PASSPHONE: Outsourcing Phone-based Web Authentication while Protecting User Privacy*

Martin Potthast[1], Christian Forler[2], Eik List[1], and Stefan Lucks[1]

[1]Bauhaus-Universität Weimar, Germany,
[2]Beuth Hochschule für Technik Berlin, Germany
[1]<first name>.<last name>(at)uni-weimar.de
[2]cforler(at)beuth-hochschule.de

Abstract This work introduces PASSPHONE, a new smartphone-based authentication scheme that outsources user verification to a trusted third party without sacrificing privacy: neither can the trusted third party learn the relation between users and service providers, nor can service providers learn those of their users to others. When employed as a second factor in conjunction with, for instance, passwords as a first factor, our scheme maximizes the deployability of two-factor authentication for service providers while maintaining user privacy. We conduct a twofold formal analysis of our scheme, the first regarding its general security, and the second regarding anonymity and unlinkability of its users. Moreover, we provide an automatic analysis using AVISPA, a comparative evaluation to existing schemes under Bonneau et al.'s framework, and an evaluation of a prototypical implementation.

Keywords: Cryptographic protocol · two-factor authentication · unlinkability

1 Introduction

Two-factor authentication is an effective means to strengthen user authentication on the Internet. In particular, the use of software-based second-factor tokens is attractive for service providers since it relieves them from considerable costs that come along with developing and delivering custom hardware tokens. For their users, phone-based two-factor solutions have the advantage of employing the nowadays omnipresent smartphone, avoiding the inconvenience of carrying around yet another device for the sole purpose of authentication. However, offering two-factor authentication is not at all the default, yet.

Meanwhile, small and medium enterprises, and especially startups, outsource user verification. This is due to the fact that the proper implementation of a secure authentication solution is a non-trivial task, and that many struggle to get even basic password authentication right [12]. Hence, delegating user verification to a competent trusted third party appears reasonable. In the context of password authentication, corresponding infrastructures have been successfully established via OpenID [36] and OAuth [23]

* An earlier version of this work will be presented at NordSec2016 [35] and will appear in Springer Lecture Notes of Computer Science. This is the full version from February 18, 2017.

(e.g., Google, Yahoo, and Wordpress for OpenID and Twitter, Facebook, and PayPal for OAuth). On the upside, outsourcing user verification is convenient for users and reduces development costs for service providers, mitigating the risks of developing a custom solution from scratch. On the downside, however, outsourcing authentication has been justly criticized for its impact on privacy: the authentication provider serving as trusted third party gains precise information about a user's preferred services, her usage behavior, as well as the success of a given service. While undesirable for both service providers and their users, the former often choose user convenience and development speed over privacy, whereas most of the latter apparently do not care. Clearly, there is a lot of room for improving the outsourcing of authentication in terms of user privacy. The privacy of phone-based three-party authentication, however, has not been considered until now.

This paper proposes PASSPHONE, a smartphone-based two-factor authentication scheme which outsources user verification to a trusted third party while protecting user privacy. To the best of our knowledge, our scheme is the first smartphone-based one to incorporate anonymity and unlinkability despite employing a trusted third party. We conduct a systematic analysis of our scheme in terms of its security, privacy, feasibility, and competitiveness. In particular, we analyze its security and privacy properties formally, report on a practical implementation, and evaluate its competitiveness under the framework of Bonneau et al. [11]. We also conduct an automatic security analysis using the well-known computer-aided proof system AVISPA [4]. In what follows, after a brief review of related work, Section 3 introduces our authentication scheme. Section 4 formally analyzes its authentication security and privacy properties and Section C reports results from an automatic security analysis. Section 6 discusses insights gained from implementing our scheme, Section 7 compares it to a selection of existing phone-based solutions, and Section 8 discusses its practical application.

2 Related Work

Privacy in federated authentication. Dey and Weis [16] propose PseudoID, which can be considered the complement of our scheme for traditional password authentication. Their scheme also employs blinding to render users unlinkable across service providers. Dey and Weis show the unlinkability of their authentication scheme, but give neither an actual protocol nor an analysis. A proof of concept had been published, but the associated web page has disappeared. Otherwise, the privacy issues of federated authentication services have been highlighted in many contexts: for example, Urueña et al. [43] consider a privacy problem that concerns OpenID and Facebook Connect. They find that the unique identifier assigned to users by both services may leak to third parties, allowing to track users across web services since they encode user identifiers in the GET parameters of URLs. Riesch and Du [37] and Nuñez et al. [32] propose ways to solve the privacy issues of OpenID; Nuñez and Agudo [31] finally proposed a blinded version of OpenID called BlindIdM.

Phone-based two-factor authentication. Banks have been among the first to roll out two-factor authentication schemes for transactions, whereas online games and Google first deployed this technology at scale for web user authentication [21]. In light of recent security breaches [20,26,27], a shift toward two-factor authentication can be observed since several major companies such as Microsoft, Apple, and Facebook, some of which suffered attacks, rolled out their own implementations [3,30,39].

In the literature, Dodson et al. propose Snap2Pass [18,19] and van Rijswijk and van Dijk propose Tiqr [44]: both are phone-based schemes that use QR codes to transmit a challenge from a service provider via a user's browser to her phone, which responds to the challenge. Dodson et al. also consider outsourcing authentication to a trusted third party (an OpenID provider); though, they do not tackle the privacy issues associated with this approach. The authentication schemes by Aloul et al. [1] and Hallsteinsen et al. [22] are also phone-based challenge-response protocols based on one-time passwords (OTPs) that are generated using a previously shared secret between a user and a key server. This OTP is then transmitted to the device and used as a second means of authentication. In both two-factor authentication schemes, the key server can learn precisely which user tries to authenticate at which service. Karapanos et al.'s SoundProof [25] aims at increasing the adoption of two-factor authentication by avoiding the need for user interaction with their device. Instead, their authentication detects the physical proximity of the smartphone via matching the ambient sound of their environment. While the approach puts forth usability, it can protect neither against physical nor against man-in-the-middle or phishing attacks, and it is not easily deployable for service providers. Shirvanian et al. [38] categorize smartphone-based two-factor authentication schemes concerning the amount of data transmitted between client and phone. They concern four challenge/response formats: (1) a low-bandwidth variant which uses a PIN as second factor, (2) a mid-bandwidth variant with a QR-code challenge, a full-bandwidth variant which transmits challenge and response via Bluetooth, and another full-bandwidth variant which transmits challenge and response via WiFi. Their protocols are simpler and applicable on a wide range of devices; however, their low-bandwidth variants provide only 20 bits of additional security from a PIN or a low-resolution QR code, and the mid-bandwidth and the full-bandwidth versions require a complex setup with either a webcam, Bluetooth, or WiFi channel controlled by the client.

While the above schemes are those closely related to ours, a number of other schemes concern *transaction* authentication via untrusted devices, such as the ones of Clarke et al. [14], Wu et al. [46], Parno et al.'s PhoolProof [34], Starnberger et al.'s QR-TAN [40], Mannan and van Oorschot's MP-Auth [28,29], and Czeskis et al.'s PhoneAuth [15]. Altogether, however, we are unaware of any phone-based authentication scheme that improves deployability for service providers via outsourcing while incorporating user privacy.

3 The Passphone Authentication Scheme

This section introduces our authentication scheme. We overview the three parties involved, the devices at their disposal, and how they interact within protocols for bootstrapping and authentication. For completeness, we also introduce protocols for key management.

Parties and their devices. Our scheme involves the following parties:

- P A prover who wants to use a service provided by S.
- S A service provider, who wants to authenticate P.
- T A trusted third party of prover P and service provider S.

The prover is a human, while the service provider and the trusted third party host server-side services. The prover uses the following means to interact with these services:

– PS The prover's browser to access a service of S.
– PT The prover's phone to authenticate with T.
– PM The prover's mail box.

We assume that servers and the prover's devices have computational power at least comparable to that of current commercial off-the-shelf computer hardware and that they can communicate with each other via the Internet. The prover has all her devices under her full control (i.e., they are not compromised).

3.1 Bootstrapping

To get started, a prover P completes two bootstrapping steps: registration with the trusted third party T, and activation of our authentication scheme at her service provider S.

Registration protocol. For registration, P installs an authentication App PT on her phone (authenticator, for short). The App may be shipped by T and is ideally available open source. When P launches PT for the first time, PT generates a new key pair (K^p_{PT}, K^s_{PT}), asks for P's mail address ID_{PM}, and then initiates the registration protocol. Table 1 lists the protocol's communication steps; each step is denoted as:

$$(\text{<step>})\ \text{<sender>} \rightarrow \text{<receiver>} : \text{<message>},$$

where a message is optionally encrypted and consists of a header, a payload, and an optional signature:

$$\text{<message>} ::= E_K((\text{<header>}, \text{<payload>})_{\text{<signature>}}),$$

where E_K denotes an encryption scheme with key K. The <header> contains a domain identifier, step number, protocol version, and sender identifier:

$$\text{<header>} ::= [\text{<domain>},\text{<step>}, \text{<version>}, \text{<sender>}].$$

In Step (1) of the registration protocol, the authenticator chooses uniformly at random a nonce N_{PT} and derives the hash value $h_{PT} = H(N_{PT})$. Prior, it generates a key pair with a secret part K^s_{PT} and a public part K^p_{PT}; the public part, together with ID_{PM} and h_{PT}, is signed by PT and sent to the trusted third party T. Since the identifier ID_{PT} has not been verified by T, yet, we reserve the zero byte value as sender identifier. To verify the prover's mail box PM, the trusted third party sends a signed challenge containing a nonce N_T in Step (2). The prover forwards this message X to her authenticator in Step (3), which responds to the challenge by signing X and sending it back to T in Step (4). After successful verification, the trusted third party generates a new unique nonce N'_T, generates $ID_{PT} = H(N'_T, h_{PT})$, and sends N'_T to PT in Step (5), which henceforth uses ID_{PT} to identify itself. PT completes the bootstrapping protocol by sending an encrypted key- management ticket for rekeying to its mail account in Step (6). The prover keeps the ticket secret for later recovery of her account. Since T is not aware of N_{PT}, it cannot regenerate the tickets nor be compelled to do so, e.g., by law enforcement.

Activation protocol. To activate our scheme, the prover P creates an account at S using PS. S initiates the activation protocol shown in Table 2, the purpose of which is

Table 1. Protocol to register with the trusted third party.

Protocol 1:	**Registration of P at T**
Parties:	PT, PM, and T
Pre-conditions:	PT is blank, T is ignorant of PT
Post-conditions:	PT stores (K^p_{PT}, K^s_{PT}) and obtained ID_{PT}, P received tickets for rekeying and key transfer, T verified ID_{PM} and stores $(ID_{PT}, K^p_{PT}, ID_{PM})$
(1) $PT \rightarrow T$:	$\mathrm{TLS}(([\mathrm{REG}, 1, \mathtt{v}, 0],\ K^p_{PT},\ ID_{PM},\ h_{PT})_{PT})$
(2) $PM \leftarrow T$:	$\underbrace{([\mathrm{REG}, 2, \mathtt{v}, ID_T],\ N_T)_T}_{X}$
(3) $PM \rightarrow PT$:	X
(4) $PT \rightarrow T$:	$\mathrm{TLS}(([\mathrm{REG}, 3, \mathtt{v}, 0],\ X)_{PT})$
(5) $PT \leftarrow T$:	$\mathrm{TLS}(([\mathrm{REG}, 4, \mathtt{v}, ID_T],\ N'_T)_T)$
(6) $PT \rightarrow PM$:	$\mathrm{TLS}(([\mathrm{REKEY}, 1, \mathtt{v}, ID_{PT}], N_{PT}, N'_T, K^p_{PT})_{PT})$

to verify that P is capable of authenticating via T, and to learn the blinded identifier h_{PT} of PT.

In Step (1) of the activation protocol, S sends a nonce N_S. Next, PS computes the hash $h_S = H(ID_S \| N_S)$ to hide the identity of S from T. In Step (2), PS sends h_S to T. Note that for messages from PS, we use a constant 1 that is identical for all users. In Step (3), T responds with a signed challenge, consisting of the nonce N_T along with the blinded identifier h_S. In Step (4), PS forwards the entire previous message X to PT along with ID_S and N_S. PT checks the message, and in particular if h_S found in X fulfills $h_S = H(ID_S \| N_S)$. Meanwhile, the prover has to confirm manually that she wants to sign up for the service provider S. In that case, PT responds to T's challenge by sending a copy of the message X in Step (5). After verification, in Step (6), the trusted third party computes $h_{PT} = H(ID_{PT} \| N_T)$ to blind the prover's identity, ID_{PT}, and sends a signed authentication ticket to PS which consists of the blinded identifiers h_{PT} and h_S. Henceforth, the trusted third party maps h_{PT} to ID_{PT}. In Step (7), PS forwards the ticket to S. Finally, if the ticket is valid, S assigns h_{PT} to the prover's user account and activates our authentication scheme.

This protocol ensures the privacy properties of our authentication scheme by two means: first, the identifier ID_S of the service provider is blinded to obtain h_S, so that the trusted third party cannot figure out which service provider the prover uses. Second, the trusted third party blinds ID_{PT} to obtain a provider-specific identifier h_{PT}. This way, colluding service providers cannot identify shared users by comparing authenticator identifiers.

3.2 Authentication

A prover P authenticates herself at her service provider S, e.g., when signing in for a new session. Here, the second factor is checked using the authentication protocol shown in Table 3. While all other protocols of our scheme are invoked only occasionally, this protocol is run on a regular basis.

S initiates the authentication protocol. This protocol is designed similar to the aforementioned activation protocol, with the difference that the prover's provider-specific identifier h_{PT} is carried through all steps. In Step (1), the service provider sends a

Table 2. Protocol to activate two-factor authentication.

Protocol 2:	**Activation of the second factor for P at S**
Parties:	PS, PT, S, and T
Pre-conditions:	S is ignorant of PT, T is ignorant of P using S
Post-conditions:	S has verified that P uses h_{PT}, and S stores h_{PT} T stores (ID_{PT}, h_{PT}); T is ignorant of P using S
(1) $PS \leftarrow S$:	$\mathrm{TLS}([\text{ACTIVATE},1,\mathtt{v},ID_S],\ N_S)$
(2) $PS \rightarrow T$:	$\mathrm{TLS}([\text{ACTIVATE},2,\mathtt{v},1],\ h_S)$
(3) $PS \leftarrow T$:	$\mathrm{TLS}(\underbrace{([\text{ACTIVATE},3,\mathtt{v},ID_T],\ h_S,\ N_T)_T}_{X})$
(4) $PS \rightarrow PT$:	$([\text{ACTIVATE},4,\mathtt{v},1],\ X,\ N_S,\ ID_S)$
(5) $PT \rightarrow T$:	$\mathrm{TLS}(([\text{ACTIVATE},5,\mathtt{v},ID_{PT}],\ X)_{PT})$
(6) $PS \leftarrow T$:	$\mathrm{TLS}(\underbrace{([\text{ACTIVATE},6,\mathtt{v},ID_T],\ h_{PT},\ h_S)_T}_{Y})$
(7) $PS \rightarrow S$:	$\mathrm{TLS}([\text{ACTIVATE},7,\mathtt{v},1],\ Y)$

session nonce N_S to ensure freshness along with h_{PT} to PS. In Step (2), PS blinds the service provider's identifier by computing $h_S = H(ID_S \| N_S)$, and sends it together with h_{PT} to T. In Step (3), T responds with a signed challenge containing N_T, h_{PT}, and h_S. PS forwards the entire previous message X along with ID_S and N_S to PT in Step (4), which verifies the incoming message. The prover then is asked to confirm that she wants to authenticate herself at the service provider S. In the affirmative, PT responds to T's challenge by sending a signed copy of the message X in Step (5). After successful verification, in Step (6), T sends a signed authentication ticket consisting of h_{PT} and h_S to PS, which forwards it to the service provider S in Step (7). Finally, if the ticket is valid, S grants P access to her service.

Again, the trusted third party never obtains information about the service provider's identity. Each time the prover logs into her service provider, the provider's identifier is blinded using a fresh nonce. Thus, from the perspective of the trusted third party, every run of the authentication protocol is unique.

3.3 Key Management

The prover's private key is stored on her phone. Losing it locks her out of service providers where she activated our authentication scheme, whereas the lost authenticator may still be used by an adversary to gain access to the prover's accounts. To react in case of such an emergency, corresponding protocols for key revocation and rekeying are provided, which are concerned in the following.

Key-revocation protocol. As an immediate reaction upon the loss of her authenticator, the prover turns to her service provider and logs in with her first factor. When the service provider initiates the authentication protocol, its first three steps are executed automatically. In Step (4), however, instead of proceeding, the prover initiates the key-revocation protocol shown in Table 4 (top). In this case, PS sends a revocation request to T, including the previous message X, and then cancels the login attempt at S. Meanwhile, T revokes the prover's public key if the signature of the revocation request could be verified with the prover's old key. Finally, a confirmation mail is sent to the prover's mail box PM.

Table 3. Protocol to authenticate the second factor.

Protocol 3:	**Authentication of** P **at** S
Parties:	PS, PT, S, and T
Pre-conditions:	S is ignorant of P using PS
Post-conditions:	S has verified that P uses PS

(1) $PS \leftarrow S$: TLS($[\text{AUTH},1,\mathtt{v},ID_S]$, h_{PT}, N_S)
(2) $PS \rightarrow T$: TLS($[\text{AUTH},2,\mathtt{v},1]$, h_{PT}, h_S)
(3) $PS \leftarrow T$: TLS($\underbrace{([\text{AUTH},3,\mathtt{v},ID_T],\ h_{PT},\ h_S,\ N_T)_T}_{X}$)
(4) $PS \rightarrow PT$: ($[\text{AUTH},4,\mathtt{v},1]$, X, N_S, ID_S)
(5) $PT \rightarrow T$: TLS($([\text{AUTH},5,\mathtt{v},ID_{PT}],\ X)_{PT}$)
(6) $PS \leftarrow T$: TLS($\underbrace{([\text{AUTH},6,\mathtt{v},ID_T],\ h_{PT},\ h_S)_T}_{Y}$)
(7) $PS \rightarrow S$: TLS($[\text{AUTH},7,\mathtt{v},1]$, Y)

Rekeying protocol. To regain control of her accounts after key revocation, the prover uses a rekeying ticket that was generated during registration (see Table 1, Step (6)). Using this ticket, the prover initiates the rekeying protocol shown in Table 4 (bottom) to exchange her revoked public key with a new one at the trusted third party. To do so, the prover orders a new, blank authenticator PT from T and forwards the rekeying ticket to PT in Step (1). PT checks the ticket's validity by verifying that $ID_{PT} = H(N_T \parallel H(N_{PT}))$ and then generates a new key pair (K'^s_{PT}, K'^p_{PT}). PT samples a new nonce N'_{PT} at random and computes $h'_{PT} = H(N'_{PT})$. In Step (2), the new public key K'^p_{PT} is sent along with the ticket and h'_{PT} to T. The message is signed using the new secret key K'^s_{PT}. From the ticket, T extracts ID_{PT}, and verifies if $ID_{PT} = H(N_T \| H(N_{PT}))$ holds and if ID_{PT} corresponds to K^p_{PT} in T's database. If successful, T registers K'^p_{PT} as P's new public key and generates a new unique identifier $ID'_{PT} = H(N'_T \| h'_{PT})$, using a fresh nonce N'_T. In Step (3), N'_T is sent to PT, which also computes ID'_{PT} and uses it as its new identifier. Rekeying is completed by sending a new rekeying ticket to the prover's mail box PM in Step (4).

Altogether, from a prover's perspective, the infrequently invoked key-management protocols provide for a consistent experience since manual actions (i.e., passing challenges to the authenticator) are unified with those of registration and authentication.

4 Formal Security Analysis

This section summarizes the results of an in-depth analysis of the security and privacy of the PASSPHONE scheme when employed as second factor in a two-factor-authentication setup. The proofs to our theorems in this section are given in Appendices A and B.

4.1 Authentication-Attack Resistance

Notation. The quality of an adversary $\mathcal{A}$ against a security notion sec is measured by its success probability $\Pr[\mathsf{Succ}_{\mathsf{sec}}]$ in winning a game $\mathcal{G}_{\mathsf{sec}}$ that models sec. Let $x \twoheadleftarrow \mathcal{X}$ denote the sampling of x uniformly at random from a distribution $\mathcal{X}$ and let $\{0,1\}^n$ denote the set of all n-bit strings. We consider a set of provers $\mathcal{P}$ and a set of service

Table 4. Protocols for key revocation and rekeying.

Protocol 4:	**Key revocation via** PS
Parties:	PS, PM, S, and T
Pre-conditions:	T considers K^p_{PT} active; T is ignorant of P using S
Post-conditions:	T has revoked K^p_{PT}; T is ignorant of P using S
	Steps 1-3 of Protocol 3, the authentication protocol.
(4) $PS \rightarrow T$:	$\mathrm{TLS}([\text{REVOKE},1,\mathtt{v},1],\ X)$
(5) $PS \rightarrow S$:	TLS(Cancel login)
(6) $PM \leftarrow T$:	$([\text{REVOKE},2,\mathtt{v},ID_T],\ X)_T$

Protocol 5:	**Rekeying for** PT
Parties:	P, PT, PM, and T
Pre-conditions:	PT may be blank
Post-conditions:	T revoked K^p_{PT} and stores $(ID'_{PT}, K'^p_{PT}, ID_{PM})$
	P received new tickets for rekeying and key transfer
(1) $P \rightarrow PT$:	$\underbrace{([\text{REKEY},1,\mathtt{v},ID_{PT}], N_{PT}, N_T, K^p_{PT})_{PT}}_{X}$
(2) $PT \rightarrow T$:	$\mathrm{TLS}(([\text{REKEY},2,\mathtt{v},0], K'^p_{PT},\ h'_{PT},\ X)_{PT})$
(3) $PT \leftarrow T$:	$\mathrm{TLS}(([\text{REKEY},3,\mathtt{v},ID_T], N'_T)_T)$
(4) $PT \rightarrow PM$:	$\mathrm{TLS}(([\text{REKEY},1,\mathtt{v},ID'_{PT}], N'_{PT}, N'_T, K'^p_{PT})_{PT})$

providers $\mathcal{S}$, where we define that each prover $P^i \in \mathcal{P}$ has a browser instance PS^i and her authenticator PT^i under her control. The set $\mathcal{U}$ denotes the union of $\mathcal{P} \cup \mathcal{S} \cup \{T\}$.

Assumptions. We follow the standard assumption that legitimate parties (provers and service providers in our case) behave *honestly*: they do not understand the semantics of a message before a protocol run completed successfully. We assume that provers, service providers, and the trusted third party communicate over the open Internet, relying on the existing Public-Key Infrastructure (PKI) of TLS for establishing a secure channel with one-sided authentication of S and T towards the prover (PS, PT). This means, we assume that all service providers S and the trusted third party T possess a public key encoded in a valid TLS certificate. The PKI trust assumption is a current best practice for securing the communication between web services and their users. Further, our cryptographic model assumes that the client PS does not manage any permanent state, which is reasonable for a web browser, and that PS executes a correct version of the protocols (e.g., code that was signed by T).

We recommend that all honest parties employ certificate or public-key pinning for the trusted third party and for service providers (i.e., mapping the hosts to their expected X.509 certificate or public key by explicit whitelisting). Moreover, we propose to bind TLS connections to specific channels by employing a fixed version of either the *tls-unique* approach from RFC 5929 [2] or Google's *Channel ID* [6] (see [8,9,24] for attacks and fixes).

Adversarial model. The goal of the probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ is to authenticate as some honest prover P^i at some honest service provider S^j. $\mathcal{A}$ is aware of the behavioral limitations of honest parties and tries to exploit them. The adversary can eavesdrop, intercept, insert, modify, or delete all communication that is transmitted over the network, but cannot modify the communication transmitted

from the prover's browser to her authenticator, which is a fair assumption when using, e.g., scanned QR codes. $\mathcal{A}$ can impersonate a prover, a service provider, or both. The use of TLS prevents it from acting as T or as an honest S in the view of the prover. Moreover, we assume that the cryptographic primitives used are secure. So, $\mathcal{A}$ cannot recover a secret key, predict a random value, find a hash-value's preimage, a collision, or forge a signature with significant advantage. Prior to registration and activation, all parties agree on a security parameter τ,[1] so that all signatures are of length at least τ bits, all nonces and hash values created by H have 2τ bits, and all symmetric and asymmetric secret keys for encryption (again, for TLS) and signing have an effective key length of at least τ bits.

We define an authentication game denoted $\mathcal{G}^{\mathsf{Auth}}$, which takes as input a tuple (τ, q_{exe}, q_{send}, q_{test}), and provides $\mathcal{A}$ with access to the following queries:

- $\mathsf{Setup}(1^\tau)$: The registration and activation steps are executed once to generate the secrets of all involved parties.
- $\mathsf{Execute}(P^i, S^j, T)$: Models a *passive* adversary $\mathcal{A}$ who eavesdrops a correct execution of the authentication protocol between a prover P^i, a service provider S^j, and T. The output is given by the transcript of the protocol between P^i, S^j, and T.
- $\mathsf{Send}(U, U', m)$: Models an *active* attack, wherein the adversary $\mathcal{A}$ intercepts, modifies, replays, forwards, or creates a message m in the name of party U to party U', where $U, U' \in \mathcal{U}$. The output of such a query is the message that U' would generate after receiving m. A special message Start can be sent in the name of a prover to a service provider to initiate a session between them with the trusted third party.
- $\mathsf{Corrupt}(P^i, S^j)$: Models that the secret for the first factor $pwd^{i,j}$ of P^i at S^j has been compromised. The output of this query is $pwd^{i,j}$.
- $\mathsf{Test}(P^i, S^j)$: Models an authentication request of $\mathcal{A}$ in the name of P^i at service provider S^j. The output is a bit b, which is 1 if and only if the authentication succeeds and P^i and S^j are honest; otherwise b is 0.

For all inputs, the output bit b of $\mathsf{Test}(P^i, S^j)$ after a correct execution of the authentication protocol between honest P^i and S^j will always be 1. We define that any honest party immediately aborts a protocol run if it detects an invalid message, i.e., an incorrect signature, unexpected service provider, incorrect ID, non-matching hash, or invalid message format.

Theorem 1. *Let the employed public-key signature scheme be EUF-CMA-secure and H be a random oracle. Then, for any PPT adversary $\mathcal{A}$ whose run time is bounded by t and whose number of execute, send, and test queries are bounded by q_{exe}, q_{send} and q_{test}, respectively, it holds for a random execution of $\mathcal{G}^{Auth}$ on our protocol $\mathbb{P}$ that $\Pr[Succ_{Auth}] \leq q \cdot 4/2^\tau$, where $q = q_{exe} + q_{send} + q_{test}$.*

4.2 Anonymity

In the context of an outsourced three-party protocol, user anonymity refers to the goal that an honest but curious trusted third party is unable to learn which service provider(s) an individual prover has registered with and wants to authenticate to. We

[1] In practice, $\tau \geq 128$ is fixed a-priori by the protocol (version).

model this goal by a game $\mathcal{G}^{\mathsf{Anon}}$ and an adversary $\mathcal{A}$ who plays the role of T, i.e., $\mathcal{A}$ has access to IDs, public keys, and blinded IDs $\langle ID^i_{PT}, K^p_{PT^i}, \langle h^j_{PT^i} \rangle \rangle$ of all provers P^i. We define that at least one honest prover P and two honest service providers S^0 and S^1 exist in the game. At setup, the challenger tosses a fair coin to obtain a bit b. Depending on b, P registers with S^b, and generates a secret pwd for the first factor. We define a special service provider $\widehat{S}$ which wraps S^0 and S^1 and appears as a black box to $\mathcal{A}$. So, every time S^0 or S^1 are involved in an execution of our protocols, the game models it as an execution with $\widehat{S}$ in the view of $\mathcal{A}$.

$\mathcal{A}$ is given access to the queries $\mathsf{Setup}(1^\tau)$, $\mathsf{Execute}(\pi, P^i, S^j, T)$, $\mathsf{Send}(\pi, U, U', m)$, which work similarly to their equivalents in the authentication game above. As a difference, $\mathcal{A}$ must provide a parameter $\pi \in \{\text{REG, ACTIVATE, AUTH, REKEY, REVOKE}\}$ to execute the different protocols. $\mathcal{A}$ is not given access to $\mathsf{Corrupt}$ queries, assuming an honest but curious adversary. Wlog., we assume that $\mathcal{A}$ asks no Send queries to T since it can always answer them without interaction from other parties with the help of T's private key. Moreover, we define that $\mathcal{A}$ is prohibited from using S^0 or S^1 in its send or execute queries, and may only use $\widehat{S}$ instead. At the end of the game, $\mathcal{A}$ makes a $\mathsf{Test}(b')$ query, to which it must provide a bit b'. $\mathcal{A}$ wins the game $\mathcal{G}^{\mathsf{Anon}}$ if and only if $b' = b$, i.e., if it successfully guesses which service provider P has registered with. We denote this event by $\mathsf{Succ}_{\mathsf{Anon}}$ and define the anonymity advantage of $\mathcal{A}$ against a protocol scheme $\mathbb{P}$ as $\mathbf{Adv}^{\mathsf{Anon}}_{\mathbb{P}}(\mathcal{A}) = 2 \cdot |\Pr[\mathsf{Succ}_{\mathsf{Anon}}] - 0.5|$.

Theorem 2 (Anonymity). *Let the employed public-key signature scheme be EUF-CMA-secure and H be a random oracle. Then, for any PPT adversary $\mathcal{A}$ whose run time is bounded by t and which asks at most q_{exe} execute and q_{send} send queries, respectively, it holds for a random execution of $\mathcal{G}^{Anon}$ on our protocol $\mathbb{P}$:*

$$\boldsymbol{Adv}^{Anon}_{\mathbb{P}}(\mathcal{A}) \leq (q_{exe} + q_{send}) \cdot 1/2^{2\tau}.$$

4.3 Unlinkability

For authenticated key-exchange schemes, Tsudik and Xu [41] define unlinkability as the property that no adversary $\mathcal{A}$ can associate two handshakes involving the same honest party even if $\mathcal{A}$ participated in both executions. In the context of web authentication, unlinkability means that no set of colluding service providers is able to link a prover registered with multiple of their services. Clearly, there must be at least two uncorrupted users to prevent the adversary from deducing trivially which two executions involve the same party.

We define a third game $\mathcal{G}^{\mathsf{Unlink}}$ wherein $\mathcal{A}$ plays the role of two disjoint service providers S^0 and S^1. The challenger plays the role of two honest provers P^0 and P^1 and T. At the beginning, the challenger tosses a fair coin to obtain a bit b; if $b = 1$, the challenger registers P^0 with both S^0 and S^1, and P^1 with none of them. If $b = 0$, the challenger registers P^0 with S^0 but not with S^1, and P^1 with S^1 but not with S^0. Likewise to the anonymity game, we define a special prover $\widehat{P}$ which wraps P^0 and P^1 and appears as a black box to $\mathcal{A}$. So, every time P^0 or P^1 is involved in an execution of a protocol, the game models this as an execution with $\widehat{P}$ instead of P^0 or P^1 in the view of $\mathcal{A}$. As before, this configuration can be augmented by many more honest provers and service providers. Additionally, $\mathcal{A}$ can control a set of malicious provers $\mathcal{E}_P$ as well as malicious service providers $\mathcal{E}_S$.

$\mathcal{A}$ is given access to queries of the types $\mathsf{Setup}(1^\tau)$, $\mathsf{Execute}(\pi, P^i, S^j, T)$, and $\mathsf{Send}(\pi, U, U', m)$, for parties $U, U' \in \mathcal{U}$, which work similar to their equivalents in the anonymity game above. This time, $\mathcal{A}$ is prohibited from using P^0 or P^1 in its queries, and must use $\widehat{P}$ as a replacement. When $\mathcal{A}$ uses $\widehat{P}$ and either of S^0 and S^1 in an execute or send query, the challenger uses the prover as a replacement for $\widehat{P}$ that can process the execution of the protocol correctly. Moreover, if $\mathcal{A}$ invokes the registration, activation, rekeying, or revocation protocol for $\widehat{P}$, the challenger executes it for both P^0 and P^1. At the end of the game, $\mathcal{A}$ makes a $\mathsf{Test}(b')$ query and has to provide the bit b'. $\mathcal{A}$ wins the game $\mathcal{G}^{\mathsf{Unlink}}$ if and only if $b' = b$. We denote this event by $\mathsf{Succ}_{\mathsf{Unlink}}$ and define the unlinkability advantage of an adversary $\mathcal{A}$ against a protocol scheme $\mathbb{P}$ as $\mathbf{Adv}_{\mathbb{P}}^{\mathsf{Unlink}}(\mathcal{A}) = 2 \cdot |\Pr[\mathsf{Succ}_{\mathsf{Unlink}}] - 0.5|$.

Theorem 3 (Unlinkability). *Let the employed public-key signature scheme be EUF-CMA-secure and H be a random oracle. Then, for any PPT adversary $\mathcal{A}$ whose run time is bounded by t and which asks at most q_{exe} execute and q_{send} send queries, it holds for a random execution of $\mathcal{G}^{Unlink}$ on our protocol $\mathbb{P}$:*

$$\boldsymbol{Adv}_{\mathbb{P}}^{Unlink}(\mathcal{A}) \leq (q_{exe} + q_{send}) \cdot 1/2^{2\tau}.$$

5 Automatic Security Analysis

Besides the formal security analysis, we also conducted an automatic security analysis of Passphone using the well-known computer-aided proof system AVISPA. After a brief overview of AVISPA's capabilities, we describe the HLPSL implementations of our protocols and the results obtained from feeding them to AVISPA. Moreover, we conduct experiments by deliberately removing security features from our protocols and observing the results from the proof system.

Background. AVISPA provides four backends for protocol verification: a Constraint-Logic-based ATtack SEarcher (CL-ATSE) [42], an On-the-Fly Model Checker (OFMC) [7], a SAT-based Model Checker (SAT-MC) [5], and a Tree-Automata-based backend (TA4SP) [10]. We rely on the widespread CL-ATSE, OFMC, and SAT-MC backends; TA4SP does not support our setup. As input to AVISPA, protocols must be implemented in the High-Level Protocol Specification Language (HLPSL) [13]. HLPSL is a role-centric language well-suited for software engineers and protocol designers.

Implementation details. All of Passphone's protocols have been implemented in HLPSL. Appendix D lists the protocol implementations, and the source code is also available via Passphone's web page at `http://www.passphone.org`. Special care was taken to align the implementation as closely as possible with the protocol specifications found in this paper so as to ensure that the results obtained from AVISPA allow for drawing conclusions about them. For consistency and where the syntax allowed it, variable names have been chosen to correspond with those used in the formal specification as well. The two communication channels send (SND) and receive (RCV) are defined in terms of the Dolev-Yao model (dy).

Since our protocols make use of TLS, this has to be reflected in our HLPSL implementation. However, at present, neither AVISPA nor HLPSL support modularization of protocol implementations, so that the implementation of the TLS protocol in HLPSL cannot be invoked from ours. When mixing both protocol implementations into one file, this severely affects legibility. Therefore, for simplicity, we model TLS by means of

Table 5. Results from AVISPA when omitting TLS in individual protocols. A • indicates that TLS is mandatory to uphold security, and a ∘ that TLS is optional.

Protocol	Communication step						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Registration	•	n/a	n/a	•	•	n/a	
Activation	•	•	•	n/a	∘	•	•
Authentication	•	•	•	n/a	∘	•	•
Key Revocation	•	•	•	•	•		
Rekeying	n/a	•	•	n/a			

public keys assigned to each party, which ensure both encryption and sender authenticity. This approach is sound and has been applied in several other high-level protocol implementations using TLS.

Experiments and results. We fed each protocol's HLPSL implementation to AVISPA and found that all of the aforementioned backends report that they cannot identify any attacks. However, since implementations can be erroneous and since there is currently no standardized unit-testing framework for HLPSL protocol implementations, we conduct experiments and sanity checks in order to verify that our implementation meets our expectations from the manual security analysis. First, we changed each protocol's implementation in a deliberate attempt to make it insecure. The flaws introduced include the removal of TLS for data-origin authentication, signatures, and nonces which opened various attack vectors. We then fed the flawed versions to AVISPA in order to check whether it picks up the vulnerabilities. Without fail, AVISPA identified them. This experiment serves to raise confidence both that the authentication scheme comprises little redundancy and that our implementation reflects well our scheme's formal specification. Second, we were particularly interested whether and to what extent TLS is required to secure our protocols. We employ TLS mainly as a means for data-origin authentication, whereas message encryption is optional. Since TLS is the de facto standard in secure web communications, using a different protocol would severely limit the applicability and acceptance of our authentication scheme in practice. We systematically disabled TLS in a given step of a protocol, re-running AVISPA each time to identify potential attacks that result from doing so. Table 7 summarizes the results for each protocol. As expected, turning TLS off allows for man-in-the-middle attacks in most steps that result from missing data-origin authentication.

6 Prototype Implementation

We implemented all of Passphone's protocols as a proof-of-concept prototype, which is freely available at `https://www.passphone.org`. This section discusses a selection of implementation details.

Trusted third party T**.** The trusted third party is a web service that offers an API used by authenticators and the prover's browser PS. We implemented it as a Java Servlet to share the implementations for message encoding and cryptography between T and that from our current smartphone implementation. To protect the signing key, we recommend the use of a cryptographic module—e.g., according to the FIPS-140 standard [33]—which protects the signing key of the trusted third party from being

copied and which would accelerate cryptographic computations for scalability. This would render compromising the trusted third party's key much more difficult compared to keeping it on hard disk.

Service provider S and client PS. For our prototype, we implemented two service provider stacks: one as a Ruby-based service running on an nginx server with a MySQL database, and a similar second service provider as a Java Servlet. The trusted third party provides plugins for the most widely-used web software stacks (LEMP/ LAMP, Ruby on Rails, etc.), authentication libraries, and web applications. However, given the large number of possible configurations, it is difficult to provide a plugin for each one right away. To minimize the development overhead, we divide plugins into a major, canonical part, and a lightweight, stack-specific part. The major components may be deployed into a virtual machine or on a dedicated server to be run next to an existing service. The lightweight plugins offer the stack-specific API to handle our authentication scheme so that the required changes to existing services are minimally invasive.

Authenticator PT. We implemented the prover's mobile authenticator as a smartphone App for Android devices with SDK 16 and above which currently supports more than 96% of Android smartphones on the market.[2] The widespread distribution of Android smartphones made this design decision straightforward in terms of usability since they are among the few things many people carry with them at all times. We employed the BouncyCastle library[3] for cryptographic primitives, using SHA-256 as hash function and 256-bit EC-DSA as signature scheme, and the ZXing library[4] for handling QR codes.

Challenge encoding and transmission. We resort to QR codes for encoding challenges to reduce the typing effort for the user [19,38,40,44]. QR codes exploit the physical proximity of the prover's devices by changing the communication medium in a way so that an adversary cannot intercept a transmitted message unless looking over the prover's shoulder. In general, the more coarse-grained a QR code can be made, the more robust it is with regard to legibility in various situations of screens, lighting, and camera quality. In our setup, we keep the messages that are transmitted via QR codes small by the use of EC-DSA instead of, for example, RSA-based signatures. Our tests show that scanning QR codes is a robust channel when employing version-10 codes (which can encode up to 213 bytes) and medium-level error correction (15% of codewords can be restored).

Performance and usability. To estimate the performance of our implementation, we evaluate the run times needed for the authentication protocol. We use two dual-core mobile phones with 1.2 GHz (Samsung-Intrinsity Exynos S5PV310) and 1.7 GHz (Qualcomm Snapdragon 400) processors and cameras with resolutions of eight megapixels. We conduct 20 authentication processes. Besides logging in with the first factor, the majority of time was spent to align the QR code, which took trained smartphone users about 3-5 seconds on average, whereas the ZXing library picks up a QR code as soon as it is in view.

In terms of usability, our implementation adopts the current best practices—e. g., scanning of QR codes—employed in phone-based authentication. Since the required user actions do not differ from those of other authentication schemes employed in practice, we omit a detailed discussion of usability. Nevertheless, we have tested and used our implementation on human test subjects. Our prototype has been deployed as an

[2] `https://developer.android.com/about/dashboards`, State of Aug 1, 2016.

[3] `http://bouncycastle.org/`

[4] `https://github.com/zxing/zxing`

exhibit at a recent open house presentation. On that occasion, laymen from the general public as well as interested colleagues from other universities for a total of 55 people have tried our prototype. We observed that all visitors expressed concern for their own security, and understood the concept and importance of privacy preservation in authentication. All regular smartphone users among our testers had little to no difficulty in following the instructions given by our App, as all of them said they occasionally scan QR codes, and, with little explanation (i.e., within less than three minutes), all interested visitors also managed to perform a test run of the rekeying protocol. Altogether, in terms of usability, our prototype is on par with the state of the art in that it adopts their best practices, but of course a lot has still to be done to achieve maturity.

7 Comparative Evaluation

This section compares PASSPHONE to others from the literature under the framework of Bonneau et al. [11]. Table 6 summarizes the results of comparing our scheme to 10 other smartphone-based two-factor authentication schemes with respect to 25 common features an authentication scheme can offer.[5] The features have been collected by Bonneau et al., and while their names may seem self-explanatory, some of their definitions are intricate. For many features, Bonneau et al. also specify a *quasi*-variant, where an authentication scheme offers a feature with some reservations. In what follows, we discuss PASSPHONE's rating in comparison to that of the others.

Usability. As outlined above, PASSPHONE is on par with previously published schemes in terms of usability since it adopts their best practices (i.e., transmitting QR codes via smartphones has been studied already). Therefore, we consider our scheme *Quasi-Scalable-for-Users* since it reduces the risks of password reuse similar to PHONEAUTH, and *Quasi-Nothing-To-Carry*, based on the assumption that smartphones will continue to spread. Likewise, our scheme is quite *Easy-to-Learn* since scanning QR codes is a daily routine for regular smartphone users. During authentication, the user has to enter only her password as a first factor, which results in *Quasi-Infrequent-Errors*, and which makes it *Quasi-Efficient-to-Use*. More generally, our scheme provides equivalent usability compared to GOOGLE 2-STEP, but performs better than PHOOLPROOF, CRONTO, and TIQR, because it features *Easy-Recovery-From-Loss* based on our extensive key management protocols. Arguably, key management may be added to these schemes, but corresponding research is still missing.

Deployability. Concerning deployability, PASSPHONE outperforms most other solutions. PHONEAUTH and TIQR have the highest ratings with respect to Bonneau et al.'s framework, whereas TIQR is more mature. Our scheme is *Quasi-Accessible* since it is compatible with screen readers on both desktop and mobile. Moreover, it has *Quasi-Negligible-Cost-per-User* since no SMS need to be delivered. Our scheme requires only small changes at service site (i.e., the integration of a plugin), which renders it *Quasi-Server-Compatible*. In this regard, our scheme is comparable to PHOOLPROOF, which has been similarly assessed in [11]. Beyond JavaScript, our scheme has no requirements to the prover's browser, which sets it apart from PHONEAUTH or PHOOLPROOF.

We do not fully agree with the rating of PHONEAUTH provided by its authors regarding *Maturity* as well as *Browser-Compatibility*: currently, the research prototype

[5] Regarding GOOGLE 2-STEP, we adopt the rating from [15] since one of that paper's authors works at Google Security and may have deeper insights into their scheme; regarding the proposals from [38], we consider the mid-bandwidth and the full-bandwidth schemes with a similar security level as ours.

Table 6. Comparison of phone-based two-factor authentication schemes according to the evaluation framework for authentication schemes by Bonneau et al. [11]. The framework considers 25 features an authentication scheme can offer with respect to usability, deployability, and security. Each column names one feature, and each scheme is rated based on whether it offers the feature (•), it quasi offers the feature with reservations (◦), or it does not offer the feature (–).

Authentication scheme		**Usability**								**Deployability**						**Security** (*Res.* = Resilient)											**Summary**	
		Memorywise-Effortless	Scalable-for-Users	Nothing-to-Carry	Physically-Effortless	Easy-to-Learn	Efficient-to-Use	Infrequent-Errors	Easy-Recovery-from-Loss	Accessible	Negligible-Cost-per-User	Server-Compatible	Browser-Compatible	Mature	Non-Proprietary	Res.-to-Physical-Observation	Res.-to-Targeted-Impersonation	Res.-to-Throttled-Guessing	Res.-to-Unthrottled-Guessing	Res.-to-Internal-Observation	Res.-to-Leaks-from-Other-Verifiers	Res.-to-Phishing	Res.-to-Theft	No-Trusted-Third-Party	Requiring-Explicit-Consent	Unlinkable	#•	#◦
Cronto	[45]	–	–	◦	–	•	◦	◦	–	–	◦	–	•	•	–	•	•	•	•	◦	•	•	•	•	•	•	13	5
FBD-BT-BT/WF-WF	[38]	–	◦	◦	–	•	•	•	–	◦	◦	–	–	–	•	•	•	•	•	•	•	–	•	•	–	•	13	4
FBD-QR-BT/WF	[38]	–	◦	◦	–	•	•	◦	–	◦	◦	–	–	–	•	•	•	•	•	•	•	–	•	•	•	•	13	5
Google 2-step	[21]	–	–	◦	–	•	◦	◦	◦	◦	–	–	•	•	–	–	◦	•	–	–	•	•	•	•	•	•	10	6
MBD-QR-QR	[38]	–	◦	◦	–	◦	◦	–	–	◦	◦	–	◦	–	•	–	•	•	•	–	•	–	•	•	•	•	9	7
MP-Auth	[29]	–	–	◦	–	•	◦	–	◦	◦	◦	–	–	–	•	–	◦	–	–	–	–	•	•	•	•	•	7	6
PhoneAuth (opportunistic)	[15]	–	◦	◦	–	•	•	◦	•	•	•	◦	–	◦	•	◦	◦	◦	◦	◦	◦	◦	•	•	•	◦	9	13
PhoolProof	[34]	–	–	◦	–	•	◦	◦	–	◦	◦	◦	–	–	•	•	•	•	•	◦	•	•	•	•	•	•	12	7
SoundProof	[25]	–	–	◦	–	•	•	◦	◦	•	•	–	•	–	•	◦	–	•	•	–	•	•	•	•	•	–	13	4
Tiqr	[44]	–	–	◦	–	•	◦	◦	–	◦	◦	◦	•	•	•	–	•	–	–	◦	•	◦	•	•	•	•	10	8
Passphone	(this paper)	–	◦	◦	–	•	◦	◦	•	◦	◦	◦	•	–	•	•	•	•	•	–	•	•	•	–	•	•	**13**	**7**

seems unavailable at any public outlet, and the scheme works only with an experimental version of Google Chrome. Thus, we demoted PhoneAuth's ratings accordingly, compared to those reported in [15]. Obviously, being a research prototype, our scheme is also not mature, yet.

Security. Concerning security, Passphone is almost on par with the two best-performing schemes PhoolProof and Cronto, the only difference being that our scheme involves a trusted third party. While resorting to trusted third parties is often avoided in security protocols, we argue that including a trusted third party becomes a lot less detrimental when incorporating user privacy. It is an open question if this consideration merits introducing the feature *Quasi-No-Trusted-Third-Party* into Bonneau et al.'s framework, but we refrained from doing so in our evaluation. In general, our scheme covers all security-related features, but we cannot guarantee *Resilience-to-Internal-Observation*; if an adversary has full control over the prover's device, she might be able to recover the secret key. Our threat model does not cover this case and we leave it for future work. Finally, we would like to point out that our scheme features *Unlinkability* despite the fact that it uses a trusted third party.

For ease of comparison, Column "Summary" in Table 6 gives the counts of features and quasi-features. Altogether, our scheme offers as many full features as the competition despite suffering losses for introducing a trusted third party and for not being mature, yet. This is encouraging since this evaluation demonstrates the potential of our authentication scheme for future research and development as well as for transfer into practice.

8 Practical Application

Choosing the first factor. Similar to other phone-based two-factor authentication schemes from the literature, PASSPHONE does not aim at replacing the still prevalent password authentication, but at strengthening it in a two-factor setup. The option of outsourcing the verification of the second factor plus the privacy properties of our scheme, however, renders it attractive for small service providers since it enables them to add two-factor authentication with comparably small development overhead to their existing authentication solution. The first factor used in conjunction with our protocols is therefore not at all tied to the use of login and password; for example, it can be based on physical tokens, biometric properties, or another challenge-response protocol. In practice, however, most service providers still employ passwords as a first factor, exchanging passwords over TLS, processing them with a password-hashing function, and storing them at server side as salted password hashes. Nevertheless, PASSPHONE's security does not rest with the first factor employed.

Limitations of web-based authentication. Regarding authentication for web services, we concede that privacy-unaware users may still easily be tracked by means not related to our protocol (e.g., by searching for reused mail addresses or credentials). Moreover, users should be aware that their browser and OS configuration is used by many tracking services. Anonymous communication techniques, such as TOR [17], can be combined with PASSPHONE to also provide IP-level anonymity and unlinkability; however, securing the user from all privacy perils is clearly beyond the scope of what a web-based authentication protocol can address. We stress, however, that PASSPHONE does not introduce yet another angle of de-anonymizing users, which is a first in the domain of web authentication.

9 Conclusion

This work introduces PASSPHONE, a new phone-based two-factor authentication scheme, consisting of all protocols necessary for bootstrapping, authentication, and key management. PASSPHONE is designed with a focus on deployability: it allows for easy integration at service providers by outsourcing authentication to a trusted third party. Moreover, it is the first web-based three-party authentication scheme that protects the privacy of its users by minimizing the amount of information shared among the parties involved, hiding the relation of users and service providers from the trusted third party, and rendering users unlinkable among service providers. We analyze PASSPHONE's security, show its privacy properties, and present insights from a proof-of-concept implementation. Under the authentication scheme evaluation framework of Bonneau et al., our scheme competes with the best-performing ones from the literature. In conclusion, with the success of outsourcing first-factor authentication, also outsourcing the second-factor authentication in a two-factor setup is reasonable, albeit, ideally using different trusted third parties for each factor to spread risks. We hope that PASSPHONE's privacy properties will inspire more privacy-awareness in future protocol designs.

Acknowledgments

The authors thank Anne Barsuhn, Thomas Dressel, Paul Christoph Götze, André Karge, Tom Kohlberg, Kevin Lang, Christopher Lübbemeier, Kai Gerrit Lünsdorf,

Nicolai Ruckel, Sascha Schmidt, and Clement Welsch for implementing the first prototype within student projects. Our special thanks go to Thomas Dressel and André Karge for their pursuing work, and to Benno Stein and the anonymous reviewers for valuable comments and suggestions.

References

1. Fadi A. Aloul, Syed Zahidi, and Wassim El-Hajj. Two Factor Authentication Using Mobile Phones. In E. M. Aboulhamid and J. L. Sevillano, editors, *AICCSA'09*, pages 641–644. IEEE Computer Society.
2. J. Altman, N. Williams, and L. Zhu. Channel Bindings for TLS. RFC 5929 (Proposed Standard), July 2010.
3. Apple. Two-factor authentication for Apple ID, 2016.
4. Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In K. Etessami and S. K. Rajamani, editors, *CAV*, volume 3576 of *LNCS*, pages 281–285. Springer, 2005.
5. Alessandro Armando, Luca Compagna, and Pierre Ganty. SAT-Based Model-Checking of Security Protocols Using Planning Graph Analysis. In Keijiro Araki, Stefania Gnesi, and Dino Mandrioli, editors, *FME*, volume 2805 of *LNCS*, pages 875–893. Springer, 2003.
6. Dirk Balfanz and R. Hamilton. Transport Layer Security (TLS) Channel IDs, Nov 8 2013. IETF Internet Draft v01, expired May 12, 2013.
7. David A. Basin, Sebastian Mödersheim, and Luca Viganò. An On-the-Fly Model-Checker for Security Protocol Analysis. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS*, volume 2808 of *LNCS*, pages 253–270. Springer, 2003.
8. Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Alfredo Pironti, and Pierre-Yves Strub. Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. In *IEEE Symposium on Security and Privacy*, pages 98–113, 2014.
9. Karthikeyan Bhargavan, Antoine Delignat-Lavaud, and Alfredo Pironti. Verified Contributive Channel Bindings for Compound Authentication. In *NDSS*. The Internet Society, 2015.
10. Yohan Boichut, Pierre-Cyrille Héam, and Olga Kouchnarenko. Automatic Verification of Security Protocols Using Approximations. Technical report, INRIA-Lorraine - CASSIS Project, October 2005.
11. Joseph Bonneau, Cormac Herley, Paul C. van Oorschot, and Frank Stajano. The Quest to Replace Passwords: A Framework for Comparative Evaluation of Web Authentication Schemes. In *IEEE Symposium on Security and Privacy*, pages 553–567, 2012.
12. Joseph Bonneau and Sören Preibusch. The password thicket: Technical and market failures in human authentication on the web. In *WEIS*, 2010.
13. Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Jacopo Mantovani, Sebastian Moedersheim, and Laurent Vigneron. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *SAPS*, page 13 p, 2004.

14. Dwaine E. Clarke, Blaise Gassend, Thomas Kotwal, Matt Burnside, Marten van Dijk, Srinivas Devadas, and Ronald L. Rivest. The Untrusted Computer Problem and Camera-Based Authentication. In F. Mattern and M. Naghshineh, editors, *Pervasive*, volume 2414 of *LNCS*, pages 114–124. Springer, 2002.
15. Alexei Czeskis, Michael Dietz, Tadayoshi Kohno, Dan S. Wallach, and Dirk Balfanz. Strengthening User Authentication Through Opportunistic Cryptographic Identity Assertions. In T. Yu, G. Danezis, and V. D. Gligor, editors, *ACM CCS*, pages 404–414, 2012.
16. Arkajit Dey and Stephen Weis. PseudoID: Enhancing Privacy in Federated Login. In A. Serjantov and C. Troncoso, editors, *Hot Topics in PETS*, pages 95–107, 2010.
17. Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.
18. Ben Dodson, Debangsu Sengupta, Dan Boneh, and Monica S. Lam. Secure, Consumer-Friendly Web Authentication and Payments with a Phone. In M. L. Gris and G. Yang, editors, *MobiCASE*, volume 76 of *LNICST*, pages 17–38, 2010.
19. Ben Dodson, Debangsu Sengupta, Dan Boneh, and Monica S. Lam. Snap2Pass: Consumer-Friendly Challenge-Response Authentication with a Phone. http://prpl.stanford.edu/papers/soups10j.pdf, 2010.
20. Gemalto. 2014 - Year of Mega Breaches and Identity Theft – Findings from the 2014 Breach Level Index, 2014.
21. Google. 2-step Authentication, April 2013.
22. Steffen Hallsteinsen, Ivar Jorstad, and Do Van Thanh. Using the Mobile Phone as a Security Token for Unified Authentication. In *ICSNC*, page 68, 2007.
23. D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), October 2012.
24. Nikolaos Karapanos and Srdjan Capkun. On the Effective Prevention of TLS Man-in-the-Middle Attacks in Web Applications. In K. Fu and J. Jung, editors, *USENIX Security*, pages 671–686. USENIX Association, 2014.
25. Nikolaos Karapanos, Claudio Marforio, Claudio Soriente, and Srdjan Capkun. Sound-Proof: Usable Two-Factor Authentication Based on Ambient Sound. In *USENIX Security*, pages 483–498, 2015.
26. Bob Lord. Keeping our users secure (Twitter Official Blog), Feb 01 2013.
27. T. Alexander Lystad. Leaked Password Lists and Dictionaries - The Password Project, May 2013.
28. Mohammad Mannan and P. C. Van Oorschot. Using a Personal Device to Strengthen Password Authentication from an Untrusted Computer. In S. Dietrich and R. Dhamija, editors, *FC*, volume 4886 of *LNCS*, pages 88–103, 2007.
29. Mohammad Mannan and P. C. van Oorschot. Leveraging Personal Devices for Stronger Password Authentication from Untrusted Computers. *J. Comput. Secur.*, 19(4):703–750, December 2011.
30. Jeffrey Meisner. The Official Microsoft Blog: Microsoft Account Gets More Secure, April 17 2013.
31. David Nuñez and Isaac Agudo. BlindIdM: A privacy-preserving approach for identity management as a service. *International Journal of Information Security*, 13(2):199–215, 2014.
32. David Nunez, Isaac Agudo, and Javier Lopez. Integrating OpenID with Proxy Re-encryption to Enhance Privacy in Cloud-based Identity Services. In *CloudCom*, pages 241–248, 2012.
33. U.S. National Institute of Standards and Technology. Validated FIPS 140-1 and FIPS 140-2 Cryptographic Modules, 2013. Last update: October 25, 2013.

34. Bryan Parno, Cynthia Kuo, and Adrian Perrig. Phoolproof Phishing Prevention. In G. Di Crescenzo and A. D. Rubin, editors, *FC*, volume 4107 of *LNCS*, pages 1–19, 2006.
35. Martin Potthast, Christian Forler, Eik List, and Stefan Lucks. Passphone: Outsourcing Phone-based Web Authentication while Protecting User Privacy. In *NordSec2016*, 2016. To appear.
36. David Recordon and Drummond Reed. OpenID 2.0: A Platform for User-Centric Identity Management. In A. Juels, M. Winslett, and A. Goto, editors, *Digital Identity Management*, pages 11–16. ACM, ACM, 2006.
37. Philip J Riesch and Xiaojiang Du. Audit Based Privacy Preservation for the OpenID Authentication Protocol. In *2012 IEEE Conference on Technologies for Homeland Security*, pages 348–352. IEEE, 2012.
38. Maliheh Shirvanian, Stanislaw Jarecki, Nitesh Saxena, and Naveen Nathan. Two-Factor Authentication Resilient to Server Compromise Using Mix-Bandwidth Devices. In *NDSS*. The Internet Society, 2014.
39. Andrew Song. Introducing Login Approvals, May 12 2011.
40. Guenther Starnberger, Lorenz Froihofer, and Karl M. Göschka. QR-TAN: Secure Mobile Transaction Authentication. In *ARES*, pages 578–583. IEEE Computer Society, 2009.
41. Gene Tsudik and Shouhuai Xu. A Flexible Framework for Secret Handshakes. In G. Danezis and P. Golle, editors, *PETS*, volume 4258 of *LNCS*, pages 295–315. Springer, 2006.
42. Mathieu Turuani. The CL-Atse Protocol Analyser. In Frank Pfenning, editor, *RTA*, volume 4098 of *LNCS*, pages 277–286, 2006.
43. Manuel Urueña, Alfonso Muñoz, and David Larrabeiti. Analysis of Privacy Vulnerabilities in Single Sign-On Mechanisms for Multimedia Websites. *Multimedia Tools and Applications*, 68(1):159–176, 2014.
44. Roland M. Van Rijswijk and Joost Van Dijk. Tiqr: A Novel Take on Two-factor Authentication. In T. A. Limoncelli and D. Hughes, editors, *LISA*. USENIX Association, 2011.
45. Data Security International VASCO. Cronto, 2013.
46. Min Wu, Simson Garfinkel, and Rob Miller. Secure Web Authentication with Mobile Phones. In *DIMACS Workshop on Usable Privacy and Security Software*, 2004.

A Security Analaysis

This section provides our proof of Theorem 1. We restate it below for the sake of readability.

Theorem 1. *Let the employed public-key signature scheme be* EUF-CMA*-secure and H be a random oracle. Then, for any PPT adversary $\mathcal{A}$ whose run time is bounded by t and whose number of execute, send, and test queries are bounded by q_{exe}, q_{send} and q_{test}, respectively, it holds for a random execution of $\mathcal{G}^{\mathsf{Auth}}$ on our protocol $\mathbb{P}$ that $\Pr[\mathsf{Succ}_{\mathsf{Auth}}] \leq q \cdot 4/2^{\tau}$, where $q = q_{exe} + q_{send} + q_{test}$.*

Proof. In the following, we say that $\mathcal{A}$ has managed to be successfully authenticated with the first factor for some honest prover P^i at some honest service provider S^j using a corruption query $\mathsf{Corrupt}(P^i, S^j)$. For also being authenticated with the second factor, $\mathcal{A}$ must provide S^j with a valid authentication ticket. Therefore, $\mathcal{A}$ must achieve at least one of the following:

1. to forge the (signature of the) ticket,
2. to replay an old accepted ticket from a previous session,
3. to obtain a fresh valid ticket.

We upper bound the success probabilities of $\mathcal{A}$ in the individual cases in the following.

Case 1. Concerning the first subcase, we can upper bound the success probability for $\mathcal{A}$ to forge the signature of the ticket in the final step by $(q_{send} + q_{test}) \cdot 1/2^{\tau}$ due to the use of a secure signature scheme. Since this involves an action of $\mathcal{A}$, the number of passive queries q_{exe} is not relevant.

Case 2. To get an old ticket from a previous session accepted by S^j, the contained value h_S in the ticket must be a valid hash value for (ID_{S^j}, N_{S^j}), where N_{S^j} denotes the nonce issued by S^j in the first step of the current protocol run. Since H is modelled as a random oracle with 2τ-bit hash values, the probability of a collision between multiple hash values h_S and $h'_{S'}$ is at most $(q_{exe} + q_{send} + q_{test})^2 \cdot 1/(2 \cdot 2^{2\tau})$. Since $\mathcal{A}$ asks only a polynomial number of queries q (which assumably is less than 2^{τ}), this can be upper bounded by $q \cdot 1/2^{\tau}$.

Case 3. To obtain a fresh valid ticket, $\mathcal{A}$ must achieve at least one of the following:

- to capture a valid ticket for a parallel session by playing S^j in the view of P^i,
- to forge the signature of PT^i for a message to T in Step (5),
- to make P^i ask T for the ticket for S^j by silently replacing ID_{S^j}, N_{S^j}, N_T in the message to PT^i in Step (4) so P will not notice the manipulation.

The use of TLS prevents that $\mathcal{A}$ can act as S^j. Hence, the success probability for $\mathcal{A}$ in the first subcase is 0. In the second subcase, the probability of $\mathcal{A}$ to forge the signature of a response of an honest prover PT^i in Step (4) is at most $q_{send} \cdot 1/2^{\tau}$ since we assume a secure signature scheme.

It remains to bound the success probability for replacing the message from T over PS^i to PT^i in Steps (3) and (4) such that the prover will *not* notice the replacement and will not abort the current run of the protocol. Since the message is signed by T, $\mathcal{A}$ can forge a new message with probability at most $1/2^{\tau}$; replaying an old message with a nonce other than that for the current session will be noticed by S^j at the end, and the success probability is 0 then. Further, since the channel between S^j and PS^i is one-sided authenticated by TLS, $\mathcal{A}$ can not replace the message with an $ID_{S'}$ of a different service provider S' that may be under control of $\mathcal{A}$. So, $\mathcal{A}$ can not impersonate S^j without the prover noticing it. So, if $\mathcal{A}$ replaces h_{S^j}, ID_{S^j}, and/or N_{S^j} to log-in as the prover in a parallel session of a different service provider, the prover instance will abort its run of the protocol the success probability for $\mathcal{A}$ is also 0 in this case. Hence, the success probability is limited to $q_{send} \cdot \max\{1/2^{\tau}, 0, 0\} = q_{send} \cdot 1/2^{\tau}$ for the third case. Summing up the terms from all cases yields that

$$\Pr[\mathsf{Succ}_{\mathsf{Auth}}] \leq \frac{q_{send} + q_{test}}{2^{\tau}} + \frac{q}{2^{\tau}} + \left(0 + \frac{q_{send}}{2^{\tau}} + \frac{q_{send}}{2^{\tau}}\right) \leq \frac{4q}{2^{\tau}},$$

which gives our claim in Theorem 1.

B Anonymity Analysis

This section provides our proof of Theorem 2.

Theorem 2 (Anonymity). *Let the employed public-key signature scheme be EUF-CMA-secure and H be a random oracle. Then, for any PPT adversary $\mathcal{A}$ whose run time is bounded by t and which asks at most q_{exe} execute and q_{send} send queries, respectively, it holds for a random execution of $\mathcal{G}^{Anon}$ on our protocol $\mathbb{P}$:*

$$\boldsymbol{Adv}_{\mathbb{P}}^{Anon}(\mathcal{A}) \leq (q_{exe} + q_{send}) \cdot 1/2^{2\tau}.$$

Proof. From the setup, execute, and send queries, $\mathcal{A}$ can learn the following relations between provers and service providers:

- From an execution of the registration protocol between a prover P^i and T, the adversary can learn the relation between ID_{PT^i}, $K^p_{PT^i}$, and ID_{PM^i}.
- From an execution of the activation protocol between a prover P^i, service provider S^j, and T, $\mathcal{A}$ can learn the relation of h_{S^j} to ID_{PT^i} and $h^j_{PT^i}$.
- From an execution of the authentication protocol between P^i, S^j, and T, the adversary can learn the relation between ID_{PT^i}, $h^j_{PT^i}$, and $h'_{S^j} = H(ID_{S^j} \| N_{S^j})$.
- The key-revocation protocol recalls the first three steps of the authentication protocol; so, from an execution between P^i, S^j, and T, the adversary can observe again ID_{PT^i}, $h^j_{PT^i}$, and a fresh value h''_{S^j}.
- The rekeying protocol provides $\mathcal{A}$ with no information about the relation between provers and service providers.

From the above, we can see that the only information about service providers is contained in the blinded hash values $h_{ID^{\mathbb{S}}}$, $h'_{ID^{\mathbb{S}}}$, $h''_{ID^{\mathbb{S}}}$. So, the only way for $\mathcal{A}$ can only determine if P interacts with S^0 or S^1 is to find a value N such that $H(ID_{S^b} \| N)$ is a preimage to any of the blinded hashes. In the protocols, the hash values have been computed using a fresh unpredictable nonce each, for which we assume that they have been sampled uniformly at random from $\{0,1\}^{2\tau}$ and that H is a random oracle. In each execution of a protocol, $\mathcal{A}$ observes at most one hash value. So, we can upper bound the advantage for $\mathcal{A}$ by $(q_{exe} + q_{send}) \cdot 1/2^{2\tau}$, which gives our bound in Theorem 2.

Remarks. Clearly, a malicious trusted third party could – like any other adversary – try to guess the secret of P for the first-factor at $\widehat{S}$. In case of success, $\mathcal{A}$ could try to log-in as P at both service providers S^0 and S^1 and create the authentication ticket for the second factor itself. Since P is registered at only one of $\{S^0, S^1\}$, $\mathcal{A}$ would be able to easily determine b and win the anonymity game. Though, the security of the first factor is not a concern covered by a protocol that focuses on the security of the second factor.

Unlinkability Analysis

This section provides our proof of Theorem 3.

Theorem 3 (Unlinkability). *Let the employed public-key signature scheme be EUF-CMA-secure and H be a random oracle. Then, for any PPT adversary $\mathcal{A}$ whose run time is bounded by t and which asks at most q_{exe} execute and q_{send} send queries, it holds for a random execution of $\mathcal{G}^{Unlink}$ on our protocol $\mathbb{P}$:*

$$\boldsymbol{Adv}_{\mathbb{P}}^{Unlink}(\mathcal{A}) \leq (q_{exe} + q_{send}) \cdot 1/2^{2\tau}.$$

Proof. From the setup, execute, and send queries, $\mathcal{A}$ can learn the following relations between provers and service providers:

- From an execution of the activation protocol with a prover P^i, $\mathcal{A}$, and T, the adversary chooses a fresh nonce N_{S^j} and can learn the relation between $h^j_{PT^i}$ and the chosen values $h_{S^j} = H(ID_{S^j} \| N_{S^j})$.
- From a correct execution of the authentication or key-revocation protocol with a prover P^i, $\mathcal{A}$, and T, the adversary can learn only $h^j_{PT^i}$ (as after an execution of the activation protocol) as information about P^i.
- Executions of registration and rekeying protocols give $\mathcal{A}$ no information about the relation of provers to service providers.

From the above, we can see that the only information about provers that is visible to $\mathcal{A}$ in our protocols is contained in the blinded hash values $h^j_{\widehat{P}} = H(ID_{PT^b} \| N_T)$. So, under the assumption that provers are privacy-aware (use distinct credentials for their first factor, and blind their browser/OS configuration, IP, etc.), the only way for colluding service providers to link common provers is to find a preimage (ID_{PT^b}, N_T) for $h_{\widehat{P}}$. Though, by definition, the hash values h_{PT^b} were generated from a 2τ-bit nonce N_T that was chosen uniformly at random from $\{0,1\}^{2\tau}$. Since H is modeled as a random oracle, $\mathcal{A}$'s advantage is at most $(q_{exe} + q_{send}) \cdot 1/2^{2\tau}$, which gives Theorem 3.

C Automatic Security Analysis

Besides our formal security analysis, we also conducted an automatic security analysis of our authentication scheme using the well-known computer-aided proof system AVISPA. After a brief overview of AVISPA's capabilities, we describe the HLPSL implementations of our protocols and the results obtained from feeding them into AVISPA. Moreover, we conduct experiments by deliberately removing security features from our protocols and observing the results from the proof system.

Background

AVISPA provides four backends for protocol verification: a Constraint-Logic-based ATtack SEarcher (CL-ATSE) [42], an On-the-Fly Model Checker (OFMC) [7], a SAT-based Model Checker (SAT-MC) [5], and a Tree-Automata-based backend (TA4SP) [10]. We rely on the widespread CL-ATSE, OFMC, and SAT-MC backends; TA4SP does not support our setup. As input to AVISPA, protocols must be implemented in the High-Level Protocol Specification Language (HLPSL) [13]. HLPSL is a role-centric language that is well-suited for software engineers and protocol designers.

Implementation Details

All protocols of our authentication scheme have been implemented in HLPSL. Appendix D shows the implementation of the authentication protocol (see Table 3 for comparison). The full source code of all our protocols will be provided to the public domain. Four roles are implemented, one for each agent that takes part in the protocol, namely the service provider S, the prover's browser PS, the trusted third party T, and the prover's phone authenticator PT. Each role implements the role- specific steps of

Table 7. Results from AVISPA when omitting TLS in individual protocols. A • indicates that TLS is mandatory to uphold security, and a ◦ that TLS is optional.

Protocol	**Communication step**						
	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Registration	•	n/a	n/a	•	•	n/a	
Activation	•	•	•	n/a	◦	•	•
Authentication	•	•	•	n/a	◦	•	•
Key Revocation	•	•	•	•	•		
Rekeying	n/a	•	•	n/a			

the authentication protocol, whereas the step numbers correspond with those of the protocol's formal specification of Table 3. For further consistency and where the syntax would allow it, variable names have been chosen to correspond with those used in the formal specification as well. The two communication channels send (SND) and receive (RCV) are defined in terms of the Dolev-Yao model (dy).

Since our protocols make use of TLS, this has to be reflected in our HLPSL implementation. However, at present, neither AVISPA nor HLPSL support modularization of protocol implementations, so that the implementation of the TLS protocol in HLPSL cannot be invoked from ours. When mixing both protocol implementations into one file, this severely affects legibility. Therefore, for simplicity, we model TLS by means of public keys assigned to each role, which ensure both encryption and sender authenticity. This approach is sound and has been applied similarly in several other high-level protocol implementations built on top of TLS.

Results and Experiments

We fed each protocol's HLPSL implementation to AVISPA and found that all of the aforementioned backends report that they cannot identify any attacks. However, since implementations can be erroneous and since there is currently no standardized unit-testing framework for HLPSL protocol implementations, we conduct experiments and sanity checks in order to verify that our implementation meets our expectations from the manual security analysis.

As a first experiment, we changed each protocol's implementation in a deliberate attempt to make it insecure. The flaws introduced include the removal of TLS for data-origin authentication, signatures, and nonces which opened various attack vectors. We then fed the flawed versions to AVISPA in order to check whether it picks up the vulnerabilities. Without fail, AVISPA identified them. This experiment serves to raise confidence both that the authentication scheme comprises little redundancy and that our implementation reflects well our scheme's formal specification.

We were particularly interested whether and to what extent TLS is required to secure our protocols. We employ TLS mainly as a means for data-origin authentication, whereas message encryption is optional. Since TLS is the de facto standard in secure web communications, using a different protocol would severely limit the applicability and acceptance of our authentication scheme in practice. We conducted a second experiment wherein we systematically disabled TLS in a given step of a protocol, re-running AVISPA each time to identify potential attacks that result from doing so. Table 7 summarizes the results for each protocol. As expected, turning TLS off allows for man-in-the-middle attacks that result from the missing data-origin authentication.

D AVISPA Code for the Authentication Protocol

HLPSL implementation of the authentication protocol:

```
role service(S,PS,T,PT : agent,
             SND, RCV : channel(dy),
             Ks, Kps : public_key,
             H : hash_func,
             Hpt : text)
  played_by S def=
  local
    Step : nat,
    Ns : text,
    Hs : hash(text.agent)
  init Step := 0
  transition
    % Start the protocol, create a new session nonce and
    % send it with Hpt to P
    0. Step  = 0 /\ RCV(start) =|>
       Step':= 7 /\ Ns' := new()
                 /\ SND({{S.Hpt.Ns'}_S}_Kps)
                 /\ Hs' := H(Ns'.S)
                 /\ witness(S,PT,nsSPT,Ns')
                 /\ witness(S,PT,hptSPT,Hpt)
    % receive the authentication ticket -> protocol end
    7. Step  = 7 /\ RCV({{{T.Hpt.Hs}_T}_PS}_Ks) =|>
       Step':= 8 /\ request(S,PS,hsPSS,Hs)
end role

role browser(S,PS,T,PT : agent,
             SND, RCV : channel(dy),
             Ks, Kps, Kt, Kpt : public_key,
             H : hash_func)
  played_by PS def=
  local
    Step : nat,
    Hpt, Ns, Nt : text,
    Hs : hash(text.agent)
  init Step := 1
  transition
    % Receive session nonce and hpt, blind it and
    % send it to T
    1. Step  = 1 /\ RCV({{S.Hpt'.Ns'}_S}_Kps) =|>
       Step':= 3 /\ Hs' := H(Ns'.S)
                 /\ SND({{Hpt'.Hs'}_PS}_Kt)
                 /\ witness(PS,S,hsPSS,Hs')
    % Receive the challenge and display it for PT
    % (with additional information)
    3. Step  = 3 /\ RCV({{T.Hpt.Hs.Nt'}_T}_Kps) =|>
       Step':= 6 /\ SND({{{T.Hpt.Hs.Nt'}_T.Ns.S}_PS}_Kpt)
    % Receive the authentication ticket and
    % forward it to S
    6. Step  = 6 /\ RCV({{T.Hpt.Hs}_T}_Kps) =|>
       Step':= 8 /\ SND({{{T.Hpt.Hs}_T}_PS}_Ks)
end role

role ttp(S,PS,T,PT : agent,
         SND, RCV : channel(dy),
         Kps, Kt, Kpt : public_key,
         Hpt : text,
         H : hash_func)
  played_by T def=
  local
    Step : nat,
    Hs : hash(text.agent),
    Nt : text
  init Step := 2
  transition
    % Receive a new authentication request and
```

```
    % create the challenge
    2. Step  = 2 /\ RCV({{Hpt.Hs'}_PS}_Kt) =|>
       Step':= 5 /\ Nt' := new()
                 /\ SND({{T.Hpt.Hs'.Nt'}_T}_Kps)
                 /\ witness(T,PT,ntTPT,Nt')
    % Receive the challenge from PT (implicit check for
    % all values) and send a signed authentication ticket
    5. Step  = 5 /\ RCV({{PT.{T.Hpt.Hs.Nt}_T}_PT}_Kt) =|>
       Step':= 8 /\ SND({{T.Hpt.Hs}_T}_Kps)
end role

role smartphone(S,PS,T,PT : agent,
                SND, RCV : channel(dy),
                Kps, Kt, Kpt : public_key)
  played_by PT def=
  local
    Step : nat,
    Hpt, Nt, Ns : text,
    Hs : hash(text.agent)
  init Step := 4
  transition
    % Scan the challenge, check possible values,
    % sign it and send it back to T
    4. Step  = 4
             /\ RCV({{{T.Hpt'.Hs'.Nt'}_T.Ns'.S}_PS}_Kpt)
             =|>
       Step':= 8 /\ SND({{PT.{T.Hpt'.Hs'.Nt'}_T}_PT}_Kt)
                 /\ request(PT,S,nsSPT,Ns')
                 /\ request(PT,T,ntTPT,Nt')
                 /\ request(PT,S,hptSPT,Hpt')
end role

role session(S,PS,T,PT : agent,
             H : hash_func)
  def=
  local
    SS, RS, SPS, RPS, ST, RT, SPT, RPT : channel(dy)
  const
    % Initialize the keys for every session
    ks, kps, kt, kpt : public_key,
    % Simulate a valid hpt
    hpt : text
  composition
       service(S,PS,T,PT,SS,RS,ks,kps,H,hpt)
    /\ browser(S,PS,T,PT,SPS,RPS,ks,kps,kt,kpt,H)
    /\ ttp(S,PS,T,PT,ST,RT,kps,kt,kpt,hpt,H)
    /\ smartphone(S,PS,T,PT,SPT,RPT,kps,kt,kpt)
end role

role environment()
  def=
  const
    s,ps,t,pt : agent,
    h : hash_func,
    nsSPT, ntTPT, hsPSS, hptSPT : protocol_id
  intruder_knowledge = {s,ps,t,pt,h}
  composition
    session(s,ps,t,pt,h)
end role

goal
  authentication_on nsSPT
  authentication_on ntTPT
  authentication_on hsPSS
  authentication_on hptSPT
end goal

environment()
```