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

August 25, 2013

Abstract

We revisit the problem of secure cross-domain 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 generic framework for designing *four-party password-based authenticated key exchange* (4PAKE) protocols. Our framework 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 use to 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 (2PAKE) protocol and two-party asymmetric-key/symmetric-key based key exchange (2A/SAKE) protocol as black boxes, we combine them to obtain generic and provably secure 4PAKE protocols.

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 these scenarios, 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 a 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 [1, 6, 11, 26, 32], it is not clear how they can be directly applied to establish a secure cross-domain communication channel.

In this work, 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. We assume that each

*An extended abstract of this paper has appeared in the Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM), 2013.

security or administrative domain[1] 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 the user does not necessarily own a public key certificate. Further, 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 [13, 45], 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, 46] and C2C-PAKE [14, 45]. The latter either requires a user to obtain a public key certificate, or assumes 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 new generic framework for designing 4PAKE protocols that meet the requirements discussed above. Our framework 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 or symmetric-key authenticated key exchange (2A/SAKE) protocol — into a secure four-party password-based authenticated key exchange protocol. We describe the cryptographic primitives and building blocks used in this work in Section 3 and give an overview of our approach in Section 4. The details of our new framework and some concrete instantiations of generic 4PAKE protocols are presented in Section 5.

Moreover, we define a security model for our generic 4PAKE protocols based on the Real-Or-Random (ROR) security model [1] and show that our protocols are secure in the model. Our security definitions and analyses are shown in Sections 6 and 7, respectively. Our security proofs rely solely on the security properties of the cryptographic primitives on which our protocols are based, and do not make use of the Random Oracle model [7].

In an extended abstract of this work [19], we presented a 4PAKE protocol that uses 2PAKE and 2AAKE as building blocks and showed that it is provably secure in our security model; however, no implementation results were given. This paper presents the full version of our results where we further propose two new 4PAKE protocols: a 4PAKE variant that optimises our earlier protocol shown in [19] by allowing key reuse; and a 4PAKE protocol that builds on 2PAKE and 2SAKE (instead of 2PAKE and 2AAKE). We also implement our protocols and provide some empirical performance evaluation in Section 8. Our experimental results show that all of the 4PAKE protocols have only slightly higher (between 1.05× and 1.1×) client-side computational overhead than that of public key Kerberos. However, in return, our protocols are approximately 1.1× to 1.2× faster than public key Kerberos from the server's perspective, and thus, can *scale* better. Further, our approach is more *usable* in the sense that the user only needs to remember a password, and *flexible* in terms of reuse of existing two-party key exchange protocols.

We give some concluding remarks in Section 10.

2 Previous Work

The design and analysis of authentication and key exchange protocols are fundamental research problems in information security and have been extensively studied over the last three decades (see for example [8, 21, 29, 30, 41] and many others). In this paper, we focus on designing authenticated key exchange protocols for protecting cross-domain communication between two users belonging to different security domains.

[1]Here a domain is analogous to a Kerberos *realm*. It is a logical network that defines a group of system users, computers and/or servers, such that it is easy for a system administrator to enforce security policies and to protect these entities.

2.1 Kerberos

Initially developed by an MIT research team led by Miller and Neuman, Kerberos[2] [32, 33] 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 [46] 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 [46]. 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.[3]
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 4.

[2] `http://web.mit.edu/kerberos/`

[3] 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.

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.* [13], and Yin and Bao [45]. 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 [32], 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 [14, 23, 43].

2.3 Others

Our work shares similar spirit with proposals on inter-domain password-based key exchange in the public key setting by Yeh and Sun [44], and Wong and Lim [42]. 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.

3 Preliminaries

3.1 Cryptographic Primitives

We first describe some cryptographic primitives required for our protocols. In our description, we let κ denote a security parameter.

Message Authentication Codes A message authentication code (MAC) scheme is a tuple of 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;
- $\textsc{Mac}(K; m)$, 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.

Security. We consider security against strong existential unforgeability under a 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 [4]. Let $\textsc{Succ}_{\mathsf{MAC}}$ denote the event in which the adversary $\mathcal{A}$ is able to output a message m along with a tag μ such that: (i) $\textsc{Ver}(K; m; \mu) = 1$, and (ii) $\mathcal{A}$ had not previously requested a tag μ on the message m. (Clearly, we assume that $\mathcal{A}$ has no knowledge of the key K.) The advantage of $\mathcal{A}$ in violating the strong existential unforgeability of the MAC scheme under chosen-message attacks [4] is defined as $\mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(\mathcal{A}) = \Pr[\textsc{Succ}_{\mathsf{MAC}}]$. The associated advantage function, $\mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(t, q_{\text{mac}}, q_{\text{ver}})$, is then defined as the maximum value of $\mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(\mathcal{A})$ 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.

Digital Signatures A signature scheme is a tuple of polynomial-time algorithms (GEN, SIG, VER) satisfying the following:

- $\text{GEN}(1^\kappa)$, the key generation algorithm, takes as input the security parameter 1^κ and outputs a pair of public/private keys (pk, sk);
- $\text{SIG}(sk; m)$, the signing algorithm, takes as input a private key sk and a message $m \in \{0,1\}^*$, and outputs a signature σ;
- $\text{VER}(pk; m; \sigma)$, 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.

SECURITY. 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 without knowing the corresponding signing key. This is so even if the adversary is allowed to ask for signing of multiple messages chosen adaptively [24]. Let $\text{SUCC}_{\mathsf{Sig}}$ denote the event in which the adversary $\mathcal{A}$ is able to output a forged signature σ for a message m such that: (i) $\text{VER}(pk; m; \sigma) = 1$, 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 [24] is defined as $\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(\mathcal{A}) = \Pr[\text{SUCC}_{\mathsf{Sig}}]$. 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.

Authenticated Encryption An authenticated encryption (AE) scheme (a symmetric-key encryption scheme that provides both privacy and authenticity/integrity) is a tuple of polynomial-time algorithms (GEN, ENC, DEC) satisfying the following:

- $\text{GEN}(1^\kappa)$, the key generation algorithm, takes as input the security parameter 1^κ and outputs a symmetric key sk;
- $\text{ENC}(sk; m)$, the encryption algorithm, takes as input a secret key sk and a message $m \in \{0,1\}^*$, and outputs a ciphertext c;
- $\text{DEC}(sk; c)$, the decryption algorithm, takes as input a secret key sk and a ciphertext c; it outputs m if c is authenticated, or 0 otherwise indicating an invalid ciphertext.

SECURITY. For an authenticated encryption (AE) scheme, we consider both privacy and authenticity/integrity [5]. In terms of privacy, we consider the conventional notion of indistinguishability under a chosen-plaintext attack (IND-CPA). Let $b \leftarrow \{0,1\}$ denote a random bit. An adversary is allowed access to a left-or-right (LR) oracle, which returns $\text{ENC}(K; M_b)$ upon receiving a pair of messages (M_0, M_1) from the adversary. However, the adversary has no knowledge of the key K and its goal is to guess the value of b. The advantage function, $\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(t, q_{\text{lr}})$, is defined as the maximum value of $\mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t, and asking at most q_{lr} queries to the LR oracle.

To define authenticity, we consider an adversary who has access to an encryption oracle ENC, which returns the encryption of any message m chosen by the adversary; and a decryption oracle DEC, which tells if the decryption of a ciphertext c is successful or not. The goal of the adversary is to forge a ciphertext c^* such that: (i) c^* can be successfully decrypted; and (ii) c^* is not the output of the encryption oracle. The associated advantage function, $\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{AE}}(t, q_{\text{enc}}, q_{\text{dec}})$, is defined as the maximum value of $\mathbf{Adv}^{\text{euf-cma}}_{\mathsf{AE}}(\mathcal{A})$ over all $\mathcal{A}$ with time-complexity at most t, and asking at most q_{enc} and q_{dec} queries to the encryption and verification oracles, respectively.

Key Derivation Function A Key derivation function $\mathsf{KD} : \{0,1\}^\kappa \times \{0,1\}^\ell \to \{0,1\}^n$ takes a key $K \in \{0,1\}^\kappa$ and a string $x \in \{0,1\}^\ell$ as input and outputs another string $y \in \{0,1\}^n$. In this paper, we consider a key derivation function KD is secure if it satisfies the security requirement of a pseudo-random function. Namely, for any randomly selected key K, $\mathsf{KD}(K;\cdot)$ should behave like a truly random function $\mathsf{RF}(\cdot)$. We define the advantage of an adversary D against a key derivation function as

$$\mathbf{Adv}_{\mathsf{KD}}(D) = |\Pr[D^{\mathsf{KD}(K,\cdot)}(\kappa,\ell,n) = 1] - \Pr[D^{\mathsf{RF}(\cdot)}(\kappa,\ell,n) = 1)]|$$

where t denotes the maximum running time of D and q_{kd} is the maximum number of oracle queries D is allowed to make. The associated advantage function, $\mathbf{Adv}_{\mathsf{KD}}(t, q_{\mathrm{kd}})$, is then defined as the maximum value of $\mathbf{Adv}_{\mathsf{KD}}(D)$ over all D with time-complexity at most t, and asking at most q_{kd} oracle queries.

Non-interactive Key Derivation Scheme A non-interactive key derivation scheme consists of two polynomial-time algorithms (Gen, NKD) 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);
- $\mathrm{NKD}(pk_A, sk_B)$, the non-interactive key derivation function, takes as input a public key pk_A and a private key sk_B, and outputs a symmetric key $K_{AB} \in \{0,1\}^n$.

There are two security requirements involved for a Non-interactive Key Derivation Scheme:

- completeness: $\mathsf{NKD}(sk_A, pk_B) = \mathsf{NKD}(sk_B, pk_A)$;
- key privacy: $\{pk_A, pk_B, \mathsf{NKD}(sk_A, pk_B)\}$ is computationally indistinguishable from $\{pk_A, pk_B, \mathsf{U}_n\}$ where U_n denotes a string selected uniformly at random from $\{0,1\}^n$. We define the advantage of an adversary D against a non-interactive key derivation scheme as

$$\mathbf{Adv}_{\mathsf{NKD}}(D) = |\Pr[D(pk_A, pk_B, \mathsf{NKD}(sk_A, pk_B)) = 1] - \Pr[D(pk_A, pk_B, \mathsf{U}_n) = 1)]|.$$

The associated advantage function, $\mathbf{Adv}_{\mathsf{NKD}}(t)$, is then defined as the maximum value of $\mathbf{Adv}_{\mathsf{NKD}}(D)$ over all D with time-complexity at most t.

3.2 Two-Party Authenticated Key Exchange

A two-party authenticated key exchange (2AKE) protocol consists of two probabilistic polynomial time algorithms:

- $\textsc{Gen}(1^\kappa)$, the key generation algorithm, takes as input the security parameter 1^κ and outputs a long-term key K_U for entity U.
- $\mathsf{P}(K_A, K_B)$, a protocol execution algorithm that can be executed by two distinct parties holding K_A and K_B, respectively.

Depending on the type of the long-term key, one can divide 2AKE protocols into three major categories: (i) two-party password-based authenticated key exchange (2PAKE), (ii) two-party asymmetric-key authenticated key exchange (2AAKE), and (iii) two-party symmetric-key authenticated key exchange (2SAKE).

Many 2AKE protocols are extensions of the classic Diffie-Hellman key exchange [21] by incorporating an authentication component into the protocol. Here authentication can be achieved by using passwords (*i.e.*, 2PAKE), asymmetric-keys (*i.e.*, 2AAKE), or symmetric-keys (*i.e.*, 2SAKE). Take for example, the 2PAKE protocol presented in [6] uses a password-derived key to encrypt the Diffie-Hellman key exchange message flows so that only the two users sharing the same password can decrypt the messages and compute the Diffie-Hellman key.

In what follows, we present the security model for each type of 2AKE protocol. We first give an overview of the Real-Or-Random (ROR) model for 2PAKE [1]. We then define a security model in the Find-Then-Guess (FTG) sense for 2AAKE and 2SAKE.

3.2.1 Password-based

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 [6]. 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.* [1] 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 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 password-based protocols in this paper.

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 [6]. 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. A protocol *session* between a client instance and a server instance is labeled by a unique session identifier. As remarked by Bellare et al. [6], a session identifier can be constructed based on the partial protocol messages exchanged between the client and the server instances before the acceptance. Two instances U_1^i and U_2^i are said to be partners if the following conditions are met [6]:

(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;

(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 session identifier equal to that of U_1^i or U_2^i.

The oracle queries in the ROR security model for 2PAKE are then classified as follows [1]:

- 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 [1] 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.

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 [1]. 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

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

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

$$\mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_e, q_s, q_t) = \max_{\mathcal{A}}\{\mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(\mathcal{A})\}$$

where the maximum is over all $\mathcal{A}$ with time-complexity at most t and making at most q_e execute queries, q_s send queries, and q_t test queries.

We say that a 2PAKE protocol is secure in the ROR model if the advantage $\mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_e, q_s, q_t)$ is only negligibly larger than $cn/|\mathcal{D}|$, where c is a constant, n is the number of active sessions[6] and $|\mathcal{D}|$ is the size of the dictionary $\mathcal{D}$.

3.2.2 Asymmetric-key

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.

Generally speaking, an adversary in the Find-Then-Guess 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 [9]:

- 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): 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 participant U.
- TEST(U^i): Let b be a bit chosen uniformly at random at the beginning of an experiment defining indistinguishability in the FTG model. Denote V^j the partner (if it exists) of U^i. If no session key for instance U^i is defined, or if a REVEAL query was asked to either U^i or V^j, or if the adversary has made a CORRUPT(U) or CORRUPT(V) query, 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 $b = 0$.

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 the TEST query. 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, $\mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(\mathcal{A})$, and the associated advantage function $\mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(t, q_e, q_s, q_r, q_c)$ are then defined as in the password-based setting where q_r and q_c denote the maximum number of reveal and corrupt queries, respectively.

We say that a 2AAKE protocol is secure in the FTG model if the advantage $\mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(t, q_e, q_s, q_r, q_c)$ is negligible (in the associated security parameter).

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

3.2.3 Symmetric-key

The security definition of two-party symmetric-key based authenticated key exchange (2SAKE) is almost identical to that of 2AAKE. The only difference is that in a 2SAKE protocol, each protocol participant $A \in \mathcal{U}$ shares with another participant $B \in \mathcal{U} \backslash \{A\}$ a symmetric long-lived secret key K_{AB} $(= K_{BA})$.

4 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 [22, 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.[7]

From a security perspective, the complexity of a cross-domain authenticated key exchange often complicates its security analysis. As shown in [12, 2], 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 [17]. 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, 40], undetectable on-line password guessing and unknown key-share attacks [35], insider attacks (by malicious clients and servers) [18, 34, 35], and password-compromise impersonation attacks [16]. Recent proposals [43, 23] attempt to address these security problems.

Taking all the above observation into consideration, we propose a generic 4PAKE framework and give a set of example protocols that we believe are suitable for secure cross-domain communication between two users. In our framework, intuitively, 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 the key materials and agree on a session key. Clearly, we assume that the user trusts the remote server to only certify key

[7] Administrators must maintain separate keys for every domain which wishes to exchange authentication information with another domain, implying $n(n-1)$ keys for n domains, or they must utilise a hierarchical arrangement of domains, which may increase network traffic and complicate the trust model by requiring evaluation of transited domains [25].

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 two-party AKE protocol.

Inspired by the work of Abdalla *et al.* on a compositional approach to three-party password-based authenticated key exchange [1], we adopt a similar approach to our 4PAKE protocol, which comprises 2PAKE and 2A/SAKE as the building blocks. This is to simplify the security analysis of our protocol. By making use of secure 2PAKE and 2A/SAKE, 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 2A/SAKE protocols, implying that one can simply build a 4PAKE protocol based on existing deployed 2PAKE and 2A/SAKE protocols. In fact, each domain may deploy a different 2PAKE 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.

5 Generic Four-Party Key Exchange Protocols

In this section, we first present our generic framework for 4PAKE. Using our framework, we then give three concrete instantiations denoted by 4PAKEv1, 4PAKEv2, and 4PAKEv3. Succinctly, 4PAKEv1 is a generic protocol constructed from 2PAKE and 2AAKE; while 4PAKEv2 is a variant of 4PAKEv1 that considers key reuse. In 4PAKEv3, we consider the case where 2SAKE (instead of 2AAKE) is used in order to derive a more efficient protocol than 4PAKEv1.

5.1 Notation

Table 1: Notation.

$X \in \{A, B, S_A, S_B\}$	Entities involved in our 4PAKE protocol
id_X	Unique identifier of X
ts_X	Timestamp created by 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
sid	Session identifier

The notation used in our 4PAKE protocols 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, and which takes pwd_{X_1,X_2} as input and establishes ssk_{X_1,X_2}. We then use $ssk_{X_1,X_2} \leftarrow \mathsf{2AAKE}((esk_{X_1}, epk_{X_1}, \sigma_{X_1}), (esk_{X_2}, epk_{X_2}, \sigma_{X_2}))$ 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 pair (esk_X, epk_X) and a signature σ_X (ensuring the authenticity of the public key epk_X) of each entity, and establishes ssk_{X_1,X_2}. Moreover, we use $ssk_{X_1,X_2} \leftarrow \mathsf{2SAKE}(sk_{X_1,X_2})$ to denote the execution of a MAC-based 2SAKE protocol between X_1 and X_2. It takes as input a shared symmetric key sk_{X_1,X_2} and outputs a session key ssk_{X_1,X_2}. Also, we define a session identifier sid as $(id_A, epk_A, id_B, epk_B)$ in all the 4PAKE protocols shown in this paper. (A similar definition of sid is used in [8].)

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. We assume that the execution of any of our 4PAKE protocols involve 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.

5.2 Generic Framework

We begin by describing a generic framework for constructing 4PAKE protocols. Our framework consists of the following two phases:

1. Setup. In the initial phase, each server S generates its long-term public-private key pair (pk_S, sk_S), and each client C chooses a password pwd_C that is shared with its domain server.
2. 4PAKE Execution. The second phase is divided into the following steps, as illustrated in Figure 1:
   - Initialisation ($M1, M2$): Should two clients A and B wish to securely communicate with each other, they first exchange their identity information, id_A, id_B, and their ephemeral public keys, epk_A, epk_B. If A and B already have knowledge of each other's identity information prior to the start of the protocol (*e.g.*, they may obtain such information from a higher application layer), then this step can be skipped.
   - Local authentication and key exchange ($M3$): A and B then each runs a 2PAKE protocol with their own domain (authentication) server to establish a temporary session key for their following communication in the next step.
   - Token acquirement ($M4, M5$): Subsequently, A and B each obtains an authentication token issued by their respective domain servers via the temporary session key established in the previous step.
   - Two-party authentication and key exchange ($M6$): Lastly, A and B use their respective authentication tokens and ephemeral public/private keys to execute the 2A/S/PAKE protocol in order to establish a session key.

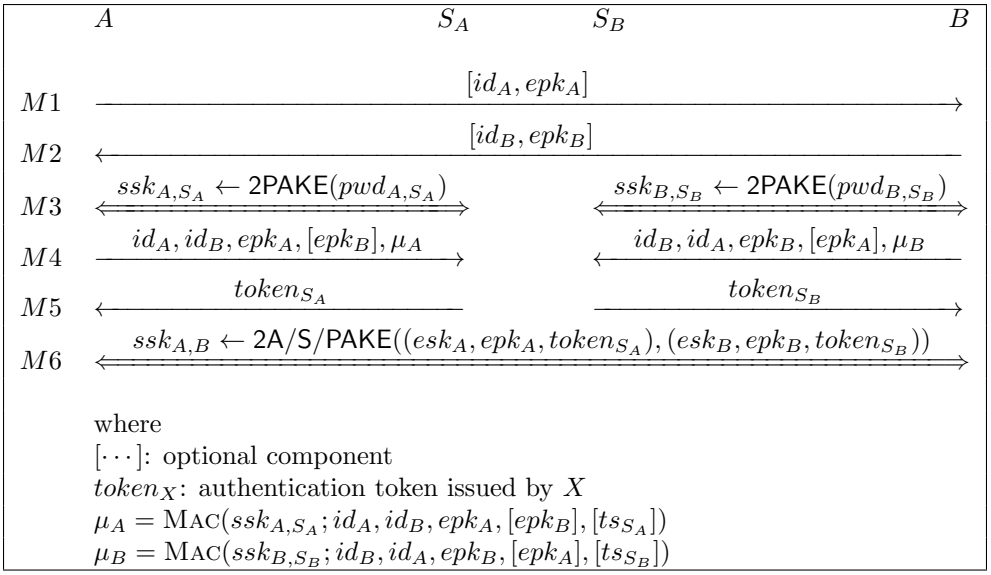

Figure 1: A generic framework for 4PAKE.

REMARK 1. In scenarios where clients A and B may already have knowledge of each other's identity information prior to the start of a 4PAKE protocol, we can remove the first two message flows ($M1, M2$). Hence the signature σ created by each domain server does not include the remote domain client's ephemeral public key (but its local domain client's). We note that, however, the ephemeral public key epk chosen by a client can be seen as a challenge to the other communicating client. Thus, the removal of $M1$ & $M2$ opens up the possibility of a replay attack. For example, an adversary who somehow managed to learn A's ephemeral secret key esk_A can reuse σ_{S_A} to establish a session key with B; or even if A has been revoked by S_A after obtaining σ_{S_A}, A can still reuse the signature to continue to establish secure channels with B without B knowing it. Hence, it is essential to include a timestamp ts in the signature, such that σ_{S_A} is now being tied to a specific time, allowing B to check the freshness of the signature.

REMARK 2. We assume that each domain server has access to the authenticated credentials (*e.g.*, public key certificates) of other domain servers. Moreover, each client possesses a copy of its server's credential in order to verify the authentication token issued by the server in $M5$. This is not necessarily required to be done in advance. In practice, the servers can distribute their credentials 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 their respective servers' credentials 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 credential.

REMARK 3. 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. (See Section 8 for further details of our performance evaluation.)

In what follows, we provide example protocols derived from the above framework. Through these instantiations, we demonstrate that a 4PAKE protocol can be constructed from various different basic two-party key exchange protocols and cryptographic primitives in a compositional way.

5.3 4PAKEv1 (from 2PAKE and 2AAKE)

Our first instantiation of 4PAKE protocol, denoted by 4PAKEv1, entails "piggybacking" 2PAKE (password-based) and 2AAKE (asymmetric-key based) protocols. This is illustrated in Figure 2.

In $M1$ & $M2$, as described before, clients A and B exchange information about their identities (id_A, id_B),[8] and ephemeral public keys (epk_A, epk_B), such as Diffie-Hellman components. 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 request for an authentication token by submitting 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 by generating and returning a token in the form of a signature over the information received 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.

One may instantiate a 2PAKE protocol using Jablon's SPEKE protocol [26], or that proposed by Bellare *et al.* [6], or Boyko *et al.* [11], for example. On the other hand, a simple, classic instantiation of 2AAKE is the typical two-pass Diffie-Hellman key exchange protocol, involving exchanges of σ_{S_A} and σ_{S_B} between clients A and B. The output session key is then $ssk_{A,B} = \mathsf{KD}(g^{esk_A esk_B}, sid)$, for example. Indeed, we can

[8] 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.

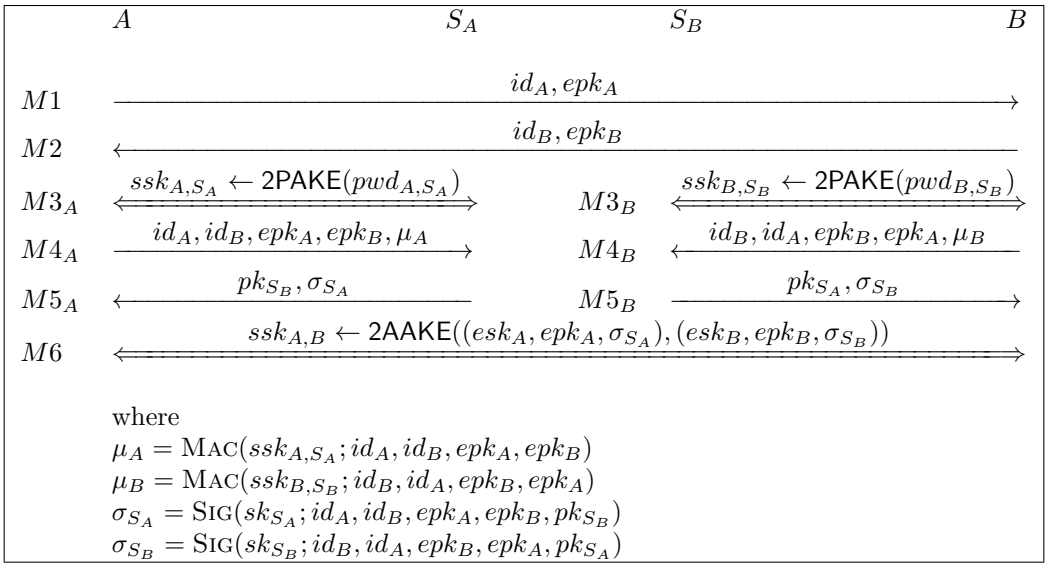

Figure 2: The 4PAKEv1 protocol.

use any signature-based message transmission (MT) authenticator proposed by Bellare *et al.* [3, 15] in $M6$. (See [37, 28] for other concrete examples of signature-based Diffie-Hellman key exchange.) As explained in our motivations, 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. 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 [3, 15], to reduce computational overhead. (We give a example protocol that makes use of MAC-based key exchange in Section 5.5.)

5.4 4PAKEv2 (Key Reuse)

In our second example protocol, denoted by 4PAKEv2, we consider a scenario where key reuse is tolerated. Our goal is to simplify 4PAKEv1 such that some of the steps can be skipped should a client wishes to reconnect to the same remote client within a predefined period of time. To achieve this, we now require the client to generate two ephemeral key pairs: one can be reused over multiple sessions, while the other can be used just for a session and regenerated for each new session.

Let (epk^0, esk^0) and (epk^1, esk^1) be the two key pairs generated by the client, and let sk_{CA} be the secret key of a CA. We also let $[sk \succ pk]$ denote a certificate signed using secret key sk over public key pk; and $[sk \succ pk] \rightarrow [sk' \succ pk']$ denote a certificate chain rooted at sk. Our protocol is then illustrated in Figure 3.

Here, we treat σ_{S_A} as a public key certificate with respect to epk_A^0, which is issued by the domain server S_A and has a validity period of more than just a session, for example, a day, week, or month. The client then creates another new pair of ephemeral keys (esk_A^1, epk_A^1) that are taken as input for the signature-based 2AAKE protocol. Particularly, a certificate-chain of the form:

$$[sk_{CA} \succ pk_{S_A}] \rightarrow [sk_{S_A} \succ epk_A^0] \rightarrow [esk_A^0 \succ epk_A^1]$$

is created in this setting. This way, the authenticity of ephemeral public key epk_A^1 is assured by a signature under esk_A^0, which can be verified by B using pk_{S_A} and epk_A^0, during the execution of 2AAKE.

Should A want to establish a session key with B again within the validity period (*i.e.*, ts_{S_A}) of epk_A^0, A simply generates a new (epk_A^1, esk_A^1) key pair and runs the 2AAKE protocol with B directly without going through $M1$ to $M5$. Henceforth, we use 4PAKEv2* to denote a 4PAKEv2 protocol that executes only the last message flow $M6$. We show, in our performance evaluation in Section 8, that 4PAKEv2* is roughly 2× faster than PKCROSS and our other 4PAKE protocols.

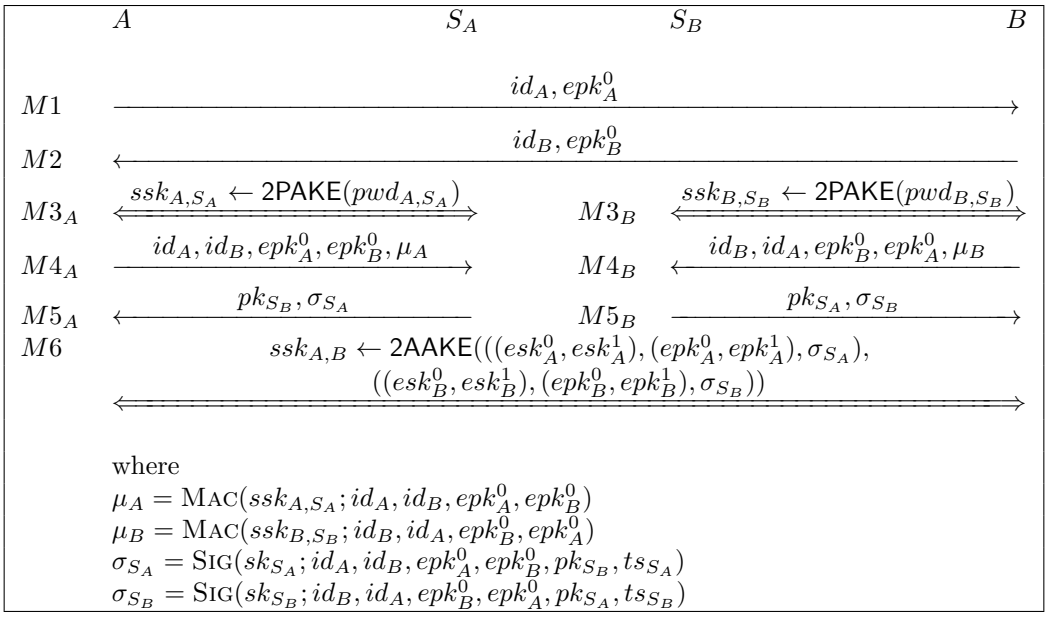

Figure 3: The 4PAKEv2 protocol.

Notice that in fact, more generally, we can use the domain information of B instead of id_B in μ_A and σ_{S_A}, such that A can reuse the (epk_A^0, esk_A^0) key pair and σ_{S_A} to establish a secure communicate session with any client from the same domain as B.

5.5 4PAKEv3 (from 2PAKE and 2SAKE)

We now give an example that relies on 2PAKE and 2SAKE to improve the computational efficiency of 4PAKEv1.

Intuitively, we assume that each local domain server is able to derive a shared symmetric key with a remote server via a non-interactive key derivation scheme. This can be achieved if, for example, the server's long-term public key is of the form of a Diffie-Hellman component g^{sk}, where sk is the corresponding secret key. This is so since we can simply compute a common secret key, which is a Diffie-Hellman key, based on the local server's secret key and the remote server's public key. In the case of S_A and S_B, the Diffie-Hellman key will be of the form $g^{sk_{S_A} sk_{S_B}}$. The secret key shared between the two servers will then be used to derive a pre-session key for the clients, such that the latter can use the pre-session key to further exchange and establish a session key using a 2SAKE. Our protocol is illustrated in Figure 4.

The domain servers S_A and S_B first use a non-interactive key derivation scheme to generate a long-term shared key lsk_{S_A,S_B}. This key is then used to generate a short-term pre-session key ssk_{S_A,S_B} based on the session specific data $(id_A, epk_A, id_B, epk_B)$.

After the establishment of a local session key ssk_{A,S_A} in $M3$, the client A and the domain server S_A use a key derivation function to generate two sub-keys ssk_{A,S_A}^0 and ssk_{A,S_A}^1. The former sub-key is used as a MAC key in $M4$ to let A deliver the session specific data to S_A, while the latter is used as an encryption key in $M5$ to let S_A securely deliver the pre-session key ssk_{S_A,S_B} to A. The same procedures are also performed between B and S_B. This way, A and B will both obtain the pre-session key ssk_{S_A,S_B} at the end of step $M5$. Finally, in $M6$, A and B use the shared pre-session key to execute a MAC-based 2SAKE protocol. A good example of a MAC-based 2SAKE is the REKEY protocol of [15].

Alternative. Here we briefly sketch another example of 4PAKE that aims to optimise the implementation cost of the above 4PAKEv3 protocol. The core idea is that we now construct a 4PAKE protocol from 2PAKE

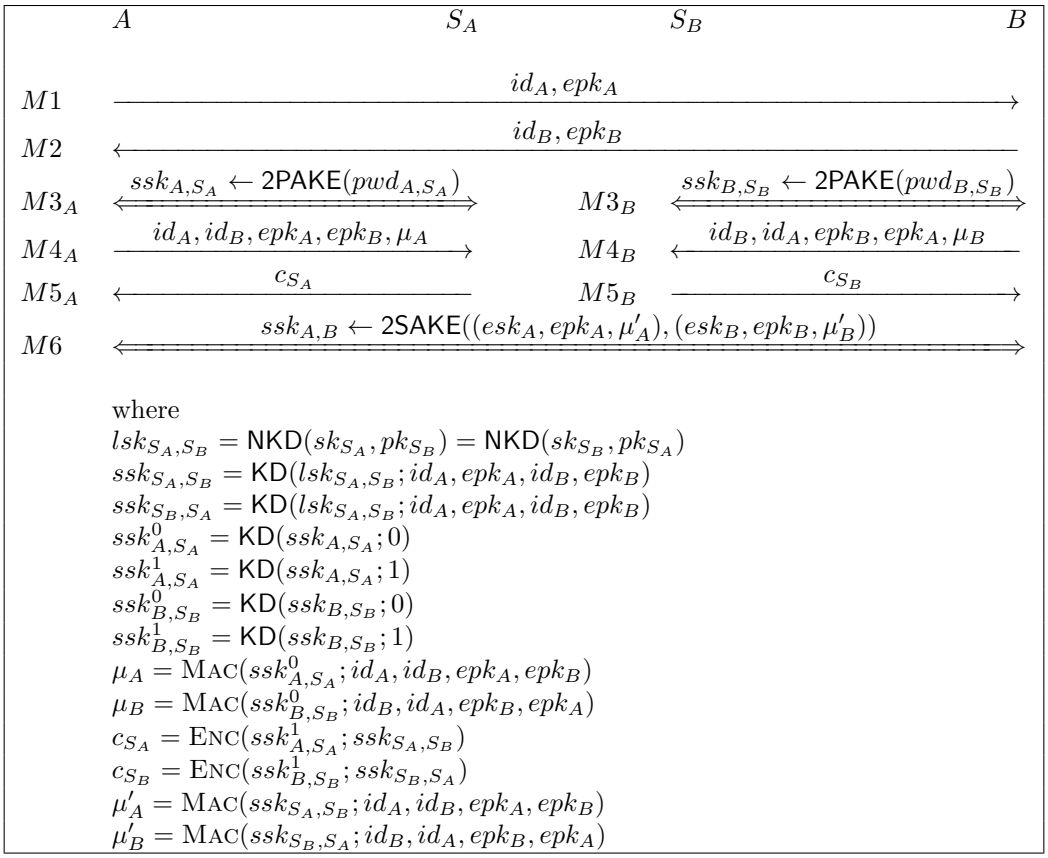

Figure 4: The 4PAKEv3 protocol.

as the only building block. That is, 2PAKE is used not only between the client and the domain server, but also between the two clients. Our motivation for doing this is that implementing a single protocol is much cheaper than implementing two separate protocols (2PAKE and 2SAKE/2AAKE) in terms of memory space or circuit size, particularly if the protocol runs within a constrained hardware device, for example, smart card and mobile phone. Such a protocol is very similar to 4PAKEv3. The only difference between the two protocols is that we replace $M6$ in Figure 4 with $ssk_{A,B} \leftarrow \mathsf{2PAKE}(ssk_{S_A,S_B})$, where ssk_{S_A,S_B} is regarded as the "password" shared between clients A and B.

6 Security Model

We now define an ROR security model for 4PAKE by extending the work of Abdalla *et al.* for the three-party case [1]. In the 4PAKE setting, we assume that each protocol participant is a client $U \in \mathcal{U}$ or a trusted server $S \in \mathcal{S}$.[9] 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.

[9]Note that in Section 3.2, 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).

6.1 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 [1].

The notion of partnering (between two clients) in the four-party setting is similar to that for the two-party setting (between client and server), 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, $U_1^{i_1}$ and $U_2^{i_2}$, and trusted server instances, $S_1^{j_1}$ and $S_2^{j_2}$. 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.

- 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 the client instance U^i, then return the undefined symbol $\perp$.

The advantage of an adversary $\mathcal{A}$ in violating the indistinguishability of the 4PAKE protocol in the ROR sense, $\mathbf{Adv}^{\text{ror}}_{\text{4PAKE},\mathcal{D}}(\mathcal{A})$, and the associated advantage function $\mathbf{Adv}^{\text{ror}}_{\text{4PAKE},\mathcal{D}}(t, q_e, q_s, q_t)$ are then defined as in the two-party setting.

Note that for simplicity of presentation, we will not consider the notion of *perfect forward secrecy* [6] in this paper. Defining such a notion is a straightforward exercise.

6.2 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 [1] 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 [1]:

- 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, $\mathbf{Adv}^{\text{kp}}_{\mathsf{4PAKE}}(\mathcal{A})$, and the associated advantage function $\mathbf{Adv}^{\text{kp}}_{\mathsf{4PAKE}}(t, R)$ are defined as before.

7 Security Analysis

Before presenting our security proofs, we first give a high level description of the proof idea with regards to the generic framework illustrated in Figure 1.

We adopt a game-hopping approach which starts with the original game, or **Game** $\mathbf{G}_0$, defined in our 4PAKE model (as described in the last section), and ends with **Game** $\mathbf{G}_5$. These games proceed in the following manner:

- In **Game** $\mathbf{G}_1$, we replace the key ssk_{A,S_A} with a random key. The difference between $\mathbf{G}_1$ and $\mathbf{G}_0$ will be negligible to the adversary if the 2PAKE protocol is secure.
- **Game** $\mathbf{G}_2$ proceeds in a similar way as **Game** $\mathbf{G}_1$, except that we make the same modification to ssk_{B,S_B}, that is, replacing it with a random key.
- In **Game** $\mathbf{G}_3$, we further change **Game** $\mathbf{G}_2$ by rejecting all the MACs forged by the adversary in $M4$. Due to the unforgeability of the MAC, the difference between $\mathbf{G}_3$ and $\mathbf{G}_2$ is also negligible.
- In **Game** $\mathbf{G}_4$, we do a similar modification by rejecting all the authentication tokens forged by the adversary in $M5$. The difference between $\mathbf{G}_4$ and $\mathbf{G}_3$ is again negligible if the token is unforgeable.
- In **Game** $\mathbf{G}_5$, we replace the key $ssk_{A,B}$ with a random key. The adversary will not notice such a change if the 2A/S/PAKE protocol is secure.
- Finally, in **Game** $\mathbf{G}_5$, it is obvious that the adversary has no advantage since a random session key will be returned in each TEST query regardless of the value of b.

7.1 4PAKEv1

Here we give a security proof for 4PAKEv1 from which we can also relate and derive the security proofs for 4PAKEv2 and 4PAKEv3.

7.1.1 Indistinguishability of Session Keys

As the following theorem states, our 4PAKEv1 protocol shown in Figure 2 is secure in the ROR model (as defined in Section 6), provided that the underlying primitives it uses are secure.

Theorem 1. *Let* 4PAKEv1 *be the four-party password-based authenticated key exchange protocol. Let* q_{exe} *be the number of queries to the* EXECUTE *oracle of the* 4PAKEv1 *protocol and* q_{test} *be the number of* TEST *queries. Let also* q^{Mx}_{send} *denote the number of* SENDCLIENT *or* SENDSERVER *queries related to message* Mx *of the* 4PAKEv1 *protocol for* $x \in \{1, 2, 3A, 3B, 4, 5, 6\}$. *Then*

$$
\begin{aligned}
&\mathbf{Adv}^{\text{ror}}_{\mathsf{4PAKEv1}}(t, q_{\text{exe}}, q^{Mx}_{\text{send}}, q_{\text{test}}) \leq \\
&\quad 2 \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{\text{exe}}, q^{M3_A}_{\text{send}}, q_{\text{exe}} + q^{M3_A}_{\text{send}}) \\
&\quad + 2 \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{\text{exe}}, q^{M3_B}_{\text{send}}, q_{\text{exe}} + q^{M3_B}_{\text{send}}) \\
&\quad + 2 \cdot q^{M4}_{\text{send}} \cdot \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(t, 2, 0) \\
&\quad + 4 \cdot \mathbf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(t, q^{M5}_{\text{send}}, 0) \\
&\quad + 2 \cdot q_{\text{test}} \cdot \mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(t, q_{\text{exe}}, q^{M1,M2,M6}_{\text{send}}, q_{\text{test}}, 0)
\end{aligned}
$$

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 4PAKEv1 in the ROR sense. Our security proof is a sequence of security games simulated using techniques from Abdalla *et al.* [1].

As described earlier, we start with the real security game against the 4PAKEv1 protocol, and end with a game with the adversary's advantage is zero, 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 SUCC_n to be the event in which the adversary correctly guesses the hidden bit b used in the TEST queries (as defined in Section 6).

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

$$\mathbf{Adv}^{\text{ror}}_{\text{4PAKEv1}}(\mathcal{A}) = 2 \cdot \Pr[\text{SUCC}_0] - 1$$

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

Proof of Lemma 2. In order to prove this lemma, we simulate an adversary $\mathcal{A}_{\text{2PAKE}}$ against the indistinguishability of the 2PAKE protocol using a distinguisher, $\mathcal{A}_1$, between games G_0 and G_1. Adversary $\mathcal{A}_{\text{2PAKE}}$ 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 starts answering oracles queries from $\mathcal{A}_1$ as follows:

- SENDCLIENT queries: If $\mathcal{A}_1$ makes a query on an instance of the 2PAKE protocol run between A and S_A, then $\mathcal{A}_{\text{2PAKE}}$ responds by sending the corresponding query to its 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 TEST query to that instance, unless such a query had already been made to its partner. The output of the 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 SENDCLIENT query targeting an instance of the 2PAKE protocol run between B and S_B, $\mathcal{A}_{\text{2PAKE}}$ responds using the password of client B that it has chosen at the beginning of the simulation.

  All remaining 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.

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

- EXECUTE queries: $\mathcal{A}_{\text{2PAKE}}$ can easily answer these queries using its own EXECUTE oracle and the output of the relevant TEST queries, just as how SENDCLIENT and SENDSERVER queries are responded.

- TEST queries: $\mathcal{A}_{\text{2PAKE}}$ 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 $\mathcal{A}_{\text{2PAKE}}$ outputs 1. Otherwise, it outputs 0.

Let b^* denote the random bit in the 2PAKE experiment for $\mathcal{A}_{\mathsf{2PAKE}}$. Then we have

$$\begin{aligned}
& \mathbf{Adv}^{\mathrm{ror}}_{\mathsf{2PAKE},\mathcal{D}}(\mathcal{A}_{\mathsf{2PAKE}}) \\
= \;& 2(\Pr[b'=b|b^*=1]\Pr[b^*=1] + \Pr[b'\neq b|b^*=0]\Pr[b^*=0]) - 1 \\
= \;& \Pr[b'=b|b^*=1] + \Pr[b'\neq b|b^*=0] - 1 \\
= \;& \Pr[b'=b|b^*=1] - \Pr[b'=b|b^*=0] \\
= \;& \Pr[\textsc{Succ}_0] - \Pr[\textsc{Succ}_1]
\end{aligned}$$

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 EXECUTE oracle, at most $q^{M3_A}_{\mathrm{send}}$ queries to its SEND oracle, and at most $q_{\mathrm{exe}} + q^{M3_A}_{\mathrm{send}}$ queries to its TEST oracle. □

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

Game $\mathbf{G}_3$: 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 4PAKEv1 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. $|\Pr[\textsc{Succ}_3] - \Pr[\textsc{Succ}_2]| \leq q^{M4}_{\mathrm{send}} \cdot \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{MAC}}(t, 2, 0)$.

Proof of Lemma 4. We use a hybrid argument to proof this lemma. We define a sequence of hybrid experiments V_i, where $0 \leq i \leq q^{M4}_{\mathrm{send}}$. (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 $\mathcal{A}^i_{\mathrm{mac}}$ for breaking the security of the MAC scheme.

The description of the adversary $\mathcal{A}^i_{\mathrm{mac}}$ is as follows. For the first $i-1$ sessions, the adversary $\mathcal{A}^i_{\mathrm{mac}}$ 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, $\mathcal{A}^i_{\mathrm{mac}}$ 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 $\mathcal{A}^i_{\mathrm{mac}}$, then $\mathcal{A}^i_{\mathrm{mac}}$ 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, $\mathcal{A}^i_{\mathrm{mac}}$ 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 $\mathcal{A}^i_{\mathrm{mac}}$ can forge a new message-tag pair under a chosen-message attack. Since $\mathcal{A}^i_{\mathrm{mac}}$ has time-complexity t and makes

at most two queries to its MAC tag generation oracle (to answer the SendClient queries from $\mathcal{A}_1^i$ 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

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

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

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

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 4PAKEv1 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. The following Lemma shows the difference between G_3 and G_4 is negligible.

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

Proof of Lemma 5. Let $\mathcal{A}_{\text{sig}}$ denote an adversary against the digital signature scheme. $\mathcal{A}_{\text{sig}}$ receives a public key pk of the digital signature scheme and simulates the game as follows.

$\mathcal{A}_{\text{sig}}$ chooses a random client $C \in \{A, B\}$ and guesses that a forge event would happen on C. $\mathcal{A}_{\text{sig}}$ assigns pk as the public key of the server S_C, and generates the public/private key pairs for all the other servers and the passwords for all the clients honestly. $\mathcal{A}_{\text{sig}}$ then simulates the game G_3 for the adversary $\mathcal{A}_{\mathsf{2PAKE}}$. When a signature of S_C is required to respond a SendServer query, $\mathcal{A}_{\text{sig}}$ makes a query to its signing oracle to obtain a valid signature and uses it to answer the SendServer query. Let F_2 denote the event that the adversary makes a SendClient query containing a valid signature with respect to S_C and which is not previously returned by $\mathcal{A}_{\text{sig}}$. We then have

$$|\Pr[\text{Succ}_4] - \Pr[\text{Succ}_3]| \leq \Pr[F_2] \leq 2 \cdot \mathbf{Adv}^{\text{euf-cma}}_{\mathsf{Sig}}(t, q^{M5}_{\text{send}}, 0).$$

□

Game G_5: This game is identical to the previous game, except that we replace the session key $ssk_{A,B}$ (output by the 4PAKEv1 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. $|\Pr[\text{Succ}_5] - \Pr[\text{Succ}_4]| \leq q_{\text{test}} \cdot \mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(t, q_{\text{exe}}, q^{M1,M2,M6}_{\text{send}}, q_{\text{test}}, 0)$.

Proof of Lemma 6. Again, we prove the lemma by a hybrid argument. Let V_i $(0 \leq i \leq q_{\text{test}})$ denote a variant of the game G_4 such that for the first i Test queries, the real session keys $ssk_{A,B}$ is returned to the adversary, while for the remaining Test queries, random session keys are returned. Then we have $V_0 = \mathrm{G}_5$ and $V_{q_{\text{test}}} = \mathrm{G}_4$. If there exists an adversary $\mathcal{A}_5$ that can distinguish G_4 and G_5 with advantage ϵ, then there must exists an index i such that $\mathcal{A}_5$ can distinguish V_i and V_{i+1} with advantage at least ϵ/q_{test}.

Given such a distinguisher $\mathcal{A}_5$, we can construct an adversary $\mathcal{A}_{\mathsf{2AAKE}}$ against the indistinguishability of the signature-based 2AAKE protocol. $\mathcal{A}_{\mathsf{2AAKE}}$ first selects a password for each client and uses the public keys pk_{S_A} and pk_{S_B} in the 2AAKE game as the public keys for S_A and S_B in the simulated game for $\mathcal{A}_5$. $\mathcal{A}_{\mathsf{2AAKE}}$ then generates asymmetric-key pairs for the other servers in the system. Next, $\mathcal{A}_{\mathsf{2AAKE}}$ responds to queries from $\mathcal{A}_5$ as follows:

- SendClient and SendServer queries: $\mathcal{A}_{\mathsf{2AAKE}}$ answers the SendClient and SendServer queries from $\mathcal{A}_5$ by using pwd_A and pwd_B, together with queries to its own Send oracle. Note that since no forgery event with respect to $\mu_A, \mu_B, \sigma_{S_A}, \sigma_{S_B}$ would occur in game G_4, $\mathcal{A}_{\mathsf{2AAKE}}$ can successfully embed the answers from its own Send oracle into the simulated game for $\mathcal{A}_5$.

– EXECUTE queries: $\mathcal{A}_{\mathsf{2AAKE}}$ can easily answer these queries by using its own EXECUTE oracle and pwd_A, pwd_B.

– TEST queries: For the first i TEST queries made by $\mathcal{A}_5$, $\mathcal{A}_{\mathsf{2AAKE}}$ uses its REVEAL oracle to obtain the real session keys and use them to answer the TEST queries. For the $i+1$-th TEST query made by $\mathcal{A}_5$, $\mathcal{A}_{\mathsf{2AAKE}}$ uses the key obtained from its own TEST oracle to answer the query. For the rest of the TEST queries made by $\mathcal{A}_5$, $\mathcal{A}_{\mathsf{2AAKE}}$ responds with random keys.

It is obvious that if $\mathcal{A}_5$ can distinguish V_i from V_{i+1}, then $\mathcal{A}_{\mathsf{2PAKE}}$ can successfully guess the value of b in the 2AAKE game.

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_{test} queries to its REVEAL oracle, and at most 1 query to its TEST oracle. □

Clearly, $\Pr[\text{SUCC}_5] = \frac{1}{2}$. All the above lemmas yield the result in Theorem 1. □

7.1.2 Key Privacy against Servers

As explained in Section 6, we assume that the servers are honest-but-curious. Hence, we should also show that if the 4PAKEv1 protocol is executed as expected and does not abort, the servers should not gain any knowledge about the resulting session key.

Theorem 7. *Let* 4PAKEv1 *be the four-party password-based authenticated key exchange protocol. Then*

$$\mathbf{Adv}^{\text{kp}}_{\mathsf{4PAKEv1}}(t, q_{\text{exe}}, q_{\text{send}}^{Mx}, q_{\text{test}}) \leq 2 \cdot q_{\text{test}} \cdot \mathbf{Adv}^{\text{ftg}}_{\mathsf{2AAKE}}(t, q_{\text{exe}}, q_{\text{send}}^{M1,M2,M6}, q_{\text{test}}, 0)$$

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. □

7.2 4PAKEv2

The security proof for the 4PAKEv1 protocol can be easily extended to prove the security of 4PAKEv2 by following the same sequence of games outlined at the beginning of this section. The only difference is that we now adopt a signature-based authenticator, which relies on time-stamps [39]. We obtain the same security bounds for 4PAKEv2 as with those for 4PAKEv1.

7.3 4PAKEv3

We now provide a sketch of the security proof for our 4PAKEv3 protocol, which is based on 2PAKE and 2SAKE, by using the same techniques (particularly game-hopping and hybrid argument) we have used in proving the security for 4PAKEv1.

Theorem 8. *Let* 4PAKEv3 *be the four-party password-based authenticated key exchange protocol based on 2SAKE. Let* q_{exe} *be the number of queries to the* EXECUTE *oracle of the* 4PAKEv3 *protocol and* q_{test} *be the*

number of TEST *queries. Let also* q_{send}^{Mx} *denote the number of* SENDCLIENT *or* SENDSERVER *queries related to message* Mx *of the* 4PAKEv3 *protocol for* $x \in \{1_A, 1_B, 2, 3, 4\}$. *Then*

$$\begin{aligned}
&\mathbf{Adv}^{\text{ror}}_{\mathsf{4PAKEv3}}(t, q_{\text{exe}}, q_{\text{send}}^{Mx}, q_{\text{test}}) \leq \\
&\quad 2 \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{\text{exe}}, q_{\text{send}}^{M3_A}, q_{\text{exe}} + q_{\text{send}}^{M3_A}) \\
&\quad + 2 \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},\mathcal{D}}(t, q_{\text{exe}}, q_{\text{send}}^{M3_B}, q_{\text{exe}} + q_{\text{send}}^{M3_B}) \\
&\quad + 2 \cdot (q_{\text{send}}^{M3} \cdot \mathbf{Adv}_{\mathsf{KD}}(t, 2) + q_{\text{send}}^{M4} \cdot \mathbf{Adv}_{\mathsf{KD}}(t, 1)) \\
&\quad + 2 \cdot \mathbf{Adv}_{\mathsf{NKD}}(t) \\
&\quad + 2 \cdot q_{\text{send}}^{M5} \cdot (\mathbf{Adv}^{\text{suf-cma}}_{\mathsf{AE}}(t, 1, 0) + \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(t, 1)) \\
&\quad + 2 \cdot q_{\text{test}} \cdot \mathbf{Adv}^{\text{ftg}}_{\mathsf{2SAKE}}(t, 1, q_{\text{send}}^{M1,M2,M6}, 0, 0)
\end{aligned}$$

assuming the 2PAKE *and* 2SAKE *protocols, and the* KD *and* NKD *functions, and the authenticated encryption scheme used in the protocol are secure.*

Proof of Theorem 8 (Sketch). The adversarial games $\mathbf{G}_0, \mathbf{G}_1, \mathbf{G}_2$ are the same as in the proof of Theorem 1. However, we define two additional games $\mathbf{G}_{2.1}$ and $\mathbf{G}_{2.2}$ between $\mathbf{G}_2$ and $\mathbf{G}_3$.
Game $\mathbf{G}_{2.1}$: In this game, we replace the keys ssk^0_{A,S_A} and ssk^1_{A,S_A} with two independent and random keys. From the assumption that the key derivation function KD is secure and by a hybrid argument, we obtain

$$|\Pr[\text{SUCC}_{2.1}] - \Pr[\text{SUCC}_2]| \leq q_{\text{send}}^{M3_A} \cdot \mathbf{Adv}_{\mathsf{KD}}(t, 2).$$

Game $\mathbf{G}_{2.2}$: Similarly, in this game, we replace ssk^0_{B,S_B} and ssk^1_{B,S_B} with two independent and random keys and we have

$$|\Pr[\text{SUCC}_{2.2}] - \Pr[\text{SUCC}_{2.1}]| \leq q_{\text{send}}^{M3_B} \cdot \mathbf{Adv}_{\mathsf{KD}}(t, 2).$$

Game $\mathbf{G}_3$ is the same as in the proof of Theorem 1. We then define two additional games $\mathbf{G}_{3.1}$ and $\mathbf{G}_{3.2}$ between Game $\mathbf{G}_3$ and $\mathbf{G}_4$.
Game $\mathbf{G}_{3.1}$: In this game, we replace the values of $\mathsf{NKD}(sk_{S_A}, pk_{S_B})$ and $\mathsf{NKD}(sk_{S_B}, pk_{S_A})$ with a random key. Then from the security of the non-interactive key derivation scheme, we have

$$|\Pr[\text{SUCC}_{3.1}] - \Pr[\text{SUCC}_3]| \leq \mathbf{Adv}_{\mathsf{NKD}}(t).$$

Game $\mathbf{G}_{3.2}$: Here, we further replace the values of ssk_{S_A,S_B} and ssk_{S_B,S_A} in each session with a random key. Once again, since KD is a secure key derivation function, the difference between game $\mathbf{G}_{3.1}$ and game $\mathbf{G}_{3.2}$ can be bounded by

$$|\Pr[\text{SUCC}_{3.2}] - \Pr[\text{SUCC}_{3.1}]| \leq q_{\text{send}}^{M4} \cdot \mathbf{Adv}_{\mathsf{KD}}(t, 1).$$

In game $\mathbf{G}_4$, we modify the way the oracle instances respond to SENDCLIENT queries on $M5$: if the adversary makes a SENDCLIENT query containing a new ciphertext not previously generated by an oracle, then we consider the ciphertext invalid and force the instance in question to terminate without accepting. By following a proof similar to that of Lemma 5, we have

$$|\Pr[\text{SUCC}_4] - \Pr[\text{SUCC}_3]| \leq q_{\text{send}}^{M5} \cdot \mathbf{Adv}^{\text{euf-cma}}_{\mathsf{AE}}(t, 1, 0).$$

We then define an additional game $\mathbf{G}_{4.1}$ between game $\mathbf{G}_4$ and game $\mathbf{G}_5$.
Game $\mathbf{G}_{4.1}$: In this game, we replace the key $ssk_{S_A,S_B} (= ssk_{S_B,S_A})$ used in the 2SAKE protocol with a random key rsk that is independent of c_{S_A} and c_{S_B}. Since AE is IND-CPA secure, by a hybrid argument, we have

$$|\Pr[\text{SUCC}_{4.1}] - \Pr[\text{SUCC}_4]| \leq q_{\text{send}}^{M5} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathsf{AE}}(t, 1).$$

Finally, in game $\mathbf{G}_5$, we replace the session keys $ssk_{A,B}$ in all the sessions with random keys. By following a hybrid argument similar to that of Lemma 6, we have

$$|\Pr[\text{SUCC}_5] - \Pr[\text{SUCC}_{4.1}]| \leq q_{\text{test}} \cdot \mathbf{Adv}^{\text{ftg}}_{\mathsf{2SAKE}}(t, 1, q_{\text{send}}^{M1,M2,M6}, 0, 0).$$

Notice that the number of queries in the bound is different from that in Lemma 6, this is because in 4PAKEv3 the 2SAKE protocol will use a different pre-session key ssk_{S_A,S_B} in each session, while in 4PAKEv1 the 2AAKE protocol will use the same public keys pk_A and pk_B in all the sessions.

It is obvious that $\Pr[\text{SUCC}_5] = \frac{1}{2}$. Combining all together yields the result in Theorem 8. □

We can also use an almost identical approach to prove the security of a variant of our 4PAKEv3 protocol that makes use of only 2PAKE (as described at the end of Section 5.5) to obtain similar bounds as above except the following:

$$|\Pr[\text{SUCC}_5] - \Pr[\text{SUCC}_{4.1}]| \leq q_{\text{test}} \cdot \mathbf{Adv}^{\text{ror}}_{\mathsf{2PAKE},2^{|ssk_{S_A,S_B}|}}(t, 1, q^{M6}_{\text{send}}, 1)$$

where $|ssk_{S_A,S_B}|$ denotes the length of ssk_{S_A,S_B}.

8 Efficiency Analysis

In this section, we evaluate and compare the communication and computational costs of our 4PAKE protocols against PKCROSS [25]. For this purpose, we instantiate concrete 2PAKE, 2AAKE, and 2SAKE based on SPEKE [26][10], the signature-based authenticator of [15], and the MAC-based authenticator of [1], respectively.

Notation. We use PENC/PDEC, SIG/VER, and GEXP to denote public key encryption/decryption, public key signing/verifying, and group exponentiation, respectively. As before, we let ENC denote symmetric-key (authenticated) encryption, and MAC denote MAC tag generation. We then use $|\mathcal{X}|$ to denote the length of the output of an algorithm or a function $\mathcal{X}$, for example, $|\text{ENC}|$ denotes the output size of the ENC algorithm. Moreover, we let $\mathbf{C} \leftrightarrow \mathbf{C}$ denote the interaction between two clients in 4PAKE (or a client and a service/application server in PKCROSS); let $\mathbf{S} \leftrightarrow \mathbf{S}$ denote the interaction between two servers in 4PAKE (or two KDCs in PKCROSS); and let $\mathbf{C} \leftrightarrow \mathbf{S}$ represent the interaction between a client and a server (or KDC).

8.1 Communication Overhead

Our generic protocols may understandably be 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 2 shows that our generic protocols are still better than PKCROSS in terms of the total numbers of message flows exchanged between $\mathbf{C} \leftrightarrow \mathbf{S}$, $\mathbf{C} \leftrightarrow \mathbf{C}$, and $\mathbf{S} \leftrightarrow \mathbf{S}$. This is not suprising given that all of our protocols do not require interaction between the local and the remote servers; while Kerberos is designed such that a client must obtain a ticket-granting ticket and a service ticket through a remote server (KDC) before it can access a target service. Also, clearly, our 4PAKEv2* requires the smallest number of message flows since authentication tokens are reused multiple times within a predefined validity period.

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

Protocol	$\mathbf{C} \leftrightarrow \mathbf{S}$	$\mathbf{C} \leftrightarrow \mathbf{C}$	$\mathbf{S} \leftrightarrow \mathbf{S}$	Total
PKCROSS	6	2	2	10
4PAKEv1/v2/v3	4	4	0	8
4PAKEv2*	0	2	0	2

[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 $M4$.

Delving into the bandwidth requirement, Table 3 compares the communication complexity of PKCROSS and our protocols.

Table 3: Communication complexity of PKCROSS and our protocols.

Protocol	**C** $\leftrightarrow$ **S**	**C** $\leftrightarrow$ **C**	**S** $\leftrightarrow$ **S**
PKCROSS	$1\|\text{PEnc}\| + 4\|\text{Sig}\| + 5\|\text{Enc}\|$	$3\|\text{Enc}\|$	$1\|\text{PEnc}\| + 4\|\text{Sig}\|$
4PAKEv1	$4\|\text{Sig}\| + 8\|\text{GExp}\| + 2\|\text{Mac}\|$	$2\|\text{Sig}\| + 4\|\text{GExp}\|$	0
4PAKEv2	$4\|\text{Sig}\| + 4\|\text{GExp}\| + 2\|\text{Mac}\|$	$4\|\text{Sig}\| + 2\|\text{GExp}\|$	0
4PAKEv2*	0	$4\|\text{Sig}\| + 2\|\text{GExp}\|$	0
4PAKEv3	$4\|\text{GExp}\| + 2\|\text{Enc}\| + 2\|\text{Mac}\|$	$2\|\text{GExp}\| + 2\|\text{Mac}\|$	0

In summary, the communication complexity for **C** $\leftrightarrow$ **S** in our protocols are comparable to that of PKCROSS (except that our 4PAKEv2* has zero communication cost). However, all of our protocols have higher bandwidth requirement for **C** $\leftrightarrow$ **C**. On the contrary, no communication overhead is incurred by our protocols with respect to **S** $\leftrightarrow$ **S**.

8.2 Computational Overhead

First, we evaluate the computational complexity of our protocols by counting the computationally expensive operations (*i.e.*, PEnc/PDec, Sig/Ver, and GExp) in a protocol run, and comparing them with those for PKCROSS. This is summarised in Table 4. From the table, we see that the client-side computational overhead in our protocols is expected to be slightly higher, while the server-side overhead is expected to be slightly lower, than that of PKCROSS.

Table 4: Cryptographic operation counts in PKCROSS and our protocols.

Protocol		**Operation count**
PKCROSS	– Client	1 PEnc + 1 Sig + 2 Ver
	– Service/App server	0
	– Local server	1 PEnc + 2 Sig + 4 Ver
	– Remote server	1 PEnc + 1 PDec + 1 Sig + 2 Ver
4PAKEv1	– Client	1 Ver + 4 GExp
	– Server	1 Sig + 2 GExp
4PAKEv2	– Client	1 Sig + 2 Ver + 4 GExp
	– Server	1 Sig + 2 GExp
4PAKEv2*	– Client	1 Sig + 1 Ver + 2 GExp
	– Server	0
4PAKEv3	– Client	4 GExp
	– Server	3 GExp

We then verify the efficiency differences between PKCROSS and our protocols through concrete experiments. Our experimental results are based on implementation of PKCROSS and our protocols using C++ and the OpenSSL 1.0.1 crypto library[11]. Particularly, we access and make use of the `libcrypto` library through the EVP (envelope) functions provided by OpenSSL. The experiments run on a machine equipped with an Inter 3.30GHz quad-core i3-2120 CPU, 8GB of RAM, and Ubuntu Linux 12.04.2 LTS. Table 5 presents the base time for our chosen cryptographic algorithms for the protocols.

[11] `http://www.openssl.org/`

Table 5: Raw cryptographic computation times in milliseconds (ms).

Operation	Time	Operation	Time
RSA-1024 Enc	0.0389	DH-1024 Exp	0.2093
RSA-1024 Dec	0.2274	AES/CBC-128 Enc	0.0030
RSA-1024 Sig	0.2568	AES/CBC-128 Dec	0.0041
RSA-1024 Ver	0.0170	SHA-1	0.0018
DH-1024 Gen	0.0798	HMAC (SHA-1)	0.0029

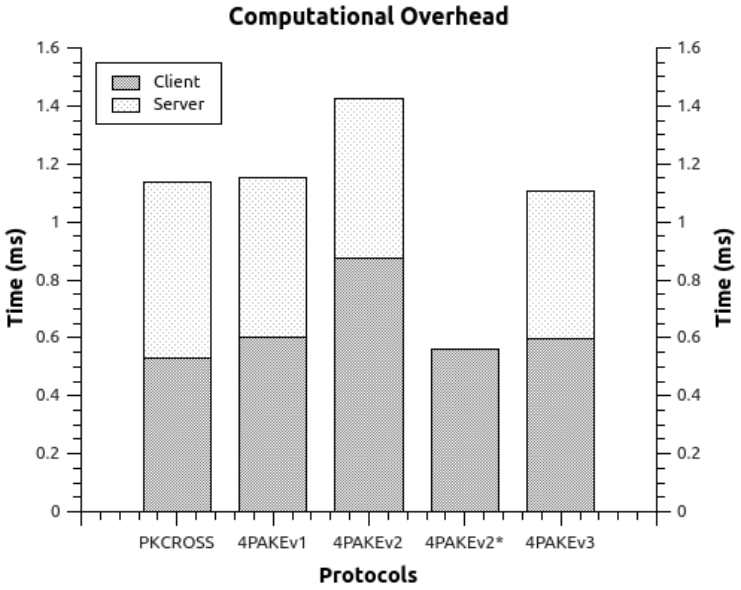

Figure 5: Comparison in terms of an average computation time for a protocol run.

Protocol execution time. We measure the computation time for each protocol. The protocol is run and an average execution time is calculated over $1,000$ iterations. (Here both the client and the server are simulated on the same machine, and thus we do not consider network latency. However, we will come back to this issue in Section 9.1 when we discuss an application of 4PAKE to secure communications over the Internet.) Figure 5 shows the results of our experiments. We also give a break down of the protocol execution time, *i.e.*, the client-side and the server-side[12] overhead. The computational overhead includes cryptographic operations and a number of management functions for initialising and clearing data structures.

From Figure 5, it is clear that the protocol execution times for 4PAKEv1 and 4PAKEv3 are very close to that of PKCROSS. Although 4PAKEv2 is less efficient than PKCROSS, it is run only once within a predefined period of time; each subsequent execution of 4PAKEv2* is approximately $2\times$ faster than PKCROSS. As expected, the client-side computational cost in all of our protocols is slightly higher (approximately between $1.05\times$ and $1.1\times$) than that of PKCROSS. This is because the clients in our protocols may have to perform not only public key signing/verifying, but also group exponentiation (during Diffie-Hellman key agreement). This is a trade-off between performance and usability (since the clients in our protocol avoid public key management and depend on only passwords). We also stress that our protocols, when relying on the Diffie-Hellman key exchange technique, additionally provide forward secrecy—a property not achievable

[12]In PKCROSS, unlike our protocols, the local server has different computational overhead than the remote server (by only roughly 0.1ms). For simplicity of exposition, our results are based on the computational cost of only the local server.

by PKCROSS. In fact, if we do not make use of Diffie-Hellman key exchange, our protocols are expected to be more efficient than PKCROSS from the client's perspective. For example, in 4PAKEv3, we can use the REKEY protocol of [15] as the 2SAKE protocol to avoid performing group exponentiation, and hence having a more lightweight client.

On the other hand, we also confirm that the server-side computational cost in our protocols (except 4PAKEv2* with no overhead) is slightly lower (approximately between 0.8× and 0.9×) in comparison with that of PKCROSS. The higher cost in PKCROSS is due to the extra interaction between the local and the remote servers.

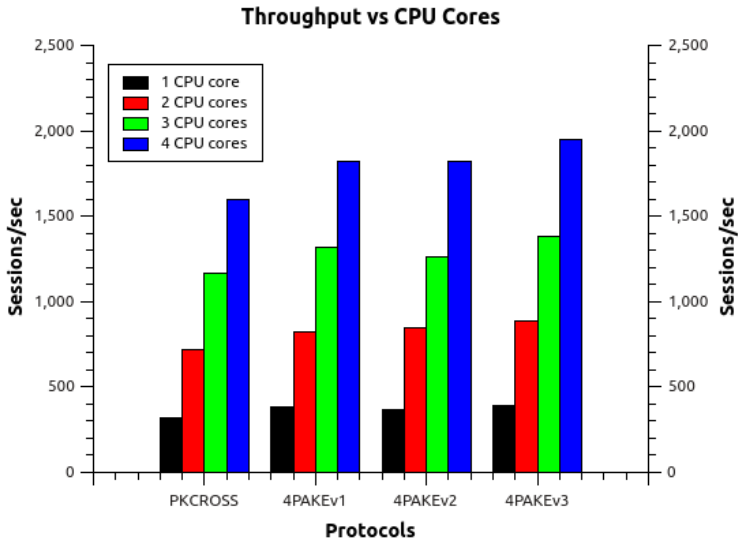

Figure 6: The maximum throughput for the server in terms of protocol sessions per second.

Scalability of server. We simulate the server for each protocol and see how well it scales with the addition of more computational resources in the form of CPU cores. In these experiments, we artificially restrict computation (through a tool called `cpulimit`[13]) to a subset of the possible cores from one to four cores supported by our machine. We then measure the number of protocol sessions that can be executed by the server in a second.

Figure 6 shows the throughput scaling of our four-core machine. For all protocols, the addition of a CPU core increases approximately between 1.4× and 2.3× the number of executed protocol sessions per second; while the use of all four cores increases the throughput by roughly 5× in comparison with just one core. More importantly, our experiments show that all our protocols are more scalable than PKCROSS. For example, the server running our 4PAKEv3 protocol with a quad-core CPU can handle up to almost $2,000$ sessions per second. This is about 25% more than the number supported by PKCROSS, which is upper bounded at approximately $1,600$ with a similar amount computational resources. Using a similar empirical analysis, we deduce that 4PAKEv1 and 4PAKEv2 are approximately 14% more scalable than PKCROSS.

9 Applications

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

[13] `http://cpulimit.sourceforge.net/`

9.1 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 online applications, such as web-based emails, online booking and Internet banking, for mutual authentication and key establishment between a user and a server. 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* (C2C-TLS) protocol in the four-party setting can be constructed in a natural way by using the following building blocks:

- hybTLS, the hybrid server-authenticated TLS handshake and username/password approach (between a client and a server);[14]
- dhTLS, the Diffie-Hellman authenticated key exchange TLS handshake protocol (between two clients) [20].

In our C2C-TLS protocol, a client first authenticates to its domain server using the hybTLS 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 C2C-TLS protocol from our generic 4PAKE framework of Figure 1, we simply replace the 2PAKE protocol by the certTLS protocol, and the 2AAKE protocol by the dhTLS protocol, such that (epk_A, esk_A) and (epk_B, esk_B) are now ephemeral Diffie-Hellman keys. Hence, $token_{S_A}$ and $token_{S_B}$ are of the form of signed Diffie-Hellman keys created by S_A and S_B, respectively.

We build a prototype of our C2C-TLS protocol in order to evaluate its feasibility in real world applications. The prototype is based on the OpenSSL API and the BIO (abstraction) library, and written in C++. We choose the `DES-CBC3-SHA` (168 bits) and the `DHE-RSA-AES256-SHA` (256 bits) TLSv1/SSLv3 cipher suites for hybTLS and dhTLS, respectively. We then conduct experiments to quantify how much time is required for a client to setup a secure communication channel with a remote client using our C2C-TLS protocol.

In our experiments, we simulate a local client using our machine (with the same specification described in Section 8.2), which is connected to the Internet at 100 Mbits/s. We also simulate a local domain server using the webmail server of Nanyang Technological University (NTU), Singapore. The local client performs hybTLS with the NTU webmail server, which in turn, artificially generates and returns an authentication token. (The token is in fact locally generated using the same machine on which the client runs.) The local client then runs dhTLS with a target remote client. For our purposes, we use four geographically distributed SSL supported web servers of educational institutions: Singapore Management University – SG, the University of Melbourne – AU, the University of Bristol – UK, and New York University – US, as the artificial remote clients. Each of these web servers already possesses signed Diffie-Hellman keys required to perform dhTLS. Hence, we treat the web server as a remote client without considering their corresponding domain server in the context of 4PAKE. This will not have any noticeable impact on our results since our goal is to determine the network latency experienced by the local client. It suffices to run hybTLS between the local client and the NTU webmail server, and dhTLS between the local client and one of the distributed web servers.

Our experimental results are obtained hourly for a period of 15 hours from 09:00 to 23:00 (GMT+8), on August 20, 2013. These are shown in Figure 7. We have network latency ranging from approximately 0.14s, when communicating with another client within SG, to approximately 1.7s, when connecting to a remote client in the UK. This is only slightly higher than a standard server-authenticated HTTPS connection between a client and a web server for online applications, *i.e.*, conventional two-party key exchange. For example, establishing an HTTPS connection (using the `AES256-SHA` cipher suite) with the web server of the University of Bristol requires approximately 1.5s.

[14] 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.

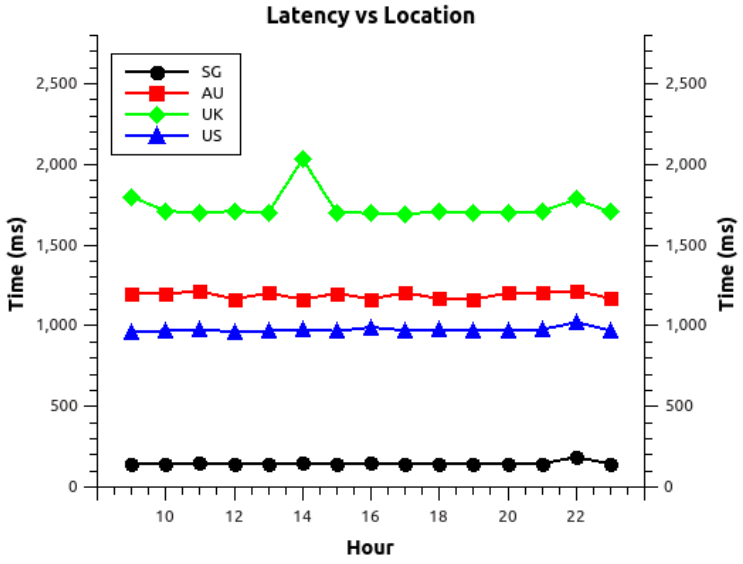

Figure 7: The latency requirement for a client running the C2C-TLS protocol.

9.2 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[15], 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 protocols fit 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, our protocols provide key privacy with respect to servers, *i.e.*, HLRs. The latter is a desirable feature that is not supported by current mobile communication systems.

9.3 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 [31], 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.

10 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

[15] http://www.3gpp.org/

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 three variants of provably secure four-party password-based authenticated key exchange protocols by using two-party key exchange protocols as building blocks. The efficiency of our protocols is comparable to that of public key Kerberos. We believe that our approach is useful for extending a legacy system with two-party protocols to the four-party setting.

Acknowledgement

We thank the anonymous reviewers for their invaluable comments and suggestions on an earlier version of this paper.

References

[1] 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.

[2] 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 (IJIS)*, 10(2):107–134, Jun 2011.

[3] 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.

[4] 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.

[5] M. Bellare, C. Namprempre. Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology – Proceedings of ASIACRYPT*, pages 531–545. Springer LNCS 1976, Dec 2000.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 72–84. IEEE Computer Society, May 1992.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] L. Chen, H.W. Lim, and G. Yang. Cross-domain password-based authenticated key exchange revisited. In *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM)*, 2013.

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

[21] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976.

[22] 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.

[23] 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.

[24] 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.

[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. *The Internet Engineering Task Force (IETF)*, Internet Draft, Nov 2001 (expires May 2002).

[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). *The Internet Engineering Task Force (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] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: theory and practice. *ACM Transactions on Computer Systems (TOCS)*, 10(4):265–310, Nov 1992.

[30] L. Law, A. Menezes, M. Qu, J. A. Solinas, and S. A. Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography (DCC)*, 28(2):119–134, Mar 2003.

[31] 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.

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

[33] 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.

[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)*, 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] Y.S.T. Tin, H. Vasanta, C. Boyd, and J. M. G. Nieto. Protocols with Security Proofs for Mobile Applications. In *Proceedings of the 9th Australasian Conference on Information Security and Privacy (ACISP)*, pages 358–369. Springer LNCS 3108, Jul 2004.

[40] 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.

[41] E. Wobber, M. Abadi, M. Burrows, and B. Lampson. Authentication in the Taos operating system. *ACM Transactions on Computer Systems (TOCS)*, 12(1):3–32, Feb 1994.

[42] 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)*, pages 544–550. IEEE Computer Society Press, May 2007.

[43] 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.

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

[45] 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.

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