An extended abstract version of this work appears in: Liqun Chen, editor, *Proc. 1st International Conference on Research in Security Standardisation (SSR) 2014*, *LNCS*. Springer, 2014. This is the full version.

Secure modular password authentication for the web using channel bindings

Mark Manulis[1] Douglas Stebila[2] Nick Denham[2]

[1] *Surrey Centre for Cyber Security, University of Surrey, Guildford, UK*

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

mark@manulis.eu stebila@qut.edu.au n.denham@qut.edu.au

September 19, 2014

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 protocol, where the two protocols are bound together using either the transcript of the secure channel's handshake or the server's certificate, results in a secure PACCE protocol. Our prototype based on TLS is available as a cross-platform client-side Firefox browser extension and a server-side web application which can easily be installed on deployed web browsers and servers.

Keywords: password authentication, Transport Layer Security, channel binding

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 [SDOF07, SEA+09]. Hypertext Transport Protocol (HTTP) basic and digest access authentication [FHBH+99] 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.

1.1 Password-authenticated key exchange (PAKE)

Password-authenticated key exchange (PAKE) protocols, which were introduced by Bellovin and Merritt [BM92], and the security of which was formalized in several settings [BPR00, CHK+05, ACCP08], 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 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 of PAKE, and the presence of a variety of existing protocols in the academic literature and in standards [Int06, ITU07, IEE08], 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), patents on proposed standards such as the Secure Remote Password (SRP) protocol [Wu98], lack of agreement on the appropriate layer within the networking stack for the integration of PAKE [EKSS09], 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. SRP has been standardized as a TLS ciphersuite [TWMP07] and has several reference implementations but none in major web browsers or servers. Abdalla et al. [ABC+06] proposed the provably secure Simple Open Key Exchange (SOKE) ciphersuite, which uses a variant of the PAKE protocol from [AP05] that is part of the IEEE-P1363.2 standard [IEE08]. The J-PAKE protocol [HR08] 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 any modifications to the TLS specification or implementation; in particular, without proposing any new TLS ciphersuites or changing any of the steps of TLS handshake protocol.

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), or even the server's certificate. A recent standard [AWZ10] 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.

The high-level approach of running PAKE at the application level is given in Figure 1. Using PAKE at the application-level supplements, rather than replaces, the use of public key certificates.

Several recent works have proposed protocols of this form. Oiwa et al. [OTWS09, OWT09, OWT+12, AIS] published an Internet-Draft that employs an ISO-standardized PAKE protocol (KAM3 [Int06, §6.3],[Kwo01]) 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. [DAT12] proposed the DVCert protocol which aims to achieve direct validation of the TLS server certificates by using a modification of the protocol from [BMP00] for secure server-to-client password-based authentication. Dacosta et al. used an automated cryptographic protocol verifier, ProVerif, to demonstrate that their protocol does not leak password information without addressing any further security properties of secure channels. In particular, the analysis carried out in [DAT12] is insufficient for showing the actual benefit of using PAKE protocols to strengthen the security of the TLS channel: Dacosta et al. simply show that the protocol does not leak password information, which is not surprising since they build on a PAKE protocol which already did not leak information. Rather, the security goals of the overall channel must be fully modelled to provide a complete analysis. This is where our model for PACCE fills the gap: it explains the expected security goals from the combination of PAKE and TLS. While we have no reason to expect that the protocol from [DAT12] is insecure, our model would enable such an analysis of that protocol.

Outside of the world of TLS, the Off-the-Record Messaging protocol [BGB04, AG07] uses a PAKE-like password authentication functionality that is based on secure two-party computation techniques to authenticate the long-term public keys used in establishing confidential (yet deniable) channel, but with no justification for the security of the combined construction.

1.3 Contributions

We analyze the modular approach to secure password authentication on the web, in which a secure channel protocol such as TLS is combined—in a black box way—with a password-authentication protocol. The black-box approach enables 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 certificates for server authentication.

At a high level, in our approach, a normal secure channel is established with no assumptions on correct validation of certificates; then a PAKE protocol is run *within* the secure channel to demonstrate (a) mutual knowledge of the password, and (b) absence of a man-in-the-middle attack on the channel; once the PAKE succeeds, the parties continue application communication. Notably, the session key established by the PAKE is not used for symmetric encryption; while it *could* replace the secure channel session key, in practice, such as in TLS, there is no standardized mechanism to do so. Moreover, using the existing channel session key is fine provided we bind the execution of the PAKE and the original channel establishment in a provably secure way. We show how to realize this binding using available TLS standards.

Our formal approach is as follows.

First, we define formally in Section 3 the security property we aim for in our protocol: 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. [JKSS12], we give a corresponding *password-based ACCE (PACCE)* notion. We apply our PACCE model to analyzing strong

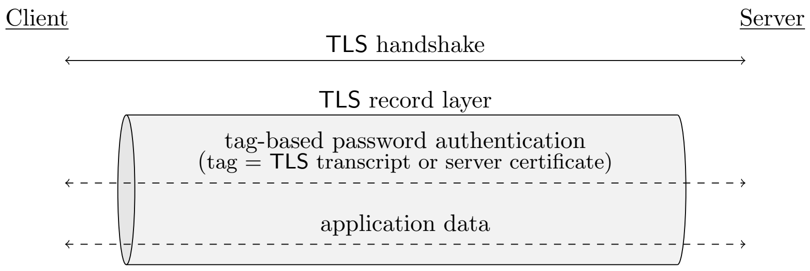

Client
Server
TLS handshake
TLS record layer
tag-based password authentication
(tag = TLS transcript or server certificate)
application data

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

passwords in TLS; as ACCE has been used to analyze many real world protocols (SSH [BDK+14], EMV [BSWW13], and QUIC), PACCE should be suitable for strong password variants of those.

Next, we define the various primitives we employ to achieve this goal; these include the original ACCE notion, as well as an unauthenticated *confidential channel establishment (CCE)* protocol. Rather than using a PAKE protocol, we actually employ a newly defined *tag-based password authentication (tPAuth)* protocol, which provides mutual authentication based on knowledge of a shared password, with acceptance only if both parties input the same, possibly public, *tag* to the protocol.

Then, when we run the tPAuth protocol inside the established confidential channel, we bind the two protocols together by setting the tag to be either the transcript of the channel establishment or the long-term public key used by the server in the channel establishment. We prove in Sections 4 and 5 that both of these tags are sufficient to achieve the end result of a password-based authenticated and confidential channel establishment (PACCE) protocol. Our two security theorems provide a qualitative distinction between those two tags as binding mechanisms because they use slightly different assumptions on the confidential channel establishment protocol: when the tag used is the transcript, it suffices to use a CCE protocol, but when the tag used is the server's public key, we require the underlying protocol to be an ACCE protocol. (These results give the first security justification of two standardized TLS channel binding mechanisms, `tls-unique` and `tls-server-end-point` [AWZ10].)

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, not just TLS. These results justify the *general approach* of the recent proposals of Oiwa et al.'s protocol [OTWS09, OWT09, OWT+12, AIS] and Dacosta et al.'s DVCert protocol [DAT12]. We caution that our theorems do not immediately imply security of those particular protocols for several reasons.

Our theorems depend on a formal construction called a tag-based password authentication (tPAuth) protocol. Only the tSOKE protocol we describe in Appendix A is known to be a secure tPAuth. For most PAKE protocols, such as the PAK protocol employed by Dacosta et al., it seems not hard to modify the PAKE security proof to demonstrate tPAuth, but this requires additional work for each particular protocol.

Oiwa et al. provide three channel binding mechanisms: the server's TLS certificate, the server host string (e.g., `http://www.example.com:80`), and the server's TLS master secret key.

- Our results do not address the cryptographic security of server host string binding; indeed, Oiwa et al. intend for this mode mainly to be used when TLS is not employed.
- Section 5.3 provides a justification for the use of the TLS server certificate, though as noted our results only apply when the PAKE is also a tPAuth.
- Section 4.4 provides a justification for the use of the TLS `Finished` message, whereas Oiwa et al. allow use of the TLS master secret key. Since the TLS handshake will not complete unless both parties compute the same `Finished` messages, our results also justify the use of the TLS master secret key for channel binding, though security of the whole construction again is not implied by our results due to the caveat above that our results only apply with the PAKE is also shown to be a tPAuth.

Dacosta et al.'s DVCert protocol only provides server-to-client password-based authentication, whereas our PACCE notion provides mutual authentication.

Implementation. In Section 6, we demonstrate the practical merit of our approach with a reference implementation for the Mozilla Firefox web browser and the Apache web server that takes advantage of the modularity of the construction.

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

Binding the password authentication protocol to the TLS channel is achieved using the server's TLS certificate as the tag; both Firefox and Apache have APIs exposing the server's certificate to the extension and PHP application, respectively; the server certificate is one of two channel binding mechanisms standardized for TLS [AWZ10]. The source code size of our implementation is quite small. Our pure Javascript implementation is completely cross-platform and provides tolerable performance, with total round-trip time under half a second on a laptop, while our native C implementation using OpenSSL libraries provides high client and server performance with a total protocol execution time, including network latency in a corporate network, around 109ms. Our implementation is available for immediate download.

Though the ultimate goal of this line of work would be for such a protocol to be built into the browser, our Javascript extension may be amenable to gradual deployment to seed adoption while still achieving good performance, especially when making use of native libraries.

2 Approach

Our general approach for secure modular password-based authentication on the web 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.

From a theoretical perspective, we want to ensure that the combination of these 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 but then relay the password authentication across successfully. We will provide formal justifications of the generic construction using two different channel binding strings: transcripts and public keys.

For a practical implementation involving TLS, one also needs to specify the format of messages and how they are delivered. Because the portions used for password authentication follow after the establishment of the TLS channel, 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 protocol.

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. [JKSS12] 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 [KPW13, KSS13] 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 [GKS13].

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 protocol

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 construction, 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).

We require a *tag-based password authentication (tPAuth) protocol*, 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 [JKSS10, FMA12].

The formal definition of tPAuth security appears in Section 4.2. The tPAuth protocol we use in our reference implementation, tSOKE, derived from the Simple Open Key Exchange (SOKE) protocol [ABC+06], is described in Appendix A; note that any tPAuth protocol could be used in our generic construction, and any PAKE protocol can in fact be transformed into a tPAuth protocol by hashing the original password with the tag and using the result as a new password [FMA12].

2.1.3 TLS channel bindings

To securely bind the password authentication protocol and the underlying 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 [AWZ10] 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", [AWZ10, §3.1], which corresponds to the `Finished` message from the client 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` through to `ChangeCipherSpec`.
- `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 [Res10] 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.

2.2 Construction

Our generic construction for secure PACCE between a client C and a server S from password-based authentication and a secure channel is as follows. First, the channel establishment protocol (CCE or ACCE) is run until it accepts. Then, using the secure channel, the two parties run a tag-based password authentication protocol where the tag is a binding value from the secure channel; when the tPAuth protocol accepts, then the parties accept in the overall PACCE protocol, and then continue to use the channel for communication. Our 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 establishment protocol or the long-term public key of the server.

We now concretely describe the combination of a tPAuth protocol with TLS, as detailed in Figure 2. C and S first build up 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 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` messages 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.

In the following sections, we will show that this generic construction using either binding mechanism, and hence the concrete TLS-based construction of Figure 2, provably leads 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. [JKSS12]. ACCE seems to be the most suitable for describing the security requirements of real-world secure channel protocols such as TLS [JKSS12, KPW13, KSS13, GKS13] and SSH [BDK$^+$14], 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.

[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.

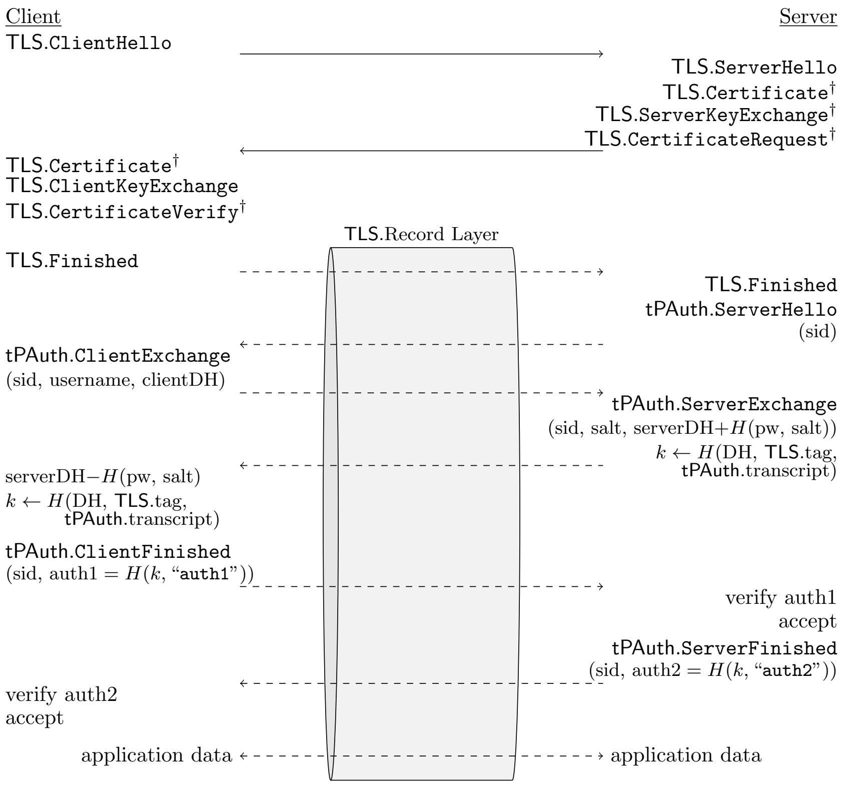


Figure 2: Protocol message diagram for TLS with tunnelled tag-based password authentication, with example messages for the tSOKE protocol. TLS.tag is either TLS.Finished or TLS.Certificate. † denotes optional messages.

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_U^s. Each instance Π_U^s records several variables:

- $\Pi_U^s.\mathsf{pid}$: the *partner identity* with which Π_U^s believes to be interacting in the protocol session.
- $\Pi_U^s.\rho \in \{\mathtt{init}, \mathtt{resp}\}$: the *role* of this instance in the session, either initiator or responder. $\mathtt{init}(\Pi_U^s)$ and $\mathtt{resp}(\Pi_U^s)$ denote Π_U^s's view of who the initiator and responder are in the session, namely $(U, \Pi_U^s.\mathsf{pid})$ when $\Pi_U^s.\rho = \mathtt{init}$, and $(\Pi_U^s.\mathsf{pid}, U)$ when $\Pi_U^s.\rho = \mathtt{resp}$.
- $\Pi_U^s.T$: a *transcript* composed of all messages sent and received by the instance in temporal order.
- $\Pi_U^s.\alpha \in \{\mathtt{active}, \mathtt{accept}, \mathtt{reject}\}$: the *status* of this instance.
- $\Pi_U^s.k$: the *session key* computed by this stage; initially set to empty $\emptyset$; when non-empty, it consists of two symmetric keys $\Pi_U^s.k^{\mathsf{enc}}$ and and $\Pi_U^s.k^{\mathsf{dec}}$ for encryption and decryption with some stateful length-hiding authenticated encryption scheme [JKSS12] used to provide confidentiality in the record layer stage. If $\Pi_U^s.\alpha = \mathtt{accept}$, then $\Pi_U^s.k \neq \emptyset$
- $\Pi_U^s.b \in \{0, 1\}$: a randomly sampled bit used in the Encrypt oracle.

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* [JKSS12], 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:

- $\mathsf{Send}^{\mathsf{pre}}(\Pi_U^s, m)$: 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 Enc/Dec, whereby `len` is the message length and `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. $(C^{(0)}, st_e^{(0)}) \leftarrow_R \mathsf{Enc}(\Pi_U^s.k^{\mathsf{enc}}, \mathtt{len}, \mathtt{head}, m_0, st_e)$
2. $(C^{(1)}, st_e^{(1)}) \leftarrow_R \mathsf{Enc}(\Pi_U^s.k^{\mathsf{enc}}, \mathtt{len}, \mathtt{head}, m_1, st_e)$
3. If $(C^{(0)} = \bot)$ OR $(C^{(1)} = \bot)$, then return $\bot$
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 $\bot$.
  4. $(m, st_d) \leftarrow \mathsf{Dec}(\Pi_U^s.k^{\mathsf{dec}}, \mathtt{head}, C, st_d)$
  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 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 [LLM07] 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 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) $\Pi_U^s.\alpha = \mathtt{accept}$, and
   (b) $\mathcal{A}$ did not issue $\mathsf{Corrupt}(\mathtt{init}(\Pi_U^s), \mathtt{resp}(\Pi_U^s))$ 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 dictionary $\mathcal{D}$ in a single session.

4 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 confidential channel establishment (CCE) protocol π and a tag-based password authentication (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+tSOKE with `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 $\mathsf{RevealSK}$, $\mathsf{Encrypt}$ and $\mathsf{Decrypt}$ 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) $\Pi^s_U.\alpha = \texttt{accept}$*, and*

(b) there exists an instance $\Pi^{s'}_{U'}$ that is partnered to Π^s_U, and

(c) $\mathcal{A}$ did not issue $\mathsf{RevealSK}(\Pi^s_U)$ or $\mathsf{RevealSK}(\Pi^{s'}_{U'})$ for any $\Pi^{s'}_{U'}$ that is partnered to Π^s_U,

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

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: tag-based password authentication

Tag-based authentication [JKSS10] 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 [JKSS10] for public key-based authentication protocols and then generalized in [FMA12] for other types of authentication factors, including passwords and biometrics. In our analysis we will use a *tag-based password authentication* protocol, denoted tPAuth.

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^s_C 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^s_C and $\Pi^{s'}_S$ are *partnered* if $\Pi^s_C.\mathsf{pid} = S$, $\Pi^{s'}_S.\mathsf{pid} = C$, $\Pi^s_C.T \approx \Pi^{s'}_{S'}.T$ (matching transcripts), and $\Pi^s_C.\tau = \Pi^{s'}_S.\tau$ (equal tags).

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

- $\mathsf{Send}(\Pi^s_U, m)$: This query is identical to $\mathsf{Send}^{\mathsf{pre}}$ from the PACCE model except for one important difference — when $\mathcal{A}$ initializes some instance Π^s_U using the special messages $m = (\texttt{init}, U', \tau)$ or $m = (\texttt{resp}, U', \tau)$ 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^s_U such that*

1. $\Pi_U^s.\alpha = \texttt{accept}$*, and*
2. $\mathcal{A}$ *did not issue* $\mathsf{Corrupt}(\texttt{init}(\Pi_U^s), \texttt{resp}(\Pi_U^s))$ *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 Appendix A, we present tSOKE, a tag-based variant of the Simple Open Key Exchange (SOKE) variant [ABC+06]. Since SOKE is a password-authenticated key exchange protocol, it does establish a secure session key; however, we only use its properties of *mutual authentication* and *resistance to dictionary attacks*. The tag is inserted into the key used for explicit key confirmation; it does not need resistance to dictionary attacks since it is known to the adversary, and thus is not used in the Diffie–Hellman portion like the password.

4.3 Security analysis of construction #1

Theorem 1 ($\mathsf{CCE} + \mathsf{tPAuth}_{\tau = T_{\mathsf{CCE}}} \implies \mathsf{PACCE}$)**.** *The generic construction of a PACCE protocol* $\Gamma_T(\pi, \xi)$ *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 PACCE-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.

The proof appears in Appendix B.

4.4 Using `tls-unique` channel binding

The `tls-unique` channel binding mechanism [AWZ10] 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.

5 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 authenticated and confidential channel establishment (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 (hash of) a certificate containing the public key, allowing us to justify the security of using TLS+tSOKE with `tls-server-end-point` channel binding.

5.1 Building block: ACCE (with key registration)

ACCE [JKSS12] 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. [GKS13]; 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 $\mathsf{Register}$ query; only sessions where the key is not an adversary-registered key are legitimate targets for breaking.

Formally, the differences of 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 key / secret key pair (pk_U, sk_U) and all parties, including the adversary, are assumed to have copies of all long-term public keys.
- Π_U^s maintains a variable $\Pi_U^s.\omega \in \{\texttt{server-only}, \texttt{mutual}\}$ indicating whether it is expecting server-only or mutual authentication.
- Π_U^s maintains a variable $\Pi_U^s.\mathsf{ppk}$ of the public key observed to be used by the peer.
- In the $\mathsf{Send}^{\mathsf{pre}}$ 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 $\mathsf{Corrupt}(\Pi_U^s.\mathsf{pid})$ before Π_U^s accepted.
- For breaking authentication, an additional condition is added:
  (e) if $\Pi_U^s.\omega = \texttt{server-only}$, then $\Pi_U^s.\rho = \texttt{init}$.
- For breaking both authentication and authenticated encryption, an additional condition is added:
  (f) $\mathcal{A}$ did not issue $\mathsf{Register}(\Pi_U^s.\mathsf{ppk})$.

5.2 Security analysis of construction #2

Theorem 2 ($\mathsf{ACCE} + \mathsf{tPAuth}_{\tau=pk} \implies \mathsf{PACCE}$)**.** *The generic construction of a PACCE protocol $\Gamma_T(\pi, \xi)$ 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 PACCE-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. The proof appears in Appendix B.

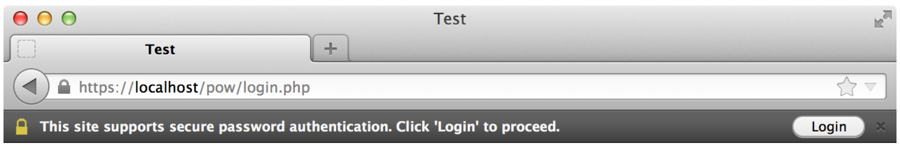
5.3 Using `tls-server-end-point` channel binding

The `tls-server-end-point` channel binding mechanism [AWZ10] 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.

6 Implementation

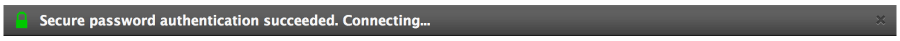
As an important motivation for our modular protocol design was the ability to modularly implement the protocol, we produced a prototype to demonstrate this.



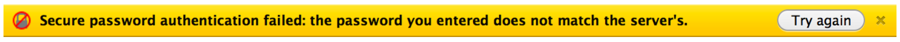
https://localhost/pow/login.php

This site supports secure password authentication. Click 'Login' to proceed.

(a) Login notification bar.

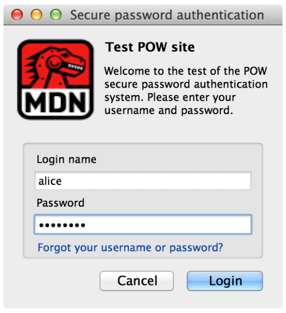

Secure password authentication succeeded. Connecting...

(c) Login success notification bar.



Secure password authentication failed: the password you entered does not match the server's.

(d) Login failure notification bar.


Secure password authentication
MDN
Forgot your username or password?
Cancel
Login
Test POW site

Welcome to the test of the POW secure password authentication system. Please enter your username and password.

Login name: alice

Password: ••••••••

(b) Login dialog box.

Figure 3: User interface for Mozilla Firefox extension.

For the tag-based password authentication protocol, we propose tSOKE, a tag-based version of the SOKE protocol [ABC+06], a highly efficient Diffie–Hellman based PAKE; see Appendix A for details of tSOKE. We used the NIST P-192 elliptic curve group [Nat99].

We implemented the client side of the protocol as an extension for Mozilla Firefox (320 lines of custom Javascript, plus libraries), and the server side of the protocol as a PHP application (210 lines of custom PHP code, plus libraries) on an Apache web server. No modifications to the source code of the underlying web browser (Firefox) 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 or Apache. Since the mechanism that the server code uses to obtain the certificate of the TLS connection is an Apache CGI (Common Gateway Interface) variable, any server-side language would work, not just PHP.

Our implementation is available online under an open-source license at `http://www.douglas.stebila.ca/research/papers/MSD14/` and `http://eprints.qut.edu.au/76270`.

6.1 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 6.4) a page that supports the protocol, it displays a notification bar that secure password authentication is supported (Figure 3(a)). The user then clicks on the "Login" button in the notification bar to bring up the password entry dialog box (Figure 3(b)). Note that the notification bar is displayed using Firefox's 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 [DT05, EKSS09]. The status of the mutual authentication is displayed in the notification bar (Figures 3(c), (d)); if successful the browser is redirected to the URL indicated by the server.

At present, Firefox is the only web browser whose extension APIs offer partial implementation of the channel bindings for TLS from RFC5929 [AWZ10], 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 validated, in that Chrome, Safari, and IE would only need to implement the recommendations from [AWZ10] such that our extension could do the rest of the protocol, rather than requiring the full protocol be implemented within the core browser source code as many other approaches require.

[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`

Operation	Pure Javascript	Native C	HTML form + SSL
Client cryptographic computations	354.06 ± 5.12	5.32 ± 0.23	—
Server cryptographic computations	36.27 ± 2.20	6.34 ± 0.48	—
Total runtime	487.72 ± 49.93	109.04 ± 47.96	66.16 ± 27.80

Software: Mozilla Firefox 21.0, Apache 2.2.22, PHP 5.4.14, GMP 5.1.1, MySQL 5.5.28, OpenSSL 1.0.1e, Mac OS X 10.8.3.
Hardware: 2.6 GHz Intel Core i7 (3720QM), 16 GB of RAM.
Network: Corporate network, ping time 48.55 ± 37.76ms

Table 1: Average runtime in ms (± standard deviation) of extension using cross-platform Javascript cryptographic code, native C (OpenSSL) cryptographic code, compared with standard passwords submitted using an HTML form over SSL.

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 3(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 3(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 notfication bar.

6.2 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. 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.

6.3 Performance

In Table 1, we report timings for our implementation. Timings reported are an average of 10 timings, with standard deviation. The total runtime of the protocol includes the network latency for the communication within a corporate network where the client and server machines were located. The average ping time on the network was 48.55 ms (stdev. 37.76).

We report two different sets of timings: "cross-platform" timings, using pure Javascript on the client side and PHP with built-in GMP libraries on the server side; and "native" timings, using calls to OpenSSL for cryptographic operations on both the client side and the server side.

The average total runtime from when the user clicked "Login" after entering their password until the protocol completes was 487.72 ms (stdev. 49.93) using cross-platform code, and 109.04 ms (stdev. 47.96) using native code. In comparison, the average total runtime of password authentication based on an HTML form over TLS in our setting takes 66.16 ms (stdev. 27.80)). The difference of about 40ms with our native code implementation corresponds to one additional round trip and is unlikely to be perceptible by the users. Our native cryptographic code is further comparable to Dacosta et al.'s reported performance of DVCert on laptops [DAT12]. 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.

6.4 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

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

new HTTP Authentication method [FHBH+99], with protocol messages transmitted via HTTP headers. It could be implemented within HTML, with the initial `tPAuth.ServerHello` message delivered as an HTML microdata object, and the remaining messages transmitted over asynchronous requests HTTP POST requests with responses formatted for example as JSON messages; this is then combined with standard HTTP cookie-based session management. For our prototype, 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.

7 Discussion

Although PAKE protocols have been known in the literature since their invention 1992, they have seen almost no deployed adoption for user authentication in real-world protocols and implementations, with the exception of the use of the socialist millionaires' protocol in the Off-the-Record Messaging (OTR) protocol for private instant messaging [AG07]. Engler et al. [EKSS09] 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 [ABC+07].

Deployment challenges This 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 [TWMP07] and an earlier draft of the HTTPS-PAKE approach of Oiwa et al. [OWT+12]. Adding SRP as a TLS ciphersuite has benefits in that, once implemented, 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.

User interface challenges Engler et al. [EKSS09] identify several user interface challenges. We do not aim to fully solve all 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, usually achieved by placing the password entry visibly in the browser chrome (i.e., 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 [DT05].

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 they do not fall back on insecure methods. Both of these remain a challenge for usability designers, though again, delivering failure notifications via Firefox's trusted path for notifications may provide some benefit. Providing forgotten password resets securely remains an open challenge both in practice and in theory and is outside the scope of our goals.

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, some explanatory text, as shown in Figure 3(b). 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 in to all major 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 to all of its users' browsers without needing to wait for the browser vendor to support the protocol. The more adoption via extension, the more evidence for interest in the technology, and the greater incentive for vendors to provide a native implementation.

Acknowledgements

Mark Manulis was supported by the German Research Foundation (DFG), project PRIMAKE (MA 4957). Douglas Stebila acknowledges funding from Australian Research Council (ARC) Discovery Project DP130104304.

References

[ABC⁺06] Michel Abdalla, Emmanuel Bresson, Olivier Chevassut, Bodo Möller, and David Pointcheval. Provably secure password-based authentication in TLS. In Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Lin, Shiuhpyng Shieh, and Sushil Jajodia, editors, *ASIACCS 06*, pp. 35–45. ACM Press, March 2006.

[ABC⁺07] Michel Abdalla, Emmanuel Bresson, Olivier Chevassut, Bodo Möller, and David Pointcheval. 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. DOI:10.1504/IJSN.2007.013181.

[ACCP08] Michel Abdalla, Dario Catalano, Céline Chevalier, and David Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In Tal Malkin, editor, *CT-RSA 2008*, *LNCS*, volume 4964, pp. 335–351. Springer, April 2008.

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, *LNCS*, volume 3386, pp. 65–84. Springer, January 2005.

[AG07] Chris Alexander and Ian Goldberg. Improved user authentication in Off-The-Record messaging. In Ting Yu, editor, *ACM Workshop on Privacy in Electronic Society (WPES) 2007*, pp. 41–47. ACM Press, 2007. DOI:10.1145/1314333.1314340.

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

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, *LNCS*, volume 3376, pp. 191–208. Springer, February 2005.

[AWZ10] J. Altman, N. Williams, and L. Zhu. Channel Bindings for TLS. RFC 5929 (Proposed Standard), July 2010. URL `http://www.ietf.org/rfc/rfc5929.txt`.

[BDK⁺14] Florian Bergsma, Benjamin Dowling, Florian Kohlar, Jörg Schwenk, and Douglas Stebila. Multi-ciphersuite security of the Secure Shell (SSH) protocol. In Moti Yung and Ninghui Li, editors, *ACM CCS 14*. ACM Press, 2014. DOI:10.1145/2660267.2660286. EPRINT `http://eprint.iacr.org/2013/813`.

[BGB04] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In *ACM Workshop on Privacy in Electronic Society (WPES) 2004*, pp. 77–84. ACM Press, 2004.

[BM92] Steven M. Bellovin and Michael Merritt. 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.

[BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In Bart Preneel, editor, *EUROCRYPT 2000*, *LNCS*, volume 1807, pp. 156–171. Springer, May 2000.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, *LNCS*, volume 1807, pp. 139–155. Springer, May 2000.

[BSWW13] Christina Brzuska, Nigel P. Smart, Bogdan Warinschi, and Gaven J. Watson. An analysis of the EMV channel establishment protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pp. 373–386. ACM Press, November 2013.

[Cer09] Certicom Research. SEC 1: Elliptic curve cryptography, 2009. Version 2.0.

[CHK$^+$05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, *LNCS*, volume 3494, pp. 404–421. Springer, May 2005.

[DAT12] Italo Dacosta, Mustaque Ahamad, and Patrick Traynor. Trust no one else: Detecting MITM attacks against SSL/TLS without third-parties. In Sara Foresti, Moti Yung, and Fabio Martinelli, editors, *ESORICS 2012*, *LNCS*, volume 7459, pp. 199–216. Springer, September 2012.

[DT05] Rachna Dhamija and J. D. Tygar. The battle against phishing: Dynamic security skins. In Lorrie Faith Cranor and Mary Ellen Zurko, editors, *Symposium on Usable Privacy and Security (SOUPS) 2005*, pp. 77–88. ACM Press, 2005. DOI:10.1145/1073001.1073009.

[EKSS09] John Engler, Chris Karlof, Elaine Shi, and Dawn Song. Is it too late for PAKE? In *Web 2.0 Security and Privacy (W2SP) 2009*, 2009. URL http://w2spconf.com/2009/papers/s4p1.pdf.

[FHBH$^+$99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999. URL http://www.ietf.org/rfc/rfc2617.txt. Updated by RFC 7235.

[FMA12] Nils Fleischhacker, Mark Manulis, and Amir Azodi. A Modular Framework for Multi-Factor Authentication and Key Exchange. Cryptology ePrint Archive, Report 2012/181, 2012. http://eprint.iacr.org/2012/181.

[GKS13] Florian Giesen, Florian Kohlar, and Douglas Stebila. On the security of TLS renegotiation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pp. 387–398. ACM Press, November 2013.

[HR08] Feng Hao and Peter Y. A. Ryan. Password authenticated key exchange by juggling. In *Security Protocols Workshop*, *LNCS*, volume 6615, pp. 159–171. Springer, 2008.

[IEE08] IEEE P1363.2. Standard specifications for password-based public-key cryptographic techniques, 2008. URL http://grouper.ieee.org/groups/1363/passwdPK/.

[Int06] International Organization for Standardization (ISO). ISO/IEC 11770-4: Information technology — security techniques — key management — part 4: Mechanisms based on weak secrets, May 2006. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=39723.

[ITU07] ITU-T X.1035. Password-authenticated key exchange (PAK) protocol, 2007. URL http://www.itu.int/rec/T-REC-X.1035-200702-I/en.

[JKSS10] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. Generic compilers for authenticated key exchange. In Masayuki Abe, editor, *ASIACRYPT 2010*, *LNCS*, volume 6477, pp. 232–249. Springer, December 2010.

[JKSS12] Tibor Jager, Florian Kohlar, Sven Schäge, and Jörg Schwenk. On the security of TLS-DHE in the standard model. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, *LNCS*, volume 7417, pp. 273–293. Springer, August 2012.

[KPW13] Hugo Krawczyk, Kenneth G. Paterson, and Hoeteck Wee. On the security of the TLS protocol: A systematic analysis. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, *LNCS*, volume 8042, pp. 429–448. Springer, August 2013. DOI:10.1007/978-3-642-40041-4_24.

[KSS13] Florian Kohlar, Sven Schäge, and Jörg Schwenk. 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.

[Kwo01] Taekyoung Kwon. Authentication and key agreement via memorable passwords. In *NDSS 2001*. The Internet Society, February 2001.

[LLM07] Brian A. LaMacchia, Kristin Lauter, and Anton Mityagin. Stronger security of authenticated key exchange. In Willy Susilo, Joseph K. Liu, and Yi Mu, editors, *ProvSec 2007*, *LNCS*, volume 4784, pp. 1–16. Springer, November 2007.

[Nat99] 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.

[OTWS09] Yutaka Oiwa, Hiromitsu Takagi, Hajime Watanabe, and Hirofumi Suzuki. PAKE-based mutual HTTP authentication for preventing phishing attacks. In Yoelle Maarek and Wolfgang Nejdl, editors, *Proc. 18th International World Wide Web Conference (WWW) 2009*, pp. 1143–1144. ACM, 2009. DOI:10.1145/1526709.1526898. URL http://www2009.org/proceedings/pdf/p1143.pdf.

[OWT09] Yutaka Oiwa, Hajime Watanabe, and Hiromitsu Takagi. PAKE-based mutual HTTP authentication for preventing phishing attacks, November 2009. http://arxiv.org/abs/0911.5230.

[OWT+12] Yutaka Oiwa, Hajime Watanabe, Hiromitsu Takagi, Yuichi Ioku, and Tatsuya Hayashi. Mutual authentication protocol for HTTP, July 2012. Internet-Draft. http://tools.ietf.org/html/draft-oiwa-http-mutualauth-12.

[Res10] E. Rescorla. Keying Material Exporters for Transport Layer Security (TLS). RFC 5705 (Proposed Standard), March 2010. URL http://www.ietf.org/rfc/rfc5705.txt.

[SDOF07] Stuart E. Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. The emperor's new security indicators. In *2007 IEEE Symposium on Security and Privacy*, pp. 51–65. IEEE Computer Society Press, May 2007.

[SEA+09] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *USENIX Security 2009*, 2009. URL http://www.usenix.org/events/sec09/tech/full_papers/sunshine.pdf.

[TWMP07] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the Secure Remote Password (SRP) Protocol for TLS Authentication. RFC 5054 (Informational), November 2007. URL http://www.ietf.org/rfc/rfc5054.txt.

[Wu98] Thomas D. Wu. The secure remote password protocol. In *NDSS'98*. The Internet Society, March 1998.

A tSOKE: A tag-based password authentication protocol

To achieve secure password authentication, we make use of a tagged version tSOKE of the Simple Open Key Exchange (SOKE) variant [ABC+06], which in turn builds on the Simple PAKE (SPAKE) protocol [AP05], standardized in [IEE08]. Since we only require password authentication, we only make use of the *mutual authentication* aspects of SOKE, and do not output a session key. The only difference between tSOKE and SOKE is that the tag τ is inserted in the (session) key computation: $k \leftarrow H(\mathsf{id}_A, h, \tau, X, Y^*, Z)$, where H is id_A is the client's username, h is the hash of the user's password, 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 session key computation. Recall that parties subsequently demonstrate knowledge of the session key to each other by sending the PRF of fixed strings under the session key.

The tagged version tSOKE of the SOKE protocol is specified in Figure 4. The system parameters consist of an elliptic curve group (which we fix to be the NIST P-192 group [Nat99]) and a second generator G' constructed verifiably at random. In the registration stage, the user selects a password and then hashes the password, along with a salt, using the PBKDF2 iterated construction, and gives the seed and the hashed result h to the server to store. In the login stage, the user inputs the password, reconstructs the hashed value h using PBKDF2, and the client and server together employ a masked Diffie–Hellman key exchange to demonstrate mutual knowledge of the hashed password, along with a tag τ.

Theorem 3. tSOKE *is a secure tPAuth protocol, assuming that* SHA-256 *behaves like a random oracle.*

Proof. We prove this theorem by closely following the game-hopping proof of [ABC+06, Theorem 5.1] which states that assuming that SHA-256 hash function is modeled as a random oracle the original SOKE protocol is a secure PAKE protocol according to the security definitions from [AFP05]. Although any secure PAKE is also a secure password-based authentication-only protocol, there are two main issues with regard to the proof of [ABC+06, Theorem 5.1] that we need to address in order to prove that tSOKE is a secure tPAuth protocol according to Definition 3.

First, tSOKE does not compute any session keys. In particular, the difference to SOKE is that in tSOKE there is no derivation of the secrets `MasterSecret` and `KeyBlock` that were computed in SOKE

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`)
(constructed verifiably at random [Cer09, §3.1.3.2] with seed string "This is the seed string for the web passwords protocol.")

Registration stage (takes place over a secure channel)

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

Login stage

	Client A		**Server** B
	Input: tag τ		Input: tag τ
1.	Enter username id_A.		
2.	Enter password pw_{AB}.		
3.	$x \leftarrow_R \{2,\ldots,n-1\}$		
4.	$X \leftarrow xG$		
5.		$\xrightarrow{\mathsf{id}_A, X}$	
6.			Look up salt, c, h for id_A.
7.			$y \leftarrow_R \{2,\ldots,n-1\}$
8.			$Y \leftarrow yG$
9.			$Y^* \leftarrow Y + (h \bmod n)G'$
10.		$\xleftarrow{\mathsf{salt}, c, Y^*}$	
11.	$h \leftarrow \mathsf{PBKDF2}(\mathsf{SHA\text{-}256}, \mathsf{pw}_{AB}, \mathsf{salt}, c, 256)$		
12.	$Y \leftarrow Y^* - (h \bmod n)G'$		
13.	$Z \leftarrow xY$		$Z' \leftarrow yX$
14.	$pms \leftarrow \mathsf{SHA\text{-}256}(\mathsf{id}_A, h, \tau, X, Y^*, Z)$		$K \leftarrow \mathsf{SHA\text{-}256}(\mathsf{id}_A, h, \tau, X, Y^*, Z)$
15.	$A_1 \leftarrow \mathsf{SHA\text{-}256}(pms, \text{"auth1"})$		
16.		$\xrightarrow{A_1}$	
17.			Abort if $A_1 \neq \mathsf{SHA\text{-}256}(pms, \text{"auth1"})$
18.			$A_2 \leftarrow \mathsf{SHA\text{-}256}(pms, \text{"auth2"})$
19.		$\xleftarrow{A_2}$	
20.	Abort if $A_2 \neq \mathsf{SHA\text{-}256}(pms, \text{"auth2"})$		
21.	Accept		Accept

Figure 4: tSOKE protocol registration and login stages

using a pseudo-random function `PRF`. The security of this derivation step for SOKE in the proof of [ABC$^+$06, 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 no longer needed for the proof of tSOKE since the protocol does not use any pseudorandom functions.

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 3). Observe that the input tag τ is processed in tSOKE as an additional input to the hash function SHA-256 in the computation of `PreMasterSecret`. The original game G_2 in the proof of [ABC$^+$06, 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. □

B Proofs

B.1 Proof of Theorem 1

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 $\mathsf{Send}^{\mathsf{pre}}$, $\mathsf{RevealSK}$, $\mathsf{Corrupt}$, $\mathsf{Encrypt}$, and $\mathsf{Decrypt}$.

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:

- $\mathsf{Send}^{\mathsf{pre}}(\Pi_U^s, m)$: $\mathcal{B}$ emulates a call for the CCE protocol π to $\mathsf{Send}^{\mathsf{pre}}(\widehat{\Pi_U^s}, m)$ 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 $\Pi_U^s.k^{\mathsf{enc}}$ and $\Pi_U^s.k^{\mathsf{dec}}$ 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 $\overline{\Pi_U^s}$ accepts or rejects, so too does Π_U^s. If $\overline{\Pi_U^s}$ returns a tPAuth protocol message m'', then $\mathcal{B}$ encrypts it by emulating a call to $\mathsf{Encrypt}(\widehat{\Pi_U^s}, m'', m'', \mathtt{len}(m''), \mathtt{head})$ and returns the resulting ciphertext C'' to $\mathcal{A}$.
  If the CCE portion is in the record layer stage and $\Pi_U^s.\alpha = \mathtt{accept}$, then $\mathcal{B}$ emulates a call for the CCE protocol π to $\mathsf{Decrypt}(\widehat{\Pi_U^s}, C, \mathtt{head})$ and returns the result m' to $\mathcal{A}$.

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

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

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_U^s of the PACCE protocol and sessions $\overline{\Pi_U^s}$ 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 $\mathsf{Send}^{\mathsf{pre}}$, $\mathsf{Corrupt}$, 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,

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

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_U^s that accepts without having a partnered instance (i.e., if $\Pi_U^s.\mathsf{pid} = U'$, then there does not exist an instance $\Pi_{U'}^{s'}$ such that Π_U^s and $\Pi_{U'}^{s'}$ have matching transcripts) and $\mathcal{A}$ has not issued a $\mathsf{Corrupt}(\mathtt{init}(\Pi_U^s), \mathtt{resp}(\Pi_U^s))$ 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_U^s. Let Since PACCE instance Π_U^s accepted, the underlying tPAuth instance $\overline{\Pi_U^s}$ must also have accepted; its transcript is $\Pi_U^s.T_{\mathsf{tPAuth}}$ and it was initialized on tag $\Pi_U^s.T_{\mathsf{CCE}}$.

However, no partner instance to Π_U^s exists: namely, there is no PACCE instance with transcript $T' = T'_{\mathsf{CCE}} \| T'_{\mathsf{tPAuth}}$ matching $\Pi_U^s.T$. Now, either (i) there exists a PACCE instance with T'_{CCE} equal to $\Pi_U^s.T_{\mathsf{CCE}}$ but not with T'_{tPAuth} matching $\Pi_U^s.T_{\mathsf{tPAuth}}$, or (ii) there is no other PACCE instance with T'_{CCE}

equal to $\Pi_U^s.T_{\mathsf{CCE}}$. In case (i), this means there exists a tPAuth instance initialized on tag $T'_{\mathsf{CCE}} = \Pi_U^s.T_{\mathsf{CCE}}$ but not with transcript T'_{tPAuth} matching $\Pi_U^s.T_{\mathsf{tPAuth}}$; thus, $\overline{\Pi_U^s}$ 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 $\Pi_U^s.T_{\mathsf{CCE}}$; thus, $\overline{\Pi_U^s}$ has no partnered instance initialized with the same tag $\Pi_U^s.T_{\mathsf{CCE}}$, 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_U^s of the PACCE protocol and sessions $\widehat{\Pi_U^s}$ 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 $\mathsf{Send}^{\mathsf{pre}}$, $\mathsf{RevealSK}$, 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 $\Pi_U^s.b = b'$ and $\mathcal{A}$ did not query $\mathsf{RevealSK}(\Pi_U^s)$ or $\mathsf{RevealSK}(\Pi_{U'}^{s'})$ for any partnered instance $\Pi_{U'}^{s'}$. Since in game G_2 $\mathcal{B}$ would have aborted if Π_U^s accepted without having a partnered instance, there must exist some partnered instance $\Pi_{U'}^{s'}$. Correspondingly, in $\mathcal{C}^\pi$ there exists an instance $\widehat{\Pi_U^s}$ with partnered instance $\widehat{\Pi_{U'}^{s'}}$. Since $\mathcal{A}$ did not issue the prohibited $\mathsf{RevealSK}$ queries, $\mathcal{B}$ also did not issue the prohibited queries $\mathsf{RevealSK}(\widehat{\Pi_U^s})$ or $\mathsf{RevealSK}(\widehat{\Pi_U^s})$. 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:

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

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

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

and we obtain the security of the combined construction. □

B.2 Proof of Theorem 2

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 $\mathsf{Send}^{\mathsf{pre}}$, $\mathsf{RevealSK}$, $\mathsf{Corrupt}$, $\mathsf{Encrypt}$, 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_U^s of the PACCE protocol and sessions $\overline{\Pi_U^s}$ 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 $\mathsf{Send}^{\mathsf{pre}}$, $\mathsf{Corrupt}$, 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_U^s that used some public key pk as an input tag to the tPAuth part of the protocol and accepted with the corresponding tPAuth transcript $\Pi_U^s.T_{\mathsf{tPAuth}}$ but for which there exists no instance $\Pi_{U'}^{s'} = \Pi_U^s.\mathsf{pid}$ with the matching tPAuth transcript and tag and $\mathcal{A}$ has not issued a $\mathsf{Corrupt}(\mathtt{init}(\Pi_U^s), \mathtt{resp}(\Pi_U^s))$ 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 $\mathsf{Send}^{\mathsf{pre}}$ queries of $\mathcal{A}$ are answered using corresponding queries to $\mathcal{C}^\pi$ except for the case where the query $\mathsf{Send}^{\mathsf{pre}}$ 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^s_U that has accepted without having a partnered instance $\Pi^{s'}_{U'}$. Game G_2 guarantees that for any Π^s_U that accepts in PACCE there must be an instance $\Pi^{s'}_{U'}$ 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^s_U and $\Pi^{s'}_{U'}$ 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,

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

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

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

and we obtain the security of the combined construction. □