not distributed through the usual release channels (i.e., the *stores*) but it is *ad-hoc* built for each user and embeds his secret key (used to generate the OTPs) and the IMEI of his mobile phone. Such an ad-hoc app is released after a process which includes a physical verification of the user identity documents and can be downloaded and installed on the device only within a safe environment (i.e., an *ad-hoc* WiFi network). The overall process is structured in such a way that no human or software owns all the elements needed to complete it. Indeed, the process completion requires an interaction between three parties: the user, a software system and an administrative employee.

Process Overview

The AH-OTP release process is structured in five steps:

Step 1- Registration and App Request: The user registers to the service, which also requires to specify his identity card number and requests the AH-OTP app release, specifying his smartphone model and IMEI code. The system returns a *case number* to be used afterwards in the process.

Step 2- Physical Identification: The identification takes place into an administrative office. The user gives to an authorized staff member his case number and identity document, which is checked against the data given during the registration step.

Step 3- App Generation: The system builds an ad-hoc instance of the AH-OTP app, which embeds the user randomly generated secret key and his smartphone IMEI.

Step 4- App Download: The user connects to the administrative office WiFi network using its registration credentials and is guided through the app download and installation process.

Step 5- App Activation: The user logs again in the online system and requests the app activation. He is asked to run the AH-OTP app, generate an OTP and enter it in a form submitted to the system. If this OTP is verified, then the app is considered correctly installed and is enabled.

From the process summary above, it should be clear that the process has been designed to be very similar to the one usually adopted in the distribution e-banking tokens. Therefore, it should be simple and intuitive to follow by any citizen that also uses online banking systems. On the other hand, thanks to the particular App structure and installation process, the proposed solution is very hard to attack, achieving a security degree very similar to the hardware security tokens, still preserving the convenience deriving from the use of a software OTP generators installed on the

user's smartphone. To better understand these points, let us give a more detailed description of the five steps above.

Process Details and Discussion

In this section we give more technical details about the development of each step of the proposed process. We assume that state-of-the-art, standard security technologies are employed where needed: These basic "security assumptions" are highlighted within the description below:

Step 1- Registration and App Request: The user connects to the authentication portal and registers to the service. The registration data include the user identity card number. The system returns a valid user ID and password. Once logged with these credentials, the user requests the AH-OTP app release, specifying the mobile device model and IMEI code. The system provides a *case number* linked to the activation request and invites the user to go to the nearest administrative office to complete the process.

Security assumption 1 (Weak authentication on web clients). *To achieve a better usability on the web client, the registration step uses weak authentication (e-mail check), since the identity will be later physically verified in the administrative office (step 2).*

Step 2- Physical Identification: At the administrative office, the user declares his assigned case number and is physically identified by the authorized staff through his identity document, whose data is compared to the registration data entered in the first step. To further enforce security, the staff cannot add or update any identification data, i.e., they have a read-only access to the system and can only execute the "confirm identity" action. Note that such a step, even if it makes the overall process longer and more complex both for the citizen and for the administration, cannot be skipped in an e-government scenario. Indeed, in most of the national law systems, the physical identification of a citizen is a prerequisite for the release of any identity-related artefact.

If the identification succeeds, the system starts the AH-OTP app generation and distribution process, which is completely automatic.

Security assumption 2 (Not falsifiable identity document). *Identity documents are always a reliable certification of the user identity. In particular, identity documents are compliant with the international standards (ICAO 9303, ISO/IEC 7810, ISO/IEC 7816) and contain verifiable security elements such as markers, holograms, etc. which allow a quick visual check of the document validity IOS (2003). Furthermore, we can assume that the administrative office can also exploit hardware devices that scan the document and verify its integrity e.g., IOS (2003); ICAO (2015).*

Security assumption 3 (Strong employee fairness policy). *Administrative employees should never have the user smartphone in their hands and, more in general, get any object from the user except his identity card. This strong policy is part of the employee contract and failing to adhere to it results in disciplinary actions. Such rule is also recalled inside the administrative office and clearly written in the registration website.*

Security assumption 4 (Office with video surveillance). *The administrative office is equipped with cameras that record the customer/employee interactions. In particular, the video surveillance system can be used to detect disallowed actions (as user objects given to the employees). Note that this detection can take place whenever an anomaly is notified by the citizen (i.e., the video must recorded and stored for later use) or, in the future, by using a real-time automatic image analysis software.*

Step 3- App Generation: The system generates and stores a random secret key associated with the user. Then, the AH-OTP app is built from the sources specific for the target device indicated by the user in the first step. The app is customized embedding the user secret key and the mobile device IMEI number in its obfuscated binary code. Therefore, such information will never be stored, even in an encrypted form, in the device data storage unit. This ensures that the app cannot be moved to another device, thus making it a permanent piece of the mobile device software and makes the OTP generation parameters very difficult to steal from the device itself.

Security assumption 5 (Obfuscated app data). *The target device IMEI and the OTP secret key are hard-coded in the app binary. To prevent an attacker to obtain such information by reverse engineering the app, its code is obfuscated (through one of the available tools like, e.g., PROGUARD for Java (GuardSquare nv, 2017). Moreover, the source code is randomly interleaved by meaningless code lines, to make understanding the decompiled binary more difficult. Finally, the sensible data are not included as simple constants, but rather split in a random number of pieces which are assigned to variables with random names and different types declared among the code e.g., Drape et al. (2007) and are merged on demand to re-generate the original data.*

Step 4- App Download: At this point, the system enables the user to access the office WiFi through the credentials released in the first step.

Security assumption 6 (Wi-Fi security). *The administrative office WiFi network makes use of standard communication security techniques. Moreover, the office WiFi network is continuously scanned in order to verify that only one network exists with the requested*

SSID and that the network correctly responds to a challenge-response procedure. If the scan fails, the staff is notified and all the download processes are disabled.

The user connects to the office WiFi network with his credentials and is redirected to a captive portal with a single link that can be used to download and install the application instance built in the previous step (clearly such a link can be accessed only within the WiFi LAN). The installation procedure, specific for the user's device and operating system, is illustrated on the same web page. Both the link and the WiFi access are available for a small time interval, after that the WiFi access is disabled and the app is deleted from the server. This also happens once the user successfully downloads the app.

It is worth noting that forcing the user to download and install the application only within a safe, controlled environment and in a limited time slot is required to give to the app, in our process, the same security features of the hardware tokens in e-banking scenarios. Indeed, in this way we may consider the app, in some sense, directly consigned to the user by the administration after the identification, as hardware tokens are physically given to the user after their identity is confirmed.

Allowing the application to be downloaded later, for example at user's home, would open a number of attack scenarios since, for example, we cannot assert the security of a generic WiFi network. To mitigate these attacks, we would need to introduce further levels of security in the process, e.g., by giving the user (secret) download codes and checksums to verify the downloaded app before installing it. This would make the overall process more complex and, possibly, expensive for the administration.

On the other hand, we may safely assume that nowadays the main administrative offices are already equipped with an internal Wifi network and that the user involved in the AH-OTP process wants to complete it as soon as possible in a place where he may find further assistance if needed. Therefore, forcing him to download the app immediately is not a real restriction.

Step 5- App Activation: In the last step, the user logs again in the online system using his credentials and requests the app activation. He is asked to run the AHOTP app that has been installed on his mobile device.

Each time it is started, the app verifies that the device IMEI corresponds to the one embedded in its code and that the device operating system is not *rooted* or *jailbreaked*, i.e., unlocked. This last check is performed to further enforce the app security since, in rooted phones, apps are free to access and modify the operating system services as well and other apps (Hassan and Pantaleon, 2017).

If both the above checks succeed, the app presents a simple, standard interface with a button that, when pressed, generates a, say, 30 sec valid OTP. This password is derived, using the standard TOTP algorithm, from the device internal clock and the user secret stored in the app binary. When the password expires, the user can click again the button to get a new one.

The user enters the generated OTP in a form and submits it to the system. If this OTP is verified, then the app is considered correctly installed and properly working, thus its use is enabled for all linked the e-Government tasks.

Again, if the activation is not performed within a reasonable amount of time after the release, the system invalidates the app (in particular by forgetting the associated secret), so that it must be uninstalled and requested again.

AH-OTP Security

In this section we analyse a number of possible attacks that could be achieved on the AH-OTP app itself or during its release. Each attack is described in detail, together with the conditions that should make it infeasible in the practice, given the security assumptions of Section 4. Such issues will be better formalized in Section 6.

Table 1 summarizes the considered attacks. In particular, the attacks are classified in two types:

- **An Outsider attack** is performed by someone not involved in the AH-OTP app release process.
- **An Insider attack** is performed by a malicious employee of the administrative office where the app is actually released (steps 2-4 of the process described in the previous section)

Table 1: Summary of the analysed attacks

Attack name	Attack type
Fake User Profile	*Outsider*
Compromised User Client	*Outsider*
Stolen/Fake Identity Document	*Outsider*
App Copy	*Outsider*
Smartphone OS Manipulation	*Outsider*
Secret Key Copy	*Outsider*
Phone Consignment	*Insider*
WiFi intrusion	*Insider*
Registration Data Manipulation	*Insider*
App Download	*Insider*

Table 2: Security elements

Element	Description
Name	The user first name
Surname	The user last name
Email	The user email address
UserID	The user account ID
Password	The user account password
IMEI	The user phone IMEI
(User) profile	The name, surname and email
(User) credentials	The user ID and password
(Identity) document	The user physical identity document
(Secret) key	The user secret key
Application (file)	The application executable package
(Office) WiFi	The administrative office WiFi network
(Smartphone) OS	The user mobile operating system

Element	U	S	E	A
Profile		☑	☑	☑
Credentials		☑		☑
IMEI		☑		☑
Document			☑	
Key		☑		
Application		☑		
WiFi		☑		
OS				☑

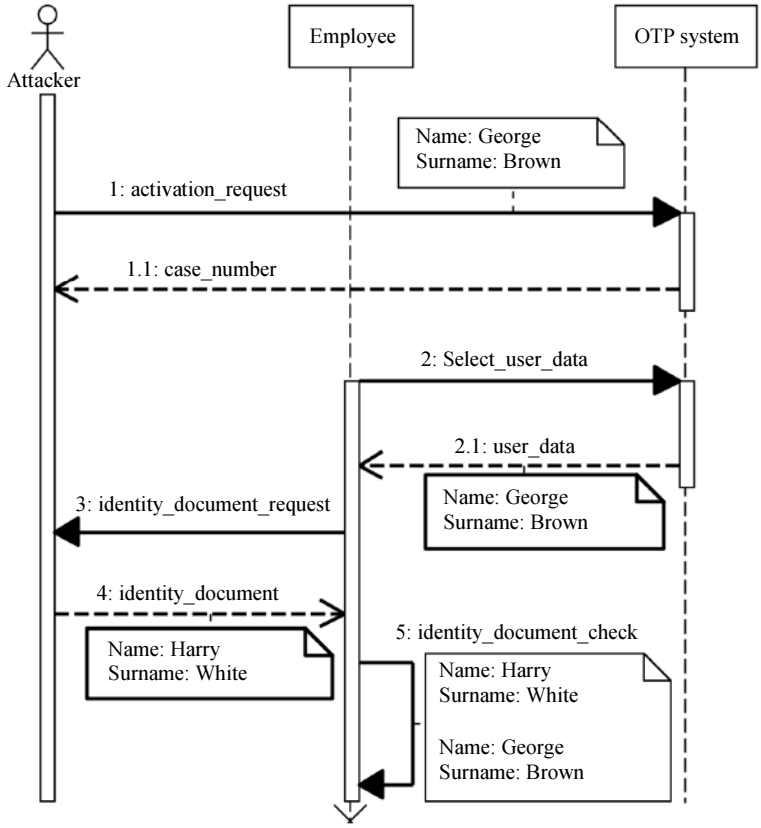

Fig. 2: ”Fake user profile” attack

To simplify the analysis, in the following we will make reference to a set of meaningful *vulnerable elements*, described in Table 2, that can be exploited by an attacker to break the process security.

Outsider Attacks

Fake User Profile

The attacker tries to get an AH-OTP App using a fake user profile.

The attacker creates a user profile by entering false data and requests the release of an AH-OTP app. In the administrative office, the employee selects the profile and verifies the data by checking the identity document.

Figure 2 shows the attack progress and the elements involved, where a checkmark in a cell means that the actor (User, System, Employee and Attacker) associated with the column owns the element identified by the row. Moreover, we use the text style to denote the role of each element in the attack:

- *Italic style* indicates that the element is being exploited for the current attack
- **Bold style** indicates that the element is blocking the current attack

- Normal (not-bold, not-italic) style indicates that the element is not being involved in the current attack. Italic style indicates that the element

In this case, the attack is blocked by the identity document check (security assumption 2).

Compromised User Client

The attacker tries to get an AH-OTP app using real identity data stolen through some malware installed on the user client.

The user, who is unaware of operating on a compromised machine, registers and issues the request to activate the AH-OTP app. The attacker collects all the data entered by the user during such process (i.e., the complete user profile) and goes to the administrative office to obtain the app. Figure 3 shows the attack progress and the elements involved. Again, the attack is blocked by the identity document check.

Stolen/Fake Identity Document

The attacker tries to get an AH-OTP app using a stolen or fake identity document.

The attacker steals or falsifies an identity document and requests the activation of the AH-OTP app using data from such document. Figure 4 shows the attack progress and the elements involved. Again, the attack is blocked by the identity document check.

Element	U	S	E	A
Profile	☑	☑	☑	☑
Credentials	☑	☑		☑
IMEI	☑	☑		☑
Document	☑		☑	
Key		☑		
Application		☑		
WiFi		☑		
OS				☑

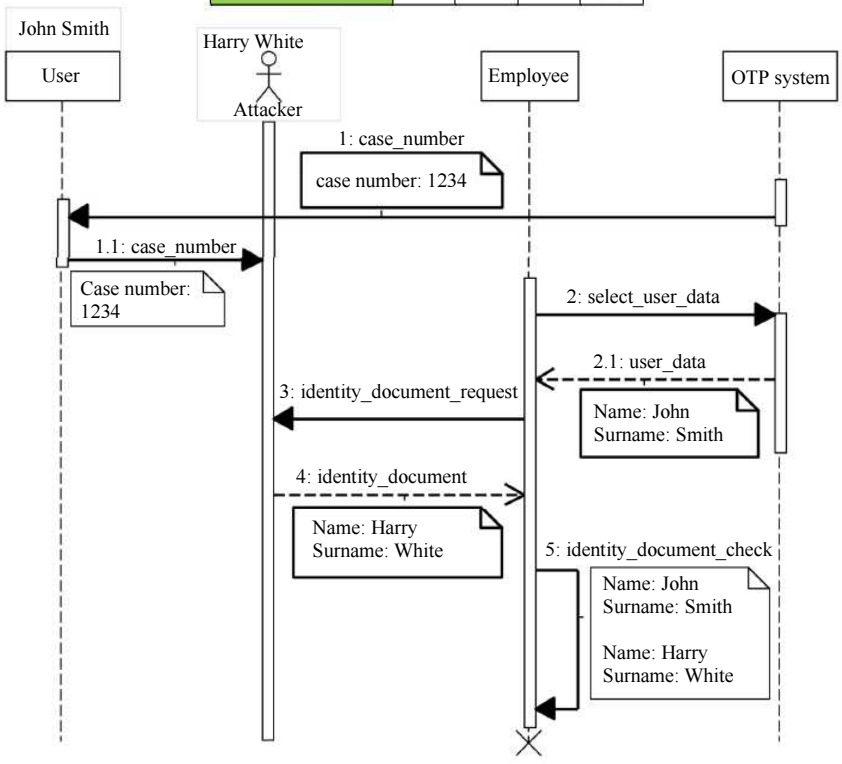

Fig. 3: "Compromised user client" attack

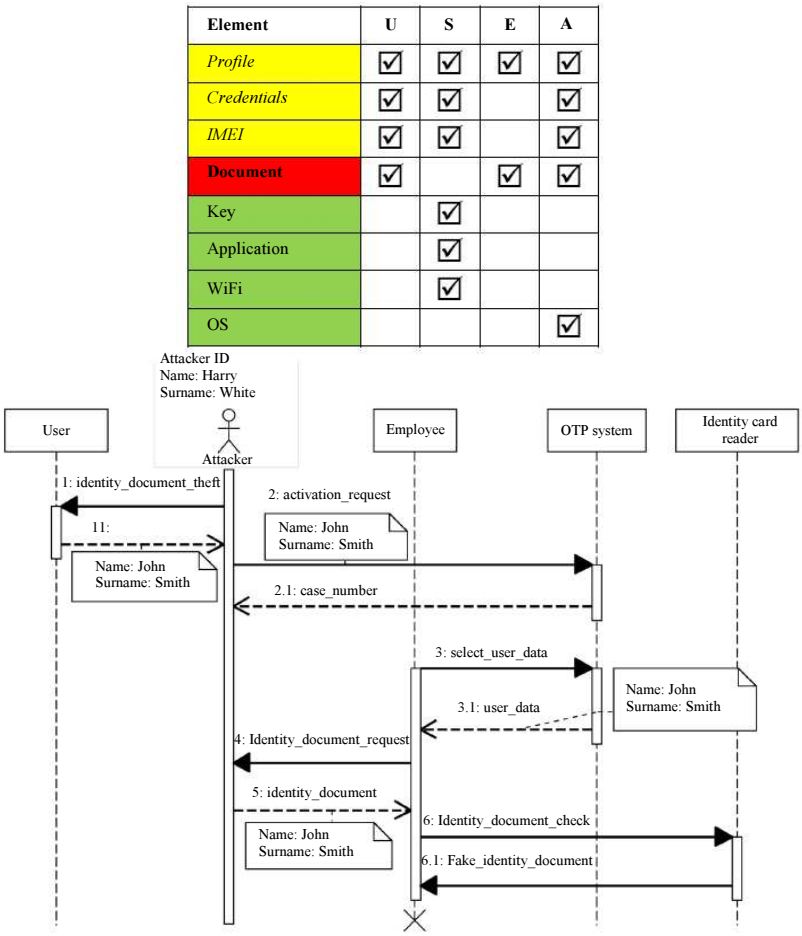

Fig. 4: "Stolen/fake identity document" attack

App Copy

The attacker tries to copy the AH-OTP app to another smartphone.

The attacker extracts AH-OTP application file (e.g., the APK file on Android devices) from the user smartphone, installs it on another device and tries to use it to generate an OTP. Figure 5 shows the attack progress and the elements involved. The attack is blocked by the codelevel binding between the app and the user phone IMEI (security assumption 5).

Smartphone OS Manipulation

The attacker tries to copy the AH-OTP app to another smartphone and manipulates the OS to return a specific IMEI code and/or disable other security checks.

The attacker wants to bypass the smartphone OS security, in particular to return a specific IMEI to the app. To achieve such modifications on the phone operating system, the attacker has to obtain root permissions on the device. Figure 6 shows the attack progress and the elements involved. The attack is then blocked by the "rooted device" verification executed at the application startup.

Secret Key Copy

The attacker tries to copy the AH-OTP App secret key to use it in another OTP token application.

With the secret key, the attacker could use another standard OTP token to generate the same OTP sequence of the user. Figure 7 shows the attack progress and the elements involved. The attack is blocked by the hard-coding of the secret key in the obfuscated app code (security assumption 5).

Insider Attacks

Phone Consignment

An administrative employee tries to get the user phone.

In the administrative office, an unfaithful employee asks the user to consign the phone (thus deliberately violating security assumption 3).

Element	U	S	E	A
profile	☑	☑	☑	
credentials	☑	☑		
IMEI	☑	☑		
document	☑		☑	
key		☑		
application	☑	☑		☑
WiFi		☑		
OS	☑			

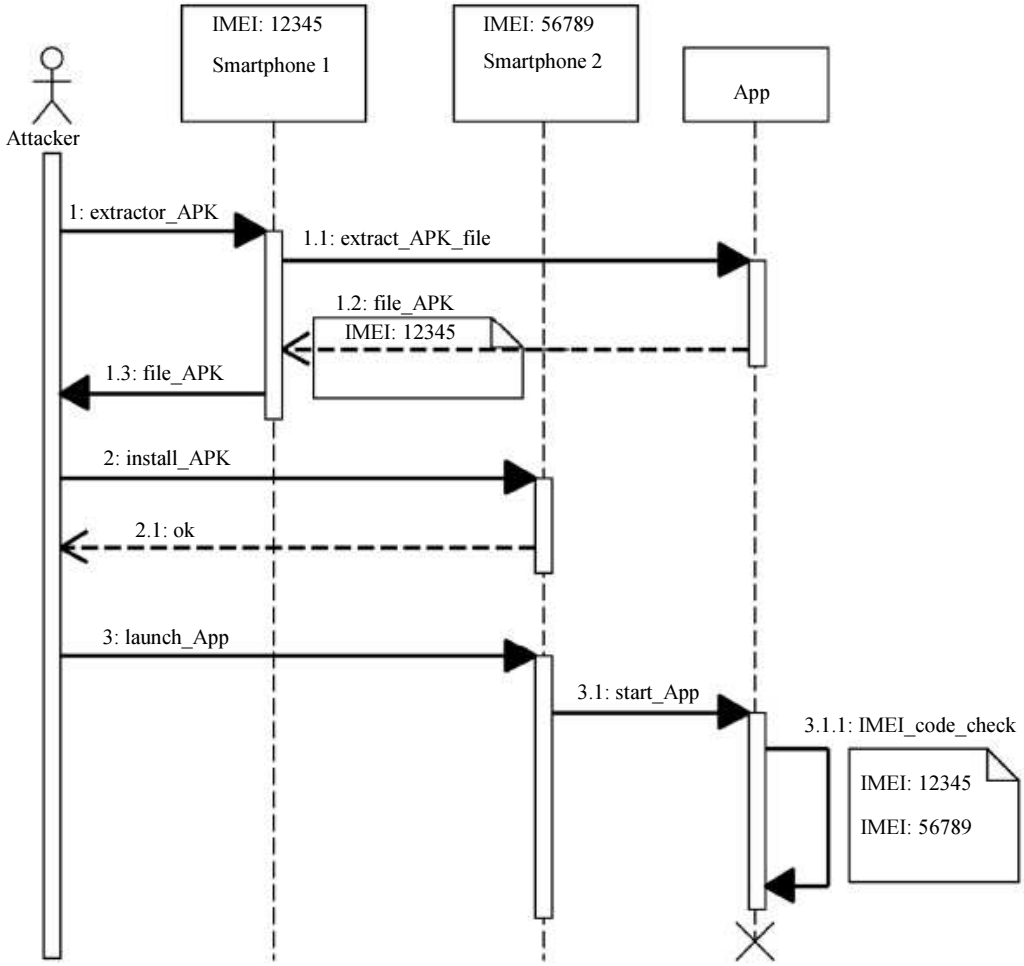

Fig. 5: ”App Copy” attack

In this way, the employee can read the phone memory, extract applications or read its IMEI. Figure 8 shows the attack progress and the elements involved. The attack is blocked by the video surveillance that records the objects passed between the employee and the user (security assumption 4).

Element	U	S	E	A
profile	☑	☑	☑	
credentials	☑	☑		
IMEI	☑	☑		☑
document	☑		☑	
key		☑		
application	☑	☑		☑
WiFi		☑		
OS	☑			☑

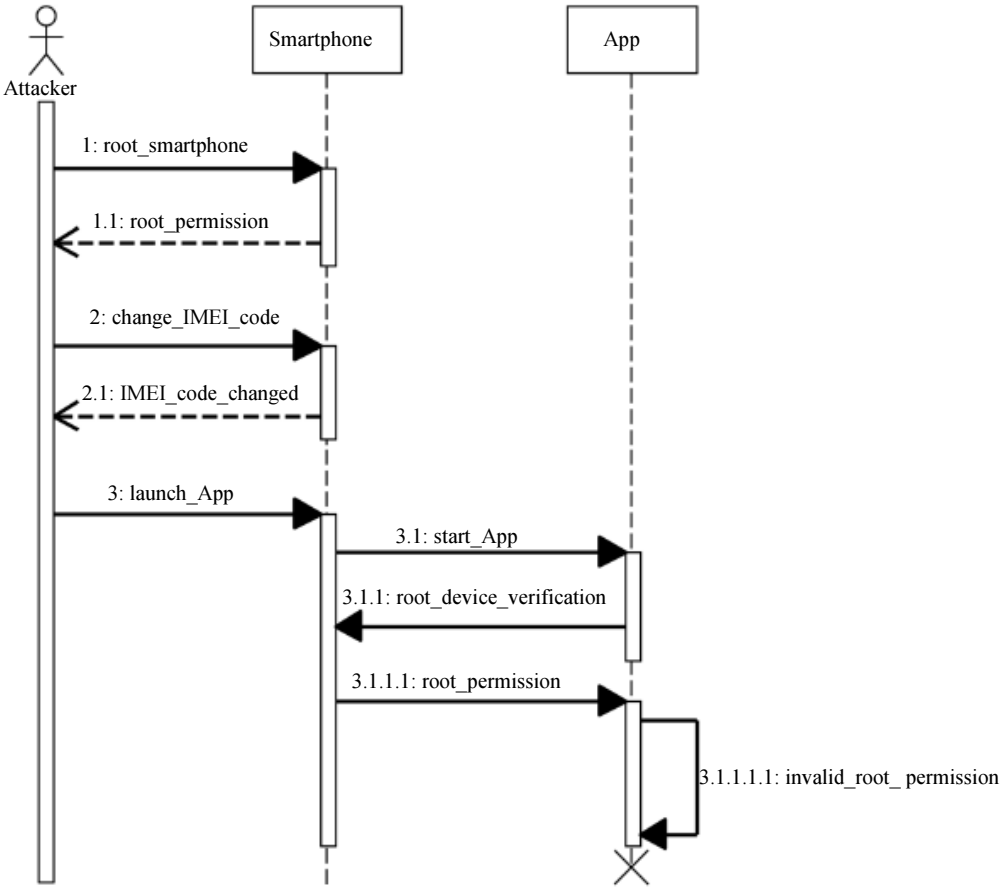

Fig. 6: ”Smartphone OS manipulation” attack

Wifi Intrusion

The attacker tries to break into the administrative office WiFi network.

In the administrative office, the attacker tries to interfere with the WiFi network to intercept, read or modify the data exchanged between the system and the user. Figure 9 shows the attack progress and the elements involved. The attack is blocked by the presence of an Intrusion Detection System that analyses the traffic in order to identify anomalies or intrusions (security assumption 6).

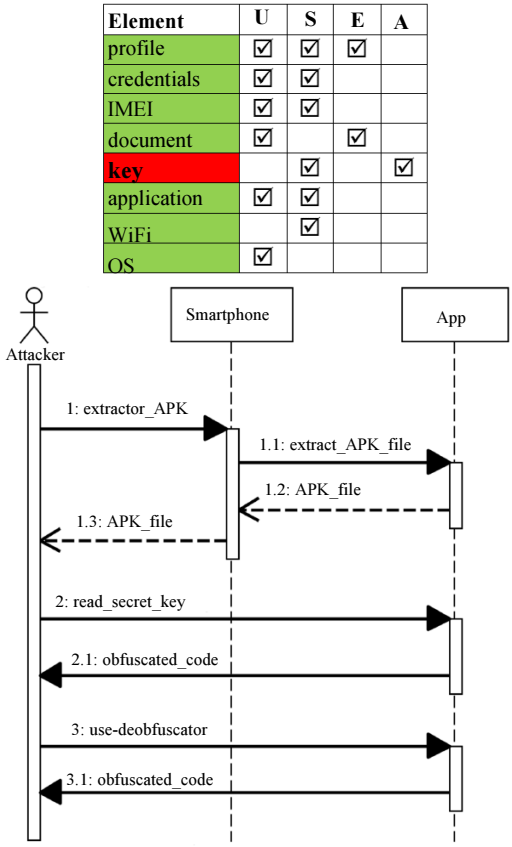

Fig. 7: ”Secret Key Copy” attack

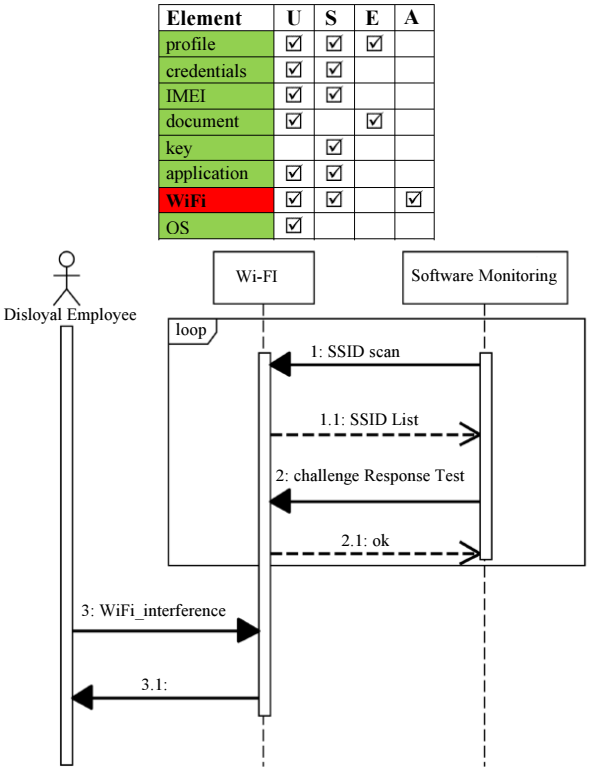

Fig. 9: ”WiFi Intrusion” attack

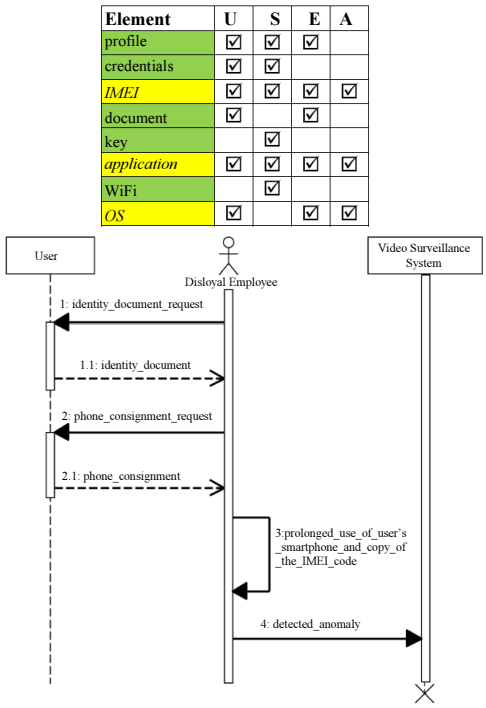

Fig. 8: ”Phone consignment” attack

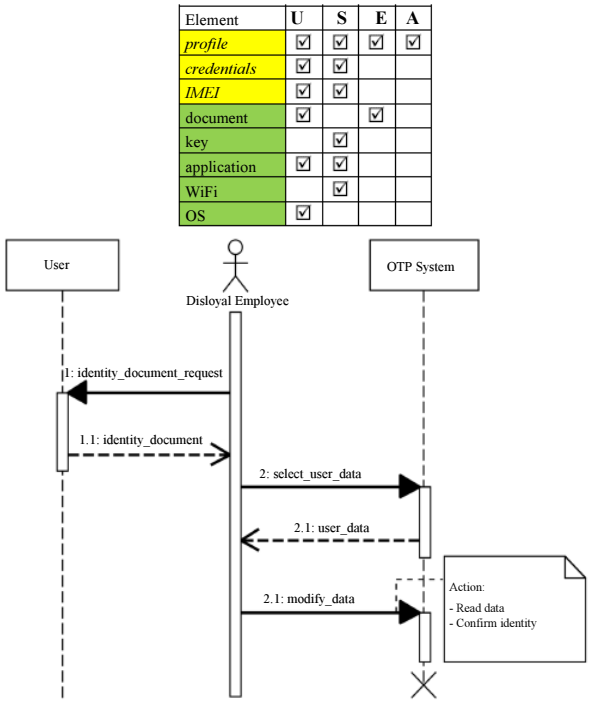

Fig. 10: ”Registration data manipulation” attack

Registration Data Manipulation

The administrative employee tries to change the user registration data.

In the administrative office, an unfaithful employee tries to modify the user profile data in order to match a different identity document. Figure 10 shows the attack progress and the elements involved. The attack is blocked since the registration data is read-only for the administrative office employees.

App Download

The administrative employee tries to download the OTP token app of another user.

After the user has been successfully identified and the release phase of the AH-OTP app has started, an unfaithful employee convinces the user to reveal his credentials and tries to use them to download the OTP token application on another smartphone. Figure 11 shows the attack progress and the elements involved. The attack is blocked by the code-level binding between the app and the user phone IMEI.

AH-OTP Security Verification

Although the discussion following each attack described in Section 5 may be convincing, so far we have no definitive evidence that such attacks, or a sequence of them executed in some order, may not break the security of the proposed approach. To this aim, in this section we develop a formal model for both the AH-OTP process described in section 4 and the attacks listed in Section 5 and then use model checking to verify if the attacks can break the process in some way. As it is well known, model checking techniques, applied on a correct model of a system, are able to produce a formal, mathematically correct proof of any suitably modelled system property (see, e.g., Burch *et al.* (1992); Dill *et al.* (1992); Holzmann (1991) for a general introduction to model checking).

Element	U	S	E	A
profile	☑	☑	☑	
credentials	☑	☑	☑	☑
IMEI	☑	☑		
document	☑		☑	
key		☑		
application	☑	☑	☑	☑
WiFi		☑		
OS	☑			

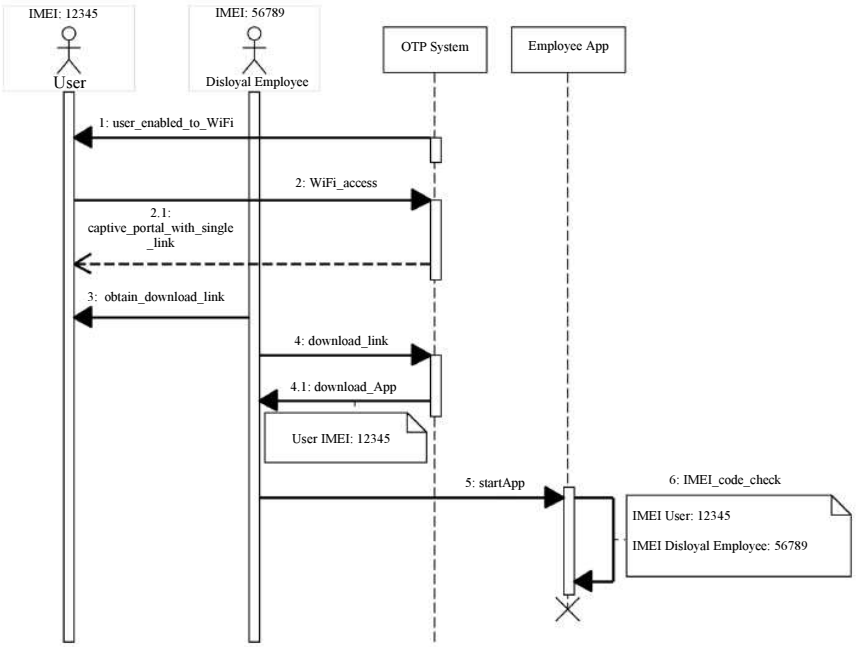

Fig. 11: ”App download” attack

The Verification Tool

To perform model checking, in this paper we make use of the CMurφ tool (Della Penna *et al*., 2004), a fork of the Murφ model checker (SU, 2004) originally developed by the SU, (2004) which has been often used for the verification of security protocols and hardware systems. CMurφ can be downloaded from (URLS, 2017).

The CMurφ input consists of a definition of the system to be verified and a definition of the property to be checked. Both definitions are encoded in the CMurφ description language, a high-level Pascal-like programming language that offers many features also found, e.g., in C or Java.

More in detail, the CMurφ input contains declarations of constants, types, global variables, procedures and functions, followed by a collection of *transition rules*, a description of the *initial states* and a set of *invariant properties*. The system model itself is given by the collection of transition rules: Each rule is a guarded command with a condition (a Boolean expression on the global variables) and an action (a statement that can modify the global variables).

Roughly speaking, CMurφ starts from the given initial state(s) of the system and applies all the enabled transition rules to generate all the possible next states in a loop called *explicit state space exploration*. Of course, known states are not regenerated, thus the exploration always ends in a finite state system. On every generated state, the tool checks the value of the invariant properties: If all the properties are true, the exploration continues, otherwise it stops and the tool reports the violated invariant and (optionally) the sequence of rules/actions that lead to the error, which constitute a counterexample for the invariant property. On the other hand, if the exploration ends without errors, then we have a certification that the invariant always holds in the system. Note that CMurφ models may be nondeterministic, since different rules can be active on the same state: In this case, the model checker verifies that the property holds regardless of the chosen rule.

The AH-OTP Process Model

To make the CMurφ model of the AH-OTP process and its possible attacks easily understandable also by non-expert users, we try to remain as adherent as possible to the terminology used in Section 5. In the following we report the most important parts of the model, whereas the complete source code is available upon request from the authors.

We start by defining the normal AH-OTP Process as reported in Section 4. Figure 12 shows the declarations: Here we define enumerated constants for all the process actors (user, attacker, system: For sake of simplicity, we merged the malicious employee with the attacker) and the security elements explained in section 5. Note that, to better model the concepts of "valid document" and "profile modification rights" we introduced here the elements validdocument and profilemodify. Moreover, the new OTP element indicates a fully-working OTP generator, i.e., it is the last element obtained by the user after the activation phase. Finally, the process state is an array representation of the element/actor tables used in Section 5 to indicate that a particular actor owns an element.

```
Type
  -- human actors
  mainactors: Enum {user,attacker};
  -- all the actors
  actors: Union {mainactors,Enum{system}};
  -- watched elements
  elements: Enum {
    profile, --user profile
    profilemodify, --profile modification access
    credentials, -- user credentials
    IMEI, --phone IMEI
    document, --identity document
    validdocument, --valid identity document
    key, --secret key
    application, --application for IMEI and key
    WiFi, --office WiFi control (mutually exclusive
      )
    validos, --non-rooted phone OS
    OTP --working OTP generator
  };
Var
  -- process state
  element: Array[actors] of Array[elements] of
    boolean;
  -- element[actor][name] is true if the actor
  owns the element
```

Fig. 12: AH-OTP process model: Declarations and global state

```
-- tests whether an actor has an element
Function has(a : actors; e : elements): boolean;
Begin
  return (element[a][e]=true);
End;
-- removes an element from an actor
Procedure releases(a : actors; e : elements);
Begin
  element[a][e]:=false;
End;
-- gives an element to an actor
Procedure gets(a : actors; e : elements);
Begin
  element[a][e]:=true;
  -- some resources have mutually exclusive
    ownership
  if (e=WiFi) then
    for OA:actors Do
      if (OA!=a) then releases(oa,e); endif;
    end;
  endif;
End;
-- Startstate initializer
Procedure commonInit();
Begin
  -- everybody has nothing
  For e : elements Do
     For a : actors Do releases(a,e); End;
End;
-- normally, we start with the user having
gets(user,profile); -- his personal data
gets(user,IMEI); -- the IMEI of his phone
gets(user,document); -- a valid identity
  document
gets(user,validdocument);
gets(user,validos); -- a non-rooted phone OS
gets(system,WiFi); -- office wifi network is ok
gets(user,validos); -- attacker has his own
  mobile
End;
```

Fig. 13: AH-OTP process model: support functions

```
-- the following actions can be taken by both the
   user and the attacker
Ruleset actor : mainactors Do
Rule "1.Registration and App Request"
  !has(actor,credentials) -- if actor not already
     registered
  & !has(system,profile) -- and there is no
     identical profile in the system
  & has(actor,profile) & has(actor,IMEI) -- and
     actor has the required information
==>
Begin
  -- actor gets his credentials
  gets(actor,credentials);
  -- system stores actor information
  gets(system,credentials);
  gets(system,profile);
  gets(system,IMEI);
End;
Rule "2.Physical Identification"
  !has(system,document) -- if actor not already
     identified
  & has(system,profile) & has(system,credentials)
     -- actor registered
  & has(actor,profile) & has(actor,credentials) &
     has(actor,document) & has(actor,validdocument)
     -- actor has required information and a valid
     document
==>
Begin
  -- system gets (validates) the document
  gets(system,document);
End;
Rule "4.App Download"
  !has(actor,application) -- if app not downloaded
  & has(actor,credentials) -- actor registered
  & has(system,application) & has(system,
     credentials) & has(system,key) -- app
     generated
  & has(system,WiFi) -- WiFi is ok
==>
Begin
  -- actor downloads app
  gets(actor,application);
  -- system deletes app
  releases(system,application);
End;
Rule "5.App Activation"
  !has(actor,OTP) -- if app not activated
  & has(actor,application) -- app downloaded
  & has(actor,IMEI) & has(actor,validos) -- actor
     device is ok
  & has(system,key) -- system has secret key
==>
Begin
  -- actor activates app
  gets(actor,OTP);
End;
End; -- ruleset
-- this rule is not actor-dependant
Rule "3.App Generation"
  !has(system,application) -- if app not generated
  & has(system,profile) & has(system,IMEI) --
     profile registered
  & has(system,document) -- profile identified (
     system has his document)
==>
Begin
  -- system generates secret key and app
  gets(system,key);
  gets(system,application);
End;
```

Fig. 14: AH-OTP process model: process rules

Then, we define some useful support functions and procedures, reported in Fig. 13. It is worth noting how we initialize the process in function commonInit by giving to each actor the elements that he should initially own.

```
-- the attacker registers to the service using a
   fake (or stolen) profile
Rule "A1.Fake user profile"
  !has(attacker,credentials)
==>
Begin
  -- the attacker gets the credentials relative to
     his fake identity
  gets(attacker,profile);
  gets(attacker,IMEI);
  gets(attacker,credentials);
  -- the attacker has an identity document for the
     profile (but not valid!)
  gets(attacker,document);
End;
  -- the attacker spies the user client and gets
     all the registration data
Rule "A2.Compromised user client"
  !has(attacker,credentials)
  & has(user,profile)
  & has(user,IMEI)
==>
Begin
  -- the attacker gets the user profile, IMEI
     number and credentials
  gets(attacker,profile);
  gets(attacker,IMEI);
  gets(attacker,credentials);
  -- the attacker may also have an identity
     document for the profile (but not valid!)
  gets(attacker,document);
End;
  -- the attacker stoles the user identity document
Rule "A3.Stolen identity document"
  !has(attacker,document)
==>
Begin
  -- the attacker gets the user document, but not a valid one
  -- since the theft is known to the police
     gets(attacker,document);
End;
  -- the attacker tries to copy the app from the user phone
Rule "A4. App copy"
  has(user,application) -- app downloaded
  & !has(attacker,application)
==>
Begin
  -- attacker stoles user app
  gets(attacker,application);
End;
  -- the attacker (after getting the user phone
     IMEI, e.g., with rule A2) forces his phone OS to
       report the stolen IMEI)
Rule "A5.OS manipulation"
  has(attacker,validos) -- the attacker phone is initially non-rooted
  & has(attacker,IMEI) -- the attacker has the user IMEI
==>
Begin
  -- the attacker manipulates the OS of his phone to
  -- possibly return a different IMEI
  releases(attacker,validos); -- attacker phone is
     now rooted
End;
```

Fig. 15: AH-OTP process model: Attack rules, part 1

Then core of the model is given by the transition rules in Fig. 14. Each rule is named as the corresponding AH-OTP process step and commented, so it should be easy to read. In particular, the rule guard, written before the ==> symbol, makes each rule available only when the state meets its preconditions. The rule-set statement allows us to make the process steps (except step 3) available to both the user and the

attacker (here generically called actor), so the attacker can interfere, when possible, with the process followed by the user. As an example, rule 1 ("Registration and App Request") can be executed only if the actor has no system credentials yet !has (actor,credentials), but has all his profile information available as well as the IMEI of his device has (actor,profile) and has (actor,IMEI). Obviously, the rule cannot be executed if the system already has an identical profile registered (!has (system,profile). When the rule is fired, the actor obtains his credentials gets (actor,credentials) and the system registers all of his data gets (system,credentials); gets (system,profile); gets (system,IMEI). Note that, when the process ends (rule 5), the actor (user or attacker) gets the OTP element, i.e., the fully working OTP generator.

Now we are ready to model the attacks. Each attack is encoded in a single rule, as shown in Fig. 15 and 16, whose name recalls the corresponding attack described in Section 5. The guards of these rules allow each attack to be launched whenever possible during the normal process, so the attacker can interfere with the user in various ways and also perform attack combinations. Remember that CMurφ will try every possible sequence of allowed rules, so it will check any possible attack configuration. As an example, with rule A2 ("Compromised user client") the attacker steals the profile, IMEI and credentials gets (attacker,profile); gets (attacker,IMEI); gets (attacker,credentials)) from the user web interaction and also obtains the corresponding (false) identity document gets (attacker,document).

Finally, we define the system initial state (via the commonInit procedure in Fig. 13) and the invariant: "The attacker must never have a fully working OTP application", which is encoded as! has (attacker,OTP), as shown in Fig. 17.

Verification Results

When running the model above through CMurφ, we obtain the report shown in Fig. 18. Note that part of the report headers, which describe some verification technical details, have been omitted for sake of brevity. The statistics show that there are 132 possible states in our model (corresponding to all the possible process and attacks inter-leavings) and that some rules were never used (fired): This happens since some attacks, like the "Registration data manipulation" are impossible by design, as explained in Section 5, so the corresponding rule cannot fire.

The model checker found no errors, meaning that the invariant always holds. Therefore, within the limits of the model correctness and all the security assumptions it relies on, *the attacker never obtains a working OTP generator*.

```
-- the attacker tries to get the secret key from
   the user to exploit it in his personal copy of
   the OTP app
Rule "A6.Secret Key Copy"
   !has(attacker,key)
   & has(user,key) -- the key is somehow obtainable
     from the user or his device
==>
Begin
   -- the attacker gets the user key
   gets(attacker,key);
   -- note that, since the key is embedded in the
      app, this action is useless
End;
-- the attacker gets the user phone and copies
   his IMEI (copying the app itself is handled by
   rule A1)
Rule "A7.Phone consignment"
   !has(attacker,imei)
==>
Begin
   -- attacker gets the phone and reads its IMEI
   gets(attacker,IMEI);
End;
-- the attacker tries to intercept the office
   WiFi network during the app download
Rule "A8.Wifi intrusion"
   !has(attacker,WiFi)
==>
Begin
   -- the attacker takes control of the office WiFi
   gets(attacker,WiFi);
End;
-- ther attacker tries to modify the user
   registration data in order to match a different
   identity document
Rule "A9.Registration data manipulation"
   has(attacker,profilemodify) -- the attacker can
     modify the profile: this will never happen
==>
Begin
   -- the attacker has his own valid document
   gets(attacker,document);
   gets(attacker,validdocument);
   -- the user profile becomes the attacker profile
   gets(attacker,profile);
   gets(attacker,IMEI);
End;
-- the attacker tries to download the app before
   the user installs it on his phone
Rule "A10. App download"
   has(system,application) -- app ready (not
     downloaded)
   & !has(attacker,application)
==>
Begin
   -- attacker stoles user app
   gets(attacker,application);
End;
```

Fig. 16: AH-OTP process model: Attack rules, part 2

```
-- startstate: initializes the process
Startstate "OTP main"
Begin
  commonInit();
End;
-- invariant: the attacker should never own a
   working otp generator
Invariant "attackerCanUseOTP"
  !has(attacker,otp);
```

Fig. 17: AH-OTP process model: Start state and invariant

```
Caching Murphi Release 5.4.9
========================================
Protocol: AH-OTP
Algorithm:
   Verification by breadth first search.
   with symmetry algorithm 3
      -- Heuristic Small Memory Normalization
      with permutation trial limit 10.
Status:
   No error found.
State Space Explored:
   132 states, 573 rules fired in 0.10s.
Omission Probabilities (caused by Hash Compaction):
   Pr[even one omitted state] <= 0.000000
   Pr[even one undetected error] <= 0.000000
   Diameter of reachability graph: 10
Analysis of State Space:
   There are rules that are never fired.
```

Fig. 18: CMurφ verification results

Conclusion

In this paper we presented an one-time password generation mechanism, which has been specifically designed to support two-factor authentication schemes in e-government processes. In this sense, it aims to offer the highest security without sacrificing usability and accessibility, i.e., it is a good compromise between the increasing security requirements of e-government authentication schemes and the digital divide, which still prevents a part of the population from accessing digital government services.

In particular, the ad-hoc nature of the proposed OTP generator application, together with its specific generation and distribution process, makes our solution secure as an hardware token and, at the same time, easy to use as a smartphone app and has a limited cost for both the user and the administration. This is in contrast with most of the current similar approaches, which are often unbalanced and sacrifice security for usability or have relatively high implementation costs.

We developed prototypes of all the software artefacts supporting our authentication process, including the registration website, the app customization and building system and the AH-OTP app itself for the Android platform. Since the whole process relies on a combination of known technologies, the development of such applications is not complex and does not require big investments in terms of time, efforts or hardware infrastructures, which can be considered another advantage of our approach.

Indeed, most of the proposed process is designed also to be very easy and cheap to implement in a pre-existing administrative structure: The physical identification phase is easy to achieve in a government context, where offices and staff are already at hand. The app distribution, which in our approach is achieved through direct download within an ad-hoc WiFi network, would only require some amount of WiFi configuration in the administrative offices, where WiFi connection is usually already present.

The only element in our process that may require attention is the app installation: As already discussed, for security reasons we build an ad-hoc app for every citizen, thus such an app cannot be installed from the mobile stores. On the Android platform, this implies temporarily disabling the "install only from known sources" security flag which, however, is a common practice since many other well-known third-party app stores already exist (e.g., Amazon). On the other hand, Apple allows to apply for special development licenses which provide the option to deploy apps on the iOS without the app store.

Author's Contributions

Giuseppe Della Penna: Contributed to the writing of the manuscript and the development of the research.

Pietro Frasca: Participated in the experimentation and validation, and developed the prototypes.

Benedetto Intrigila: Developed the original research idea and contributed to the writing of the manuscript.

Ethics

This work is original and has not been published elsewhere. The authors confirm that there are no ethical issues involved.

References

APLD, 2018. Sistema pubblico di identita digitale. Agenzia per l'Italia digitale.

Apple Inc, 2018. Apple pay. Apple Inc.

Aruba it, 2018. Otp da smartphone. Arubait. Baeuo, M., A. Rahim, N. and A. Alaraibi, 2017. Technology factors influencing e-government readiness. J. Theoretical Applied Informat. Technol., 95: 1637-1645.

Bailey, K.O., J.S. Okolica and G.L. Peterson, 2014. User identification and authentication using multi-modal behavioral biometrics. Comput. Security, 43: 77-89. DOI: 10.1016/j.cose.2014.03.005

Barkan, E., E. Biham and N. Keller, 2008. Instant ciphertext-only cryptanalysis of GSM encrypted communication. J. Cryptol., 21: 392-429. DOI: 10.1007/s00145-007-9001-y

Bettacchi, A., B. Re and A. Polzonetti, 2017. E-government and cloud: Security implementation for services. Proceedings of the 4th International Conference on eDemocracy and E-government, Apr. 19-21, IEEE Xplore Press, Quito, Ecuador, pp: 79-85. DOI: 10.1109/ICEDEG.2017.7962516

Boyer-Wright, K.M. and J.E. Kottemann, 2015. E-government and related indices: Telecommunications infrastructure, human capital, institutional efficacy and online services. Int. J. Electro. Govt. Res., 11: 24-37. DOI: 10.4018/IJEGR.2015100102

Burch, J.R., E.M. Clarke, K.L. McMillan, D.L. Dill and L.J. Hwang, 1992. Symbolic model checking: 10^{20} states and beyond. Inform. Comput., 98: 142-170. DOI: 10.1016/0890-5401(92)90017-A

Choudhary, N. and A. Jain, 2018. Comparative analysis of mobile phishing detection and prevention approaches. Proceedings of the International Conference on Information and Communication Technology for Intelligent Systems, (TIS' 18), Springer, Cham, pp: 349-356. DOI: 10.1007/978-3-319-63673-3_43

Dasgupta, D., A. Roy and A. Nag, 2016. Toward the design of adaptive selection strategies for multi-factor authentication. Comput. Security, 63: 85-116.

Della Penna, G., B. Intrigila, I. Melatti, E. Tronci and M. Venturini Zilli, 2004. Exploiting transition locality inautomatic verification of finite-state concurrent systems. Int. J. Software Tools Technol. Transfer, 6: 320-341.

Dill, D.L., A.J. Drexler, A.J. Hu and C.H. Yang, 1992. Protocol verification as a hardware design aid. Proceedings of the IEEE International Conference on Computer Design on VLSI in Computer and Processors, Oct. 11-14, IEEE Xplore Press, Cambridge, MA, USA, pp: 522-525. DOI: 10.1109/ICCD.1992.276232

Distel, B. and J. Becker, 2017. All citizens are the same, aren't they? – Developing an e-government user typology. Proceedings of the 16th IFIP WG 8.5 International Conference on Electronic Government, (CEG' 17), Springer, Cham, pp: 336-347.

Drape, S., C. Thomborson and A. Majumdar, 2007. Specifying imperative data obfuscations. Proceedings of the 10th International Conference on Information Security, Oct. 09-12, Springer, Valparaíso, Chile, pp: 299-314. DOI: 10.1007/978-3-540-75496-1_20

Ebbers, W.E., M.G. Jansen and A.J. van Deursen, 2016. Impact of the digital divide on e-government: Expanding from channel choice to channel usage. Govt. Inform. Quarterly, 33: 685-692. DOI: 10.1016/j.giq.2016.08.007

Google Inc, 2018. Google pay. Google Inc.

GuardSquare nv, 2017. Proguard. GuardSquare nv. Hassan, M. and L. Pantaleon, 2017. An investigation into the impact of rooting android device on user data integrity. Proceedings of the 7th International Conference on Emerging Security Technologies, Sept. 6-8, IEEE Xplore Press, Canterbury, UK, pp: 32–37. DOI: 10.1109/EST.2017.8090395

Holzmann, G.J., 1991. Design and Validation of Computer Protocols. 1st Edn., Prentice Hall, Englewood Cliffs. ISBN-10: 0135399254, pp: 500.

ICAO, 2015. Machine readable travel documents (doc 9303). http://www.icao.int/publications/Documents /9303 p1 cons en.pdf.

InfoCert, 2018. Infocert id. https://identitadigitale.infocert.it/.

IOS, 2003. Identification cards-physical characteristics (iso/iec 7810:2003). International Organization for Standardization.

IETF, 1998. A one-time password system (rfc2289). Internet Engineering Task Force.

IETF, 2011. Totp: Time-based one-time password algorithm (rfc6238). Internet Engineering Task Force. https://tools.ietf.org/html/rfc6238.

Kumar, S., S. Paul and D.K. Shaw, 2017. Real-time multimodal biometric user authentication for web application access in wireless LAN. J. Comput. Sci., 13: 680-693. DOI: 10.3844/jcssp.2017.680.693

Meyer, D., 2016. NIST prepares to phase out SMS-based login security codes. http://fortune.com/2016/07/26/nistsms-two-factor/

NIST, 2017. NIST special publication 800-63b. Digital Identity Guidelines, National Institute of Standards and Technology.

Olabode, O., 2011. Smart card identification management over a distributed database model. J. Comput. Sci., 7: 1770-1777.

Orgeron, C.P. and D. Goodman, 2011. Evaluating citizen adoption and satisfaction of e-government. Int. J. Electr. Govt. Res., 7: 57-78.

Papadomichelaki, X., V. Koutsouris, D. Konstantinidis and G. Mentzas, 2015. An analytic hierarchy process for the evaluation of e-government service quality. Int. J. Electr. Govt. Res., 9: 19-44. DOI: 10.4018/jegr.2013010102

Santoso, H.A., J. Zeniarja, A. Luthfiarta and B.J. Wijaya, 2016. An ontological crawling approach for improving information aggregation over e-government websites. J. Comput. Sci., 12: 455-463. DOI: 10.3844/jcssp.2016.455.463

Siadati, H., T. Nguyen, P. Gupta, M. Jakobsson and N. Memon, 2017. Mind your SMSes: Mitigating social engineering in second factor authentication. Comput. Security, 65: 14-28. DOI: 10.1016/j.cose.2016.09.009

Siadati, H., T. Nguyen and N. Memon, 2016. Verification code forwarding attack. Proceedings of the 9th International Conference on Technology and Practice of Passwords, Dec. 7-9, Cambridge, UK, pp: 65-71. DOI: 10.1007/978-3-319-29938-9_5

SU, 2004. Murphi web page. Stanford University.

Stanislav, M., 2015. Two-factor authentication. IT Governance Publishing.

URLS, 2017. Model checking laboratory. University of Rome "La Sapienza".

U.S. General Services Administration, 2018. login.gov. https://www.login.gov/

Velásquez, I., A. Caro and A. Rodríguez, 2018. Kontun: A framework for recommendation of authentication schemes and methods. Inform. Technol., 96: 27-37. DOI: 10.1016/j.infsof.2017.11.004

Yulistiawan, B.S., H. Prabowo, D. Budhiastuti and F.L. Gaol, 2014. Maturity model to measure the government institutions of Indonesia (the environment bureaucracy of education) in the implementation of e-government. J. Comput. Sci., 10: 2653-2657-16. DOI: 10.3844/jcssp.2014