

Auth4App: Protocols for Identification and Authentication using Mobile Applications

Diego Kreutz¹, Rafael Fernandes¹, Giulliano Paz¹, Tadeu Jenuario¹
Rodrigo Mansilha¹, Roger Immich², Charles C. Miers³

¹Advanced Studies Laboratory (LEA)
Postgraduate Program in Software Engineering (PPGES)
Federal University of Pampa (UNIPAMPA) – Brazil
diegokreutz@unipampa.edu.br, faelsfernandes@gmail.com
giulliano94@gmail.com, tadeujenuario@gmail.com
rodrigomansilha@unipampa.edu.br

²Metropolis Digital Institute (IMD)
Federal University of Rio Grande do Norte (UFRN) – Brazil
roger@imd.ufrn.br

³Graduate Program in Applied Computing (PPGCA)
Santa Catarina State University (UDESC) – Brazil
charles.miers@udesc.br

Abstract. *The increasing adoption of mobile applications as a means of user authentication is revealing new security challenges and opportunities. In order to modernize their physical identification and authorization procedures (e.g., access turnstile), some institutions have adopted static QR Codes generated using simple and static user data, such as some type of individual citizen national identification number. This procedure is easy to implement and verify, but it represents a critical security vulnerability. To address this issue, we propose Auth4App, a set of protocols for identification and authentication using mobile applications. Auth4App has two main protocols, one for binding user credentials to the mobile device (i.e., identification) and another one for generating one-time authentication codes (OTACs). Both protocols were formally verified using Scyther, an automated verification tool. Based on the automated analysis, our results show Auth4App protocols are robust enough and meet safe relevant criteria. Our prototype simulates access control using electronic turnstiles and was developed to present how our solution works and its deployment feasibility. The results show Auth4App enables accurate user authentication with a low computational cost.*

1. Introduction

Mobile devices are part of everyday life for most people. The arise of the Internet of Things (IoT), services available on the cloud, and the ubiquity of these devices will increase considerably in the next few years [Curado et al., 2019, Bittencourt et al., 2018]. According to recent statistics [S. O’Dea, 2020], there are more than 4.78 billion mobile devices connected to the Internet. Nearly 73% (3.5 billion) of these devices are smartphones, which means approximately 45% of the world’s population might already own one.

Smartphones are increasingly being used for authentication (i.e., identification and verification), and access control (permission or denial of entry) to a particular space (physical or logical/virtual). For instance, smartphones are being used to identify and allow users to access different types of facilities, such as gyms, libraries, swimming pools, sports courts, and water parks. This identification can be done in different ways, ranging from contactless codes (e.g., authentication code sent through Bluetooth/NFC) to virtual cards/credentials having a QR Code used for authentication purposes. However, existing virtual card-based solutions use an authentication code consisting of a static QR Code. This code is obtained only from this person's registration number or some other identifier, such as some type of individual citizen national unique identification number, e.g., Social Security Number (SSN) in the USA, and Individual Taxpayer Registration (CPF) in Brazil. Authentication mechanisms like this are known as static single-factor authentication. They are relatively simple to be circumvented since it is enough for the malicious agent to gain access to the user's credentials to compromise the authentication mechanism. For example, in systems using a static single-factor, the malicious agent only needs to know this person's credentials or clone the QR Code. Moreover, a QR Code can be easily read/ copied/ cloned from distance due to its nature. According to recent research [Belani, 2020], user credentials are still one of the main targets of hackers, and one of the root causes of data leakage. Furthermore, leaking user credentials is the main cause of improper access to private data [Verma et al., 2019, InfoArmor, 2017].

In order to mitigate security vulnerabilities, multi-factor authentication protocols have been proposed [Di Pietro et al., 2005, Maliki and Seigneur, 2007, Starnberger et al., 2009, Lee et al., 2010, Kaur et al., 2016, Ferrag et al., 2018, Wu et al., 2019]. Two-factor authentication can be performed using a username/password as the first factor and another authentication code, generated by a specific application (e.g., Google Authenticator) or sent via Short Message Service (SMS), as a second factor. However, the SMS protocol does not provide end-to-end security [Androulidakis, 2016] and it can lead to usability issues and challenges. The same is true for other multi-factor authentication mechanisms [Cristofaro et al., 2013]. For example, the additional factor would increase the delay for users passing through a turnstile to access a certain facility (e.g., gym, and library). This overhead is undesirable and, perhaps, a reason for the multi-factor mechanism that has not been broadly deployed [Ferrag et al., 2018]. Therefore, we notice the following research question: *How to solve the authentication problem, using a single factor, without compromising security and usability?*

In order to address the issue of the previous question, we present Auth4App, a set of protocols for identification and authentication using mobile applications. The utmost purpose of Auth4App is to provide secure authentication with a single dynamic factor, avoiding to compromise the usability. Our solution is composed of two protocols: the first one for linking user credentials (i.e., identification) to the mobile device, and the second one for generating disposable one-time authentication codes, a.k.a., OTAC. The core idea here is to limit the linking of the user's identity to just a single smartphone, mitigating the possibility of using the same credentials by multiple people through several distinct devices. The binding protocol generates a master key, which is used by the second protocol to derive unique authentication codes.

The main contributions of this work are as follow:

1. the design and implementation of a protocol for linking users to unique devices;
2. the conceptual design and development of a protocol for generating unique authentication codes;
3. the discussion of a use case to demonstrate the practical application of the solution;
4. implementation of the Auth4App prototype that simulates the use of the solution in electronic turnstiles through QR Code; and
5. formal verification of the protocols using the Scyther tool.

The remainder of this paper is organized as follows. In §3 and §2, we discuss the requirements, assumptions and protocols. We introduce our use case and its main challenges in §4. Following, we provide the formal verification of the protocols (§5) and the related works (§6). Finally, we provide our final remarks in §7.

2. Proposal requirements and assumptions

We assume the user device as unreliable, which means an attacker can compromise the device and may have access to identification and verification data. In order to guide the design of the Auth4App solution, we defined the following requirements:

- **Unique bond.** A user must be linked to a single device at a given time t . The single link reduces the risks associated with any attempt to use leaked user credentials.
- **General-purpose solution.** The proposed solution must be of general-purpose, i.e., it must apply to different use cases and scenarios.
- **Out-of-band channels.** To guarantee a single and secure link between a user U and a device D , out-of-band channels must be used to send additional security codes.
- **Revocation of compromised identities.** In case of theft or loss of the user's smartphone, mechanisms to identify and revoke the link of the device's application must be put in place.

3. The design of Auth4App

Auth4App consists of two protocols: (i) identification, in which the user's application is linked to a single mobile device, and (ii) verification, which deals with the disposable authentication codes and its generation. Fig. 1 shows a sequence diagram of the authorization process verification using Auth4App. There are three actors: the user, the turnstile (as an example of an access control mechanism), and the authentication service. First, the user requests a registration within the corporate authentication service. Sequentially, the user can request authentication to the turnstile. This authentication can be off-line, if the OTAC mechanism is implemented inside the turnstile, or online, using the authentication service. Both registration and authentication protocols are discussed in detail in the following sections.

3.1. The Identification Protocol

After installing the mobile application, in the first use, the user is asked for his/her credentials for identification and authentication (e.g., login/password). Then, during the first access, the registration and linking protocol of the application is started.

In Auth4App, a single device can be linked to the user's knowledge factor (e.g., login/password). The identification protocol ensures that, in the event of a possible device cloning, the action is automatically detected.

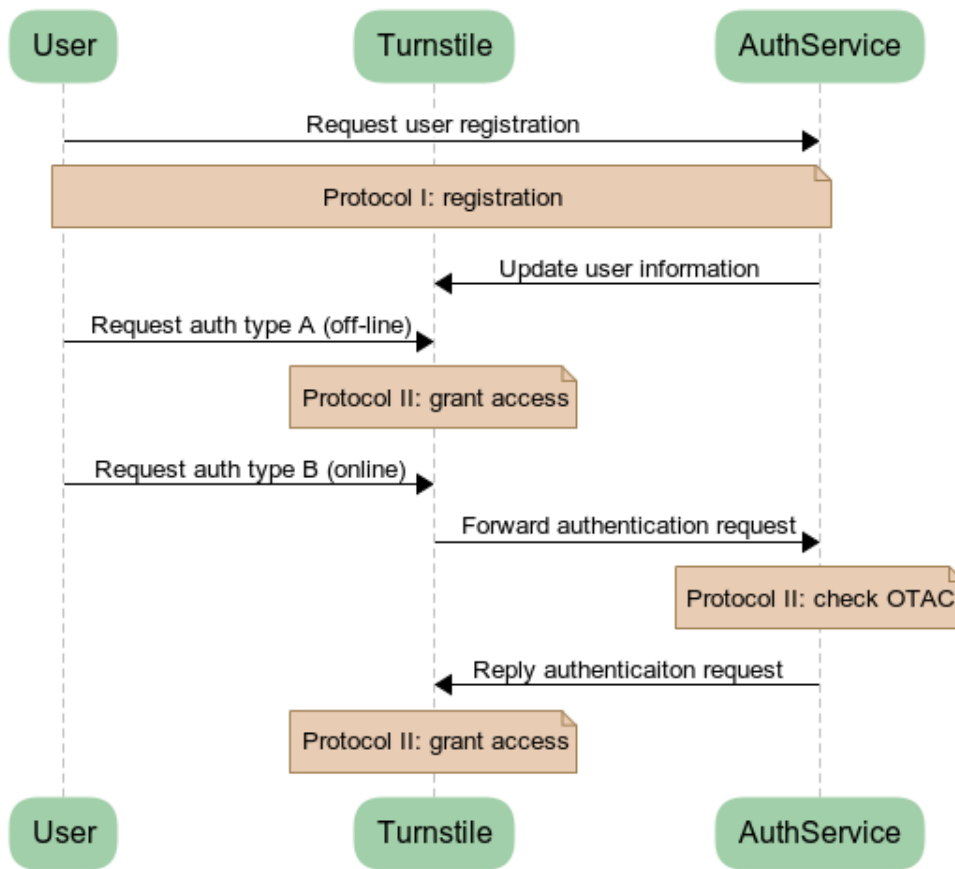


Figure 1. Auth4App: Sequence Diagram.

Protocol 1 describes the registration process details. It starts with a connection using Transport Layer Security (TLS) between the application and the server (line 1). Next, the server sends three different codes, using the TLS channel (line 2) and two out-of-band channels to the client; in this specific example an SMS and an e-mail, lines 3 and 4, respectively. We assume the attacker does not have the necessary resources to compromise all communication channels (e.g., out-of-band channels) simultaneously, thus increasing authentication reliability.

The first temporary key $KT1$ is generated from the TLS session key along with the three codes sent earlier by the server. A strong cryptographic hash function is used to generate the key (line 5), as proposed and proven to be a robust alternative for generating high entropy authentication codes [Kreutz et al., 2018] and secret keys [Kreutz et al., 2019]. The $KT1$ key is then used to encrypt (represented by E) the International Mobile Equipment Identity (IMEI) and the pseudo-random number mk_rnd . The same key is used to sign the message (i.e., Compute the Hash-based Message Authentication Code (HMAC)) which is transmitted from the client to the server. The values of IMEI, mk_rnd , and the key $KT1$ are used to generate the second temporary key $KT2$. This second key is assumed to be stronger since it includes a unique global number (the IMEI of the device) and a pseudo-random number of the application (the mk_rnd). This, of course, increases the entropy of the $KT2$ key.

Finally, the server sends a pseudo-random number srv_rnd to the client (line 8),

Protocol 1. Device linking and master key generation.

1. Client — Server	Secure connection to the Server
2. Server → Client	[CODE_TLS, $code_1$]
3. Server → Client	[CODE_SMS, $code_2$]
4. Server → Client	[CODE_EMAIL, $code_3$]
5. Client, Server	$KT1 \leftarrow H(K code_1 code_2 code_3)$
6. Client → Server	[Client, <i>nonce</i> , $E_{KT1}(IMEI, app_rnd)$], $HMAC_{KT1}$
7. Client, Server	$KT2 \leftarrow H(IMEI app_rnd KT1)$
8. Server → Client	[Server, <i>nonce</i> , $E_{KT2}(srv_rnd)$], $HMAC_{KT2}$
9. Client, Server	$KM \leftarrow H(KT1 KT2 IMEI app_rnd srv_rnd)$
10. Client → Server	[Client, V_M, <i>nonce</i> , $E_{KM}(mk_rnd)$], $HMAC_{KM}$
11. Server → Client	[Server, V_M, <i>nonce</i> , $E_{KM}(mk_rnd + 1)$], $HMAC_{KM}$

and both generate the high entropy master key KM (line 9). However, the master key still needs to be verified to finalize the protocol. Thus, the user sends an encrypted pseudo-random number mk_rnd to the server (line 10). The server decrypts the received pseudo-random number using its KM key, then increments it by one (+1), encrypts the new value, and sends it back to the client (line 11). If the client is able to validate the received value, it means that the keys are the same and the protocol execution was successfully completed.

As long as the IMEI and the master key are valid or in use by the user, it is not possible to register another device using the same credentials. The user can use only one device at a time. In order to use a new device, the user must first revoke the current application/registration of the former device. This security procedure is adopted by different fintechs and digital banks, such as Revolut (<https://www.revolut.com>), N26 (<https://n26.com>), and NuBank (<https://nubank.com.br/>).

3.2. The Authentication Protocol

The proposed authentication protocol was designed for generating unique codes that can be used for authentication. As aforementioned, the identification protocol generates a high entropy master key KM , and because of that, the key to the unique code generator can be derived from it. A secure cryptographic hash function can be used for derivation, such as those of groups SHA2 and SHA3 (<https://csrc.nist.gov/projects/hash-functions>). The initial key for generating unique authentication codes can be as simple as $K_c = H(K_m || K_c)$. Since the key K_c starts empty, the first K_c is equal to $H(KM)$.

The proposed solution uses a OTAC, which means the generated code is valid to authenticate a user's identity for only one transaction or login session. In Auth4App, these codes are generated from the key K_c . In order to synchronize the generation of these codes, it is necessary to use the indexes iA , in the application, and iS on the server. Then, at the beginning of the generation, the OTAC is equal to K_c . As soon as it is generated, the K_c is evolved to the next value, which is $K_c = H(K_m || K_c)$. Following, OTACs are

generated as follows: $OTAC = H^N(OTAC)$, in which the cryptographic hash function can be applied N times to generate N distinct and unique authentication codes. These codes have the security property called Perfect Forward Secrecy (PFC), as it is not possible to discover old OTAC codes from the current ones. This is ensured through the irreversibility property of the cryptographic hash functions.

Let us assume, for example, the iA and iS indexes are temporarily set to 1 and 0, respectively. The application just needs to send one message, with the current index of the local OTAC, to perform the authentication on the server. When the application sends the message “[GET, file_name, nonce, iA], HMAC” to the server, the HMAC (message signature) is generated using the application’s OTAC as a key. The server will update its OTAC to $OTAC = H^{iA-iS}(OTAC)$, using the index value received from the application. Using the new OTAC value, the server will verify the HMAC signature of the message. Finally, the server will confirm the authentication if the signatures match, and deny otherwise.

In our prototype implementation (using Python 3.7.3), one OTAC can be generated and verified in just 0.07ms (average of 10k runs), on a laptop with the following configuration: CPU Intel CPU i7-4510U 2.0 GHz, Debian (10 Buster) Linux. In other words, as expected, OTAC have (arguably) a low computational cost.

4. Use case and Challenges

The adoption of electronic turnstiles for access control is becoming increasingly common in a broad range of facilities (e.g., gyms, universities). However, these establishments tend to adopt the static data model (e.g., SSN or a pseudo-random token in a static QR Code) for access control, thereby presenting a security threat in the case of a user’s credentials leak.

In our solution, the OTAC can be used for access control even in simple turnstiles that are capable of reading QR Codes, as shown in Protocol 2. Each individual in the system has a digital identification card (line 1). Thus, the user opens the application, which automatically generates a one-time use QR Code for authentication (line 2). Next, the user brings the QR Code closer to the reader (line 3) and the turnstile reads the code (line 4). Afterward, the turnstile updates the OTAC (line 5) and checks it against the HMAC (line 6), providing or denying user access.

Protocol 2. Electronic turnstile OTAC.

1. User	Opens the identification application
2.	QR Code = [id, iA], $HMAC_{OTAC}$
3.	Brings the QR Code closer to the Turnstile
4. Turnstile	Reads the QR Code
5.	Updates the OTAC $\leftarrow H^{iA-iS}(OTAC)$
6.	Checks HMAC using the OTAC as key

We argue the process is efficient and simple because it can be carried out entirely offline. There is no network communication dependence, neither time and clock synchrono-

nization between devices to generate, read, and validate the QR Codes. The turnstile only needs a network connection to update the user's database, which can be scheduled. However, in such a case, we have to assume the turnstile as trustworthy as it might be able to impersonate a user.

The alternative to the off-line process is to put the authentication online, removing the OTAC generator from the turnstile. In this case, the authentication can be done by a Trusted Third Party (TTP), or outsourced to a specialized third party, which is a common approach taken in different domains and scenarios [Aloqaily et al., 2017, Zhan et al., 2018, Kreutz et al., 2019]. It is worth also emphasizing previous works that have shown the technical feasibility of resilient security services [Kreutz et al., 2014, Kreutz et al., 2016], which shows us a TTP might indeed be an interesting choice.

It is worth noticing that high entropy unique codes (e.g., OTACs) can be considered more reliable than biometric data, which are static in nature [Rui and Yan, 2019]. In other words, biometric data is comparable, to some extent, to an unchangeable password. If an immutable password leaks, all security on all systems based solely upon it will be compromised. Additionally, biometric readers should also be improved to provide anti-spoofing reading (e.g., silicone and Play-Doh fingerprints clones). For this reason, biometric authentication should be adopted cautiously and preferably together with other authentication formats.

5. Automatic Protocol Verification

The automatic verification of the security properties of a protocol is crucial to demonstrate its efficiency and correctness. Tools such as Scyther [Cremers, 2006] can contribute to this process [Amin et al., 2020, Almuzaini and Ahmad, 2019].

5.1. Identification Protocol Security Analysis

Algorithm 1 describes the identification protocol (Protocol 1) in the semantics of the Scyther tool. The predefined types *Code*, *TemporaryKey* and *MasterKey* are declared on line 1. The keys *K*, *KT1*, *KT2*, and *KM* are declared globally on lines 4, 5, and 6. Lines 2 and 3 define a cryptographic hash function *H* and an HMAC function, respectively.

The call to the `protocol` function is the beginning of the specification of the `Auth4App` protocol (line 7). It contains four agents that have explicit roles, namely `KGC` (line 8), `MKG` (line 11), `Client` (line 22), and `Server` (line 36).

As can be seen in the algorithm semantics, each sending event (e.g., `send_1`, line 9) has a corresponding receiving event (e.g., `recv_1`, line 14). The syntax of the `send_1` event indicates that the transmission is from Key Generation Center (KGC) to Master Key Generation (MKG). This simulates the generation of the session key *K*. The first temporary key *KT1* is generated through the cryptographic hash function *H*, which takes the session key *K* and the codes *code1*, *code2* and *code3* as a parameter. The key *KT1* of the `Client` and `Server` agents are generated on lines 15 and 16, respectively, ensuring that they are the same on both agents.

Furthermore, the `claim` function is used with two specific security requirements (`Secret` and `Nisynch`) to ascertain a term is secret and authentic, as can be noted from

Algorithm 1: Identification Protocol Security Analysis.

```
1 usertype Code, TemporaryKey, MasterKey;           28 recv_5(MKG, Client, KT2(H(imei, appRand1,
2 hashfunction H;                                     KT1)));
3 const HMAC: Function;                               29 recv_7(Server, Client, HMAC(nonce,
4 secret K: SessionKey;                                {serverRand}KT2(Server, Client));
5 secret KT1, KT2: TemporaryKey;                     30 recv_8(MKG, Client, KM(H(KT1, KT2, imei,
6 secret KM: MasterKey;                                appRand1, serverRand)));
7 protocol Auth4App(KGC, MKG, Client, Server){        31 claim(Client, Secret, KT1);
8   role KGC{// Key Generation Center                 32 claim(Client, Secret, KT2);
9     send_1(KGC, MKG, H(K));                          33 claim(Client, Secret, KM);
10  }                                                  34 claim(Client, Nisynch);
11 role MKG{// Master Key Generation                  35  }
12   fresh code1, code2, code3: Code;                  36 role Server{
13   fresh appRand1, serverRand, imei: Nonce;          37   var nonce:Nonce;
14   recv_1(KGC, MKG, H(K));                            38   var imei, appRand1:Nonce;
15   send_2(MKG, Client, KT1(H(K,code1, code2,         39   var code1, code2, code3:Code;
16     code3)));                                       40   fresh serverRand: Nonce;
17   send_3(MKG, Server, KT1(H(K,code1, code2,         41   recv_3(MKG, Server, KT1(H(K,code1, code2,
18     code3)));                                       code3)));
19   send_5(MKG, Client, KT2(H(imei, appRand1, KT1))); 42   recv_4(Client, Server, HMAC(nonce, {imei,
20   send_6(MKG, Server, KT2(H(imei, appRand1, KT1))); appRand1}KT1(Client, Server)));
21   send_8(MKG, Client, KM(H(KT1, KT2, imei,         43   recv_6(MKG, Server, KT2(H(imei, appRand1,
22     appRand1, serverRand)));                          KT1)));
23   send_9(MKG, Server, KM(H(KT1, KT2, imei,         44   send_7(Server, Client, HMAC(nonce,
24     appRand1, serverRand)));                          {serverRand}KT2(Server, Client));
25  }                                                  45   recv_9(MKG, Server, KM(H(KT1, KT2, imei,
26   role Client{                                       appRand1, serverRand)));
27   fresh nonce: Nonce;                               46   claim(Server, Secret, KT1);
28   fresh imei, appRand1, serverRand: Nonce;          47   claim(Server, Secret, KT2);
29   var code1, code2, code3: Code;                    48   claim(Server, Secret, KM);
30   recv_2(MKG, Client, KT1(H(K, code1, code2, code3))); 49   claim(Server, Nisynch);
31   send_4(Client, Server, HMAC(nonce, {imei,       50  }
32     appRand1}KT1(Client, Server)));                    51  }
```

lines 31 to 34, and 46 to 49. If a term, for example, the generated keys $KT1$, $KT2$, or KM , is claimed to be secret, they should be unknown to any adversary. In this case, the statements created can verify that the keys remain secret and authentic during communications. At the same time, the `Nisynch` claims all messages of the client (line 34) or the server (line 49) are indeed sent and received by their righteousness communication partner.

Scyther tool generates a full report containing the final status of all tests/attacks performed. When there are failures, Scyther also presents a flowchart illustrating details of how the attack can be carried out. In the case of the identification protocol, the communication between `Client` and `Server` is secure, according to the automatic analysis of Scyther (Fig. 2). `Auth4App` was not susceptible to any of the attacks implemented

in Scyther. It is worth emphasizing that we used the `--all-attacks` option of the Scyther tool to automatically verify our algorithms taking into account our own claims and special roles, such as `KGC` and `MGK`.

Claim				Status	Comments	
Auth4App	Client	Auth4App,Client1	Secret KT1	Ok	Verified	No attacks.
		Auth4App,Client2	Secret KT2	Ok	Verified	No attacks.
		Auth4App,Client3	Secret KM	Ok	Verified	No attacks.
		Auth4App,Client4	Nisynch	Ok	Verified	No attacks.
Server		Auth4App,Server1	Secret KT1	Ok	Verified	No attacks.
		Auth4App,Server2	Secret KT2	Ok	Verified	No attacks.
		Auth4App,Server3	Secret KM	Ok	Verified	No attacks.
		Auth4App,Server4	Nisynch	Ok	Verified	No attacks.

Done.

Figure 2. Report: Identification Protocol Security Analysis.

5.2. Authentication Protocol Security Analysis

Algorithm 2 describes the Protocol 2 (authentication) in Scyther’s semantics. The variables used, the cryptographic hash function H , and the HMAC function was defined from lines 1 to 4. The authentication process starts with the user’s OTAC update, which will be used for authentication with the turnstile. The `KGC` agent generates and sends a new code to the user (line 7). Following this, the user sends his `id`, `iA`, and the message HMAC (line 14) to the turnstile.

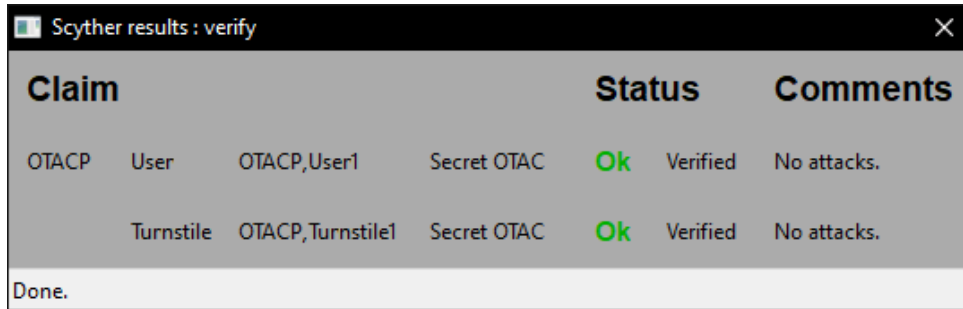
It is worth mentioning, due to the Scyther limitations, it is not possible to represent the difference in algorithm indexes (i.e., $iA - iS$) shown in Section 4. An abstraction was implemented to overcome this limitation, in which the turnstile receives and updates its OTAC code through the `KGC` agent (lines 19 and 20). Finally, the claim events depicted in lines 15 and 21 are executed to determine whether the OTACs of the two agents remain secure during Scyther’s automatic security analysis. Fig. 3 shows the result of Scyther’s analysis, the analysis returned `Ok` for the user and turnstile statements, indicating OTACs are secure.

6. Related work and Auth4App

Table 1 summarizes different identification and verification solutions as a second or third authentication factor that use mobile devices, tokens, smart-cards, among other resources.

Algorithm 2: Authentication Protocol Security Analysis.

```
1  usertype UniqueCode;
2  hashfunction H;
3  secret OTAC: UniqueCode;
4  const HMAC: Function;
5  protocol OTACG(KGC, User, Turnstile){
6    role KGC{
7      send_1(KGC, User, H(OTAC));
8      send_3(KGC, Turnstile, H(OTAC));
9    }
10   role User{
11     fresh id, iA: Nonce;
12     var nr: Nonce;
13     recv_1(KGC, User, H(OTAC));
14     send_2(User, Turnstile, id, iA, HMAC(id, iA));
15     claim(User, Secret, OTAC);
16   }
17   role Turnstile{
18     var iA, id: Nonce;
19     recv_2(User, Turnstile, id, iA, HMAC(id, iA));
20     recv_3(KGC, Turnstile, H(OTAC));
21     claim(Turnstile, Secret, OTAC);
22   }
23 }
```



The image shows a screenshot of a Scyther results window titled "Scyther results : verify". It contains a table with three columns: Claim, Status, and Comments. The table lists two claims, both of which are verified and have no attacks.

Claim	Status	Comments
OTACP User OTACP,User1 Secret OTAC	Ok Verified	No attacks.
Turnstile OTACP,Turnstile1 Secret OTAC	Ok Verified	No attacks.

Done.

Figure 3. Report: Authentication Protocol Security Analysis.

Most of these solutions are designed specifically for Internet Banking and banking systems (e.g., ATMs), prioritizing security over usability. Except for the proposed solution (Auth4App), all of them use multiple authentication factors, which go from simple to more complex procedures, and are domain-specific.

Auth4App is a general-purpose solution and can be used for identification and verification (i.e., authentication, authorization) in smartphone applications. Moreover, the verification is performed without the need for out-of-band channels. In other words, Auth4App is, to the best of our knowledge, the only solution that ensures a robust verification using only a single authentication factor. Assuming the access to the smartphone is sufficiently secure, the employment of the user's identity will be protected even in cases of loss or theft of the mobile device.

7. Conclusion

There is an increasing need for robust identification and authentication solutions nowadays. Auth4App provides a set of protocols for enabling identification and authentication using applications in smartphones, for instance. There are two core protocols, one for linking user credentials to the device and the other for generating unique codes. Those protocols can be used to increase the system's security, without impairing usability. This is possible because Auth4App provides a single, yet dynamic and robust factor of identification and authentication.

Table 1. Related work comparison.

Solutions	Design for	Authentication	Out-of-band channel
[Eldefrawy et al., 2011]	Internet Banking	User/password and OTP via mobile application	OTP is sent to mobile device
[Pratama and Prima, 2016]	Internet Banking	User/password and OTP via mobile application	QR Code on the electronic terminal or PC
[Khamis et al., 2017]	ATMs	Password, PIN, and mobile application	Bluetooth
[Putra et al., 2017]	Internet Banking	User/password, OTP, PIN, and smart card	NFC
Auth4App	ID apps	Single-factor authentication using OTACs	Codes sent via SMS and email

The key principle of Auth4App is the OTAC, i.e., the generation and verification of robust and disposable one-time authentication codes. One OTAC is used only once, for a single authentication, and then discarded. This makes it significantly more difficult to clone the generated authentication codes by malicious users. Furthermore, Auth4App protocols can be considered secure since the essential security properties have been formally verified using the Scyther tool.

Future work:

1. in-depth performance evaluation of the protocols on different real use cases (e.g., smartphone-turnstile, smartphone-smartphone, smartphone-PC/web application);
2. evaluation of hardware-assisted solutions (e.g., trusted execution environments, a.k.a. TEEs [Asokan, 2019, Pinto and Santos, 2019, Coppolino et al., 2019]) for securely storing and using the master key on mobile phones;
3. threats modeling and security analysis of the protocols taking into account different use cases and stealthy attacks;
4. use symbolic analysis tools, such as the TAMARIN prover [Meier et al., 2013] (<https://tamarin-prover.github.io/>), to formally prove the protocols against powerful adversaries; and
5. use formal proof management systems, such as Coq [Brauer et al., 2004] (<https://coq.inria.fr/>), to provide formally-verified implementations (e.g., in C programming language) of the protocols.

References

- [Almuzaini and Ahmad, 2019] Almuzaini, N. Z. and Ahmad, I. (2019). Formal analysis of the signal protocol using the scyther tool. In *2019 2nd International Conference on Computer Applications Information Security (ICCAIS)*, pages 1–6.
- [Aloqaily et al., 2017] Aloqaily, M., Kantarci, B., and Mouftah, H. T. (2017). Trusted third party for service management in vehicular clouds. In *13th International Wireless Comm. and Mobile Computing Conference*, pages 928–933.

- [Amin et al., 2020] Amin, R., Lohani, P., Ekka, M., Chourasia, S., and Vollala, S. (2020). An enhanced anonymity resilience security protocol for vehicular ad-hoc network with scyther simulation. *Computers & Electrical Engineering*, 82:106554.
- [Androulidakis, 2016] Androulidakis, I. I. (2016). *SMS Security Issues*, pages 71–86. Springer International Publishing, Cham.
- [Asokan, 2019] Asokan, N. (2019). Hardware-assisted trusted execution environments: Look back, look ahead. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, page 1687, New York, NY, USA. Association for Computing Machinery.
- [Belani, 2020] Belani, G. (2020). 5 Cybersecurity Threats to Be Aware of in 2020 | IEEE Computer Society. Library Catalog: www.computer.org.
- [Bittencourt et al., 2018] Bittencourt, L., Immich, R., Sakellariou, R., Fonseca, N., Madeira, E., Curado, M., Villas, L., DaSilva, L., Lee, C., and Rana, O. (2018). The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things*, 3-4:134 – 155.
- [Brauer et al., 2004] Brauer, W., Salomaa, A., Rozenberg, G., and Paulin-Mohring, C. (2004). Coq'art: The calculus of inductive constructions.
- [Coppolino et al., 2019] Coppolino, L., D'Antonio, S., Mazzeo, G., and Romano, L. (2019). A comprehensive survey of hardware-assisted security: From the edge to the cloud. *Internet Things*, 6.
- [Cremers, 2006] Cremers, C. J. F. (2006). *Scyther: Semantics and verification of security protocols*. Eindhoven University of Technology Eindhoven.
- [Cristofaro et al., 2013] Cristofaro, E. D., Du, H., Freudiger, J., and Norcie, G. (2013). Two-factor or not two-factor? A comparative usability study of two-factor authentication. *CoRR*, abs/1309.5344.
- [Curado et al., 2019] Curado, M., Madeira, H., da Cunha, P. R., Cabral, B., Abreu, D. P., Barata, J., Roque, L., and Immich, R. (2019). *Internet of Things*, pages 381–401. Springer International Publishing.
- [Di Pietro et al., 2005] Di Pietro, R., Me, G., and Strangio, M. A. (2005). A two-factor mobile authentication scheme for secure financial transactions. In *International Conference on Mobile Business (ICMB'05)*, pages 28–34. IEEE.
- [Eldefrawy et al., 2011] Eldefrawy, M. H., Alghathbar, K., and Khan, M. K. (2011). OTP-Based Two-Factor Authentication Using Mobile Phones. In *8th Int. Conf. on Info. Tech.: New Generations*, pages 327–331.
- [Ferrag et al., 2018] Ferrag, M. A., Maglaras, L. A., Derhab, A., Vasilakos, A. V., Rallis, S., and Janicke, H. (2018). Authentication schemes for smart mobile devices: Threat models, countermeasures, and open research issues. *CoRR*, abs/1803.10281.
- [InfoArmor, 2017] InfoArmor (2017). Understanding the impact of compromised credentials.
- [Kaur et al., 2016] Kaur, N., Devgan, M., and Bhushan, S. (2016). Robust login authentication using time-based OTP through secure tunnel. In *3rd Int. Conf. on Comp. for Sustainable Global Development*, pages 3222–3226. IEEE.

- [Khamis et al., 2017] Khamis, M., Hasholzner, R., Bulling, A., and Alt, F. (2017). GT-moPass: Two-factor Authentication on Public Displays Using Gaze-touch Passwords and Personal Mobile Devices. In *6th ACM International Symposium on Pervasive Displays*, pages 8:1–8:9, New York, NY, USA. ACM.
- [Kreutz et al., 2014] Kreutz, D., Bessani, A., Feitosa, E., and Cunha, H. (2014). Towards secure and dependable authentication and authorization infrastructures. In *IEEE 20th Pacific Rim International Symposium on Dependable Computing*, pages 43–52.
- [Kreutz et al., 2016] Kreutz, D., Malichevskyy, O., Feitosa, E., Cunha, H., [da Rosa Righi], R., and [de Macedo], D. D. (2016). A cyber-resilient architecture for critical security services. *Journal of Network and Computer Applications*, 63:173 – 189.
- [Kreutz et al., 2018] Kreutz, D., Yu, J., Esteves-Veríssimo, P., Magalhães, C., and Ramos, F. M. V. (2018). The KISS principle in software-defined networking: A framework for secure communications. *IEEE Security & Privacy*, 16(5):60–70.
- [Kreutz et al., 2019] Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2019). ANCHOR: Logically centralized security for software-defined networks. *ACM Transactions on Privacy and Security*, 22(2):8:1–8:36.
- [Lee et al., 2010] Lee, Y. S., Kim, N. H., Lim, H., Jo, H., and Lee, H. J. (2010). Online banking authentication system using mobile-OTP with QR-code. In *5th Int. Conf. on Comp. Sciences and Convergence Information Technology*, pages 644–648.
- [Maliki and Seigneur, 2007] Maliki, T. E. and Seigneur, J. (2007). A Survey of User-centric Identity Management Technologies. In *The Int. Conf. on Emerging Security Information, Systems, and Technologies*, pages 12–17.
- [Meier et al., 2013] Meier, S., Schmidt, B., Cremers, C., and Basin, D. (2013). The tamarin prover for the symbolic analysis of security protocols. In Sharygina, N. and Veith, H., editors, *Computer Aided Verification*, pages 696–701, Berlin, Heidelberg. Springer Berlin Heidelberg.
- [Pinto and Santos, 2019] Pinto, S. and Santos, N. (2019). Demystifying arm trustzone: A comprehensive survey. *ACM Comput. Surv.*, 51(6).
- [Pratama and Prima, 2016] Pratama, A. and Prima, E. (2016). 2FMA-NetBank: A proposed two factor and mutual authentication scheme for efficient and secure internet banking. In *8th Int. Conf. on Info. Tech. and Electrical Eng.*, pages 1–4.
- [Putra et al., 2017] Putra, D. S. K., Sadikin, M. A., and Windarta, S. (2017). S-Mbank: Secure mobile banking authentication scheme using signcryption, pair based text authentication, and contactless smart card. In *15th Int. Conf. on Quality in Research (QiR)*, pages 230–234.
- [Rui and Yan, 2019] Rui, Z. and Yan, Z. (2019). A survey on biometric authentication: Toward secure and privacy-preserving identification. *IEEE Access*, 7:5994–6009.
- [S. O’Dea, 2020] S. O’Dea (2020). Number of smartphone users worldwide from 2016 to 2021.
- [Starnberger et al., 2009] Starnberger, G., Frohofer, L., and Goeschka, K. M. (2009). Qr-tan: Secure mobile transaction authentication. In *2009 International Conference on Availability, Reliability and Security*, pages 578–583.

- [Verma et al., 2019] Verma, R. S., Chandavarkar, B. R., and Nazareth, P. (2019). Mitigation of hard-coded credentials related attacks using QR code and secured web service for IoT. In *10th International Conference on Computing, Communication and Networking Technologies*, pages 1–5.
- [Wu et al., 2019] Wu, L., Wang, J., Choo, K. R., and He, D. (2019). Secure key agreement and key protection for mobile device user authentication. *IEEE Trans. on Information Forensics and Security*, 14(2):319–330.
- [Zhan et al., 2018] Zhan, J., Fan, X., Cai, L., Gao, Y., and Zhuang, J. (2018). TPTVer: A trusted third party based trusted verifier for multi-layered outsourced big data system in cloud environment. *China Communications*, 15(2):122–137.