Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords

Jonathan Katz[1] Rafail Ostrovsky[2] Moti Yung[3]

[1] Telcordia Technologies and
Department of Computer Science, Columbia University.
jkatz@cs.columbia.edu
[2] Telcordia Technologies, Inc., 445 South Street, Morristown, NJ 07960.
rafail@research.telcordia.com
[3] CertCo, Inc.
moti@cs.columbia.edu

Abstract. There has been much interest in password-authenticated key-exchange protocols which remain secure even when users choose passwords from a very small space of possible passwords (say, a dictionary of English words). Under this assumption, one must be careful to design protocols which cannot be broken using *off-line dictionary attacks* in which an adversary enumerates all possible passwords in an attempt to determine the correct one. Many heuristic protocols have been proposed to solve this important problem. Only recently have formal validations of security (namely, proofs in the idealized random oracle and ideal cipher models) been given for specific constructions [3, 10, 22].

Very recently, a construction based on general assumptions, secure in the standard model with human-memorable passwords, has been proposed by Goldreich and Lindell [17]. Their protocol requires no public parameters; unfortunately, it requires techniques from general multi-party computation which make it impractical. Thus, [17] only proves that solutions are possible "in principal". The main question left open by their work was finding an efficient solution to this fundamental problem.

We show an efficient, 3-round, password-authenticated key exchange protocol with human-memorable passwords which is provably secure under the Decisional Diffie-Hellman assumption, yet requires only (roughly) 8 times more computation than "standard" Diffie-Hellman key exchange [14] (which provides no authentication at all). We assume public parameters available to all parties. We stress that we work in the standard model only, and do not require a "random oracle" assumption.

1 Introduction

1.1 Background

Protocols which allow for mutual authentication of two parties and for generating a cryptographically-strong shared key between them (*authenticated key* *exchange*) underly most interactions taking place on the Internet. The importance of this primitive has been realized for some time by the security community (see [11] for exhaustive references), followed by an increasing recognition that precise definitions and formalization were needed. The first formal treatments [4, 6, 2, 20, 9, 28, 11] were in a model in which participants already share some cryptographically-strong information: either a secret key which can be used for encryption/authentication of messages, or a public key which can be used for encryption/signing of messages. The setting arising most often in practice — in which (human) users are only capable of storing "human-memorable" passwords (*password-authenticated key exchange*) — remains much less studied, though many heuristic protocols exist. Indeed, only recently have formal definitions of security for this setting appeared [3, 10, 22, 17].

The problem (in the standard model; i.e., without random oracles) is difficult precisely because it requires "bootstrapping" from a weak shared secret to a strong one. In fact, it is not even *a priori* clear that a solution is possible. Completeness results for multi-party computation [18] do not directly apply here due to the strong adversarial model considered (see Section 2). In particular, the adversary may ask for concurrent (arbitrarily-interleaved) executions of the protocol, may modify messages or even prevent their delivery, may impersonate participants in the protocol and act as a "man-in-the-middle", and may corrupt *all* protocol participants. Nevertheless, in a very recent paper, Goldreich and Lindell [17] have shown that in principle, this problem is solvable based on any trapdoor permutation (leaving open the question of whether a practical solution is possible). We show, perhaps somewhat surprisingly, the existence of an *efficient* solution for human-memorable passwords under the Decisional Diffie-Hellman assumption.

1.2 The Adversarial Model

The setting is as follows (a formal discussion appears in Section 2): two parties within a larger network who share a *weak* (low-entropy) password wish to authenticate each other and generate a strong session key for protecting their subsequent communication. An adversary controls all communication in the network. Thus, messages may be tampered with, delivered out-of-order, or not delivered at all; the adversary may also ask for arbitrarily-interleaved executions of the protocol. Finally, the adversary may corrupt selected instances (see below) of the participants and obtain the session keys generated by successful executions of the protocol. The adversary succeeds if he can cause a participant to compute a session key which the adversary can then distinguish from random.

Since the space of possible passwords is small, an adversary who has monitored a conversation may enumerate all possible passwords and try to match the recorded conversation to each one. As an example, any challenge-response protocol in which one party sends challenge N and the other responds with $f(\mathsf{password}, N)$ is trivially susceptible to this attack, regardless of f (note that such an attack is *not possible* by a poly-time adversary, for appropriate choice of f, when the parties share a *high-entropy* password). Additionally, the fact that the adversary can corrupt instances and determine the actual session key means that the protocol must ensure consistency between the recorded conversation and these session keys, even while not revealing any information about the password. These complications make this problem much harder than the case in which participants already share a strong key at the outset of the protocol.

What does security mean in a model which is inherently insecure? Indeed, since passwords are chosen from a small space, an adversary can always try each possibility one at a time in an impersonation (on-line) attack. Thus, we say a protocol is secure (informally) if this exhaustive guessing is the best an adversary can do. For a real-world adversary, such on-line attacks are the hardest to mount, and they are also the easiest to detect. It is very realistic to assume that the number of on-line attacks an adversary is allowed is severely limited, while other attacks (eavesdropping, off-line password guessing) are not.

1.3 Previous Work

The problem of off-line attacks in password-authenticated protocols was first noted by Bellovin and Merritt [7], followed by a flurry of work in the security community providing additional solutions with heuristic arguments for their security (see [11] for exhaustive references). More recently, two formal models for password-authenticated key exchange have been proposed: one by Bellare, Pointcheval, and Rogaway [3], based on [4, 6] with extensions suggested by [21]; and a second by Boyko, MacKenzie, and Patel [10], following [2] with extensions given in [28]. While both models have their advantages, we choose to work in the first model and review the appropriate definitions in Section 2.

These models all assume that two parties wishing to communicate share only a human-memorable password; in particular, they do not assume a public-key infrastructure (PKI) which allows participants to generate and share public keys. Definitions for security in this setting have also been proposed [20, 9, 28] and, in fact, the first protocols resistant to off-line dictionary attacks were given in this model. However, the requirement of a secure PKI is a strong one, and we wish to avoid it.

Only recently have formal validations of security for specific protocols appeared [3, 10, 22]. However, these validations are not proofs in the standard model; [3] relies on ideal ciphers, while [10, 22] rely on random oracles. More recently, Goldreich and Lindell [17] have shown a protocol based on general assumptions which is secure in the standard model. Interestingly, in contrast to the present work, their protocol does *not* require public parameters. Unfortunately, their construction requires a non-constant number of rounds and also requires techniques from generic multi-party computation [18]. Thus, their scheme serves as a general plausibility result (a terminology coined in [16]), but is much too inefficient for practical use. Finally, as pointed out by the authors themselves, the solution of [17] does not allow for concurrent executions of the protocol between parties using the same password.

1.4 Our Contribution

Security validation via proofs in the random oracle and ideal cipher models are useful, as they lend a measure of confidence to protocols whose security would otherwise be only heuristic. On the other hand, proofs of security in these models do not necessarily translate to real-world security [12], so it is important to have proofs under standard cryptographic assumptions. We prove the security of our construction using only the Decisional Diffie-Hellman (DDH) assumption.

Efficiency is especially important in this setting, where security concerns are motivated by very practical considerations (human users' inability to remember long secrets). We stress that our scheme, though provably secure, is very practical even when compared to heuristically-secure protocols such as [3, 10] or the original Diffie-Hellman protocol [14] (which does not provide *any* authentication). Our protocol requires only three rounds and has communication and computational complexity only (roughly) 5-8 times greater than the above solutions. Furthermore, we are able to construct our scheme without making stronger assumptions (the DDH assumption is used in [14, 3, 10]).

Although our solution relies on public-key techniques (in fact, this is necessary [20]) we emphasize that our protocol is not a "public-key solution" (as in [2, 20, 9]). In particular, we do not require *any* participant to have a public key, but instead rely on one set of common parameters shared by everyone in the system. This avoids problems associated with public key infrastructures (such as revocation, centralized trust, key management issues, etc.), and also allows new servers and clients to join the network *at any time* during execution of the protocol without requiring access to an on-line, centralized (trusted) authority (in fact, they do not even need to inform anyone else of their presence). Furthermore, no participants know the "secret key" associated with the public parameters. This eliminates the risk that compromise of a participant will compromise the security of the entire system.

The construction given here is secure under both the notion of basic security and the stronger notion of "forward security" (in the weak corruption model). In this initial version we concentrate on basic security only, and leave the topic of forward security for the final version.

2 Model and Definitions

The reader is assumed to be familiar with the model of [3], which is the model in which we prove security of our protocol. For completeness, we review the main points of their definition here, and refer the reader to [3] for more details.

PRINCIPALS, PASSWORDS, AND INITIALIZATION. We have a fixed set of protocol participants (*principals*) each of which is either a client $C \in \mathsf{Client}$ or a server $S \in \mathsf{Server}$ (Client and Server are disjoint). We let $\mathsf{User} \stackrel{\text{def}}{=} \mathsf{Client} \cup \mathsf{Server}$. Each $C \in \mathsf{Client}$ has a password pw_C. Each $S \in \mathsf{Server}$ has a vector $PW_S = \langle pw_C \rangle_{C \in \mathsf{Client}}$ which contains the passwords of each of the clients (we assume that all clients share passwords with all servers). Recall that pw_C is what client C remembers to log in; therefore, it is assumed to be chosen from a relatively small space of possible passwords.

Before the protocol is run, an initialization phase occurs during which public parameters are set and passwords are chosen for each client. We assume that passwords for each client are chosen independently and uniformly[1] at random from the set $\{1, \ldots, N\}$, where N is a constant, independent of the security parameter.

Execution of the Protocol. In the real world, protocol P determines how principals behave in response to signals (input) from their environment. Each principal is able to execute the protocol multiple times with different partners; this is modeled by allowing each principal an unlimited number of *instances* in which to execute the protocol (see [6]). We denote instance i of user U as Π_U^i. A given instance is used only once. The adversary is assumed to have complete control over all communication in the network. Thus, the adversary's interaction with the principals is modeled via access to oracles whose inputs may range over $U \in \mathsf{User}$ and $i \in \mathbb{N}$; this allows the adversary to "interact with" different instances. Global state is maintained throughout the entire execution for each instance with which the adversary interacts (this global state is not directly visible to the adversary); the global state for an instance may be updated by an oracle during an oracle call, and the oracle's output may depend upon this state. The oracle types, as defined in [3], are:

- $\mathsf{Send}(U, i, M)$ — This sends message M to instance Π_U^i. The oracle runs this instance as in a real execution, maintaining state as appropriate. The output of Π_U^i is given to the adversary in addition to other information; see [3].
- $\mathsf{Execute}(C, i, S, j)$ — This oracle executes the protocol between instances Π_C^i and Π_S^j, where $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$, and outputs a transcript of this execution. This transcript includes everything an adversary would see when eavesdropping on a real-world execution of the protocol, as well as other information; see [3].
- $\mathsf{Reveal}(U, i)$ — This outputs the session key sk_U^i (stored as part of the global state) of instance Π_U^i.
- $\mathsf{Test}(U, i)$ — This query is allowed only once, at any time during the adversary's execution. A random bit b is generated; if $b = 1$ the adversary is given sk_U^i, and if $b = 0$ the adversary is given a random session key.

Advantage of the Adversary. Event Succ occurs (adversary $\mathcal{A}$ *succeeds*) if she asks a single Test query, outputs a bit b', and $b' = b$ (where b is the bit chosen by the Test oracle). The advantage of $\mathcal{A}$ in attacking protocol P, is defined as as $\mathsf{Adv}_{P,\mathcal{A}}^{\mathrm{ake}} \overset{\mathrm{def}}{=} 2\Pr[\mathsf{Succ}] - 1$. If the adversary were unrestricted, success would be trivial (since the adversary could submit a Reveal query for the same instance submitted to the Test oracle). Clearly, some restrictions must be imposed. Before describing these, we formalize the idea of *partnering*. Intuitively, instances Π_U^i

[1] This is for ease of presentation only, as our analysis can be extended easily to handle arbitrary distributions, including users with inter-dependent passwords.

and $\Pi^j_{U'}$ are partnered if they have jointly run protocol P. Formally, we define a *session-id* (*sid*) for each instance, and say that two instances are partnered if they hold the same *sid* (which is not null). Here, we define the *sid* as the concatenation of all messages sent and received by an instance (i.e., a transcript of the execution). The following restriction may now be imposed on an adversary whose Test query is (U, i): that a Reveal query may not be called on (U, i) or on (U', j), where $\Pi^j_{U'}$ is partnered with Π^i_U. Furthermore, instance Π^i_U must have completed execution, and therefore have a non-null session key defined.

A poly-time adversary will be able to break any protocol by attempting to impersonate a user and trying all passwords one-by-one (the size of the password space is independent of the security parameter — indeed, this is what distinguishes the problem from that of [4, 6]). So, we say that a given protocol is *secure* when this kind of attack is the best an adversary can do. More formally, let q_{send} be the number of calls the adversary makes to the Send oracle. A protocol is secure if, when passwords are chosen from a dictionary of size N, the adversary's advantage in attacking the protocol is bounded by

$$\mathcal{O}(q_{\text{send}}/N) + \varepsilon(k),$$

for some negligible function $\varepsilon(\cdot)$. The first term represents the fact that the adversary can (essentially) do no better than guess a password during each call to the Send oracle[2]. In particular, even polynomially-many calls to the Execute oracle (i.e., passive observations of valid executions) and the Reveal oracle (i.e., compromise of short-term session keys) are of no help to an adversary; only online impersonation attacks (which are harder to mount and easier to detect) give the adversary a non-negligible advantage.

Concrete security is particularly important in this setting since the adversary's advantage is non-negligible (assuming Send queries are made). We quantify an adversary's maximum advantage as a function of the adversary's running time t and the number of queries made to the Send, Execute, and Reveal oracles ($q_{\text{send}}, q_{\text{execute}}$, and q_{reveal} respectively).

3 A Provably Secure Protocol for password-AKE

3.1 Building Blocks

Our protocol and proof of security rely on a number of building blocks. First, our protocol uses the Cramer-Shoup cryptosystem [13] which is secure under adaptive chosen-ciphertext attack. Actually, we require an extension of the Cramer-Shoup cryptosystem, which remains secure under adaptive chosen-ciphertext attack. Our extension defines two "types" of encryption: client-encryption and

[2] A tighter definition of security would require that the adversary's advantage be bounded by $q_{\text{send}}/rN + \varepsilon(k)$, where r is the minimum number of messages an adversary needs to send in order to cause (completion of the protocol and) a non-null session key to be defined. An analysis of our proof proof indicates that the security of our construction is indeed tight in this respect.

server-encryption. Details appear in Appendix B. We will also need a one-time signature scheme [15] secure against existential forgery [19]. Finally, our proof of security relies on the Decisional Diffie-Hellman (DDH) assumption [14, 8] (note that the security of the Cramer-Shoup cryptosystem requires the DDH assumption already). We review these components in Appendix A, and also explicitly quantify their (in)security which is necessary for an explicit analysis of the adversary's maximum advantage in attacking the key exchange protocol.

Chosen-ciphertext-secure encryption has been used previously in the context of secure key exchange [2, 20, 9, 28]. However, as pointed out above, our protocol differs from these works in that it does *not* require the assumption of a public-key infrastructure, and no participant holds a secret key or publishes a public key. Indeed, "decryption" is never performed during execution of our protocol.

3.2 The Protocol

A high-level description of the protocol is given in Figure 1. Let p, q be primes such that $q|p-1$, and let $\mathcal{G}$ be a subgroup of $\mathbb{Z}_p^*$ of order q in which the DDH assumption holds. During the initialization phase, generators $g_1, g_2, h, c, d \in \mathcal{G}$ and a function $\mathcal{H}$ from a family of universal one-way hash functions [23] (which can be based on any one-way function [26]) are chosen at random and published. Note that this public information is *not* an added assumption[3]; "standard" Diffie-Hellman key exchange [14] typically assumes that parties use a fixed generator g (although this is not necessary), and [3, 10] seem to require a public generator g for their proofs of security. However, we *do* require that no one know the discrete logarithms of any of the generators with respect to any other, and thus we need either a trusted party who generates the public information or else a source of randomness which can be used to publicly derive the information.

As part of the initialization phase, passwords are chosen randomly for each client. We assume that all passwords lie in (or can be mapped to) $\mathbb{Z}_q$. For typical values of $|q|$, this will be a valid assumption for human-memorable passwords.

Execution of the protocol is as follows (see Figure 1): When client C wants to connect to server S, the client first runs the key generation algorithm for the one-time signature scheme, giving VK and SK. Then, the client computes a client-encryption (see Appendix B) of $g_1^{pw_C}$. This, along with the client's name, is sent to the server as the first message. The server chooses random elements x_2, y_2, z_2, w_2 from $\mathbb{Z}_q$, computes α' using the first message, and forms $g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2}$. The server then computes a server-encryption (see Appendix B) of $g_1^{pw_C}$. This is sent back to the client as the second message. The client selects random elements x_1, y_1, z_1, w_1 from $\mathbb{Z}_q$, computes β' using the second message, and forms $K = g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta'})^{w_1}$. Finally, β' and K are signed using the signing key which was generated in the first step. The *sid* is defined as the transcript of the entire conversation. A formal description of the protocol appears in Appendix C.

[3] The protocols of [22, 17], however, do not require any public information.

Public information: $p, q, g_1, g_2, h, c, d, \mathcal{H}$

Client		Server
$(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{SigGen}(1^k)$		
$r_1 \longleftarrow \mathbb{Z}_q$		
$A = g_1^{r_1}; B = g_2^{r_1}$		
$C = h^{r_1} g_1^{pw_C}$		
$\alpha = \mathcal{H}(\mathit{Client} \mid \mathsf{VK} \mid A \mid B \mid C)$		
$D = (cd^{\alpha})^{r_1}$	$\xrightarrow{\mathit{Client} \mid \mathsf{VK} \mid A \mid B \mid C \mid D}$	
		$x_2, y_2, z_2, w_2, r_2 \longleftarrow \mathbb{Z}_q$
		$\alpha' = \mathcal{H}(\mathit{Client} \mid \mathsf{VK} \mid A \mid B \mid C)$
		$E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2}$
		$F = g_1^{r_2}; G = g_2^{r_2}$
		$I = h^{r_2} g_1^{pw_C}$
		$\beta = \mathcal{H}(\mathit{Server} \mid E \mid F \mid G \mid I)$
	$\xleftarrow{\mathit{Server} \mid E \mid F \mid G \mid I \mid J}$	$J = (cd^{\beta})^{r_2}$
$x_1, y_1, z_1, w_1 \leftarrow \mathbb{Z}_q$		
$\beta' = \mathcal{H}(\mathit{Server} \mid E \mid F \mid G \mid I)$		
$K = g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta'})^{w_1}$		
$\mathsf{Sig} = \mathsf{Sign}_{\mathsf{SK}}(\beta' \mid K)$	$\xrightarrow{K \mid \mathsf{Sig}}$	
		if $\mathsf{Verify}_{\mathsf{VK}}((\beta \mid K), \mathsf{Sig}) = 1$
		$C' = C / g_1^{pw_C}$
$I' = I / g_1^{pw_C}$		$sk_S = K^{r_2} A^{x_2} B^{y_2} (C')^{z_2} D^{w_2}$
$sk_C = E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}$		else $sk_S \leftarrow \mathcal{G}$

Fig. 1. The protocol for password-AKE. See text for details.

The protocol description in Figure 1 omits many implementation details which are important for the proof of security to hold. Most important is for both client and server to perform a "validity check" on the messages they receive. In particular, each side should check that the values they receive are actually in the group $\mathcal{G}$ and are not the identity (in other words, it is required to check that the group elements indeed have order q). Note that such validity checks are required even for chosen-ciphertext security of the underlying Cramer-Shoup cryptosystem.

Correctness. In an honest execution of the protocol, C and S calculate identical session keys. To see this, first note that $\alpha = \alpha'$ and $\beta = \beta'$ in an honest execution. Then:

$$sk_C = (g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2})^{r_1} g_1^{r_2 x_1} g_2^{r_2 y_1} h^{r_2 z_1} (cd^{\beta})^{r_2 w_1}$$

and

$$sk_S = (g_1^{x_1} g_2^{y_1} h^{z_1} (cd^\beta)^{w_1})^{r_2} g_1^{r_1 x_2} g_2^{r_1 y_2} h^{r_1 z_2} (cd^\alpha)^{r_1 w_2},$$

and one can verify that these are equal.

Mutual Authentication. We note that the protocol as presented above achieves key exchange only, and not mutual authentication. However, we can trivially add mutual authentication by adding a fourth message to the protocol. Details will appear in the final version.

3.3 Practical Considerations

In practice, a collision resistant hash function (say, SHA-1) can be used instead of a universal one-way hash function. This has the advantage of increased efficiency, at the expense of requiring a (possibly) stronger assumption for security.

Efficient one-time signatures [15] can be based on (presumed) one-way functions like SHA-1 or DES. In particular, one-time signatures are much more efficient than signature schemes which are secure against adaptive (polynomially-many) chosen message attacks.

Client computation can be reduced (which is important when the client is smartcard-based) as follows: instead of using a one-time signature scheme where fresh keys need to be generated each time a connection is made, a signing key/verification key can be generated once (upon initialization) and used for the lifetime of the client. Particularly suited for such applications are "on-the-fly" signature schemes such as [27, 24, 25]. This initialization step may be done by a host computer (with the keys then downloaded to the smartcard) or this step may be done off-line before the first connection is made. The proof of security given in Section 4 still holds. The disadvantage is that this signature scheme is now required to be secure against existential forgeries even when polynomially-many messages are signed (and not just a single message). In some cases, however, this tradeoff may be acceptable.

Finally, note that we may store $g_1^{pw_C}$ at the server instead of pw_C and thereby avoid computing the exponentiation each time the protocol is executed.

4 Security of the Protocol

We concentrate here on the basic security of the protocol, and leave the corresponding results about forward security to the full paper. The following theorem indicates that the protocol is secure, since all lower order terms are negligible in k (see Appendix A for definitions of the lower order terms).

Theorem 1. *Let P be the protocol of Figure 1, where passwords are chosen from a dictionary of size N, and let $k = |q|$ be the security parameter. Let $\mathcal{A}$ be an adversary which runs in time t and asks $q_{\text{execute}}, q_{\text{send}}$, and q_{reveal} queries to the respective oracles. Then:*

$$\mathsf{Adv}^{\mathrm{ake}}_{P,\mathcal{A}} < \frac{q_{\mathrm{send}}}{2N} + 2q_{\mathrm{send}}\varepsilon_{\mathrm{sig}}(k,t) + 2\varepsilon_{\mathrm{ddh}}(k,t) + 2q_{\mathrm{send}}\varepsilon_{\mathrm{cs}}(k,t,q_{\mathrm{send}}/2) + \frac{min\{2q_{\mathrm{reveal}}, q_{\mathrm{send}}\}}{q} + \frac{2min\{q_{\mathrm{reveal}}, q_{\mathrm{execute}}\}}{q^2}.$$

It will be helpful to develop some intuition and notation before presentation of the full proof. First, note that the $\mathsf{Execute}$ oracle cannot help the adversary. The reason is that Diffie-Hellman key exchange [14] forms the "heart" of this protocol, and this is secure under a passive attack.

Next, consider active "impersonation attacks" by the adversary. The protocol has three flows. When an adversary tries to impersonate a client (in an attempt to determine the eventual session key of a server), the adversary must send the first and third messages; when the adversary wants to impersonate a server (in an attempt to determine the eventual session key of a client), the adversary must "prompt" the client to generate the first message and must then send the second message. Consider an adversary impersonating a client, and let the first message (which comes from the adversary) be $\langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$. We say this message is *valid* if:

$$\log_{g_1} A = \log_{g_2} B = \log_h(C/g_1^{pw_C}) = \log_{cd^{\alpha'}} D, \tag{1}$$

where $\alpha' = \mathcal{H}(\mathit{Client}, \mathsf{VK}, A, B, C)$, and pw_C is the password for *Client.* We define *valid* analogously for the second message of an adversary impersonating a server (note that here the password which determines validity depends upon the name of the client to which the adversary sends the message). We do not define any notion of validity for the third message. The following fact is central to our proof:

Fact 1 *When an invalid message is sent to an instance, the session key computed by that instance is* information-theoretically *independent of all messages sent and received by that instance. This holds for both clients and servers.*

Proof. Consider the case of an adversary interacting with a server, with the first message as above. Let $\theta_1 \stackrel{\mathrm{def}}{=} \log_{g_1} g_2; \theta_2 \stackrel{\mathrm{def}}{=} \log_{g_1} h$; and $\theta_3 \stackrel{\mathrm{def}}{=} \log_{g_1}(cd^{\alpha'})$. Consider the random values x_2, y_2, z_2, w_2 (see Figure 1) used by the server instance during its execution. Element E of the second message constrains these values as follows:

$$\log_{g_1} E = x_2 + y_2\theta_1 + z_2\theta_2 + w_2\theta_3. \tag{2}$$

The session key is calculated as K^{r_2} multiplied by $sk'_S = A^{x_2}B^{y_2}(C/g_1^{pw_C})^{z_2}D^{w_2}$. But we have:

$$\log_{g_1} sk'_S = x_2\log_{g_1} A + y_2\theta_1\log_{g_2} B + z_2\theta_2\log_h(C/g_1^{pw_C}) + w_2\theta_3\log_{cd^{\alpha'}} D. \tag{3}$$

When equation (1) does not hold (i.e., the message is invalid), equations (2) and (3) are linearly independent and $sk'_S \in_R \mathcal{G}$ is information-theoretically independent of the transcript of the execution. A similar argument holds for the case of an adversary interacting with a client. ■

Let Π_U^i be an instance to which the adversary has sent an invalid message. Fact 1 implies that the adversary has advantage 0 in distinguishing the session key generated by this instance from a random session key. Thus, an adversary's (non-zero) advantage can come about only by sending a valid message to an instance.

We call a message sent by an adversary *previously-used* if the message was previously output by a client or server running the protocol (that is, the adversary has simply "copied" and re-used the message), and is *new* otherwise. The following lemma bounds the adversary's probability of coming up with a new, valid first or second message:

Lemma 1. *An adversary's probability of sending, at any point during the protocol, a first or second message which is both new and valid is bounded by $\mathcal{O}(q_{\text{send}}/N) + \varepsilon(k)$, for some negligible function $\varepsilon(\cdot)$.*

This lemma essentially follows from the chosen-ciphertext security (and hence non-malleability) of extended Cramer-Shoup encryption (see [13] and Appendix B). Detail appear in the full proof, below. The lemma reflects the fact that the adversary can (trivially) "guess" the appropriate password[4] each time he sends a first or second message.

The only remaining point to argue is that previously-used messages cannot significantly help the adversary. First note that if an adversary re-uses a first message, the adversary will (with high probability) not be able to compute a valid signature to include with the third message. If an adversary re-uses a second message, the full proof indicates that without knowing the randomness used to generate that message, the adversary will gain only negligible advantage.

Proof (of Theorem 1). We refer to the formal specification of the protocol as it appears in Appendix C. The number of clients and servers is polynomial in the security parameter, and this number is fixed in advance[5] and public.

We imagine a simulator who controls all oracles to which the adversary has access. The simulator runs the protocol initialization as described in Appendix C, Figure 2, including selecting passwords for each client[6]. The simulator answers the adversary's oracle queries as defined in Appendix C, Figures 3 and 4. The adversary succeeds if it can guess the bit b that the simulator uses during the Test query (see Section 2 for additional details).

We define a sequence of transformations $P_1, \ldots$ to the original protocol P_0, and bound the effect each transformation has on the adversary's advantage.

[4] The lemma assumes that passwords are chosen uniformly at random from the password space, but can be appropriately modified to handle arbitrary distributions.

[5] As mentioned in Section 1.4, clients and servers can in fact be dynamically added to the protocol *during execution* at the request of the adversary (and even with passwords chosen by the adversary, when forward security is considered). For simplicity, we focus on the static case.

[6] For simplicity we assume that users choose passwords independently and with uniform distribution. The analysis can easily be modified to accommodate arbitrary distributions.

Then, we bound the adversary's advantage in the final (transformed) protocol; this gives an explicit bound on the adversary's advantage in the original protocol.

Consider the verification keys output by the Send_0 oracle during the course of the protocol. We may restrict ourselves to the case where the adversary is unable to forge a new message/signature pair for any of these keys during the course of the protocol. This can change the adversary's success probability (as a simple hybrid argument shows) by at most $q_{\text{send}_0}\varepsilon_{\text{sig}}(k,t) \leq q_{\text{send}}\varepsilon_{\text{sig}}(k,t)$.

In protocol P_1, calls to the $\mathsf{Execute}$ oracle are answered as before, except that C and I are chosen at random from $\mathcal{G}$. The following bounds the effect on the adversary's advantage:

Lemma 2. *The adversary's success probability in P_1 differs by at most $\varepsilon_{\text{ddh}}(k,t)$ from its advantage in P_0.*

Proof. The simulator uses the adversary as a black box to distinguish Diffie-Hellman quadruples from random quadruples. Given quadruple (g,h,s,t) and group $\mathcal{G}$, it runs the initialization as follows:

$a, b, \ell \leftarrow \mathbb{Z}_q$
$g_1 = g; g_2 = g^a; c = g^b, d = g^\ell$
$\mathcal{H} \leftarrow \mathsf{UOWH}$
Publish parameters $(q, g_1, g_2, h, c, d, \mathcal{H})$ and group $\mathcal{G}$
$\langle pw_C \rangle_{C \in \mathsf{Client}} \leftarrow \{1, \ldots, N\}$

By a random self-reducibility property [28, 1], the simulator can generate s_T, t_T (for $T = 1, \ldots$) such that, if (g,h,s,t) is a Diffie-Hellman quadruple, so is (g,h,s_T,t_T); on the other hand, if (g,h,s,t) is a random quadruple, then (g,h,s_T,t_T) is distributed among random quadruples with g and h fixed. The T-th call to $\mathsf{Execute}$ is answered as:

```
$\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$ —
    $(\mathsf{VK},\mathsf{SK}) \overset{R}{\longleftarrow} \mathsf{SigGen}(1^k)$    $x_1, x_2, y_1, y_2, z_1, z_2, w_1, w_2 \overset{R}{\longleftarrow} \mathbb{Z}_q$
    $A = s_{2T};\ B = s_{2T}^a;\ C = t_{2T} \cdot g_1^{pw_C}$    $\alpha = \mathcal{H}(\mathit{Client} \mid \mathsf{VK}|A|B|C)$
    $D = s_{2T}^{b+\alpha\ell}$    $\mathit{msg\text{-}out}_1 \longleftarrow \langle \mathit{Client} \mid \mathsf{VK} \mid A \mid B \mid C \mid D \rangle$
    $E = g_1^{x_1} g_2^{x_1} h^{z_1} (cd^\alpha)^{w_1}$    $F = s_{2T+1};\ G = s_{2t+1}^a;\ I = t_{2T+1} \cdot g_1^{pw_C}$
    $\beta = \mathcal{H}(\mathit{Server}|E|F|G|I)$
    $J = s_{2T+1}^{b+\beta\ell}$    $\mathit{msg\text{-}out}_2 \longleftarrow \langle \mathit{Server} \mid E \mid F \mid G \mid I \mid J \rangle$
    $K = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^\beta)^{w_2}$    $\mathit{msg\text{-}out}_3 \longleftarrow \langle K \mid \mathsf{Sign}_{\mathsf{SK}}(\beta|K) \rangle$
    $sk_S^j \longleftarrow sk_C^i \longleftarrow A^{x_1} B^{y_1} (C \cdot g_1^{-pw_C})^{z_1} D^{w_1} F^{x_2} G^{y_2} (I \cdot g_1^{-pw_C})^{z_2} J^{w_2}$
    $sid_S^j \longleftarrow sid_C^i \longleftarrow \langle \mathit{msg\text{-}out}_1 \mid \mathit{msg\text{-}out}_2 \mid \mathit{msg\text{-}out}_3 \rangle$
    return $\langle \mathit{msg\text{-}out}_1, \mathit{msg\text{-}out}_2, \mathit{msg\text{-}out}_3 \rangle$
```

If (g,h,s,t) is a Diffie-Hellman quadruple, this is an exact simulation of P_0; on the other hand, if it is a random quadruple, this is an exact simulation of P_1. ■

In protocol P_2, calls to $\mathsf{Execute}$ are answered as before except that the session key is chosen randomly from $\mathcal{G}$. The adversary's view (and thus its success probability) is within statistical distance $\min\{q_{\text{reveal}}, q_{\text{execute}}\}/q^2$ from the adversary's view in protocol P_1. Indeed, Fact 1 shows that the session key is independent of the transcript of the execution seen by the adversary whenever *msg-out*$_1$ or *msg-out*$_2$ are not valid (for the appropriate password). But when C and I are chosen randomly, the probability that both *msg-out*$_1$ and *msg-out*$_2$ are valid is exactly $1/q^2$.

In protocol P_3, the public parameters are generated by choosing g_1 and g_2 randomly from $\mathcal{G}$, then choosing x_1, x_2, y_1, y_2, and z randomly from $\mathbb{Z}_q$ and setting $c = g_1^{x_1} g_2^{x_2}$, $d = g_1^{y_1} g_2^{y_2}$, and $h = g_1^z$; $\mathcal{H}$ is chosen as before. Furthermore, the Send_3 oracle is changed as follows: the simulator first checks whether *first-msg-in* (which was the message sent to the Send_1 oracle for the same instance) is previously-used (see above). If so, the current query to the Send_3 oracle is answered normally. Otherwise, let *first-msg-in* $= \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$. The simulator computes $\alpha = \mathcal{H}(\mathit{Client}|\mathsf{VK}|A|B|C)$ and checks whether $A^{x_1+y_1\alpha} B^{x_2+y_2\alpha} = D$ and $g_1^{pw_C} A^z = C$. If so, *first-msg-in* is said to *appear valid*, and the query is answered normally. If not, *first-msg-in* is said to *appear non-valid*, and the query is answered normally except that the session key is chosen randomly from $\mathcal{G}$.

Calls to $\mathsf{Send}_2(\mathit{Client}, i, \mathit{msg\text{-}in})$ are answered in similar fashion. If *msg-in* is previously-used, the query is answered normally. Otherwise, let *msg-in* $= \langle \mathit{Server}|E|F|G|I|J\rangle$. The simulator computes $\beta = \mathcal{H}(\mathit{Server}|E|F|G|I)$ and checks whether $F^{x_1+y_1\beta} G^{x_2+y_2\beta} = J$ and $g_1^{pw_C} F^z = I$. If so, *msg-in* is said to *appear valid*, and the query is answered normally. If not, *msg-in* is said to *appear non-valid*, and the query is answered normally but the session key for instance Π_C^i is chosen randomly from $\mathcal{G}$.

The adversary's view of this protocol is exactly equivalent to its view of protocol P_2. When *first-msg-in* or *msg-in* appear non-valid, they are in fact not valid for password pw_C, and Fact 1 shows that the resulting session key is independent of the adversary's view. On the other hand, a message which appears valid may in fact be invalid, but since the query is answered normally the adversary's view is not affected.

In protocol P_4, the definition of the adversary's success is changed:

- If, during the course of answering a Send_3 oracle query, *first-msg-in* is new and appears valid, the session key is set to the special value ∇. If the adversary ever asks a Reveal query for this instance, the simulator halts immediately and the adversary succeeds.
- If, during the course of answering a Send_2 oracle query, *msg-in* is new and appears valid, the session key is set to the special value ∇. If the adversary ever asks a Reveal query for this instance, the simulator halts immediately and the adversary succeeds.
- Otherwise, the adversary succeeds, as before, by guessing the bit b.

This can only increase the advantage of the adversary.

In protocol P_5, calculation of the session key by the Send_3 oracle is modified. First, every time K is computed by the simulator when answering a call to the Send_2 oracle, the simulator stores K along with its associated values of x, y, z, w. When a call is made to the Send_3 oracle with *msg-in* $= \langle K|\mathsf{Sig}\rangle$, there are four possibilities:

- *first-msg-in* is new and appears valid. In this case the session key is set to ∇ and the simulator behaves as in P_4 (above).
- *first-msg-in* is new and appears non-valid. In this case, the simulator chooses the session key randomly (as in P_3, P_4).
- *first-msg-in* is previously-used and $\mathsf{Verify}_{\mathsf{VK}}((\beta|K), \mathsf{Sig}) = 0$. In this case, the simulator chooses the session key randomly (as in $P_0, \ldots, P_4$).
- *first-msg-in* is previously-used and $\mathsf{Verify}_{\mathsf{VK}}((\beta|K), \mathsf{Sig}) = 1$. It must be the case that $\langle K, \mathsf{Sig} \rangle$ was previously output by the Send_2 oracle (since we assume the adversary has not forged any new message/signature pairs). The simulator therefore knows values x', y', z', w' such that $K = g_1^{x'} g_2^{y'} h^{z'} (cd^{\beta})^{w'}$. Let *first-msg-out* $= \langle \mathit{Server}|E|F|G|I|J \rangle$ and $I^* = I \cdot g_1^{-pw_C}$. The simulator calculates the session key as:

$$sk_S^i \longleftarrow A^x B^y (C^*)^z D^w F^{x'} G^{y'} (I^*)^{z'} J^{w'}.$$

The adversary's view is exactly equivalent to the adversary's view in P_4 (since K^r does equal $F^{x'} G^{y'} (I^*)^{z'} J^{w'}$ when *first-msg-out* is a valid message; it is valid since it was generated by the simulator who knows the appropriate password).

In protocol P_6 we change oracle Send_1 so that component I is chosen at random from $\mathcal{G}$. This cannot change the adversary's success probability by more than $q_{\mathrm{send}_1} \varepsilon_{\mathrm{cs}}(k, t, q_{\mathrm{send}_2} + q_{\mathrm{send}_3})$. If it did, the simulator could break extended-CS encryption under a chosen ciphertext attack as follows: parameters for extended-CS encryption become the public parameters for the protocol. During the course of the protocol, the simulator may determine whether a new message appears valid by submitting it to the decryption oracle and checking whether the returned plaintext is equal to the appropriate password. When calls to the Send_1 oracle are made, the simulator submits the appropriate password as the plaintext along with the server name, the value α, and a request for a server-encryption (see Appendix B). In return, the simulator is given $\langle \mathit{Server}|E|F|G|I|J \rangle$ (which may be an encryption of either the appropriate password or a random group element) along with x, y, z, w such that $E = g_1^x g_2^y h^z (cd^{\alpha})^w$. A simple hybrid argument bounds the change in the adversary's success probability.

In protocol P_7, the Send_2 oracle is changed so that whenever *msg-in* was previously-used the session key is chosen at random from $\mathcal{G}$. To ensure consistency, the $\mathsf{Send}_3(\mathit{Server}, i, *)$ oracle is changed as follows: if sid_S^i matches sid_C^j for some other instance Π_C^j, then sk_S^i is set equal to sk_C^j. The statistical difference between the adversary's view in this protocol and the previous one is bounded by $\min\{q_{\mathrm{reveal}}, q_{\mathrm{send}_2}\}/q$. Indeed, Fact 1 shows that the views are equivalent when *msg-in* is invalid. Furthermore, the probability that *msg-in* is valid for the appropriate password is $1/q$ (since I was chosen at random).

In protocol P_8, the Send_0 oracle is changed so that component C is chosen randomly from $\mathcal{G}$. Following a similar analysis to that of protocol P_6, this cannot change the adversary's success probability by more than $q_{\mathrm{send}_0} \varepsilon_{\mathrm{cs}}(k, t, q_{\mathrm{send}_2} + q_{\mathrm{send}_3})$. Finally, in protocol P_9 the Send_3 oracle is changed so that a random session key is chosen when *first-msg-in* is previously-used. Following a similar analysis to that of protocol P_7, the statistical difference between the adversary's view in this protocol and the previous protocol is bounded by $\min\{q_{\text{reveal}}, q_{\text{send}_3}\}/q$.

Consider the adversary's advantage in protocol P_9. The adversary's view is entirely independent of the passwords chosen by the simulator unless the adversary manages to submit a new *msg-in* which appears valid at some point during execution of the protocol; i.e., succeeds in guessing the password. The adversary's probability of guessing the password, however, is precisely $(q_{\text{send}_2} + q_{\text{send}_3})/N$ (this assumes that passwords are selected uniformly; an analogous calculation can be done when this is not the case). The adversary's advantage in protocol P_9 is thus bounded by $q_{\text{send}}/2N$ (note that the adversary must ask a q_{send_0} query for a q_{send_2} query to be meaningful, and similarly must ask a q_{send_1} query for a q_{send_3} query to be meaningful). The adversary's advantage in the original protocol is therefore bounded by the expression in Theorem 1. ■

5 Acknowledgments

Thanks to Yehuda Lindell, Philip MacKenzie, and Steven Myers for many helpful discussions on the topic of password-authenticated key exchange.

References

1. M. Bellare, A. Boldyreva, and S. Micali. Public-Key Encryption in a Multi-User Setting: Security Proofs and Improvements. Eurocrypt 2000.
2. M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. STOC '98.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. Eurocrypt 2000.
4. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. Crypto '93.
5. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. ACM CCCS '93.
6. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: the Three Party Case. STOC '95.
7. S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks. IEEE Symposium on Security and Privacy, 1992.
8. D. Boneh. The Decision Diffie-Hellman Problem. Proceedings of the Third Algorithmic Number Theory Symposium, 1998.
9. M. Boyarsky. Public-Key Cryptography and Password Protocols: The Multi-User Case. ACM CCCS '99.
10. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. Eurocrypt 2000.
11. V. Boyko. On All-or-Nothing Transforms and Password-Authenticated Key Exchange Protocols. PhD Thesis, MIT, Department of Electrical Engineering and Computer Science, Cambridge, MA, 2000.
12. R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. STOC '98.

13. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Chosen Ciphertext Attack. Crypto '98.
14. W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Trans. Info. Theory*, 22(6): 644–654, 1976.
15. S. Even, O. Goldreich, and S. Micali. On-Line/Off-Line Digital Signatures. Crypto '89.
16. O. Goldreich. On the Foundations of Modern Cryptography. Crypto '97.
17. O. Goldreich and Y. Lindell. Personal Communication and Crypto 2000 Rump Session. Session-Key Generation using Human Passwords Only. Available at `http://eprint.iacr.org/2000/057`.
18. O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game, or a Completeness Theorem for Protocols with an Honest Majority. STOC '87.
19. S. Goldwasser, R. Rivest, and S. Micali. A Digital Signature Scheme Secure Against Adaptive Chosen Message Attacks. SIAM J. Comp. 17(2): 281–308, 1988.
20. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. ACM Transactions on Information and System Security, 2(3): 230–268, 1999.
21. S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. Proceedings of the Workshop on Security Protocols, 1997.
22. P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. Asiacrypt 2000.
23. M. Naor and M. Yung. Universal One-Way Hash Functions and Their Cryptographic Applications. STOC '89.
24. G. Poupard and J. Stern. Security Analysis of a Practical "on the fly" Authentication and Signature Generation. Eurocrypt '98.
25. G. Poupard and J. Stern. On the Fly Signatures Based on Factoring. ACM CCCS '99.
26. J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. STOC '90
27. C.-P. Schnorr. Efficient Signature Generation by Smartcards. J. Crypto. 4(3): 161–174 (1991).
28. V. Shoup. On Formal Models for Secure Key Exchange. Available at `http://philby.ucsd.edu/cryptolib`.

A Building Blocks

Decisional Diffie-Hellman (DDH) Assumption (see [8]). For concreteness, we let $\mathcal{G}$ be a subgroup of $\mathbb{Z}_p^*$ of order q where p, q are prime, $q|p-1$, and $|q| = k$, the security parameter. Let g be a generator of $\mathcal{G}$. The DDH assumption states that it is infeasible for an adversary to distinguish between the following distributions:

$$\{x, y, z \leftarrow \mathbb{Z}_q : (g^x, g^y, g^{xz}, g^{yz})\} \text{ and } \{x, y, z, w \leftarrow \mathbb{Z}_q : (g^x, g^y, g^z, g^w)\}.$$

More precisely, choose at random one of the above distributions and give adversary $\mathcal{A}$ an element chosen from this distribution. The adversary succeeds by guessing which distribution was chosen; the advantage is defined as usual. Let $\varepsilon_{\mathrm{ddh}}(k,t)$ be the maximum advantage of any adversary which runs in time t. The DDH assumption is that for $t = \mathrm{poly}(k)$, the advantage $\varepsilon_{\mathrm{ddh}}(k,t)$ is negligible.

One-Time Digital Signatures (see [19, 15]). Let $\mathsf{SigGen}(1^k)$ be a probabilistic algorithm generating a public verification key/private signing key $(\mathsf{VK}, \mathsf{SK})$. Signing message M is denoted by $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(M)$, and verification is denoted by $b = \mathsf{Verify}_{\mathsf{PK}}(M, \mathsf{Sig})$ (the signature is correct if $b = 1$). Consider the following experiment: $\mathsf{SigGen}(1^k)$ is run to generate $(\mathsf{VK}, \mathsf{SK})$. Message M is chosen, and the signature $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(M)$ is computed. Adversary $\mathcal{A}$ is given $(\mathsf{PK}, M, \mathsf{Sig})$ and outputs a pair (M', Sig') which is not equal to the message/signature pair it was given. The adversary's advantage is defined as the probability that $\mathsf{Verify}_{\mathsf{PK}}(M', \mathsf{Sig}') = 1$. Let $\varepsilon_{\mathrm{sig}}(k, t)$ be the maximum possible advantage of any adversary which runs in time t. The assumption is that for $t = \mathrm{poly}(k)$, this value is negligible. Note that a signature scheme meeting this requirement can be constructed [15, 26] given any one way function[7].

Extended Cramer-Shoup Encryption ([13]). The Cramer-Shoup cryptosystem is an encryption scheme secure under adaptive chosen ciphertext attack (see [13] for formal definitions). We extend their cryptosystem, as discussed in Appendix B; our extension remains secure under adaptive chosen ciphertext attack. The extension gives two "types" of encryption algorithms: a client-encryption algorithm and a server-encryption algorithm, both using the identical public parameters.

Consider the following experiment: $\mathsf{ExtCSGen}(1^k)$ is run to generate public key/private key pair (pk, sk). Adversary $\mathcal{A}$ is given pk and is also given access to a decryption oracle which, given ciphertext C, returns the corresponding plaintext P (or $\perp$ if the ciphertext is invalid). The adversary outputs a plaintext x, and may request either a client-encryption or a server-encryption of x. A random bit b is chosen; if $b = 0$ the adversary is given a random encryption (of the type requested) of x, while if $b = 1$ the adversary is given a random encryption (of the type requested) of a random element. The adversary may continue to submit queries to the decryption oracle, but cannot ask for decryption of the challenge ciphertext. The adversary succeeds by guessing b; the advantage is defined as usual. Let $\varepsilon_{\mathrm{cs}}(k, t, d)$ be the maximum possible advantage of any adversary which runs in time t and asks at most d decryption oracle queries. In [13] (see also Appendix B) it is proved that for $t, d = \mathrm{poly}(k)$, the advantage $\varepsilon_{\mathrm{cs}}(k, t, d)$ is negligible (under the DDH assumption). A concrete security bound can be found in [1].

B Extended Cramer-Shoup Encryption

We consider here an extension of the Cramer-Shoup encryption scheme [13] which is chosen-ciphertext secure. No new techniques are used, and the proof of security for the modified scheme is exactly the same as for the original with the exception of a few details which one must be careful to get right.

Public parameters are generators $g_1, g_2, h = g_1^z, c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2} \in \mathcal{G}$ along with a universal one-way hash function $\mathcal{H}$. Ciphertexts are of the form:

[7] The DDH assumption implies that $f(x) = g^x$ is a one-way function.

$\langle A|B|C|D|E|F\rangle$. Decryption is done as in [13]: first, $\alpha = \mathcal{H}(A, B, C, D, E)$ is computed, and the following condition is checked:

$$C^{x_1+y_1\alpha} D^{x_2+y_2\alpha} \overset{?}{=} F.$$

If it fails output $\perp$. Otherwise, output the plaintext E/C^z.

The essential difference lies in the definition of the encryption oracle. The adversary submits a plaintext m but also submits additional information, and the encryption oracle returns some side information in addition to the ciphertext. More precisely, the adversary includes a bit $b \in \{0, 1\}$, which determines whether the plaintext is encrypted via *client-encryption* or *server-encryption*. For the case of client-encryption, the adversary also includes $\mathit{Client} \in \mathsf{Client}$. For the case of server-encryption, the adversary includes $\mathit{Server} \in \mathsf{Server}$ and a value $\alpha \in \mathbb{Z}_q$. The encryption oracle sets $m' = m$ with probability 1/2 and chooses m' randomly from $\mathcal{G}$ otherwise. Encryption is then carried out as follows:

```
Client-encryption($m'$, $\mathit{Client}$)
  $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{SigGen}(1^k)$
  $A = \mathit{Client}; B = \mathsf{VK}$
  $r \leftarrow \mathbb{Z}_q$
  $C = g_1^r; D = g_2^r; E = h^r m'$
  $\alpha = H(A, B, C, D, E)$
  $F = (cd^\alpha)^r$
  $\mathsf{return}(\langle A, B, C, D, E, F\rangle, \mathsf{SK})$

Server-encryption($m'$, $\mathit{Server}$, $\alpha$)
  $x, y, z, w, r \leftarrow \mathbb{Z}_q$
  $A = \mathit{Server}; B = g_1^x g_2^y h^z (cd^\alpha)^w$
  $C = g_1^r; D = g_2^r; E = h^r m'$
  $\beta = H(A, B, C, D, E)$
  $F = (cd^\beta)^r$
  $\mathsf{return}(\langle A, B, C, D, E, F\rangle, x, y, z, w)$
```

Theorem 2. *The encryption scheme outlined above is secure (in the sense of indistinguishability) under an adaptive chosen ciphertext attack.*

Sketch of Proof (Informal) The proof of security exactly follows [13], and it can be easily verified that the additional information given to the adversary does not improve her advantage. One point requiring careful consideration is the adversary's probability of finding a collision in $\mathcal{H}$. If $\mathcal{H}$ is collision resistant (a stronger assumption than being universal one-way), there is nothing left to prove. If $\mathcal{H}$ is universal one-way, however, it can first be noted that VK or B could be selected by a simulator before $\mathcal{H}$ is given to it (if the simulator prepares the public key such that it knows $\log_{g_1} g_2$ it can produce a representation of B for any value α given to it by the adversary). But, we must also deal with the fact that the adversary gets to choose A (and the bit b which determines whether client-encryption or server-encryption is used) *after* seeing $\mathcal{H}$. However, since the set User is fixed in advance, and (at worst) of size polynomial in the security parameter, the simulator can "guess" the adversary's choices in advance (before being given $\mathcal{H}$) and this will only affect the simulator's probability of finding a collision by a polynomial factor (details omitted). ■

C Formal Specification of the Protocol

```
$\mathsf{Initialize}(1^k)$ —
    Select $p, q$ prime with $|p| = k$ and $q|p-1$; this defines group $\mathcal{G}$
    Choose random generators $g_1, g_2, h, c, d \leftarrow \mathcal{G}$
    $\mathcal{H} \leftarrow \mathsf{UOWHF}$
    Publish parameters $(q, p, g_1, g_2, h, c, d, H)$
    $\langle pw_C \rangle_{C \in \mathsf{Client}} \leftarrow \{1, \ldots, N\}$
```

Fig. 2. Specification of protocol initialization.

```
$\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$ —
    $(\mathsf{VK}, \mathsf{SK}) \xleftarrow{R} \mathsf{SigGen}(1^k)$    $x_1, x_2, y_1, y_2, z_1, z_2, w_1, w_2, r_1, r_2 \xleftarrow{R} \mathbb{Z}_q$
    $A = g_1^{r_1};\ B = g_2^{r_1};\ C = h^{r_1} g_1^{pw_C}$    $\alpha = \mathcal{H}(\mathit{Client} \mid \mathsf{VK}|A|B|C)$
    $D = (cd^{\alpha})^{r_1}$    $\mathit{msg\text{-}out}_1 \longleftarrow \langle \mathit{Client} \mid \mathsf{VK} \mid A \mid B \mid C \mid D \rangle$
    $E = g_1^{x_1} g_2^{x_1} h^{z_1} (cd^{\alpha})^{w_1}$    $F = g_1^{r_2};\ G = g_2^{r_2};\ I = h^{r_2} g_1^{pw_C}$
    $\beta = \mathcal{H}(\mathit{Server} \mid E|F|G|I)$
    $J = (cd^{\beta})^{r_2}$    $\mathit{msg\text{-}out}_2 \longleftarrow \langle \mathit{Server} \mid E \mid F \mid G \mid I \mid J \rangle$
    $K = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\beta})^{w_2}$    $\mathit{msg\text{-}out}_3 \longleftarrow \langle K \mid \mathsf{Sign}_{\mathsf{SK}}(\beta|K) \rangle$
    $sk_S^j \longleftarrow sk_C^i \longleftarrow A^{x_1} B^{y_1} (C \cdot g_1^{-pw_C})^{z_1} D^{w_1} F^{x_2} G^{y_2} (I \cdot g_1^{-pw_C})^{z_2} J^{w_2}$
    $sid_S^j \longleftarrow sid_C^i \longleftarrow \langle \mathit{msg\text{-}out}_1 \mid \mathit{msg\text{-}out}_2 \mid \mathit{msg\text{-}out}_3 \rangle$
    return $\langle \mathit{msg\text{-}out}_1, \mathit{msg\text{-}out}_2, \mathit{msg\text{-}out}_3 \rangle$

$\mathsf{Reveal}(\mathit{User}, i)$ —
    return $sk_U^i$

$\mathsf{Test}(\mathit{User}, i)$ —
    $b \xleftarrow{R} \{0, 1\}, sk \leftarrow \mathcal{G}$
    if $b = 0$ return $sk$ else return $sk_U^i$
```

Fig. 3. Specification of the Execute, Reveal, and Test oracles to which the adversary has access. Note that $q, g_1, g_2, h, c, d, \mathcal{H}$ are public, and $\mathcal{G}$ is the underlying group. Subscript S refers to the server, and C to the client.

```
$\mathsf{Send}_0(Client, i, Server)$ —
    $(\mathsf{VK}, \mathsf{SK}) \xleftarrow{R} \mathsf{SigGen}(1^k)$    $r \xleftarrow{R} \mathbb{Z}_q$
    $A = g_1^r;\ B = g_2^r;\ C = h^r g_1^{pw_C}$    $\alpha = \mathcal{H}(Client|\mathsf{VK}|A|B|C)$
    $msg\text{-}out \longleftarrow \langle Client \mid \mathsf{VK} \mid A \mid B \mid C \mid (cd^{\alpha})^r \rangle$
    $state_C^i \longleftarrow \langle \mathsf{SK}, r, msg\text{-}out \rangle$
    return $msg\text{-}out$

$\mathsf{Send}_1(Server, i, \langle Client, \mathsf{VK}, A, B, C, D \rangle)$ —
    $x, y, z, w, r \xleftarrow{R} \mathbb{Z}_q$    $\alpha = \mathcal{H}(Client|\mathsf{VK}|A|B|C)$    $E = g_1^x g_2^y h^z (cd^{\alpha})^w$
    $F = g_1^r;\ G = g_2^r;\ I = h^r g_1^{pw_C}$    $\beta = \mathcal{H}(Server \mid E|F|G|I)$
    $msg\text{-}out \longleftarrow \langle Server \mid E \mid F \mid G \mid I \mid (cd^{\beta})^r \rangle$
    $state_S^i \longleftarrow \langle msg\text{-}in, x, y, z, w, r, \beta, msg\text{-}out \rangle$
    return $msg\text{-}out$

$\mathsf{Send}_2(Client, i, \langle Server, E, F, G, I, J \rangle)$ —
    $\langle \mathsf{SK}, r, first\text{-}msg\text{-}out \rangle \longleftarrow state_C^i$    $\beta = \mathcal{H}(Server|E|F|G|I)$
    $x, y, z, w \xleftarrow{R} \mathbb{Z}_q$    $K = g_1^x g_2^y h^z (cd^{\beta})^w$
    $msg\text{-}out \longleftarrow \langle K \mid \mathsf{Sign}_{\mathsf{SK}}(\beta|K) \rangle$    $sid_C^i \longleftarrow \langle first\text{-}msg\text{-}out \mid msg\text{-}in \mid msg\text{-}out \rangle$
    $I^* = I \cdot g_1^{-pw_C}$    $sk_C^i \longleftarrow E^r F^x G^y (I^*)^z J^w$
    return $msg\text{-}out$

$\mathsf{Send}_3(Server, i, \langle K, \mathsf{Sig} \rangle)$ —
    $\langle first\text{-}msg\text{-}in, x, y, z, w, r, \beta, first\text{-}msg\text{-}out \rangle \longleftarrow state_S^i$
    $\langle \mathsf{VK}, A, B, C, D \rangle \longleftarrow first\text{-}msg\text{-}in$
    $sid_S^i \longleftarrow \langle first\text{-}msg\text{-}in \mid first\text{-}msg\text{-}out \mid msg\text{-}in \rangle$
    if $\mathsf{Verify}_{\mathsf{VK}}((\beta|K), \mathsf{Sig}) = 1$
        $C^* = C \cdot g_1^{pw_C}$    $sk_S^i \longleftarrow A^x B^y (C^*)^z D^w K^r$
    else
        $sk_S^i \xleftarrow{R} \mathcal{G}$
    return $\varepsilon$
```

Fig. 4. Specification of the Send oracles to which the adversary has access. Note that $q, g_1, g_2, h, c, d, \mathcal{H}$ are public, and $\mathcal{G}$ is the underlying group. Subscript S refers to the server, and C to the client. The third argument to the Send oracles is denoted *msg-in*.