A Method for Making Password-Based Key Exchange Resilient to Server Compromise*

Craig Gentry[1], Philip MacKenzie[2], and Zulfikar Ramzan[3]

[1] Stanford University, Palo Alto, CA, USA, cgentry@cs.stanford.edu
[2] Google, Inc., Mountain View, CA, USA, philmac@google.com
[3] Symantec, Inc., Redwood City, CA, USA, zulfikar_ramzan@symantec.com

Abstract. This paper considers the problem of password-authenticated key exchange (PAKE) in a client-server setting, where the server authenticates using a stored password file, and it is desirable to maintain some degree of security even if the server is compromised. A PAKE scheme is said to be *resilient to server compromise* if an adversary who compromises the server must at least perform an offline dictionary attack to gain any advantage in impersonating a client. (Of course, offline dictionary attacks should be infeasible in the absence of server compromise.) One can see that this is the best security possible, since by definition the password file has enough information to allow one to play the role of the server, and thus to verify passwords in an offline dictionary attack.
While some previous PAKE schemes have been proven resilient to server compromise, there was no known general technique to take an arbitrary PAKE scheme and make it provably resilient to server compromise. This paper presents a practical technique for doing so which requires essentially one extra round of communication and one signature computation/verification. We prove security in the universal composability framework by (1) defining a new functionality for PAKE with resilience to server compromise, (2) specifying a protocol combining this technique with a (basic) PAKE functionality, and (3) proving (in the random oracle model) that this protocol securely realizes the new functionality.

1 Introduction

THE BASIC PROBLEM. We start by describing the basic problem of setting up a secure channel between two parties, Alice and Bob, who only share a short secret password. Neither of them knows a public key corresponding to the other party, and neither has a certified public key (i.e., a public key whose certificate can be verified by the other party). If Alice and Bob shared a high-strength cryptographic key (i.e., a *long* secret), then this problem could be solved using standard solutions for setting up a secure channel, such as the protocol of Bellare and Rogaway [5]. However, since Alice and Bob only share a short secret

* This work was carried out while all the authors were at DoCoMo USA Labs.

password, they must also be concerned with *offline dictionary attacks*. An offline dictionary attack occurs when an attacker obtains some information that can be used to perform offline verification of password guesses. We will call this *password verification information*. For a specific example, consider the following. Say Alice and Bob share a password π, and say an attacker somehow obtained a hash of the password $h(\pi)$, where h is some common cryptographic hash function such as SHA-1 [41]. Then an attacker could go offline and run through a dictionary of possible passwords $\{\pi_1, \pi_2, \ldots\}$, testing each one against $h(\pi)$. For instance, to test if π_i is the correct password, the attacker computes $h(\pi_i)$ and checks if $h(\pi_i) = h(\pi)$. In general, the password verification information obtained by the attacker may not be as simple as a hash of a password, and an attacker may not always be able to test all possible passwords against the password verification information, but if he can test a significant number of passwords, this is still considered an offline dictionary attack. For some fairly recent demonstrations of how effective an offline dictionary attack can be, see [40, 44, 50]. So the problem remains: how do Alice and Bob set up a secure channel? In other words: how do Alice and Bob bootstrap a short secret (the password) into a long secret (a cryptographic key) that can be used to provide a secure channel?

A protocol to solve this problem is called a *password-authenticated key exchange (PAKE)* protocol. Informally, a PAKE protocol is secure if the only feasible way to attack the protocol is to run a trivial *online dictionary attack* of simply iteratively guessing passwords and attempting to impersonate one of the parties. (Note that this type of attack can generally be detected and stopped by well-known methods.) The problem of designing a secure PAKE protocol was proposed by Bellovin and Merritt [7] and by Gong *et al.* [24], and has since been studied extensively. Below we discuss the many techniques that have been proposed.

Resilience to Server Compromise. Consider a PAKE protocol run in a client-server setting, where the client device receives a password input by a user, but where the server stores a "password file" that contains data that can be used to authenticate each user. In this scenario it is natural to be concerned about the security of this password file, since an adversary that compromises the server could obtain this password file.[4] In the case of many existing PAKE protocols, the consequences of an adversary obtaining the server's password file are disastrous, with the adversary obtaining enough information to impersonate a client. That is why there has been a significant amount of work on making PAKE schemes "as secure as possible" even if the server gets compromised. Naturally, if an adversary obtains a server password file, he possesses password verification information, so he can always mount an offline dictionary attack. The goal, therefore, in improving resilience to server compromise is to make the offline dictionary attack the best he can do.

[4] From the many recent reports of theft of credit cards and other personal information from e-commerce servers, it seems that compromise of a server is a real threat.

In the remainder of the paper, a *symmetric PAKE scheme* refers to one in which the two parties use identical strings corresponding to the same password (and which, consequently is trivially insecure in the client-server setting when the server is compromised). An *asymmetric PAKE scheme* refers to one which is designed to maintain security (as discussed above) despite a server compromise. In particular, this implies that the server does not store the plaintext password.

Related Work. Since the PAKE problem was introduced, it has been studied extensively, and many PAKE protocols have been proposed, e.g., [24, 23, 26, 34, 48, 32]. Many of these protocols have been shown to be insecure [9, 45]. More recent protocols, e.g., [3, 10, 1, 37, 51, 19], have proofs of security, based on certain well-known cryptographic assumptions, in the random oracle and/or ideal cipher models. Other PAKE protocols have been proven secure in the common reference string (CRS) model, e.g., [31, 18, 29, 14]. Finally the PAKE protocols in [20, 42] were proven secure based on a general assumption (trapdoor permutations) without any setup assumptions, but with a restriction that concurrent sessions with the same password are prohibited.

The problem of PAKE with resilience to server compromise has also been studied extensively, and many protocols have been proposed, e.g., [8, 27, 49, 33].[5] Some more recent protocols also have proofs of security based on well-known cryptographic assumptions, in the random oracle model, e.g., [10, 37, 35]. Although these protocols (along with the protocols of [8, 27]) are based on symmetric PAKE protocols, and the techniques used to convert the symmetric PAKE protocols into asymmetric PAKE protocols seem somewhat modular, no modular versions were ever presented, and there were no attempts to prove (in a modular way) anything about the techniques themselves. Each asymmetric PAKE protocol was presented in its entirety, and was proven secure from scratch. Note that no protocols for PAKE with resilience to server compromise have yet been proven secure without relying on random oracles.

Results. We were inspired by the PAK-Z protocol from MacKenzie [36], which is essentially the PAK protocol from [10] modified using the "Z-method" to be resilient to server compromise.[6] While the Z-method was claimed to be a general technique, it was only described and analyzed with respect to the PAK protocol. We first show that the general Z-method does not provide resilience to server compromise by exhibiting an attack that exploits any instantiation using discrete-log based signature schemes. Next, we present a new method, called the Ω-method, that fixes the critical flaw in the Z-method. The Ω-method is the first general and modular technique that takes any secure symmetric PAKE scheme as a building block and converts it into one that is resilient to server compromise. The Ω-method is efficient and practical, essentially adding one

[5] There has also been work on protecting the server password file using threshold techniques, e.g., [17, 28, 16, 30, 38].

[6] Previous to PAK-Z, there were PAK-X and PAK-Y protocols, with their own methods for modifying PAK to be resilient to server compromise.

extra round of communication and one signature generation/verification to the underlying symmetric PAKE scheme.[7]

We prove security in the universal composability (UC) framework [11] (in the random oracle model). A symmetric PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ was recently introduced in [14]. Our original plan was to (1) extend $\mathcal{F}_{\mathsf{pwKE}}$ into an asymmetric PAKE functionality $\mathcal{F}_{\mathsf{apwKE}}$, and (2) prove that a protocol based on the Ω-method (which we call the Ω-protocol) securely realizes $\mathcal{F}_{\mathsf{apwKE}}$ in the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid model. This would imply, by the universal composition theorem [11], that the Ω-protocol would be secure when instantiated with any secure symmetric PAKE scheme. For step (1) we added notions of a server setting up a password record for the client and using that password record for each session, the notion of stealing the password file, and the notion of explicitly aborting.[8] Unfortunately, step (2) was problematic, since the Ω-method relies on the notion of a protocol transcript, which does not exist in the symmetric PAKE functionality of [14]. Therefore, we added the notion of a transcript to the symmetric PAKE functionality to make a revised symmetric PAKE functionality $\mathcal{F}_{\mathsf{rpwKE}}$, and completed step (2) using the $\mathcal{F}_{\mathsf{rpwKE}}$-hybrid model. In Section 4 we discuss why adding this notion of a transcript is natural and does not have any substantial effect on whether a protocol securely realizes the functionality.

Applicability. Currently there is only one PAKE protocol that has been shown to securely realize the symmetric PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ in the UC framework, specifically, the one of Canetti *et al.* [14]. However, we conjecture that many of the PAKE protocols cited above that were proven secure in the random oracle model, but not in the UC framework, could also be proven secure in the UC framework. Since the Ω-protocol relies on the random oracle model anyway, it would make sense to combine it with these symmetric PAKE protocols to achieve (very efficient) asymmetric PAKE protocols. Thus the results of this paper should have wide applicability.

2 Preliminaries

Symmetric Encryption Schemes. A *symmetric encryption scheme* $\mathcal{E}$ is a pair (E, D) of algorithms, both running in polynomial time. E takes a symmetric key k and a message m as input and outputs an encryption c for m; we denote this $c \leftarrow E_k(m)$. D takes a ciphertext c and a symmetric key k as input and returns either a message m such that c is a valid encryption of m, if such an m exists, and otherwise returns $\perp$.

[7] As a result of this work, the PAK-Z protocol in the IEEE P1363.2 (Password-based Public-Key Cryptography) proposed standard has had the Z-method replaced with the Ω-method to provide resilience to server compromise.

[8] We do not consider the notion of explicitly aborting to be necessary for an asymmetric PAKE functionality, but it is very convenient and allows some natural protocols (including our protocol) to securely realize the functionality. There is more discussion on this notion in Section 4.

We will use specific symmetric encryption schemes based on hash functions and one-time pads.[9] The first scheme is $E_k(m) = H(k) \oplus m$, where $H()$ is a hash function with output that is the same length as m (and assumed to behave like a random oracle - see below) and where $\oplus$ is taken as a bit-wise exclusive OR operation. Note that this encryption scheme is inherently malleable. For instance, given a ciphertext c of an unknown message m under an unknown key k, one can construct a ciphertext $c' = c \oplus 00\cdots001$ of a related message $m' = m \oplus 00\cdots001$ (i.e., m with the last bit flipped), without determining m.

The second scheme is $E'_k(m) = H(k) \oplus m|H'(m)$, for a hash function $H'()$ (assumed to be one-way and to behave like a random oracle). The second hash protects against malleability since modifying the one-time pad portion requires recomputing the hash with the correct message, implying the message has been determined.

Hash functions. Cryptographic hash functions will be used for key generation and for producing verification values. Here we assume these functions are random oracles [4],[10] i.e., they behave like black-box perfectly random functions. In practice, one would need to verify that the actual hash function used is suitable to be used as a random oracle. See [4] for a discussion on how to instantiate random oracles, and see [25] for a discussion on key generation functions.

Signature schemes A *signature scheme* $\mathcal{S}$ is a triple $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Verify})$ of algorithms, the first two being probabilistic, and all running in (probabilistic) polynomial time. $\mathsf{Gen}_{\mathcal{S}}$ takes as input the security parameter (usually denoted as κ and represented in unary format, i.e., 1^κ) and outputs a public-key pair (pk, sk), i.e., $(pk, sk) \leftarrow \mathsf{Gen}_{\mathcal{S}}(1^\kappa)$. Sign takes a message m and a secret key sk as input and outputs a signature σ for m, i.e., $\sigma \leftarrow \mathsf{Sig}_{sk}(m)$. Verify takes a message m, a public key pk, and a candidate signature σ' for m as input and returns the bit $b = 1$ if σ' is a valid signature for m for the corresponding private key, and otherwise returns the bit $b = 0$. That is, $b \leftarrow \mathsf{Verify}_{pk}(m, \sigma')$. Naturally, if $\sigma \leftarrow \mathsf{Sig}_{sk}(m)$, then $\mathsf{Verify}_{pk}(m, \sigma) = 1$. We require signature schemes that are existentially unforgeable against chosen message attacks in the sense of [21].

3 The Ω-method

Basic Idea of the Ω-Method. Similar to some previous work [10, 37, 35, 36], the Ω-method constructs an asymmetric PAKE protocol Ω by enhancing a symmetric PAKE protocol P as follows. First, the server only stores the output of a one-way function of the password, i.e., a value $f(\pi)$ that is easy to compute from the password, but from which the password is difficult to compute. Then the protocol operates by first running the protocol P using $f(\pi)$ in place of π,[11]

[9] Proving security using generic encryption schemes is left as an open problem.

[10] We stress that whether schemes proven secure in the random oracle model can be instantiated securely in the real world (i.e., with polynomial-time computable functions in place of random oracles) is uncertain [13, 12, 43, 22, 6, 39].

[11] We always assume a password protocol takes an arbitrary length password string, and thus would work correctly with $f(\pi)$ in place of π.

and second having the client somehow prove knowledge of a π such that $f(\pi)$ is the server's stored value.

The Ω-method uses the following specific instantiation of this basic idea.[12] The server stores a hash of the password $H(\pi)$ to be used for the protocol P. The server also stores a public/secret key pair for a secure signature scheme, with the secret key encrypted using the specific encryption scheme (E', D') defined above in which the key to the encryption scheme is the password. (Recall that this encryption scheme is a one-time pad concatenated to a cryptographic hash.) In all, the server stores $(H(\pi), pk, E'_\pi(sk))$. The protocol Ω first runs P (using $H(\pi)$). Once P is finished and has derived a cryptographically strong shared key K, the server uses a temporary session key K' derived from K to securely send $E'_\pi(sk)$ to the client, using the specific encryption scheme (E, D) defined above. (Recall that this encryption scheme is simply a one-time pad.) The client uses K and π to derive the appropriate keys, performs the necessary decryptions to obtain sk, and then creates a signature with sk on the transcript of P. In effect, this proves that the client (the one communicating with the server in protocol P) knows π. The final output of Ω is another key K'' derived from K.

The Ω-method is very similar to the Z-method in [36]. However, the Z-method specifies that both encryption schemes be simple one-time pads, which, as discussed previously, are malleable. Because of this, it can be shown that the Z-method is insecure for certain signature schemes and in particular, for certain representations of the private key generated in certain signature schemes. In fact, the Z-method is not known to be secure for any signature scheme. In contrast, the Ω-method is secure for every signature scheme.

High-level description. A high-level description of the Ω-method is shown in Figure 1, with some details given here.

Client Part 1: The client computes $H(\pi)$ and performs its part in the symmetric PAKE protocol P using $H(\pi)$, obtaining a shared cryptographic key K.

Server Part 1: The server performs its part in the symmetric PAKE protocol P using the value $H(\pi)$ it had stored, obtaining a shared cryptographic key K.

Server Part 2, Step 1: The server first derives key $K' = H'(K)$ and then sends $E_{K'}(E'_\pi(sk))$ to the client.

Client Part 2: The client receives the value $E_{K'}(E'_\pi(sk))$ sent by the server, computes $K' = H'(K)$, and decrypts using K' and π to obtain sk. (If the decryption fails when checking the hash, the client aborts.) Then the client signs the transcript of P, using sk, i.e., it computes $\sigma = \mathsf{Sig}_{sk}(\mathsf{transcript})$, and sends the result σ to the server. The client also derives a session key K'' from K (e.g., by using a cryptographic hash function $K'' = H''(K)$).

[12] Note that for security we require that the hash and encryption functions used by the Ω-method are not used by the underlying symmetric PAKE protocol P.

Server Part 2, Step 2: Once it receives the signature σ, the server computes $b = \mathsf{Verify}_{pk}(\mathsf{transcript}, \sigma)$. If $b = 1$, then the server derives a session key $K'' = H''(K)$ and outputs it. Otherwise it aborts.

Client(π)		Server($H(\pi), (pk, E'_\pi(sk))$)
Compute $H(\pi)$		
	Symmetric PAKE: $H(\pi)$ $\longleftrightarrow$	
Output: K Generate keys K' and K'' from K		Output: K Generate keys K' and K'' from K Let $c = E'_\pi(sk)$
$sk \leftarrow D'_\pi(D_{K'}(c'))$	$\xleftarrow{c'}$	$c' \leftarrow E_{K'}(c)$
$s \leftarrow \mathsf{Sig}_{sk}(\mathsf{transcript})$	$\xrightarrow{s}$	If $\neg\mathsf{Verify}_{pk}(\mathsf{transcript}, s)$, abort.

Fig. 1. The Ω-method: Augmenting a PAKE protocol to make it resilient to server compromise.

The advantage of the Ω-method over the Z-method was discussed above. The advantage of the Ω-method over other asymmetric PAKE protocols is that it is modular and general. The Ω-method allows an asymmetric PAKE protocol to be constructed using any PAKE protocol and any signature scheme, and these could be chosen based on which cryptographic assumptions are used to prove their security. For instance, one could choose a PAKE protocol and a signature scheme that are based on the same cryptographic assumption. Notice also that as opposed to asymmetric PAKE protocols in which the password is used to derive the secret key of a signature scheme, e.g., [35], the Ω-method has the advantage that the secret key does not need to be computed online by the client, a potentially expensive operation.

As for efficiency, this method adds one extra round of communication along with a few hash operations, a signature calculation by the client, and a signature verification by the server, to the PAKE protocol P. The extra round of communication can often be piggybacked on actual protocol messages, as was shown for the PAK-Z protocol in [36]. But there is still the extra computation involved with the signature. Some asymmetric PAKE protocols have been designed specifically to avoid this extra computation. SRP [49] and AMP [33] are two such protocols, but neither has a proof of security, even in the random oracle model.

The Z-Method and An Attack. The Z-method [36] looks exactly like the Ω-method, except that the encryption functions E and E' are simply one-time pads. That is, $E_{K'}(c) = K' \oplus c$ and $E'_\pi(sk) = H(\pi) \oplus sk$. One problem with a one-time pad encryption scheme is that in the absence of an explicit integrity-verification mechanism, the resulting ciphertext is malleable. This leads to an attack on the PAKE protocol. In particular, suppose that the Z-method is instantiated using a discrete logarithm based signature scheme (e.g., Schnorr, DSS, El-Gamal, signature schemes based on bilinear pairings, etc.). In a typical instantiation of such a scheme, we would have $(sk, pk) = (x, y)$ be the signing and verification keys respectively, where $y = g^x$ and g is a generator for a (multiplicative) group in which the discrete logarithm problem is believed to be intractable.

Now, consider an active adversary who flips the least significant bit of the server's response c'. When the client decrypts, he will either compute that $sk = x+1$ (if the least significant bit of x were a 0) or $x-1$ (or if the least significant bit of x were a 1). The client signs the transcript using the computed signing key. The adversary verifies the signature using yg as the public verification key. If it verifies correctly, then he deduces that the least significant bit of x was 0; otherwise he deduces that it is 1. The adversary can repeat an analogous procedure $|x|$ times to determine the remaining bits of x. Since $|x|$ is typically much smaller than the dictionary of possible passwords, we violate the security requirements of the protocol.

Evaluation of Security. We wish to prove that the Ω-method can be combined with any (symmetric) PAKE protocol to yield an asymmetric PAKE protocol. It is often very difficult to prove such general statements. However, by using the Universal Composability (UC) framework of Canetti [11], this type of proof, while still very complicated, becomes much easier. We assume the reader is familiar with the UC framework. We remark that we focus on static adversaries that cannot corrupt parties during the execution.[13] (However, our asymmetric PAKE functionality includes a notion of an adversary stealing a password file.)

4 Password-based key exchange functionalities

The original symmetric PAKE functionality. We first consider the original symmetric PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ from Canetti *et al.* [14] and presented in Figure 2.[14] The functionality is similar to that of the standard key exchange functionality $\mathcal{F}_{\mathsf{KE}}$ given in Canetti [11]. In the $\mathcal{F}_{\mathsf{KE}}$ functionality, the parties that start a session receive the same uniformly-distributed session key, except when one of the parties is corrupted, in which case the adversary has the power to set the session key. In the $\mathcal{F}_{\mathsf{pwKE}}$ functionality, each party starting a session is given a password as input from the environment, and the power to set the session key for a party is also given to the adversary if it succeeds in guessing the password used by that party. When the adversary guesses the password, the party's session is marked compromised. An additional property of the definition is that a failed attempt at guessing a party's password is detected. This results in the session being marked interrupted and the party receiving an independent uniformly-distributed session key.

[13] Nevertheless, as was shown in [14], this implies the "weak corruption" model of [3] in which passwords can be adaptively obtained.

[14] Note that the variable names in the functionality have been slightly modified for consistency with protocols that we present later.

A session that is neither compromised nor interrupted (and is still in progress) is considered fresh. Such sessions (between honest parties) conclude with both parties receiving the same, uniformly-distributed session key if they were given the same password as input from the environment, and otherwise with the parties receiving independent uniformly-distributed session keys. In any case, once a party receives a session key, that session is marked completed.

Functionality $\mathcal{F}_{\mathsf{pwKE}}$

The functionality $\mathcal{F}_{\mathsf{pwKE}}$ is parameterized by a security parameter κ. It interacts with an adversary S and a set of parties via the following queries:

Upon receiving a query $(\mathsf{NewSession}, sid, P_i, P_j, \pi, \text{role})$ **from party** P_i**:**
Send $(\mathsf{NewSession}, sid, P_i, P_j, \text{role})$ to S. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record (P_j, P_i, π'), then store record (P_i, P_j, π) and mark this record fresh.
Upon receiving a query $(\mathsf{TestPwd}, sid, P_i, \pi')$ **from the adversary** S**:**
If there is a record of the form (P_i, P_j, π) which is fresh, then do: If $\pi = \pi'$, mark the record compromised and reply to S with "correct guess". If $\pi \neq \pi'$, mark the record interrupted and reply with "wrong guess".
Upon receiving a query $(\mathsf{NewKey}, sid, P_i, k)$ **from** S**, where** $|k| = \kappa$**:**
If there is a record of the form (P_i, P_j, π), and this is the first NewKey query for P_i, then:
- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, k) to player P_i.
- If this record is fresh, and there is a record (P_j, P_i, π') with $\pi' = \pi$, and a key k' was sent to P_j, and (P_j, P_i, π) was fresh at the time, then output (sid, k') to P_i.
- Any other case: pick new random key k' ($|k'| = \kappa$); send (sid, k') to P_i.

Either way, mark the record (P_i, P_j, π) as completed.

Fig. 2. The password-based key-exchange functionality $\mathcal{F}_{\mathsf{pwKE}}$

Two enhancements to the original symmetric PAKE functionality. We describe two enhancements, transcripts and explicit authentication, to the original $\mathcal{F}_{\mathsf{pwKE}}$ functionality [14].

Transcripts. The main problem with building a secure asymmetric PAKE protocol using the $\mathcal{F}_{\mathsf{pwKE}}$ functionality is that there is no way to indicate to the environment through the $\mathcal{F}_{\mathsf{pwKE}}$ functionality whether the two sessions (one for each party) involved in the key exchange are both fresh.[15] Note that extending the functionality with queries that output the state of each session (i.e., fresh

[15] This seems critical, since after a password file compromise, it seems the adversary could compromise both client and server sessions in any symmetric PAKE functionality using the information obtained from the password file. (According to the $\mathcal{F}_{\mathsf{pwKE}}$

or $\mathsf{compromised}$) is ineffective since a real party would not know individually if its session was fresh or compromised, and thus could not output such an indication.[16] Thus allowing such a query in the ideal world makes it distinguishable from the real world.

Instead, we extend the functionality in the following way. Once a session is complete, the adversary may query the functionality with an extra value tr, which is output to the party as long as it meets the following condition: If either of the two sessions in the key exchange is not fresh, then the tr values output to each party must not be equal.

Since the query is not mandatory, any protocol that securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ will securely realize the functionality extended in this way. We conjecture something stronger, in that any protocol that securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ and in which each party is fresh and outputs a key if the adversary simply forwards messages between two parties in a session, and a party is not fresh if the adversary sends a message not output as the next message by the other party in a session, can be modified to securely realize the extended $\mathcal{F}_{\mathsf{pwKE}}$ functionality, in which each party outputs the tr value immediately after it outputs a key. The modification is to simply have each party output its transcript as tr once it has completed. Assuming the ideal adversary simulates the real protocol by running a simulated real protocol with the real adversary, the ideal adversary could also output the transcript of a simulated session, and this obviously would preserve indistinguishability between the ideal world and real world.

Explicit authentication. Although not critical to building an asymmetric PAKE functionality, the $\mathcal{F}_{\mathsf{pwKE}}$ functionality has another limitation in that it does not allow for some common types of explicit authentication. Specifically, a protocol that performs explicit authentication and aborts if the authentication fails, and otherwise sends one more message, will not securely realize the $\mathcal{F}_{\mathsf{pwKE}}$ functionality. Intuitively this is because an ideal adversary could not learn from the functionality whether the explicit authentication should be successful, and so could only guess whether to send the final message. (This would be easily distinguishable from the real protocol.) To allow for this type of authentication, one can add a new "test abort" query that tests whether the authentication would fail, informs the simulator, and informs the environment that the session is aborting in case of an authentication failure. A feature similar to this is mentioned in [14], in which the ideal adversary is notified if passwords for the two parties in a session do not match. This is claimed (rightly so) to weaken the definition, especially since an eavesdropper may not learn this. However, we are interested only in the case where the eavesdropper *does* learn this information, by noticing whether the protocol aborts or not. Thus in some sense we also

functionality, this would even allow the adversary to set the session keys between himself and each party to be the same.) As a man-in-the middle, he could simply forward the remaining messages between client and server sessions to complete the asymmetric PAKE protocol.

[16] A party may know if its session "succeeded" or not, but both fresh and compromised sessions may be successful.

"weaken" the definition of PAKE, but in a way that makes sense and allows some natural PAKE protocols (secure according to, say, an indistinguishability-based definition of PAKE) to securely realize the (extended) PAKE functionality. As above, since the "test abort" query is not mandatory, any protocol that securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ will securely realize the functionality extended in this way; i.e., this extended functionality does not imply a protocol must have explicit authentication, but only allows for it.

The Revised Symmetric PAKE Functionality. Figure 3 describes our *revised* symmetric password-based key exchange functionality called $\mathcal{F}_{\mathsf{rpwKE}}$ that includes a transcript query as discussed above, but not a test abort query.[17] More specifically, the revised functionality $\mathcal{F}_{\mathsf{rpwKE}}$ is exactly $\mathcal{F}_{\mathsf{pwKE}}$ (with a minor wording change in NewKey that has no effect on the $\mathcal{F}_{\mathsf{pwKE}}$), but with NewTranscript queries added. We note that it is possible to prove that a protocol securely realizing $\mathcal{F}_{\mathsf{rpwKE}}$ also is secure according to the definition of [3].

Asymmetric PAKE functionalities. Now we discuss our asymmetric PAKE functionality $\mathcal{F}_{\mathsf{apwKE}}$, which is presented in Figure 4. At a high level, this functionality expands on the $\mathcal{F}_{\mathsf{pwKE}}$ functionality to allow a server to store password data, and then use this stored password data for authentication instead of a password received as input from the environment. This functionality also accounts for the possibility that password data may be stolen by the adversary. As discussed above, this allows the adversary to perform an offline dictionary attack.[18] It also allows the adversary to impersonate the server, but not the client. In more detail, the changes are as follows.

- The $\mathcal{F}_{\mathsf{pwKE}}$ functionality was a single-session functionality. However, asymmetric PAKE requires that a password file be used across multiple sessions, so we define the $\mathcal{F}_{\mathsf{apwKE}}$ functionality as a multiple-session functionality. Note that this cannot be accomplished simply using "composition with joint state" [15] because the functionality itself requires shared state that needs to be maintained between sessions.
- In $\mathcal{F}_{\mathsf{pwKE}}$, sessions are started by sending NewSession queries to two parties, including a password in each query. In $\mathcal{F}_{\mathsf{apwKE}}$, these queries are replaced with CltSession and SvrSession queries. The CltSession queries include a password, but the SvrSession queries do not. The server password is taken from the password file, which is placed on the server using a StorePWfile query that includes the password. Note that if the server is corrupted when it receives this query, then the adversary learns the password. However, a trusted initial

[17] The test abort query is omitted since it is not integral to our result. A test abort query is included in our asymmetric PAKE functionality because it is necessary there to handle explicit aborting in our Ω-protocol. However, the transcript query is omitted there because it is not integral to our result.

[18] Note that because this functionality accounts for each offline password guess individually, it seems to require the random oracle model (or some similar idealized model) to be securely realized.

Functionality $\mathcal{F}_{\mathsf{rpwKE}}$

The functionality $\mathcal{F}_{\mathsf{rpwKE}}$ is parameterized by a security parameter κ. It interacts with an adversary S and a set of parties via the following queries:

Upon receiving a query $(\mathsf{NewSession}, sid, P_i, P_j, \pi, \mathrm{role})$ **from party** P_i**:**
Send $(\mathsf{NewSession}, sid, P_i, P_j, \mathrm{role})$ to S. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record (P_j, P_i, π'), then record (P_i, P_j, π) and mark this record fresh.

Upon receiving a query $(\mathsf{TestPwd}, sid, P_i, \pi')$ **from the adversary** S**:**
If there is a record of the form (P_i, P_j, π) which is fresh, then do: If $\pi = \pi'$, mark the record compromised and reply to S with "correct guess". If $\pi \neq \pi'$, mark the record interrupted and reply with "wrong guess".

Upon receiving a query $(\mathsf{NewKey}, sid, P_i, k)$ **from** S**, where** $|k| = \kappa$**:**
If there is a record of the form (P_i, P_j, π) that is not marked completed, then:
- If this record is compromised, or either P_i or P_j is corrupted, then output (sid, k) to player P_i.
- If this record is fresh, and there is a record (P_j, P_i, π') with $\pi' = \pi$, and a key k' was sent to P_j, and (P_j, P_i, π) was fresh at the time, then output (sid, k') to P_i.
- In any other case, pick a new random key sk' of length κ and send (sid, k') to P_i.

Either way, mark the record (P_i, P_j, π) as completed.

Upon receiving a query $(\mathsf{NewTranscript}, sid, P_i, tr)$ **from** S**:**
If there is a record of the form (P_i, P_j, π) that is marked completed, then:
- If (1) there is a record (P_j, P_i, π') for which tuple $(\mathsf{transcript}, sid, tr'')$ was sent to P_j, (2) either (P_i, P_j, π) or (P_j, P_i, π') was ever compromised or interrupted, and (3) $tr = tr''$, ignore the query.
- In any other case, send $(\mathsf{transcript}, sid, tr)$ to P_i.

Fig. 3. The revised password-based key-exchange functionality $\mathcal{F}_{\mathsf{rpwKE}}$

setup between the client and server is generally assumed, so this would not be a problem.[19]

– In $\mathcal{F}_{\mathsf{apwKE}}$, the adversary may "steal" the server's password data using a StealPWfile query. No actual data is sent to the adversary, but after this the adversary may make queries to test passwords using OfflineTestPwd queries. These queries are "offline" as they do not correspond to any sessions.
  The OfflineTestPwd queries may actually be made either before or after the StealPWfile query, but queries made before are not answered until the StealPWfile query is made. Specifically, when a StealPWfile query is made, if there was a previous OfflineTestPwd query with the correct password, that password is simply returned to $\mathcal{A}$.

[19] If one is concerned about this, then one could possibly change the StorePWfile to contain some data (perhaps a one-way function of the password), along with a way to verify passwords against this data. Our work focuses on the issue of password file compromise, so we did not explore these other issues.

We also change the UC framework slightly to allow the queries StealPWfile and OfflineTestPwd to be accounted for by the environment, similar to the way Corrupt queries are accounted for.[20] Specifically, a StealPWfile query by the adversary is not allowed until the environment sends a StealPWfile message to the adversary,[21] and similarly, each OfflineTestPwd query by the adversary is not allowed until the environment sends an OfflineTestPwd message to the adversary. It is easy to see that the composition theorem holds despite these changes.[22]

- In $\mathcal{F}_{\mathsf{apwKE}}$, in addition to TestPwd queries, an adversary can also make an Impersonate query to compromise a client session without supplying a password. This will succeed if there has already been a StealPWfile query.
- We add the TestAbort query to $\mathcal{F}_{\mathsf{apwKE}}$. The main reason is that our asymmetric PAKE protocol specifies some verifications, where an instance will abort if a verification fails. As discussed above, the $\mathcal{F}_{\mathsf{pwKE}}$ would need an extension to handle this type of protocol.
- In contrast to $\mathcal{F}_{\mathsf{rpwKE}}$, we did not include a transcript query in $\mathcal{F}_{\mathsf{apwKE}}$ – primarily because our proofs did not need it. The query could be added, and our asymmetric PAKE protocol could be modified to output a transcript.

Finally, in $\mathcal{F}_{\mathsf{apwKE}}$, we assume that a given sid is unique to a given client/server pair for which the server stores a password. We also assume that a given $(sid, ssid)$ pair is unique to a given session between the client and server corresponding to sid. These assumptions are valid in the UC framework, as discussed in [11].

5 The Ω protocol and its Security

We present the Ω-protocol in the UC framework in Figure 6. It uses the revised password-based key exchange functionality $\mathcal{F}_{\mathsf{rpwKE}}$ defined in Figure 3 and the random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ defined in Figure 5.[23] The random oracle functionality is parameterized by an output length ℓ, and for simplicity, this argument is implicit (albeit different depending on how the random oracle is being used). In particular if the arguments are $\langle sid, 1\rangle, \langle sid, 3\rangle, \langle sid, ssid, 2\rangle$, then the output length $\ell = \kappa$. If the arguments are $\langle sid, 2\rangle$, then $\ell = |sk|$. Finally, if the arguments are $\langle sid, ssid, 1\rangle$, then $\ell = |c|$ (i.e., the size of the ciphertext being encrypted).

When applying the Ω-method in the UC framework to obtain the protocol Ω, there were some issues that needed to be addressed.

[20] In fact, we could define these queries as Corrupt queries with certain parameters, which are handled by the functionality in certain specific ways, but we felt it was more clear to make them separate queries.

[21] Technically, this is enforced by the “control function” (see [11]).

[22] This is assuming that these messages are based on sid values, and the sid values used in the original and emulating protocol somehow correspond. This is the case, but we need to define it explicitly.

[23] Note in particular that $\mathcal{F}_{\mathsf{rpwKE}}$ has no access to $\mathcal{F}_{\mathsf{RO}}$, and thus a protocol securely realizing $\mathcal{F}_{\mathsf{rpwKE}}$ should have no access to $\mathcal{F}_{\mathsf{RO}}$.

Functionality $\mathcal{F}_{\mathsf{apwKE}}$

The functionality $\mathcal{F}_{\mathsf{apwKE}}$ is parameterized by a security parameter κ. It interacts with an adversary S and a set of parties via the following queries:

Password storage and authentication sessions

Upon receiving a query $(\mathsf{StorePWfile}, sid, P_i, \pi)$ **from party** P_j**:**
If this is the first StorePWfile query, store password data record $(\mathsf{file}, P_i, P_j, \pi)$ and mark it uncompromised.

Upon receiving a query $(\mathsf{CltSession}, sid, ssid, P_j, \pi)$ **from party** P_i**:**
Send $(\mathsf{CltSession}, sid, ssid, P_i, P_j)$ to S, and if this is the first CltSession query for $ssid$, store session record $(ssid, P_i, P_j, \pi)$ and mark it fresh.

Upon receiving a query $(\mathsf{SvrSession}, sid, ssid)$ **from party** P_j**:**
If there is a password data record $(\mathsf{file}, P_i, P_j, \pi)$, then send $(\mathsf{SvrSession}, sid, ssid, P_i, P_j)$ to S, and if this is the first SvrSession query for $ssid$, store session record $(ssid, P_j, P_i, \pi)$, and mark it fresh.

Stealing password data

Upon receiving a query $(\mathsf{StealPWfile}, sid)$ **from adversary** S**:**
If there is no password data record, reply to S with "no password file". Otherwise, do the following. If the password data record $(\mathsf{file}, P_i, P_j, \pi)$ is marked uncompromised, mark it as compromised. If there is a tuple $(\mathsf{offline}, \pi')$ stored with $\pi = \pi'$, send π to S, otherwise reply to S with "password file stolen".

Upon receiving a query $(\mathsf{OfflineTestPwd}, sid, \pi')$ **from adversary** S**:** If there is no password data record, or if there is a password data record $(\mathsf{file}, P_i, P_j, \pi)$ that is marked uncompromised, then store $(\mathsf{offline}, \pi')$. Otherwise, do: If $\pi = \pi'$, reply to S with "correct guess". If $\pi \neq \pi'$, reply with "wrong guess".

Active session attacks

Upon receiving a query $(\mathsf{TestPwd}, sid, ssid, P, \pi')$ **from adversary** S**:**
If there is a session record of the form $(ssid, P, P', \pi)$ which is fresh, then do: If $\pi = \pi'$, mark the record compromised and reply to S with "correct guess". Otherwise, mark the record interrupted and reply with "wrong guess".

Upon receiving a query $(\mathsf{Impersonate}, sid, ssid)$ **from adversary** S**:**
If there is a session record of the form $(ssid, P_i, P_j, \pi)$ which is fresh, then do: If there is a password data record $(\mathsf{file}, P_i, P_j, \pi)$ that is marked compromised, mark the session record compromised and reply to S with "correct guess", else mark the session record interrupted and reply with "wrong guess".

Key Generation and Authentication

Upon receiving a query $(\mathsf{NewKey}, sid, ssid, P, k)$ **from** S**, where** $|k| = \kappa$**:**
If there is a record of the form $(ssid, P, P', \pi)$ that is not marked completed, then:
- If this record is compromised, or either P or P' is corrupted, then output $(sid, ssid, k)$ to P.
- If this record is fresh, there is a session record $(ssid, P', P, \pi')$, $\pi' = \pi$, a key k' was sent to P', and $(ssid, P', P, \pi)$ was fresh at the time, then let $k'' = k'$, else pick a random key k'' of length κ. Output $(sid, ssid, k'')$ to P.
- In any other case, pick a random key k'' of length κ and output $(sid, ssid, k'')$ to P.

Finally, mark the record $(ssid, P, P', \pi)$ as completed.

Upon receiving a query $(\mathsf{TestAbort}, sid, ssid, P)$ **from** S**:**
If there is a record of the form $(ssid, P, P', \pi)$ that is not marked completed, then:
- If this record is fresh, there is a record $(ssid, P', P, \pi')$, and $\pi' = \pi$, let $b' = \mathsf{succ}$.
- In any other case, let $b' = \mathsf{fail}$.

Send b' to S. If $b' = \mathsf{fail}$, send $(\mathsf{abort}, sid, ssid)$ to P, and mark record $(ssid, P, P', \pi)$ completed.

Fig. 4. The Asymmetric PAKE functionality $\mathcal{F}_{\mathsf{apwKE}}$

Functionality $\mathcal{F}_{\mathsf{RO}}$

The functionality $\mathcal{F}_{\mathsf{RO}}$ is parameterized by an (implicit) output length ℓ.

Upon receiving query $(\mathsf{Hash}, sid, msg)$ **from any party** P **or adversary** S**:**
If a tuple (msg, r) is stored, return r; else, generate $r \xleftarrow{R} \{0,1\}^{\ell}$. Record (msg, r) and return r.

Fig. 5. The random oracle functionality $\mathcal{F}_{\mathsf{RO}}$

- In the Ω-protocol, one must use $(sid, ssid, tr)$ in place of the transcript of the symmetric PAKE protocol. This pair $(sid, ssid)$ is unique to a given pair of client/server instances, and these two instances only generate the same tr if they are fresh and use the same password. This is exactly what is needed in the proof of security, in particular, to ensure that a signature produced by a client cannot be used to impersonate a client in another session.
- We had wanted to use an ideal signature functionality instead of an explicit signature scheme. However, this does not seem possible, since the Ω-protocol explicitly encrypts and hashes the signing key, but the ideal signature functionality doesn't have any notion of a secret key.
- Similarly, it does not seem possible to use an ideal secure channel functionality in place of the symmetric encryptions, because keys are generated using a hash function. The ideal secure channel functionality does not have any notion of a secret key.

The following theorem characterizes the security of the Ω protocol and is proven in the full version of this paper.

Theorem 1. *Assume that $\mathcal{S}$ is an existentially unforgeable signature scheme. Then protocol Ω of Figure 6 securely realizes the $\mathcal{F}_{\mathsf{apwKE}}$ functionality in the $\mathcal{F}_{\mathsf{rpwKE}}, \mathcal{F}_{\mathsf{RO}}$-hybrid model, in the presence of static-corruption adversaries.*

References

1. M. Abdalla and D. Pointcheval. Simple Password-Based Encrypted Key Exchange Protocols. In *RSA Conference, Cryptographer's Track*, pp. 191–208, 2005
2. B. Barak, Y. Lindell, and T. Rabin. Protocol initialization for the framework of universal composability. In *Cryptology ePrint Archive*, Report 2004/006, `http://eprint.iacr.org/`, 2004.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT*, pp. 139–155, 2000.
4. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.
5. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO*, pp. 232–249, 1993.
6. M. Bellare, A. Boldyreva and A. Palacio. An Uninstantiable Random-Oracle-Model Scheme for a Hybrid-Encryption Problem. In *EUROCRYPT*, pp. 171–188, 2004.

UC Asymmetric PAKE Protocol Ω

Setup: This protocol uses a random oracle functionality $\mathcal{F}_{\mathsf{RO}}$ and a revised PAKE functionality $\mathcal{F}_{\mathsf{rpwKE}}$, as well as an existentially unforgeable signature scheme $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Verify})$.

Password Storage: When P_j is activated using $\mathsf{StorePWfile}(sid, P_i, w)$ for the first time, he does the following. He first sends $(\mathsf{Hash}, \langle sid, d\rangle, w)$ to the $\mathcal{F}_{\mathsf{RO}}$ functionality for $d \in \{1, 2\}$, and receives responses r and k_w. He generates a signature key pair $(sk, pk) \leftarrow \mathsf{Gen}(1^\kappa)$. Next he sends $(\mathsf{Hash}, \langle sid, 3\rangle, sk)$ to the $\mathcal{F}_{\mathsf{RO}}$ functionality and receives response h_{sk}. He computes $c = (k_w \oplus sk, h_{sk})$ and sets $\mathsf{file}[sid] = (r, c, pk)$.

Protocol Steps:

1. When P_j receives input $(\mathsf{SvrSession}, sid, ssid, P_i)$, he obtains r from the tuple stored in $\mathsf{file}[sid]$ (aborting if this value is not properly defined), sends $(\mathsf{NewSession}, \langle sid, ssid\rangle, P_j, P_i, r, \mathsf{server})$ to the $\mathcal{F}_{\mathsf{rpwKE}}$ functionality, and awaits a response.
2. When P_i receives input $(\mathsf{CltSession}, sid, ssid, P_j, w)$, he sends $(\mathsf{Hash}, \langle sid, 1\rangle, w)$ to the functionality $\mathcal{F}_{\mathsf{RO}}$ and obtains the response r. He then sends $(\mathsf{NewSession}, \langle sid, ssid\rangle, P_i, P_j, r, \mathsf{client})$ to the $\mathcal{F}_{\mathsf{rpwKE}}$ functionality and awaits a response.
3. When P_j (who is a server and is awaiting a response from $\mathcal{F}_{\mathsf{rpwKE}}$) receives responses $(\langle sid, ssid\rangle, k)$ and $(\mathsf{transcript}, \langle sid, ssid\rangle, tr)$, he does the following. First he sends $(\mathsf{Hash}, \langle sid, ssid, d\rangle, k)$ for $d \in \{1, 2\}$ to receive responses k' and k'' respectively. Then he retrieves c from the tuple $\mathsf{file}[sid]$. He encrypts $c' = k' \oplus c$ and sends the message $(\mathsf{flow\text{-}zero}, sid, ssid, c')$ to P_i.
4. When P_i (who is a client and is also awaiting a response from $\mathcal{F}_{\mathsf{rpwKE}}$) receives responses $(\langle sid, ssid\rangle, k)$ and $(\mathsf{transcript}, \langle sid, ssid\rangle, tr)$, he sends $(\mathsf{Hash}, \langle sid, ssid, d\rangle, k)$ for $d \in \{1, 2\}$ to the $\mathcal{F}_{\mathsf{RO}}$ functionality, and receives responses k' and k'' respectively.
5. When P_i receives a message $(\mathsf{flow\text{-}zero}, sid, ssid, c')$ he computes the decryption $c = k' \oplus c'$, and parses $c = (c_1, c_2)$. He then sends $(\mathsf{Hash}, \langle sid, 2\rangle, w)$ to the $\mathcal{F}_{\mathsf{RO}}$ functionality, and receives the response k_w. He computes $sk = k_w \oplus c_1$, sends $(\mathsf{Hash}, \langle sid, 3\rangle, sk)$ to $\mathcal{F}_{\mathsf{RO}}$, receives response h_{sk}, and verifies that $h_{sk} = c_2$. If not, he outputs $(\mathsf{abort}, \langle sid, ssid\rangle)$ and terminates the session. Otherwise he computes $s = \mathsf{Sig}_{sk}(\langle sid, ssid, tr\rangle)$, sends $(\mathsf{flow\text{-}one}, sid, ssid, s)$ to P_j, outputs $(sid, ssid, k'')$, and terminates the session.
6. When P_j receives a message $(\mathsf{flow\text{-}one}, sid, ssid, s)$, he checks that $\mathsf{Verify}_{pk}(\langle sid, ssid, tr\rangle, s) = 1$. If not, he outputs $(\mathsf{abort}, \langle sid, ssid\rangle)$ and terminates the session. Otherwise, he outputs $(sid, ssid, k'')$, and terminates the session.

Stealing the password file: When P_j (who is a server) receives a message $(\mathsf{StealPWfile}, sid, P_j, P_i)$, from the adversary $\mathcal{A}$, if $\mathsf{file}[sid]$ is defined, P_j sends it to $\mathcal{A}$.

Fig. 6. The UC Asymmetric PAKE Protocol Using the Ω-method

7. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symp. on Research in Security and Privacy*, pp. 72–84, 1992.
8. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *1st ACM Conf. on Computer and Communications Security*, pp. 244–250, 1993.
9. D. Bleichenbacher. Personal communication.
10. V. Boyko, P. MacKenzie, and S. Patel. Provably secure password authentication and key exchange using Diffie-Hellman. In *EUROCRYPT*, pp. 156–171, 2000.
11. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Cryptology ePrint Archive*, Report 2000/067. `http://eprint.iacr.org/`, 2005.
12. R. Canetti, O. Goldreich, and S. Halevi. On the random-oracle methodology as applied to length-restricted signature schemes. In *Theory of Cryptography Conference - TCC*, pp. 40–57, 2004.
13. R. Canetti, O. Goldreich and S. Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.
14. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally-composable password-based key exchange. In *EUROCRYPT*, pp. 404–421, 2005.
15. R. Canetti and T. Rabin. Universal Composition with Joint State In *CRYPTO*, pp. 265–281, 2003.
16. M. Di Raimondo and R. Gennaro. Provably Secure Threshold Password Authenticated Key Exchange. In *EUROCRYPT*, pp. 507–523, 2003.
17. W. Ford and B. S. Kaliski, Jr. Server-assisted generation of a strong secret from a password. In *5th IEEE International Workshop on Enterprise Security*, 2000.
18. R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. In *EUROCRYPT*, pp. 524–543, 2003.
19. C. Gentry, P. MacKenzie, and Z. Ramzan. Password Authenticated Key Exchange Using Hidden Smooth Subgroups. In *12th ACM Conf. on Computer and Communications Security*, pp. 299–309, 2005.
20. O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. In *CRYPTO*, pp. 408–432, 2001.
21. S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing* 17(2):281–308, April 1988.
22. S. Goldwasser and Y. Tauman Kalai. "On the (In)security of the Fiat-Shamir Paradigm." In *44th IEEE Symp. on Foundations of Computer Science (FOCS)*, pp. 102–115, 2003.
23. L. Gong. Optimal authentication protocols resistant to password guessing attacks. In *8th IEEE Computer Security Foundations Workshop*, pp. 24–29, 1995.
24. L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.
25. IEEE Standard 1363-2000, Standard specifications for public key cryptography, 2000.
26. D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review, ACM SIGCOMM*, 26(5):5–20, 1996.
27. D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *WETICE'97 Workshop on Enterprise Security*, 1997.
28. D. Jablon Password authentication using multiple servers. In *Proc. RSA Conference, Cryptographer's Track, 2001.*

29. S. Jiang and G. Gong. Password based key exchange with mutual authentication. In *Workshop on Selected Areas of Cryptography (SAC)*, 2004.
30. J. Katz, P. MacKenzie, G. Taban, V. Gligor. Two-Server Password-Only Authenticated Key Exchange. In *Applied Cryptography and Network Security, 3rd Intl. Conf. (ACNS 2005)*, pp. 1–16, 2005.
31. J. Katz, R. Ostrovsky, and M. Yung. Practical password-authenticated key exchange provably secure under standard assumptions. In *EUROCRYPT*, pp. 475–494, 2001.
32. C. Kaufmann and R. Perlman. PDM: A New Strong Password-Based Protocol. In *10th Usenix Security Symposium*, 2001.
33. T. Kwon. Authentication and Key Agreement via Memorable Passwords. In *Internet Society Network and Distributed System Security Symposium (NDSS)*, 2001.
34. S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proc. of the Workshop on Security Protocols*, 1997.
35. P. MacKenzie. More Efficient Password-Authenticated Key Exchange. In *RSA Conference, Cryptographer's Track*, pp. 361–377, 2001.
36. P. MacKenzie. The PAK suite: Protocols for password-authenticated key exchange. DIMACS Technical Report 2002-46, October, 2002.
37. P. MacKenzie, S. Patel, and R. Swaminathan. Password authenticated key exchange based on RSA. In *ASIACRYPT*, pp. 599–613, 2000.
38. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold Password-Authenticated Key Exchange. *J. Cryptology*, 19(1):27–66, 2006.
39. U. Maurer, R. Renner, and C. Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *Theory of Cryptography Conference - TCC*, pp. 21–39, 2004.
40. A. Narayanan and V. Shmatikov. Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff. In *ACM Conf. on Computer and Communications Security (CCS)*, pp. 364–372, 2005.
41. National Institute of Standards and Technology (NIST). Announcing the Secure Hash Standard, FIPS 180-1, U.S. Department of Commerce, April, 1995.
42. M. Nguyen and S. Vadhan. Simpler Session-Key Generation from Short Random Passwords. In *Theory of Cryptography Conference - TCC*, pp. 428–445, 2004.
43. J. B. Nielsen. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-Committing Encryption Case Jesper Buus Nielsen. In *CRYPTO*, pp. 111–126, 2002
44. P. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *CRYPTO*, pp. 617–630, 2003.
45. S. Patel. Number theoretic attacks on secure password schemes. In *IEEE Symposium on Research in Security and Privacy*, pages 236–247, 1997.
46. D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT*, pp. 387–398, 1996.
47. C. P. Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pp. 235–251, 1989.
48. M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *ACM Operating System Review*, 29:22–30, 1995.
49. T. Wu. The secure remote password protocol. In *Internet Society Network and Distributed System Security Symposium (NDSS)*, pages 97–111, 1998.
50. T. Wu. A real-world analysis of Kerberos password security. In *Internet Society Network and Distributed System Security Symposium (NDSS)*, February 1999.
51. M. Zhang. New Approaches to Password Authenticated Key Exchange Based on RSA. In *ASIACRYPT*, pp. 230–244, 2004.