Threshold Password-Authenticated Key Exchange

(Extended Abstract)

Philip MacKenzie[1], Thomas Shrimpton[2], and Markus Jakobsson[3]

[1] Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974 USA
philmac@lucent.com

[2] Dept. of Electrical and Computer Engineering
UC Davis
Davis, CA 95616 USA
teshrim@ucdavis.edu

[3] RSA Laboratories
RSA Security, Inc.
Bedford, MA 01730 USA
mjakobsson@rsasecurity.com

Abstract. In most password-authenticated key exchange systems there is a single server storing password verification data. To provide some resilience against server compromise, this data typically takes the form of a one-way function of the password (and possibly a salt, or other public values), rather than the password itself. However, if the server is compromised, this password verification data can be used to perform an offline dictionary attack on the user's password. In this paper we propose an efficient password-authenticated key exchange system involving a set of servers, in which a certain threshold of servers must participate in the authentication of a user, and in which the compromise of any fewer than that threshold of servers does not allow an attacker to perform an offline dictionary attack. We prove our system is secure in the random oracle model under the Decision Diffie-Hellman assumption against an attacker that may eavesdrop on, insert, delete, or modify messages between the user and servers, and that compromises fewer than that threshold of servers.

1 Introduction

Many real-world systems today rely on password authentication to verify the identity of a user before allowing that user to perform certain functions, such as setting up a virtual private network or downloading secret information. There are many security concerns associated with password authentication, due mainly to the fact that most users' passwords are drawn from a relatively small and easily generated dictionary. Thus if information sufficient to verify a password guess is leaked, the password may be found by performing an offline dictionary attack: one can run through a dictionary of possible passwords, testing each one against the leaked information in order to determine the correct password.

When password authentication is performed over a network, one must be especially careful not to allow any leakage of information to one listening in, or even actively attacking, the network. If one assumes the server's public key is known (or at least can be verified) by the user, then performing password authentication after setting up an anonymous secure channel to the server is generally sufficient to prevent leakage of information, as is done in SSH [28] or on the web using SSL [13]. The problem becomes more difficult if the server's public key cannot be verified by the user. Solutions to this problem have been coined *strong password authentication protocols*, and have the property that (informally) the probability of an active attacker (i.e., one that may eavesdrop on, insert, delete, or modify messages on a network) impersonating a user is only negligibly better than a simple on-line guessing attack, consisting of the attacker iteratively guessing passwords and running the authentication protocol. Strong password authentication protocols were proposed by Jablon [23] and Wu [29], among others. Recently, some protocols were proven secure in the random oracle model[1] (Bellare *et al.* [1], Boyko *et al.* [8] and MacKenzie *et al.* [26]), in the public random string model (Katz *et al.* [25]), and in the standard model (Goldreich and Lindell [21]). However, all of these protocols, even the ones in which the server's public key is known to the user, are vulnerable to server compromise in the sense that compromising the server would allow an attacker to obtain the password verification data on that server (typically some type of one-way function of the password and some public values). This could then be used to perform an offline dictionary attack on the password. To address this issue (without resorting to assumptions like tamper resistance), Ford and Kaliski [17] proposed to distribute the functionality of the server, forcing an attacker to compromise several servers in order to be able to obtain password verification data. Note that the main problem is not just to distribute the password verification data, but to distribute the functionality, i.e., to distribute the password verification data such that it can be used for authentication without ever reconstructing the data on any set of servers smaller than a chosen threshold.

While distributed cryptosystems have been studied extensively (and many proven secure) for other cryptographic operations, such as signatures (e.g., [7, 11, 20, 18]), to our knowledge Ford and Kaliski were the first ones to propose a distributed password-authenticated key exchange system. However, they give no proof of security for their system. Jablon [24] extends the system of Ford and

[1] In the random oracle model [2], a hash function is modeled as a black box containing an ideal random function. This is not a standard cryptographic assumption. In fact, it is possible for a scheme secure in the random oracle model to be insecure for any real instantiation of the hash function [9]. However, a proof of security in the random oracle model is generally thought to be strong evidence of the practical security of a scheme.

Kaliski, most notably to not require the server's public key to be known to the user, but again does not give a proof of security.

Our contributions. In this paper we propose a completely different distributed password authenticated key exchange system and prove it secure in the random oracle model, assuming the hardness of the Decision Diffie-Hellman (DDH) problem [14] (see [6]). While the system of Ford and Kaliski and the system of Jablon require all servers to perform authentication, our system is a k-out-of-n threshold system (for any $1 \leq k \leq n$), where k servers are required for authentication and the compromise of $k-1$ servers does not affect the security of the system. This is the first distributed password-authenticated key exchange system proven secure under any standard cryptographic assumption in any model, including the random oracle model. To be specific, we assume the client may store public data, and our security is against an active attacker that may (statically) compromise any number of servers less than the specified threshold.

Technically, we achieve our result by storing a semantically-secure encryption of a function of the password at the servers (instead of simply storing a one-way function of the password), and then leveraging off some known solutions for distributing secret decryption keys, such as Feldman verifiable secret sharing [16]. In other words, we transform the problem of distributing password authentication information to the problem of distributing cryptographic keys. However, once we make this transformation, verifying passwords without leaking information becomes much more difficult, requiring intricate manipulations of ElGamal encryptions [15] and careful use of efficient non-interactive zero-knowledge proofs [5].

We note that a threshold password authentication system does not follow from techniques for general secure multi-party computation (e.g., [22]) since we are working in an asynchronous model, allow concurrent executions of protocols, and assume no authenticated channels. (Note in particular that the *goal* of the protocol is for the client to be authenticated.) The only work on general secure multi-party computation in an asynchronous model, and allowing concurrency, assumes authenticated channels [10].

2 Model

We extend the model of [1] (which builds on [3] and [4], and is also used by [25]). The model of [1] was designed for the problem of authenticated key exchange (ake) between two parties, a client and a server. The goal was for them to engage in a protocol such that after the protocol was completed, they would each hold a session key that is known to nobody but the two of them. Our model is designed for the problem of *distributed authenticated key exchange* (dake) between a client and k servers. The goal is for them to engage in a protocol such that after the protocol is completed, the client would hold k session keys, one being shared with each server, such that the session key shared between the client and a given server is known to nobody but the two of them, even if up to $k-1$ other servers were to conspire together.

Note that a secure dake protocol allows for secure downloadable credentials, by, e.g., having the servers store an encrypted credentials file with a decryption key stored using a threshold scheme among them, and then having each send a partial decryption of the credentials file to the client, encrypted with the session key it shares with the client. Note that the credentials are secure in a threshold sense: fewer than the given threshold of servers are unable to obtain the credentials. Details are beyond the scope of this paper.

In the following, we will assume some familiarity with the model of [1].

Protocol participants. We have two types of protocol participants: clients and servers. Let $ID \stackrel{\text{def}}{=} \mathit{Clients} \cup \mathit{Servers}$ be a non-empty set of protocol participants, or *principals*.

We assume *Servers* consists of n servers, denoted $\{S_1, \ldots, S_n\}$, and that these servers are meant to cooperate in authenticating a client.[2] Each client $C \in \mathit{Clients}$ has a secret password π_C, and each server $S \in \mathit{Servers}$ has a vector $\pi_S = [\pi_S[C]]_{C \in \mathit{Clients}}$. Entry $\pi_S[C]$ is the *password record*. Let $\mathit{Password}_C$ be a (possibly small) set from which passwords for client C are selected. We will assume that $\pi_C \stackrel{R}{\leftarrow} \mathit{Password}_C$ (but our results easily extend to other password distributions). Clients and servers are modeled as probabilistic poly-time algorithms with an input tape and an output tape.

Execution of the protocol. A protocol P is an algorithm that determines how principals behave in response to inputs from their environment. In the real world, each principal is able to execute P multiple times with different partners, and we model this by allowing unlimited number of *instances* of each principal. Instance i of principal $U \in ID$ is denoted Π_i^U.

To describe the security of the protocol, we assume there is an adversary $\mathcal{A}$ that has complete control over the environment (mainly, the network), and thus provides the inputs to instances of principals. We will further assume the network (i.e., $\mathcal{A}$) performs aggregation and broadcast functions.[3] In practice, on a point-to-point network, the protocol implementor would most likely have to implement these functionalities in some way, perhaps using a single intermediate (untrusted) node to aggregate and broadcast messages[4]. Formally, the adversary is a probabilistic algorithm with a distinguished query tape. Queries written to this tape are responded to by principals according to P; the allowed queries are formally defined in [1] and summarized here (with slight modifications for multiple servers):

Send (U, i, M)**:** causes message M to be sent to instance Π_i^U. The instance computes what the protocol says to, state is updated, and the output of the

[2] Our model could be extended to have multiple sets of servers, but for clarity of presentation we omit this extension.

[3] This is more for notational convenience than anything else. In particular, we make no assumptions about synchronicity or any type of distributed consensus.

[4] Note that since $\mathcal{A}$ controls the network and can deny service at any time, we do not concern ourselves with any denial-of-service attacks that this single intermediate node may facilitate.

computation is given to $\mathcal{A}$. If this query causes Π_i^U to accept or terminate, this will also be shown to $\mathcal{A}$. To initiate a session between client C and a set of servers, the adversary should send a message containing a set I of k indices of servers in *Servers* to an unused instance of C.

Execute $(C, i, ((S_{j_1}, \ell_{j_1}), \ldots, (S_{j_k}, \ell_{j_k})))$**:** causes P to be executed to completion between Π_i^C (where $C \in$ *Clients*) and $\Pi_{\ell_{j_1}}^{S_{j_1}}, \ldots, \Pi_{\ell_{j_k}}^{S_{j_k}}$, and outputs the transcript of the execution. This query captures the intuition of a passive adversary who simply eavesdrops on the execution of P.

Reveal $(C,\ i,\ S_j)$**:** causes the output of the session key held by Π_i^C corresponding to server S_j, i.e., sk_{C,S_j}^i.

Reveal $(S_j,\ i)$**:** causes the output of the session key held by $\Pi_i^{S_j}$, i.e., $sk_{S_j}^i$.

Test $(C,\ i,\ S_j)$**:** causes Π_i^C to flip a bit b. If $b = 1$ the session key sk_{C,S_j}^i is output and if $b = 0$ a string drawn uniformly from the space of session keys is output. A Test query (of either type) may be asked at any time during the execution of P, but may only be asked once.

Test $(S_j,\ i)$**:** causes $\Pi_i^{S_j}$ to flip a bit b. If $b = 1$ the session key $sk_{S_j}^i$ is output; otherwise, a string is drawn uniformly from the space of session keys and output. As above, a Test query (of either type) may be asked at any time during the execution of P, but may only be asked once.

The Reveal queries are used to model an adversary who obtains information on session keys in some sessions, and the Test queries are a technical addition to the model that will allow us to determine if an adversary can distinguish a true session key from a random key.

We assume $\mathcal{A}$ may compromise up to $k - 1$ servers, and that the choice of these servers is static. In particular, without loss of generality, we may assume the choice is made before initialization, and we may simply assume the adversary has access to the private keys of the compromised servers.

Partnering. A server instance that accepts holds a partner-id *pid*, session-id *sid*, and a session key *sk*. A client instance that accepts holds a partner-id *pid*, a session-id *sid*, and a set of k session keys $(sk_{j_1}, \ldots, sk_{j_k})$. Let *sid* be the concatenation of all messages (or pre-specified compacted representations of the messages) sent and received by the client instance in its communication with the set of servers. (Note that this excludes messages that are sent only between servers, but not to the client.) Then instances Π_i^C (with $C \in$ *Clients*) holding $(pid, sid, (sk_{j_1}, \ldots, sk_{j_k}))$ for some set $I = \{j_1, \ldots, j_k\}$ and $\Pi_{\ell_j}^{S_j}$ (with $S_j \in$ *Servers*) holding (pid', sid', sk) are said to be *partnered* if $j \in I$, $pid = S_j$, $pid' = C$, $sid = sid'$, and $sk_j = sk$. This is the so-called "matching conversation" approach to defining partnering, as used in [3, 1].

Freshness. A client instance/server pair (Π_i^C, S_j) is *fresh* if (1) S_j is not compromised, (2) there has been no Reveal $(C,\ i,\ S_j)$ query, and (3) if $\Pi_\ell^{S_j}$ is a partner to Π_i^C, there has been no Reveal $(S_j,\ \ell)$ query. A server instance $\Pi_i^{S_j}$ is *fresh* if (1) S_j is not compromised, (2) there has been no Reveal $(S_j,\ i)$ query, and (3) if Π_ℓ^C is the partner to $\Pi_i^{S_j}$, there has been no $\mathsf{Reveal}\,(C,\ \ell,\ S_j)$ query. Intuitively, the adversary should not be able to distinguish random keys from session keys held by fresh instances.

Advantage of the adversary. We now formally define the distributed authenticated key exchange (dake) advantage of the adversary against protocol P. Let $\mathsf{Succ}_P^{\mathrm{dake}}(\mathcal{A})$ be the event that $\mathcal{A}$ makes a single Test query directed to some instance Π_i^U that has terminated and is fresh, and eventually outputs a bit b', where $b' = b$ for the bit b that was selected in the Test query. The dake advantage of $\mathcal{A}$ attacking P is defined to be

$$\mathsf{Adv}_P^{\mathrm{dake}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} 2\Pr\left[\mathsf{Succ}_P^{\mathrm{dake}}(\mathcal{A})\right] - 1.$$

The following fact is easily verified.

Fact 1.
$$\Pr(\mathsf{Succ}_P^{\mathrm{dake}}(\mathcal{A})) = \Pr(\mathsf{Succ}_{P'}^{\mathrm{dake}}(\mathcal{A}))+\epsilon \quad \Longleftrightarrow \quad \mathsf{Adv}_P^{\mathrm{dake}}(\mathcal{A}) = \mathsf{Adv}_{P'}^{\mathrm{dake}}(\mathcal{A})+2\epsilon.$$

3 Definitions

Let κ be the cryptographic security parameter. Let $G_q \in \mathcal{G}$ denote a finite (cyclic) group of order q, where $|q| = \kappa$. Let g be a generator of G_q, and assume it is included in the description of G_q.

Notation. We use $(a, b)\times(c, d)$ to mean elementwise multiplication, i.e., (ac, bd). We use $(a, b)^r$ to mean elementwise exponentiation, i.e., (a^r, b^r). For a tuple V, the notation $V[j]$ means the jth element of V.

We denote by Ω the set of all functions H from $\{0,1\}^*$ to $\{0,1\}^\infty$. This set is provided with a probability measure by saying that a random H from Ω assigns to each $x \in \{0,1\}^*$ a sequence of bits each of which is selected uniformly at random. As shown in [2], this sequence of bits may be used to define the output of H in a specific set, and thus we will assume that we can specify that the output of a random oracle H be interpreted as a (random) element of G_q.[5] Access to any public random oracle $H \in \Omega$ is given to all algorithms; specifically, it is given to the protocol P and the adversary $\mathcal{A}$. Assume that secret session keys are drawn from $\{0,1\}^\kappa$.

A function $f : \mathbb{Z} \to [0,1]$ is negligible if for all $\alpha > 0$ there exists an $\kappa_\alpha > 0$ such that for all $\kappa > \kappa_\alpha$, $f(\kappa) < |\kappa|^{-\alpha}$. We say a multi-input function is negligible if it is negligible with respect to each of its inputs.

4 Protocol

In this section we describe our protocol for threshold password-authenticated key exchange. In the next section we prove this protocol is secure under the DDH assumption [6, 14] in the random-oracle model [2].

[5] For instance, this can be easily defined when G_q is a q-order subgroup of $\mathbb{Z}_p^\star$, where q and p are prime.

4.1 Server Setup

Let there be n servers $\{S_i\}_{i\in\{1,2,\ldots,n\}}$. Let (x, y) be the servers' *global* key pair such that $y = g^x$. The servers share the global secret key x using a (k, n)-threshold Feldman secret sharing protocol [16]. Specifically, a polynomial $f(z) = \sum_{j=0}^{k-1} a_j z^j \bmod q$ is chosen with $a_0 \leftarrow x$ and random coefficients $a_j \overset{R}{\leftarrow} \mathbb{Z}_q$ for $j > 0$. Then each server S_i gets a secret share $x_i = f(i)$ and a corresponding public share $y_i = g^{x_i}$, $1 \le i \le n$. (In this paper we assume that a trusted dealer generates these shares, but it should be possible to have the servers generate them using a distributed protocol, as in Gennaro *et al.* [19].) In addition, each server S_i independently generates its own *local* key pair (x'_i, y'_i) such that $y'_i = g^{x'_i}$, $1 \le i \le n$. Each server S_i publishes its *local public key* y'_i along with its share of the global public key y_i. Let $H_0, H_1, H_2, H_3, H_4, H_5, H_6 \overset{R}{\leftarrow} \Omega$ be random oracles with domain and range defined by the context of their use. Let $h \leftarrow H_0(y)$ and $h' \leftarrow H_1(y)$ be generators for G_q.

Remark 1. We note that in the following protocol the servers are assumed to have stored the $2n+1$ public values y, $\{y'_i\}_{i=1}^n$, and $\{y_i\}_{i=1}^n$. Likewise, the client is assumed to have stored the $n+1$ public values y and $\{y'_i\}_{i=1}^n$. (Alternatively, a trusted certification authority (CA) could certify these values, but we choose to keep our model as simple as possible.)

4.2 Client Setup

A client $C \in$ *Clients* has a secret password π_C drawn from a set $\mathit{Password}_C$. We assume $\mathit{Password}_C$ can be mapped into $\mathbb{Z}_q$, and for the remainder of the paper, we use passwords as if they were elements of $\mathbb{Z}_q$. C creates an ElGamal ciphertext E_C of the value $g^{(\pi_C)^{-1}}$, using the servers' global public key y. More precisely, he selects $\alpha \overset{R}{\leftarrow} Z_q$ and computes $E_C \leftarrow (y^\alpha g^{(\pi_C)^{-1}}, g^\alpha)$. He sends E_C to each of the servers S_i, $1 \le i \le n$, who record (C, E_C) in their database. (Alternatively, a trusted CA could be used, but again we choose to keep our model as simple as possible.)

Remark 2. We assume the the adversary does not observe or participate in either the system or client setup phases. We assume the client saves a copy of E_C locally. It should be clear that since E_C is public information, this is not the same as storing a shared secret key with the client, which would then obviate the need to use a password for authentication. In particular, it should be noted that instead of storing E_C locally, a client alternatively could obtain a certified copy of E_C through interaction with the servers. Details are beyond the scope of the paper.

4.3 Client Login Protocol

A high level description of the protocol is given in Figure 1, and the formal description may be found in the full paper. Our protocol for a client $C \in$ *Clients* relies on a simulation-sound non-interactive zero-knowledge proof (SS-NIZKP) scheme (see De Santis *et al.* [12] for a definition of an SS-NIZKP scheme) $\mathcal{Q} =$

$(\mathsf{Prove}_{\Phi_Q}, \mathsf{Verify}_{\Phi_Q}, \mathsf{Sim}_{\Phi_Q})$ over a language defined by a predicate Φ_Q that takes elements of $\{0,1\}^* \times (G_q \times G_q)^3$ and is defined as

$$\Phi_Q(\tau, E_C, B, V) \stackrel{\text{def}}{=}$$
$$\exists \beta, \pi, \gamma : \big(B = \big(y^\beta, g^\beta\big) \times (E_C)^\pi \times (g^{-1}, 1)\big) \textbf{ and } (V = (h^\gamma g^\pi, g^\gamma))\,.$$

The algorithms Prove_{Φ_Q}, Verify_{Φ_Q}, and Sim_{Φ_Q} use a random oracle H_3. Prove_{Φ_Q} may be implemented in a standard way as a three-move honest-verifier proof made non-interactive by using the hash function to generate the verifier's random challenge, and having τ be an extra input to the hash function. Other proofs defined below may be implemented similarly.

Here we discuss Figure 1. The client $C \in \mathit{Clients}$ receives a set I of k servers in *Servers* and initiates the protocol with that set, by broadcasting I along with its own identity C. (As stated above, we assume aggregation and broadcast functionalities in the network for the communication between the client and the servers, and among the servers themselves.) In return C receives nonces from the servers in I. The client then "removes" the password from the ciphertext E_C by raising it to π_C and dividing g out of the first element of the tuple, and reblinds the result to form B. The quantity V is then formed to satisfy the predicate Φ_Q, and an SS-NIZKP σ is created to bind B, V, the session public key $\tilde{y}$, and the nonces from the servers. This SS-NIZKP also forces the client to behave properly, and in particular to allow a simulator in the proof of security to operate correctly. (The idea is similar to the use of a second encryption to achieve (lunchtime) chosen-ciphertext security in [27].) After verifying the SS-NIZKP, if the client has used the password $\pi = \pi_C$, it will be that $B[1] = y^{\beta+\alpha\pi}$ and $B[2] = g^{\beta+\alpha\pi}$. The servers then run $\mathsf{DistVerify}(\tau, B, V)$ to verify that $\log_g y = \log_{B[2]} B[1]$. Effectively, they are verifying (without decryption) that B is a valid encryption of the plaintext message 1. Each server S_i then computes a session key K_i, which has also been computed by the client.

Efficiency For the following calculations we use the proof constructions of Figures 3 through 7. Recall that there are k servers involved in the execution of the protocol. The protocol requires six rounds, where each round is an exchange of messages among some of the participants. All messages are of length proportional to the size of a group element. The client is involved in only the first three rounds, while the servers are involved in all rounds. The client performs $15 + k$ exponentiations, and each server performs $22 + 34k$ exponentiations.

Remark 3. These costs are obviously much higher than the Ford-Kaliski scheme, but remember that our protocol is the first to achieve *provable* security (in the random oracle model). Also, the costs may be reasonable for practical implementations with k in the range of 2 to 5.

Remark 4. Our protocol does not provide forward security. To achieve forward security, each server S_i would need to generate its Diffie-Hellman values dynamically, instead of simply using y_i'. Then these values would need to be certified somehow by S_i to protect the client against a man-in-the-middle attack. Details are beyond the scope of this paper.

Client C		**Server $\mathbf{S_i}$ ($\mathbf{i \in I}$)**
	$\xrightarrow{C, I = \langle i_1, \ldots, i_k \rangle}$	
		$c_i \xleftarrow{R} \mathbb{Z}_q$ Broadcast: c_i
	$\xleftarrow{\{c_i\}_{i \in I}}$	
$\tilde{x}, \beta, \gamma \xleftarrow{R} \mathbb{Z}_q$ $\tilde{y} \leftarrow g^{\tilde{x}}$ $B \leftarrow (y^\beta, g^\beta) \times (E_C)^\pi \times (g^{-1}, 1)$ $V \leftarrow (h^\gamma g^\pi, g^\gamma)$ $\tau \leftarrow \langle \tilde{y}, c_{i_1}, \ldots, c_{i_k} \rangle$ $\sigma \leftarrow \mathsf{Prove}_{\Phi_\mathcal{Q}}((\tau, E_C, B, V), (\beta, \pi, \gamma))$ $\forall i \in I,\ \tilde{y}_i \leftarrow (y'_i)^{\tilde{x}}$ $\forall i \in I,\ K_i \leftarrow H_2(I, \tau, \tilde{y}_i)$		
	$\xrightarrow{B, V, \tilde{y}, \sigma}$	
		$\tau \leftarrow \langle \tilde{y}, c_{i_1}, \ldots, c_{i_k} \rangle$ If $\neg \mathsf{Verify}_{\Phi_\mathcal{Q}}((\tau, E_C, B, V), \sigma)$ Then Abort $\mathsf{DistVerify}(\tau, B, V)$ $\tilde{y}_i \leftarrow \tilde{y}^{x'_i}$ $K_i \leftarrow H_2(I, \tau, \tilde{y}_i)$

Fig. 1. Protocol P.

4.4 The DistVerify Protocol

The DistVerify protocol takes three parameters, τ, B, and V, and is run by the servers $\{S_i\}_{i \in I}$ to verify that $\log_g y = \log_{B[2]} B[1]$, i.e., B is an encryption of 1. The parameter V is used in order to allow a proof of security. The protocol is shown in Figure 2, and uses the standard notation for Lagrange coefficients: $\lambda_{j,I} = \prod_{\ell \in I \setminus \{j\}} \frac{-\ell}{j - \ell} \bmod q$. The basic idea of the protocol is as follows. First the servers distributively compute B^r, thus using the (standard) technique of randomizing the quotient $B[1]/(B[2])^x$ if it is not equal to 1. Then they take the second component (i.e., $(B[2])^r$) and distributively compute $((B[2])^r)^x$ using their shared secrets. Finally they verify that $((B[2])^r)^x = (B[1])^r$, implying $B[1] = (B[2])^x$, and hence B is an encryption of 1. DistVerify uses an SS-NIZKP scheme $\mathcal{R} = (\mathsf{Prove}_{\Phi_\mathcal{R}}, \mathsf{Verify}_{\Phi_\mathcal{R}}, \mathsf{Sim}_{\Phi_\mathcal{R}})$ over a language defined by a predicate

$\Phi_{\mathcal{R}}$ that takes elements of $\mathbb{Z} \times (G_q \times G_q)^6$ and is defined as

$$\begin{aligned}\Phi_{\mathcal{R}}(i, B, V, B_i, V_i, V_i', V_i'') \stackrel{\text{def}}{=} \exists r_i, r_i', \gamma_i, \gamma_i', \gamma_i'' : B_i = B^{r_i} \times (y, g)^{r_i'} \textbf{ and}\\ V_i = (h^{\gamma_i} g^{r_i}, g^{\gamma_i}) \textbf{ and } V_i' = (h^{\gamma_i'} (V[1])^{r_i}, g^{\gamma_i'}) \textbf{ and}\\ V_i'' = (h^{\gamma_i''} (V[2])^{r_i}, g^{\gamma_i''}).\end{aligned}$$

The algorithms $\mathsf{Prove}_{\Phi_{\mathcal{R}}}$, $\mathsf{Verify}_{\Phi_{\mathcal{R}}}$, and $\mathsf{Sim}_{\Phi_{\mathcal{R}}}$ use a random oracle H_4.

$\mathsf{DistVerify}$ also uses an SS-NIZKP scheme $\mathcal{S} = (\mathsf{Prove}_{\Phi_{\mathcal{S}}}, \mathsf{Verify}_{\Phi_{\mathcal{S}}}, \mathsf{Sim}_{\Phi_{\mathcal{S}}})$ over a language defined by a predicate $\Phi_{\mathcal{S}}$ that takes elements of $\mathbb{Z} \times \{0,1\}^* \times G_q \times (G_q \times G_q)$ and is defined as

$$\Phi_{\mathcal{S}}(i, \tau', C_i, R_i) \stackrel{\text{def}}{=} \exists a, \gamma : C_i = g^a \textbf{ and } R_i = (h^{\gamma} (h')^a, g^{\gamma}).$$

The algorithms $\mathsf{Prove}_{\Phi_{\mathcal{S}}}$, $\mathsf{Verify}_{\Phi_{\mathcal{S}}}$, and $\mathsf{Sim}_{\Phi_{\mathcal{S}}}$ use a random oracle H_5.

Finally, $\mathsf{DistVerify}$ uses an SS-NIZKP scheme $\mathcal{T} = (\mathsf{Prove}_{\Phi_{\mathcal{T}}}, \mathsf{Verify}_{\Phi_{\mathcal{T}}}, \mathsf{Sim}_{\Phi_{\mathcal{T}}})$ over a language defined by a predicate $\Phi_{\mathcal{T}}$ that takes elements of $\mathbb{Z} \times \{0,1\}^* \times G_q \times G_q \times G_q \times (G_q \times G_q)$ and is defined as

$$\Phi_{\mathcal{T}}(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i) \stackrel{\text{def}}{=} \exists a, \gamma : \overline{C}_i = \overline{g}^a \textbf{ and } C_i = g^a \textbf{ and } R_i = (h^{\gamma} (h')^a, g^{\gamma}).$$

The algorithms $\mathsf{Prove}_{\Phi_{\mathcal{T}}}$, $\mathsf{Verify}_{\Phi_{\mathcal{T}}}$, and $\mathsf{Sim}_{\Phi_{\mathcal{T}}}$ use a random oracle H_6.

5 Security of the Protocol

Here we state the DDH assumption. Following that we prove that the protocol P is secure, based on the DDH assumption.

Decision Diffie-Hellman. Here we formally state the DDH assumption. For full details, see [6]. Let G_q be as in Section 3, with generator g. For two values $X = g^x$ and $Y = g^y$, let $\mathsf{DH}(X, Y) = g^{xy}$. Let $\mathcal{A}$ be an algorithm that on input (X, Y, Z) outputs "1" if it believes that $Z = \mathsf{DH}(X, Y)$, and "0" otherwise. For any $\mathcal{A}$ running in time t

$$\begin{aligned}\mathsf{Adv}_{G_q}^{\mathrm{DDH}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[(x, y) \stackrel{R}{\leftarrow} \mathbb{Z}_q; X \leftarrow g^x; Y \leftarrow g^y; Z \leftarrow g^{xy} : \mathcal{A}(X, Y, Z) = 1\right]\\ - \Pr\left[(x, y, z) \stackrel{R}{\leftarrow} \mathbb{Z}_q; X \leftarrow g^x; Y \leftarrow g^y; Z \leftarrow g^z : \mathcal{A}(X, Y, Z) = 1\right]\end{aligned}$$

Let $\mathsf{Adv}_{G_q}^{\mathrm{DDH}}(t) = \max_{\mathcal{A}} \left\{\mathsf{Adv}_{G_q}^{\mathrm{DDH}}(\mathcal{A})\right\}$, where the maximum is taken over all adversaries of time complexity at most t. The DDH assumption states that for t polynomial in κ, $\mathsf{Adv}_{G_q}^{\mathrm{DDH}}(t)$ is negligible.

5.1 Protocol P

Here we prove that protocol P is secure, in the sense that an adversary attacking the system that compromises fewer than k out of n servers cannot determine session keys with significantly greater advantage than that of an online dictionary attack. Recall that we consider only *static* compromising of servers, i.e., the adversary chooses which servers to compromise before the execution of the system. Let t_{exp} be the time required to perform an exponentiation in G_q.

Step 1: $r_i, r'_i, \gamma_i, \gamma'_i, \gamma''_i \xleftarrow{R} \mathbb{Z}_q$
$B_i \leftarrow B^{r_i} \times (y, g)^{r'_i} \quad V_i \leftarrow (h^{\gamma_i} g^{r_i}, g^{\gamma_i})$
$V'_i \leftarrow (h^{\gamma'_i}(V[1])^{r_i}, g^{\gamma'_i}) \quad V''_i \leftarrow (h^{\gamma''_i}(V[2])^{r_i}, g^{\gamma''_i})$
$\sigma_i \leftarrow \mathsf{Prove}_{\Phi_\mathcal{R}}((i, B, V, B_i, V_i, V'_i, V''_i), (r_i, r'_i, \gamma_i, \gamma'_i, \gamma''_i))$
Broadcast $(B_i, V_i, V'_i, V''_i, \sigma_i)$

Step 2: $\forall j \in I \setminus \{i\}$: Receive $(B_j, V_j, V'_j, V''_j, \sigma_j)$
$\forall j \in I \setminus \{i\}$: If $\neg\mathsf{Verify}_{\Phi_\mathcal{R}}((j, B, V, B_j, V_j, V'_j, V''_j), \sigma_j)$ Then Abort
$(\overline{y}, \overline{g}) \leftarrow \prod_{j \in I} B_j$
$\tau' \leftarrow \langle \tau, B, V, B_{i_1}, \ldots, B_{i_k} \rangle$
$a_i \leftarrow \lambda_{i,I} x_i \quad \overline{C}_i \leftarrow \overline{g}^{a_i} \quad R_i \leftarrow (h^\zeta (h')^{a_i}, g^\zeta)$
$\forall j \in I : C_j \leftarrow (y_j)^{\lambda_{j,I}}$
$\Gamma_i \leftarrow \mathsf{Prove}_{\Phi_\mathcal{S}}((i, \tau', C_i, R_i), a_i)$
Broadcast (R_i, Γ_i)

Step 3: $\forall j \in I \setminus \{i\}$: Receive (R_j, Γ_j)
$\forall j \in I \setminus \{i\}$: If $\neg\mathsf{Verify}_{\Phi_\mathcal{S}}((j, \tau', C_j, R_j), \Gamma_j)$ Then Abort
$\Gamma'_i \leftarrow \mathsf{Prove}_{\Phi_\mathcal{T}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), a_i)$
Broadcast $(\overline{C}_i, \Gamma'_i)$

Step 4: $\forall j \in I \setminus \{i\}$: Receive $(\overline{C}_j, \Gamma'_j)$
$\forall j \in I \setminus \{i\}$: If $\neg\mathsf{Verify}_{\Phi_\mathcal{T}}((j, \tau', \overline{g}, \overline{C}_j, C_j, R_j), \Gamma'_j)$ Then Abort
If $\Pi_{j \in I} \overline{C}_j \neq \overline{y}$ Then Abort

Fig. 2. Protocol $\mathsf{DistVerify}(\tau, B, V)$ for Server S_i $(i \in I)$.

Theorem 1. *Let P be the protocol described in Figure 1 and Figure 2, using group G_q, and with a password dictionary of size N (that may be mapped into $\mathbb{Z}_q$). Fix an adversary $\mathcal{A}$ that runs in time t, and makes $n_{\rm se}, n_{\rm ex}, n_{\rm re}$ queries of type* Send, Execute, Reveal*, respectively, and $n_{\rm ro}$ queries directly to the random oracles. Then for $t' = O(t + (n_{\rm ro} + kn_{\rm se} + k^2 n_{\rm ex})t_{exp})$:*

$$\mathsf{Adv}_P^{\rm dake}(\mathcal{A}) = \frac{n_{\rm se}}{N} + O\left(\mathsf{Adv}_{G_q}^{\rm DDH}(t') + \frac{n^2 + kn_{\rm ro}n_{\rm se} + n_{\rm ro}n + (n_{\rm se} + kn_{\rm ex})^2}{q} + \frac{(n_{\rm se} + kn_{\rm ex})(n_{\rm ro} + n_{\rm se} + kn_{\rm ex})}{q^2}\right).$$

Proof: Our proof will proceed by introducing a series of protocols $P_0, P_1, \ldots, P_7$ related to P, with $P_0 = P$. In P_7, $\mathcal{A}$ will be reduced to simply "guessing" the correct password π_C. We describe these protocols informally in Figure 8. For each i from 1 to 7, we will prove that the difference between the advantage of $\mathcal{A}$ attacking protocols P_{i-1} and P_i is negligible.

$\mu_1, \mu_2, \nu \xleftarrow{R} \mathbb{Z}_q$	$(e, z_1, z_2, z_3) \leftarrow \Gamma_i$
$B' \leftarrow (y^{\mu_1}, g^{\mu_1}) \times (E_C)^{\mu_2}$	
$V' \leftarrow (h^{\nu} g^{\mu_2}, g^{\nu})$	$B' \leftarrow (y^{z_1}, g^{z_1}) \times (E_C)^{z_2} \times (B \times (g, 1))^{-e}$
$e \leftarrow H(\tau, E_C, B, V, B', V')$	$V' \leftarrow (h^{z_3} g^{z_2}, g^{z_3}) \times V^{-e}$
$z_1 \leftarrow \beta e + \mu_1 \bmod q$	
$z_2 \leftarrow \pi e + \mu_2 \bmod q$	Return TRUE if $e = H(\tau, E_C, B, V, B', V')$
$z_3 \leftarrow \gamma e + \nu \bmod q$	
$\sigma \leftarrow (e, z_1, z_2, z_3)$	
Return σ	

Fig. 3. $\mathsf{Prove}_{\Phi_{\mathcal{Q}}}((\tau, E_C, B, V), (\beta, \pi, \gamma))$ and $\mathsf{Verify}_{\Phi_{\mathcal{Q}}}((\tau, E_C, B, V), (e, z_1, z_2, z_3))$

$\mu_1, \mu_2, \nu_1, \nu_2, \nu_3 \xleftarrow{R} \mathbb{Z}_q$
$\tilde{B}_i \leftarrow B^{\mu_1} \times (y^{\mu_2}, g^{\mu_2})$
$\tilde{V}_i \leftarrow (h^{\nu_1} g^{\mu_1}, g^{\nu_1})$
$\tilde{V}_i' \leftarrow (h^{\nu_2} (V[1])^{\mu_1}, g^{\nu_2})$
$\tilde{V}_i'' \leftarrow (h^{\nu_3} (V[2])^{\mu_1}, g^{\nu_3})$
$e \leftarrow H(i, B, V, B_i, V_i, V_i', V_i'', \tilde{B}_i, \tilde{V}_i, \tilde{V}_i', \tilde{V}_i'')$
$z_1 \leftarrow r_i e + \mu_1 \bmod q$
$z_2 \leftarrow r_i' e + \mu_2 \bmod q$
$z_3 \leftarrow \gamma_i e + \nu_1 \bmod q$
$z_4 \leftarrow \gamma_i' e + \nu_2 \bmod q$
$z_5 \leftarrow \gamma_i'' e + \nu_3 \bmod q$

$\sigma \leftarrow (e, z_1, z_2, z_3, z_4, z_5)$
Return σ

Fig. 4. $\mathsf{Prove}_{\Phi_{\mathcal{R}}}((i, B, V, B_i, V_i, V_i', V_i''), (r_i, r_i', \gamma_i, \gamma_i', \gamma_i''))$

$(e, z_1, z_2, z_3, z_4, z_5) \leftarrow \Gamma_i$

$\tilde{B}_i \leftarrow B^{z_1} \times (y^{z_2}, g^{z_2}) \times (B_i)^{-e}$
$\tilde{V}_i \leftarrow (h^{z_3} g^{z_1}, g^{z_3}) \times (V_i)^{-e}$
$\tilde{V}_i' \leftarrow (h^{z_4} (V[1])^{z_1}, g^{z_4}) \times (V_i')^{-e}$
$\tilde{V}_i'' \leftarrow (h^{z_5} (V[2])^{z_1}, g^{z_5}) \times (V_i'')^{-e}$

Return TRUE if $e = H(i, B, V, B_i, V_i, V_i', V_i'', \tilde{B}_i, \tilde{V}_i, \tilde{V}_i', \tilde{V}_i'')$

Fig. 5. $\mathsf{Verify}_{\Phi_{\mathcal{R}}}((i, B, V, B_i, V_i, V_i', V_i''), (e, z_1, z_2, z_3, z_4, z_5))$

$\mu, \nu \xleftarrow{R} \mathbb{Z}_q$	$(e, z_1, z_2) \leftarrow \Gamma_i$
$W \leftarrow g^{\mu}$	
$R' \leftarrow (h^{\nu}(h')^{\mu}, g^{\nu})$	$R' \leftarrow (h^{z_2}(h')^{z_1}(R_i[1])^{-e}, g^{z_2}(R_i[2])^{-e})$
$e \leftarrow H(i, \tau', C_i, R_i, W, R')$	$W \leftarrow g^{z_1}(C_i)^{-e}$
$z_1 \leftarrow ae + \mu \bmod q$	
$z_2 \leftarrow \gamma e + \nu \bmod q$	Return TRUE if $e = H(i, \tau', C_i, R_i, W, R')$
$\Gamma_i \leftarrow (e, z_1, z_2)$	
Return Γ_i	

Fig. 6. $\mathsf{Prove}_{\Phi_{\mathcal{S}}}((i, \tau', C_i, R_i), (a, \gamma))$ and $\mathsf{Verify}_{\Phi_{\mathcal{S}}}((i, \tau', C_i, R_i), \Gamma_i)$

$\mu, \nu \xleftarrow{R} \mathbb{Z}_q$	$(e, z_1, z_2) \leftarrow \Gamma_i'$
$\overline{W} \leftarrow \overline{g}^{\mu}$	
$W \leftarrow g^{\mu}$	$R' \leftarrow (h^{z_2}(h')^{z_1}(R_i[1])^{-e}, g^{z_2}(R_i[2])^{-e})$
$R' \leftarrow (h^{\nu}(h')^{\mu}, g^{\nu})$	$\overline{W} \leftarrow \overline{g}^{z_1}(\overline{C}_i)^{-e}$
$e \leftarrow H(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, \overline{W}, W, R')$	$W \leftarrow g^{z_1}(C_i)^{-e}$
$z_1 \leftarrow ae + \mu \bmod q$	
$z_2 \leftarrow \gamma e + \nu \bmod q$	Return TRUE
	if $e = H(i, \tau', \overline{g}, \overline{C}_i, C_i, R_i, \overline{W}, W, R')$
$\Gamma_i' \leftarrow (e, z_1, z_2)$	
Return Γ_i'	

Fig. 7. $\mathsf{Prove}_{\Phi_{\mathcal{T}}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), (a, \gamma))$ and $\mathsf{Verify}_{\Phi_{\mathcal{T}}}((i, \tau', \overline{g}, \overline{C}_i, C_i, R_i), \Gamma_i')$

We will sketch these proofs here, and leave the details to the full paper.

$P_0 \to P_1$ The probability of a collision of nonces is easily seen to be negligible.

$P_1 \to P_2$ This can be shown using a standard reduction from DDH. On input (X, Y, Z), we plug in random powers of Y for the servers' local public keys, and random powers of X for the clients' $\tilde{y}$ values, and then check H_2 queries for appropriate powers of Z.

$P_2 \to P_3$ This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for $h = H_0(y)$, and we use X and Z to create (randomized) encryptions for all V, V_i, V_i', V_i'', and R_i values. Also, we must factor in the negligible probability of a simulation error in one of the SS-NIZKP proofs,

$P_3 \to P_4$ This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for y, simulate the public shares of the uncompromised servers, and use X and Z to create (randomized) encryptions for all B values. To make sure authentication succeeds for a client that uses this B value, we generate $\overline{C}_i$ values from uncompromised servers in such as way that the product is the $\overline{y}$ value, and we simulate the SS-NIZKP proofs.

P_0 The original protocol P.
P_1 The nonces are assumed to be distinct (and thus Reveal queries do not reveal anything that could help in a Test query).
P_2 The Diffie-Hellman key exchange between a client and an uncompromised server is replaced with a perfect key exchange (and thus an adversary that does not succeed in impersonating a client to an uncompromised server does not obtain any information that could help in a Test query).
P_3 Value V from a client, and values V_i, V_i', V_i'', R_i from uncompromised servers, are replaced by random values. The $\mathcal{Q}$-SS-NIZKP σ, and each $\mathcal{R}$-SS-NIZKP σ_i, $\mathcal{S}$-SS-NIZKP Γ_i, and $\mathcal{T}$-SS-NIZKP Γ_i', are constructed using the associated simulators.
P_4 Value B from a client is replaced with a random value, but the $\overline{C}_i$ values from uncompromised servers are changed to force the associated authentication to succeed.
P_5 The adversary succeeds if it ever sends a V value associated with the correct password.
P_6 Abort if the adversary creates a new and valid $\mathcal{S}$-SS-NIZKP proof or $\mathcal{T}$-SS-NIZKP proof associated with an uncompromised server.
P_7 Value E_C for each client is changed to a random value, and on any adversary login attempt for C, the $\overline{C}_i$ values from uncompromised servers are replaced with values generated to form a random $\overline{y}$ (so as to force a failure).

Fig. 8. Informal description of protocols P_0 through P_7

The difficulty is now in performing authentication on B values chosen by the adversary, since we do not know the secret shares (the x_i values) for the uncompromised servers. Therefore to perform authentication, we plug a value with a known discrete log in for $h = H_0(y)$ so we can decrypt all V, V_i, V_i', and V_i'' values, and then use these decryptions to aid in computing the correct value of $\overline{g}^x$ (even though we don't know x). Finally, we generate $\overline{C}_i$ values from uncompromised servers in such as way that the product is $\overline{g}^x$, and we simulate the SS-NIZKP proofs.

$P_4 \rightarrow P_5$ This is straightforward, since this could only increase the probability of the adversary succeeding.

$P_5 \rightarrow P_6$ This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for y, simulate the public shares of the uncompromised servers, and let $h' = X$. Given a correct SS-NIZKP for an uncompromised server, we can compute $(h')^x$, where $y = g^x$ (where x is not known). Then we simply check if $Z = (h')^x$.

$P_6 \rightarrow P_7$ This can be shown using a reduction from DDH. On input (X, Y, Z), we plug Y in for y, simulate the public shares of the uncompromised servers, and use X and Z to create (randomized) encryptions for all E_C values. This does not affect authentication using B values generated by clients (since these values are random at this point, anyway). The difficulty is in obtaining the right distribution of $\overline{C}_i$ values while authenticating B values chosen by the adversary. To do this we use X and Z in our creation of the B_i values for uncompromised servers, which leaves $\overline{C}_i$ values correct if (X, Y, Z) is a true DH triple, but has the affect of randomizing the $\overline{C}_i$ values if (X, Y, Z) is a random triple. Again, the decryptions of V, V_i, V_i', and V_i'' are used to aid in computing the true $\overline{g}^x$ value (even though we don't know x) when (X, Y, Z) is a true DH triple, or the appropriate random value, when (X, Y, Z) is a random triple.

One can see that in P_2, an adversary that does not succeed in impersonating a client to an uncompromised server gains negligible advantage in determining a real session key from a random session key. The remainder of the protocols are used to show that an adversary gains negligible advantage in impersonating a client over a simple online guessing attack. In particular, in P_7 the password is only used to check V values submitted by the adversary attempting to impersonate a client. The theorem follows. ∎

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000* (LNCS 1807), pp. 139–155, 2000.
2. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In 1^{st} *ACM Conference on Computer and Communications Security*, pages 62–73, November 1993.
3. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO '93* (LNCS 773), pp. 232–249, 1993.
4. M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *27th ACM Symposium on the Theory of Computing*, pp. 57–66, 1995.
5. M. Blum, P. Feldman and S. Micali. Non-interactive zero-knowledge and its applications. In *20th ACM Symposium on the Theory of Computing*, pp. 103–112, 1988.
6. D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium* (LNCS 1423), pp. 48–63, 1998.
7. C. Boyd. Digital multisignatures. In H. J. Beker and F. C. Piper, editors, *Cryptography and Coding*, pages 241–246. Clarendon Press, 1986.
8. V. Boyko, P. MacKenzie, and S. Patel. Provably secure password authentication and key exchange using Diffie-Hellman. In *EUROCRYPT 2000* (LNCS 1807), pp. 156–171, 2000.
9. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In *30th ACM Symposium on the Theory of Computing*, pp. 209–218, 1998.
10. R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai. Universally Composable Two-party Computation. In *34th ACM Symposium on the Theory of Computing*, 2002.
11. Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *CRYPTO '89* (LNCS 435), pages 307–315, 1989.
12. A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano and A. Sahai. Robust non-interactive zero knowledge. In *CRYPTO 2001* (LNCS 2139), pp. 566–598, 2001.
13. T. Dierks and C. Allen. The TLS protocol, version 1.0, IETF RFC 2246, January 1999.

14. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22(6):644–654, 1976.
15. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Info. Theory*, 31:469–472, 1985.
16. P. Feldman. A Practical Scheme for Non-Interactive Verifiable Secret Sharing. In *28th IEEE Symp. on Foundations of Computer Science*, pp. 427-437, 1987
17. W. Ford and B. S. Kaliski, Jr. Server-assisted generation of a strong secret from a password. In *Proceedings of the* 5^{th} *IEEE International Workshop on Enterprise Security*, 2000.
18. Y. Frankel, P. MacKenzie, and M. Yung. Adaptively-secure distributed threshold public key systems. In *European Symposium on Algorithms* (LNCS 1643), pp. 4–27, 1999.
19. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. The (in)security of distributed key generation in dlog-based cryptosystems. In *EUROCRYPT '99* (LNCS 1592), pp. 295–310, 1999.
20. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *EUROCRYPT '96* (LNCS 1070), pages 354–371, 1996.
21. O. Goldreich and Y. Lindell. Session-key generation using human passwords only. In *CRYPTO 2001* (LNCS 2139), pp. 408–432, 2001.
22. O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.
23. D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review, ACM SIGCOMM*, 26(5):5–20, 1996.
24. D. Jablon. Password authentication using multiple servers. In em RSA Conference 2001, Cryptographers' Track (LNCS 2020), pp. 344–360, 2001.
25. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *EUROCRYPT 2001* (LNCS 2045), pp. 475–494, 2001.
26. P. MacKenzie, S. Patel, and R. Swaminathan. Password authenticated key exchange based on RSA. In *ASIACRYPT 2000*, (LNCS 1976), pp. 599–613, 2000.
27. M. Naor and M. Yung. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *22nd ACM Symposium on the Theory of Computing*, pp. 427–437, 1990.
28. SSH Communications Security. `http://www.ssh.fi`, 2001.
29. T. Wu. The secure remote password protocol. In *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium*, pp. 97–111, 1998.