A preliminary version of this paper appears in the proceedings of the *1st International Conference on Security Standardisation Research (SSR) 2014*, available at link.springer.com, DOI: 10.1007/978-3-319-14054-4_11. The full version appears in the *International Journal of Information Security*, DOI: 10.1007/s10207-016-0348-7. This is the author's copy of the full version.

Secure modular password authentication for the web using channel bindings*

Mark Manulis[1] Douglas Stebila[2] Franziskus Kiefer[3]
Nick Denham[4]

[1] *Surrey Centre for Cyber Security, Department of Computer Science, University of Surrey, Guildford, UK*
mark@manulis.eu

[2] *McMaster University, Hamilton, Ontario, Canada*
stebilad@mcmaster.ca

[3] *Mozilla, Berlin, Germany*
mail@franziskuskiefer.de

[4] *Queensland University of Technology, Brisbane, Australia*

October 13, 2016

Abstract

Secure protocols for password-based user authentication are well-studied in the cryptographic literature but have failed to see wide-spread adoption on the Internet; most proposals to date require extensive modifications to the Transport Layer Security (TLS) protocol, making deployment challenging. Recently, a few modular designs have been proposed in which a cryptographically secure password-based mutual authentication protocol is run inside a confidential (but not necessarily authenticated) channel such as TLS; the password protocol is bound to the established channel to prevent active attacks. Such protocols are useful in practice for a variety of reasons: security no longer relies on users' ability to validate server certificates and can potentially be implemented with no modifications to the secure channel protocol library.

We provide a systematic study of such authentication protocols. Building on recent advances in modelling TLS, we give a formal definition of the intended security goal, which we call *password-authenticated and confidential channel establishment* (PACCE). We show generically that combining a secure channel protocol, such as TLS, with a password authentication or password authenticated key exchange protocol, where the two protocols are bound together using the transcript of the secure channel's handshake, the server's certificate, or the server's domain name, results in a secure PACCE protocol. Our prototypes based on TLS are available as a cross-platform client-side Firefox browser extension as well as an Android application and a server-side web application that can easily be installed on servers.

Keywords: password authentication; Transport Layer Security; channel binding

*MM and FK acknowledge support by the German Research Foundation (DFG), project PRIMAKE (MA 4957). DS acknowledges funding from Australian Research Council (ARC) Discovery Project DP130104304.

Contents

1 Introduction

Authentication using passwords is perhaps the most prominent and human-friendly user authentication mechanism widely deployed on the Web. In this ubiquitous approach, which we refer to as *HTML-forms-over-TLS*, the user's password is sent encrypted over an established server-authenticated Transport Layer Security (TLS, previously known as Secure Sockets Layer (SSL)) channel in response to a received HTML form. This approach is subject to many threats: the main problems with this technique are that security fully relies on a functional X.509 public key infrastructure (PKI) and on users correctly validating the server's X.509 certificate. In practice, these assumptions are unreliable due to a variety of reasons: the many reported problems with the trustworthiness of certification authorities (CAs), inadequate deployment of certificate revocation checking, ongoing threats from phishing attacks, and the poor ability of the users to understand and validate certificates [39, 40]. Hypertext Transport Protocol (HTTP) basic and digest access authentication [22] has been standardized, and digest authentication offers limited protection for passwords, but usage is rare. Public-key authentication of users, e.g. using X.509 certificates, is also rare. This work proposes a novel way of performing cryptographically secure mutual password-based authentication on the internet.

1.1 Password-authenticated key exchange

Password-authenticated key exchange (PAKE) protocols, which were introduced by Bellovin and Merritt [11], and the security of which was formalized in several settings [10, 15, 17, 3], could mitigate many of the risks of the HTML-forms-over-TLS approach as they do not rely on any PKI and offer stronger protection for client passwords against server impersonation attacks, such as phishing. PAKE protocols allow two parties to determine whether they both know a particular string while cryptographically hiding any information about the string. They are resistant to offline-dictionary attacks: an adversary who observes or participates in the protocol cannot test many passwords against the transcript. Successful execution of a PAKE protocol also provides parties with secure session keys which can be used for encryption.

Despite the many benefits and the presence of a variety of PAKE protocols in the academic literature and in standards [26, 27, 25], PAKE-based approaches for client authentication have not been adopted in practice. There is no PAKE standard that has been agreed upon and implemented in existing web browser and server technologies. This is due to several practical obstacles, including: patents covering PAKE in general (some of which have recently expired in the US) and PAKE standards such as the Secure Remote Password (SRP) protocol [42], lack of agreement on the appropriate layer within the networking stack for the integration of PAKE [20], complexity of backwards-compatible deployment with TLS, and user-interface challenges.

There have been a few proposals to integrate PAKE into TLS by adding password-based ciphersuites as an alternative to public-key authenticated ciphersuites. For instance, SRP has been standardized as a TLS ciphersuite [41] and has several reference implementations but none in major web browsers or servers. Abdalla et al. [1] proposed the provably secure Simple Open Key Exchange (SOKE) ciphersuite, which uses a variant of the PAKE protocol from [5] that is part of the IEEE-P1363.2 standard [25]. The J-PAKE protocol [24] is used in a few custom applications. Common to all PAKE ciphersuite approaches is that the execution of PAKE becomes part of the TLS handshake protocol: the key output by PAKE is treated as the TLS pre-master secret, which is then used to derive further encryption keys according to the TLS specification. An advantage of this approach is that secure password authentication could subsequently be used in any application that makes use of TLS, and that standard TLS mechanisms for key derivation and

secure record-layer communication can continue to be used. However, a major disadvantage is that any new ciphersuites in TLS require substantial vendor-side modifications of the web browser and server software. This is problematic for modern web server application architectures within large organizations, where a TLS accelerator immediately handles the TLS handshake and encryption, then hands the plaintext off to the first of many application servers; requiring the TLS accelerator to have access to the list of valid usernames and passwords may mean a substantial re-architecting. Moreover, using solely PAKE in TLS means abandoning the web public key infrastructure.

1.2 Running PAKE at the application layer

A better approach for realizing secure password-based authentication on the web may be to rely on existing TLS implementations to provide confidential communication between clients and servers, and integrate application-level PAKE for password-based authentication, without requiring proposing any new TLS ciphersuites or changing any of the steps of TLS handshake specification.

However, if the TLS channel is only assumed to provide confidentiality, not authentication, then one must use an alternative mechanism to rule out man-in-the-middle attacks on the TLS channel. Since it is the password-based protocol that provides mutual authentication, there should be a binding between the TLS channel and the password-based protocol. There are several potential values which might be used for binding: the transcript of the TLS handshake protocol, the TLS master secret key (or a value derived from it, such as the TLS `Finished` message), the server's certificate, or even the server's domain name. A recent standard [9] describes three TLS channel bindings, two of which are relevant to us: `tls-unique` in which the binding string is the `Finished` message, and `tls-server-end-point` in which the binding string is the hash of the server's certificate.

Notably, TLS channel bindings do not change the TLS protocol itself: all TLS protocol messages, ciphersuites, data transmitted, and all other values are entirely unchanged. Rather, TLS channel bindings expose an additional value to the application that can be obtained locally, thereby requiring minimal changes to TLS implementations. Using the domain name rather than a property of the TLS channel allows performing the authentication prior to establishment of the TLS channel.

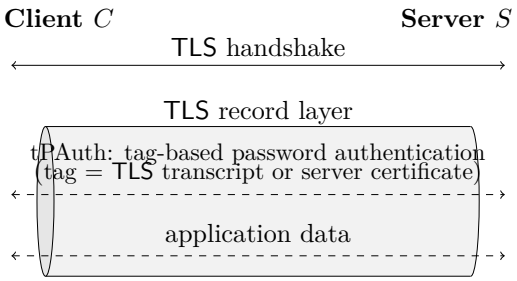
Several recent works have proposed protocols of this form, running PAKE on the application layer. Oiwa et al. [35, 36, 37, 7] published an Internet-Draft that employs an ISO-standardized PAKE protocol (KAM3 [26, §6.3],[32]) and binds it to the TLS channel using either the server's certificate or the TLS master secret key, but no formal justification is given for security of the combined construction. Dacosta et al. [18] proposed the DVCert protocol which aims to achieve direct validation of the TLS server certificates by using a modification of the protocol from [15] for secure server-to-client password-based authentication.

1.3 Contributions

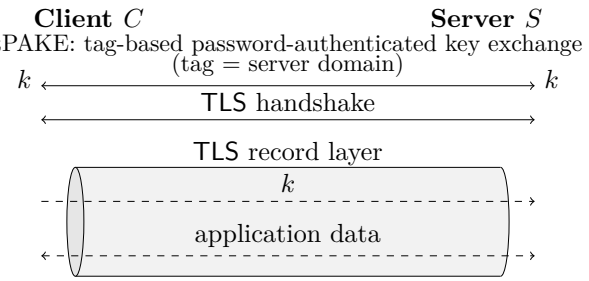
We propose and analyze three modular constructions for smooth integration of PAKE functionality with secure channels such as TLS without requiring any modification to the original channel protocol specification, nor requiring abandoning public key server certificates.

This is achieved through a black-box combination of a secure channel establishment protocol with a password-only authentication or key exchange protocol as illustrated in Figure 1. The latter is bound to the channel establishment phase in order to detect and prevent any man-in-the-middle attacks. This binding is achieved using *tag-based* versions of password protocols, which in addition to mutual knowledge of the password ensure the equality of (possibly public or adversary-controlled) tags that serve as additional input to the protocol by both parties.

In our constructions 1 and 2, a secure TLS channel is established first with no assumptions



(a) Constructions 1 and 2: Tunnelling a tag-based password authentication protocol through a secure channel.



(b) Construction 3: Establishing a tag-based password-authenticated session key k and sending that through the secure channel.

Figure 1: High-level approaches for combining password authentication with a TLS channel to establish a single password-authenticated secure channel.

---

on correct validation of certificates; then a tPAuth protocol is run *within* the TLS channel to demonstrate mutual knowledge of the password and absence of a man-in-the-middle attack on the TLS channel. Once tPAuth protocol succeeds, the parties continue with the exchange of the application data using the established TLS session keys. Note that tPAuth is an authentication-only protocol and not a full-fledged key exchange protocol, i.e. tPAuth does not need to output any keys. In construction 1 tPAuth execution is bound to the TLS channel using TLS transcript whereas in construction 2 this binding is performed using TLS server certificate. In construction 3 we replace tPAuth with a tPAKE protocol (which additionally outputs its own session key) and let parties executes a tPAKE session *before* establishing the TLS channel. However, before parties proceed with secure communication using TLS session keys the computed tPAKE session key is transmitted from the client to the server through the channel. This step completes mutual authentication and guarantees the absence of a man-in-the-middle attack on the TLS channel.

We observe that constructions 1 and 2 require modifications within the client's user agent, while construction 3 can be implemented without changing the client's user agent but therefore needs assumptions on a functioning domain name system (DNS) and PKI. Although this is obviously a drawback compared to the other two constructions it still allows a binding between the confidential channel and password-based authentication while preventing man-in-the-middle attacks, which is a significant improvement over HTML-forms-over-TLS. Thus, while constructions 1 and 2 offer strong security guarantees with minimal assumptions, construction 3 is more flexible in its deployment but requires more assumptions. This can also be seen in Section 8 where construction 2 is implemented as Firefox extension (i.e. integrated in the client's user agent) and construction 3 as Android application (i.e. indepently from the browser).

Since the most suitable definition for the security of the combined TLS handshake and record layer protocols is the *authenticated and confidential channel establishment (ACCE)* model of Jager et al. [29], in Section 3 we define a corresponding *password-based ACCE (PACCE)* model and apply it to the analysis of our constructions. Since ACCE has been used to analyze many real-world protocols (e.g. SSH [12], EMV [16]), our PACCE model would also be suitable for strong password-based variants of those protocols. We prove that all our constructions are secure PACCE protocols. In our analysis we generalise the underlying channel protocols that are used as building blocks to possess the properties of an ACCE or an unauthenticated *confidential channel establishment (CCE)*. In this way we show that our approach for password-based authentication of secure channels is applicable to a wider class of channel establishment protocols.

Applicability of results to other approaches. Our results, by employing the CCE/ACCE frameworks, are generic and could be applicable to constructions employing a wide variety of protocols, beyond the TLS. For example, our results justify the *general approach* behind the proposals by Oiwa et al.' [35, 36, 37, 7] and by Dacosta et al. [18]. We caution that our theorems, which make use of security models for tPAuth and tPAKE, do not immediately imply security of those particular protocols.

Oiwa et al. adopt three channel binding mechanisms: server's TLS certificate, TLS master secret key, and for non TLS-based connections server's host string (e.g., `http://www.example.com:80`). Our results on the use of TLS server certificate show that the underlying PAKE protocol must possess tPAuth properties. This requirement also applies to the use of TLS `Finished` message. Since the TLS handshake will not complete unless both parties compute the same `Finished` messages, our results indirectly justify the binding based on the TLS master secret key.

Dacosta et al.'s protocol [18] only provides server-to-client password-based authentication, whereas our PACCE constructions aim at mutual authentication of both parties.

Reference implementation In Section 8, we describe our reference implementations which are available for immediate download:

Our first client side implementation is a Firefox *extension*: it is a cross-platform Javascript-based bundle that can be installed by the user at run-time, without any modifications to the source code of the Firefox browser or its TLS library, Network Security Services (NSS). This implementation realizes the channel binding based on server's TLS certificate [9] that can be accessed through Firefox API. The Javascript-based implementation provides tolerable performance, with total round-trip time of around 300ms on a laptop, including network latency. If realized in native C using OpenSSL libraries the total protocol execution time can be reduced further to 180ms (at the cost of being non cross-platform).

Our second client side implementation is an Android application, which can be directly installed by the user from the Google Play store. This implementation binds to the server's domain name and requires less than 2.5 seconds for the protocol to complete.

On the server side, our implementation is achieved entirely as a cross-platform PHP application, which can be added at run-time without any modifications to the source code of the Apache web server or its TLS library, OpenSSL.

2 Approach

Our first general approach for secure modular password-based authentication on the web that is realized in constructions 1 and 2 is as follows:

1. Establish a secure (TLS) channel as normal in web browsing.
2. Use a tag-based password authentication (tPAuth) protocol to perform secure authentication based on *mutual knowledge of the shared password*, and *mutual agreement on a (possibly public) tag*, which binds the tPAuth protocol to the secure channel; the tag is either (i) the transcript of the secure channel establishment (for TLS, the `Finished` message from the handshake, which contains a hash of the transcript of the TLS handshake), or (ii) the (hash of the) server's certificate.

Our second general approach for modular password-based authentication on the web that is realized in construction 3 is as follows:

1. Establish a common session key between the client and the server using a tag-based password authenticated key exchange (tPAKE) protocol with mutual authentication, using the server's

domain name as tag to bind the session key to this domain.

2. Establish a secure (TLS) channel as normal in web browsing and send the session key from the client to the server. The session key is used to authenticate the client.

From a theoretical perspective, we want to ensure that the combination of tPAuth/tPAKE and TLS protocols is secure—in particular, that the password authentication is really bound to the channel, so that an adversary cannot perform a man-in-the-middle attack on the channel. We will provide formal justifications for generic constructions using three different channel binding strings: transcripts (construction 1), public keys (construction 2), and domain names (construction 3).

For a practical implementation involving TLS, one also needs to specify the format of messages and how they are delivered. The messages can be delivered in any suitable format or medium: as HTTP authentication headers, as a micro-format within HTML, or as appropriate in other application-layer protocols. In particular, no modification of TLS is required, beyond support in the application for obtaining TLS channel binding information, which is already partially supported by some web browsers and web servers.

In the remainder of this section, we will give an overview of the theoretical building blocks, and their corresponding practical realizations, then discuss how these building blocks are combined to construct our main generic protocols.

2.1 Building blocks

2.1.1 Channel establishment protocols

Since TLS provides both key establishment and secure communications, it does not suffice to model it as just an authenticated key exchange protocol; recent work by Jager et al. [29] instead models TLS as an *authenticated and confidential channel establishment (ACCE) protocol*. Here, there are two stages to a protocol: a *pre-accept* stage, which corresponds to the TLS handshake protocol, in which two parties establish a shared session key, and a *post-accept* stage, which corresponds to the TLS record layer protocol, in which they use the established session key to provide confidentiality and integrity of communications using authenticated encryption. ACCE is quickly becoming the accepted security definition for TLS, and recent work [31, 30] has shown that many TLS ciphersuites are ACCE-secure, including RSA-key-transport and signed-Diffie–Hellman ciphersuites; and that ACCE can be suitably modified for modelling TLS renegotiation [23]. These results of course depend on the implementation avoiding flaws (such as the state machine attack [13]), and also assume strong cryptographic primitives, such as strong RSA or Diffie–Hellman parameters (which can be sidestepped in some settings with attacks like FREAK [13] and Logjam [6]).

In ACCE, parties are authenticated to each other based on long-term public keys, either mutually or server-only. As with most models for authenticated key exchange, it is abstractly assumed that these long-term public keys are distributed in an authentic way, and that parties always correctly map public keys to the intended communication partner. Since in our approach authentication will come from mutual knowledge of a password, we can put aside the authentication aspects of ACCE to derive the weaker notion of a *confidential channel establishment (CCE) protocol*, in which confidentiality (and integrity) of the established channel is guaranteed only for sessions in which the adversary was passive during the handshake phase. CCE provides no entity authentication guarantees.

Of course, every ACCE protocol is also a CCE protocol when we lift the authentication requirement on the ACCE protocol. This corresponds with how we will use TLS in our construction. TLS does, when public keys are managed and used properly, provide strong authentication based on public keys, and (certain ciphersuites) can be proven to be ACCE-secure. But, as we observed

in the introduction, practice suggests we cannot rely on the web PKI to provide ideal authentic distribution and mapping of public keys to identities. Thus, TLS can in practice be seen as a CCE protocol: even though long-term public keys may be used in TLS, we are not confident in their distribution, and thus we only take TLS to provide CCE, rather than ACCE, security. A formal definition of CCE security appears in Section 4.1.

2.1.2 Tag-based password authentication and key- exchange

PAKE protocols provide secure mutual authentication based on knowledge of a shared secret password *and* establish a shared secret key that can be used for encryption. In our first two constructions, we already have a session key from the secure channel, so we only need a password authentication (PAuth) protocol, not a full PAKE protocol (although little computational effort is saved with just PAuth, as the public key (typically Diffie–Hellman) operations that prevent offline dictionary attacks are still required). In construction three however we consider a full PAKE protocol that provides us with a session key that is then used to authenticate the client.

We require *tag-based password authentication protocol (tPAuth)* and *tag-based password authenticated key-exchange protocol (tPAKE)*, which will provide secure mutual authentication (resistant to offline-dictionary attacks) based on knowledge of a shared secret password, and with acceptance only if both parties use the same, possibly public, auxiliary tag [28, 21].

The formal definition of tPAuth and tPAKE security appears in Section 4.2 and 6.1, respectively. In our reference implementation we use tSOKE a tagged version of the Simple Open Key Exchange (SOKE) protocol [1] that is described in Section 7; note that any tPAuth or tPAKE protocol could be used in our generic constructions, and any PAKE protocol can in fact be transformed into a tPAuth or tPAKE protocol by hashing the original password with the tag and using the result as a new password [21].

2.1.3 TLS channel bindings

To securely bind the password authentication protocol and the used secure channel, we must incorporate some identifier for the channel into the authentication protocol.

It is not enough to bind to just random nonces, for example, as a man-in-the-middle attacker can relay those. Conventional cryptographic wisdom recommends that using the full transcript of a protocol suffices for identifying the channel in a binding way. However, since our goal is to accommodate practical scenarios in which the ideal cryptographic techniques cannot always be used, we must consider what other mechanisms are available.

Channel bindings for TLS [9] are a standardized mechanism for retrieving information from a TLS connection that can be used to identify the connection. Three mechanisms are provided, two of which are relevant to us:

- `tls-unique`: The binding string is "the first [(plaintext)] TLS `Finished` message sent in the most recent TLS handshake of the TLS connection being bound to", [9, §3.1], which corresponds to the client's `Finished` message to the server. This is computed as $\mathsf{PRF}(ms, \text{“}\texttt{client finished}\text{”} \| H(T))$ where PRF is the TLS pseudorandom function, ms is the TLS *master secret* from which the session keys are derived, H is a cryptographic hash function, and T is the transcript of the TLS handshake messages up to this point, namely from `ClientHello` up to (but not including) `ChangeCipherSpec` (excluding any `Hello` and non-handshake messages).
- `tls-server-end-point`: The binding string is the hash of the TLS server's certificate.

Notably, TLS channel bindings do not change the TLS protocol itself: all TLS protocol messages, ciphersuites, data transmitted, and all other values are entirely unchanged. Rather, TLS

channel bindings expose an additional value to the application that can be obtained locally, thereby requiring minimal changes to TLS implementations. `tls-unique` channel binding works with all TLS ciphersuites, whereas `tls-server-end-point` only works with TLS ciphersuites that employ certificate-based server authentication, though these are most widely used in practice.

`tls-server-end-point` may be easier to deploy on the server side since the server certificate is often fixed for long periods, and thus more suitable for multi-server architectures where for example an SSL accelerator handles the TLS connection and then passes the plaintext onto one of potentially many layers of application servers. `tls-server-end-point` is also easily deployable on the client side: for example, the Firefox extension API already makes the server certificate, but not the `Finished` message, available.

We will see that in some sense `tls-unique` is a stronger channel binding string, as when it is used we can achieve security of our generic construction using only CCE security of the TLS channel, whereas when `tls-server-end-point` is used we rely on the stronger ACCE security notion of TLS; in the end, both allow us to achieve our goal.

TLS keying material exporters [38] are another option for binding to the TLS channel, as they allow an application to obtain keying material derived from the master secret key for a given label. However, TLS channel bindings appear to be the preferred mechanism, and so we focus on them.

For our third construction we deploy a different channel binding mechanism using the server's domain name. Assuming a properly functioning DNS and PKI this approach is practically equivalent to the `tls-server-end-point` binding but does not require a TLS session to be established.

The Triple Handshake Attack [14] shows that TLS channel binding via `tls-unique` and `tls-server-end-point` are not secure when session resumption is allowed, since an attacker can cause two distinct sessions to have the same `tls-unique` binding value via resumption. In order for the techniques proposed in Sections 4 and 5 to be secure when instantiated with TLS channel bindings, we must either disable session resumption or adopt the mitigations recommended in [14, §7]. Note that this work does not deal with security issues stemming from TLS implementation errors or attacks on TLS outside the security model introduced for ACCE in [29].

2.2 Three PACCE constructions

Our first generic PACCE protocol between a client C and a server S from a tag-based password-based authentication protocol and a secure channel protocol is illustrated in Figure 1a. First, the channel establishment protocol (CCE or ACCE) is run until it accepts. Then, using the established channel, the two parties run a tPAuth protocol where the tag serves as a binding value to the established channel; when the tPAuth protocol accepts, then the parties accept in the overall PACCE protocol, and can continue using this channel for secure communication. Our first two constructions in Sections 4 and 5 differ only in the way the tPAuth is bound to the established channel: the tag is either the transcript of the channel protocol or the long-term public key of the server.

Our second generic PACCE protocol between a client C and a server S from a tag-based password authenticated key exchange protocol and a secure channel protocol is illustrated in Figure 1b. First, the tPAKE protocol is run to generate a common session key k using the server's domain name as a tag. Then, after establishing a secure channel (ACCE), the client C sends the session key k to the server S to authenticate the established channel.

We now describe the combination of a tPAuth protocol with TLS for constructions 1 and 2, as detailed in Figure 2. Client C and server S first establish a standard TLS channel: that is, they execute a normal TLS handshake, then exchange `ChangeCipherSpec` messages to start authenticated encryption within the TLS record layer, and then exchange their `Finished` messages

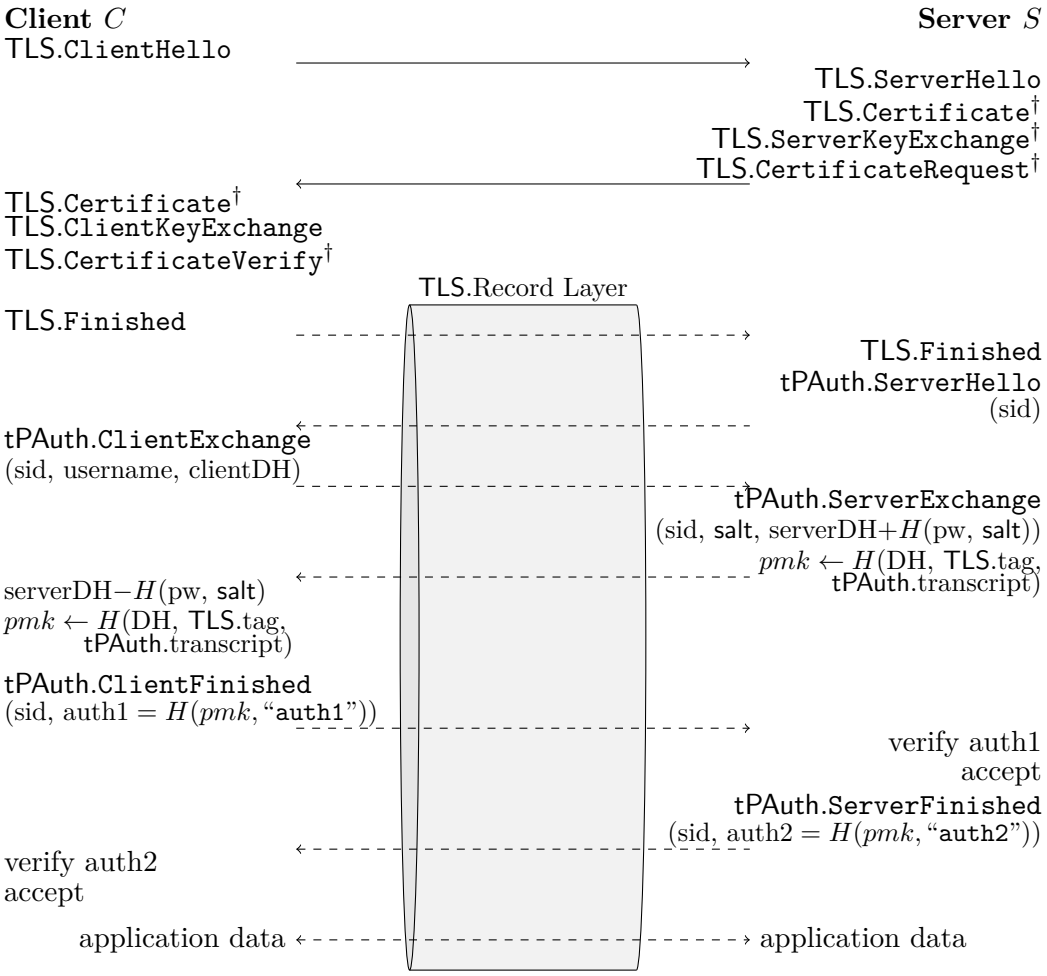


Figure 2: Constructions 1 and 2 – Protocol message diagram for TLS with tunnelled tPAuth protocol (using example messages of the tSOKE protocol in parentheses, where H is a hash function). TLS.tag is either TLS.Finished or TLS.Certificate. † denotes optional messages (the server's TLS.Certificate is necessary if used as TLS.tag).

---

for explicit key confirmation. Once Finished messages are successfully exchanged, the parties continue using the authenticated encryption mechanism of the TLS record layer to communicate messages of the tag-based password-authentication protocol tPAuth. In our first construction this binding is achieved by using a Finished message as the tag; note that Finished messages depend on the (hash of the) entire TLS handshake transcript. In our second construction the tag is the server's certificate (which includes the server's public key) that was communicated by S in its Certificate message of the TLS handshake. Upon successful completion of the password authentication phase both parties continue using session keys and authenticated encryption mechanism of the established TLS channel for secure communication. This construction is realized as Firefox extension (cf. Section 8.1).

Construction 3 executes a tPAKE protocol with the tag being the server's domain name before establishing the TLS channel over which the session key computed in tPAKE is transmitted for binding purposes. This construction is described in Figure 6 (cf. Section 8.2) from the perspective

of our implementation for mobile browsers. Due to limitations of mobile browsers and easier deployment the client is split into browser and application. We require a tPAKE protocol (rather than tPAuth) in this construction, which is executed between the mobile application and the server, in order to receive a session key, which can then be used to authenticate the TLS session between the mobile browser and the server. This construction further requires a working public key infrastructure that binds the domain name to the public key, used for the TLS session.

In the following sections, we will show that the aforementioned generic constructions, and hence their concrete TLS-based instantiations from Figure 2 and Figure 6, provably lead to the establishment of a secure password-based authenticated and confidential channel (PACCE).

3 Password-authenticated confidential channels

The security goal for our main construction is that it be a secure *password-authenticated and confidential channel establishment (PACCE)* protocol, which is a new password-based variant of the ACCE model of Jager et al. [29]. ACCE seems to be the most suitable for describing the security requirements of real-world secure channel protocols such as TLS [29, 31, 30, 23] and SSH [12], and so it is natural to adapt it to the password setting.

A PACCE protocol is a two-party protocol that proceeds in two stages: in the *handshake* stage both participants perform an initial cryptographic handshake to establish session keys which are then used in the *record layer* stage to authenticate and encrypt the transmitted session data.[1] At some time during execution, the parties may accept the session as being legitimately authenticated, or reject. The main difference in PACCE compared to the original ACCE model is the use of passwords instead of long-term public keys for authentication.

At a high level, a PACCE protocol is secure if the adversary cannot break authentication, meaning it cannot cause a party to accept without having interacted with its intended partner, and cannot break the confidential channel, meaning it cannot read or inject ciphertexts.

We consider the standard client-server communication model where a party is either a *client* C or a *server* S. For each client-server pair (C, S) there exists a corresponding password $\mathsf{pw}_{C,S}$ drawn from a dictionary $\mathcal{D}$.

An instance of party $U \in \{C, S\}$ in a session s is denoted as Π^s_U. Each instance Π^s_U records several variables:

- $\Pi^s_U.\mathsf{pid}$: the *partner identity* with which Π^s_U believes to be interacting in the protocol session
- $\Pi^s_U.\rho \in \{\mathtt{init}, \mathtt{resp}\}$: the *role* of this instance in the session, either initiator or responder. $\mathtt{init}(\Pi^s_U)$ and $\mathtt{resp}(\Pi^s_U)$ denote Π^s_U's view of who the initiator and responder are in the session, namely $(U, \Pi^s_U.\mathsf{pid})$ when $\Pi^s_U.\rho = \mathtt{init}$, and $(\Pi^s_U.\mathsf{pid}, U)$ when $\Pi^s_U.\rho = \mathtt{resp}$
- $\Pi^s_U.T$: a *transcript* composed of all messages sent and received by the instance in temporal order
- $\Pi^s_U.\alpha \in \{\mathtt{active}, \mathtt{accept}, \mathtt{reject}\}$: the *status* of this instance
- $\Pi^s_U.k$: the *session key* computed by this stage; initially set to empty $\emptyset$; when non-empty, it consists of two symmetric keys $\Pi^s_U.k^{\mathtt{enc}}$ and and $\Pi^s_U.k^{\mathtt{dec}}$ for encryption and decryption with some stateful length-hiding authenticated encryption scheme [29] used to provide confidentiality in the record layer stage. If $\Pi^s_U.\alpha = \mathtt{accept}$, then $\Pi^s_U.k \neq \emptyset$
- $\Pi^s_U.b \in \{0, 1\}$: a randomly sampled bit used in the $\mathsf{Encrypt}$ oracle

[1] In the original ACCE model, these stages were called the *pre-accept* and *post-accept* stages respectively. In PACCE, the parties may start sending encrypted data before accepting, so we have renamed the stages to handshake and record layer, which is suggestive of TLS, but of course can be used to model any appropriate protocol.

Two instances Π_U^s and $\Pi_{U'}^{s'}$ are said to be *partnered* if and only if $\Pi_U^s.\mathsf{pid} = U'$, $\Pi_{U'}^{s'}.\mathsf{pid} = U$, $\Pi_U^s.\rho \neq \Pi_{U'}^{s'}.\rho$, and their transcripts form *matching conversations* [29], denoted $\Pi_U^s.T \approx \Pi_{U'}^{s'}.T$.

The adversary $\mathcal{A}$ controls all communications and can interact with parties using certain oracle queries. Normal operation of the protocol is modelled by the following queries:

- $\mathsf{Send}^{\mathtt{pre}}(\Pi_U^s, m)$: This query is answered as long as $\Pi_U^s.k = \emptyset$. In response the incoming message m is processed by Π_U^s and any outgoing message which is generated as a result of this processing is given to $\mathcal{A}$. Special messages $m = (\mathtt{init}, U')$ and $m = (\mathtt{resp}, U')$ are used to initialize the instance as initiator or responder, respectively, and to specify the identity of the intended partner U'. Note that processing of m may eventually lead to the end of the handshake stage, in which case Π_U^s either computes $\Pi_U^s.k$ and switches to the record layer stage or terminates with a failure.
- $\mathsf{Encrypt}(\Pi_U^s, m_0, m_1, \mathtt{len}, \mathtt{head})$: If $\Pi_U^s.\alpha \neq \mathtt{accept}$, then return $\perp$. Otherwise the processing of this query is detailed below. The result depends on the random bit b sampled by Π_U^s upon initialization, which is used to decide which of the two input message m_0 and m_1 to encrypt using stateful length-hiding authenticated encryption scheme $\mathsf{Enc}/\mathsf{Dec}$, whereby $\mathtt{len}$ is the message length and $\mathtt{head}$ is the header. The experiment maintains an encryption state st_e, a counter u_U^s, and a list C_U^s of ciphertexts for each instance.
  1. $(C^{(0)}, st_e^{(0)}) \leftarrow_R \mathsf{Enc}(\Pi_U^s.k^{\mathtt{enc}}, \mathtt{len}, \mathtt{head}, m_0, st_e)$
  2. $(C^{(1)}, st_e^{(1)}) \leftarrow_R \mathsf{Enc}(\Pi_U^s.k^{\mathtt{enc}}, \mathtt{len}, \mathtt{head}, m_1, st_e)$
  3. If $(C^{(0)} = \perp)$ OR $(C^{(1)} = \perp)$, then return $\perp$
  4. $u_U^s \leftarrow u_U^s + 1$
  5. $(C_U^s[u_U^s], st_e) \leftarrow (C^{(\Pi_U^s.b)}, st_e^{(\Pi_U^s.b)})$
  6. Return $C_U^s[u_U^s]$
- $\mathsf{Decrypt}(\Pi_U^s, C, \mathtt{head})$: The processing of this query is detailed below. The experiment maintains a decryption state st_d and a counter v_U^s. Whenever $\mathcal{A}$ mounts an active attack on encryption communication (by injecting new ciphertexts, delivering modified ciphertexts, or changing their delivery order), the flag phase is set; if the active attack succeeds, the adversary effectively learns the hidden bit $\Pi_U^s.b$.
  1. $(U', s') \leftarrow (U', s')$, if there exists $\Pi_{U'}^{s'}$ that is partnered to Π_U^s, or $(0, 0)$ otherwise
  2. $v_U^s \leftarrow v_U^s + 1$
  3. If $\Pi_U^s.b = 0$, then return $\perp$
  4. $(m, st_d) \leftarrow \mathsf{Dec}(\Pi_U^s.k^{\mathtt{dec}}, \mathtt{head}, C, st_d)$
  5. If $v_U^s > u_{U'}^{s'}$ or $C \neq C_{U'}^{s'}[v_U^s]$ then $\mathsf{phase} \leftarrow 1$
  6. If $\mathsf{phase} = 1$, then return m

  Note that, compared with ACCE, we allow protocol messages to be sent on the encrypted channel: If $\Pi_U^s.\alpha = \mathtt{active}$, then the returned plaintext message m is processed as a protocol message; the resulting outgoing message m' is encrypted using $\mathsf{Encrypt}(\Pi_U^s, m', m', \mathtt{len}(m'), \mathtt{head})$ and the resulting ciphertext C is returned to $\mathcal{A}$. Otherwise, when $\Pi_U^s.\alpha = \mathtt{accept}$, the output of $\mathsf{Decrypt}$ is returned to $\mathcal{A}$.

Furthermore, the adversary may obtain some secret information:

- $\mathsf{RevealSK}(\Pi_U^s)$: Return $\Pi_U^s.k$
- $\mathsf{Corrupt}(C, S)$: Return $\mathsf{pw}_{C,S}$

Note that, compared with AKE models like the eCK model [33] that use public key authentication, password-based protocols cannot tolerate ephemeral key leakage while maintaining resistence to offline dictionary attacks, hence we do not include an ephemeral key leakage query.

Definition 1 (PACCE security)**.** An adversary $\mathcal{A}$ is said to (t, ϵ)-*break* a PACCE protocol if $\mathcal{A}$ runs in time t and at least one of the following two conditions hold:

1. $\mathcal{A}$ *breaks mutual authentication:* When $\mathcal{A}$ terminates, then with probability at least $\epsilon + O(n/|\mathcal{D}|)$ where n is the number of initialized PACCE instances there exists an instance Π_U^s such that
   (a) $\Pi_U^s.\alpha = \texttt{accept}$, and
   (b) $\mathcal{A}$ did not issue $\mathsf{Corrupt}(\texttt{init}(\Pi_U^s), \texttt{resp}(\Pi_U^s))$ before Π_U^s accepted, and
   (c) $\mathcal{A}$ did not issue $\mathsf{RevealSK}(\Pi_U^s)$ or $\mathsf{RevealSK}(\Pi_{U'}^{s'})$ for any $\Pi_{U'}^{s'}$ that is partnered to Π_U^s, and
   (d) there is no unique instance $\Pi_{U'}^{s'}$ that is partnered to Π_U^s.
2. $\mathcal{A}$ *breaks authenticated encryption:* When $\mathcal{A}$ terminates and outputs a triple (U, s, b') such that conditions (a)–(c) from above hold, then we have that

$$\left| \Pr\left[b' = \Pi_U^s.b\right] - \frac{1}{2} \right| \geq \epsilon + O(n/|\mathcal{D}|).$$

A PACCE protocol is (t, ϵ)-*secure* if there is no $\mathcal{A}$ that (t, ϵ)-breaks it; it is *secure* if it is (t, ϵ)-secure for all polynomial t and negligible ϵ in security parameter κ.

Observe that Definition 1 accounts for online dictionary attacks against PACCE protocols by using a lower bound $\epsilon + O(n/|\mathcal{D}|)$ for the adversarial success probability, which models $\mathcal{A}$'s ability to test at most one password (or a constant number) from the uniformly distributed dictionary $\mathcal{D}$ in a single session.

Remark 1. Our security definitions assume that passwords are uniformly distributed over the dictionary $\mathcal{D}$, which is a commonly adopted assumption in game-based security models for cryptographic password authentication protocols. This assumption does cover other password distributions that may occur in practice.

4 PACCE construction 1: Binding using CCE transcript

Our first generic PACCE protocol $\Gamma_T := \Gamma_T(\pi, \xi)$ is constructed as in Section 2.2 from a CCE protocol π and a tPAuth protocol ξ where the tag τ used is the transcript T of the CCE handshake stage. We will see that, because we are using the full transcript from the channel establishment to bind the two protocols together, we need not rely on any authenticity properties of the channel, and thus can use a CCE protocol, not an ACCE protocol.

In this section, we give formal definitions of a CCE protocol and of tag-based password authentication, then prove the security of this generic construction Γ_T using the full transcript as a tag. Finally, we comment that security still holds when we use a cryptographic hash of the transcript of the tag, allowing us to justify the security of using TLS with tSOKE and `tls-unique` channel binding.

4.1 Building block: CCE

As a building block in our analysis we use the notion of *confidential channel establishment (CCE)* that differs from (P)ACCE in that it is supposed to guarantee only confidentiality (and integrity) of the established channel, but not authentication of partners; hence, security is only assured for sessions in which the adversary $\mathcal{A}$ remains passive during the handshake stage. We thus model CCE by slightly modifying the PACCE model from Section 3. The first difference is that there are no passwords nor identities involved; hence no $\mathsf{Corrupt}$ oracle is needed, nor is the U' parameter required in the initialization in the $\mathsf{Send}^{\mathsf{pre}}$ query. Further, the security condition is adjusted so that only sessions where the adversary was passive in the handshake stage are considered. The

oracles RevealSK, Encrypt and Decrypt remain unchanged. The following definition of CCE security is obtained from Definition 1 by considering the above mentioned modifications.

Definition 2 (CCE security)**.** An adversary $\mathcal{A}$ is said to (t, ϵ)-*break* a CCE protocol if $\mathcal{A}$ runs in time t and, when $\mathcal{A}$ terminates and outputs a triple (U, s, b') such that

(a) $\Pi_U^s.\alpha = \texttt{accept}$, and
(b) there exists an instance $\Pi_{U'}^{s'}$ that is partnered to Π_U^s, and
(c) $\mathcal{A}$ did not issue $\mathsf{RevealSK}(\Pi_U^s)$ or $\mathsf{RevealSK}(\Pi_{U'}^{s'})$ for any $\Pi_{U'}^{s'}$ that is partnered to Π_U^s,

then $\left|\Pr\left[b' = \Pi_U^s.b\right] - \frac{1}{2}\right| \geq \epsilon$.

Every secure (P)ACCE protocol is also CCE-secure: if we ignore the authentication aspects, then we still get confidential channel establishment in sessions where the adversary is passive during the handshake.

4.2 Building block: tPAuth

Tag-based authentication [28] accounts for the use of auxiliary, possibly public, strings (tags) in authentication protocols — each party uses a tag, in addition to the authentication factor, and the protocol guarantees that if parties accept then their tags match. This concept was introduced in [28] for public key-based authentication protocols and then generalized in [21] for other types of authentication factors, including passwords and biometrics. Our PACCE constructions 1 and 2 will make use of a *tag-based password authentication (tPAuth)* protocol.

The model of tPAuth can be described using the setting of PACCE protocols from Section 3. A tPAuth session is executed between a client instance Π_C^s and a server instance $\Pi_S^{s'}$ on input the corresponding password $\mathsf{pw}_{C,S}$ from the dictionary $\mathcal{D}$ and some tag $\tau \in \{0,1\}^*$. A tPAuth session is successful if both instances use the same password $\mathsf{pw}_{C,S}$ and tag τ as their input. The requirement on tag equality leads to the extended definition of partnering: two instances Π_C^s and $\Pi_S^{s'}$ are *partnered* if $\Pi_C^s.\mathsf{pid} = S$, $\Pi_S^{s'}.\mathsf{pid} = C$, $\Pi_C^s.T \approx \Pi_{S'}^{s'}.T$ (matching transcripts), and $\Pi_C^s.\tau = \Pi_S^{s'}.\tau$ (equal tags).

A tPAuth adversary $\mathcal{A}$ is active and interacts with instances of $U \in \{C, S\}$ using the following oracles:

- $\mathsf{Send}(\Pi_U^s, m)$: This query is identical to $\mathsf{Send}^{\mathtt{pre}}$ from the PACCE model except for one important difference — when $\mathcal{A}$ initializes some instance Π_U^s using the special messages $m = (\mathtt{init}, U', \tau)$ or $m = (\mathtt{resp}, U', \tau)$ then it additionally provides as input a tag τ which will be used by the instance in the tPAuth session. This essentially gives $\mathcal{A}$ full control over the tags that are used in the protocol.
- $\mathsf{Corrupt}(C, S)$: Like in the PACCE model this query reveals the corresponding password $\mathsf{pw}_{C,S}$.

The security of tPAuth protocols, defined in the following, extends the traditional password authentication requirement that accounts for online dictionary attacks with the requirement of tag equality, which is implied by condition 3 due to the extended definition of partnering.

Definition 3 (tPAuth security)**.** An adversary $\mathcal{A}$ is said to (t, ϵ)-break a tPAuth protocol if after the termination of $\mathcal{A}$ that runs in time t with probability at least $\epsilon + O(n/|\mathcal{D}|)$ where n is the number of initialized tPAuth instances, there exists an instance Π_U^s such that

1. $\Pi_U^s.\alpha = \texttt{accept}$, and
2. $\mathcal{A}$ did not issue $\mathsf{Corrupt}(\mathtt{init}(\Pi_U^s), \mathtt{resp}(\Pi_U^s))$ before Π_U^s accepted, and
3. there is no unique instance $\Pi_{U'}^{s'}$ that is partnered to Π_U^s.

A tPAuth protocol is (t, ϵ)-*secure* if there is no $\mathcal{A}$ that (t, ϵ)-breaks it; it is *secure* if it is (t, ϵ)-secure for all polynomial t and negligible ϵ in security parameter κ.

In Section 7, we present tSOKE, a tag-based variant of the Simple Open Key Exchange (SOKE) protocol from [1]. Since SOKE is a PAKE protocol, its tag-based version tSOKE also establishes a secure session key k. This key will only be used in our PACCE construction 3 whereas our PACCE constructions 1 and 2 will be using tSOKE as a tPAuth protocol, that is only relying on its mutual authentication property.

4.3 Security analysis of PACCE construction 1

Theorem 1 ($\mathsf{CCE}+\mathsf{tPAuth}_{\tau=T_{\mathsf{CCE}}} \implies \mathsf{PACCE}$)**.** *The generic PACCE protocol construction* $\Gamma_T(\pi, \xi)$ *from a CCE protocol* π *and a tPAuth protocol* ξ*, with the tag equal to the transcript* T_{CCE} *from the CCE handshake stage, is secure, assuming the underlying protocols are secure.*

The proof consists of a sequence of games. In the first game, the simulator continues to simulate the CCE portion of the protocol but undetectably replaces the tPAuth simulation with that of a real tPAuth challenger. Next, the simulator aborts if any of its instances accept without a partnered instance existing; this will correspond to a violation of authentication in the underlying tPAuth challenger. In the third game, the simulator now undetectably replaces the CCE simulation with that of a real CCE challenger. An adversary who can win against the resulting PACCE simulator can be used to win against the underlying CCE challenger.

Proof. Let $\Pi_U^s.T = \Pi_U^s.T_{\mathsf{CCE}} \| \Pi_U^s.T_{\mathsf{tPAuth}}$ denote the CCE handshake stage and (plaintext) tPAuth portions of the transcript of Π_U^s.

Game G_0. This is the original PACCE security experiment from Definition 1 played with a PACCE adversary $\mathcal{A}$ that is given access to the oracles $\mathsf{Send}^{\mathtt{pre}}$, RevealSK, Corrupt, Encrypt, and Decrypt.

In particular, the simulator $\mathcal{B}$ first initializes passwords $\mathsf{pw}_{C,S}$ for all pairs of parties (C, S). For every session Π_U^s of the PACCE protocol, the simulator $\mathcal{B}$ will maintain "shadow" sessions of the tPAuth protocol ($\overline{\Pi_U^s}$) and CCE protocol ($\widehat{\Pi_U^s}$). $\mathcal{B}$ activates the PACCE adversary $\mathcal{A}$ and responds to queries by $\mathcal{A}$ as follows:

- $\mathsf{Send}^{\mathtt{pre}}(\Pi_U^s, m)$: $\mathcal{B}$ emulates a call for the CCE protocol π to $\mathsf{Send}^{\mathtt{pre}}(\widehat{\Pi_U^s}, m)$ to obtain an outgoing message m'; note that calls from $\mathcal{A}$ to this oracle are rejected once the CCE instance $\widehat{\Pi_U^s}$ has accepted. When the CCE instance $\widehat{\Pi_U^s}$ accepts:
  If $\Pi_U^s.\rho = \mathtt{init}$, $\mathcal{B}$ constructs the first message $\hat{m}$ of the tPAuth protocol ξ by emulating a call for ξ to $\mathsf{Send}(\overline{\Pi_U^s}, \mathtt{init}, \Pi_U^s.T_{\mathsf{CCE}})$. Then, $\mathcal{B}$ encrypts $\hat{m}$ by emulating a call for π to $\mathsf{Encrypt}(\widehat{\Pi_U^s}, \hat{m}, \hat{m}, \mathtt{len}(\hat{m}), \mathtt{head})$ to obtain a ciphertext $\hat{C}$.
  If $\Pi_U^s.\rho = \mathtt{resp}$, $\mathcal{B}$ initializes the corresponding instance of the tPAuth protocol ξ by emulating a call for ξ to $\mathsf{Send}(\overline{\Pi_U^s}, \mathtt{resp}, \Pi_U^s.T_{\mathsf{CCE}})$. This does not return anything.
  $\mathcal{B}$ returns to $\mathcal{A}$ any outgoing CCE handshake message m' as well as any encrypted tPAuth message $\hat{C}$.
- $\mathsf{RevealSK}(\Pi_U^s)$: $\mathcal{B}$ returns the CCE session keys $\Pi_U^s.k^{\mathtt{enc}}$ and $\Pi_U^s.k^{\mathtt{dec}}$ to $\mathcal{A}$ if $\Pi_U^s.k \neq \emptyset$.
- $\mathsf{Corrupt}(C, S)$: $\mathcal{B}$ returns $\mathsf{pw}_{C,S}$ to $\mathcal{A}$.
- $\mathsf{Encrypt}(\Pi_U^s, m_0, m_1, \mathtt{len}, \mathtt{head})$: If $\Pi_U^s.\alpha \neq \mathtt{accept}$, return $\perp$. Otherwise, $\mathcal{B}$ emulates a call for the CCE protocol π to $\mathsf{Encrypt}(\widehat{\Pi_U^s}, m_0, m_1, \mathtt{len}, \mathtt{head})$ and returns the result C to $\mathcal{A}$.
- $\mathsf{Decrypt}(\Pi_U^s, C, \mathtt{head})$: If the CCE portion of the protocol π for this session is still in the handshake stage, return $\perp$. If the CCE portion is in the record layer stage but $\Pi_U^s.\alpha \neq \mathtt{accept}$, then $\mathcal{B}$ emulates a call for the CCE protocol π to $\mathsf{Decrypt}(\widehat{\Pi_U^s}, C, \mathtt{head})$ and processes the resulting plaintext message m' for the tPAuth protocol ξ by emulating a call to $\mathsf{Send}(\overline{\Pi_U^s}, m')$.

If $\overline{\Pi^s_U}$ accepts or rejects, so too does Π^s_U. If $\overline{\Pi^s_U}$ returns a tPAuth protocol message m'', then $\mathcal{B}$ encrypts it by emulating a call to $\mathsf{Encrypt}(\widehat{\Pi^s_U}, m'', m'', \mathtt{len}(m''), \mathtt{head})$ and returns the resulting ciphertext C'' to $\mathcal{A}$.

If the CCE portion is in the record layer stage and $\Pi^s_U.\alpha = \mathtt{accept}$, then $\mathcal{B}$ emulates a call for the CCE protocol π to $\mathsf{Decrypt}(\widehat{\Pi^s_U}, C, \mathtt{head})$ and returns the result m' to $\mathcal{A}$.

Since $\mathcal{B}$ follows the original experiment exactly,

$$\mathsf{Adv}^{\mathsf{PACCE}}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{G}_0}_{\mathcal{A},\Gamma_T(\pi,\xi)} \ . \tag{1}$$

Game G_1. In this game, the simulator $\mathcal{B}$ makes use of a challenger $\mathcal{C}^\xi$ for the tPAuth protocol ξ to simulate the tPAuth portion of the combined protocol. $\mathcal{B}$ will maintain a one-to-one mapping between sessions Π^s_U of the PACCE protocol and sessions $\overline{\Pi^s_U}$ of the tPAuth protocol. In particular, the tPAuth challenger $\mathcal{C}^\xi$ initializes passwords for all pairs of parties. Then, the simulator $\mathcal{B}$ activates the PACCE adversary $\mathcal{A}$ and responds to queries by $\mathcal{A}$ as in game G_0, except in $\mathsf{Send}^{\mathtt{pre}}$, $\mathsf{Corrupt}$, and $\mathsf{Decrypt}$: where it would have emulated calls to the tPAuth protocol, it relays those calls to $\mathcal{C}^\xi$. Since $\mathcal{B}$'s use of the tPAuth challenger $\mathcal{C}^\xi$ perfectly matches how it would use ξ if it were implementing ξ itself,

$$\mathsf{Adv}^{\mathsf{G}_0}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{G}_1}_{\mathcal{A},\Gamma_T(\pi,\xi)} \ . \tag{2}$$

Game G_2. In this game, the simulator $\mathcal{B}$ acts as in game G_1, except that it aborts when there exists an instance Π^s_U that accepts without having a partnered instance (i.e., if $\Pi^s_U.\mathsf{pid} = U'$, then there does not exist an instance $\Pi^{s'}_{U'}$ such that Π^s_U and $\Pi^{s'}_{U'}$ have matching transcripts) and $\mathcal{A}$ has not issued a $\mathsf{Corrupt}(\mathtt{init}(\Pi^s_U), \mathtt{resp}(\Pi^s_U))$ query. $\mathcal{B}$'s behaviour in game G_2 is exactly the same as in game G_1 except when this abort event occurs. Suppose the abort event occurs because of instance Π^s_U. Let Since PACCE instance Π^s_U accepted, the underlying tPAuth instance $\overline{\Pi^s_U}$ must also have accepted; its transcript is $\Pi^s_U.T_{\mathsf{tPAuth}}$ and it was initialized on tag $\Pi^s_U.T_{\mathsf{CCE}}$.

However, no partner instance to Π^s_U exists: namely, there is no PACCE instance with transcript $T' = T'_{\mathsf{CCE}} \| T'_{\mathsf{tPAuth}}$ matching $\Pi^s_U.T$. Now, either (i) there exists a PACCE instance with T'_{CCE} equal to $\Pi^s_U.T_{\mathsf{CCE}}$ but not with T'_{tPAuth} matching $\Pi^s_U.T_{\mathsf{tPAuth}}$, or (ii) there is no other PACCE instance with T'_{CCE} equal to $\Pi^s_U.T_{\mathsf{CCE}}$. In case (i), this means there exists a tPAuth instance initialized on tag $T'_{\mathsf{CCE}} = \Pi^s_U.T_{\mathsf{CCE}}$ but not with transcript T'_{tPAuth} matching $\Pi^s_U.T_{\mathsf{tPAuth}}$; thus, $\overline{\Pi^s_U}$ has no partnered instance, violating condition 3 of Definition 3. In case (ii), this means there does not exist any other tPAuth instance initialized on tag $\Pi^s_U.T_{\mathsf{CCE}}$; thus, $\overline{\Pi^s_U}$ has no partnered instance initialized with the same tag $\Pi^s_U.T_{\mathsf{CCE}}$, violating condition 3 of Definition 3. Since $\mathcal{B}$ aborts only when the authentication condition in the underling tPAuth protocol ξ is violated,

$$\left|\mathsf{Adv}^{\mathsf{G}_1}_{\mathcal{A},\Gamma_T(\pi,\xi)} - \mathsf{Adv}^{\mathsf{G}_2}_{\mathcal{A},\Gamma_T(\pi,\xi)}\right| \leq \mathsf{Adv}^{\mathsf{tPAuth}}_{\mathcal{B},\xi} \ . \tag{3}$$

Game G_3. In this game, the simulator $\mathcal{B}$ acts as in game G_2, except it makes use of a challenger $\mathcal{C}^\pi$ for the CCE protocol π to simulate the CCE portions of the combined protocol. $\mathcal{B}$ will maintain a one-to-one mapping between sessions Π^s_U of the PACCE protocol and sessions $\widehat{\Pi^s_U}$ of the CCE protocol. The simulator $\mathcal{B}$ activates the PACCE adversary $\mathcal{A}$ and responds to queries by $\mathcal{A}$ as in game G_2, except in $\mathsf{Send}^{\mathtt{pre}}$, $\mathsf{RevealSK}$, and $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$: where it would have emulated calls to the CCE protocol, it relays those calls to $\mathcal{C}^\pi$. Since $\mathcal{B}$'s use of the CCE challenger $\mathcal{C}^\pi$ perfectly matches how it would use π if it were implementing π itself,

$$\mathsf{Adv}^{\mathsf{G}_2}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{G}_3}_{\mathcal{A},\Gamma_T(\pi,\xi)} \ . \tag{4}$$

Analysis of game G_3. Suppose $\mathcal{A}$ outputs a tuple (U, s, b') that would win the PACCE experiment, namely $\Pi^s_U.b = b'$ and $\mathcal{A}$ did not query $\mathsf{RevealSK}(\Pi^s_U)$ or $\mathsf{RevealSK}(\Pi^{s'}_{U'})$ for any partnered instance

$\Pi^{s'}_{U'}$. Since in game G_2 $\mathcal{B}$ would have aborted if Π^{s}_{U} accepted without having a partnered instance, there must exist some partnered instance $\Pi^{s'}_{U'}$. Correspondingly, in $\mathcal{C}^{\pi}$ there exists an instance $\widehat{\Pi^{s}_{U}}$ with partnered instance $\widehat{\Pi^{s'}_{U'}}$. Since $\mathcal{A}$ did not issue the prohibited $\mathsf{RevealSK}$ queries, $\mathcal{B}$ also did not issue the prohibited queries $\mathsf{RevealSK}(\widehat{\Pi^{s}_{U}})$ or $\mathsf{RevealSK}(\widehat{\Pi^{s}_{U}})$. Thus, if $\mathcal{B}$ outputs (U, s, b') to the CCE challenger $\mathcal{C}^{\pi}$, it will win the CCE experiment whenever $\mathcal{A}$ wins the PACCE experiment:

$$\mathsf{Adv}^{\mathsf{G}_3}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{CCE}}_{\mathcal{B},\pi} \quad . \tag{5}$$

Final result. Combining equations (1)–(5),

$$\mathsf{Adv}^{\mathsf{PACCE}}_{\mathcal{A},\Gamma_T(\pi,\xi)} \leq \mathsf{Adv}^{\mathsf{tPAuth}}_{\mathcal{B},\xi} + \mathsf{Adv}^{\mathsf{CCE}}_{\mathcal{B},\pi} \quad ,$$

and we obtain the security of the combined construction. □

4.4 Using `tls-unique` channel binding

The `tls-unique` channel binding mechanism [9] can be used to instantiate construction 1. Recall from Section 2.1.3 that for `tls-unique` the channel binding string is the first `Finished` message, which is the output of a pseudorandom function on the hash of the TLS handshake transcript.

It is straightforward to see that if the `Finished` message is used as the tag for channel binding instead of the full transcript in an analogous generic construction $\Gamma_{fin}(\pi, \xi)$, and the hash function H is collision-resistant and the pseudorandom function PRF is secure, then Γ_{fin} is a secure PACCE. This follows by noting that, except with negligible probability, the parties must use the same transcript in order to arrive at the same tag, and then the proof of Theorem 1 applies. This assumes that session resumption is disabled (see Section 2.1.3) to avoid the triple handshake attack [14].

5 PACCE construction 2: Binding using server public key

Our second generic PACCE protocol $\Gamma_{pk} := \Gamma_{pk}(\pi, \xi)$ is constructed as in Section 2.2 from an ACCE protocol π and a tPAuth protocol ξ where the tag τ used is the long-term public key used by the server in the ACCE protocol.

Because we are using only the server's long-term public key, and not the full transcript from the channel establishment, to bind the two protocols together, we now must rely on some authenticity properties of the channel. However, we will not be relying on *users* to correctly validate the server's public key or decide which long-term server public key corresponds with which password: from the external perspective, the protocol is still a PACCE protocol, with authentication only coming from passwords, not from long-term server public keys.

In this section, we give a formal definition of a ACCE protocol, then prove the security of this generic construction Γ_{pk} using the server's long-term public key as a tag. Finally, we comment that security still holds when we use a a certificate or a hash of a certificate containing the public key, allowing us to justify the security of using TLS with tSOKE and `tls-server-end-point` channel binding.

5.1 Building block: ACCE (with key registration)

ACCE [29] is currently the most complete model for the security properties of the core TLS protocol. We use a variant of ACCE as a building block in our generic construction. The first variation is that we allow for either server-only or mutual authentication, as in Giesen et al. [23]; when server-only

authentication is used, only client instances are legitimate targets for breaking authentication. The second variation is in how static public keys are distributed. In typical AKE and ACCE models, it is simply assumed that parties have authentic copies of all static public keys, abstracting the problem away. Since we will use ACCE as a building block under the assumption that the "static" public keys are not to be trusted as authentic, we allow the adversary to cause any public key to be accepted as a static public key using a Register query; only sessions where the key is not an adversary-registered key are legitimate targets for breaking.

Formally, the differences between the ACCE model with key registration, compared to the PACCE model, are as follows:

- There are no passwords and hence no need to consider online dictionary attacks in the security definition.
- Each party U generates a long-term public-private key pair (pk_U, sk_U) and all parties, including the adversary, are assumed to have copies of all long-term public keys.
- Π^s_U maintains a variable $\Pi^s_U.\omega \in \{\texttt{server-only}, \texttt{mutual}\}$ indicating whether it is expecting server-only or mutual authentication.
- Π^s_U maintains a variable $\Pi^s_U.\mathsf{ppk}$ of the public key observed to be used by the peer.
- In the $\mathsf{Send}^{\mathtt{pre}}$ query, the initialization messages no longer specify the identity of the intended partner (this is learned as the protocol runs), but do include the required authentication mode.
- $\mathsf{Register}(pk)$: All parties add pk to the list of long-term public keys.
- $\mathsf{Corrupt}(U)$: Returns sk_U.
- Condition (b) of the security definition is: $\mathcal{A}$ did not issue $\mathsf{Corrupt}(\Pi^s_U.\mathsf{pid})$ before Π^s_U accepted.
- For breaking authentication, an additional condition is added:
  (e) if $\Pi^s_U.\omega = \texttt{server-only}$, then $\Pi^s_U.\rho = \texttt{init}$.
- For breaking both authentication and authenticated encryption, an additional condition is added:
  (f) $\mathcal{A}$ did not issue $\mathsf{Register}(\Pi^s_U.\mathsf{ppk})$.

5.2 Security analysis of PACCE construction 2

Theorem 2 (ACCE+$\mathsf{tPAuth}_{\tau=pk} \implies \mathsf{PACCE}$)**.** *The generic PACCE protocol construction $\Gamma_T(\pi, \xi)$ from an ACCE protocol π and a tPAuth protocol ξ, with tag set to the server's public key pk from the ACCE handshake stage, is secure, assuming the underlying protocols are secure.*

The strategy of the proof is similar to that of Theorem 1. In the first game, the simulator simulates the ACCE portion of the protocol and undetectably replaces the tPAuth portion using messages that it obtains from a real tPAuth challenger. Next, the simulator aborts if any of its instances accept without an instance whose tPAuth transcripts and the input tags match, which corresponds to an attack against the tPAuth protocol. In the third game, the simulator will use messages obtained from a real ACCE challenger such that it can use any adversary who wins against the resulting PACCE simulator to break the security of the ACCE protocol; when the adversary uses its own long-term public keys (which is allowed since they are not authenticated), we use the key registration functionality of the (modified) ACCE challenger.

Proof. **Game G_0.** This is the original PACCE security experiment from Definition 1 played with a PACCE adversary $\mathcal{A}$ that is given access to the oracles $\mathsf{Send}^{\mathtt{pre}}$, $\mathsf{RevealSK}$, $\mathsf{Corrupt}$, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$. It thus proceeds identical to G_0 from the proof of Theorem 1 except for the following modifications since π is now an ACCE protocol. In particular, the simulator $\mathcal{B}$ additionally needs

to generate all long-term public key / secret key pairs (pk_U, sk_U) that are used in π. Since $\mathcal{B}$ follows the original experiment exactly,

$$\mathsf{Adv}^{\mathsf{PACCE}}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{G}_0}_{\mathcal{A},\Gamma_T(\pi,\xi)} \ . \tag{6}$$

Game G_1. In this game, the simulator $\mathcal{B}$ makes use of a challenger $\mathcal{C}^\xi$ for the tPAuth protocol ξ to simulate the tPAuth portion of the combined protocol. $\mathcal{B}$ maintains a one-to-one mapping between sessions Π^s_U of the PACCE protocol and sessions $\overline{\Pi^s_U}$ of the tPAuth protocol. In particular, the tPAuth challenger $\mathcal{C}^\xi$ initializes passwords for all pairs of parties. Then, the simulator $\mathcal{B}$ runs the PACCE adversary $\mathcal{A}$ and responds to its queries as in game G_0, except in $\mathsf{Send}^{\mathtt{pre}}$, $\mathsf{Corrupt}$, and $\mathsf{Decrypt}$: where it would have emulated calls to the tPAuth protocol, it relays calls to $\mathcal{C}^\xi$. Since $\mathcal{B}$'s use of the tPAuth challenger $\mathcal{C}^\xi$ perfectly matches how it would use ξ if it were implementing ξ itself,

$$\mathsf{Adv}^{\mathsf{G}_0}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{G}_1}_{\mathcal{A},\Gamma_T(\pi,\xi)} \ . \tag{7}$$

Game G_2. In this game, the simulator $\mathcal{B}$ acts as in game G_1, except that it aborts when there exists an instance Π^s_U that used some public key pk as an input tag to the tPAuth part of the protocol and accepted with the corresponding tPAuth transcript $\Pi^s_U.T_{\mathsf{tPAuth}}$ but for which there exists no instance $\Pi^{s'}_{U'} = \Pi^s_U.\mathsf{pid}$ with the matching tPAuth transcript and tag and $\mathcal{A}$ has not issued a $\mathsf{Corrupt}(\mathtt{init}(\Pi^s_U), \mathtt{resp}(\Pi^s_U))$ query. Note that, other than this abort event, $\mathcal{B}$'s behaviour in game G_2 is exactly the same as in game G_1, i.e. $\mathcal{B}$ generates (pk_U, sk_U) itself and answers all queries of $\mathcal{A}$ related to ACCE protocol part on its own. The occurrence of this abort event would violate condition 3 of tPAuth security from Definition 3, thus

$$\left|\mathsf{Adv}^{\mathsf{G}_1}_{\mathcal{A},\Gamma_T(\pi,\xi)} - \mathsf{Adv}^{\mathsf{G}_2}_{\mathcal{A},\Gamma_T(\pi,\xi)}\right| \leq \mathsf{Adv}^{\mathsf{tPAuth}}_{\mathcal{B},\xi} \ . \tag{8}$$

The consequence of this game is that if no abort event takes place when for every PACCE instance Π^s_U that accepts there must exist an instance $\Pi^{s'}_{U'}$ with which the tPAuth protocol part was securely executed. Since this game implies the equality of the input tags used by Π^s_U and $\Pi^{s'}_{U'}$ it therefore also excludes the case where the public key pk used as the input tag by Π^s_U was maliciously generated by the PACCE adversary $\mathcal{A}$. In other words, any PACCE instance Π^s_U that accepts in this game must have used any of the public keys pk that were generated honestly by the simulator $\mathcal{B}$.

Game G_3. In this game, the simulator $\mathcal{B}$ acts as in game G_2, except it makes use of a challenger $\mathcal{C}^\pi$ for the ACCE protocol π to simulate the ACCE portions of the combined protocol. $\mathcal{B}$ will maintain a one-to-one mapping between sessions Π^s_U of the PACCE protocol and sessions $\widehat{\Pi^s_U}$ of the ACCE protocol.

The simulator $\mathcal{B}$ receives public keys pk_U from $\mathcal{C}^\pi$, activates the PACCE adversary $\mathcal{A}$, and responds to the queries by $\mathcal{A}$ as in game G_2, except that its replies to $\mathsf{RevealSK}$, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$ oracles are obtained by first forwarding those queries to $\mathcal{C}^\pi$ and passing on the responses back to $\mathcal{A}$. Also $\mathsf{Send}^{\mathtt{pre}}$ queries of $\mathcal{A}$ are answered using corresponding queries to $\mathcal{C}^\pi$ except for the case where the query $\mathsf{Send}^{\mathtt{pre}}$ queries of $\mathcal{A}$ contains an ACCE public key pk that is different from the public keys that $\mathcal{B}$ received from $\mathcal{C}^\pi$; in which case $\mathcal{B}$ first needs to issue a $\mathsf{Register}(pk)$ query to $\mathcal{C}^\pi$. This offers perfect simulation of the ACCE part of the combined protocol so that

$$\mathsf{Adv}^{\mathsf{G}_2}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{G}_3}_{\mathcal{A},\Gamma_T(\pi,\xi)} \ . \tag{9}$$

Analysis of game G_3. Suppose $\mathcal{A}$ outputs a tuple (U, s, b') and wins in the PACCE experiment. According to Definition 1 $\mathcal{A}$ can win if one of the following two conditions is satisfied: either $\mathcal{A}$ breaks the authentication in the pre-accept phase of the PACCE protocol as per condition 1 or $\mathcal{A}$ breaks the authenticated encryption in the post-accept phase as per condition 2.

First assume $\mathcal{A}$ wins by condition 1. In this case there exists an instance Π_U^s that has accepted without having a partnered instance $\Pi_{U'}^{s'}$. Game G_2 guarantees that for any Π_U^s that accepts in PACCE there must be an instance $\Pi_{U'}^{s'}$ with matching tPAuth transcript and input tag (ACCE public key pk). Hence, the only possibility for $\mathcal{A}$ to win by condition 1 in G_3 is when the ACCE transcripts of Π_U^s and $\Pi_{U'}^{s'}$ do not match. However, since both parties must have used an honestly generated public key pk as tag (this is implied by G_2) any mismatch in their ACCE transcript parts would contradict the assumed security of the underlying ACCE protocol.

Assume now that $\mathcal{A}$ wins by condition 2. In this case the output (U, s, b') of $\mathcal{A}$ would also break the authenticated encryption security of the ACCE protocol. Hence, if $\mathcal{B}$ outputs (U, s, b') to the ACCE challenger $\mathcal{C}^\pi$, it will win the ACCE experiment whenever $\mathcal{A}$ wins the PACCE experiment. Thus,

$$\mathsf{Adv}^{\mathsf{G}_3}_{\mathcal{A},\Gamma_T(\pi,\xi)} = \mathsf{Adv}^{\mathsf{ACCE}}_{\mathcal{B},\pi} \quad . \tag{10}$$

Final result. Combining equations (6)–(10),

$$\mathsf{Adv}^{\mathsf{PACCE}}_{\mathcal{A},\Gamma_T(\pi,\xi)} \leq \mathsf{Adv}^{\mathsf{tPAuth}}_{\mathcal{B},\xi} + \mathsf{Adv}^{\mathsf{ACCE}}_{\mathcal{B},\pi} \quad ,$$

and we obtain the security of the combined construction. □

5.3 Using `tls-server-end-point` channel binding

The `tls-server-end-point` channel binding mechanism [9] can be used to instantiate construction 2. Recall from Section 2.1.3 that for `tls-server-end-point` the channel binding string is the hash of the server's X.509 certificate. Note that the certificate contains the server's public key as a canonically identifiable substring.

It is straightforward to see that if the hash of the certificate is used as the tag for channel binding instead of the raw public key in an analogous generic construction $\Gamma_{cert}(\pi, \xi)$, and the hash function is second-preimage-resistant, then Γ_{cert} is a secure PACCE. This follows by noting that an active adversary must use a certificate that hashes to the same value as the server's certificate, and then incorporating that as an additional game hop in the proof of Theorem 2. This assumes that session resumption is disabled (see Section 2.1.3) to avoid the triple handshake attack [14].

6 PACCE construction 3: Binding using server domain name

Our third generic PACCE protocol $\Gamma_S := \Gamma_S(\xi, \pi)$ is constructed as in Figure 1b from a tPAKE protocol ξ where the tag τ used is the server domain name S followed by an ACCE protocol π that uses the server's long-term public key pk_S. In particular, client and server first run a tPAKE protocol on input of the client's password and the server's domain name as tag. After successful completion of tPAKE, client and server establish a secure channel using an ACCE protocol that is then used to authenticate the client using the key exchanged with tPAKE before. For a more concrete description we refer to Figure 6. Because we are using only the server's domain name, and not the public key, to bind the two protocols together, we now must rely on the authenticated distribution of public keys.

6.1 Building block: tPAKE

We expand the tPAuth model and its security definitions from Section 4.2 towards tPAKE protocols that in addition to authentication allow the participating tPAKE instances Π_U^s with the `accept` status to derive a secure session key $\Pi_U^s.k$. We define an additional oracle Test_b that is parameterized with a random bit b and can be queried by a tPAKE adversary $\mathcal{A}$ multiple times:

- $\mathsf{Test}_b(\Pi_U^s)$: The oracle generates a response only if $\Pi_U^s.\alpha = \texttt{accept}$ and $\mathcal{A}$ did not ask $\mathsf{Corrupt}(\texttt{init}(\Pi_U^s), \texttt{resp}(\Pi_U^s))$ before Π_U^s reached that status. If $b = 1$ the oracle responds with the real key $\Pi_U^s.k$. If $b = 0$ the oracle checks whether there exists an instance $\Pi_{U'}^{s'}$ that is partnered with Π_U^s and for which a random key k was generated in response to a Test_b query and if so returns that k; otherwise the oracle responds with a new random key k.

The security definition of tPAKE protocols extends those for tPAuth protocols towards the indistinguishability property for the session keys.

Definition 4 (tPAKE security)**.** An adversary $\mathcal{A}$ is said to (t, ϵ)-break a tPAKE protocol if $\mathcal{A}$ runs in time t and at least one of the following conditions hold:

1. $\mathcal{A}$ *breaks mutual authentication:* When $\mathcal{A}$ terminates, then with probability of at least $\epsilon + O(n/|\mathcal{D}|)$ there exists an instance Π_U^s for which the conditions 1-3 from Definition 3 hold.
2. $\mathcal{A}$ *breaks key indistinguishability:* When $\mathcal{A}$ terminates and outputs bit b' such that conditions 1-2 from Definition 3 hold, then we have that

$$\left|\Pr\left[b' = b\right] - \frac{1}{2}\right| \geq \epsilon + O(n/|\mathcal{D}|).$$

A tPAKE protocol is (t, ϵ)-*secure* if there is no $\mathcal{A}$ that (t, ϵ)-breaks it; it is *secure* if it is (t, ϵ)-secure for all polynomial t and negligible ϵ in security parameter κ.

The following lemma establishes a relationship between tPAKE and tPAuth protocols.

Lemma 1 (tPAKE $\implies$ tPAuth)**.** *Any (t, ϵ)-secure tPAKE protocol is also a (t, ϵ)-secure tPAuth protocol.*

Proof. This lemma follows directly by inspection of the corresponding security definitions. In particular, condition 1 for the security of tPAKE protocols from Definition 4 implies conditions 1-3 for the security of tPAuth protocols from Definition 3. □

As a tPAKE instantiation we will use tSOKE, the tag-based version of the SOKE protocol. While our constructions 1 and 2 were only relying on the authentication properties of tSOKE, our construction 3 also makes use of the session key that can be derived from the pre-master key k computed in tSOKE.

6.2 Building block: ACCE (without key registration)

Because the tPAKE protocol will be executed first and uses solely the domain name S as the input tag, the server's public key pk_S must be validated within the ACCE protocol. This means that the ACCE adversary $\mathcal{A}$ from Section 5.1 is no longer given access to the $\mathsf{Register}(pk)$ query and condition (f) which relates to this query in the definition of ACCE security must be removed. That is, by using the server's domain name as the tag we require stronger authenticity guarantees on the distributed public keys utilized in the ACCE protocol than in construction 2.

6.3 Security analysis of PACCE construction 3

Theorem 3 ($\mathsf{tPAKE}_{\tau=S} + \mathsf{ACCE} \implies \mathsf{PACCE}$)**.** *The generic PACCE protocol construction $\Gamma_S(\xi, \pi)$ from a tPAKE protocol ξ with tag set to the server's domain name S and an ACCE protocol π that uses the server's public key pk_S, is secure, assuming the underlying protocols are secure.*

Proof. **Game G_0.** This is the original PACCE security experiment from Definition 1 played with a PACCE adversary $\mathcal{A}$ that is given access to the oracles $\mathsf{Send}^{\mathtt{pre}}$, $\mathsf{RevealSK}$, $\mathsf{Corrupt}$, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$. It thus proceeds identical to G_0 from the proof of Theorem 2. In particular, the simulator $\mathcal{B}$ additionally needs to generate all long-term public key / secret key pairs (pk_U, sk_U) that are used in π. Since $\mathcal{B}$ follows the original experiment exactly,

$$\mathsf{Adv}^{\mathsf{PACCE}}_{\mathcal{A},\Gamma_S(\xi,\pi)} = \mathsf{Adv}^{\mathsf{G}_0}_{\mathcal{A},\Gamma_S(\xi,\pi)} \ . \tag{11}$$

Game G_1. In this game, the simulator $\mathcal{B}$ makes use of a challenger $\mathcal{C}^{\xi}$ for the tPAKE protocol ξ to simulate the tPAKE portion of the combined protocol and the tPAKE key k that is sent over the ACCE channel. In particular, $\mathcal{B}$ maintains a one-to-one mapping between sessions Π^s_U of the PACCE protocol and sessions $\overline{\Pi^s_U}$ of the tPAKE protocol. The tPAKE challenger $\mathcal{C}^{\xi}$ initializes passwords for all pairs of parties. The simulator $\mathcal{B}$ runs the PACCE adversary $\mathcal{A}$ and responds to its queries as in game G_0, except in $\mathsf{Send}^{\mathtt{pre}}$, $\mathsf{Corrupt}$, and $\mathsf{Decrypt}$: where it would have emulated calls to the tPAKE protocol, it relays calls to $\mathcal{C}^{\xi}$. The simulator $\mathcal{B}$ aborts when there exists an instance Π^s_U that used some domain name S as an input tag to the tPAKE part of the protocol and accepted with the corresponding tPAKE transcript $\Pi^s_U.T_{\mathsf{tPAKE}}$ but for which there exists no instance $\Pi^{s'}_{U'} = \Pi^s_U.\mathsf{pid}$ with the matching tPAKE transcript and tag and $\mathcal{A}$ has not issued a $\mathsf{Corrupt}(\mathtt{init}(\Pi^s_U), \mathtt{resp}(\Pi^s_U))$ query. If the simulator does not abort it continues with the execution of the ACCE portion π for a PACCE session Π^s_U as in previous game G_0, except that it obtains tPAKE key k by querying $\mathsf{Test}_b(\overline{\Pi^s_U})$ to the corresponding tPAKE challenger $\mathcal{C}^{\xi}$.

The use of the tPAKE challenger $\mathcal{C}^{\xi}$ for the tPAKE portion of the PACCE protocol perfectly matches how the simulator $\mathcal{B}$ would use ξ if it were executing ξ itself. The occurrence of the abort event would violate condition 1 of tPAKE security from Definition 4, whereas the ability of PACCE adversary $\mathcal{A}$ to distinguish between G_0 and G_1 based on the usage of either real or random tPAKE key k would violate condition 2. Therefore,

$$\left|\mathsf{Adv}^{\mathsf{G}_0}_{\mathcal{A},\Gamma_S(\xi,\pi)} - \mathsf{Adv}^{\mathsf{G}_1}_{\mathcal{A},\Gamma_S(\xi,\pi)}\right| \leq \mathsf{Adv}^{\mathsf{tPAKE}}_{\mathcal{B},\xi} \ . \tag{12}$$

The consequence of this game is that if no abort event takes place, then for every PACCE instance Π^s_U that accepts there must exist an instance $\Pi^{s'}_{U'}$ with which the tPAKE protocol part was securely executed. Since this game implies the equality of the input tags used by Π^s_U and $\Pi^{s'}_{U'}$ it therefore also excludes the case where the domain name S used as the input tag by Π^s_U was not matching the public key pk_S used in the ACCE portion of the protocol, following the assumption that public keys are distributed honestly. In other words, any PACCE instance Π^s_U that accepts in this game must have used the domain name S and the corresponding public key pk_S that was generated honestly by the simulator $\mathcal{B}$.

Game G_2. In this game, the simulator $\mathcal{B}$ acts as in game G_1, except it makes use of a challenger $\mathcal{C}^{\pi}$ for the ACCE protocol π to simulate the ACCE portions of the combined protocol. In particular, $\mathcal{B}$ will maintain a one-to-one mapping between sessions Π^s_U of the PACCE protocol and sessions $\widehat{\Pi^s_U}$ of the ACCE protocol. The simulator $\mathcal{B}$ receives public keys pk_U from $\mathcal{C}^{\pi}$, activates the PACCE adversary $\mathcal{A}$, and responds to the queries by $\mathcal{A}$ as in game G_1, except that its replies to $\mathsf{Send}^{\mathtt{pre}}$, $\mathsf{RevealSK}$, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$ oracles are obtained by first forwarding those queries to $\mathcal{C}^{\pi}$ and passing on the responses back to $\mathcal{A}$. This offers perfect simulation of the ACCE part of the combined protocol so that

$$\mathsf{Adv}^{\mathsf{G}_1}_{\mathcal{A},\Gamma_S(\xi,\pi)} = \mathsf{Adv}^{\mathsf{G}_2}_{\mathcal{A},\Gamma_S(\xi,\pi)} \ . \tag{13}$$

Analysis of game G_2. Suppose $\mathcal{A}$ outputs a tuple (U, s, b') and wins in the PACCE experiment. According to Definition 1 $\mathcal{A}$ can win if one of the following two conditions is satisfied: either $\mathcal{A}$

breaks the authentication in the pre-accept phase of the PACCE protocol as per condition 1 or $\mathcal{A}$ breaks the authenticated encryption in the post-accept phase as per condition 2.

First assume $\mathcal{A}$ wins by condition 1. In this case there exists an instance Π_U^s that has accepted without having a partnered instance $\Pi_{U'}^{s'}$. Game G_1 guarantees that for any Π_U^s that accepts in PACCE there must be an instance $\Pi_{U'}^{s'}$ with matching tPAKE transcript and input tag (domain name S). Hence, the only possibility for $\mathcal{A}$ to win by condition 1 in G_2 is when the ACCE transcripts of Π_U^s and $\Pi_{U'}^{s'}$ do not match. However, since both parties must have used an honestly generated public key pk_S as tag (this is implied by G_1) any mismatch in their ACCE transcript parts would contradict the assumed security of the underlying ACCE protocol.

Assume now that $\mathcal{A}$ wins by condition 2. In this case the output (U, s, b') of $\mathcal{A}$ would also break the authenticated encryption security of the ACCE protocol. Hence, if $\mathcal{B}$ outputs (U, s, b') to the ACCE challenger $\mathcal{C}^\pi$, it will win the ACCE experiment whenever $\mathcal{A}$ wins the PACCE experiment. Thus,

$$\mathsf{Adv}^{\mathsf{G}_2}_{\mathcal{A},\Gamma_S(\xi,\pi)} = \mathsf{Adv}^{\mathsf{ACCE}}_{\mathcal{B},\pi} \quad . \tag{14}$$

Final result. Combining equations (11)–(14),

$$\mathsf{Adv}^{\mathsf{PACCE}}_{\mathcal{A},\Gamma_S(\xi,\pi)} \leq \mathsf{Adv}^{\mathsf{tPAKE}}_{\mathcal{B},\xi} + \mathsf{Adv}^{\mathsf{ACCE}}_{\mathcal{B},\pi} \quad ,$$

and we obtain the security of the combined construction. □

7 tSOKE: A tag-based password authentication protocol with optional session key derivation

To achieve secure password authentication, we make use of a tagged version tSOKE of the Simple Open Key Exchange (SOKE) variant [1], which in turn builds on the Simple PAKE (SPAKE) protocol [5], standardized in [25].

The tagged version tSOKE, used in our implementations, is specified in Figure 3. The system parameters consist of an elliptic curve group (which we fix to be the NIST P-192 group [34]) and a second generator G' constructed verifiably at random. We use PBKDF2 for password hashing and SHA-256 as a (concrete) hash function, modeled as a random oracle.

In the registration stage, the user selects a password $\mathsf{pw}_{C,S}$, hashes it along with a random salt, using the PBKDF2 iterated construction, and gives salt, counter c, and the password hash h to the server to store. In the login stage, the user inputs $\mathsf{pw}_{C,S}$, reconstructs the password hash h using PBKDF2, and the client and server together employ a masked Diffie–Hellman key exchange to compute the session key k and demonstrate mutual knowledge of the hashed password, along with a tag τ.

In general, the main difference between tSOKE and SOKE is that the tag τ is used in the computation of the pre-master key $pmk \leftarrow \mathsf{SHA\text{-}256}(\mathsf{id}_C, h, \tau, X, Y^*, Z)$ where id_C is the client's username, h is the password hash, τ is the tag, X and Y^* are exchanged (masked, in the case of Y^*) Diffie–Hellman public keys, and Z is the Diffie–Hellman shared secret. Notably, the tag need not be used in the masking of the Diffie–Hellman public key; the masking operation is used to protect the secret password from offline dictionary attacks, and since the tag need not be kept secret, it only needs to appear in the computation of pmk. Recall that parties subsequently demonstrate knowledge (and check for equality) of the pmk to each other by exchanging authentication values A_1 and A_2.

Our PACCE constructions 1 and 2 rely only on the password authentication property of tSOKE, for which the mutual authentication aspects are used. In particular, the derivation of the session

System parameters

Elliptic curve group nistp192 with generator G of order n
Second generator G' constructed verifiably at random with $\langle G \rangle = \langle G' \rangle$
$G' =$ (`0x8da36f68628a18107650b306f22b41448cb60fe5712dd57a`,
`0x1f64a649852124528a09455de6aad151b4c0a9a8c2e8269c`)

Registration stage (takes place over a secure channel)

	Client C		**Server** S
1.	Enter username id_C.		
2.	Enter password $\mathsf{pw}_{C,S}$.		
3.	$\mathsf{salt} \leftarrow \{0,1\}^{128}$.		
4.	Choose iteration counter $c \in \mathbb{N}^+$.		
5.	$h \leftarrow \mathsf{PBKDF2}(\mathsf{SHA\text{-}256}, \mathsf{pw}_{C,S}, \mathsf{salt}, c, 256)$		
6.		$\xrightarrow{\mathsf{id}_C, \mathsf{salt}, c, h}$	
7.			Store salt, c, h for id_C.

Login stage

	Client C		**Server** S
	Input: tag τ		Input: tag τ
1.	Enter username id_C.		
2.	Enter password $\mathsf{pw}_{C,S}$.		
3.	$x \leftarrow_R \{2, \ldots, n-1\}$		
4.	$X \leftarrow xG$		
		$\xrightarrow{\mathsf{id}_C, X}$	
5.		`tSOKE.ClientExchange`	
6.			Look up salt, c, h for id_C.
7.			$y \leftarrow_R \{2, \ldots, n-1\}$
8.			$Y \leftarrow yG$
9.			$Y^* \leftarrow Y + (h \bmod n)G'$
		$\xleftarrow{\mathsf{salt}, c, Y^*}$	
10.		`tSOKE.ServerExchange`	
11.	$h \leftarrow \mathsf{PBKDF2}(\mathsf{SHA\text{-}256}, \mathsf{pw}_{C,S}, \mathsf{salt}, c, 256)$		
12.	$Y \leftarrow Y^* - (h \bmod n)G'$		
13.	$Z \leftarrow xY$		$Z' \leftarrow yX$
14.	$pmk \leftarrow \mathsf{SHA\text{-}256}(\mathsf{id}_C, h, \tau, X, Y^*, Z)$		$pmk \leftarrow \mathsf{SHA\text{-}256}(\mathsf{id}_C, h, \tau, X, Y^*, Z)$
15.	$k \leftarrow \mathsf{SHA\text{-}256}(pmk, \text{“skey”})$		$k \leftarrow \mathsf{SHA\text{-}256}(pmk, \text{“skey”})$
16.	$A_1 \leftarrow \mathsf{SHA\text{-}256}(pmk, \text{“auth1”})$		
		$\xrightarrow{A_1}$	
17.		`tSOKE.ClientFinished`	
18.			Abort if $A_1 \neq \mathsf{SHA\text{-}256}(pmk, \text{“auth1”})$
19.			$A_2 \leftarrow \mathsf{SHA\text{-}256}(pmk, \text{“auth2”})$
		$\xleftarrow{A_2}$	
20.		`tSOKE.ServerFinished`	
21.	Abort if $A_2 \neq \mathsf{SHA\text{-}256}(pmk, \text{“auth2”})$		
22.	Accept		Accept

Figure 3: tSOKE protocol registration and login stages

key k can be omitted by participants in those PACCE constructions. In contrast, our PACCE construction 3 requires tSOKE participants to compute the session key k. The following theorem proves that tSOKE is a secure tPAKE protocol, and by Lemma 1 also a secure tPAuth protocol.

Theorem 4. tSOKE *is a secure tPAKE protocol, assuming that* SHA-256 *behaves like a random oracle.*

Proof. We prove this theorem by closely following the game-hopping proof of [1, Theorem 5.1] which states that assuming that SHA-256 is modeled as a random oracle the original SOKE protocol is a secure PAKE protocol according to the security definitions from [4]. There are two main issues with regard to the proof of [1, Theorem 5.1] that we need to address in order to prove that tSOKE is a secure tPAKE protocol according to Definition 4.

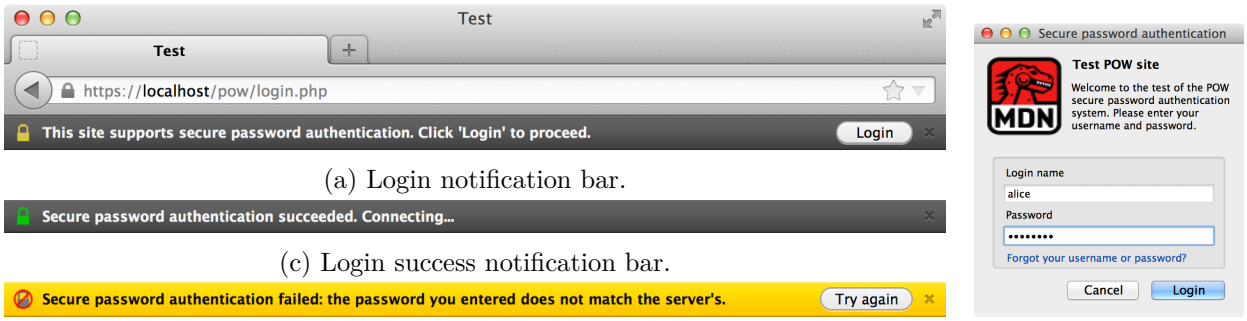
First, in tSOKE there is no derivation of the secret `KeyBlock` that was computed in SOKE using a pseudo-random function `PRF`, while the derivation of the `MasterSecret`, denoted by k in Figure 3 is still in place. The security of these derivation steps for SOKE in the proof of [1, Theorem 5.1] was addressed in two games, namely G_2 that showed that collisions for the two derived secrets may happen only with negligible probability according to the birthday paradox and G_6 that addressed the security of the pseudo-random function `PRF` by replacing it with the random function. Since game G_2 also dealt with collisions of the protocol transcripts we still need this game to prove the security of tSOKE but the (negligible) probability distance to the game G_1 becomes now even smaller (by the amount that corresponds to the collision probability for the two secrets `MasterSecret` and `KeyBlock`). In contrast, the game G_6 is now modified to take into account that k is derived using a random oracle (rather than a `PRF`). This change has no further impact since random oracle was used in SOKE to derive the `PreMasterSecret`, denoted by pmk in Figure 3.

The second and perhaps more important difference is that we additionally need to argue that a successful tSOKE session between a client and a server guarantees the equality of input tags τ for both parties in case any of them accepts (as required by Definition 4). Observe that the input tag τ is processed in tSOKE as an additional input to the hash function SHA-256 in the computation of pmk. The original game G_2 in the proof of [1, Theorem 5.1] that treats SHA-256 as a random oracle and excludes collisions using the birthday paradox therefore also implies the required equality of tags. □

8 PACCE implementations for desktop and mobile browsers

As an important motivation for our modular protocol design was the ability to modularly implement the protocol, we produced two prototypes to demonstrate this. For the tag-based password authentication protocol, we used the previously described tSOKE, a tag-based version of the SOKE protocol [1], a highly efficient Diffie–Hellman based PAKE. We also used the NIST P-192 elliptic curve group [34].

The client side of the protocol was implemented in two variants. The first version is an extension for Mozilla Firefox (320 lines of custom JavaScript, plus libraries) and the second implementation is an Android application (770 lines of custom Java and 182 lines of XML, plus libraries). The server side of the protocol was implemented as a PHP application (210 lines of custom PHP code, plus libraries) that can run on an Apache web server. No modifications to the source code of the underlying web browser (Firefox or mobile browser) or the underlying web server (Apache with OpenSSL) were required—in particular, we did not have to alter the SSL/TLS implementation and we did not have to recompile Firefox, the mobile browser or Apache. Since the mechanism that the server code uses to obtain the certificate of the TLS connection is an Apache CGI (Common



(a) Login notification bar.

(c) Login success notification bar.

(d) Login failure notification bar.

(b) Login dialog box.

Figure 4: User interface for Mozilla Firefox extension.

Gateway Interface) variable, any server-side language would work, not just PHP. In the following we describe the two different clients and the server implementation more in detail. The implementation is available online under an open-source license at https://www.franziskuskiefer.de/pow/.

8.1 PACCE implementation for desktop browsers

8.1.1 Client: Firefox extension

The client-side Firefox extension is written in JavaScript and uses an existing Firefox API to obtain the certificate of the TLS connection. The client implementation of our protocol (excluding underlying cryptographic primitives) is just 320 lines of JavaScript code. Cryptographic operations can be done in either pure JavaScript (relying on about 1400 lines of code from Wu's JavaScript elliptic curve cryptography and big integer arithmetic implementation[2] and about 6KB of minified JavaScript from the Stanford Javascript Crypto Library[3] for the PBKDF2 algorithm) or can make use of native C OpenSSL libraries using Firefox's js-ctypes API[4].

When the extension detects (using an appropriate triggering mechanism; see Section 8.4) a page that supports the protocol, it displays a notification bar that secure password authentication is supported (Figure 4(a)). The user then clicks on the "Login" button in the notification bar to bring up the password entry dialog box (Figure 4(b)). Note that the notification bar is displayed using Firefox API for notifications, similar to how alerts are rendered for missing plugins. By using the standard notification mechanism, we provide a trusted UI path to the notification bar, and then, through the login button on that bar, a trusted UI path to the dialog box, somewhat mitigating concerns about the difficulty of providing a trusted UI path in browser-based secure password authentication [19, 20]. The status of the mutual authentication is displayed in the notification bar (Figures 4(c), (d)); if successful the browser is redirected to the URL indicated by the server.

At present, Firefox is the only web browser we examined whose extension APIs offer partial implementation of the channel bindings for TLS from RFC 5929 [9], providing access to the certificate of the page's TLS connection. The APIs for Google Chrome and Apple Safari extensions do not seem to permit this ability so far, nor does the API for Microsoft Internet Explorer browser toolbars. However, our modular approach is still valid, in that Chrome, Safari, and IE would only need to implement the recommendations from [9] such that our extension could do the rest of the

[2] http://www-cs-students.stanford.edu/~tjw/jsbn/
[3] http://crypto.stanford.edu/sjcl/
[4] https://developer.mozilla.org/en-US/docs/Mozilla/js-ctypes

protocol, rather than requiring the full protocol be implemented within the core browser source code as many other approaches require. Alternatively an approach similar to the mobile architecture with a standalone application could be implemented, which does not require any such API (cf. Section 8.2).

Branding Our prototype allows the server to specify some limited "branding" customizations to the login dialog box, including displaying a logo and explanatory text, as can be seen in Figure 4(b). A common objection to server-specified branding is that the protocol becomes insecure due to phishing attacks: while it is true that an attacker could put in a different logo or text in Figure 4(b), the attacker *gains nothing* in doing so: the protocol cryptographically protects the password, even when the user's browser runs the protocol with attacker's server. At best, the attacker can interrupt communication, but will gain no information. Our limited branding does not give the attacker enough power to completely spoof the user interface and trick the user into using the attacker's own dialog box, due to the trusted UI path via the browser notification bar.

In addition to branding, personalization can be used to help prevent UI spoofing attacks. We give more details on personalization and its benefits in Section 8.2 where it is also implemented. (See Section 9 for further discussion of spoofing attacks.)

8.1.2 Server: PHP application

The server-side PHP application uses an existing Apache CGI variable (`$_SERVER['SSL_SERVER_CERT']`) to obtain the certificate of the TLS connection. (If this variable is not available the certificate can also be hard coded or read from the certificate file directly.) The server implementation of our protocol is just 210 lines of PHP code. Cryptographic operations can be done either in PHP (relying on Danter's PHP elliptic curve cryptography implementation[5]) using pre-packaged PHP extensions for big integer arithmetic, or can make use of native C OpenSSL libraries.

8.2 PACCE implementation for mobile browsers based on Android application

8.2.1 Client: Android application

The second prototype is implemented as an Android application according to construction 3 and allows the use of PACCE with any browser on Android devices. Figure 1b depicts the high-level architecture of the mobile implementation of construction 3. We start with a description of the protocol used by the prototype, detailed in Figure 6. The message flow in the figure splits the client into the browser application and the app used to perform the cryptographic operations. We assume that communication between those two applications is private, i.e. no attacker is able to modify messages or eavesdrop on the communication.

The user is presented with a common HTML button to trigger the login mechanism when visiting a website. When requesting the login form a special URL with setup parameters to open the Android application is returned instead of an HTML login form. The parameters include in particular the server domain that is used by the application to perform the tSOKE protocol. The application presents a trusted login form to the user who enters his username and password and starts the login process.

Similar to the Firefox extension the Android application uses provided branding information (icon and URL) to enhance user experience by displaying them in the action bar on top of the login form. Android application and server now run the tSOKE protocol as described before.

[5] https://github.com/mdanter/phpecc/

The cryptographic operations in the Android application are implemented using SpongyCastle[6], a repackage of BouncyCastle[7] for Android. Client and server compute the same session key if and only if they use the same password and domain in tSOKE. The session key, generated with tSOKE, is used to open the browser with the same domain it used to perform the tSOKE protocol. This authenticates the browser session to the server. The server accepts the browser session if and only if the browser is the one used to open the login application and if the browser is able to provide the correct session key. (Common cookie techniques can be used to ensure security of this handover.)

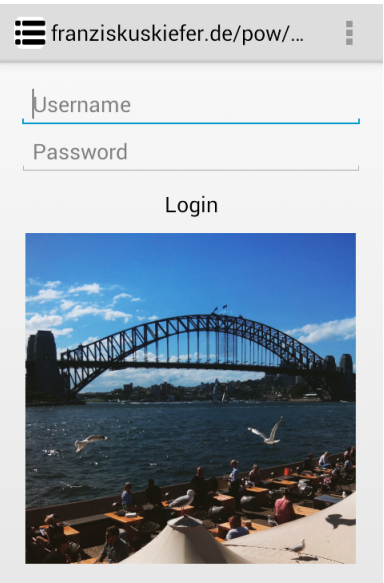


Figure 5: User interface of Android Application with personalization.

Personalization against UI spoofing As mentioned previously a trusted path and branding is not enough to tackle more sophisticated UI spoofing attacks. A malicious website may for example rebuild the user interface of the trusted input (Firefox extension or Android application) to fool the user in assuming he is using a trusted input and secure login mechanism. To tackle UI spoofing and usage of malicious applications, the Android application deploys an additional mechanism, personalization. On the first start of the application the user is prompted to choose an image from phone or online storage, which is then shown below the login form to personalize the application (see Figure 5). Every subsequent use of this application is secured by this personalized image that can not be easily spoofed by an attacker. In particular, users should not proceed with the login process if they do not see the image they have chosen on first use.

8.2.2 Server: PHP application

The server is essentially the same as the previously described PHP server with some minor modifications. First, the server computes authentication tokens and the session key k in tSOKE (recall that construction 3 uses tSOKE as a tPAKE protocol). Further, as described in construction 3 the tag used in tSOKE is the server's domain name. The session key computed in tSOKE is used by the application to call the server via the browser and thus allow the server to verify the browser session.

[6] https://rtyley.github.io/spongycastle/
[7] http://www.bouncycastle.org/java.html

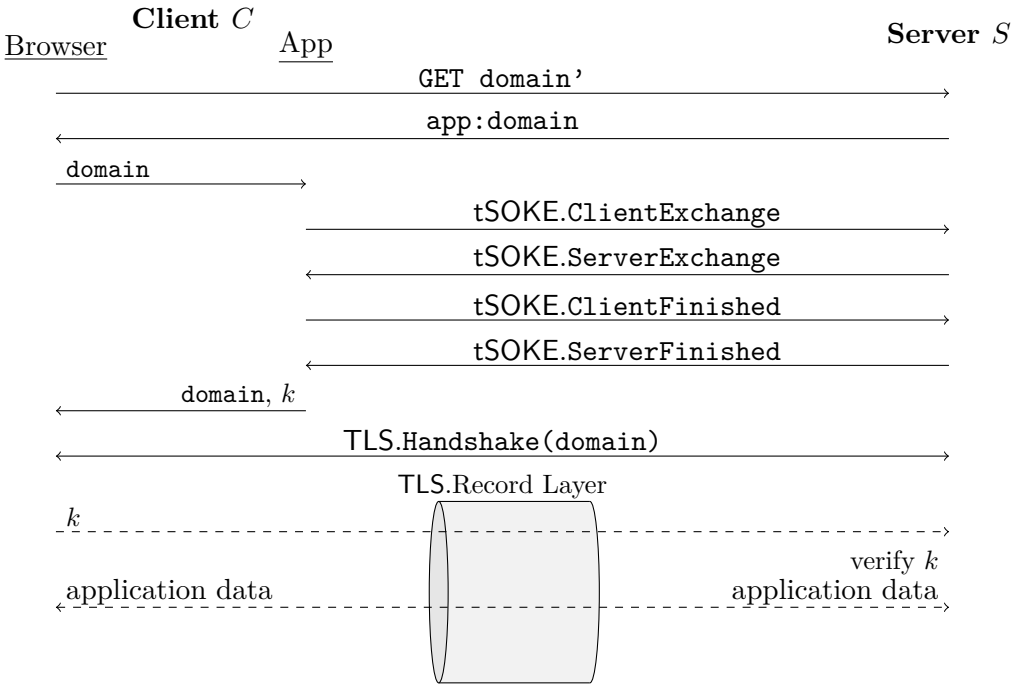


Figure 6: Construction 3 – Protocol message diagram for TLS with tag-based password-authenticated session key k for mobile browsers using tSOKE and server's domain name as a tag.

This requires only minor changes to the previously described PHP server in order to generate the session key in addition to the authentication tokens.

8.3 Performance analysis

In Table 1, we report timings for our implementations. Timings reported are an average of 10 timings, with standard deviation. The total runtime of the protocol includes the network latency for the communication from a corporate internet connection in the UK where the client machines were located and a server in Germany. The average ping time on the network from the laptop was 27.45 ms (stdev. 5.24) and 280.5 ms (stdev. 16.75) from the mobile phone.

We report three different sets of timings: "cross-platform" timings of the Firefox extension, using pure Javascript on the client side; "native" timings of the Firefox extension, using calls to OpenSSL for cryptographic operations on the client side; and "Java" timings of the Android application, using pure Java on the client side. All timings are performed against a server, implemented in PHP with built-in GMP libraries, accounting for 62.4 ms (stdev. 7.9) in the total runtime.

The average total runtime of the Firefox extension from when the user clicked "Login" after entering their password until the protocol completed was 306.68 ms (stdev. 17.51) using Javascript code, and 180.68 ms (stdev. 13.44) using native code. On the mobile device with Android OS the average total runtime from when the user clicked "Login" after entering their password until the protocol completed was 2433.8 ms (stdev. 49.33).[8]

[8]Note that the slower performance, compared to the desktop applications, is due to a slow implementation of NIST P-192 curve in the library used, which requires about 600 ms per multiplication on Android. At the time of writing, SpongyCastle library was the only available ECC library for Android.

Operation	Pure Javascript	Native C	Android (Java)
Client cryptographic computations	123.79 ± 11.13	7.79 ± 2.79	1828.1 ± 42.76
Total runtime†	306.68 ± 17.51	180.69 ± 13.44	2433.8 ± 49.33

Laptop Client: Arch Linux, Kernel 3.18.6-1-ARCH, Firefox 35.0.1, OpenSSL 1.0.2-1, 2.4 GHz Intel Core 2Duo (P8600), 8 GB of RAM

Mobile Client: Android 4.1.1 with Kernel 3.4.0, 1 GHz MT6577, 1 GB of RAM

Server: Apache 2.4.12-2, PHP 5.6.6-1 with fpm, MariaDB 10.0.16-1, OpenSSL 1.0.2-1, 2.4 GHz Intel Xeon (E5645) 1 Core, 1 GB of RAM, KVM simulated.

Network: Corporate internet connection, ping time 27.45 ± 5.24ms (laptop), 280.5 ± 16.75ms (mobile phone)

† : includes network time and 62.4 ± 7.9 server cryptographic computations

Table 1: Average runtime in ms ($\pm$ standard deviation) of extension using cross-platform Javascript cryptographic code and native C (OpenSSL) cryptographic code, Android application using a pure Java implementation

Our native cryptographic code is comparable to Dacosta et al.'s reported performance of DVCert on laptops [18]. Our protocol implementation includes a variety of operations beyond cryptographic computations, so the total runtime is greater than the sum of cryptographic runtime and communication time. We do not compare the timings to the runtime of password authentication based on an HTML form over TLS since this introduces only a negligible overhead to the overall page-load time.

8.4 Integration with HTTP/HTML

The exact form of integration with HTTP or HTML for web applications is a question best left to the web standards community. Our protocol can be integrated in several ways. It could be implemented as a new HTTP Authentication method [22], with protocol messages sent in HTTP headers. It could also be implemented within the HTML, with the initial `tPAuth.ServerHello` message delivered as an HTML microdata object, and the remaining messages transmitted over asynchronous HTTP POST requests with responses formatted for example as JSON messages combined with standard HTTP cookie-based session management. For our prototypes, we chose the latter approach, but the former approach would work equally well. Two benefits of the HTML-based approach are that it avoids the "logout problem" of HTTP authentication, in which there is no standardized mechanism to terminate transmission of HTTP authentication headers, and that it is compatible with cookie-based state management.

9 Discussion

Although PAKE protocols have been known in the literature since their invention in 1992, they have seen almost no deployed adoption for user authentication in real-world protocols and implementations. One exception is the use of the socialist millionaires' protocol in the Off-the-Record Messaging (OTR) protocol for private instant messaging [8]. Engler et al. [20] recently identified several challenges—divided into two classes, user interface and deployment challenges—to adopting cryptographic protocols for password authentication in the web. It has also been noted that the myriad patents related to PAKE have had a negative impact on adoption [2].

Deployment challenges Our modular architecture may address certain deployment challenges. Engler et al. ask "What is the appropriate layer in the networking stack to integrate PAKE protocols?" They compare two proposed options: TLS-SRP [41] and an earlier draft of the HTTPS-PAKE approach of Oiwa et al. [37]. Adding SRP as a TLS ciphersuite has benefits in that, once implemented, TLS-SRP allows multiple applications to use the same TLS implementation. But many drawbacks are identified by Engler et al., including: (i) the need to integrate the application layer with the TLS layer on both the client side (necessitating a complex API between the TLS library and the web browser, for example) and on the server side (which could negatively affect the ability of HTTPS load balancers to terminate TLS connections and then hand them off to web application servers); and (ii) the difficulty of supporting multiple authentication realms within the same domain.

HTTPS-PAKE, running as an HTTP authentication mechanism at the application layer, avoids both of these problems. The version of HTTPS-PAKE reviewed by Engler et al. did not have cryptographically strong binding between the two protocols and thus could not prevent man-in-the-middle attacks, but later revisions addressed that issue. Our modular approach avoids the problems that Engler et al. identify for TLS-SRP.

Our approach also better handles the transition from unauthenticated encrypted browsing to authenticated encrypted browsing: a user may browse an HTTPS site for a while before logging in; with TLS-SRP, a new TLS connection is required (and the mechanism for triggering a new TLS connection is unclear); it is much easier to trigger the authentication at the application layer when it is required.

Our modular architecture is particularly simple to deploy as it only requires the user to install a Firefox extension or Android application and the server is a standalone PHP application that can be used as authentication server in conjunction with regular web applications.

User interface challenges Engler et al. [20] identify several user interface challenges. We do not aim to fully solve all of these challenges in our prototype, as demonstrating a convincing solution to these challenges requires critical examination by usability experts and appropriate user studies. Nonetheless, we have endeavoured to follow some best practices that may at least partially address the identified UI challenges.

It is essential for the security of PAKE protocols that the user always enter their password into a secure dialog box. If the entry mechanism can be spoofed by an attacking website, then the user could be tricked into entering their password directly into a textfield controlled by the attacker. Thus, there must be a *trusted path* to the dialog box in the UI. This is usually achieved by placing the password entry visibly in the parts of the window that make up the browser UI, such as the location bar, rather than the page content. In our prototype, we follow this practice by using Firefox's notification bar. It has been suggested that permitting users to customize notification bars helps to reduce spoofing attacks [19]. This has been explored further in the Android application by allowing the user to choose a private picture to personalize the application (as discussed earlier).

The second and third of Engler et al.'s UI challenges are about how to train users to use the system in the first place, and how to communicate failures to users in a way that does not encourage them to fall back on insecure methods. Both of these remain a challenge for usability designers, though again, delivering failure notifications via Firefox's or Android's trusted path for notifications may provide some benefit. A procedure for resetting forgotten password securely remains an open challenge both in practice and in theory and is outside the scope of our goals. One particular challenge worth mentioning here is that with the proposed architecture the system is unable to distinguish between a login error due to a wrong username or password and an error due to a

wrong tag inside the PACCE protocol, i.e. an actual attack. This may confuse users. To solve this problem, i.e. to distinguish between an attack and a wrong username or password, an additional authentication token may be computed similar to auth1 but without tag τ. This allows the server to distinguish the two aforementioned cases and return an appropriate error message to the client.

The final challenge noted by Engler et al. is on how to allow website designers to customize and brand the login dialog without compromising security; it has been suggested that lack of customization and branding was a contributing factor to the lack of adoption of HTTP basic and digest authentication. Our prototype allows the server to provide a few customizations to the login dialog box, including a logo and some explanatory text, as shown in Figures 4(b) and 5. While an attacker could use stolen images, the benefit to the attacker is minimal since the password entry will be cryptographically protected.

Adoption challenges A final challenge for any new security technology is facilitating widespread adoption. Such protocols see a "network effect": it is only useful for a client Alice to use the technology if there are many Bobs who support it, and vice versa. In the end, any secure password authentication technology will be most successful once built into all major mobile and desktop web browsers and web application frameworks. In the meantime, the modular approach in this paper is suitable for gradual deployment. For example, an organization can internally standardize on the use of this approach by deploying an extension or application to all of its users without needing to wait for the browser vendor to support the protocol. The more adoption via extension and application, the more evidence for interest in the technology, and the greater incentive for vendors to provide a native implementation.

References

[1] Abdalla, M., Bresson, E., Chevassut, O., Möller, B., Pointcheval, D.: Provably secure password-based authentication in TLS. In: Lin, F.C., Lee, D.T., Lin, B.S., Shieh, S., Jajodia, S. (eds.) ASIACCS 06. pp. 35–45. ACM Press (Mar 2006)

[2] Abdalla, M., Bresson, E., Chevassut, O., Möller, B., Pointcheval, D.: Strong password-based authentication in TLS using the three-party group Diffie–Hellman protocol. International Journal of Security and Networks 2(3/4), 284–296 (2007)

[3] Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Efficient two-party password-based key exchange protocols in the UC framework. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 335–351. Springer, Heidelberg (Apr 2008)

[4] Abdalla, M., Fouque, P.A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (Jan 2005)

[5] Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (Feb 2005)

[6] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J.A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., Zimmermann, P.: Imperfect forward secrecy: How Diffie–Hellman fails in practice (May 2015), https://weakdh.org/

[7] AIST Research Center for Information Security: Mutual authentication protocol for HTTP, https://www.rcis.aist.go.jp/special/MutualAuth

[8] Alexander, C., Goldberg, I.: Improved user authentication in Off-The-Record messaging. In: Yu, T. (ed.) ACM Workshop on Privacy in Electronic Society (WPES) 2007. pp. 41–47. ACM Press (2007)

[9] Altman, J., Williams, N., Zhu, L.: Channel Bindings for TLS. RFC 5929 (Proposed Standard) (Jul 2010), http://www.ietf.org/rfc/rfc5929.txt

[10] Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (May 2000)

[11] Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy. pp. 72–84. IEEE Computer Society Press (May 1992)

[12] Bergsma, F., Dowling, B., Kohlar, F., Schwenk, J., Stebila, D.: Multi-ciphersuite security of the secure shell (SSH) protocol. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 14. pp. 369–381. ACM Press (Nov 2014)

[13] Beurdouche, B., Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Kohlweiss, M., Pironti, A., Strub, P.Y., Zinzindohoue, J.K.: A messy state of the union: Taming the composite state machines of TLS. In: 2015 IEEE Symposium on Security and Privacy. pp. 535–552. IEEE Computer Society Press (May 2015)

[14] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., Strub, P.Y.: Triple handshakes and cookie cutters: Breaking and fixing authentication over TLS. In: 2014 IEEE Symposium on Security and Privacy. pp. 98–113. IEEE Computer Society Press (May 2014)

[15] Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (May 2000)

[16] Brzuska, C., Smart, N.P., Warinschi, B., Watson, G.J.: An analysis of the EMV channel establishment protocol. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 373–386. ACM Press (Nov 2013)

[17] Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (May 2005)

[18] Dacosta, I., Ahamad, M., Traynor, P.: Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In: Foresti, S., Yung, M., Martinelli, F. (eds.) ESORICS 2012. LNCS, vol. 7459, pp. 199–216. Springer, Heidelberg (Sep 2012)

[19] Dhamija, R., Tygar, J.D.: The battle against phishing: Dynamic security skins. In: Cranor, L.F., Zurko, M.E. (eds.) Symposium on Usable Privacy and Security (SOUPS) 2005. pp. 77–88. ACM Press (2005)

[20] Engler, J., Karlof, C., Shi, E., Song, D.: Is it too late for PAKE? In: Web 2.0 Security and Privacy (W2SP) 2009 (2009), http://w2spconf.com/2009/papers/s4p1.pdf

[21] Fleischhacker, N., Manulis, M., Azodi, A.: A Modular Framework for Multi-Factor Authentication and Key Exchange. In: Security Standardisation Research (SSR 2014). LNCS, vol. 8893, pp. 190–214. Springer (2014)

[22] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., Stewart, L.: HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard) (Jun 1999), http://www.ietf.org/rfc/rfc2617.txt, updated by RFC 7235

[23] Giesen, F., Kohlar, F., Stebila, D.: On the security of TLS renegotiation. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 387–398. ACM Press (Nov 2013)

[24] Hao, F., Ryan, P.Y.A.: Password authenticated key exchange by juggling. In: Security Protocols Workshop. LNCS, vol. 6615, pp. 159–171. Springer (2008)

[25] IEEE P1363.2: Standard specifications for password-based public-key cryptographic techniques (2008), http://grouper.ieee.org/groups/1363/passwdPK/

[26] International Organization for Standardization (ISO): ISO/IEC 11770-4: Information technology — security techniques — key management — part 4: Mechanisms based on weak secrets (May 2006), http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=39723

[27] ITU-T X.1035: Password-authenticated key exchange (PAK) protocol (2007), http://www.itu.int/rec/T-REC-X.1035-200702-I/en

[28] Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: Generic compilers for authenticated key exchange. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 232–249. Springer, Heidelberg (Dec 2010)

[29] Jager, T., Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DHE in the standard model. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 273–293. Springer, Heidelberg (Aug 2012)

[30] Kohlar, F., Schäge, S., Schwenk, J.: On the security of TLS-DH and TLS-RSA in the standard model. Cryptology ePrint Archive, Report 2013/367 (2013), http://eprint.iacr.org/2013/367

[31] Krawczyk, H., Paterson, K.G., Wee, H.: On the security of the TLS protocol: A systematic analysis. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 429–448. Springer, Heidelberg (Aug 2013)

[32] Kwon, T.: Authentication and key agreement via memorable passwords. In: NDSS 2001. The Internet Society (Feb 2001)

[33] LaMacchia, B.A., Lauter, K., Mityagin, A.: Stronger security of authenticated key exchange. In: Susilo, W., Liu, J.K., Mu, Y. (eds.) ProvSec 2007. LNCS, vol. 4784, pp. 1–16. Springer, Heidelberg (Nov 2007)

[34] National Institute of Standards and Technology: Recommended elliptic curves for federal government use (July 1999), http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf

[35] Oiwa, Y., Takagi, H., Watanabe, H., Suzuki, H.: PAKE-based mutual HTTP authentication for preventing phishing attacks. In: Maarek, Y., Nejdl, W. (eds.) Proc. 18th International World Wide Web Conference (WWW) 2009. pp. 1143–1144. ACM (2009), http://www2009.org/proceedings/pdf/p1143.pdf

[36] Oiwa, Y., Watanabe, H., Takagi, H.: PAKE-based mutual HTTP authentication for preventing phishing attacks (November 2009), http://arxiv.org/abs/0911.5230

[37] Oiwa, Y., Watanabe, H., Takagi, H., Kihara, B., Ioku, Y., Hayashi, T.: Mutual authentication protocol for HTTP (June 2012), internet-Draft. http://tools.ietf.org/html/draft-oiwa-http-mutualauth-12

[38] Rescorla, E.: Keying Material Exporters for Transport Layer Security (TLS). RFC 5705 (Proposed Standard) (Mar 2010), http://www.ietf.org/rfc/rfc5705.txt

[39] Schechter, S.E., Dhamija, R., Ozment, A., Fischer, I.: The emperor's new security indicators. In: 2007 IEEE Symposium on Security and Privacy. pp. 51–65. IEEE Computer Society Press (May 2007)

[40] Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., Cranor, L.F.: Crying wolf: An empirical study of SSL warning effectiveness. In: USENIX Security 2009 (2009), http://www.usenix.org/events/sec09/tech/full_papers/sunshine.pdf

[41] Taylor, D., Wu, T., Mavrogiannopoulos, N., Perrin, T.: Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054 (Informational) (Nov 2007), http://www.ietf.org/rfc/rfc5054.txt

[42] Wu, T.D.: The secure remote password protocol. In: NDSS'98. The Internet Society (Mar 1998)