User Authentication Protocol for Legacy Mobile Devices

Vasilios Mavroudis
vmavroudis@turing.ac.uk

Doc version 1.0
https://github.com/alan-turing-institute/grID

This document specifies an online authentication protocol for the legacy mobile devices. In several regions, a substantial part of the population uses feature phones due to them being inexpensive and widely available in the second-hand market. Such legacy devices come with several limitations in their interfacing and extensibility capabilities. This protocol relies on the WAP capabilities of the device to request and issue authentication tokens, which are then relayed to the verifying party using Quick Response (QR) codes.

Contents

1 Setup **3**

2 Use Cases **3**

3 Threat Model **4**

4 Protocol **4**

5 Implementation **6**

6 Technical Requirements **6**

6.1 Server hosting the WAP content 6

6.2 WAP Services supported by the mobile network provider 6

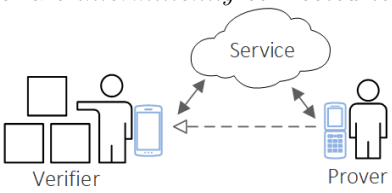
6.3 WAP browser available in the users' phones. 6

1 Setup

Our goal is to enable feature phone users to participate in modern authentication protocols without any additional investment in equipment.

In the rest of this document, we refer to the feature phone user as *Prover*. The Prover wishes to prove their identity or an attribute (e.g., their age) to the *Verifier*. The Verifier can be a government official, a local merchant, a bank representative or a local organisation running a food subsidy. Besides these actors, we also introduce an external *Service* that is available over the Internet and is tasked with keeping records of the users' details and can issue *Tokens*. Tokens encapsulate various types of data such as digital certificates, signatures, URIs and plain user information depending on the use case.

Figure 1: **Setup.** The Prover wishes to prove their identity or status to the Verifier. The Verifier uses a modern device, while the Prover possesses a feature phone, making it hard to interface with nearby devices (dotted arrow). Both the Prover and Verifier are *intermittently* connected to the Internet.



As seen in Figure 1, both the Prover and the Verifier need to be able to connect to the Internet using their devices. This connectivity may be intermittent for one or both parties. In line with the sparse mobile network coverage seen in many rural and less developed regions, their connectivity is assumed to intermittent. Moreover, the Prover is able to access online services only through the low-bandwidth Wireless Application Protocol (WAP) due to the limitations of their device. The Verifier's device needs to be capable of scanning QR codes. A basic smartphone or a considerably cheaper smart-feature phone could be used.

2 Use Cases

We now outline two use cases that require a cross-device interface. Both of them require the Prover and the Verifier to be in physical proximity.

1. Identity or Status Credentials. The Prover wants to convince the Verifier of their identity or an attribute/status (e.g., over 18). From their end, the Verifier needs to be presented with adequate proof that the claims of the Prover are truthful. We assume that the two parties do not necessarily have access to the Internet during the execution of the protocol. The implementation of the Covid-19 certificates in the EU [1] is a straightforward instantiation of this use case.

2. Food Subsidy Management. The Prover wants to convince the Verifier that

Figure 2: Service login screen and QR Code as displayed on a Nokia 3510 (2002) microbrowser.

they can receive the food subsidy. The Verifier needs to be presented with adequate proof that the Prover is eligible for the subsidy and that they have not already used their entitlement.

3 Threat Model

A Prover can be malicious and may attempt to claim attributes or request access to services they are not entitled to. We assume that provers can collude with other malicious users and are byzantine i.e., may deviate from the correct execution of the protocol. For example, such an adversary may attempt to craft malicious tokens, replay tokens from other users or modify their own past tokens. The Verifier is also potentially malicious and may, for instance, capture credentials submitted by users and attempt to replay them to other victim verifiers.

We also assume that the Service is trustworthy and that the provider does not act maliciously (either independently or by colluding with other malicious parties). All online communications are encrypted using standard encryption techniques (e.g., TLS 1.3 [2]). Denial of service attacks from the Verifier or the service towards provers are not within the scope of this work. Moreover, we do not consider cases where a malicious Prover compromises a Verifier's terminal or a Prover colludes with a Verifier, as the Verifier can always opt to not execute the protocol at all.

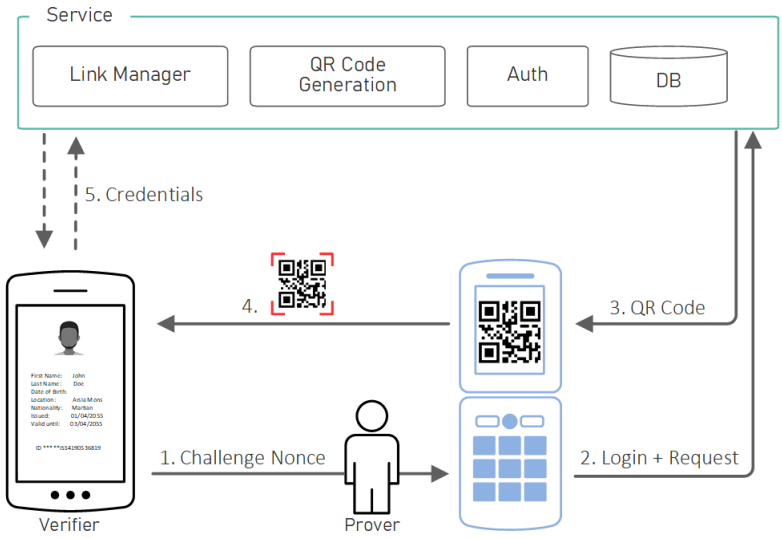
4 Protocol

In this section, we outline our system architecture and discuss our design choices.

As seen in Figure 3, the Prover is first given a cryptographic nonce from the Verifier (step 1). This nonce serves as a challenge to prevent replay attacks and is relatively short (6-10 digits) as the Prover manually types it into the feature phone. Following the nonce input and a successful user login (Figure 2b) to the online service, the Prover submits a request for a token (step 2). The service responds to the request and the feature phone's microbrowser displays the QR code (step 3). Subsequently, the Verifier scans and decodes the QR code (step 4). Following this step, the token verification may be carried out either online or

Demonstration videos can be found at: https://tiny.cc/ETAA2021_demos and a sample implementation here: https://github.com/alan-turing-institute/grID.

Figure 3: **Authentication protocol.** The feature phone user (the Prover) authenticates to the user of a smartphone (the Verifier) by fetching and displaying a (single-use) QR code on the screen of their phone. The QR code token may encapsulate a digital signature, a URI or arbitrary other data. The recipient decodes the token and verifies the validity of the contents (if applicable) either locally or through a trusted online service.



using offline credentials.

Offline Credentials. The Verifier checks the integrity and validity of the token locally on their device (e.g., by verifying its digital signature). In particular, they ensure that the data is signed under the online service's public key along with the correct nonce. If the verification is successful, the Verifier accepts the Prover's claim (use case #1 in Section 2).

Note that in cases such as the Covid-19 certificates [1], the tokens do not need to be "fresh" and thus the nonce step can be omitted. The advantage of the nonce-free approach is that the feature phone user can use the microbrowser's cache to request tokens ahead of the time and use them later even in areas with no network coverage. In contrast, if the tokens, need to be newly-issued, the Verifier can request for the random nonce to be incorporated in the generated token.

Online Verification. Alternatively, the Verifier may use the trusted online service to fetch the user data. In this case, the token encoded in the QR code is simply a (single-use) URI that allows the Verifier to load the user's data from the service. To prevent replay attacks, where a user presents the same token several times the service should allow each URI to be used only once (use case #2 in Section 2). Note that this protocol does not prevent the Verifier from launching relay attacks and impersonating the Prover to other Verifiers. More specifically, a malicious Verifier could use a delegate who will initiate a transaction with a

victim Verifier and relay their nonce. In our use case, this is not a concern as the Verifier will not be able to claim government subsidy twice (the online service prevents token reuse).

5 Implementation

An example implementation of an eID portal can be found here: https://github.com/alan-turing-institute/grID. Videos detailing the user experience can be found here: http://tiny.cc/ETAA2021_demos.

6 Technical Requirements

For this design to be deployed and usable there are three prerequisites: 1) A WAP server, 2) Network support for WAP services, and 3) A mobile device that supports WAP connectivity.

6.1 Server hosting the WAP content

This is a server similar to that one would use to host a website. The difference with a classic HTTP server is that the pages are formatted as WML (cf. to HTML). We have already setup such a server at the Turing and can confirm that this requirement is easy to satisfy.

6.2 WAP Services supported by the mobile network provider

For a mobile device to load a WAP website, the MNO needs to operate a WAP Gateway. In our preliminary experiments, we found that MNOs in Europe still maintain such Gateways. It is unclear if MNOs in our areas of interest still run such Gateways (see also the Survey section). Mobile network operators tend to downscale but not completely discontinue older services so as not to disrupt legacy users (e.g., a hydroelectric dam may still rely on WAP connectivity). This design can still operate with intermittent Internet connectivity but not without any.

6.3 WAP browser available in the users' phones.

For a user to be able to load WAP websites their phone needs to support it. We have found that various older devices (e.g., Nokia 3510 release in 2002) do support WAP but not all of them. It is thus important to know the capabilities of the devices in the communities we are planning to deploy for (see also Survey below).

References

[1] "Interim guidance for developing a Smart Vaccination Certificate," ISO/IEC, Tech. Rep., Mar 2021.

[2] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," 2018.