Cross-Domain Password-Based Authenticated Key Exchange Revisited

Liqun Chen[1], Hoon Wei Lim[2], and Guomin Yang[3]

[1]HP Labs, Bristol, UK
[2]Nanyang Technological University, Singapore
[3]University of Wollongong, Australia

July 16, 2012

Abstract

We revisit the problem of cross-domain secure communication between two users belonging to different security domains within an open and distributed environment. Existing approaches presuppose that either the users are in possession of public key certificates issued by a trusted certificate authority (CA), or the associated domain authentication servers share a long-term secret key. In this paper, we propose a *four-party password-based authenticated key exchange* (4PAKE) protocol that takes a different approach from previous work. The users are not required to have public key certificates, but they simply reuse their login passwords they share with their respective domain authentication servers. On the other hand, the authentication servers, assumed to be part of a standard PKI, act as ephemeral CAs that "certify" some key materials that the users can subsequently exchange and agree on a session key. Moreover, we adopt a compositional approach. That is, by treating any secure two-party password-based key exchange protocol and two-party asymmetric-key based key exchange protocol as black boxes, we combine them to obtain a generic and provably secure 4PAKE protocol.

Keywords: Password-based protocol, key exchange, cross-domain, client-to-client.

1 Introduction

There are many cross-domain communication scenarios, such as email communication, mobile phone communication, and instant messaging, where the information being communicated may need to be protected against both passive and active attackers. In this paper, we consider the use of an authenticated key exchange protocol to establish a session key such that two users can securely transmit information from one domain to another.

In the above scenarios (we further discuss these example applications in Section 8), a user is typically registered to some kind of domain server, such as email exchange server or home location register (in the cases of email and mobile phone communications, respectively). Moreover, two communicating parties from different domains very often neither share a password nor possess each other's public key certificate. Hence, although two-party and three-party authenticated key exchange protocols have been extensively studied and widely deployed in the real world, see for example [31, 26, 7, 12, 3], it is not clear how they can be directly applied to establish a secure cross-domain communication channel.

In our work, we assume that each security or administrative domain (or identity federation [11]) has (at least) a trusted domain server acting as an authentication server governing a group of users.

Each user within the domain shares only a password with the server and she does not necessarily own a public key certificate. Moreover, we assume that a domain server makes available its public key to other domain servers in the form of a public key certificate, *i.e.* the server is connected to a public key infrastructure (PKI). In such a setting, we focus on enabling a user from one domain to establish a secure communication channel with another user from a different domain through their respective domain servers. Our approach makes use of both password-based and public-key cryptographic techniques for authentication and key exchange. We call this work *four-party* password-based authenticated key exchange (4PAKE) instead of *client-to-client* password-based authenticated key exchange (C2C-PAKE), as suggested in the literature [14, 43], because we thought that the latter may be confused with 3-party C2C-PAKE (in which both communicating parties are registered to the same domain server).

We believe that our aforementioned communication model is more realistic, user-friendly and scalable than that of related previous work, for example, public key Kerberos [25, 44] and C2C-PAKE [43, 15]. The later protocols either require a user to obtain a public key certificate, or assume that the domain servers corresponding to the communicating users share a long-term secret key. We elaborate more on previous work in Section 2.

The primary contribution of this paper is a proposal of a new generic 4PAKE protocol that meets the requirements discussed above. Our construction is based on a *compositional* approach and can be regarded as a form of compiler combining and transforming two building blocks — (i) a secure two-party password-based authenticated key exchange (2PAKE) protocol, and (ii) a secure two-party asymmetric-key authenticated key exchange (2AAKE) protocol — into a secure four-party password-based authenticated key exchange protocol. The detail of our construction is presented in Section 4. Further, we define a security model for our generic 4PAKE protocol based on the Real-Or-Random (ROR) security model [3] and show that our protocol is secure in the model. Our security definitions and analysis are shown in Sections 5 and 6, respectively. We also provide some performance analysis in Section 7.

2 Previous Work

2.1 Kerberos

Initially developed by an MIT research team led by Miller and Neuman, Kerberos [30, 31, 32] is now a widely deployed network authentication protocol. The most current version, Kerberos 5, is supported by all major operating systems, including Solaris, Linux, MacOS, and Microsoft Windows.

In Kerberos, each domain (also known as realm) is governed by a Key Distribution Center (KDC), which in turn, provides user authentication and ticket-granting services. Each user shares a password with its KDC, while local application servers that are accessible to the user share (long-term) symmetric keys with the KDC. Kerberos then allows single sign-on that authenticates clients to multiple networked services, such as remote hosts, file servers and print spoolers. This can be summarised in three rounds of communication between the client (typically acting on behalf of a user) and different principals as follows:

1. the client first performs a password-based login to its local KDC, *i.e.* authentication server, and obtains a ticket-granting ticket (TGT);

2. the TGT is then forwarded to a ticket-granting server in order to obtain a service ticket;

3. the client finally presents the service ticket to the application servers to get access to networked services.

The above standard Kerberos protocol makes use of highly efficient symmetric key techniques. However, one security weakness is that a password-derived symmetric key is used in the first round of the protocol between the client and the KDC. This opens up the possibility of allowing a passive attacker to eavesdrop the protocol messages (transmitted in the first round) and perform an off-line password guessing attack. In other words, the strength of the user authentication may be only as strong as the user's ability to choose and remember a strong password.

Public key cryptography for initial authentication in Kerberos (PKINIT) has thus been proposed by Zhu and Tung [44] to add flexibility, security and administrative convenience by replacing the password-based authentication with signature-based authentication between the client and the KDC. (The symmetric key operations in the second and third rounds of the protocol are retained.) The client and the KDC do not share a secret now. Each of them is assigned a public-private key pair instead, and they must now generate their respective signatures over the messages communicated in the first round. While the PKINIT extension offers stronger user authentication, it adds complexity to the protocol since we now require a public key infrastructure (PKI) and each user needs to manage her public-private key pair.

Moreover, Kerberos can be used to achieve cross-realm authentication (PKCROSS) by using public key techniques. This is useful when a client from a domain wishes to access networked services offered by another domain (that is governed by a remote KDC). Here, the two corresponding KDCs exchange messages following closely the PKINIT specification [44]. This avoids unnecessary administrative burden of maintaining cross-realm, shared symmetric keys. The (simplified) basic PKCROSS protocol is as follows [25]:

1. The client submits a request to its local KDC for credentials associated with the remote realm.[1]

2. The local KDC submits a request (using the standard PKINIT) to the remote KDC to obtain a cross-realm TGT.

3. The remote KDC responds as with PKINIT, and the local KDC passes the cross-realm TGT to the client.

4. The client then submits a request directly to the remote KDC and proceeds with the second and third rounds of the standard Kerberos protocol using symmetric key techniques.

Our work is closely related to PKCROSS, in the sense that we also deal with cross-domain authentication and secure communication. However, our proposal of 4PAKE is based on rather different design principles. We will elaborate on this in Section 3.

2.2 Client-to-Client Key Exchange

The idea of extending password-based key exchange between two users from the same domain to the cross-domain setting was also studied by Byun *et al.* [14], and Yin and Bao [43]. It was often known as client-to-client password-based authenticated key exchange (or key agreement) and thus the acronym C2C-PAKE (or C2C-PAKA). The key concept of C2C-PAKE is based on the cross-realm Kerberos protocol [31], in which each realm (or domain) has a KDC and that two users from two distinct realms establish a common secret key through their respective KDCs. We note that, however, existing proposals for C2C-PAKE make use of a symmetric key approach. Both KDCs that are involved in a C2C-PAKE protocol run are assumed to be sharing a long-term symmetric key. This leads to a similar limitation in symmetric key management that PKCROSS aims to address. More recent work on C2C-PAKE with improved efficiency and/or security can be found in [15, 22, 41].

Our work shares similar spirit with proposals on inter-domain password-based key exchange in the public key setting by Yeh and Sun [42], and Wong and Lim [40]. Briefly, these proposals make use of the domain servers' public keys to protect protocol messages between clients. However, no rigorous and formal security analyses of the proposed protocols have been provided.

[1] Note that the local KDC can authenticate the client using a password-based approach or PKINIT. However, as explained, the latter is usually a preferred choice since it is more secure than the former.

3 Our Motivations and Approach

Our concern here is on secure communication between two users from different administrative or security domains. While this is not a new security problem *per se*, we observe that current approaches fall short of being able to offer a satisfactory solution.

A main design goal of Kerberos was to allow single sign-on such that a user is able to access multiple networked services without the need to repeatedly enter her login password. Hence, the emphasis is on authenticating a user once and allowing secure access to multiple services within a time period. On the other hand, our work focuses on authenticated key agreement between two users of separate domains. This fundamental difference leads to other differences in terms of the protocol message flows and the security architecture. For example, in Kerberos, we assume that a user shares a password, while each application server shares a long-term symmetric key with their KDC; while in our case, we have a more "symmetric" situation where both users each shares a password with her respective domain authentication server.

Furthermore, Kerberos seems to be more suited to a rather "closed" distributed environment, where it is feasible to establish, distribute and maintain long-term secret keys between a KDC and a group of application servers, and between the KDC and other remote KDCs. Although this is compensated with the PKINIT and PKCROSS extensions, PKIs are known to be difficult to deploy for various practical reasons related to, such as cost, registration process, trust establishment, key revocation, and management of user private keys [21, 36]. We believe that what is needed here is a PKI that is more "user-friendly", *i.e.* which hides the complexity of public key management from a user's view point.

Existing work on C2C-PAKE does provide some useful insights on how to construct an efficient authenticated key agreement protocol between a local and a remote users. Unfortunately, however, these protocols rely on long-term symmetric keys shared between KDCs. This may not be a practical and scalable approach, particularly in an open distributed environment, because establishment and management of shared keys between KDCs can be a complicated and costly process.

From a security perspective, the complexity of a cross-domain authenticated key exchange often complicates its security analysis. As shown in [13, 4], the security analysis of Kerberos, with or without PKINIT, is rather complex and involved. This sometimes may hinder a security flaw in the protocol from being detected early. For example, the IETF Internet Draft for PKINIT was first circulated in 1996, but it was only after almost a decade later when Cervesato *et al.* reported a man-in-the-middle attack in PKINIT [18]. Similarly, designing a secure C2C-PAKE protocol seems to be a non-trivial task at all. Most of the C2C-PAKE protocols found in the literature have security flaws. These include off-line password guessing attacks [34, 39], undetectable on-line password guessing and unknown key-share attacks [35], insider attacks (by malicious clients and servers) [19, 34, 35], and password-compromise impersonation attacks [17]. Recent proposals [41, 22] attempt to address these security problems.

Taking all the above observation into consideration, we propose a 4PAKE protocol that we believe is suitable for secure cross-domain communication between two users. Intuitively, in our protocol, each domain server possesses a public key certificate that is publicly available to other domain servers (as with the case of web servers used for many e-commerce or online applications). In a protocol run, the servers corresponding to two communicating users "certify" some key materials that are associated with the users so that the latter can subsequently exchange and agree on a session key. Clearly, we assume that the user trusts the remote server to only certify key materials submitted by an authenticated user. We achieve this through a 2PAKE protocol between the user and her domain server. Hence, the user needs to remember only a password and does not require to deal with public key management. Once both the users have received the certified key materials (in the form of a signature) from their respective servers, they exchange the key materials following a 2AAKE protocol.

Inspired by the work of Abdalla *et al.* on a compositional approach to three-party password-based authenticated key exchange [3], we adopt a similar approach to our 4PAKE protocol, which comprises 2PAKE and 2AAKE as the building blocks. This is to simplify the security analysis of our protocol. By making use of secure 2PAKE and 2AAKE, we treat them as "black-boxes" and our analysis then focuses only on the input and output parameters of these two-party protocols. Moreover, using a compositional approach, we have the flexibility to choose any secure 2PAKE and 2AAKE protocols, implying that one can simply build a 4PAKE protocol based on existing deployed 2PAKE and 2AAKE protocols. In fact, each domain may deploy a different 2PAKE or 2AAKE protocol, but yet a user from one domain can securely establish a secret key with another user of a different domain. Further, if a serious security flaw is found on one of the building blocks, we simply replace it with another secure two-party protocol without changing the entire four-party protocol.

4 Generic Four-Party Key Exchange Protocol

We now present our generic construction of a (4PAKE) protocol.

4.1 Cryptographic Primitives

We first describe two cryptographic primitives required for our protocol construction.

Message Authentication Codes Let κ be a security parameter, a message authentication code (MAC) scheme is then a tuple of probabilistic polynomial-time algorithms (Gen, Mac, Ver) such that:

- $\textsc{Gen}(1^\kappa)$, the key generation algorithm, takes as input the security parameter 1^κ and outputs a key K with $|K| \geq \kappa$;
- $\textsc{Mac}(K; m)$, the MAC tag generation algorithm, takes as input a key K and a message $m \in \{0,1\}^*$ and outputs a tag μ;
- $\textsc{Ver}(K; m; \mu)$, the verification algorithm, takes as input a key K, a message m and a tag μ. It outputs 1 if μ is a valid tag for message m under key K, or 0 otherwise.

Signatures A signature scheme is a tuple of probabilistic polynomial-time algorithms (Gen, Sig, Ver) satisfying the following:

- $\textsc{Gen}(1^\kappa)$, the key generation algorithm, takes as input the security parameter 1^κ and outputs a pair of public/private keys (pk, sk) with $|pk|$ and $|sk|$ each have length at least κ;
- $\textsc{Sig}(sk; m)$, the signing algorithm, takes as input a private key sk and a message $m \in \{0,1\}^*$ and outputs a signature σ;
- $\textsc{Ver}(pk; m; \sigma)$, the verification algorithm, takes as input a public key pk, a message m and a signature σ. It outputs 1 if σ is a valid signature for message m under key sk, or 0 otherwise.

4.2 Protocol Construction

Our 4PAKE protocol is constructed in a compositional way, in the sense that it entails "piggy-backing" 2PAKE (password-based) and 2AAKE (asymmetric-key based) protocols. Intuitively, a client makes use of a 2PAKE protocol to mutually authenticate and establish a shared key with its domain server. In addition, the client obtains a signature over some ephemeral key materials from its server. The signature, in turn, will be exchanged with the intended communication partner, which also performs similar steps with its domain server. These two clients then agree on a session key through a 2AAKE using their respective signed ephemeral key materials.

The notation used in our 4PAKE protocol is described in Table 1. We assume that (pk_X, sk_X) and (epk_X, esk_X) are both asymmetric key pairs. We write $ssk_{X_1,X_2} \leftarrow \mathsf{2PAKE}(pwd_{X_1,X_2})$ to denote the execution of a 2PAKE protocol, where the protocol is run between X_1 and X_2,

Table 1: Notation.

$X \in \{A, B, S_A, S_B\}$	Entities involved in our 4PAKE protocol
id_X	Unique identifier of X
pk_X	Public key of X
sk_X	Secret key of X
epk_X	Ephemeral public key of X
esk_X	Ephemeral secret key of X
pwd_{X_1,X_2}	Password shared between X_1 and X_2
ssk_{X_1,X_2}	Session key shared between X_1 and X_2

and which takes pwd_{X_1,X_2} as input and establishes ssk_{X_1,X_2}. Moreover, we use $ssk_{X_1,X_2} \leftarrow \mathsf{2AAKE}((sk_{X_1}, pk_{X_1}, \sigma_{X_1}), (sk_{X_2}, pk_{X_2}, \sigma_{X_2}))$ to denote the execution of a signature-based 2AAKE protocol, where the protocol is run between X_1 and X_2. It takes as input an asymmetric key pairs (sk_X, pk_X) and a signature σ_X (verifying the authenticity of the public key pk_X) of each entity, and establishes ssk_{X_1,X_2}.

On the other hand, $\textsc{Mac}(K; m)$ and $\textsc{Sig}(sk_X; m)$ are as defined earlier.

We label a protocol message flow by Mx, where x indicates the x-th message flow. We use subscripts to differentiate protocol messages that are created and transmitted in parallel, for example Mx_A and Mx_B.

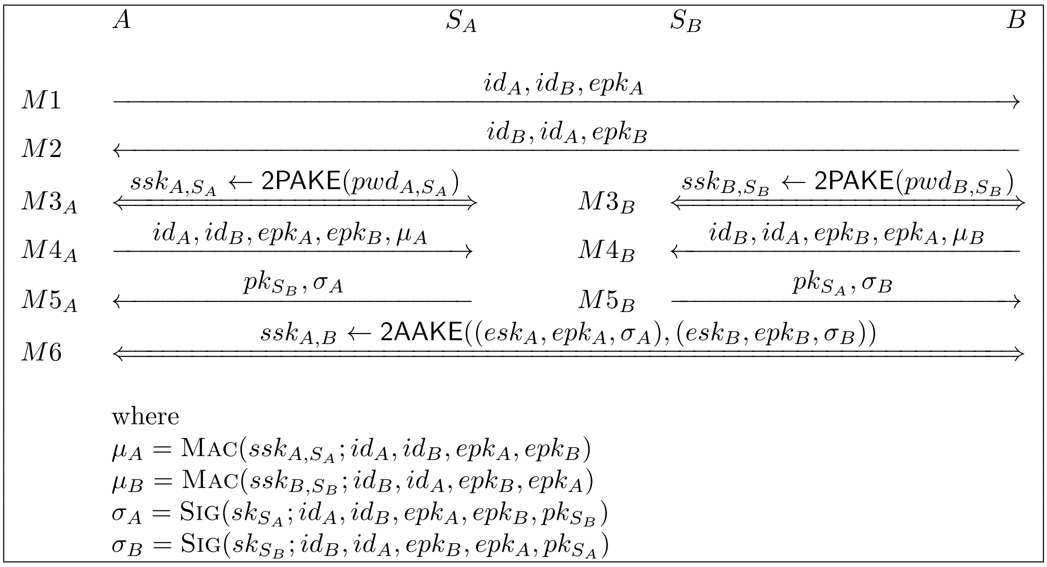

Figure 1: A generic four-party password-based key exchange (4PAKE) protocol

We assume that the execution of our 4PAKE protocol involves a pair of clients, denoted by $A, B \in \mathcal{C}$, and their respective servers, denoted by $S_A, S_B \in \mathcal{S}$. Here $\mathcal{C}$ and $\mathcal{S}$ are sets of possible clients and servers, respectively. Figure 1 shows the message flows of our protocol. Firstly in $M1$ & $M2$, clients A and B exchange information about their identities (id_A, id_B),[2] and ephemeral public keys (epk_A, epk_B), such as Diffie-Hellman components. Clients A and B then, using their passwords, perform authenticated key agreement with their domain servers S_A and S_B in $M3_A$ and $M3_B$, respectively. At the end of 2PAKE, each client establishes a shared key with its server. In $M4$, clients A and B forward their identities and ephemeral public keys to their servers. The information is protected using MAC tags generated using the shared keys from $M3$. In $M5$, servers S_A and S_B then each responds with a signature generated over the information received

[2] We can assume, in practice, that the identity information contains information about its associated domain. For example, if id_A is an IP address, then it also tells information about the domain to which id_A belongs.

from the client. Each server also provides its client the public key of the server corresponding to the intended remote client. Finally, A and B perform asymmetric-key based authenticated key exchange in $M6$ in order to agree on a session key.

REMARKS. Each domain server has access to the authentic public keys of other domain servers. Moreover, each client possesses a copy of its server's authentic public key in order to verify the signature the server generates in $M5$. This is not necessarily required to be done in advance. In practice, the servers can distribute their public keys to their clients during the execution of 2PAKE in $M3$, or alternatively in $M5$ by using a MAC algorithm in the same way as $M4$. However, note that in order to prevent password-compromise impersonation attacks, the clients must obtain the server public keys through other out-of-band mechanisms. This is because once a client's password is known to an adversary, it is trivial for the adversary to impersonate the relevant server by distributing a fake public key for which the adversary knows the corresponding private key.

Furthermore, we stress that there is no interaction between servers S_A and S_B during a protocol run. This seems to be a very attractive property since we can avoid overloading the servers with high communication cost in an open, distributed environment should they need to exchange messages in the protocol. The savings in terms of communication bandwidth is significant compared to PKCROSS, for example. (We present the detail of our performance analysis in Section 7.) If interaction between servers is tolerable, one alternative construction is to replace $M4$ & $M5$ in Figure 1 with some kind of key distribution protocol involving the servers and the clients, *i.e.* the servers generating and distributing a pre-session key to the clients. In this case, the servers must establish an authenticated channel between them to agree on a pre-session key. This can also be done using either asymmetric or symmetric techniques. The pre-session key is then distributed to the clients (through their respective secure channels established via the 2PAKE protocol in $M3$) and is used by the clients to generate a session key. This way, more efficient two-party symmetric-based authenticated key exchange can be used instead, for example, by following a MAC-based method called REKEY in [16].

It is worth noting that one simplest instantiation of 2AAKE in $M6$ is the typical two-pass Diffie-Hellman key exchange protocol, involving exchanges of σ_A and σ_B between clients A and B. The output session key is then $ssk_{A,B} = \text{KDF}(g^{esk_A esk_B}, \dots)$, for example, where KDF is a key derivation function. Indeed, we can use any signature-based message transmission (MT) authenticator proposed by Bellare *et al.* [5] and Canetti and Krawczyk [16] in $M6$. See [37, 28] for other concrete examples of signature-based Diffie-Hellman key exchange. As explained before, signature-based 2AAKE is adopted in $M6$ so that the servers can avoid sharing a long-term symmetric key that may lead to a key distribution problem.[3]

5 Security Models

Let us first recall two existing security models related to password-based authenticated key exchange protocols: the Find-Then-Guess (FTG) and the Real-Or-Random (ROR) models.

The FTG model (sometimes also known as the BPR2000 model) was proposed by Bellare *et al.* to measure the indistinguishability of a session key from a random key [7]. In the FTG model, an adversary is allowed to pose multiple queries to a reveal oracle (in addition to other oracles, for example execute and send oracles). The reveal oracle is used to model the misuse of session keys by a user. However, the adversary is restricted to ask only a *single* query to the test oracle.

Abdalla *et al.* [3] then proposed the ROR model that is very similar to the FTG model, except that the former *does not* make use of a reveal oracle. This means that the adversary no longer has access to the reveal oracle to learn session keys of user instances. However, the adversary is

[3]Otherwise, for scenarios where sharing of symmetric keys between all servers does not pose any serious concern, one can replace 2AAKE with MAC-based key exchange, for example the MAC-based MT authenticator in [16], to reduce computational overhead.

allowed to pose as *many* test queries as it wishes to different instances. Note that in the ROR model, the test oracle (instead of the reveal oracle) is used to model the misuse of keys by a user.

We remark that the recently proposed ROR model is strictly stronger than the FTG model in the password-based setting.[4] Hence, we adopt the ROR model for the security analysis of our 4PAKE protocol.

5.1 Two-Party Authenticated Key Exchange

We now present the security models for two different types of two-party authenticated key exchange protocols. We first give an overview of the ROR model for the two-party password-based authenticated key exchange (2PAKE) [3]. We then define a security model in the ROR sense for two-party authenticated key exchange in the asymmetric-key setting (2AAKE).

5.1.1 Password-based setting

A two-party password-based authenticated key exchange (2PAKE) protocol allows two communicating parties, who make use of their respective passwords, to derive a common session key. The session key, in turn, is used to establish secure channels between the two parties.

In the 2PAKE setting, we assume that each protocol participant is either a client $C \in \mathcal{C}$ or a server $S \in \mathcal{S}$. The set of all users or participants $\mathcal{U}$ is the union $\mathcal{C} \cup \mathcal{S}$. We also assume that each client $C \in \mathcal{C}$ holds a password pwd_C, while each server $S \in \mathcal{S}$ holds a vector $pwd_S = \langle pwd_C \rangle_{C \in \mathcal{C}}$ with an entry for each client [7]. Here, pwd_C and pwd_S are regarded as the long-lived keys of client C and server S.

As with a typical security model, an adversary $\mathcal{A}$ interacts with protocol participants only via oracle queries. Such queries model the adversary's capabilities in a real attack. During a protocol execution, there may be many concurrent running instances of a participant. We denote an instance i of a protocol participant $U \in \mathcal{U}$ by U^i. Two instances U_1^i and U_2^i are said to be partners if the following conditions are met [7]:

(i) Both U_1^i and U_2^i accept;[5]

(ii) Both U_1^i and U_2^i share the same session identifiers;[6]

(iii) The partner identifier for U_1^i is U_2^i, and vice versa;

(iv) No instance other than U_1^i and U_2^i accepts with a partner identifier equal to U_1^i or U_2^i.

The oracle queries in the ROR security model for 2PAKE are then classified as follows [3]:

- Execute(C^i, S^j): This query models a *passive* attack in which the adversary eavesdrops on an honest execution of the protocol between a client instance C^i and a server instance S^j. The output of the query comprises messages that were exchanged during the honest execution of the protocol.

- Send(U^i, m): This query models an *active* attack in which the adversary may intercept a message and then either modify it, create a new one, or simply forward it to the intended participant. The output of the query is the message that the participant instance U^i would generate upon receipt of message m.

- Test(U^i): This query models the misuse of a session key by a user. Let b be a bit chosen uniformly at random at the beginning of an experiment defining indistinguishability in the ROR model. The output of the query is then the session key for participant instance U^i if $b = 1$ or a random key from the same domain if $b = 0$. However, if no session key is defined for instance U^i, then return the undefined symbol $\perp$.

[4] A protocol proved secure in the ROR model is also secure in the FTG model. The reverse, however, is not necessarily true. See [3] for further details about the relation between the ROR and FTG models.

[5] An instance U^i goes into an accept mode after it has received the last expected protocol message.

[6] Typically, a session identifier can be constructed based on the partial protocol messages exchanged between the client and the server instances before the acceptance.

We note that the adversary is allowed to ask multiple queries to the TEST oracle in the ROR model (this is in contrast with the FTG model which allows only a single query to the TEST oracle). All TEST queries must be made on *fresh* instances (which have not revealed their session keys) and they should be answered using the same value for the hidden bit b (chosen at the beginning of the experiment). This implies that the keys returned by the TEST oracle are either all real or all random. Moreover, in the case where the returned key is random, the same random value should be returned for TEST queries that are asked to two instances which are partnered [3]. The goal of the adversary is to guess the value of the hidden bit b used to answer TEST queries. The adversary is considered successful if it guesses b correctly. Let SUCC denote the event in which an adversary is successful. The advantage of an adversary $\mathcal{A}$ in violating the indistinguishability of the 2PAKE protocol in the ROR sense is

$$\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{2PAKE},\mathcal{D}}(\mathcal{A}) = 2 \cdot \Pr[\textsc{Succ}] - 1$$

when passwords are drawn from a dictionary $\mathcal{D}$. The associated advantage function is then

$$\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, R) = \max_{\mathcal{A}}\{\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{2PAKE},\mathcal{D}}(\mathcal{A})\}$$

where the maximum is over all $\mathcal{A}$ with time-complexity at most t and using resources at most R, for example the number of queries to its oracles. Clearly, the advantage of an adversary that simply guesses the bit b, from the above definition, is 0 due to the rescaling of the probabilities.

We say that a 2PAKE protocol is secure in the ROR model if the advantage $\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{2PAKE},\mathcal{D}}(\mathcal{A})$ is only negligibly larger than $cn/|\mathcal{D}|$, where c is a constant, n is the number of active sessions[7] and $|\mathcal{D}|$ is the size of the dictionary $\mathcal{D}$.

5.1.2 Asymmetric-key setting

A two-party asymmetric-key based authenticated key exchange (2AAKE) protocol has a similar objective as with a 2PAKE protocol, *i.e.* to agree on a session key between a pair of communication parties. The long-lived keys of each protocol participant $U \in \mathcal{U}$ is now, however, a public key pk_U and the corresponding private key sk_U, instead of a password.

Provably secure 2AAKE protocols have been extensively studied in the past not only in the FTG sense, initiated by Bellare and Rogaway [9, 10], but also in other security models, such as those by Canetti and Krawczyk [16], and Shoup [37]. However, in this paper, we only consider 2AAKE protocols in the FTG model due to its close associations with the ROR model.

Generally speaking, an adversary in the FTG model is allowed to submit EXECUTE, SEND, REVEAL, CORRUPT and TEST queries. The first two types of queries (EXECUTE and SEND) are similar to those for 2PAKE in the ROR model. The others are defined as follows [10]:

- REVEAL(U^i): This query models leakage of information on specific session keys. If a session key is not defined for instance U^i or if a TEST query was asked to either U^i or its partner, then return $\perp$. Otherwise, return the session key held by the instance a *passive* attack in which the adversary eavesdrops on an honest execution of the protocol between a client instance U^i.

- CORRUPT(U^i): This query models the capability of an adversary being able to learn the long-term secrets of clients. The output of the query is the long-lived private key sk_U of the instance U^i.

- TEST(U^i): Let b be a bit chosen uniformly at random at the beginning of an experiment defining indistinguishability in the FTG model. If no session key for instance U^i is defined, or if either a REVEAL or a CORRUPT query was asked to either U^i or its partner, then return $\perp$. Otherwise, the output of the query is the session key for instance U^i if $b = 1$ or a random key from the same domain if $b = 0$.

[7]A session is said to be active if it involves SEND queries by the adversary.

As explained before, the adversary can query only once to the TEST oracle. However, the goal of the adversary is till the same, *i.e.* to guess the value of the hidden bit b used to answer TEST queries. Let SUCC denote the event in which an adversary guesses b correctly. The advantage of an adversary $\mathcal{A}$ in violating the indistinguishability of the 2AAKE protocol in the FTG sense, $\mathbf{Adv}^{\mathrm{ftg}}_{\mathsf{2AAKE}}(\mathcal{A})$, and the associated advantage function $\mathbf{Adv}^{\mathrm{ftg}}_{\mathsf{2AAKE}}(t, R)$ are then defined as in the password-based setting.

We say that a 2AAKE protocol is secure in the ROR model if the advantage $\mathbf{Adv}^{\mathrm{ftg}}_{\mathsf{2AAKE}}(\mathcal{A})$ is negligible (in the associated security parameter).

5.2 Four-Party Authenticated Key Exchange

We now define an ROR security model for 4PAKE by extending the work of Abdalla *et al.* for the three-party case [3].

In the 4PAKE setting, we assume that each protocol participant is a client $U \in \mathcal{U}$ or a trusted server $S \in \mathcal{S}$.[8] A protocol execution involves two client-server pairs from two distinct security domains. Each client shares a password with its domain server. (As with the two-party case, each client $U \in \mathcal{U}$ holds a password pwd_U, while each server $S \in \mathcal{S}$ holds a vector $pwd_S = \langle pwd_U \rangle_{U \in \mathcal{U}}$ with an entry for each client.) We also assume that a server has access to public information about other servers, such as their identities, public keys and so forth.

5.2.1 Indistinguishability of session keys

In order to model insider attacks, the set of clients $\mathcal{U}$ comprises two disjoint sets: $\mathcal{C}$, the set of honest clients, and $\mathcal{E}$, the set of malicious clients. We assume that all passwords of clients from the set $\mathcal{E}$ are known by the adversary [3].

The notion of partnering in the four-party setting is similar to that for the two-party setting, and thus will not be further discussed here.

The oracle queries in the ROR security model for 4PAKE are defined as follows:

- EXECUTE$(U_1^{i_1}, S_1^{j_1}, U_2^{i_2}, S_2^{j_2})$: This query models a passive attack in which the adversary eavesdrops on an honest execution of the protocol between client instances, U_1^i and U_2^i, and trusted server instances, S_1^j and S_2^j. The output of the query comprises messages that were exchanged during the honest execution of the protocol.

- SENDCLIENT(U^i, m): This query models an active attack in which the adversary may intercept a message and then either modify it, create a new one, or simply forward it to the intended participant. The output of the query is the message that the client instance U^i would generate upon receipt of message m.

- SENDSERVER(S^j, m): This query models an active attack against a server. The output of the query is the message that the server instance S^j would generate upon receipt of message m.

- CORRUPT(U^i): As defined in the FTG model in Section 5.1, except that the output of the query is the password pwd_U of the instance U^i. (Here, as with [7], we assume the weak corruption model in which the internal states of all instances of that client are not returned to the adversary.)[9]

- TEST(U^i): This query models the misuse of a session key by a user. Let b be a bit chosen uniformly at random at the beginning of an experiment defining indistinguishability in the ROR model. The output of the query is then the session key for participant instance U^i if

[8]Note that in Section 5.1, the set $\mathcal{U}$ includes both clients and servers. In the four-party case, however, the set $\mathcal{U}$ is restricted to only clients, since the goal of a 4PAKE protocol is to establish secure channels between two clients (rather than between a client and a server).

[9]We included CORRUPT queries in our ROR model so that it is consistent with the FTG model for 2AAKE protocols.

$b = 1$ or a random key from the same domain if $b = 0$. However, if no session key is defined for the client instance U^i, or if a CORRUPT query was asked to either U^i or its partner, then return the undefined symbol $\perp$.

Note that all executions completed after a CORRUPT query are answered with the relevant real session key.

The advantage of an adversary $\mathcal{A}$ in violating the indistinguishability of the 4PAKE protocol in the ROR sense, $\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{4PAKE},\mathcal{D}}(\mathcal{A})$, and the associated advantage function $\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{4PAKE},\mathcal{D}}(t, R)$ are then defined as in the two-party setting. The 4PAKE protocol is said to be secure if the advantage $\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{4PAKE},\mathcal{D}}(\mathcal{A})$ is negligible.

5.2.2 Key privacy with respect to servers

We stress that the servers involved in a 4PAKE protocol run in our ROR security model are trusted and assumed to be *honest-but-curious*. Since the servers have access to all the passwords within their respective security domains, it seems impossible to prevent any of them from impersonating a client from the same domain to another client of a different domain. However, in the security model, we allow the servers to launch passive attacks against any clients by intercepting their protocol messages.

We adopt the definition of key privacy from [3] which says that the session key shared between two instances should be known to only these two instance and no one else (including the trusted servers). Moreover, the adversary is allowed access to all passwords of the clients in the set $\mathcal{U}$, and the EXECUTE and SENDCLIENT oracles, but not the SENDSERVER oracle (which can be easily simulated by the adversary using the passwords). In order to capture the adversary's ability to tell apart a real session key from a random key, the adversary is allowed access to a TESTPAIR oracle defined as follows [3]:

- TESTPAIR(U_1^i, U_2^j): Let b be a bit chosen uniformly at random at the beginning of the experiment defining the notion of key privacy. If $b = 1$, the output of the query is the actual key shared between client instances U_1^i and U_2^j for a session in which the adversary performed only passive attacks. Else if $b = 0$, a random key from the same domain is output. However, if client instances U_1^i and U_2^j do not share the same key, then return the undefined symbol $\perp$.

Let $\mathcal{A}$ be an adversary which is given the passwords of all users and is allowed to ask multiple queries to the EXECUTE, SENDCLIENT and TESTPAIR oracles in an experiment defining the key privacy of the 4PAKE protocol. The advantage of the adversary $\mathcal{A}$ in violating the key privacy of the protocol, $\mathbf{Adv}^{\mathrm{kp}}_{\mathsf{4PAKE}}(\mathcal{A})$, and the associated advantage function $\mathbf{Adv}^{\mathrm{kp}}_{\mathsf{4PAKE}}(t, R)$ are defined as before.

Note that for simplicity of presentation, we will not consider the notion of perfect forward secrecy [7] in this paper. Defining such a notion is a straightforward exercise, see for example [2].

5.3 Security of MAC and Signatures

MAC. We consider security against a strong existential unforgeability under chosen-message attack (SUF-CMA). The adversary attacking a MAC scheme should not be able to create a new valid message-tag pair with non-negligible probability, even after seeing many such valid pairs [6]. Let $\mathrm{SUCC}_{\mathsf{MAC}}$ denote the event in which the adversary $\mathcal{A}$ is able to output a message m along with a tag μ such that: (i) $\mathrm{VER}(K; m; \mu) = 1$, and (ii) $\mathcal{A}$ had not previously requested a tag μ on the message m. The advantage of $\mathcal{A}$ in violating the strong existential unforgeability of the MAC scheme under chosen-message attacks [6] is defined as $\mathbf{Adv}^{\mathrm{suf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr[\mathrm{SUCC}_{\mathsf{MAC}}]$. The associated advantage function, $\mathbf{Adv}^{\mathrm{suf\text{-}cma}}_{\mathsf{MAC}}(t, q_{\mathrm{mac}}, q_{\mathrm{ver}})$, is then defined as the maximum value of $\mathbf{Adv}^{\mathrm{suf\text{-}cma}}_{\mathsf{MAC}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t, and asking at most q_{mac} and q_{ver} queries to the tag generation and verification oracles, respectively.

SIGNATURES. Similarly, a signature scheme is considered secure against existential unforgeability under an adaptive chosen message attack (EUF-CMA), if the adversary attacking the scheme could not create a new valid message-signature pair with non-negligible probability. This is so even if the adversary is allowed to ask for signing of multiple messages chosen adaptively [23]. Let $\textsc{Succ}_{\mathsf{Sig}}$ denote the event in which the adversary $\mathcal{A}$ is able to output a forged signature σ for a message m such that: (i) $\textsc{Ver}(pk; m; \sigma) = 1$, and (ii) $\mathcal{A}$ had not previously requested a signature on the message m from the signing oracle. The advantage of $\mathcal{A}$ in violating the existential unforgeability of the signature scheme under adaptive chosen-message attacks [23] is defined as $\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(\mathcal{A}) = \Pr[\textsc{Succ}_{\mathsf{Sig}}]$. The associated advantage function, $\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(t, q_{\text{sig}}, q_{\text{ver}})$, is then defined as the maximum value of $\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t, and asking at most q_{sig} queries to the signing oracle and at most q_{ver} queries to the verification oracle.

6 Security Analysis

Intuitively, the security of our 4PAKE protocol relies on the security of the employed 2PAKE and 2AAKE protocols, as well as the MAC and signature schemes. The security of the 2PAKE protocol and the unforgeability property of the MAC scheme ensure that both the servers receives the identity information and ephemeral public keys (in $M4_A$ and $M4_B$) in an authenticated and integrity protected manner. Furthermore, a secure signature scheme ensures that the signatures generated by the servers and forwarded to the clients (in $M5_A$ and $M5_B$) are genuine. Lastly, a secure 2AAKE (in $M6$) ensures that the final session key can only be established between the two authenticated clients, but no one else. In what follows, we show that even an adversary with attack capabilities defined in the ROR security model will not be able to learn any information about a session key established through our 4PAKE protocol. We also show that an honest-but-curious server will not gain any knowledge about any accepted (or valid) session key.

6.1 Indistinguishability of Session Keys

As the following theorem states, our generic 4PAKE protocol shown in Figure 1 is secure in the ROR model (as defined in Section 5.2), provided that the underlying primitives it uses are secure.

Theorem 1. *Let* 4PAKE *be the four-party password-based authenticated key exchange protocol. Let q_{exe} be the number of queries to the* EXECUTE *oracle of the* 4PAKE *protocol and q_{reveal} be the number of queries to the* REVEAL *oracle of the* 2AAKE *protocol. Let also q^{Mx}_{send} denote the number of* SENDCLIENT *or* SENDSERVER *queries related to message Mx of the* 4PAKE *protocol for $x \in \{3A, 3B, 4, 5, 6\}$. Then*

$$\begin{aligned}
&\mathbf{Adv}^{\text{ror}}_{\mathsf{4PAKE}}(t, q_{\text{exe}}, q^{Mx}_{\text{send}}) \leq \\
&\quad 2 \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{\text{exe}}, q^{M3_A}_{\text{send}}, q_{\text{exe}} + q^{M3_A}_{\text{send}}) \\
&\quad + 2 \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{\text{exe}}, q^{M3_B}_{\text{send}}, q_{\text{exe}} + q^{M3_B}_{\text{send}}) \\
&\quad + 2 \cdot q^{M4}_{\text{send}} \cdot \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(t, 2, 0) \\
&\quad + 2 \cdot q^{M5}_{\text{send}} \cdot \mathbf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(t, 2, 0) \\
&\quad + 2 \cdot \mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(t, q_{\text{exe}}, q^{M5,M6}_{\text{send}}, q_{\text{reveal}}, 1)
\end{aligned}$$

assuming the 2PAKE *and* 2AAKE *protocols, and the MAC and signature schemes used in the protocol are secure.*

Proof Theorem 1. Let $\mathcal{A}$ be an adversary against the indistinguishability of 4PAKE in the ROR sense. Our security proof is a sequence of security games simulated using techniques from Abdalla *et al.* [3]. For simplicity, we assume the set of honest users contains only users A and B. This can be easily extended to the more general case with essentially the same bounds.

We start with the real attack against the 4PAKE protocol, and end with a game with the adversary's advantage is negligible, and for which we can bound the difference in the adversary's advantage between any two consecutive games. For each game G_n, we define $\textsc{Succ}_n$ to be the event in which the adversary correctly guesses the hidden bit b used in the $\textsc{Test}$ queries (as defined in Section 5.2). We remark that our proof relies solely on the security properties of the underlying primitives our protocol uses, and thus does not assume the Random Oracle model.

Game G_0: This is the original attack game with respect to a given efficient adversary $\mathcal{A}$. By definition, we have

$$\mathbf{Adv}^{\mathrm{ror}}_{\mathsf{4PAKE}}(\mathcal{A}) = 2 \cdot \Pr[\textsc{Succ}_0] - 1$$

Game G_1: In this game, we model the adversary almost exactly the same as in game G_0. The only difference between these two games is that here, we replace the session key ssk_{A,S_A} output by 2PAKE by a random key ssk'_{A,S_A} in all of the sessions involving honest users. We show that the difference in success probability of the adversary $\mathcal{A}$ between games G_0 and G_1 is at most the probability of breaking the security of the underlying 2PAKE protocol between A and S_A.

Lemma 2. $|\Pr[\textsc{Succ}_1] - \Pr[\textsc{Succ}_0]| \leq \mathbf{Adv}^{ror}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{exe}, q^{M3_A}_{send}, q_{exe} + q^{M3_A}_{send})$.

Proof of Lemma 2. In order to prove this lemma, we simulate an adversary $\mathcal{A}_{\mathsf{2PAKE}}$ against the indistinguishability of the 2PAKE protocol using a distinguisher, $\mathcal{A}_1$, between games G_0 and G_1. Adversary $\mathcal{A}_{\mathsf{2PAKE}}$ first selects a bit b uniformly at random. It also chooses a password for each client in the system except A (according to the distribution $\mathcal{D}$) and generates an asymmetric-key pair for each server participating in the protocol. It then gives the chosen passwords to $\mathcal{A}_1$ and starts answering oracles queries from $\mathcal{A}_1$ as follows:

– $\textsc{SendClient}$ queries: If $\mathcal{A}_1$ makes a query on an instance of the 2PAKE protocol run between A and S_A, then $\mathcal{A}_{\mathsf{2PAKE}}$ responds by sending the corresponding query to its $\textsc{Send}$ oracle (as defined in the 2PAKE security model). If the query forces the given instance A or S_A to accept, then we also ask a $\textsc{Test}$ query to that instance, unless such a query had already been made to its partner. The output of the $\textsc{Test}$ query is subsequently used as the session key shared between A and S_A.

  On the other hand, if $\mathcal{A}_1$ issues a $\textsc{SendClient}$ query targeting an instance of the 2PAKE protocol run between B and S_B, $\mathcal{A}_{\mathsf{2PAKE}}$ responds using the password of client B that it has chosen at the beginning of the simulation.

  All remaining $\textsc{SendClient}$ queries by $\mathcal{A}_1$ can be answered either using the session key shared between A and S_A or the session keys generated during the execution of the 2PAKE protocol between B and S_B.

– $\textsc{SendServer}$ queries: $\mathcal{A}_{\mathsf{2PAKE}}$ can respond to these queries using the generated asymmetric-key pairs for servers by acting as the required signing oracles.

– $\textsc{Execute}$ queries: $\mathcal{A}_{\mathsf{2PAKE}}$ can easily answer these queries using its own $\textsc{Execute}$ oracle and the output of the relevant $\textsc{Test}$ queries, just as how $\textsc{SendClient}$ and $\textsc{SendServer}$ queries are responded.

– $\textsc{Test}$ queries: $\mathcal{A}_{\mathsf{2PAKE}}$ uses the bit b it has previoulsy selected and the session keys that it has computed to answer these queries.

Let b' be the output of $\mathcal{A}_1$. If $b' = b$, then $\mathcal{A}_{\mathsf{2PAKE}}$ outputs 1. Otherwise, it outputs 0.

Note that we omit $\textsc{Corrupt}$ queries from the game since we do not consider forward secrecy in this proof. Moreover, we assume that $\mathcal{A}_1$ has access to the passwords of all clients but A.

It is obvious that the probability of $\mathcal{A}_{\mathsf{2PAKE}}$ outputting 1 when its $\textsc{Test}$ oracle returns real keys is exactly the probability of $\mathcal{A}_1$ correctly guessing the hidden bit b in game G_0. Similarly, the probability of $\mathcal{A}_{\mathsf{2PAKE}}$ outputting 1 when its $\textsc{Test}$ oracle returns random keys is exactly the probability of $\mathcal{A}_1$ correctly guessing the hidden bit b in game G_1. The lemma follows by noticing that $\mathcal{A}_{\mathsf{2PAKE}}$ has at most time-complexity t and makes at most q_{exe} queries to its $\textsc{Execute}$ oracle, at most $q^{M3_A}_{\mathrm{send}}$ queries to its $\textsc{Send}$ oracle, and at most $q_{\mathrm{exe}} + q^{M3_A}_{\mathrm{send}}$ queries to its $\textsc{Test}$ oracle. □

Game G_2: We modify the previous game by replacing the session key ssk_{B,S_B} output by 2PAKE by a random key ssk'_{B,S_B} in all of the sessions involving honest users. Using similar arguments for proving the lemma in the previous game, we can prove the following lemma.

Lemma 3. $|\Pr[\textsc{Succ}_2] - \Pr[\textsc{Succ}_1]| \leq \mathbf{Adv}^{ror}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{exe}, q^{M3_B}_{send}, q_{exe} + q^{M3_B}_{send})$.

Game G_3: We now further modify the previous game as follows. Game G_3 is exactly the same as game G_2, except that in G_3, we modify the way the oracle instances respond to SendClient queries on $M4$ of our 4PAKE protocol. If the adversary makes a SendClient query containing a new MAC message-tag pair (forgery) not previously generated by an oracle, then we consider the MAC tag invalid and force the instance in question to terminate without accepting. As the following lemma shows, the difference between the current and previous games should be negligible if we use a secure MAC scheme.

Lemma 4. $|\Pr[\textsc{Succ}_3] - \Pr[\textsc{Succ}_2]| \leq q^{M4}_{send} \cdot \mathbf{Adv}^{suf\text{-}cma}_{\mathsf{MAC}}(t, 2, 0)$.

Proof of Lemma 4. We use a hybrid argument to proof this lemma. We define a sequence of hybrid experiments V_i, where $0 \leq i \leq q^{M4}_{\text{send}}$. (Note that we do not need to take into account Execute queries here, because they are used to simulate only passive attacks.) In experiment V_i, queries (to the SendClient oracle) in the first i sessions involving honest clients A and B are answered as in game G_3, and all other queries in the remaining sessions are answered as in game G_2. We remark that the hybrid experiments at the extremes (when $i = 0$ and $i = V_{q_s}$) are equivalent to games G_2 and G_3, respectively. Let P_i be the probability of the event Succ in experiment V_i. Since $P_0 = \Pr[\textsc{Succ}_2]$ and $P_{q_s} = \Pr[\textsc{Succ}_3]$, it follows that

$$|\Pr[\textsc{Succ}_3] - \Pr[\textsc{Succ}_2]| = \sum_{i=1}^{q_s} |P_i - P_{i-1}|.$$

Hence, it suffices to show that $|P_i - P_{i-1}|$ is at most $\mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(t, 2, 0)$, in order to prove the lemma. This can be achieved by assuming the existence of a distinguisher $\mathcal{A}^i_3$ for experiments V_{i-1} and V_i, and using it to build an adversary $\mathcal{A}^i_{\text{mac}}$ for breaking the security of the MAC scheme.

The description of the adversary $\mathcal{A}^i_{\text{mac}}$ is as follows. For the first $i-1$ sessions, the adversary $\mathcal{A}^i_{\text{mac}}$ chooses random values for the MAC key and is therefore can perfectly simulate the oracles given to $\mathcal{A}^i_3$, while imposing the restriction as defined for game G_3. In the i-th session, $\mathcal{A}^i_{\text{mac}}$ makes use of its MAC tag generation and verification oracles to answer queries from $\mathcal{A}^i_3$. In this session, if adversary $\mathcal{A}^i_3$ asks a SendClient query containing a message-tag pair not previously generated by adversary $\mathcal{A}^i_{\text{mac}}$, then $\mathcal{A}^i_{\text{mac}}$ halts and outputs the pair as its forgery. However, if no such pair is generated by $\mathcal{A}^i_3$, we output a failure indication. For all remaining sessions, $\mathcal{A}^i_{\text{mac}}$ simulates all oracles exactly as in game G_2, using actual MAC keys, to answer queries from $\mathcal{A}^i_3$.

Let F_1 be the event in which a message-tag pair is considered valid in experiment V_{i-1} but invalid in experiment V_i. It is then not difficult to see that $\Pr[F_1]$ is at most the probability that adversary $\mathcal{A}^i_{\text{mac}}$ can forge a new message-tag pair under a chosen-message attack. Since $\mathcal{A}^i_{\text{mac}}$ has time-complexity t and makes at most two queries to its MAC tag generation oracle (to answer the SendClient queries from $\mathcal{A}^i_1$ in one session) and no queries to its verification oracle, we have $\Pr[F_1] \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(t, 2, 0)$. One also sees that

$$\Pr[\textsc{Succ}_{V_{i-1}} \wedge \neg F_1] = \Pr[\textsc{Succ}_{V_i} \wedge \neg F_1]$$

since experiments V_{i-1} and V_i proceed identically until F_1 occurs. Therefore, by Lemma 1 of [38] (also known as the Difference Lemma), we have

$$|\Pr[\textsc{Succ}_{V_{i-1}}] - \Pr[\textsc{Succ}_{V_i}]| \leq \Pr[F_1].$$

Our lemma then follows by noticing that there are at most q^{M4}_{send} experiments, where $M4 = M4_A + M4_B$. □

Game G_4: In this game, we modify the way the oracle instances respond to SendClient queries on $M5$ of our 4PAKE protocol. This implies that if the adversary makes a SendClient query containing a new signature not previously generated by an oracle, then we consider the signature invalid and force the instance in question to terminate without accepting. Using similar arguments for proving the lemma in game G_3, it is straightforward to prove the following lemma.

Lemma 5. $|\Pr[\textsc{Succ}_4] - \Pr[\textsc{Succ}_3]| \leq q_{send}^{M5} \cdot \mathbf{Adv}_{\mathsf{Sig}}^{euf\text{-}cma}(t, 2, 0)$.

Proof of Lemma 5. As before, we define a sequence of hybrid experiments V_i, where $0 \leq i \leq q_{\text{send}}^{M5}$. We assume the existence of a distinguisher $\mathcal{A}_4^i$ between games G_3 and G_4. We then simulate an adversary $\mathcal{A}_{\text{sig}}^i$ that breaks the security of the signature scheme.

The description of adversary $\mathcal{A}_{\text{sig}}^i$ is analogous to $\mathcal{A}_{\text{mac}}^i$ in game G_3. During the first $i-1$ sessions, $\mathcal{A}_{\text{sig}}^i$ generates itself asymmetric-key pairs and acts as the Sig and Ver oracles to respond queries from $\mathcal{A}_4^i$. However, if $\mathcal{A}_4^i$ asks a query containing a message-signature pair that was not generated by the Sig oracle, the relevant client instance is terminated without accepting (as defined for game G_4). During the i-th session, $\mathcal{A}_{\text{sig}}^i$ forwards queries submitted by $\mathcal{A}_4^i$ to its own Sig and Ver oracles. If adversary $\mathcal{A}_4^i$ asks a SendClient query containing a message-signature pair not previously generated by $\mathcal{A}_{\text{sig}}^i$, then $\mathcal{A}_{\text{sig}}^i$ halts and outputs the pair as its forgery. However, if no such pair is generated by $\mathcal{A}_4^i$, we output a failure indication. For all remaining sessions, $\mathcal{A}_{\text{sig}}^i$ simulates all oracles exactly as in game G_3, in which actual signing keys of servers are used to respond queries from $\mathcal{A}_4^i$.

Here we assume that the adversary against the security of a signature scheme has time-complexity t and makes at most four queries to its signing oracle and no queries to its verification oracle. Moreover, there are at most q_{send}^{M5} experiments in game G_4, where $M5 = M5_A + M5_B$. The lemma shows that the difference between the current and previous games should be negligible if we use a secure signature scheme. □

Game G_5: This game is identical to the previous game, except that we replace the session key $ssk_{A,B}$ (output by the 4PAKE protocol) by a random key $ssk'_{A,B}$ in all of the sessions. As the following lemma shows, the difference in success probability between the current and previous games is at most the probability of breaking the security of the underlying signature-based 2AAKE protocol between A and B.

Lemma 6. $|\Pr[\textsc{Succ}_5] - \Pr[\textsc{Succ}_4]| \leq \mathbf{Adv}_{\mathsf{2AAKE}}^{ftg}(t, q_{exe}, q_{send}^{M5,M6}, q_{reveal}, 1)$.

Proof of Lemma 6. Let $\mathcal{A}_5$ be a distinguisher between games G_5 and G_4. We can simulate an adversary $\mathcal{A}_{\mathsf{2AAKE}}$ against the indistinguishability of the signature-based 2AAKE protocol (performed between A and B) with $\mathcal{A}_5$, using the proof techniques shown in game G_1. As before, $\mathcal{A}_{\mathsf{2AAKE}}$ starts by choosing a bit b uniformly at random. Moreover, it selects a password for each client, as well as generates an asymmetric-key pair for each server in the system except S_A and S_B. Note that $\mathcal{A}_{\mathsf{2AAKE}}$ can trivially distinguish an actual session key shared between A and B from a random value if it is allowed to access to S_A and S_B's signing (or private) keys. The generated passwords and asymmetric-key pairs are given to adversary $\mathcal{A}_5$. $\mathcal{A}_{\mathsf{2AAKE}}$ then responds to queries from $\mathcal{A}_5$ as follows:

– SendClient and SendServer queries: If $\mathcal{A}_5$ submits a SendServer query involving an instance of $M5$ involving honest-but-curious S_A and S_B, $\mathcal{A}_{\mathsf{2AAKE}}$ answer the query using its access to the relevant signing oracles. If $\mathcal{A}_5$ makes a SendClient query on an instance of the 2AAKE protocol run between A and B, then $\mathcal{A}_{\mathsf{2AAKE}}$ responds by sending the corresponding query to its Send oracle.

  If the query forces the given instance A or B to accept for the first time and such a query has not already been made to its partner, $\mathcal{A}_{\mathsf{2AAKE}}$ responds using its Test oracle. The value returned by the Test oracle is subsequently used as the session key between A and B. However, if it is not the first time the given instance A or B accepts (and such a query has not already been made to its partner), $\mathcal{A}_{\mathsf{2AAKE}}$ makes use of its Reveal oracle instead to obtain the relevant session key.

All remaining SendClient queries by $\mathcal{A}_5$ can be answered using the passwords and asymmetric keys that $\mathcal{A}_{\mathsf{2AAKE}}$ generated before.

- Execute queries: $\mathcal{A}_{\mathsf{2AAKE}}$ can easily answer these queries using its own Execute oracle and the output of the relevant Test or Reveal queries, just as how SendClient and SendServer queries are responded.
- Test queries: $\mathcal{A}_{\mathsf{2AAKE}}$ uses the bit b it has previously selected and the session keys that it has computed to answer these queries.

Let b' be the output of $\mathcal{A}_1$. If $b' = b$, then $\mathcal{A}_{\mathsf{2PAKE}}$ outputs 1. Otherwise, it outputs 0. The lemma follows by noticing that $\mathcal{A}_{\mathsf{2AAKE}}$ has at most time-complexity t and asks at most q_{exe} queries to its Execute oracle, at most $q_{\text{send}}^{M5,M6}$ queries to its Send oracle, at most q_{reveal} queries to its Reveal oracle, and at most 1 query to its Test oracle. Clearly, $\Pr[\text{Succ}_5] = \frac{1}{2}$ and adversary $\mathcal{A}_{\mathsf{2AAKE}}$ should have negligible advantage if we use a secure 2AAKE protocol. □

All the above lemmas yield the result in Theorem 1. □

We remark that our security analysis is a "generic" one for a generic 4PAKE protocol, in the sense that we do not make use of any mathematical hard problems. The latter typically depends on the type of key materials that are used to compute session keys, for example Diffie-Hellman key components can be associated with the Decisional Diffie-Hellman problem. Our analysis assumes that the 2PAKE and 2AAKE protocols on which the 4PAKE protocol is based are secure. This implies that the 4PAKE protocol inherit security properties from the 2PAKE and 2AAKE protocols. For example, if an instantiation of the underlying 2AAKE protocol provides properties such as forward secrecy and key confirmation, then the associated 4PAKE protocol can also be proved secure using an extended ROR model (from the model we defined in Section 5.2) that considers these properties.

6.2 Key Privacy against Servers

As explained in Section 5.2, we assume that the servers are honest-but-curious. Hence, we should also show that if the 4PAKE protocol is executed as expected and does not abort, the servers should not gain any knowledge about the resulting session key.

Theorem 7. *Let* 4PAKE *be the four-party password-based authenticated key exchange protocol. Then*

$$\mathbf{Adv}^{\text{kp}}_{\mathsf{4PAKE}}(t, q_{\text{exe}}, q_{\text{send}}^{Mx}) \leq 2 \cdot \mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(t, q_{\text{exe}}, q_{\text{send}}^{M5,M6}, q_{\text{reveal}}, 1)$$

where parameters are defined as in Theorem 1, and assuming the 2PAKE *and* 2AAKE *protocols, and the MAC and signature schemes are secure.*

Proof of Theorem 7. We can use arguments similar to those in game G_5 in the proof of Theorem 1. Thus, we do not repeat the details here.

Succinctly, assuming that the 2AAKE protocol used between the clients is secure, *i.e.* inherits the indistinguishability property, the servers should not be able to distinguish an accepted session key between users A and B from a random key. This holds if users A and B were honest users and the servers performed only passive attacks, *i.e.* making only Execute queries. □

7 Efficiency Analysis

In this section, we examine and show that the communication and computational costs of our generic 4PAKE protocol are reasonable compared to existing protocols. For this purpose, we instantiate a concrete 4PAKE protocol based on SPEKE [26] and the signature-based authenticator of [16] (a 2PAKE protocol and a 2AAKE protocol, respectively).[10]

Let $\mathbf{C} \leftrightarrow \mathbf{C}$ denote the interaction between two clients (in 4PAKE), or a client and an application server (in PKCROSS). Let $\mathbf{S} \leftrightarrow \mathbf{S}$ denote the interaction between two servers (in 4PAKE), or two KDCs (in PKCROSS). We let also $\mathbf{C} \leftrightarrow \mathbf{S}$ represent the interaction between a client and a server (or KDC). Table 2 shows the number of message flows for each type of the mentioned interactions. It is not suprising that our protocol has smaller number of message flows than that of PKCROSS. This is because our approach does not require interaction between the local and the remote servers. Further, Kerberos is designed such that a client must obtain a ticket-granting ticket and a service ticket before it can access a target application server. We can also see from Table 2 that the communication overhead for KDCs in PKCROSS is higher than that of our protocol, that is 8 incoming/outgoing messages compared to 4, respectively. This implies that our protocol is likely to be more scalable in the sense that the server can afford to serve more clients given a fix communication bandwidth.

Table 2: Numbers of message flows in PKCROSS and our protocol.

Protocol	$\mathbf{C} \leftrightarrow \mathbf{S}$	$\mathbf{C} \leftrightarrow \mathbf{C}$	$\mathbf{S} \leftrightarrow \mathbf{S}$	Total
PKCROSS [25]	6	2	2	10
Ours	4	4	0	8

Our generic protocol is understandably less efficient than one that is based on a standard, non-compositional approach; although in return, we achieve protocol inter-operability by reusing existing two-party protocols and our protocol is easier to analyse. However, Table 3 shows that our approach is still comparable to most existing C2C-PAKE protocols (in which two servers share a long-term symmetric key) in terms of the number of message flows exchanged.

Table 3: Numbers of message flows in C2C-PAKE.

Protocol	$\mathbf{C} \leftrightarrow \mathbf{S}$	$\mathbf{C} \leftrightarrow \mathbf{C}$	$\mathbf{S} \leftrightarrow \mathbf{S}$	Total
Byun-Lee-Lim [15]	6	2	0	8
Yin-Bao [43]	4	0	2	6
Feng-Xu [22]	5	3	0	8

We next turn to computational cost. Table 4 gives a summary of the cryptographic operations involved in our 4PAKE protocol and PKCROSS. Here, we use **2C** to denote both the clients and **2S** to denote both the servers that participate in a protocol run.

Depending on the choices of the public key encryption & signature schemes and the symmetric encryption scheme, as well as the parameters for the Diffie-Hellman modular exponentiation, the overall computational overhead of our protocol seems to be comparable to or slightly more than that of PKCROSS. Breaking this down, the overhead at the servers in our protocol may be slightly less than that of PKCROSS. However, the clients in our protocol have to perform signature verification and Diffie-Hellman exponentiation (during key agreement), and hence higher overhead is incurred compared to that of PKCROSS, in which the clients perform only symmetric key operations. This is a trade-off between performance and usability (since the clients in our protocol avoid public key management and depend on only passwords).

[10]We assume that in real world implementation, the last message of the SPEKE protocol (between a client and a server) can be combined with $M4$ illustrated in Figure 1.

Table 4: Cryptographic operations in PKCROSS and our protocol.

Protocol	2C	2S	Total
PKCROSS [25]			
– Signing / public key decryption	0	3	3
– Verifying / public key encryption	0	7	7
– Symmetric key encryption / decryption	10	9	19
Ours			
– Signing / public key decryption	0	2	2
– Verifying / public key encryption	2	0	2
– Symmetric key encryption / decryption	2	2	4
– Diffie-Hellman exponentiation	8	4	12

8 Applications

Our approach of 4PAKE seems to be applicable to many cross-domain authenticated key exchange scenarios.

8.1 Client-to-Client TLS

One of the most common password-based user authentication mechanisms is the use of username/passwords through the TLS (or known as SSL) protocol [20]. This method has been widely used in applications, such as web-based emails, online booking and Internet banking, for mutual authentication and key establishment between a user and a server. Typically, a user first establishes a secure TLS channel with a server by performing the server-authenticated TLS handshake (using the server's public key certificate). Note that at this point, the user is still not authenticated by the server. The user then transmits her authentication information, such as a username and a password, to the server (in plaintext) through the TLS channel, so that the server can verify the authenticity of the user. However, this approach is restricted to only the two-party client-server setting.

Using our compositional approach, we envisage that a client-to-client TLS protocol can be constructed in a natural way by using the following building blocks:

- certTLS, the hybrid server-authenticated TLS handshake and username/password approach (between a client and a server, as described above);
- dhTLS, the classic Diffie-Hellman authenticated key exchange TLS handshake protocol (between two clients) [20];

In the client-to-client TLS protocol, a client first authenticates to its domain server using the certTLS protocol and obtains its credential (or authenticated data) in the form of a signature. Using the credential, the client then performs the dhTLS protocol with the intended remote client.

To instantiate the client-to-client TLS protocol from our generic 4PAKE protocol from Figure 1, we simply replace the 2PAKE protocol by the certTLS protocol; the 2PAKE protocol by the dhTLS protocol; (epk_A, esk_A) and (epk_B, esk_B) by (g^a, a) and (g^b, b), respectively. Here, g is a generator of a group $\mathbb{G}$ in which the Decisional Diffie-Hellman problem is hard, and a, b are chosen at random from $\{1, \ldots, |\mathbb{G}|\}$ by users A and B, respectively.

We note that protocol (i) can be replaced by a password-based TLS protocol for better usability and stronger security [2]. Instead of sending a username/password in clear through a TLS channel, a password-based TLS protocol binds a password directly into a protocol run (which will not expose any useful information about the password).[11]

[11] This prevents a user from unintentionally revealing her password to a bogus server, in the event of the server being able to be authenticated to the user through a fake certificate, for example.

8.2 Email Communication

Typically, sensitive information, such as financial data, medical records, proprietary corporate information and so on, is exchanged between business partners and customers through emails, and thus such information must be delivered to its destination in a secure manner. One natural solution is by encrypting email communication. Very often, however, it is not clear how this is achieved in real world when a sender and her targeted recipient do not share the same domain name in their email addresses.

Google provides a paid service called Google Message Encryption [24] that allows their users to send encrypted emails. The Google Message Encryption service secures outgoing email to a data center using a secure SSL/TLS connection. At the data center, messages are encrypted for each intended recipient and are delivered to the recipients' inbox. Recipients can then view the messages by providing their respective passwords associated with their gmail accounts. However, this works only when both the sender and the recipient of an encrypted message have a gmail account.

Alternatively, one can also use Pretty Good Privacy (PGP) [33], a well-known program for encrypting emails using public key encryption. However, we must assume that the sender has a copy of the recipient's public key and there exists a means to verify the authenticity of the public key.

A simple twist of our approach allows the sender to encrypt a message for a recipient belonging to a different domain name. Moreover, the sender does not require the recipient's public key but simply relying on her existing email account password. We envisage that an email that is sent across a domain can be encrypted using a session key generated from executing our 4PAKE protocol.

We discussed in Section 5.2 that a domain server typically has knowledge of all passwords associated to users within its domain. Hence, we exploit the fact that a server can impersonate its user to another user of a different domain. We require that the recipient's email server acts on his behalf (or "impersonates" the recipient) to establish a secret key with the sender. When executing the 4PAKE protocol, the sender creates a secure channel with her email server by using her email password; while the recipient's server plays a double role (email server and recipient) in the protocol while performing key exchange with the sender. The established session key is then used to encrypt the message that needs to be protected. The recipient can subsequently retrieve the encrypted message and the encryption key from his email server.

8.3 Mobile Phone Communication

4PAKE can also be used to secure communications between mobile clients that subscribe to different mobile service providers. In mobile networks, such as the 2G and 3G telecommunication networks [1], each mobile client subscribes to a Home Location Register (HLR). The mobile client also shares a secret key (stored in the SIM card) with the HLR, and uses this key to authenticate itself to the HLR when the mobile device is turned on. Meanwhile, different HLRs are connected via wired networks.

Our 4PAKE protocol fits nicely into the mobile network structure. That is, two mobile clients that subscribe to different HLRs can establish a secure communication channel by executing the protocol. Further, the protocol provides key privacy with respect to servers, *i.e.* HLRs. The latter is a desirable feature that is not supported by current mobile communication systems.

8.4 Instant Messaging

Yet another example application of 4PAKE is to secure instant messaging (IM) between two users. Exchanging messages and monitoring availability of a list of users in real-time through IM services have been very popular for a relatively long time. There are many free public domain IM services, such as AOL Instant Messenger (AIM), ICQ, MSN Messenger (Windows Messenger in XP), and Yahoo! Instant Messenger (YIM). In [29], Mannan and van Oorschot proposed the use of a three-party password-based authenticated key exchange for securing public IM, assuming that two communication parties are using the same IM service. However, it is not uncommon that two users, wishing to communicate with each other, subscribe to two different IM services. Our four-party key exchange approach is just what is needed to secure communication in such a scenario.

9 Conclusions

The example applications that we have given show that there is a growing need and importance to the use of a four-party password-based authenticated key exchange protocol for cross-domain communications. In this paper, we proposed a compositional approach to constructing such a protocol, which allows two users, who do not share a common password and from different security domains, to establish a secret key in an authenticated and secure manner. We presented a provably secure four-party password-based authenticated key exchange protocol by using two-party key exchange protocols as building blocks. Our protocol is reasonably efficient and we believe that our approach is useful for extending a legacy system with two-party protocols to the four-party setting.

References

[1] 3GPP. The mobile broadband standard. http://www.3gpp.org/.

[2] M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Provably secure password-based authentication in TLS. In *Proceedings of the 1st ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 35–45. ACM Press, Mar 2006.

[3] M. Abdalla, P. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *Proceedings of the 8th International Workshop on Theory and Practice in Public Key Cryptography (PKC)*, pages 65–84. Springer LNCS 3386, Jan 2005.

[4] M. Backes, I. Cervesato, A.D. Jaggard, A. Scedrov, and J. Tsay. Cryptographically sound security proofs for basic and public-key Kerberos. *International Journal of Information Security*, 10(2):107–134, Jun 2011.

[5] M. Bellare, R. Canetti, and P. Rogaway. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing (STOC)*, pages 419–428. ACM Press, May 1998.

[6] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, Dec 2000.

[7] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology – Proceedings of EUROCRYPT*, pages 139–155. Springer LNCS 1807, May 2000.

[8] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Computer and Communications Security Conference (CCS)*, pages 62–73. ACM Press, Nov 1993.

[9] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology – Proceedings of CRYPTO*, pages 232–249. Springer LNCS 773, Aug 1993.

[10] M. Bellare and P. Rogaway. Provably secure session key distribution – the three party case. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing (STOC)*, pages 57–66. ACM Press, May 1995.

[11] A. Bhargav-Spantzel, A.C. Squicciarini, and E. Bertino. Establishing and protecting digital identity in federation systems. *Journal of Computer Security*, 14(3):269–300, May 2006.

[12] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology – Proceedings of EUROCRYPT*, pages 156–171. Springer LNCS 1807, May 2000.

[13] F. Butler, I. Cervesato, A.D. Jaggard, A. Scedrov, and C. Walstad. Formal analysis of Kerberos 5. *Theoretical Computer Science*, 367(1–2):57–87, Nov 2006.

[14] J.W. Byun, I.R. Jeong, D.H. Lee, and C.S. Park. Password-authenticated key exchange between clients with different passwords. In *Proceedings of the 4th International Conference on Information and Communication Security (ICICS)*, pages 134–146. Springer LNCS 2513, Dec 2002.

[15] J.W. Byun, D.H. Lee, and J.I. Lim. EC2C-PAKA: An efficient client-to-client password-authenticated key agreement. *Information Sciences*, 177(19):3995–4013, Oct 2007.

[16] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. In *Advances in Cryptology – Proceedings of EUROCRYPT*, pages 453–474. Springer LNCS 2045, May 2001.

[17] T. Cao, T. Quan, and B. Zhang. Cryptanalysis of some client-to-client password-authenticated key exchange protocols. *Journal of Networks*, 4(4):263–270, Jun 2009.

[18] I. Cervesato, A.D. Jaggard, A. Scedrov, J. Tsay, and C. Walstad. Breaking and fixing public-key Kerberos. *Information and Computation*, 206(2–4):402–424, Feb 2008.

[19] L. Chen. A Weakness of the Password-Authenticated Key Agreement between Clients with Different Passwords Scheme. Circulated at *The 27th SC27/WG2 Meeting* in Paris, France. ISO/IEC JTC1/SC27 N3716, Oct 2003.

[20] T. Dierks and E. Rescorla. The TLS protocol version 1.2. *The Internet Engineering Task Force (IETF)*, RFC 5246, Aug 2008.

[21] C. Ellison and B. Schneier. Ten risks of PKI: What you're not being told about public key infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[22] D. Feng and J. Xu. A new client-to-client password-authenticated key agreement protocol. In *Proceedings of the 2nd International Workshop on Coding and Cryptology (IWCC)*, pages 63–76. Springer LNCS 5557, Jun 2009.

[23] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, Apr 1988.

[24] Google. Google message encryption. http://www.google.com/postini/encryption.html.

[25] M. Hur, B. Tung, T. Ryutov, C. Neuman, A. Medvinsky, G. Tsudik, and B. Sommerfeld. Public key cryptography for cross-realm authentication in Kerberos. *Internet Draft*, http://tools.ietf.org/html/draft-ietf-cat-kerberos-pk-cross-08, Nov 2001.

[26] D.P. Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, Oct 1996.

[27] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5). *IETF*, RFC 1510, Sep 1993.

[28] H. Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE-protocols. In *Advances in Cryptology – Proceedings of CRYPTO*, pages 400–425. Springer LNCS 2729, Aug 2003.

[29] M. Mannan and P.C. van Oorschot. A protocol for secure public instant messaging. In *Proceedings of the 10th International Conference on Financial Cryptography and Data Security (FC)*, pages 20–35. Springer LNCS 4107, Mar 2006.

[30] MIT. Kerberos: The network authentication protocol. http://web.mit.edu/kerberos/.

[31] B.C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications*, 32(9):33–38, Sep 1994.

[32] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos network authentication service (V5). *The Internet Engineering Task Force (IETF)*, RFC 4120, Jul 2005.

[33] OpenPGP Alliance. Technical information. http://www.openpgp.org/technical/.

[34] R.C.-W. Phan and B.-M. Goi. Cryptanalysis of an improved client-to-client password-authenticated key exchange (C2C-PAKE) scheme. In *Proceedings of the 3rd International Conference on Applied Cryptography and Network Security (ACNS)*, pages 33–39. Springer LNCS 3531, Jun 2005.

[35] R.C.-W. Phan and B.-M. Goi. Cryptanalysis of two provably secure cross-realm C2C-PAKE protocols. In *Progress in Cryptology – Proceedings of INDOCRYPT*, pages 104–117. Springer LNCS 4329, Dec 2006.

[36] G. Price. PKI challenges: An industry analysis. In *Proceedings of the 4th International Workshop for Applied PKI (IWAP 2005)*, pages 3–16. Volume 128 of FAIA, IOS Press, Sep 2005.

[37] V. Shoup. On formal models for secure key exchange. *IBM Research Report*, RZ 3120, Apr 1999.

[38] V. Shoup. OAEP reconsidered. *Journal of Cryptology*, 15(4):223–249, Apr 2002.

[39] S. Wang, J. Wang, and M. Xu. Weaknesses of a password-authenticated key exchange protocol between clients with different passwords. In *Proceedings of the 2nd International Conference on Applied Cryptography and Network Security (ACNS)*, pages 414–425. Springer LNCS 3089, Jun 2004.

[40] F.L. Wong and H.W. Lim. Identity-based and inter-domain password authenticated key exchange for lightweight clients. In *Proceedings of the 3rd IEEE International Symposium on Security in Networks and Distributed Systems (SSNDS 2007)*, pages 544–550. IEEE Computer Society Press, May 2007.

[41] S. Wu and Y. Zhu. Client-to-client password-based authenticated key establishment in a cross-realm setting. *Journal of Networks*, 4(7):649–656, Sep 2009.

[42] H. Yeh and H. Sun. Password authenticated key exchange protocols among diverse network domains. *Computers and Electrical Engineering*, 31(3):175–189, May 2005.

[43] Y. Yin and L. Bao. Secure cross-realm C2C-PAKE protocol. In *Proceedings of the 11th Australasian Conference on Information Security and Privacy (ACISP)*, pages 395–406. Springer LNCS 4058, Jul 2006.

[44] L. Zhu and B. Tung. Public key cryptography for initial authentication in Kerberos (PKINIT). *The Internet Engineering Task Force (IETF)*, RFC 4556, Jun 2006.