

# Two-Server Password-Only Authenticated Key Exchange

Jonathan Katz\*    Philip MacKenzie†    Gelareh Taban‡    Virgil Gligor‡

## Abstract

Typical protocols for password-based authentication assume a single server which stores all the information (e.g., the password) necessary to authenticate a user. Unfortunately, an inherent limitation of this approach (assuming low-entropy passwords are used) is that the user’s password is exposed if this server is ever compromised. To address this issue, it has been suggested to share a user’s password information among multiple servers, and to have these servers cooperate (possibly in a threshold manner) when the user wants to authenticate.

We show here a two-server version of the password-based key-exchange protocol of Katz, Ostrovsky, and Yung (the *KOY protocol*). Our work gives the first provably-secure two-server protocol for the *password-only* setting (in which the user need remember only a password, and not the servers’ public keys), and is the first two-server protocol (in any setting) with a proof of security in the standard model. Our work thus fills a gap left by the work of MacKenzie et al. (*J. Crypto* 2006) and Di Riamondo and Gennaro (*JCSS* 2006). As an additional benefit of our work, we show modifications which improve the efficiency of the original KOY protocol.

## 1 Introduction

A well-known fact in the context of designing authentication systems is that human users/clients typically choose “weak”, low-entropy passwords from a relatively small space of possibilities. Unfortunately, protocols designed and proven secure for the case when clients use *cryptographic* (i.e., high-entropy) secrets are generally insecure when passwords (i.e., low-entropy secrets) are used instead; this is so because these protocols are typically not resistant to *off-line dictionary attacks* in which an eavesdropping adversary derives information about the password from observed transcripts of login sessions. In recent years, much attention has focused on designing password-based authenticated key-exchange protocols resistant to such attacks. (We remark that *on-line dictionary attacks* — in which an adversary simply attempts to log-in repeatedly, trying each possible password — cannot be prevented by cryptographic means but can be dealt with using other methods outside the scope of this work.)

Means of protecting against off-line dictionary attacks in a single-server setting were first suggested by Gong, et al. [23] in a “hybrid”, PKI-based model in which users are assumed to know the server’s public key in addition to a password. Bellare and Merritt [5] were the first to suggest protocols for what we term *password-only authenticated key exchange* (PAKE), where the clients are required to “store” (i.e., remember) *only* a short password and no additional information. These initial works (and others [6, 25, 30, 38]) were relatively informal and did not provide definitions or proofs of security. More recently, formal definitions and provably-secure protocols for the “hybrid”

---

\*jkatz@cs.umd.edu. Dept. of Computer Science, University of Maryland. Work supported by NSF CAREER award #0447075 and Trusted Computing Grant #0310751.

†Google labs. Work done while at DoCoMo USA Labs.

‡{gelareh, gligor}@eng.umd.edu. Dept. of Electrical and Computer Engineering, University of Maryland.

model have been given [24, 7], followed thereafter by formal models for the password-only setting [1, 8, 22] and associated protocols with proofs of security in the random oracle/ideal cipher models<sup>1</sup> [1, 8, 32] and in the standard model [28, 22, 20, 27]. (The protocols of [28, 20, 27] assume some public information which is available to all parties. Note, however, that since this information can be hard-coded into implementations of the protocol, clients do not need to memorize or store any high-entropy, cryptographic information as they are required to do in the PKI-based setting.)

Although the above protocols protect against off-line dictionary attacks, they do nothing to mitigate the concern that an adversary might obtain users' passwords via *server compromise*. Such attacks represent a serious threat since they are potentially cost-effective (in that an adversary might be able to obtain thousands of users' passwords by corrupting a single, poorly-protected server) and users frequently utilize the same password at multiple sites. Unfortunately, it is easy to show the *impossibility* of protecting against server compromise when a single server holds the necessary information to authenticate a user (assuming the only secret information held by the user is a low-entropy password). To protect against server compromise, Ford and Kaliski [19] thus proposed using a *threshold* protocol in which the authentication functionality is distributed across  $n$  servers who cooperate to authenticate a user, and who obtain independent session keys (shared with the user) following a successful authentication. Their protocol, which is designed in the PKI-based model, remains secure (and, in particular, an adversary learns nothing about users' passwords other than what it learns from its on-line password guesses) as long as  $n - 1$  or fewer servers are compromised. Jablon [26] gave a protocol with similar functionality in the password-only setting. Neither of these works, however, include rigorous definitions or proofs of security.

Subsequent to the work of Ford-Kaliski and Jablon, a number of provably-secure protocols for threshold password-based authentication have been given. We summarize what is known:

- MacKenzie, et al. [35] showed a protocol in the “hybrid”, PKI-based setting which requires only  $t$  out of  $n$  servers to cooperate in order to authenticate a user, for any values of  $t, n$  (of course, security is only obtained as long as  $t - 1$  or fewer servers are compromised). They prove security for their protocol in the random oracle model.
- Di Raimondo and Gennaro [16] proposed a protocol in the password-only setting with a proof of security in the standard model. (A second protocol given in their paper, which we will not discuss further, achieves the weaker functionality in which the user shares the same session key with all the servers.) Their protocol requires less than  $1/3$  of the servers to be compromised (i.e., they require  $n > 3t$ ) and thus does not give a solution for the two-server case.<sup>2</sup> We remark that, in general, threshold cryptosystems for the two-party case do not follow immediately from threshold solutions for the case of general  $n$ ; see, e.g., the work of [21, 33, 34, 31, 17] in this regard.
- Brainard, et al. [9] have developed a two-server protocol (called “Nightingale”), a variant of which has been proven secure in the random oracle model [37]. These protocols assume the PKI-based setting: as stated in the papers, the protocols assume a “secure channel” between the client and the server(s) which would in practice be implemented using public-key techniques such as SSL.

---

<sup>1</sup>In the random oracle model [2], parties are assumed to have “black-box” access to a random function. (The ideal cipher model assumes that parties have “black-box” access to a random keyed permutation, an even stronger assumption.) In practice, the random oracle is instantiated with a cryptographic hash function. It is known [10], however, that protocols secure in the random oracle model may not be secure for *any* choice of hash function.

<sup>2</sup>The approach in their paper does not extend to the case  $t \geq n/3$ . The authors mention (without details) that “[i]t is possible to improve the fault-tolerance to  $t < n/2 \dots$ ”, but even this would not imply a two-server solution.

## 1.1 Our Contributions

We show here a two-server protocol for password-only authenticated key exchange, with proof of security in the standard model. Ours is the first *provably-secure* two-server protocol in the password-only setting, and is the first two-server protocol (in any setting) with a proof of security in the standard model. Our protocol extends and builds upon the (single-server) password-based key-exchange protocol of Katz, Ostrovsky, and Yung [28] (the *KOY protocol*). As an additional benefit of our work, we show two modifications which improve the efficiency of the original KOY protocol even in the single-server case. (One of these modifications was also used in [11].)

In Section 4 we describe a “basic” two-server protocol which is secure against a passive (i.e., “honest-but-curious”) adversary who has access to the entire state of one of the servers throughout its execution of the protocol, but cannot cause this server to deviate from its prescribed behavior. (Even in the “passive” case, however, the adversary is assumed to control all communication between the client and the servers.) We believe this protocol is interesting in its own right (when the assumption on adversarial behavior is warranted), and anyway the basic protocol and its proof of security serve as a useful prelude to our second result. In Section 5 we show how to modify the basic protocol so as to achieve security against an *active* adversary who may cause a corrupted server to deviate arbitrarily from the prescribed protocol.

The protocols we construct are relatively efficient. Each party in the basic two-server protocol performs less than twice the amount of work as in the original KOY protocol. For the protocol secure against active adversaries, the work of the client stays the same but the work of the servers increases by a factor of (roughly) 6. (More explicit calculations of the computational cost are given in the appropriate sections.) We remark, however, that this does not take into account potential efficiency improvements such as off-line computation or pre-computation to speed up fixed-base exponentiations. We also note that our protocol is (slightly) more efficient than the protocol of Di Raimondo and Gennaro [16] even for the smallest threshold supported by their scheme (i.e.,  $n = 4, t = 1$ ). (Of course, it is not surprising that our protocol is more efficient, since we have fewer servers. The point is merely to illustrate that our protocol is as practical as theirs.)

## 2 Definitions and Preliminaries

We assume the reader is familiar with the model of Bellare, et al. [1] (building on [3, 4]) for password-based key exchange in the single-server case. Here, we generalize their model and present formal definitions for the case of two-server protocols. While the model presented here is largely equivalent to the model proposed by MacKenzie, et al. [35] (with the main difference that we do not assume a PKI), we can simplify matters a bit since we focus on the two-server setting exclusively. For convenience we describe the model for the case of a “passive” adversary first and then discuss briefly the modifications needed in the case of “active” adversaries. (As discussed below, in both the “passive” and “active” cases the adversary is free to interfere with all communication between the client and the servers. These cases only differ in the power of the adversary to control the actions of the corrupted servers: specifically, a “passive” adversary is unable to control the actions of corrupted servers, whereas a “active” adversary can.)

We first present a general overview of the system. For simplicity only, we assume that every client  $C$  in the system shares its password  $pw$  with exactly two servers  $A$  and  $B$ . In this case we say that servers  $A$  and  $B$  are *associated with*  $C$ . (A single server may be associated with multiple clients.) In addition to holding password shares, these servers may also be provisioned with arbitrary other information which is *not* stored by  $C$ . Any such information is assumed to be

provisioned by some incorruptible, central mechanism (a system administrator, say) at the outset of the protocol. Note that this does *not* represent an additional assumption or restriction in practice: the servers *must* minimally be provisioned with correct password shares anyway, and there is no reason why additional information cannot be provided to the servers at that time. Furthermore, the servers have no restriction — as the client does — on the amount of information they can store. An (honest) execution of a password-based key-exchange protocol between client  $C$  and associated servers  $A$  and  $B$  should result in the client holding two session keys  $\text{sk}_{C,A}, \text{sk}_{C,B}$ , and servers  $A$  and  $B$  holding  $\text{sk}_{A,C}$  and  $\text{sk}_{B,C}$ , respectively, with  $\text{sk}_{C,A} = \text{sk}_{A,C}$  and  $\text{sk}_{C,B} = \text{sk}_{B,C}$ .

## 2.1 Passive Adversaries

We assume an adversary who corrupts some servers at the outset of the protocol, such that for any client  $C$  at most one of the servers associated with  $C$  is corrupted. In the case of a passive adversary, a corrupted server continues to operate according to the protocol but the adversary may monitor its internal state.

Following [35, 16], we make the (largely conceptual) assumption that there exists a single *gateway* which is the only entity that *directly* communicates with the clients. This gateway essentially acts as a “router”, splitting messages from the clients to each of the two associated servers and aggregating messages from these servers to the client; we also allow the gateway to perform some simple, publicly-computable operations. Introduction of this gateway is not strictly necessary; however, it enables a simplification of the formal model and also provides a convenient and straightforward way to quantify the number of on-line attacks made by an adversary.

During the course of the protocol, then, there may potentially be three types of communication: between the clients and the gateway, between the servers and the gateway, and between the servers themselves. We assume the adversary has the ability to eavesdrop on all of these. We further assume that the client-gateway communication is under the full control of the adversary, and thus the adversary can send messages of its choice to either of these entities or may tamper with, delay, refuse to deliver, etc. any messages sent between clients and the gateway. On the other hand, in the case of a passive adversary we assume that the server-gateway and server-server communication is determined entirely by the protocol itself (since in the passive case a corrupted server follows the protocol exactly as specified).<sup>3</sup> In addition, the adversary is assumed to see the entire internal state of any corrupted servers throughout their execution of the protocol. With the above in mind, we proceed to the formal definitions.

**Participants, passwords, and initialization.** We assume a fixed set of protocol participants (also called principals) each of which is either a client  $C \in \text{Client}$  or a server  $S \in \text{Server}$ , where  $\text{Client}$  and  $\text{Server}$  are disjoint. Each  $C \in \text{Client}$  is assumed to have a password  $pw_C$  chosen uniformly and independently from the “dictionary”  $\{1, \dots, N\}$ .<sup>4</sup> As noted earlier, we make the simplifying assumption that each client shares his password with exactly two other servers (and no more). If client  $C$  shares his password with the distinct servers  $A, B$ , then  $A$  (resp.,  $B$ ) holds a *password share*  $pw_{C,A}$  (resp.,  $pw_{C,B}$ ); the mechanism for generating these shares depends on the protocol itself. We also allow each server to hold information in addition to these password shares. For example, as

---

<sup>3</sup>For the case of a passive adversary, the assumption that the adversary cannot tamper with the server-server communication is easy to realize via standard use of message authentication codes or signatures (as the servers can store long-term keys for this purpose). The assumption that the adversary cannot tamper with the server-gateway communication is essentially for convenience/definitional purposes only, as the adversary can anyway tamper with client-gateway communication (and the gateway processes messages in a well-defined and predictable way).

<sup>4</sup>As in other work, though, our protocol and proof of security may be adapted easily to handle arbitrary dictionaries and/or non-uniform (but known) distributions on passwords.

described in footnote 3, two servers associated with a particular client may be provisioned with a shared, symmetric key. As we have already discussed, the initialization phase during which this information is provisioned is assumed to be carried out by some trusted authority. Any information stored by a corrupted server is available to the adversary.

In general, additional information can be generated during the initialization phase. For example, in the “hybrid” password/PKI model [24, 7] public/secret key pairs are generated for each server and the secret key is given as input to the appropriate server, while the public key is provided to the appropriate client(s). For the protocol presented here, we require only the weaker requirement of a single set of public parameters which is provided to all parties.

**Execution of the protocol.** In the real world, a protocol determines how principals behave in response to input from their environment. In the formal model, these inputs are provided by the adversary. Each principal is assumed to be able to execute the protocol multiple times (possibly concurrently) with different partners; this is modeled by allowing each principal to have an unlimited number of *instances* [4, 1] with which to execute the protocol. We denote instance  $i$  of principal  $U$  as  $\Pi_U^i$ . A given instance may be used only once. The adversary is given oracle access to these different instances; furthermore, each instance maintains (local) state which is updated during the course of the experiment. In particular, each instance  $\Pi_U^i$  has associated with it the following variables, initialized as NULL or FALSE (as appropriate) during the initialization phase:

- $\text{sid}_U^i$ ,  $\text{pid}_U^i$ , and  $\text{sk}_U^i$  are variables containing the *session id*, *partner id*, and *session key(s)* for an instance, respectively. The session id is simply a way to keep track of the different executions of a particular user  $U$ ; we specify below how the session id is determined. The partner id denotes the identity of the principal with whom  $\Pi_U^i$  believes it is interacting. A client’s partner id will be a set of two servers; a server’s partner id will be a single client (viewed as a set for notational convenience). For  $C$  a client,  $\text{sk}_C^i$  consists of a pair  $\text{sk}_{C,A}^i, \text{sk}_{C,B}^i$ , where these are the keys shared with servers  $A$  and  $B$ , respectively. A server instance  $\Pi_S^i$  with partner  $C$  has only a single session key  $\text{sk}_{S,C}^i$ .
- $\text{term}_U^i$  and  $\text{acc}_U^i$  are boolean variables denoting whether a given instance has terminated or accepted, respectively. Termination means that a given instance is done sending and receiving messages; acceptance indicates successful termination, where the semantics of “success” depend on the protocol. In our case, acceptance implies that the instance believes it has established a session key with its intended partner; thus, when an instance  $\Pi_U^i$  accepts,  $\text{sid}_U^i$ ,  $\text{pid}_U^i$ , and  $\text{sk}_U^i$  are no longer null.  $\text{state}_U^i$  records any state necessary for execution of the protocol by  $\Pi_U^i$ .  $\text{used}_U^i$  is a boolean variable denoting whether an instance has begun executing the protocol; this is a formalism which will ensure that each instance is used only once.

For ease of presentation, we will not explicitly refer to most of the above variables when describing our protocols; however, they are implicit and would be necessary for a fully formal specification.

As highlighted earlier, the adversary is assumed to have complete control over all communication between the client and the gateway. This is modeled via access to *oracles* which are essentially as in [1] and are described now:

- $\text{Send}(C, i, \text{msg})$  — This sends message  $\text{msg}$  to instance  $\Pi_C^i$ , where  $C \in \text{Client}$ . This instance runs according to the protocol specification, and its output (i.e., the message sent by the instance in response) is given to the adversary.
- $\text{Send}(A, B, i, \text{msg})$  — This sends message  $\text{msg}$  to the gateway, which then delivers appropriate messages to instances  $\Pi_A^i$  and  $\Pi_B^i$  of servers  $A, B$ . (Since the gateway-server communication

is not under the adversary’s control, we may assume for simplicity that the gateway forwards the appropriate messages to servers using the same instance-number  $i$ .) These instances each run according to the protocol specification and then send a message to the gateway; the gateway aggregates the messages from the two servers and generates some output (i.e., a message directed to the client) which is given to the adversary. In addition, the adversary receives the transcript of any server-server communication which occurred as a result of this query, and also receives the current value of  $\text{state}_S^i$  if server  $S \in \{A, B\}$  is corrupted.<sup>5</sup>

- **Execute**( $C, i, A, B, j$ ) (where  $C \in \text{Client}$  and  $A, B \in \text{Server}$ ) — This oracle executes the protocol between instances  $\Pi_C^i$  and  $\Pi_A^j, \Pi_B^j$  (via the gateway) and outputs the transcript of this execution. The transcript includes both client-gateway communication as well as all server-server communication. In addition, the adversary is given the final value of  $\text{state}_S^i$  if server  $S \in \{A, B\}$  is corrupted.
- **Reveal**( $U, U', i$ ) — This outputs  $\text{sk}_{U,U'}^i$ , the session key held by instance  $\Pi_U^i$  and intended for  $U'$  (we assume  $U' \in \text{pid}_U^i$ ). This oracle call models possible leakage of session keys due to, for example, improper erasure of session keys, compromise of a host computer, or cryptanalysis.
- **Test**( $U, U', i$ ) — This oracle does not model any real capability of the adversary, but is instead needed for a definition of security. As in the case of a **Reveal** query, we assume  $U' \in \text{pid}_U^i$ . If  $\text{sk}_{U,U'}^i = \text{NULL}$ , then this oracle outputs  $\perp$ . Otherwise, a random bit  $b$  is generated; if  $b = 1$  the adversary is given  $\text{sk}_{U,U'}^i$ , and if  $b = 0$  the adversary is given a random session key. The adversary is allowed only a single **Test** query, at any time during its execution.

As usual, **Send** oracle calls are intended to model “active” attacks in the network (i.e., “on-line attacks”), whereas **Execute** calls are intended to model passive eavesdropping (i.e., “off-line attacks”).

**Session ids and partnering.** As indicated above, all client-server communication is mediated by the gateway; this allows us to define a natural notion of partnering based on matching transcripts. For a client instance  $\Pi_C^i$ , define  $\text{sid}_C^i$  to be the (ordered) sequence of incoming and outgoing messages for this instance; this sequence is clear from the oracle calls made by the adversary. For a server instance  $\Pi_S^j$ , let  $\text{sid}_S^j$  be the (ordered) sequence of incoming and outgoing messages sent by the gateway and corresponding to this instance. (I.e., a query **Send**( $A, B, i, \text{msg}$ ) implies that  $\text{msg}$  is included among in-going messages to both  $\Pi_A^i$  and  $\Pi_B^i$ ; the gateway’s response to this query is included among out-going messages from  $\Pi_A^i, \Pi_B^i$ .) Again, this sequence is clear from the oracle calls of the adversary. Note that  $\text{sid}_S^j$  is *not* identical to the set of messages actually sent and received by server  $S$ ; in particular, server-server messages are not included in  $\text{sid}_S^j$ , and furthermore a message sent to the gateway by  $S$  (resp., to  $S$  by the gateway) will not, in general, be identical to the message sent by the gateway to the client (resp., by the client to the gateway).<sup>6</sup>

With the above in mind, we may now define a notion of partnering. Let  $C \in \text{Client}$  and  $S \in \text{Server}$ . We say that instances  $\Pi_C^i$  and  $\Pi_S^j$  are *partnered* if: (1)  $\text{sid}_C^i = \text{sid}_S^j \neq \text{NULL}$ ; and (2)  $S \in \text{pid}_C^i$  and  $C \in \text{pid}_S^j$ .

**Correctness.** We require a key-exchange protocol to satisfy the following notion of correctness for  $C$  a client and  $S$  a server: if  $\Pi_C^i$  and  $\Pi_S^j$  are partnered and  $\text{acc}_C^i = \text{acc}_S^j = \text{TRUE}$ , then  $\text{sk}_{C,S}^i = \text{sk}_{S,C}^j$  (i.e., they conclude with the same session key).

<sup>5</sup>Providing the adversary with server-gateway communication is unnecessary since it is a publicly-computable function of the client-gateway communication.

<sup>6</sup>Lest the definition of  $\text{sid}_S^j$  seem strange, we remark that session ids are really useful only insofar as they allow a formal definition of partnering and hence formal definitions of correctness and “freshness” (see below). Session ids serve no purpose in real-world executions of the protocol.

**Freshness.** To formally define the adversary’s success we will need to define a notion of *freshness* which will incorporate the requirement that the associated participant be non-corrupted. A session key  $\text{sk}_{U,U'}^i$  is *fresh* if the following hold: (1) neither  $U$  nor  $U'$  is a corrupted server, (2) the adversary never queried  $\text{Reveal}(U, U', i)$ ; and (3) the adversary never queried  $\text{Reveal}(U', U, j)$ , where  $\Pi_{U'}^j$  and  $\Pi_U^i$  are partnered.

**Advantage of the adversary.** Informally, the adversary succeeds if it can guess the bit  $b$  used by the **Test** oracle on a “fresh” instance associated with a non-corrupted participant. We say an adversary  $A$  *succeeds* if it makes a single query  $\text{Test}(U, U', i)$  regarding a fresh key  $\text{sk}_{U,U'}^i$ , and outputs a single bit  $b'$  with  $b' = b$  (recall that  $b$  is the bit chosen by the **Test** oracle). We denote this event by **Succ**. Note that requiring the adversary to make its **Test** query regarding a fresh key is necessary for a meaningful definition of security. The advantage of adversary  $A$  in attacking protocol  $P$  is then given by:

$$\text{Adv}_{A,P}(k) \stackrel{\text{def}}{=} 2 \cdot \Pr[\text{Succ}] - 1,$$

where the probability is taken over the random coins used by the adversary as well as the random coins used during the course of the experiment.

It remains to define a secure protocol, as a PPT adversary can always succeed by trying all passwords one-by-one in an on-line impersonation attack. Informally, a protocol is secure if this is the best an adversary can do. Formally, an instance  $\Pi_U^i$  represents an *on-line attack* if the adversary had ever queried  $\text{Send}(U, i, \star)$  (when  $U$  is a client) or  $\text{Send}(U', U'', i, \star)$  with  $U \in \{U', U''\}$  (when  $U$  is a server). In particular, instances with which the adversary interacts via **Execute** queries are not counted as on-line attacks. The number of on-line attacks represents a bound on the number of passwords the adversary could have tested in an on-line fashion. This motivates the following definition:

**Definition 1** Protocol  $P$  is a secure two-server protocol for password-only authenticated key-exchange if there exists a constant  $c$  such that, for all dictionary sizes  $N$  and for all PPT adversaries  $A$  making at most  $Q(k)$  on-line attacks and corrupting at most one server associated with each client, there exists a negligible function  $\varepsilon(\cdot)$  such that  $\text{Adv}_{A,P}(k) \leq c \cdot Q(k)/N + \varepsilon(k)$ .

Of course, we would optimally like to achieve  $c = 1$  in the above definition; however, as in previous definitions and protocols [38, 1, 22] we will allow  $c \neq 1$  as well. The proof of security for our protocol shows that we achieve  $c = 2$ , indicating that the adversary can (essentially) do no better than guess *two* passwords during each on-line attack.

**Explicit mutual authentication.** The above definition captures the requirement of *implicit* authentication only (and the protocol we present here achieves only implicit authentication). Using standard techniques, however, it is easy to add explicit authentication to any protocol achieving implicit authentication.

## 2.2 Active Adversaries

The only difference in the active case is that the adversary may now cause any corrupted servers to deviate in an arbitrary way from the actions prescribed by the protocol. Thus, if a server is corrupted the adversary controls all messages sent from this server to the gateway as well as messages sent from this server to any other server. As in the passive case, however, we continue to assume that communication between the gateway and any non-corrupted servers (as well as communication between two non-corrupted servers) is *not* under adversarial control. See footnote 3 for the rationale behind these conventions.

### 3 A Review of the KOY Protocol

Here, we provide a brief review of the KOY protocol [28] which will be useful toward understanding our two-server protocol in the following section. We also discuss the modifications we introduce to the KOY protocol which have the effect of both simplifying our eventual two-server protocol as well as improving the efficiency of the KOY protocol even in the single-server setting.

The KOY protocol operates in three rounds. At a high level, in round 1 the client generates a Cramer-Shoup encryption of its password (using as a public key a value contained in the public parameters) and sends the resulting ciphertext to the server; the server acts symmetrically in round 2. We observe that the following modifications may be made without affecting security of the protocol:

- First, *the server may use El Gamal encryption in round 2* rather than Cramer-Shoup encryption (this was first observed, in a different context, in [11]). This is somewhat surprising, as the proof of security in [28] explicitly relies on the chosen-ciphertext security of Cramer-Shoup encryption, while El Gamal encryption is only semantically secure.
- Second, *there is no need for the server to compute a new El Gamal encryption of the password (in round 2) in each execution of the protocol*; instead, the server may simply store once-and-for-all an encryption  $C$  of the password (along with the randomness used to construct  $C$ ), and then use  $C$  every time it executes the protocol.

We refer to the KOY protocol with the above modifications as the *KOY\* protocol*. (We do not formally prove security of the KOY\* protocol in the single-server case; however, such a proof may be easily derived from the proofs of the two-server protocols that we give in this paper.)

In brief (we are being deliberately brief, since we will provide more detail when we discuss the two-server version of this protocol in the following section), then, the KOY\* protocol assumes public parameters containing a description of a group  $\mathbb{G}$  with specified prime order  $q$ . Additionally, the parameters include random generators  $g_1, g_2, g_3, h, c, d \in \mathbb{G}$  and a hash function  $G$ . Components  $\langle g_1, g_2, h, c, d \rangle$  will be used for Cramer-Shoup encryption by the client in round 1 (as in the original KOY protocol), while  $\langle g_1, g_3 \rangle$  will be used for El Gamal encryption by the server in round 2.

A password  $pw_C \in \mathbb{Z}_q$  is chosen at random for each client  $C$  and shared with the appropriate server  $S$ . In addition,  $S$  is pre-provisioned with both  $r'$  and the ciphertext  $\text{Com}_{S,C} \stackrel{\text{def}}{=} (g_1^{r'}, g_3^{r'} g_1^{pw_C})$ , where  $r' \in \mathbb{Z}_q$  is chosen at random; note that  $\text{Com}_{S,C}$  is simply a (random) El Gamal encryption of  $g_1^{pw_C}$ .

Execution of the protocol proceeds as follows: when a client with password  $pw_C$  wants to initiate an execution of the protocol, it runs a key-generation algorithm for a one-time signature scheme, yielding VK and SK. It then chooses a random  $r \in \mathbb{Z}_q$  and computes  $A = g_1^r$ ,  $B = g_2^r$ , and  $C = h^r \cdot g_1^{pw_C}$ . It then computes  $\alpha = H(\text{Client}|\text{VK}|A|B|C)$  and sets  $D = (cd^\alpha)^r$ . Note that  $(A, B, C, D)$  is a (labeled) Cramer-Shoup encryption of  $g_1^{pw_C}$ . The client sends

$$\text{msg}_1 \stackrel{\text{def}}{=} \langle \text{Client}, \text{VK}, A, B, C, D \rangle$$

to the server.

The server responds by re-computing  $\alpha$ , choosing random  $x', y', z', w' \in \mathbb{Z}_q$ , and computing  $E = g_1^{x'} g_2^{y'} h^{z'} (cd^\alpha)^{w'}$ . It then sends the message

$$\text{msg}_2 \stackrel{\text{def}}{=} \langle E, F, G \rangle$$

to the client, where  $(F, G) = \text{Com}_{S,C}$ .

Upon receiving  $\text{msg}_2$ , the client chooses random  $x, y \in \mathbb{Z}_q$  and computes  $K = g_1^x g_3^y$ . It also computes a signature  $\sigma$  on  $(\text{msg}_1, \text{msg}_2, K)$  using  $\text{SK}$ , and sends

$$\text{msg}_3 \stackrel{\text{def}}{=} \langle K, \sigma \rangle$$

to the server. The client concludes by computing its session key as  $\text{sk}_C = E^r F^x (G/g_1^{pw_C})^y$ .

The server, upon receiving  $\text{msg}_3$ , verifies the signature and aborts if it is incorrect. Otherwise, it computes its session key as  $\text{sk}_S = A^{x'} B^{y'} (C/g_1^{pw_C})^{z'} D^{w'} K^{r'}$ . It may be easily verified that in a correct execution the client and server compute identical session keys.

## 4 A Protocol Secure Against Passive Adversaries

### 4.1 Description of the Protocol

We assume the reader is familiar with the decisional Diffie-Hellman (DDH) assumption [15], strong<sup>7</sup> one-time signature schemes, and the Cramer-Shoup encryption scheme [14] with labels. A high-level depiction of the protocol is given in Figures 1–3, and a more detailed description, as well as some informal discussion about the protocol, follows.

**Initialization.** During the initialization phase, we assume the generation of public parameters (i.e., a common reference string) which are then made available to all parties. For a given security parameter  $k$ , the public parameters will contain a group  $\mathbb{G}$  (written multiplicatively) having specified prime order  $q$  with  $|q| = k$ ; we assume the hardness of the DDH problem in  $\mathbb{G}$ . Additionally, the parameters include random generators  $g_1, g_2, g_3, h, c, d \in \mathbb{G} \setminus \{1\}$  and a hash function  $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$  chosen at random from a collision-resistant hash family.

As part of the initialization, each server  $S$  is provisioned with an El Gamal [18] public-/secret-key pair  $(pk_S, sk_S)$ , where  $pk_S = g_1^{sk_S}$ . If  $A$  and  $B$  are associated with the same client  $C$ , then  $A$  (resp.,  $B$ ) is given  $pk_B$  (resp.,  $pk_A$ ). We stress that, in contrast to the PKI-based model, the client is not assumed or required to know the public keys of any of the servers.

Given a message  $m \in \mathbb{G}$  and a public key (i.e., group element)  $pk$ , we let  $M \leftarrow \text{ElG}_{pk}(m)$  denote the act of choosing a random  $r \in \mathbb{Z}_q$  and setting  $M = (g_1^r, pk^r m)$ . We let  $M[1]$  refer to the first component of this ciphertext, and let  $M[2]$  refer to the second. Note that if  $sk$  is the corresponding secret key (i.e.,  $pk = g_1^{sk}$ ), then we have  $m = \frac{M[2]}{M[1]^{sk}}$ .

Passwords and password shares are provisioned in the following way: a password  $pw_C$  is chosen randomly for each client  $C$  and we assume that this password can be mapped in a one-to-one fashion to  $\mathbb{Z}_q$ . If  $A$  and  $B$  are the servers associated with a client  $C$ , then password shares  $pw_{A,C}, pw_{B,C} \in \mathbb{Z}_q$  are chosen uniformly at random subject to  $pw_{A,C} + pw_{B,C} = pw_C \pmod q$ , with  $pw_{A,C}$  given to server  $A$  and  $pw_{B,C}$  given to server  $B$ . In addition, both  $A$  and  $B$  are given  $\text{Com}_{A,C}, \text{Com}'_{A,C}, \text{Com}_{B,C}$ , and  $\text{Com}'_{B,C}$ , where:

$$\begin{aligned} \text{Com}_{A,C} &\stackrel{\text{def}}{=} \text{ElG}_{g_3}(g_1^{pw_{A,C}}) = (g_1^{r_a}, g_3^{r_a} g_1^{pw_{A,C}}) \\ \text{Com}'_{A,C} &\stackrel{\text{def}}{=} \text{ElG}_{pk_A}(g_1^{pw_{A,C}}) \\ \text{Com}_{B,C} &\stackrel{\text{def}}{=} \text{ElG}_{g_3}(g_1^{pw_{B,C}}) = (g_1^{r_b}, g_3^{r_b} g_1^{pw_{B,C}}) \\ \text{Com}'_{B,C} &\stackrel{\text{def}}{=} \text{ElG}_{pk_B}(g_1^{pw_{B,C}}). \end{aligned}$$

<sup>7</sup>In a *strong* signature scheme, an adversary cannot even forge a new signature on a previously-signed message.

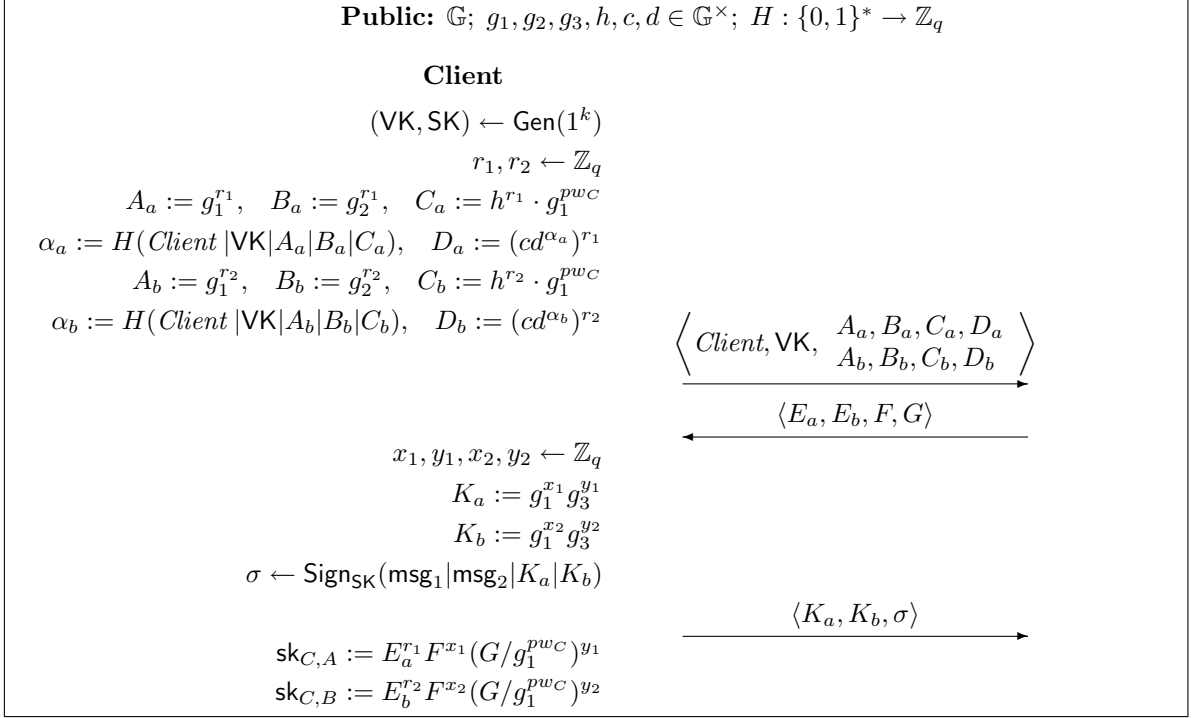


Figure 1: An execution of the protocol from the client's point of view.

(Note that the keys for these El Gamal encryptions differ.) Server  $A$  (resp., server  $B$ ) is additionally given the randomness  $r_a$  (resp.,  $r_b$ ) used to construct  $\text{Com}_{A,C}$  (resp.,  $\text{Com}_{B,C}$ ).

**Protocol execution.** At a high level one can view our protocol as two executions of the KOY\* protocol, one between the client and server  $A$  (using server  $B$  to assist with the authentication), and one between the client and server  $B$  (using server  $A$  to assist with the authentication). For efficiency, the signature and verification for the two executions are combined.

When a client with password  $pw_C$  wants to initiate an execution of the protocol, this client computes Cramer-Shoup “encryptions” of  $pw_C$  for each of the two servers. In more detail (cf. Figure 1), the client begins by running a key-generation algorithm for a one-time signature scheme, yielding verification key  $\text{VK}$  and signing key  $\text{SK}$ . The client next chooses random  $r_1 \in \mathbb{Z}_q$  and computes  $A_a = g_1^{r_1}$ ,  $B_a = g_2^{r_1}$ , and  $C_a = h^{r_1} \cdot g_1^{pw_C}$ . The client then computes  $\alpha_a = H(\text{Client}|\text{VK}|A_a|B_a|C_a)$  and sets  $D = (cd^{\alpha_a})^{r_1}$ . This exact procedure is then carried out again, using an independent random value  $r_2 \in \mathbb{Z}_q$ . The client sends

$$\text{msg}_1 \stackrel{\text{def}}{=} \langle \text{Client}, \text{VK}, A_a, B_a, C_a, D_a, A_b, B_b, C_b, D_b \rangle$$

to the gateway as the first message of the protocol. Note that this corresponds to two independent “encryptions” of  $pw_C$  using the label  $\text{Client}|\text{VK}$ .

When the gateway receives  $\text{msg}_1$  from a client, the gateway simply forwards this message to the appropriate servers. The servers act symmetrically, so for simplicity we simply describe the actions of server  $A$  (cf. Figure 2). Upon receiving  $\text{msg}_1$ , server  $A$  sends “shares” of (1) two values of the form  $g_1^x g_2^y h^z (cd^\alpha)^w$  (for  $\alpha \in \{\alpha_a, \alpha_b\}$ ), one for server  $A$  and one for server  $B$ , and (2) an El Gamal encryption of  $pw_C$ . In more detail, server  $A$  chooses random  $x_a, y_a, z_a, w_a \in \mathbb{Z}_q$  and computes  $E_{a,1} = g_1^{x_a} g_2^{y_a} h^{z_a} (cd^{\alpha_a})^{w_a}$ . It also chooses random  $x'_a, y'_a, z'_a, w'_a \in \mathbb{Z}_q$  and computes

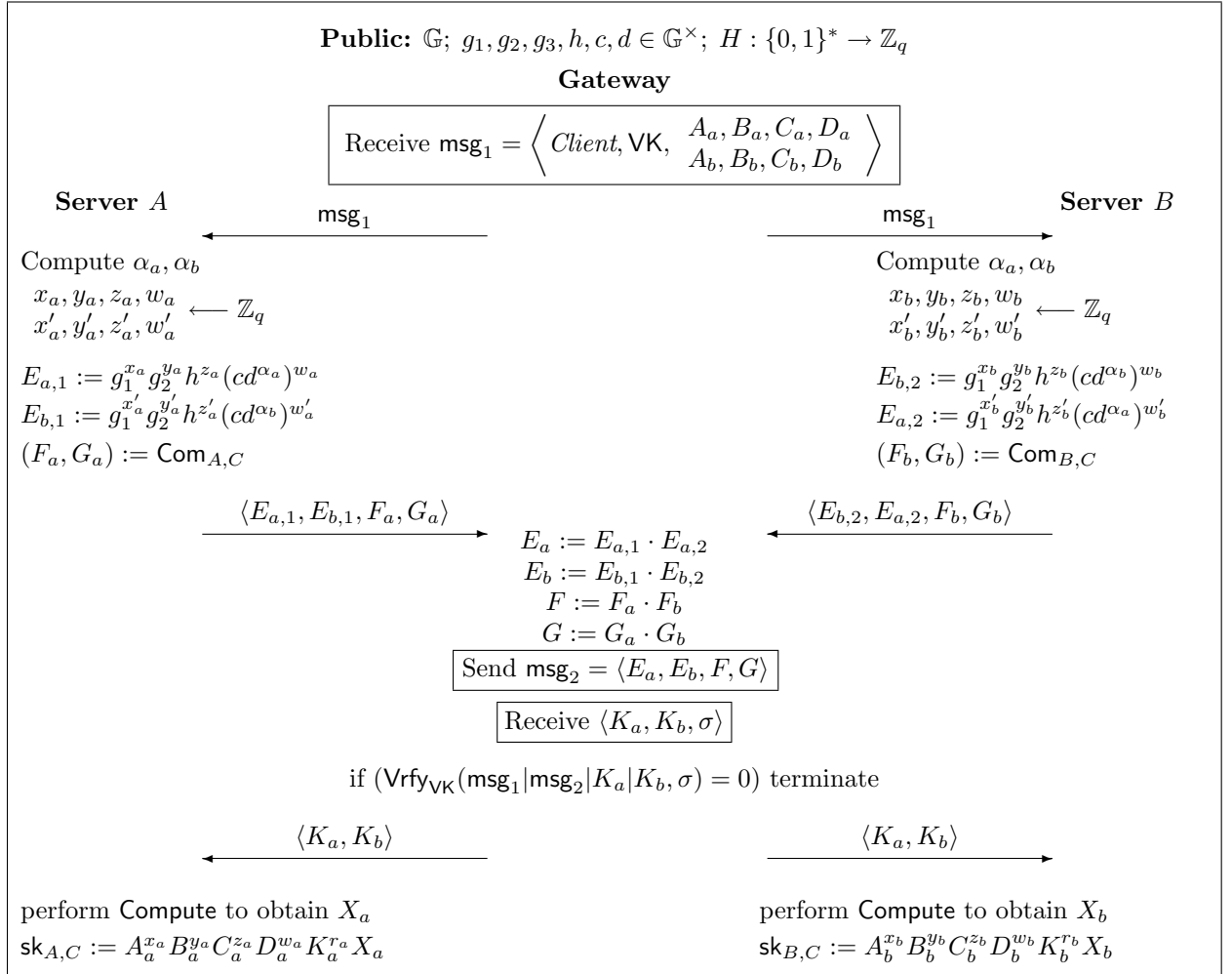


Figure 2: Execution of the protocol from the servers' points of view (messages sent to/from the gateway are boxed); see text for details. The Compute protocol is depicted in Figure 3.

$E_{b,1} = g_1^{x'_a} g_2^{y'_a} h^{z'_a} (cd^{\alpha_b})^{w'_a}$ . Finally, it sets  $(F_a, G_a)$  equal to  $\text{Com}_{A,C}$  (which, recall, is an El Gamal encryption of  $g_1^{pw_{A,C}}$  using "public key"  $g_3$  and randomness  $r_a$ ). The message  $\langle E_{a,1}, E_{b,1}, F_a, G_a \rangle$  is sent to the gateway.

The gateway combines the values it receives from the servers by multiplying them component-wise. This results in a message  $\text{msg}_2 = \langle E_a, E_b, F, G \rangle$  which is sent to the client, and for which (1) neither server knows the representation of  $E_a$  with respect to the basis  $g_1, g_2, h, (cd^{\alpha_a})$  (and similarly for  $E_b$  with respect to  $g_1, g_2, h, (cd^{\alpha_b})$ ), and (2) the values  $(F, G)$  form an El Gamal encryption of the client's password  $pw_C$  (with respect to public key  $g_3$ ).

Upon receiving  $\text{msg}_2$ , the client proceeds as follows (cf. Figure 1): it chooses random values  $x_1, y_1, x_2, y_2 \in \mathbb{Z}_q$  and computes  $K_a = g_1^{x_1} g_3^{y_1}$  and  $K_b = g_1^{x_2} g_3^{y_2}$ . It then computes a signature  $\sigma$  on  $\text{msg}_1 | \text{msg}_2 | K_a | K_b$  using the secret key  $\text{SK}$  that it had previously generated. It sends  $\langle K_a, K_b, \sigma \rangle$  to the gateway and computes session keys

$$\begin{aligned} \text{sk}_{C,A} &:= E_a^{r_1} F^{x_1} (G/g_1^{pw_C})^{y_1} \\ \text{sk}_{C,B} &:= E_b^{r_2} F^{x_2} (G/g_1^{pw_C})^{y_2}. \end{aligned}$$

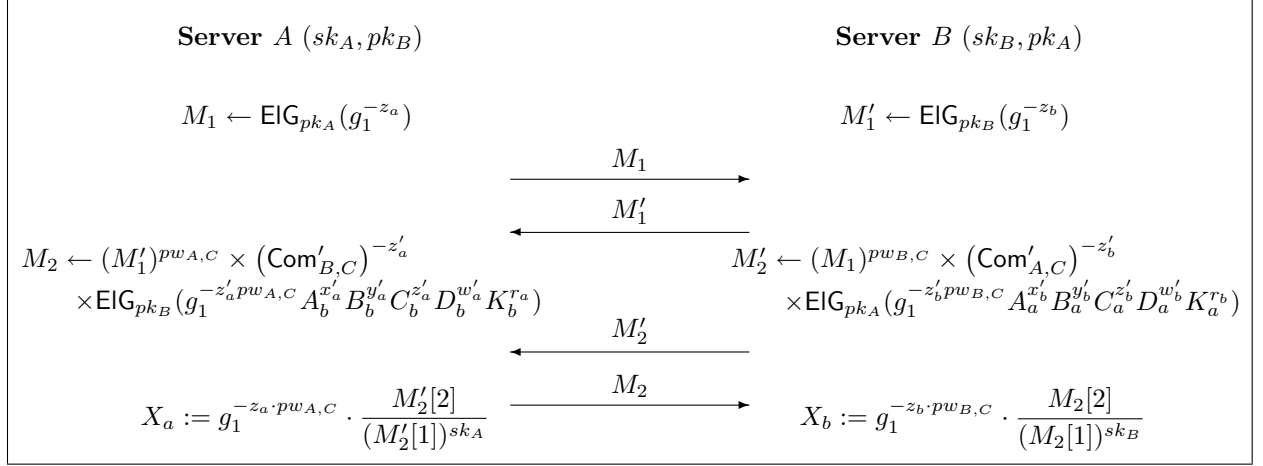


Figure 3: The Compute protocol. See text for a description of the notation used.

Upon receiving  $\langle K_a, K_b, \sigma \rangle$  from the client, the gateway first verifies that  $\sigma$  is a valid signature on  $\text{msg}_1 | \text{msg}_2 | K_a | K_b$  with respect to the key  $\text{VK}$ .<sup>8</sup> If verification fails, the gateway terminates this instance of this protocol and send a special message to the servers indicating this fact. Otherwise, it sends  $K_a, K_b$  to each of the servers. The servers then execute the Compute protocol (cf. Figure 3 and described next) in order to compute their session keys.

Before describing the Compute protocol, we introduce some notation for manipulation of El Gamal ciphertexts. If  $M, M'$  are two El Gamal ciphertexts (encrypted with respect to the same public key  $pk$ ), then we let  $M \times M'$  denote  $(M[1] \cdot M'[1], M[2] \cdot M'[2])$ . Note that if  $M$  is an encryption of  $m$  and  $M'$  is an encryption of  $m'$ , then  $M \times M'$  is an encryption of  $m \cdot m'$ . For  $x \in \mathbb{Z}_q$ , we let  $M^x$  denote the ciphertext  $(M[1]^x, M[2]^x)$ . Here, the resulting ciphertext is an encryption of  $m^x$ .

With this in mind, we now describe the Compute protocol. Since the protocol is symmetric, we simply describe it from the point of view of server A. This server sets  $M_1$  to be an El Gamal encryption (with respect to  $pk_A$ ) of  $g_1^{-z_a}$ . It then sends  $M_1$  to server B, who computes

$$M'_2 \leftarrow M_1^{pw_{B,C}} \times (\text{Com}'_{A,C})^{-z'_b} \times \text{ElG}_{pk_A}(g_1^{-z'_b pw_{B,C}} A_a^{x'_b} B_a^{y'_b} C_a^{z'_b} D_a^{w'_b} K_a^{r_b})$$

and sends this value back to server A (recall that  $r_b$  is the randomness used to construct  $\text{Com}_{B,C}$ ). Finally, server A decrypts  $M'_2$  and multiplies the result by  $g_1^{-z_a \cdot pw_{A,C}}$  to obtain  $X_a$ . Note that

$$X_a = g_1^{-(z_a + z'_b) \cdot pw_C} \cdot (A_a^{x'_b} B_a^{y'_b} C_a^{z'_b} D_a^{w'_b} K_a^{r_b}), \quad (1)$$

using the fact that  $pw_C = pw_{A,C} + pw_{B,C} \pmod q$ .

Although omitted in the above description, we assume that the client and the gateway always verify that incoming messages are well-formed, and in particular that all appropriate components of the various messages indeed lie in  $\mathbb{G}$  (we assume that membership in  $\mathbb{G}$  can be efficiently verified).

**Correctness.** Since the protocol is symmetric, we focus only on the session key shared between the client and server A. The client computes:

$$\text{sk}_{C,A} = E_a^{r_1} F^{x_1} (G/g_1^{pw_C})^{y_1}$$

<sup>8</sup>We remark that this can just as easily be done by the servers; allowing the gateway to perform this check, however, simplifies things slightly.

$$= \left( g_1^{x_a+x'_b} g_2^{y_a+y'_b} h^{z_a+z'_b} (cd^{\alpha_a})^{w_a+w'_b} \right)^{r_1} g_1^{(r_a+r_b) \cdot x_1} g_3^{(r_a+r_b) \cdot y_1}.$$

Meanwhile, server  $A$  computes:

$$\begin{aligned} \text{sk}_{A,C} &= A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a} K_a^{r_a} X_a \\ &= A_a^{x_a+x'_b} B_a^{y_a+y'_b} C_a^{z_a+z'_b} D_a^{w_a+w'_b} K_a^{r_a+r_b} \cdot g_1^{-(z_a+z'_b) \cdot pw_C} \\ &= A_a^{x_a+x'_b} B_a^{y_a+y'_b} (C_a \cdot g_1^{-pw_C})^{z_a+z'_b} D_a^{w_a+w'_b} K_a^{r_a+r_b}, \end{aligned}$$

using Equation (1) to substitute for the value of  $X_a$ . Continuing, we have:

$$\text{sk}_{A,C} = \left( g_1^{x_a+x'_b} g_2^{y_a+y'_b} h^{z_a+z'_b} (cd^{\alpha_a})^{w_a+w'_b} \right)^{r_1} (g_1^{x_1} g_3^{y_1})^{r_a+r_b},$$

which, by inspection, is equal to  $\text{sk}_{C,A}$ .

**Efficiency.** Counting exponentiations only (and assuming a multi-exponentiation is equivalent to 1.5 exponentiations), we see that the client performs 15 exponentiations in the basic protocol while each server performs 11.5. In the original KOY protocol, both the client and the server were required to perform 7.5 exponentiations. (The KOY\* protocol would be more efficient for the server.) Thus, this basic protocol increases each parties' computation by a factor of at most 2.

## 4.2 Proof of Security for Passive Adversaries

We prove the following theorem regarding the protocol of the previous section:

**Theorem 1** *Assuming (1) the DDH problem is hard for  $\mathbb{G}$ ; (2)  $(\text{Gen}, \text{Sign}, \text{Vrfy})$  is a secure one-time signature scheme; and (3)  $H$  is collision-resistant, the protocol of Figures 1–3 is a secure two-server protocol for password-only authenticated key exchange in the presence of a passive adversary (in particular, it satisfies Definition 1 with  $c = 2$ ).*

Before proving this theorem, we introduce some useful conventions. We view the adversary's queries to its `Send` oracles as queries to four different oracles `Send`<sub>0</sub>, `Send`<sub>1</sub>, `Send`<sub>2</sub>, `Send`<sub>3</sub> corresponding, respectively, to the initial request for a client to initiate the protocol as well as the three protocol messages sent between a client and the gateway. Thus, for example, a query `Send`<sub>0</sub>( $C, i$ ) represents a request for instance  $\Pi_C^i$  of client  $C$  to initiate execution of the protocol; the output of this query is an initial message  $\langle "C", \text{VK}, A_a, \dots, D_a, A_b, \dots, D_b \rangle$ . Similarly, a query `Send`<sub>3</sub>( $A, B, i, \langle K_a, K_b, \sigma \rangle$ ) represents the sending of message  $\langle K_a, K_b, \sigma \rangle$  to the gateway, intended for instances  $\Pi_A^i$  and  $\Pi_B^i$  of the servers  $A$  and  $B$ . In the proof, we always let `msg`<sub>1</sub> (resp., `msg`<sub>2</sub>, `msg`<sub>3</sub>) denote the first (resp., second or third) message of an execution of the protocol.

**Proof** Given a passive adversary  $\mathcal{A}$  attacking the protocol, we imagine a simulator which runs the protocol for  $\mathcal{A}$ . More precisely, after the adversary chooses which servers it wishes to corrupt, the simulator initializes the system (this includes selecting the public parameters, choosing passwords for all clients, and computing password shares — as well as any other necessary information — for all servers), gives  $\mathcal{A}$  the information held by all corrupted servers, and then responds to the oracle calls made by  $\mathcal{A}$ . After computing the appropriate answer to the oracle query, the simulator provides the adversary with the internal state of any corrupted servers involved in the query.

When the adversary queries the `Test` oracle, the simulator chooses (and records) a random bit  $b$ . When the adversary completes its execution and outputs a bit  $b'$ , the simulator can thus tell whether the adversary succeeds by checking whether (1) a single `Test` query was made regarding

some fresh session key  $\text{sk}_{U,U'}^i$ , and (2)  $b' = b$ . Success of the adversary is denoted by event **Succ**, and for any experiment  $P$  we define  $\text{Adv}_{\mathcal{A},P}(k) \stackrel{\text{def}}{=} 2 \cdot \Pr_{\mathcal{A},P}[\text{Succ}] - 1$ , where  $\Pr_{\mathcal{A},P}[\cdot]$  denotes the probability of an event when the simulator interacts with the adversary in accordance with experiment  $P$ . We refer to the original execution of the experiment (i.e., exactly according to the specification of the protocol) as  $P_0$ . We will introduce a sequence of transformations to the original experiment and bound the effect of each transformation on the adversary's advantage. We then bound the adversary's advantage in the final experiment; this yields a bound on the adversary's advantage in the original experiment.

Throughout the proof, we assume without loss of generality that for any client  $C$  associated with servers  $A$  and  $B$ , the adversary corrupts server  $A$ . This assumption is merely for notational convenience, and since the protocol is symmetric for the two servers it obviously makes no difference which of the two servers is the corrupted one.

**Experiment  $P'_0$ :** In experiment  $P'_0$ , the simulator interacts with the adversary as before except that the adversary does *not* succeed, and the experiment is aborted, if any of the following occur:

1. At any point, a  $\text{msg}_1$  generated by the simulator (whether in response to an **Execute** query or a **Send<sub>0</sub>** query) is repeated.
2. At any point, a  $\text{msg}_2$  generated by the simulator (whether in response to an **Execute** query or a **Send<sub>1</sub>** query) is repeated.
3. At any point, the adversary forges a new, valid message/signature pair for any verification key used in a simulator-generated  $\text{msg}_1$  (whether  $\text{msg}_1$  was generated in response to an **Execute** query or a **Send<sub>0</sub>** query).
4. At any point during the experiment, a collision occurs in the hash function  $H$  (regardless of whether this is due to a direct action of the adversary, or whether this occurs during the course of the simulator's response to an oracle query).

In addition, the simulator also changes how it computes the session key for any non-corrupted server  $B$ ; in particular, it will no longer use the long-term El Gamal secret key  $sk_B$  for any such  $B$ . Recall that this secret key is used only to compute  $X_b$  within the **Compute** protocol. Instead of computing  $X_b$  this way, the simulator simply computes  $X_b$  directly via:

$$X_b := g_1^{-(z_b+z'_a) \cdot pw_C} \cdot \left( A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a} K_b^{r_a} \right).$$

We stress that the simulator can do this precisely because we are dealing with a passive adversary (and so the simulator knows the internal state of all servers, even corrupted ones).

We claim that these changes have only a negligible effect on the adversary's advantage. It is immediate that event 1, above, occurs with only negligible probability. This is also clear for event 2, relying on the fact that we assume a passive adversary here. It is also clear that events 3 and 4 occur with only negligible probability assuming the security of  $(\text{Gen}, \text{Sign}, \text{Vrfy})$  as a one-time signature scheme and CRHF as a collision-resistant hash family. Finally, the change in computing  $X_b$  is merely a conceptual one since we assume a passive adversary (and so the value of  $X_b$  is identical in games  $P_0$  and  $P'_0$ ). Putting everything together, we have that  $\left| \text{Adv}_{\mathcal{A},P_0}(k) - \text{Adv}_{\mathcal{A},P'_0}(k) \right|$  is negligible.

**Experiment  $P''_0$ :** In this experiment, for any non-corrupted server  $B$  the pre-provisioned value  $\text{Com}'_{B,C}$  (for any client  $C$ ) is set to be an El Gamal encryption of a random value (with respect to the public key  $pk_B$ ). Similarly, the values  $M'_1$  used by any non-corrupted server  $B$  within

the `Compute` protocol are set to be El Gamal encryptions of random values (with respect to the public key  $pk_B$ ). Since, in experiment  $P'_0$ , the simulator does not use the secret key  $sk_B$  of any non-corrupted server  $B$ , it follows easily from the semantic security of El Gamal encryption with respect to the public keys of all non-corrupted servers (as well as a straightforward hybrid argument) that  $|\text{Adv}_{\mathcal{A},P'_0}(k) - \text{Adv}_{\mathcal{A},P''_0}(k)|$  is negligible.

In the experiments that follow, we distinguish between the following possibilities for a  $\text{msg}_1$  received by the gateway: (1) We say  $\text{msg}_1$  is *oracle-generated* if it was output by the simulator in response to an oracle query (i.e., either an `Execute` query or a `Send0` query); (2) we say  $\text{msg}_1$  is *adversarially-generated* otherwise. We also make the convention throughout the rest of the proof that any `Send3` queries made by the adversary contain a valid signature on the appropriate message as required by the protocol (since otherwise the gateway will simply reject the message and terminate the appropriate instances, and the adversary can determine by itself whether the signature is valid or not). Also, we assume that any components of adversarially-generated messages which are supposed to lie in  $\mathbb{G}$  actually do (as an instance will reject if this is not the case).

**Experiment  $P_1$ :** In experiment  $P_1$ , the simulator interacts with the adversary as in  $P''_0$  except for the way the simulator computes certain values in response to a  $\text{msg}_3$  being sent to the gateway when  $\text{msg}_1$  is oracle-generated (for the corresponding server-instances). (Note that, by definition, this is automatically the case for any `Execute` query being answered by the simulator.) In particular, if such a message is sent to instances  $\Pi_A^j, \Pi_B^j$  of servers  $A, B$  partnered with client  $C$  (i.e., either as a result of an `Execute`( $C, i, A, B, j$ ) query or as a result of a `Send3`( $A, B, j, \star$ ) query with  $\text{pid}_A^j = \text{pid}_B^j = C$ ), the simulator searches for the unique  $i$  such that  $\text{sid}_C^i = \text{sid}_A^j = \text{sid}_B^j$  (we comment below on the existence of such an  $i$ ). Recalling that server  $A$  is the one assumed to be compromised, the simulator then sets  $\text{sk}_{B,C}^j := \text{sk}_{C,B}^i$ . Furthermore,  $M'_2$  (in the `Compute` protocol) is computed as:

$$M'_2 \leftarrow \text{ElG}_{pk_A}(\text{sk}_{C,A}^i \cdot A_a^{-x_a} B_a^{-y_a} C_a^{-z_a} D_a^{-w_a} K_a^{-r_a} g_1^{z_a \cdot pw_{A,C}}).$$

Note that an  $i$  as desired above must exist, since otherwise the adversary has succeeded in forging a signature with respect to a verification key generated by the simulator and the simulator would then have aborted (cf. experiment  $P'_0$ ). Furthermore, this  $i$  must be unique or else an oracle-generated  $\text{msg}_1$  has repeated and the simulator would again have aborted (again, cf. experiment  $P'_0$ ).

We claim that these changes do not have any effect on the view of the adversary, keeping in mind that we assume the adversary is passive. This is clear for the case of  $\text{sk}_{B,C}^j$ , since in  $P''_0$  it is always the case that  $\text{sk}_{B,C}^j = \text{sk}_{C,B}^i$  when  $\Pi_B^j$  and  $\Pi_C^i$  are partnered. Furthermore, in both experiments  $P_1$  and  $P''_0$ , the ciphertext  $M'_2$  (sent to an instance  $\Pi_A^j$  of a corrupted server  $A$ , where this instance is partnered with  $\Pi_C^i$ ) is an encryption of the unique value  $v$  such that

$$\text{sk}_{A,C}^j \stackrel{\text{def}}{=} v \cdot (g_1^{-z_a \cdot pw_{A,C}} A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a} K_a^{r_a}) = \text{sk}_{C,A}^i.$$

Thus,  $\text{Adv}_{\mathcal{A},P_1}(k) = \text{Adv}_{\mathcal{A},P''_0}(k)$ .

We next introduce another conceptual change in the experiment by having the simulator choose public parameters  $h, c, d$  in such a way that it knows their representations with respect to  $g_1, g_2$ . In particular, when generating the public parameters the simulator now chooses  $g_1, g_2 \leftarrow \mathbb{G} \setminus \{1\}$  (as before) and then chooses  $\kappa \leftarrow \mathbb{Z}_q^*$  and  $(\chi_1, \chi_2), (\xi_1, \xi_2) \leftarrow \{(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q \mid g_1^x g_2^y \neq 1\}$ . It then sets

$$h = g_1^\kappa, \quad c = g_1^{\chi_1} g_2^{\chi_2}, \quad d = g_1^{\xi_1} g_2^{\xi_2}.$$

It is clear that the resulting distribution on the public parameters is unchanged from the previous experiment, and so this has no effect on the adversary's view.

With the above in place, we now distinguish three types of adversarially-generated  $\text{msg}_1$ , where this message has the form  $\text{msg}_1 = \left\langle \text{Client}, \text{VK}, \begin{matrix} A_a, B_a, C_a, D_a \\ A_b, B_b, C_b, D_b \end{matrix} \right\rangle$ :

- If  $\text{VK}$  is identical to a verification key used in an oracle-generated  $\text{msg}_1$  then we say  $\text{msg}_1$  *re-uses an oracle-generated verification key*.

Say  $(\text{Client}, \text{VK}, A, B, C, D)$  is a *correct encryption* if  $A^{x_1 + \alpha \xi_1} B^{x_2 + \alpha \xi_2} = D$  and  $C/A^\kappa = g_1^{pw_{\text{Client}}}$ , where  $\alpha \stackrel{\text{def}}{=} H(\text{Client}|\text{VK}|A|B|C)$ ; otherwise, it is an *incorrect encryption*. Then:

- We say  $\text{msg}_1$  is *valid* if either  $(\text{Client}, \text{VK}, A_a, B_a, C_a, D_a)$  or  $(\text{Client}, \text{VK}, A_b, B_b, C_b, D_b)$  is a correct encryption, and  $\text{msg}_1$  does not re-use an oracle-generated verification key.
- We say  $\text{msg}_1$  is *invalid* if *neither*  $(\text{Client}, \text{VK}, A_a, B_a, C_a, D_a)$  or  $(\text{Client}, \text{VK}, A_b, B_b, C_b, D_b)$  are correct encryptions, and  $\text{msg}_1$  does not re-use an oracle-generated verification key.

Note that the simulator can always tell whether a given  $\text{msg}_1$  re-uses an oracle-generated verification key and, if not, can efficiently determine whether a given  $\text{msg}_1$  is valid or invalid due to the way the public parameters are now generated.

**Experiment  $P_2$ :** In this experiment, the simulator changes the way it responds to a  $\text{msg}_3$  sent to the gateway when the  $\text{msg}_1$  for the corresponding server-instances is adversarially-generated and invalid (note that this can only occur as a result of a  $\text{Send}_3$  query following a  $\text{Send}_1$  query to the same server-instances). In particular, assume such a message is sent to instances  $\Pi_A^j$  and  $\Pi_B^j$  of servers  $A$  and  $B$  (i.e., via a query  $\text{Send}_3(A, B, j, \star)$ ), and recall that server  $A$  is assumed to be corrupted. In experiment  $P_2$ , the session key  $\text{sk}_{B,C}^j$  is now chosen uniformly at random from  $\mathbb{G}$ . Furthermore,  $M_2'$  (in the  $\text{Compute}$  protocol) is computed as an encryption (with respect to public key  $pk_A$ ) of a uniformly-random element of  $\mathbb{G}$ . We claim that:

**Claim 1**  $\text{Adv}_{\mathcal{A}, P_2}(k) = \text{Adv}_{\mathcal{A}, P_1}(k)$ .

We prove this by showing that the distributions on the adversary's view in the two experiments are identical. Let us focus first on the case of  $\text{sk}_{B,C}^j$ . Say

$$\text{msg}_1 = \left\langle \text{Client}, \text{VK}, \begin{matrix} A_a, B_a, C_a, D_a \\ A_b, B_b, C_b, D_b \end{matrix} \right\rangle,$$

where  $\text{msg}_1$  is the first message sent to the server-instances under consideration. Since  $\text{msg}_1$  is invalid, we know that  $(\text{Client}, \text{VK}, A_b, B_b, C_b, D_b)$  is an incorrect encryption. So, it must be the case that either  $A_b^{x_1 + \alpha_b \xi_1} B_b^{x_2 + \alpha_b \xi_2} \neq D_b$  or else  $C_b/pw_{\text{Client}} \neq A_b^\kappa$  (or possibly both). For any  $\mu, \nu \in \mathbb{G}$  and fixing the randomness used in the rest of experiment  $P_1$ , the probability over choice of  $x_b, y_b, z_b, w_b$  that  $E_{b,2} = \mu$  and  $\text{sk}_{B,C}^j = \nu$  is exactly the probability that

$$\log \mu = x_b + y_b \cdot \log g_2 + z_b \cdot \log h + w_b \cdot \log(cd^{\alpha_b}) \quad (2)$$

and

$$\begin{aligned} \log \nu - (r_a + r_b) \cdot \log K_b = \\ (x_b + x'_a) \cdot \log A_b + (y_b + y'_a) \cdot \log B_b + (z_b + z'_a) \cdot \log(C_b/g_1^{pw_C}) + (w_b + w'_a) \cdot \log D_b, \end{aligned} \quad (3)$$

where all logarithms are with respect to the base  $g_1$  and  $pw_C = pw_{Client}$ . Note that  $x'_a, y'_a, z'_a, w'_a$  are independent of  $x_b, y_b, z_b, w_b$  since we assume a passive adversary. As in [28], then, it can be verified that Equations (2) and (3) are linearly independent and not identically zero, when viewed as equations over  $\mathbb{Z}_q$  in the variables  $x_b, y_b, z_b, w_b$ .<sup>9</sup> Thus, the desired probability is exactly  $1/q^2$  and hence the value of  $sk_{B,C}^j$  is uniformly distributed in  $\mathbb{G}$  independent of the rest of the experiment.

A similar argument holds for the values of  $M'_2$  and  $E_{a,2}$ , viewed as functions of the random variables  $x'_b, y'_b, z'_b, w'_b$  and using now the fact that  $(Client, VK, A_a, B_a, C_a, D_a)$  is an incorrect encryption. In particular,  $Com'_{A,C}$  is an El Gamal encryption (with respect to  $pk_A$ ) of  $g_1^{pw_{A,C}}$  and hence (in experiment  $P_1$ )  $M'_2$  is an El Gamal encryption of the value

$$g_1^{-z_a \cdot pw_{B,C}} K_a^{r_b} \cdot \left( A_a^{x'_b} B_a^{y'_b} (C_a / g_1^{pw_C})^{z'_b} D_a^{w'_b} \right).$$

The rest of the argument is exactly as above. This concludes the proof of the claim.  $\square$

**Experiment  $P_3$ :** Here, the simulator changes the way it responds to a  $msg_3$  being sent to the gateway when  $msg_1$  for the corresponding server-instance is adversarially-generated and valid. In this case, the simulator halts and the adversary succeeds. (In any other case, the adversary's success is determined as in the previous experiment.) Clearly,  $Adv_{\mathcal{A}, P_2}(k) \leq Adv_{\mathcal{A}, P_3}(k)$  since there are now more ways for the adversary to succeed.

**Experiment  $P_4$ :** Let us first summarize where things stand in experiment  $P_3$ . In that experiment, for any non-corrupted server  $B$ , the simulator does not use the value  $r_b$  (i.e., the randomness used to construct  $Com_{B,C}$ ) at any point during the experiment. In particular, for a given instance  $\Pi_B^j$  with partner  $C$ :

- If  $msg_1$  to this instance is oracle-generated, the simulator computes  $sk_{B,C}^j$  and  $M'_2$  as described in experiment  $P_1$ .
- If  $msg_1$  to this instance is adversarially-generated and re-uses an oracle-generated verification key, the simulator will abort if the adversary is then able to send a  $msg_3$  to this instance which contains a valid signature. (This is so because — by definition of adversarially-generated —  $msg_1$  is not identical to any  $msg'_1$  previously output by the simulator. Since the “message” signed in  $msg_3$  includes  $msg_1$ , a valid signature would imply a signature forgery on the part of the adversary.)
- If  $msg_1$  to this instance is adversarially-generated and invalid, the simulator chooses  $sk_{B,C}^j$  and  $M'_2$  at random, as described in experiment  $P_2$ .
- Finally, if  $msg_1$  to this instance is adversarially-generated and valid, then the simulator halts and the adversary succeeds (cf. experiment  $P_3$ ).

In experiment  $P_4$ , for any non-corrupted server  $B$  and any client  $C$  the simulator sets  $Com_{B,C} \leftarrow \text{ElG}_{g_3}(g_1^{N+1-pw_{A,C}})$  (recall, the password dictionary is  $\{1, \dots, N\}$  by assumption). Note that this has the effect that the values  $(F, G)$  in any oracle-generated  $msg_2$  are now of the form

$$F = g_1^{r_a+r_b}, \quad G = g_3^{r_a+r_b} g_1^{N+1}$$

(relying on the assumption of a passive adversary); i.e.,  $(F, G)$  is an El Gamal encryption (with respect to the “public key”  $g_3$ ) of the value  $g_1^{N+1}$ . We remark that in experiment  $P_4$ , the simulator never uses  $pw_C$  or  $pw_{B,C}$  (for any client  $C$ ) in simulating actions of any non-corrupted server  $B$ .

<sup>9</sup>We use here the fact that  $M'_1$  is no longer an encryption of  $z_b$  (cf. experiment  $P_0''$ ) and hence the only information the adversary has about  $x_b, y_b, z_b, w_b$  comes from  $E_{b,2}$  and  $sk_{B,C}$ .

Since  $\text{Com}_{B,C}$  is an El Gamal encryption with respect to the “public key”  $g_3$  (i.e., an encryption of the value  $g_1^{pw_{B,C}}$  in experiment  $P_3$  and an encryption of the value  $g_1^{N+1-pw_{A,C}}$  in experiment  $P_4$ ), it follows readily from the semantic security of El Gamal encryption that  $|\text{Adv}_{\mathcal{A},P_4} - \text{Adv}_{\mathcal{A},P_3}|$  is negligible.

Before continuing, we introduce definitions for a  $\text{msg}_2$  received by a client which are analogous to those given previously for the case of  $\text{msg}_1$ . Namely, we say a  $\text{msg}_2$  is *oracle-generated* if it was output by the simulator in response to an oracle query (i.e., either an Execute query or a Send<sub>1</sub> query). We say a  $\text{msg}_2$  is *adversarially-generated* otherwise.

We also introduce another conceptual change in the experiment by now having the simulator choose public parameter  $g_3$  by first selecting random  $\lambda \leftarrow \mathbb{Z}_q^*$  and then setting  $g_3 = g_1^\lambda$ . It is clear that the resulting distribution on  $g_3$  is unchanged. With this in place, we can now define notions of validity/invalidity for  $\text{msg}_2$  in a way analogous to that done previously for the case of  $\text{msg}_1$ . Namely, for a given  $\text{msg}_2$  of the form  $\langle E_a, E_b, F, G \rangle$  sent to a client  $C$ , we say this  $\text{msg}_2$  is *valid* if  $G/F^\lambda = g_1^{pw_C}$  and *invalid* otherwise. Since the simulator now knows  $\log_{g_1} g_3$ , the simulator can efficiently determine whether a given  $\text{msg}_2$  is valid or not.

**Experiment  $P_5$ :** In experiment  $P_5$ , the simulator changes the way it responds to an invalid  $\text{msg}_2$  (note that any oracle-generated  $\text{msg}_2$  is automatically invalid; cf. experiment  $P_4$ ). Upon receiving an invalid  $\text{msg}_2$ , the simulator computes the response  $\text{msg}_3$  as in the previous experiment but then chooses keys  $\text{sk}_{C,A}$  and  $\text{sk}_{C,B}$  independently and uniformly at random from  $\mathbb{G}$ . We show that:

**Claim 2**  $\text{Adv}_{\mathcal{A},P_4}(k) = \text{Adv}_{\mathcal{A},P_5}(k)$ .

The proof of this claim follows that of Claim 1. We prove the claim by showing that the distributions on the adversary’s view in the two experiments are identical. Let  $\text{msg}_2 = \langle E_a, E_b, F, G \rangle$  denote an invalid  $\text{msg}_2$  sent to client  $C$ . By definition of invalidity, we have that  $G/F^\lambda \neq g_1^{pw_C}$ , where  $\lambda \stackrel{\text{def}}{=} \log_{g_1} g_3$ . Letting  $s = \log_{g_1} F$  and  $s' = \log_{g_1} (G/g_1^{pw_C})$ , this means that  $s' \neq \lambda s$ . For any  $\mu_1, \mu_2, \nu_1, \nu_2 \in \mathbb{G}$ , the probability over choice of  $x_1, y_1, x_2, y_2$  (fixing the randomness used in the remainder of the experiment) that  $K_a = \mu_1$ ,  $K_b = \mu_2$ ,  $\text{sk}_{C,A} = \nu_1$ , and  $\text{sk}_{C,B} = \nu_2$  is exactly the probability that the following linear equations in  $x_1, y_1, x_2, y_2$  (over  $\mathbb{Z}_q$ ) hold:

$$\begin{aligned} \log_{g_1} \mu_1 &= x_1 + \lambda \cdot y_1 \\ \log_{g_1} \mu_2 &= x_2 + \lambda \cdot y_2 \\ \log_{g_1} \nu_1 - r_1 \log_{g_1} E_a &= x_1 \cdot s + y_1 \cdot s' \\ \log_{g_1} \nu_2 - r_2 \log_{g_1} E_b &= x_2 \cdot s + y_2 \cdot s'. \end{aligned}$$

It is easy to see that these equations are linearly independent (given that  $s' \neq \lambda s$ ) and so the desired probability is  $1/q^4$ . Hence, as desired, the values of  $\text{sk}_{C,A}$  and  $\text{sk}_{C,B}$  are uniformly distributed, independent of each other as well as the rest of the experiment.

**Experiment  $P_6$ :** Paralleling the change made in experiment  $P_3$ , the simulator now changes the way it responds to a valid (adversarially-generated)  $\text{msg}_2$ . In particular, upon receiving such a message the simulator halts and the adversary succeeds. (In any other case, the adversary succeeds as in the previous experiment.) Clearly, this can only increase the adversary’s probability of success and so  $\text{Adv}_{\mathcal{A},P_5}(k) \leq \text{Adv}_{\mathcal{A},P_6}(k)$ .

**Experiment  $P_7$ :** In experiment  $P_6$ , the simulator does not use the values  $r_1, r_2$  (in simulating a client instance) other than to construct the Cramer-Shoup ciphertexts  $(A_a, \dots, D_a)$  and

$(A_b, \dots, D_b)$  which are encryptions of the correct password value  $g_1^{pw_C}$ . In experiment  $P_7$ , the simulator instead sets these ciphertexts to be encryptions of 1 (i.e., it sets  $C_a = h^{r_1}$  and  $C_b = h^{r_2}$ ).

The following bounds the effect this can have on the adversary's success probability:

**Claim 3** *Under the DDH assumption,  $|\text{Adv}_{\mathcal{A}, P_6}(k) - \text{Adv}_{\mathcal{A}, P_7}(k)|$  is negligible.*

The claim follows from the security of the Cramer-Shoup encryption scheme (with labels) under adaptive chosen-ciphertext attack. The proof follows that of [28], and we merely sketch an outline here. Given a public-key  $(g_1, g_2, h, c, d)$  for an instance of the Cramer-Shoup encryption scheme, as well as access to an encryption oracle and a decryption oracle, the simulator will first use the given values as the appropriate portion of the public parameters. The remaining public parameters are generated as in experiment  $P_6$ , and in particular the simulator will generate  $g_3$  so that it knows  $\log_{g_1} g_3$ . Every time the simulator responds to a request to initiate an interaction on behalf of a client  $C$  (i.e., as a result of an `Execute` query or a `Send0` query), it generates a verification key  $\text{VK}$  as before and then submits (twice) to its encryption oracle the query  $(\langle C, \text{VK} \rangle, g_1^{pw_C}, 1)$  and receives in return ciphertexts  $(A_a, B_a, C_a, D_a)$  and  $(A_b, B_b, C_b, D_b)$ . The simulator then uses these ciphertexts to construct  $\text{msg}_1$ . As we have already noted, the remainder of the client-side interaction can be simulated with no difficulty. As for the server-side interactions, here the simulator needs to be able to determine whether an adversarially-generated  $\text{msg}_1$  is valid or not. It can do this using its decryption oracle, with the only subtlety being to verify the claim that the simulator never needs to query the decryption oracle with a (label, ciphertext) pair it received from its encryption oracle. We remark that this is not quite as obvious in our case as in [28] because it is possible in our case for a  $\text{msg}_1$  to be adversarially-generated but for one of the component ciphertexts of this message to be equal to a component ciphertext of an oracle-generated  $\text{msg}_1$ . Yet, we can easily verify the claim about the simulator's access to the decryption oracle due to the following observations:

- If an adversarially-generated  $\text{msg}_1$  re-uses an oracle-generated verification key, the simulator will abort if the adversary is then able to send a  $\text{msg}_3$  containing a valid signature (because this will imply a signature forgery on the part of the adversary).
- Otherwise, the verification key  $\text{VK}$  used in  $\text{msg}_1$  is different from all oracle-generated verification keys and so, viewing  $\text{VK}$  as part of the ciphertext label, this means that the (label, ciphertext) pair submitted by the simulator to its decryption oracle is different from all (label, ciphertext) pairs received from its encryption oracle.

The proof concludes by noting that when the encryption oracle encrypts its left-most input, the adversary's view is identical to its view in experiment  $P_6$ , while in the case that the encryption oracle encrypts its right-most input, the adversary's view is identical to its view in experiment  $P_7$ .  $\square$

Observe that the adversary's view in experiment  $P_7$  is independent of the passwords chosen for the various clients except for the fact that the adversary learns whether adversarially-generated  $\text{msg}_1$  and  $\text{msg}_2$  are valid or not. In particular, although the adversary is given password shares  $pw_{A,C}$  for any corrupted server(s)  $A$  and any client(s)  $C$  associated with  $A$ , these shares are uniformly-distributed in  $\mathbb{Z}_q$  and therefore contain no information about the actual client password(s)  $pw_C$ . Furthermore, the simulator does not use  $pw_{B,C}$  at all (for any non-corrupted server  $B$ ), and does not use  $pw_C$  except to test whether adversarially-generated  $\text{msg}_1, \text{msg}_2$  are valid or not. Let  $\text{GuessPWD}$  denote the event that the adversary sends a valid  $\text{msg}_1$  or  $\text{msg}_2$ . Since the adversary can guess at most two passwords per adversarially-generated message (this is so since a  $\text{msg}_1$  contains two component ciphertexts which need not be encryptions of the same value), the probability of  $\text{GuessPWD}$  is at most  $2 \cdot Q(k)/N$ , where  $Q(k)$  is the number of on-line attacks made by  $\mathcal{A}$ .

Moreover, if `GuessPWD` does not occur then the adversary can succeed only by correctly guessing the value of  $b$  used by the `Test` oracle. In this case, however, all fresh session keys are uniformly-distributed in  $\mathbb{G}$  independent of the adversary’s view, and so the probability that the adversary can correctly guess  $b$  in this case is exactly  $1/2$ .

Putting everything together, we have:

$$\begin{aligned} \Pr_{\mathcal{A}, P_7}[\text{Succ}] &= \Pr[\text{GuessPWD}] + \frac{1}{2} \cdot (1 - \Pr[\text{GuessPWD}]) \\ &= \frac{1}{2} \cdot \Pr[\text{GuessPWD}] + \frac{1}{2} \\ &\leq \frac{Q(k)}{N} + \frac{1}{2}, \end{aligned}$$

and thus the adversary’s advantage in experiment  $P_7$  is at most  $2 \cdot Q(k)/N$ . The sequence of claims proven above show that

$$\text{Adv}_{\mathcal{A}, P_0}(k) \leq \text{Adv}_{\mathcal{A}, P_7}(k) + \varepsilon(k)$$

for some negligible function  $\varepsilon(\cdot)$ , and therefore the adversary’s advantage in  $P_0$  (i.e., the original protocol) is at most  $2 \cdot Q(k)/N$  plus some negligible quantity, as desired. ■

## 5 Handling Active Adversaries

Here, we describe the necessary changes to the protocol in order to handle active adversaries. We then sketch the appropriate modifications to the proof given in the previous section.

### 5.1 Overview of Changes to the Protocol

At a high level, the changes we make can be summarized as follows:

**Proofs of correctness.** We require servers to give *proofs of correctness* for their actions during the `Compute` protocol. We stress that we use only the fact that these are *proofs* (and not *proofs of knowledge*) and therefore we do not require any rewinding in our proof of security. This is crucial, as it enables us to handle concurrent executions of the protocol. Nevertheless, as part of the proofs of correctness we will have the servers encrypt certain values with respect to (additional) per-server public keys provisioned during protocol initialization. This will, in fact, enable extraction of certain values from the adversary during the security proof.

**Commitments to password shares.** The protocol as described in the previous section already assumes that each server is provisioned with appropriate El Gamal encryptions of the password share of the other server. We will use these shares (along with the proofs of correctness discussed earlier) to “force” a corrupted server to use the appropriate password share in its computations.

**Simulating proofs for non-corrupted servers.** During the course of the proof of security it will be necessary for non-corrupted servers to deviate from the prescribed actions of the protocol, yet these servers must give “valid” proofs of correctness to possibly corrupted servers. We cannot rely on “standard” use of zero-knowledge proofs in our setting, since (1) this would require rewinding which we explicitly want to avoid, and (2) potential malleability issues arise due to the fact that a corrupted server may be giving its proof of correctness at the same time a non-corrupted server is giving such a proof (this is so even if we force sequential executions of the proofs of correctness within any particular instance, since multiple instances may be simultaneously active). To enable simulatability we rely on techniques of MacKenzie [31] described in greater detail below.

## 5.2 Detailed Description of Changes to the Protocol

We first discuss the necessary modifications to the initialization phase (this is all in addition to the provisioned values already discussed in Section 4.1): (1) each server  $S$  is given a random triple  $\text{triple}_S = (U_{S,1}, U_{S,2}, U_{S,3})$  of elements chosen uniformly at random from  $\mathbb{G}$ . Furthermore, (2) if servers  $A$  and  $B$  are associated with the same client  $C$ , then  $B$  is given  $\text{triple}_A$  and  $A$  is given  $\text{triple}_B$ .

We will next describe the necessary changes to the protocol itself. In what follows, we will use witness-indistinguishable  $\Sigma$ -protocols (with negligible soundness error<sup>10</sup>) [12] of various predicates and it will be useful to develop some notation. If  $\Psi$  represents a predicate (defined over some public values), we let  $\Sigma[\Psi]$  denote a  $\Sigma$ -protocol for this predicate. If  $\Psi_1, \Psi_2$  are two predicates, then we let  $\Sigma[\Psi_1 \vee \Psi_2]$  denote a  $\Sigma$ -protocol for the “or” of these predicates. We remark that given  $\Sigma$ -protocols for  $\Psi_1$  and  $\Psi_2$ , it is easy to combine these so as to obtain a  $\Sigma$ -protocol for  $\Psi_1 \vee \Psi_2$  [13]. We define the predicate  $\text{DDH}_S$ , for any server  $S$  with  $\text{triple}_S = (U_1, U_2, U_3)$ , as follows:

$$\text{DDH}_S(U_1, U_2, U_3) \stackrel{\text{def}}{=} [\exists x, y \text{ s.t. } U_1 = g_1^x \wedge U_2 = g_1^y \wedge U_3 = g_1^{xy}];$$

i.e.,  $\text{DDH}_S$  denotes the predicate asserting that  $\text{triple}_S$  is a Diffie-Hellman triple.

Given the above, the protocol is modified in the following ways:

**Initial server message.** To construct the second message of the protocol (i.e.,  $\text{msg}_2$ ), the servers first construct  $E_{a,1}, E_{b,1}, E_{b,2}$ , and  $E_{a,2}$  as in Figure 2. The servers also choose random  $\text{nonce}_A, \text{nonce}_B \in \{0, 1\}^k$ . (These nonces are needed in the case of an active adversary to ensure that, with high probability, all  $\text{msg}_2$  are distinct.) Server  $A$  sends to the gateway the message  $\langle \text{nonce}_A, E_{a,1}, E_{b,1}, \text{Com}_{A,C}, \text{Com}_{B,C} \rangle$ . Similarly, server  $B$  sends to the gateway the message  $\langle \text{nonce}_B, E_{b,2}, E_{a,2}, \text{Com}_{A,C}, \text{Com}_{B,C} \rangle$ . The gateway verifies that  $\text{Com}_{A,C}$  and  $\text{Com}_{B,C}$  (as sent by the two servers) are identical: if so, it constructs the outgoing message as in Figure 2 but also including  $\text{nonce}_A, \text{nonce}_B$ ; if not, the gateway sends a special abort message to each of the servers. (The client will sign all of  $\text{msg}_2$  — including the nonces — exactly as in Figure 1, and this signature will be verified by the gateway as in Figure 2.)

**The Compute protocol.** After the gateway receives  $\text{msg}_3$  and verifies the signature as in Figure 2, it forwards the values  $E_{a,1}, E_{b,1}$  (resp.,  $E_{b,2}, E_{a,2}$ ) to server  $B$  (resp., server  $A$ ) in addition to forwarding  $(K_a, K_b)$  as before. The Compute protocol is then modified as follows (we describe the changes from the point of view of server  $A$ , but they are applied symmetrically to server  $B$ ): In the first phase, in addition to computing  $M_1 \leftarrow \text{EIG}_{pk_A}(g_1^{-z_a})$ , server  $A$  computes

$$v_{x_a, y_a, z_a, w_a} := A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a} \quad \text{and} \quad V_{x_a, y_a, z_a, w_a} \leftarrow \text{EIG}_{pk_A}(v_{x_a, y_a, z_a, w_a})$$

and sends these values to  $B$ . Define the predicate  $\Psi_1$  as follows:

$$\Psi_1 \stackrel{\text{def}}{=} \left[ \begin{array}{l} \exists x_a, y_a, z_a, w_a, r, \tilde{r} \text{ s.t.} \quad : \\ \begin{array}{l} E_{a,1} = g_1^{x_a} g_2^{y_a} h^{z_a} (cd^{\alpha_a})^{w_a} \\ M_1 = (g_1^r, pk_A^r \cdot g_1^{-z_a}) \\ V_{x_a, y_a, z_a, w_a} = (g_1^{\tilde{r}}, pk_A^{\tilde{r}} \cdot A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a}) \end{array} \end{array} \right].$$

Server  $A$  then acts as a prover in the protocol  $\Sigma[\Psi_1 \vee \text{DDH}_A]$ . Meanwhile,  $A$  acts as a verifier in the symmetric  $\Sigma$ -protocol being given (possibly concurrently) by server  $B$ . If  $B$ 's proof fails, then  $A$  aborts immediately.

<sup>10</sup>From now on, “ $\Sigma$ -protocol” means a witness-indistinguishable  $\Sigma$ -protocol with negligible soundness error.

In the second phase of the Compute protocol, in addition to computing  $M_2$  as in Figure 3, server  $A$  also computes

$$v_{z'_a} := g_1^{z'_a} \quad v_{x'_a, y'_a, z'_a, w'_a} := A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a}$$

$$V_{z'_a} \leftarrow \text{ElG}_{pk_A}(v_{z'_a}) \quad V_{x'_a, y'_a, z'_a, w'_a} \leftarrow \text{ElG}_{pk_A}(v_{x'_a, y'_a, z'_a, w'_a})$$

and sends these values to  $B$ . Define the predicate  $\Psi_2$  as follows:

$$\Psi_2 \stackrel{\text{def}}{=} \left[ \begin{array}{l} \exists x'_a, y'_a, z'_a, w'_a, \\ r_a, pw_{A,C}, r, \tilde{r}, \hat{r} \text{ s.t.} \\ \quad : \quad \begin{array}{l} E_{b,1} = g_1^{x'_a} g_2^{y'_a} h^{z'_a} (cd^{\alpha_b})^{w'_a} \\ V_{z'_a} = (g_1^r, pk_A^r \cdot g_1^{z'_a}) \\ V_{x'_a, y'_a, z'_a, w'_a} = (g_1^{\tilde{r}}, pk_A^{\tilde{r}} \cdot A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a}) \\ M_2 = (M'_1)^{pw_{A,C}} \times (\text{Com}'_{B,C})^{-z'_a} \\ \quad \times (g_1^{\hat{r}}, pk_B^{\hat{r}} \cdot g_1^{-z'_a \cdot pw_{A,C}} A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a} K_b^{r_a}) \\ \text{Com}_{A,C} = (g_1^{r_a}, g_3^{r_a} g_1^{pw_{A,C}}) \end{array} \end{array} \right].$$

Server  $A$  then acts as a prover in the protocol  $\Sigma[\Psi_2 \vee \text{DDH}_A]$ . Meanwhile,  $A$  acts as a verifier in the symmetric  $\Sigma$ -protocol being given (possibly concurrently) by server  $B$ . If  $B$ 's proof fails, then  $A$  aborts without computing a session key.

Relatively efficient  $\Sigma$ -protocols for the above predicates can be constructed using standard techniques (see, e.g., [31]), and so we omit further details.

**Efficiency.** Compared to the basic protocol, the computational work of the client is unchanged. As for the servers, assuming we instantiate the protocol using  $\Sigma$  protocols as in [31] we calculate that each server must perform an additional 56 exponentiations as compared to the basic protocol, for a total of 67.5 exponentiations. (I.e., each server's work is increased by a factor of roughly 6 as compared to the basic protocol.) Once again, we remark that this does not take into account any potential efficiency improvements such as off-line computation or pre-computation that could be done to speed up fixed-base exponentiations. We also note, by way of comparison, that the protocol of Di Raimondo and Gennaro [16] requires each server to perform roughly 80 exponentiations<sup>11</sup> for the smallest threshold supported by their scheme (i.e.,  $n = 4, t = 1$ ). Of course, it is not surprising that our protocol is (slightly) more efficient, since we have fewer servers. The point is merely to illustrate that achieving security against active adversaries in this setting (without random oracles) is computationally expensive.

### 5.3 Proof of Security for Active Adversaries

We prove that the modified protocol described above is secure against active adversaries; that is:

**Theorem 2** *With the modifications described above and under the same assumptions as in Theorem 1, we obtain a secure two-server protocol for password-only authenticated key exchange in the presence of an active adversary.*

In the proof of the above theorem, we only show the differences from the proof of Theorem 1. In the following, we let  $\text{ElG}_{sk}^{-1}(M) \stackrel{\text{def}}{=} \frac{M[2]}{M[1]^{sk}}$  for any  $sk \in \mathbb{Z}_q$  and El Gamal ciphertext  $M$ .

<sup>11</sup>The authors of [16] do not calculate the computational cost of their protocol in the paper.

**Proof** Given an active adversary  $\mathcal{A}$  attacking the protocol, we imagine a simulator which runs the protocol for  $\mathcal{A}$  as in the proof of Theorem 1.  $\text{Adv}_{\mathcal{A},P}(k)$  is defined as there, and we again refer to the original experiment as  $P_0$ . Throughout the proof, we will again assume without loss of generality that for any client  $C$  associated with servers  $A$  and  $B$ , the adversary corrupts server  $A$ .

**Experiment  $P'_0$ :** In experiment  $P'_0$ , the simulator makes the following changes: First, as in the proof of Theorem 1, the experiment is aborted and the adversary does not succeed if any of the following occur:

1. At any point, a  $\text{msg}_1$  generated by the simulator repeats.
2. At any point, a  $\text{msg}_2$  generated by the simulator repeats.
3. At any point, the adversary forges a new, valid message/signature pair for any verification key used in a simulator-generated  $\text{msg}_1$ .
4. At any point during the experiment, a collision occurs in the hash function  $H$ .

In addition, for any non-corrupted server  $B$  the simulator sets  $\text{triple}_B$  to be a (random) Diffie-Hellman triple, while for any corrupted server  $A$  the simulator sets  $\text{triple}_A$  to be a (random) *non*-Diffie-Hellman triple. Finally, the simulator also changes how it computes the session key for any non-corrupted server  $B$ ; in particular, it will no longer use the long-term El Gamal secret key  $sk_B$  to compute  $X_b$ . Instead, the simulator computes  $X_b$  in the following way: using  $sk_A$ , the simulator decrypts  $V_{z'_a}$  and  $V_{x'_a, y'_a, z'_a, w'_a}$  to obtain values  $v_{z'_a}$  and  $v_{x'_a, y'_a, z'_a, w'_a}$ , respectively. It then sets:

$$X_b := g_1^{-z_b \cdot pw_C} \cdot (v_{z'_a})^{-pw_C} \cdot K_b^{r_a} \cdot v_{x'_a, y'_a, z'_a, w'_a}$$

(recall that the simulator knows  $pw_C$  and  $r_a$ ).

We claim that these changes have only a negligible effect on the adversary's advantage. It is immediate that the four events listed above occur with only negligible probability (for the case of event 2, this relies on the fact that  $\text{nonce}_B$  is chosen at random and included in  $\text{msg}_2$ ). Also, the DDH assumption implies that the adversary cannot distinguish the change in the way  $\text{triple}_S$  is computed for the corrupted and non-corrupted servers. It remains only to argue that the change in computing  $X_b$  has only negligible effect. Because of the  $\Sigma$ -protocols (in which  $A$  acts as a prover) during the Compute protocol, we know that with all but negligible probability there exist values  $x'_a, y'_a, z'_a, w'_a$  such that

$$\text{ElG}_{sk_B}^{-1}(M_2) = g_1^{-z_b \cdot pw_{A,C}} \cdot g_1^{-z'_a \cdot pw_{B,C}} \cdot g_1^{-z'_a \cdot pw_{A,C}} \cdot A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a} K_a^{r_a}$$

and

$$v_{z'_a} = g_1^{z'_a} \quad \text{and} \quad v_{x'_a, y'_a, z'_a, w'_a} = A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a},$$

for values  $pw_{A,C}, pw_{B,C}, z_b, r_a$  chosen (and known) by the simulator. It follows that, with all but negligible probability, the values of  $X_b$  computed in experiments  $P_0$  and  $P'_0$  are always identical. Putting everything together, we have that  $\left| \text{Adv}_{\mathcal{A},P_0}(k) - \text{Adv}_{\mathcal{A},P'_0}(k) \right|$  is negligible.

**Experiment  $P''_0$ :** In this experiment, all  $\Sigma$ -protocols in which a non-corrupted server  $B$  acts as a prover are performed using a witness for the predicate  $\text{DDH}_B$ . (Note that this is possible since, in the previous experiment, we ensured that  $\text{triple}_B$  was indeed a DDH triple for any non-corrupted server  $B$ .) It follows immediately from the witness indistinguishability of the  $\Sigma$ -protocols that  $\left| \text{Adv}_{\mathcal{A},P'_0}(k) - \text{Adv}_{\mathcal{A},P''_0}(k) \right|$  is negligible.

**Experiment  $P_0'''$ :** Here, all ciphertexts throughout the course of the experiment that are computed by the simulator as El Gamal encryptions of some value with respect to  $pk_B$  (for a non-corrupted server  $B$ ) are now formed by encrypting a random group element. These ciphertexts include the values  $\text{Com}'_{B,C}$  (for any client  $C$ ) constructed during the initialization phase, as well as  $M'_1$  (generated during the Compute protocol) and all ciphertexts computed by server  $B$  as part of the  $\Sigma$ -protocols in which  $B$  acts as a prover.

Since neither  $sk_B$  nor the randomness used to generate any of the ciphertexts of the type considered here are ever used by the simulator in experiment  $P_0''$  (we rely here on the fact that, in experiment  $P_0''$ , the simulator uses witness for  $\text{DDH}_B$  when acting as a prover), semantic security of El Gamal encryption implies that  $\left| \text{Adv}_{\mathcal{A}, P_0''}(k) - \text{Adv}_{\mathcal{A}, P_0'''}(k) \right|$  is negligible.

In the experiments that follow, we use the same definitions as in the proof of Theorem 1 for *oracle-generated* and *adversarially-generated* messages  $\text{msg}_1$  received by the gateway.

**Experiment  $P_1$ :** In experiment  $P_1$ , the simulator interacts with the adversary as in  $P_0'''$  except for the way the simulator computes certain values in response to a  $\text{msg}_3$  sent to the gateway when  $\text{msg}_1$  is oracle-generated (for the corresponding server-instances). (Note that, by definition, this is automatically the case for any Execute query being answered by the simulator.) In particular, if such a message is sent to instances  $\Pi_A^j, \Pi_B^j$  of servers  $A, B$  partnered with client  $C$ , the simulator finds the unique  $i$  such that  $\text{sid}_C^i = \text{sid}_A^j = \text{sid}_B^j$  (as in the proof of Theorem 1, a unique such  $i$  exists). Recalling that server  $A$  is the one assumed to be compromised, the simulator then sets  $\text{sk}_{B,C}^j := \text{sk}_{C,B}^i$  (assuming instance  $\Pi_B^j$  does not abort due to an incorrect proof by server  $A$ ). Furthermore, to calculate  $M'_2$  in the Compute protocol, the simulator proceeds as follows (again, assuming  $\Pi_B^j$  does not abort due to an incorrect proof by server  $A$ ): using  $sk_A$ , the simulator decrypts  $M_1$  and  $V_{x_a, y_a, z_a, w_a}$  to obtain  $v_{z_a}$  and  $v_{x_a, y_a, z_a, w_a}$ , respectively. It then sets

$$M'_2 \leftarrow \text{ElG}_{pk_A} \left( \text{sk}_{C,A}^i \cdot (v_{x_a, y_a, z_a, w_a})^{-1} \cdot K_a^{-r_a} \cdot (v_{z_a})^{-pw_{A,C}} \right).$$

We claim that these changes have only negligible effect on the adversary's advantage. In particular, due to the  $\Sigma$ -protocols (in which  $A$  acts as a prover) we know that with all but negligible probability there exist values  $x'_a, y'_a, z'_a, w'_a$  such that

$$E_{b,1} = g_1^{x'_a} g_2^{y'_a} h^{z'_a} (cd^{\alpha_b})^{w'_a} \quad \text{and} \quad v_{z'_a} = g_1^{z'_a} \quad \text{and} \quad v_{x'_a, y'_a, z'_a, w'_a} = A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a}$$

as well as values  $x_a, y_a, z_a, w_a$  such that

$$E_{a,1} = g_1^{x_a} g_2^{y_a} h^{z_a} (cd^{\alpha_a})^{w_a} \quad \text{and} \quad v_{z_a} = g_1^{-z_a} \quad \text{and} \quad v_{x_a, y_a, z_a, w_a} = A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a}.$$

Assuming this to be the case, consider the computation of  $sk_{B,C}$ . In experiment  $P_0'''$  we have:

$$\begin{aligned} \text{sk}_{B,C}^j &= A_b^{x_b} B_b^{y_b} C_b^{z_b} D_b^{w_b} K_b^{r_b} \cdot g_1^{-z_b \cdot pw_C} \cdot (v_{z'_a})^{-pw_C} \cdot K_b^{r_a} \cdot v_{x'_a, y'_a, z'_a, w'_a} \\ &= A_b^{x_b+x'_a} B_b^{y_b+y'_a} (C_b/g_1^{pw_C})^{z_b+z'_a} D_b^{w_b+w'_a} K_b^{r_a+r_b}. \end{aligned}$$

This is exactly equal to  $\text{sk}_{C,B}^i$  (for  $i$  as defined above), and hence the values of  $\text{sk}_{B,C}^j$  computed in experiments  $P_0'''$  and  $P_1$  are identical. Considering next the case of  $M'_2$ , in experiment  $P_0'''$  this is a random El Gamal encryption (with respect to public key  $pk_A$ ) of:

$$g_1^{-z_a \cdot pw_{B,C}} \cdot g_1^{-z'_a \cdot pw_C} \cdot A_a^{x'_a} B_a^{y'_a} C_a^{z'_a} D_a^{w'_a} K_a^{r_b} = \text{sk}_{C,A}^i \cdot (v_{x_a, y_a, z_a, w_a})^{-1} \cdot K_a^{-r_a} \cdot (v_{z_a})^{-pw_{A,C}},$$

for  $i$  as defined above. Thus,  $M'_2$  is distributed identically in experiments  $P_0'''$  and  $P_1$ . Putting everything together,  $|\text{Adv}_{\mathcal{A},P_1}(k) - \text{Adv}_{\mathcal{A},P_0'''}(k)|$  is negligible.

Exactly as in the proof of Theorem 1, we next have the simulator choose public parameters  $h, c, d$  in such a way that it knows their representations with respect to  $g_1, g_2$ . We omit the details here. Once this is done, the simulator can again distinguish between incorrect and correct encryptions, and can thus distinguish between adversarially-generated  $\text{msg}_1$  which are valid or invalid; all these terms are defined exactly as in the previous proof. We also recall from that proof the notion of an adversarially-generated  $\text{msg}_1$  which re-uses an oracle-generated verification key.

**Experiment  $P_2$ :** In this experiment, the simulator changes the way it responds to a  $\text{msg}_3$  sent to the gateway when the  $\text{msg}_1$  for the corresponding server-instances is adversarially-generated and invalid. In particular, assume such a message is sent to instances  $\Pi_A^j$  and  $\Pi_B^j$  of servers  $A$  and  $B$ , and recall that server  $A$  is assumed to be corrupted. In experiment  $P_2$ , the session key  $\text{sk}_{B,C}^j$  is now chosen uniformly at random from  $\mathbb{G}$ . Furthermore,  $M'_2$  (in the Compute protocol) is computed as an encryption (with respect to public key  $pk_A$ ) of a uniformly-random element of  $\mathbb{G}$ .

We prove that  $\text{Adv}_{\mathcal{A},P_2}(k) = \text{Adv}_{\mathcal{A},P_1}(k)$ . by showing that the distributions on the adversary's view in the two experiments are identical. Let us focus first on the case of  $\text{sk}_{B,C}^j$ . Say

$$\text{msg}_1 = \left\langle \text{Client}, \text{VK}, \begin{array}{l} A_a, B_a, C_a, D_a \\ A_b, B_b, C_b, D_b \end{array} \right\rangle,$$

where  $\text{msg}_1$  is the first message sent to the server-instances under consideration. Since  $\text{msg}_1$  is invalid, we know that  $(\text{Client}, \text{VK}, A_b, B_b, C_b, D_b)$  is an incorrect encryption. So, it must be the case that either  $A_b^{x_1 + \alpha_b \xi_1} B_b^{x_2 + \alpha_b \xi_2} \neq D_b$  or else  $C_b/pw_{\text{Client}} \neq A_b^c$  (or possibly both). For any  $\mu, \nu \in \mathbb{G}$  and fixing the randomness used in the rest of experiment  $P_1$ , the probability over choice of  $x_b, y_b, z_b, w_b$  that  $E_{b,2} = \mu$  and  $\text{sk}_{B,C}^j = \nu$  is exactly the probability that

$$\log \mu = x_b + y_b \cdot \log g_2 + z_b \cdot \log h + w_b \cdot \log(cd^{\alpha_b}) \quad (4)$$

and

$$\begin{aligned} \log \nu - (r_a + r_b) \cdot \log K_b - \log v_{x'_a, y'_a, z'_a, w'_a} + pw_C \cdot \log v_{z'_a} = \\ x_b \cdot \log A_b + y_b \cdot \log B_b + z_b \cdot \log(C_b/g_1^{pw_C}) + w_b \cdot \log D_b, \end{aligned} \quad (5)$$

where all logarithms are with respect to the base  $g_1$  and  $pw_C = pw_{\text{Client}}$ . As in the proof of Theorem 1, it can be verified that Equations (4) and (5) are linearly independent and not identically zero, when viewed as equations over  $\mathbb{Z}_q$  in the variables  $x_b, y_b, z_b, w_b$ . Thus, the desired probability is exactly  $1/q^2$  and hence the value of  $\text{sk}_{B,C}^j$  is uniformly distributed in  $\mathbb{G}$  independent of the rest of the experiment.

A similar argument holds for the values of  $M'_2$  and  $E_{a,2}$ , viewed as functions of the random variables  $x'_b, y'_b, z'_b, w'_b$  and using now the fact that  $(\text{Client}, \text{VK}, A_a, B_a, C_a, D_a)$  is an incorrect encryption. In particular,  $\text{Com}'_{A,C}$  is an El Gamal encryption (with respect to  $pk_A$ ) of  $g_1^{pw_{A,C}}$  and hence the value  $M'_2$  in experiment  $P_1$  is an El Gamal encryption of the value

$$\left( \text{ElG}_{sk_A}^{-1}(M_1) \right)^{pw_{B,C}} \cdot K_a^{r_b} \cdot A_a^{x'_b} B_a^{y'_b} (C_a/g_1^{pw_C})^{z'_b} D_a^{w'_b}.$$

The rest of the argument is exactly as above.

The remainder of the proof is, verbatim, the same as the proof of Theorem 1. ■

## References

- [1] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. *Adv. in Cryptology — Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 139–155, 2000.
- [2] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proc. 1st ACM Conference on Computer and Communications Security*, ACM, pp. 62–73, 1993.
- [3] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Adv. in Cryptology — Crypto 1993*, LNCS vol. 773, Springer-Verlag, pp. 232–249, 1994.
- [4] M. Bellare and P. Rogaway. Provably-Secure Session Key Distribution: the Three Party Case. *27th ACM Symposium on Theory of Computing (STOC)*, ACM, pp. 57–66, 1995.
- [5] S.M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. *IEEE Symposium on Research in Security and Privacy*, IEEE, pp. 72–84, 1992.
- [6] S.M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise. *1st ACM Conf. on Computer and Comm. Security*, ACM, pp. 244–250, 1993.
- [7] M. Boyarsky. Public-Key Cryptography and Password Protocols: The Multi-User Case. *7th Ann. Conf. on Computer and Comm. Security*, ACM, pp. 63–72, 1999.
- [8] V. Boyko, P. MacKenzie, and S. Patel. Provably-Secure Password-Authenticated Key Exchange Using Diffie-Hellman. *Adv. in Cryptology — Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 156–171, 2000.
- [9] J. Brainard, A. Juels, B. Kaliski, and M. Szydło. Nightingale: A New Two-Server Approach for Authentication with Short Secrets. *12th USENIX Security Symp.*, pp. 201–213, 2003.
- [10] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *J. ACM* 51(4): 557–594, 2004.
- [11] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally-Composable Password Authenticated Key Exchange. *Adv. in Cryptology — Eurocrypt 2005*, LNCS vol. 3494, Springer-Verlag, pp. 404–421, 2005.
- [12] R. Cramer. Modular Design of Secure Yet Practical Cryptographic Protocols. PhD Thesis, CWI and University of Amsterdam, 1996.
- [13] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. *Adv. in Cryptology — Crypto 1994*, LNCS vol. 839, Springer-Verlag, pp. 174–187, 1994.
- [14] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Chosen Ciphertext Attack. *Adv. in Cryptology — Crypto 1998*, LNCS vol. 1462, Springer-Verlag, pp. 13–25, 1998.

- [15] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory* 22(6): 644–654, 1976.
- [16] M. Di Raimondo and R. Gennaro. Provably Secure Threshold Password-Authenticated Key Exchange. *J. Computer and System Sciences* 72(6): 978–1001 (2006). Preliminary version in Eurocrypt 2003.
- [17] Y. Dodis, M. Krohn, D. Mazieres, and A. Nicolosi. Proactive Two-Party Signatures for User Authentication. NDSS 2003.
- [18] T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* 31: 469–472, 1985.
- [19] W. Ford and B.S. Kaliski. Server-Assisted Generation of a Strong Secret from a Password. *Proc. 5th IEEE Intl. Workshop on Enterprise Security*, 2000.
- [20] R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *ACM Trans. Information and System Security* 9(2): 181–234 (2006). Preliminary version in Eurocrypt 2003.
- [21] N. Gilboa. Two-Party RSA Key Generation. *Adv. in Cryptology — Crypto 1999*, LNCS vol. 1666, Springer-Verlag, pp. 116–129, 1999.
- [22] O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. *J. Cryptology* 19(3): 241–340 (2006). Preliminary version in Crypto 2001.
- [23] L. Gong, T.M.A. Lomas, R.M. Needham, and J.H. Saltzer. Protecting Poorly-Chosen Secrets from Guessing Attacks. *IEEE J. on Selected Areas in Communications* 11(5): 648–656, 1993.
- [24] S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Trans. Information and System Security* 2(3): 230–268, 1999.
- [25] D. Jablon. Strong Password-Only Authenticated Key Exchange. *ACM Computer Communications Review* 26(5): 5–20, 1996.
- [26] D. Jablon. Password Authentication Using Multiple Servers. *RSA Cryptographers’ Track 2001*, LNCS vol. 2020, Springer-Verlag, pp. 344–360, 2001.
- [27] S. Jiang and G. Gong. Password Based Key Exchange With Mutual Authentication. *Workshop on Selected Areas of Cryptography (SAC)*, 2004.
- [28] J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. *Adv. in Cryptology — Eurocrypt 2001*, LNCS vol. 2045, Springer-Verlag, pp. 475–494, 2001.
- [29] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham. Reducing Risks from Poorly-Chosen Keys. *ACM Operating Systems Review* 23(5): 14–18, 1989.
- [30] S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. *Proc. of the Security Protocols Workshop*, LNCS 1361, Springer-Verlag, pp. 79–90, 1997.
- [31] P. MacKenzie. An Efficient Two-Party Public-Key Cryptosystem Secure against Adaptive Chosen-Ciphertext Attack. *Public Key Cryptography (PKC) 2003*, LNCS vol. 2567, Springer-Verlag, pp. 47–61, 2003.

- [32] P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. *Adv. in Cryptology — Asiacrypt 2000*, LNCS 1976, Springer-Verlag, pp. 599–613, 2000.
- [33] P. MacKenzie and M. Reiter. Networked Cryptographic Devices Resilient to Capture. *Intl. J. Information Security* 2(1): 1–20 (2003). Preliminary version in IEEE Security and Privacy 2001.
- [34] P. MacKenzie and M. Reiter. Two-Party Generation of DSA Signatures. *Intl. J. Information Security* 2(3–4): 218–239 (2004). Preliminary version in Crypto 2001.
- [35] P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold Password-Authenticated Key Exchange. *J. Cryptology* 19(1): 27–66 (2006). Preliminary version in Crypto 2002.
- [36] V. Shoup. A Proposal for an ISO Standard for Public-Key Encryption, version 2.1. Draft, 2001. Available at <http://eprint.iacr.org/2001/112>.
- [37] M. Szydło and B. Kaliski. Proofs for Two-Server Password Authentication. *RSA Cryptographers' Track 2005*, LNCS vol. 3376, Springer-Verlag, pp. 227–244, 2005.
- [38] T. Wu. The Secure Remote Password Protocol. *Proc. Internet Society Symp. on Network and Distributed System Security*, pp. 97–111, 1998.