Secure Password-Based Cipher Suite for TLS

Michael Steiner
Universität des Saarlandes
and
Peter Buhler, Thomas Eirich and Michael Waidner
IBM Research

SSL is the de-facto standard today for securing end-to-end transport on the Internet. While the protocol itself seems rather secure, there are a number of risks that lurk in its use, e.g., in web banking. However, the adoption of password-based key-exchange protocols can overcome some of these problems. We propose the integration of such a protocol (DH-EKE) in the TLS protocol, the standardization of SSL by IETF. The resulting protocol provides secure mutual authentication and key establishment over an insecure channel. It does not have to resort to a PKI or keys and certificates stored on the users computer. Additionally, its integration in TLS is as minimal and non-intrusive as possible.

Categories and Subject Descriptors: C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*client/server*; D.4.6 [**Operating Systems**]: Security and Protection—*authentication*; H.4.3 [**Information Systems Applications**]: Communications Applications—*information browsers*; K.4.4 [**Computers and Society**]: Electronic Commerce—*security*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*authentication*

General Terms: Algorithms, Human Factors, Security

Additional Key Words and Phrases: Authenticated key exchange, dictionary attack, key agreement, password, perfect forward secrecy, secure channel, transport layer security, weak secret

1. INTRODUCTION

The *Secure Socket Layer (SSL)* protocol [Freier et al. 1996] is the current de-facto standard for securing end-to-end transport over the Internet. The presence of SSL in virtually all web browsers has led to its widespread use, also in application requiring a high level of security such as home banking. Whereas early versions of SSL contained a number of flaws and shortcomings, the analysis of the latest

An earlier version appeared in the *Proceedings of Network and Distributed Systems Security Symposium*, San Diego, CA, Feb. 3–4, 2000, pp. 129–142 (Internet Society, Reston, 2000).
Author's addresses: Michael Steiner, Universität des Saarlandes, FR Informatik, Postfach 151150, D-66041 Saarbrücken, Germany; email: steiner@acm.org; Peter Buhler, Thomas Eirich, and Michael Waidner, IBM Research, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland; email: {bup,eir,wmi}@zurich.ibm.com.

version 3.0 has revealed only a few minor anomalies [Wagner and Schneier 1996; Mitchell et al. 1998]. SSL was further refined in the *Transport Layer Security (TLS)* protocol [Dierks and Allen 1999], the standardization effort of the *Internet Engineering Task Force (IETF)*, and now seems to provide a reasonable level of security.[1]

Currently, all standard methods for authentication in TLS rely on a *public-key infrastructure (PKI)*. While this is suitable for many cases it might not suit environments where the infrastructure is "light-weight" (e.g., disk-less workstations, user-to-user authentication), situations where a system has to be bootstrapped from scratch, or contexts where user mobility is required.

Furthermore, current cipher suites also pose their own risks, prominently illustrated in following example: Over the past few years many banks have built home-banking applications for the web. For their security these applications rely mainly on the integration of SSL into the web browsers. As issuing client certificates securely and reliably is quite involved, most of them use SSL for server authentication only. They set up a secure channel from the browser of the bank customer to the server and then ask the customer to authenticate herself by typing her password into a simple web form. However, in such a setup the authentication of the customer is not directly tied to the secure channel and, in fact, the security cannot be guaranteed if the customer does not explicitly verify the connection before entering her password.

As illustrated in Figure 1, it is not sufficient to observe that the lock turns golden and locked to verify that there is a secure connection. The bank customer also has to check that the certificate identifies the right bank and is issued by a *Certification Authority (CA)* that is appropriate in this context to make sure that the connection is to the right entity. This is a non-trivial task as, for example, Netscape contains by default more than 70 different root certificates. Moreover, the assurance provided by the corresponding certification procedures is difficult to figure out, and varies from a few with high guarantee to most with virtually none.[2] To counter possible attacks the customer might even have to verify the fingerprint of the CA itself. If the customer fails to do that properly she is highly susceptible to a man-in-the-middle attack and to a potential theft of her money. This seems to put too high a burden on the average customer. A reasonable system should be fool proof.

Use of one-time-use *transaction authorization numbers (TAN)* only marginally improves this situation. Using client side certificates helps but complicates the setup and requires proper protection of the client's keys, a difficult task given the (in)security of the common operating systems available today.

The above-mentioned problems related to a PKI are inherent weaknesses of general-purpose applications such as web browsers. Multiple (and fundamentally different!) trust domains (CAs) have to co-exist, and an application cannot know and enforce which policies are appropriate for a particular context. However, these issues are not intrinsic problems of SSL and will not arise with the password-based

[1] Note that the risk of the recent, very practical attack of Bleichenbacher [1998] on the RSA-based cipher suites can be reduced by careful implementations. The adoption of Version 2.0 of PKCS #1 [Kaliski and Staddon 1998] and its new encoding method `EME-OAEP` based on work by Bellare and Rogaway [1995a] should thwart such an attack completely.

[2] See Section 5.5.3 for further discussion on problems arising from the certificate management of web browsers.

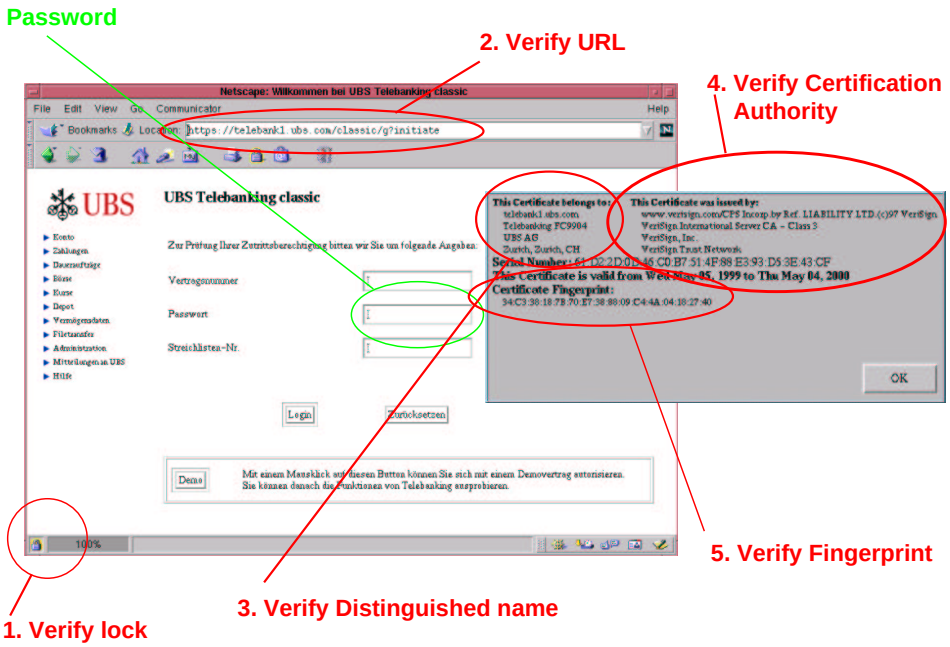

Fig. 1. One-way authenticated TLS channels: The end-user's heavy burden ...

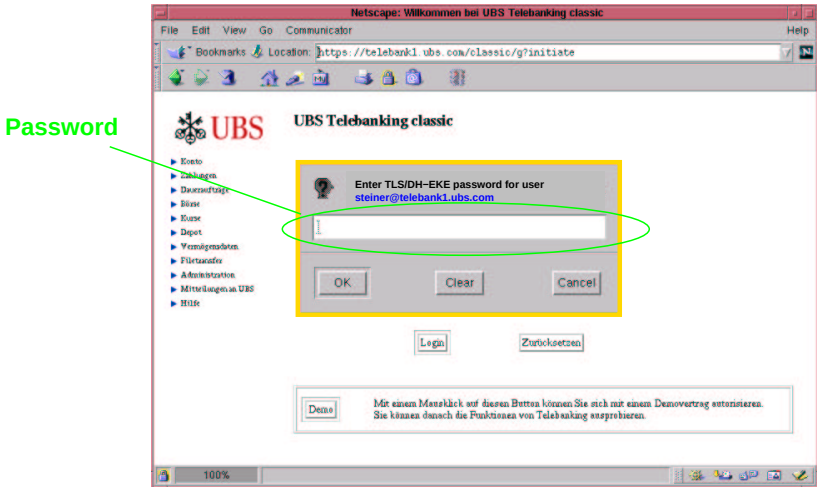

Fig. 2. TLS channels with password-based key-exchange: Minimal and fool proof ...

protocols presented in the following, regardless of the application. If we integrate such password-based key-exchange protocols in web browsers we can overcome the problems mentioned above: A password entered in the corresponding pop-up window (see Figure 2) is guaranteed not to leak to any remote[3] attacker.

The recent addition of cipher suites based on *Kerberos* [Medvinsky and Hur 1999; Kohl and Neuman 1993] eliminates the requirement of a PKI. Unfortunately, Kerberos is not really light-weight (e.g., there is no real structural difference from a PKI) and, more importantly, it is vulnerable to *dictionary attacks* when weak passwords are used [Wu 1999; Bellovin and Merritt 1991; Morris and Thompson 1979; Gong et al. 1989].[4] Given the human nature, this cannot be excluded. Pro-active password checking [Bishop and Klein 1995] can help only to a limited degree: On the one hand, the choice of passwords has to be easy and unrestricted enough to make it possible for users to remember their passwords (without having to write them down). This limits the possible entropy in such passwords. On the other hand, computing power still grows dramatically and makes dictionary attacks possible on larger and larger classes of passwords.

Luckily, there is a class of authenticated key-exchange protocols that are resistant to (off-line) dictionary attacks even when used with memorizable and potentially weak secrets, i.e., passwords. They do not have to be backed by any infrastructure such as a PKI. Assuming proper handling of online dictionary attacks, which are usually detectable, these systems are at least as secure as other systems based on strong public or shared keys. To substantiate the "at least" we note that in reality most of these other systems rely also on passwords somewhere at the user end: The key ring in PGP [Zimmermann 1995] and secret keys related to client certificates in browsers are password encrypted and are even vulnerable to undetectable off-line dictionary attacks once the key files leak (e.g., because of backups)! The security of password-based key-exchange protocols relies only on two assumptions: The integrity of the underlying hardware and software, and the availability of a reasonably good source of randomness. But this, in essence, is the minimum requirement for any secure system.

Therefore, it seems quite useful to enrich the set of current TLS cipher suites with a password-based protocol and to reduce the risks explained above. In the following, we describe the integration of an improved version of the *Diffie-Hellman Encrypted Key Exchange (DH-EKE)* [Bellovin and Merrit 1992] into TLS. The new cipher suite provides mutual authentication and key establishment with *perfect forward secrecy* over an insecure channel and, limits the damage in case an attacker gains access to the server's databases. The integration into TLS is as non-intrusive as possible and, with some optimizations, retains the 4-round handshake overhead of TLS.

The structure of the remainder of this article is as follows. In Section 2 we explain the underlying cryptographic protocol, a modified version of DH-EKE. In Section 3 we give a brief overview of the TLS flows and state some criteria for the integration

[3]Of course, Trojan horses such as the ones described by Tygar and Whitten [1996] can open another door for attackers, but this problem is orthogonal to the issues discussed in this article.
[4]Note that this vulnerability also applies to the use of Kerberos in MS Windows 2000 as far as one can tell from Chappell [1999] and other publicly available information.

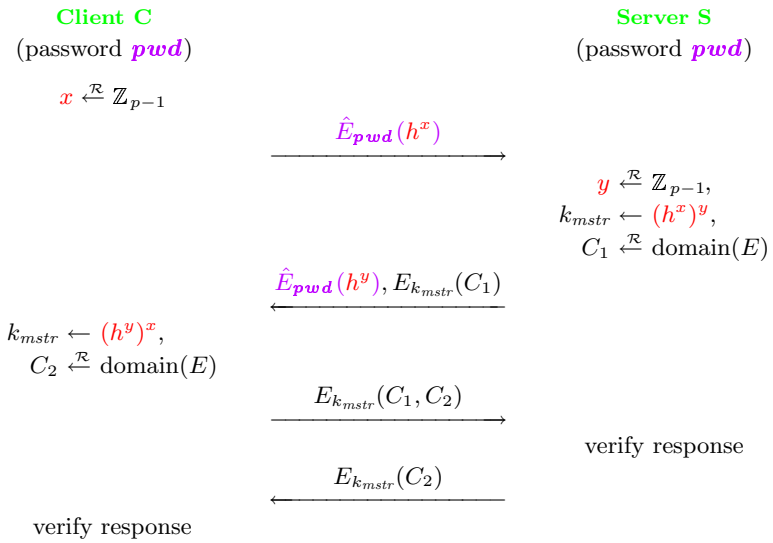

Fig. 3. DH-EKE

of a new cipher suite. In Section 4 we describe our new cipher suite in detail. We then give rationales for our choices in Section 5, and conclude in Section 6.

2. DH-EKE

2.1 Cryptographic Preliminaries

Let us first introduce the underlying algebraic structure and some notation.

The central parts of the following protocols are computed in a cyclic multiplicative group $\mathbb{Z}_p^*$, with p prime and q a large prime divisor of $\varphi(p) = (p-1)$. Let $n = \lceil \log_2 p \rceil$ and $m = \lceil \log_2 q \rceil$ be the number of bits of p and q, respectively. Typical values are 768, 1024 or 2048 for n and 160 or 320 for m. Let h be an (arbitrary) generator (primitive root) of $\mathbb{Z}_p^*$. Furthermore, let g be defined as $h^{(p-1)/q} \pmod p$. Note that g is a generator of the (unique) subgroup G of order q. Additionally, let g_* be a second generator of the subgroup G. For algorithms on finding primitive roots and efficiently computing group operations in multiplicative groups we refer the reader to other sources, e.g., the excellent book of Menezes et al. [1997].

Further commonly used notation is as follows: E and $\hat{E}$ denote an ordinary and a password-based symmetric cipher, respectively; $\mathcal{G}_i$ denotes a key-derivation function; and $\mathcal{H}_i$ refers to a pseudo-random function.

2.2 Exponential Key Exchange

In 1992 Bellovin and Merritt published a family of methods called *Encrypted Key Exchange (EKE)* [Bellovin and Merrit 1992]. These protocols provide key exchange with mutual authentication based on weak secrets (e.g., passwords). They are very carefully designed to prevent the leakage of weak secrets and to withstand dictionary attacks.

The simplest and most elegant of the methods is DH-EKE. In DH-EKE a weak

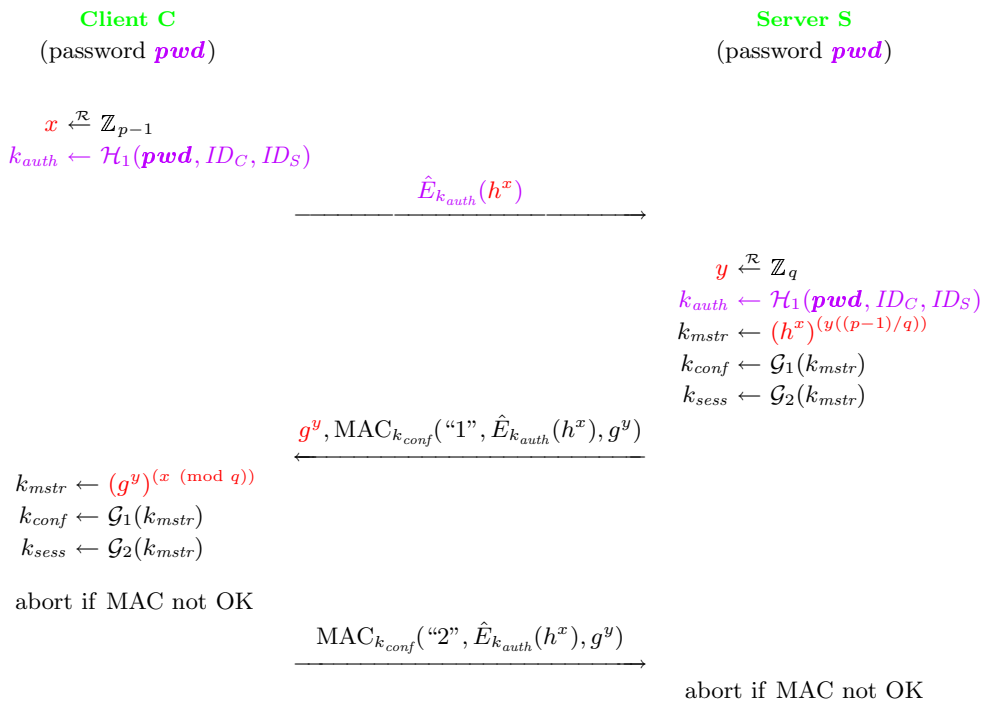

Fig. 4. Refined DH-EKE. (Recall that $g := h^{(p-1)/q}$)

secret P is used to encrypt two randomly chosen *half-keys* of a *Diffie-Hellman key exchange (DH)* [Diffie and Hellman 1976], i.e., $h^x \pmod p$ and $h^y \pmod p$.The protocol is shown in Figure 3. Note that all exponentiations in this figure as well as in all following ones are performed in the underlying finite cyclic group $\mathbb{Z}_p^*$, i.e., computed modulo p. However, to prevent cluttering the figures we omit explicit $\pmod p$s.

The session key computed by the client and the server[5] is $h^{xy} \pmod p$. This key is cryptographically strong regardless of the strength of the password as long as x and y are secret and cryptographically strong random numbers.

2.3 Refined DH-EKE

Various ways exist to optimize the number of encryptions and flows. However, these optimizations as well as the design of the encryption process and the choice of the algebraic group have to be done very carefully to prevent various attacks [Bellovin and Merrit 1992; Steiner et al. 1995; Jablon 1996; Patel 1997]. In the following we will describe the refined protocol forming the basis of our integration into TLS in more detail. Figure 4 shows an overview of the protocol.

As shown by Steiner et al. [1995], the number of flows and basic message elements can be minimized to three and four, respectively. It is also possible to omit

[5]DH-EKE is a-priori symmetric and insofar is not limited to a client-server relation. However, such a scenario is the most likely one and will be specifically addressed later, e.g., in Section 2.6.

the second encryption with the password.[6] Furthermore, the encryptions in the confirmation flows are replaced by *message authentication codes (*MAC*)*. This is cleaner in terms of functionality and more efficient. We also do not use the Diffie-Hellman key k_{mstr} directly as confirmation and session key but derive two separate and independent keys using the key-derivation functions $\mathcal{G}_i$: a confirmation key k_{conf} to provide key confirmation and a session key k_{sess} for a higher-level protocol. This allows modular protocol composition and prevents protocol interference attacks, i.e., no interference is possible between the messages of the key-exchange protocol and any higher-level protocol, regardless of its use of the session key k_{sess}. Finally, instead of using the password directly as encryption key, we use a *password authentication key* k_{auth} derived from the password and the identifiers of both parties using the pseudo-random function $\mathcal{H}_1$. This guarantees, with high probability, pair-wise unique encryption keys and eliminates the risk of interference between a client's (potentially parallel) sessions with multiple servers, even if this client reuses the same password with several of these servers.

2.4 Password Encryption

As mentioned, the design of the encryption process is a delicate issue and strongly depends on the choice of the algebraic group.

The encryption process $\hat{E}_P(\cdot)$ requires two properties to prevent an adversary from verifying candidate passwords using an observed encryption $\hat{E}_P(z)$: First, the encryption function should produce ciphertexts that contain no redundancy, and the range of the encryption function has to be the same regardless of the key chosen. Second, given a ciphertext, the possible corresponding plaintexts have to be unpredictable and close to uniformly distributed over the input domain of the encryption function.

The first condition is fulfilled by stream and block ciphers performing a permutation on the input block. The second condition is fulfilled by guaranteeing that the encrypted element is uniformly and randomly chosen from the underlying group and by encrypting it with the two-step encryption described in the next two paragraphs.

2.4.1 *Encryption of Elements in* $\mathbb{Z}_p^*$. Unfortunately, it is not secure to just map an element z of $\mathbb{Z}_p^*$ to an integer in $\{1, \ldots, p-1\}$ and to naively encrypt it with a standard stream or block cipher. The domain of these ciphers is usually a power of 2, and this can lead to following dictionary attack: Let len be the block length of a block cipher and let l be the smallest integer such that $l \cdot len$ is greater than n, the bit length of p. Furthermore, let $r := (p-1)/2^{l \cdot len}$ be the ratio of the number of possible group elements over the size of the domain of the cipher. An attacker observing an encryption $\hat{E}_P(z)$ has a probability of $(1-r)$ to reject a wrong password guess by decrypting $\hat{E}_P(z)$ with the guess and recovering an element in the illegal range $\{0, p, p+1, \ldots, 2^{l \cdot len} - 1\}$. Assuming that the attacker can observe t runs of the protocol, the probability of successfully rejecting a password guess becomes $(1-r^t)$. When $l \cdot len$ is not close to n, this value approaches unity extremely quickly. If we use a stream cipher the effect is smaller but still considerable.

[6] However, note that omitting the first rather than the second password encryption would trivially lead to a dictionary attack!

We solve this problem as follows: First, we expand the element z uniformly from an n-bit number to a random $(n+\alpha)$-bit number b. To achieve that, we choose a random integer $c \in \{0, \ldots, \lfloor (2^{\alpha+n})/p \rfloor - 1\}$ and compute $b = z + cp$. Second, we pad b with $((l \cdot len) - (n+\alpha))$ random bits if $(n+\alpha)$ is not a multiple of the block length, and encrypt the resulting value. On decryption, we simply strip off any padding and get z by reducing the retrieved value b modulo p.

On average, when expanding with 1 bit we decrease the proportion of the invalid range with respect to the complete range by half. Therefore, we also reduce the chances an attacker has by half. Let us define t_{max} as an upper bound for the number of protocol runs with a given password and 2^{-k} as the maximally tolerable probability that an attacker can reject an (incorrect) password guess after having observed some (i.e, at most t_{max}) protocol runs. Then the number of required expansion bits is $\alpha = -\log_2 (1 - (1 - 2^{-k})^{1/t_{max}}) \approx k + log_2(t_{max})$.

For the actual choice of α we refer the reader to Section 4.4, where we discuss the concrete instantiation of the encryption process in the context of TLS.

2.4.2 *Encryption of Elements in the Subgroup G.* As mentioned above, there should be no structure in the decryption as otherwise it might be open to attacks. Previous papers on DH-EKE commonly assumed that this means that we cannot operate in a (much more efficient) cyclic subgroup G but have to work in the entire group $\mathbb{Z}_p^*$ (e.g., we need a primitive root as base for the exponentiations). Encrypting elements of the subgroup would lead to following attack: The attacker chooses a candidate password, uses it to decrypt an observed encrypted half-key h^x, and rejects the password if the decrypted element is not an element of the subgroup. If the password guess was wrong the likelihood that the decrypted element is not an element of the subgroup is high and therefore the attack will be very effective.

However, if we intend to achieve semantic security in the sense that a valid session key should be indistinguishable from a random key, we encounter a problem. If we do not resort to random oracles [Bellare and Rogaway 1993], the weakest cryptographic assumption we can rely on is the hardness of the Decisional Diffie-Hellman problem (DDH). It is also clear that this cannot be done[7] in $\mathbb{Z}_p^*$ but only in a prime-order subgroup of $\mathbb{Z}_p^*$. This means that the security proof as found in the Appendix of Steiner et al. [1995] does not work for DH-EKE as originally proposed by Bellovin and Merrit [1992]. To make the proof work we have to modify the protocol such that it operates in a subgroup.

Luckily, the first observation that we cannot operate in subgroups is not completely correct: While it is true that we cannot encrypt elements of the subgroup with the password, it nevertheless does not prevent us from computing in the subgroup. The trick is simple. Instead of encrypting an element of the subgroup we randomly send one of the $((p-1)/q)$-th roots contained in the group. Assuming uniformly and randomly chosen exponents and roots, we will obtain a uniform distribution over $\mathbb{Z}_p^*$. Better still, as the sender actually chooses the element there is no need to compute roots and randomly select one of them: It is sufficient that the sender picks a random element in $\mathbb{Z}_p^*$ and the receiver constructs the element of

[7]The order of elements in $\mathbb{Z}_p^*$ leaks information that can be used to distinguish between (h^x, h^y, h^{xy}) and a triple of random elements of $\mathbb{Z}_p^*$ with high probability.

the subgroup by raising it to the power of $(p-1)/q$. Note that following equality always holds: $h^{x((p-1)/q)} \equiv (h^{(p-1)/q})^x \equiv g^x \equiv g^{x \pmod q} \pmod p$.

Therefore not only can we retain semantic security but we also improve efficiency as now only two of the four exponentiations require long exponents. Further performance improvements can be obtained if we choose h and/or g to be small. This will speed up exponentiations without any loss of security.

2.5 Subgroup Confinement

One concern for protocols based on discrete logarithms are subgroup confinement attacks [Lim and Lee 1997]: An attacker might send elements of small order to either reduce the possible key space for impersonations or attacks on the password, e.g., if the attacker sends 1 instead of g^y then the key will be 1 regardless of what the other (honest) party chooses as random exponent![8] This attack can be prevented by having the receiver of the unencrypted half-key g^y verify the order of the element. Verification of the order of decrypted values is not necessary: An attacker can either guess a password and encrypt an element of small order or send an arbitrary random value. In the unlikely case that the password guess was correct then obviously there is no point of encrypting an element of small order in the first place. Otherwise, given the pseudo-random nature of the encryption function, a decryption will yield a random element regardless whether the attacker has chosen a wrong password or an arbitrary value. But if $\varphi(p)$ has large prime factors it is highly unlikely that a randomly selected value decrypts to an element of small order.

If we choose $\mathbb{Z}_p^*$ such that $\varphi(p)/2$ would contain only prime factors of large size, i.e., they are all of at least m bits, we could improve the check for elements of small order even further. In such groups it is sufficient to test that $y^2 \pmod p \neq 1$ to verify that y has larger order [Lim and Lee 1997].

2.6 Reducing the Risk of Stolen Server Databases

As an additional precautionary measure we also reduce the risks resulting from the loss or theft of the user database from the server's machine. In the original proposal by Bellovin and Merrit [1992] the server had to store the password. This means that an attacker getting access to the server's database can immediately masquerade as both client and server. Extensions to EKE such as A-EKE [Bellovin and Merrit 1993], B-EKE [Jablon 1997] and SRP [Wu 1998] store only a (salted) hash of the password on the server side, reducing the risk of client impersonation to a dictionary attack, even when the server's database has leaked. While we argue that dictionary attacks are always feasible and therefore the password could eventually be revealed to an attacker in possession of the server's database, having such a second line of defense is nevertheless desirable.

For this reason we integrated the idea of B-EKE in our final protocol as shown in Figure 5. Instead of the password, the server now stores a *password verifier* v $(:= g^{k_{vrfy}} \pmod p)$ and a password authentication key k_{auth}, where k_{vrfy} and k_{auth} are derived from the client's password using (different) one-way functions. k_{auth} is

[8] Note that in this case the attacker can also trivially compute the necessary MAC in the second flow as $k_{conf} = \mathcal{G}_1(1)$ is known.

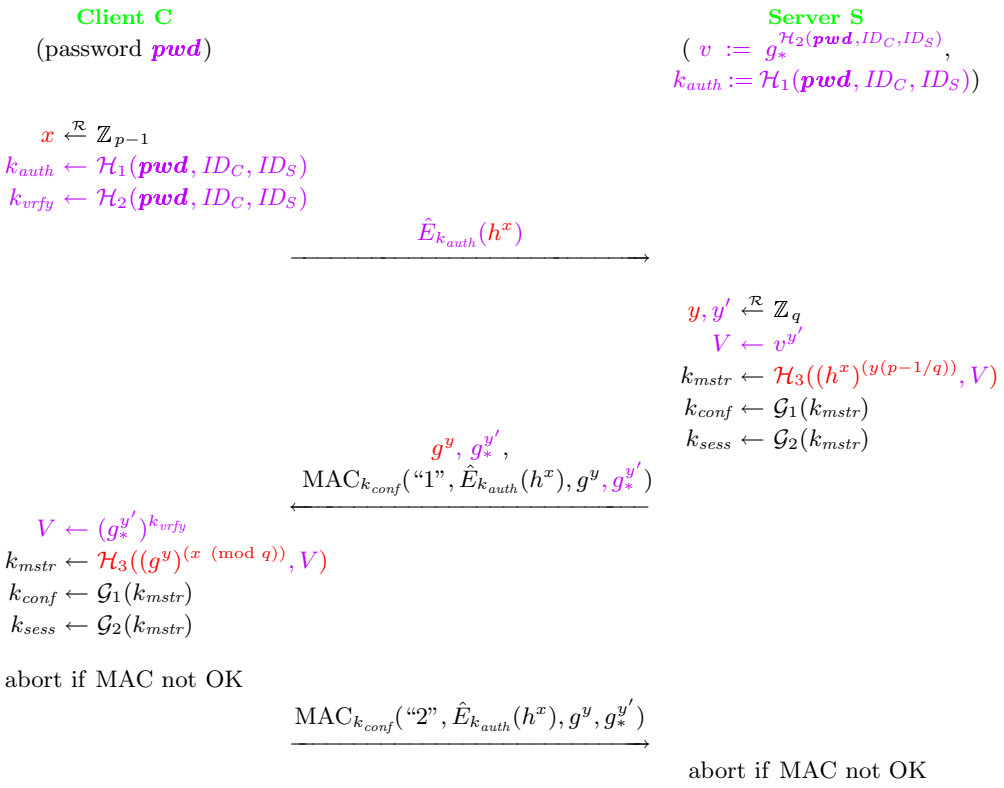

Fig. 5. Refined DH-EKE — Optimized for case of server compromise

used to encrypt the DH half-keys as before. Additionally, the client demonstrates its knowledge of k_{vrfy} (and hence the password) by being able to compute $(g_*^{y'})^{k_{vrfy}} \pmod p$.

Using the strong DH-key g^{xy} as key to the pseudo-random function $\mathcal{H}_3$ completely hides any information on the password, even if $g_*^{y'}$ is maliciously chosen or k_{mstr} were available to an attacker. We consider the added costs of the additional exponentiations (which are all with small exponents) worthwhile. However, it would be straightforward to make the use of B-EKE optional and to allow performance critical environments to trade off the risk of stolen server databases with improved performance.

A related speed improvement would be to choose the generators g and g_* identical and reuse y for y': This retains the security but saves one exponentiation at the server side. The reason for not doing this by default is that with such a modification the verifier g_* would fix the algebraic group used for the key generation. Keeping the two separate allows the server to react to increased security requirements regarding key lengths by choosing a larger (stronger) group for the DH key without having to go through a password renewal. Note that keeping the verifier in a potentially weak group might be tolerable as it merely is an independent, second line of defense. Note also that servers can use the above improvement transparently, even in the

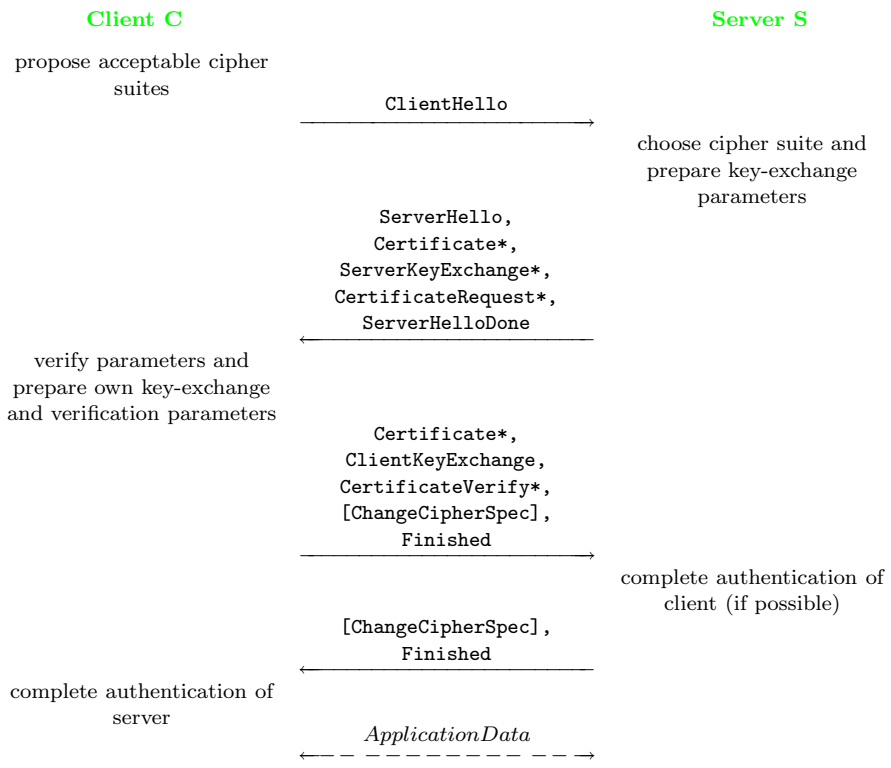

Fig. 6. Overview of TLS flows. (Situation-dependent messages are flagged with an asterisk.)

current proposal.

3. TLS

3.1 Overview

TLS is composed of two layers: the *TLS Record Protocol* and the *TLS Handshake Protocol*. The Record Protocol encapsulates higher-level protocols (such as HTTP [Berners-Lee et al. 1997]) and handles the reliability, confidentiality and compression of the messages exchanged over the connection. The TLS Handshake Protocol is responsible for setting up the secure channel between server and client, and provides the keys and algorithm information to the Record Protocol. The changes required in our integration of password-based protocols are not relevant to the Record Protocol. Therefore we will not discussion it further.

Figure 6 gives an overview of the flows of the Handshake Protocol. The main purpose of the first message, `ClientHello`, is to send a random challenge to guarantee freshness and to tell the server which cryptographic algorithms are supported by the client.

Based on this proposal the server will pick a set of algorithms, the *cipher suite*. As an illustrative example, let us assume that the cipher suite `TLS_DHE_DSS_WITH_DES_CBC_SHA` was chosen. This means that the session key will be based on a DH-key

exchange using ephemeral parameters, DSS is the signature algorithm used, and the security on the record layer will be based on DES in CBC mode and SHA-1. The cipher suite chosen is stored in the `ServerHello` message together with another random challenge to help assure the server of the freshness of the protocol run. If server authentication is required the server sends its own certificate in `Certificate`. Depending on the cipher suite chosen the server also sends the `ServerKeyExchange` message. This message contains keying data required for the key exchange. In our example it would hold the server's ephemeral DH half-key g^x signed with the server's signing key. Furthermore, a list of accepted certificate types and CAs is sent as part of the `CertificateRequest` if client authentication is required. Finally, the server marks the end of the turn by sending the `ServerHelloDone`.

In the next step the client verifies the received data. The client prepares its own contribution to the key generation, e.g., the DH half-key g^y, stores it in `ClientKeyExchange` and derives the *premaster secret* from this and the server's input contained in `ServerKeyExchange`. In our example this would mean computing the DH key g^{xy}. The premaster secret is then hashed together with two previously exchanged challenges to form the *master secret* . The master secret is, as its name implies, the main session key, and all cryptographic keys used for encryption or integrity are derived from this master secret using key derivation functions. The client now sends the `ClientKeyExchange` and, if required by the cipher suite, also `CertificateVerify` and `Certificate` for client authentication to the server. The client then issues a `ChangeCipherSpec` to the Record Protocol, instructing it to use the newly negotiated keys and algorithms. Finally, the client sends the `Finished` message, i.e., a message authentication code (MAC) on the previously sent messages using a newly derived key.

The server derives the premaster secret and the master secret from the data contained in `ClientKeyExchange` and its own inputs. If client authentication is enabled and `CertificateVerify` is present, the server verifies this message to authenticate the client. Finally, verification of the `Finished` message will assure the server of the integrity and freshness of the request.

The server then sends a similar `Finished` message to the client. This allows the client to verify the authenticity of the server and the freshness of the keys used. At this point the client can start sending application data to the server.

3.2 Adding New Cipher Suites

Before presenting the integration of DH-EKE, let us look at the requirements and constraints of integrating a new cipher suite in general. The TLS specifications [Dierks and Allen 1999] do not explicitly mention what is recommended or disallowed in the integration of a new cipher suite. But it is clear that such an integration should be as least intrusive as possible. Examining the data structures defined reveals that the ideal places to adjust TLS for new cipher suites are the `ServerKeyExchange` and `ClientKeyExchange` messages. They are already variant records and can be extended with a new element rather transparently. We can also approach the problem from the other side and look at the hard constraints. For compatibility reasons we should of course not alter messages that are sent before an agreement on a cipher suite has been reached. This means in particular that we should refrain from modifying `ClientHello`. As we will see later this has impor-

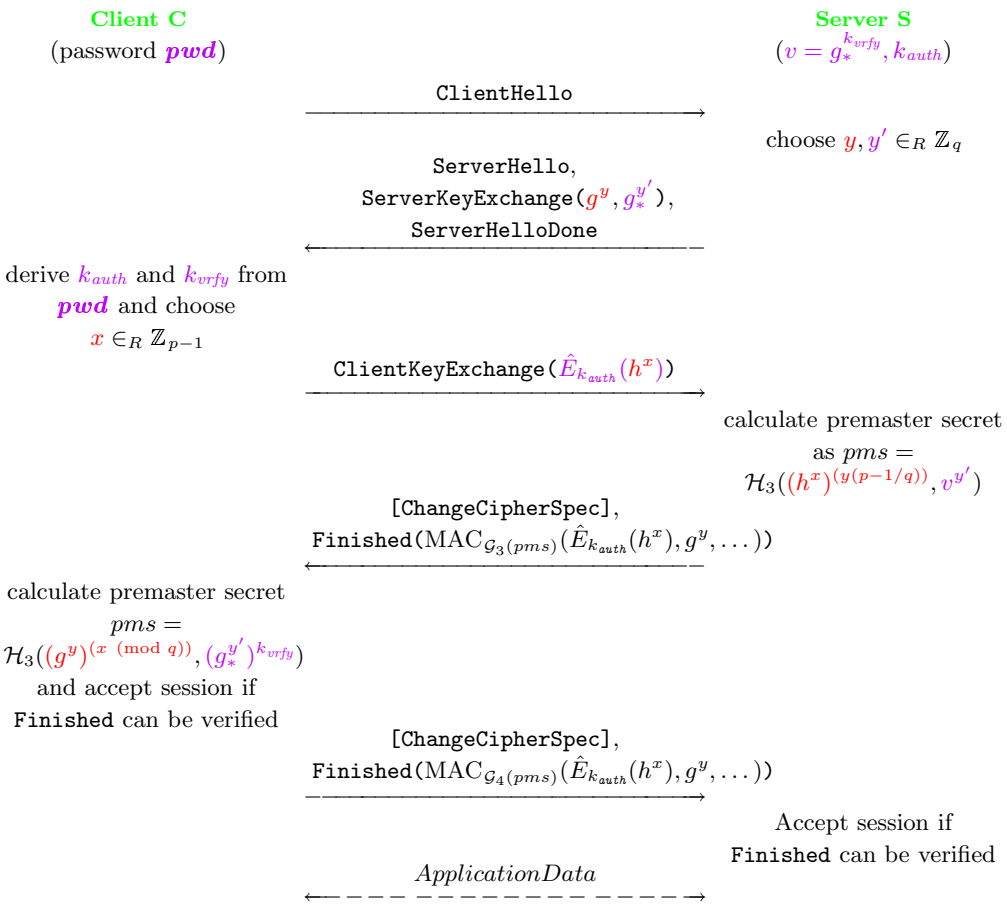

Fig. 7. Overview of flows of DH-EKE/TLS.

tant and unfortunate consequences. Further desirable properties are the reuse of cryptographic primitives already specified by TLS and a minimized setup time by keeping the number of flows and the cost of computation low.

4. INTEGRATION OF DH-EKE IN TLS

Let us now turn our attention to the integration of DH-EKE. Figure 7 gives an overview of the flows assuming that DH-EKE/TLS was among the proposed cipher suites in `ClientHello` and was selected by the server. The arguments of the TLS messages contain the security-critical protocol information in a slightly abstracted and simplified form, e.g., `Finished` is a more complicated function in reality yet for our purpose it is sufficient to consider it as a message authentication code.

At first glance, it looks like a straightforward replacement of the ephemeral DH key exchange authenticated by DSS signatures used as an example when explaining TLS in Section 3.1 by the DH-EKE protocol specified earlier in Section 2 and

shown in Figure 5. However, there a few subtle differences: Some TLS messages from Figure 6 are missing; it is the server who initiates the DH-EKE key-exchange protocol, in contrast to the protocol given in Section 2; the protocol itself is slightly changed; and the order of the client's and server's `Finished` messages is swapped.

The server's `Certificate` and `CertificateRequest` messages and the client's `Certificate` and `CertificateVerify` messages are omitted in Figure 7 for obvious reasons: No PKI is required and thus also no certificates. Note that these messages are specified as optional in the TLS protocol; therefore, omitting them is permissible.

The remaining three differences are all due to the problem of transferring identity information and to the subtle issues of dictionary attacks. However, let us first look at the protocol in more detail and return to these issues later, namely in Section 5.1.

4.1 Setup

The client first chooses a password $\boldsymbol{pwd}$. Then, the client derives a password authentication key $k_{auth} = \mathcal{H}_1(\boldsymbol{pwd}, ID_C, ID_S)$ and computes the password verifier $v = g_*^{k_{vrfy}}$ with $k_{vrfy} = \mathcal{H}_2(\boldsymbol{pwd}, ID_C, ID_S)$. Finally, v and k_{auth} are sent securely to the server and stored together with the client's name in the server's user database.

The functions $\mathcal{H}_1(z, w, w')$ and $\mathcal{H}_2(z, w, w')$ are computed as the first m bits of $PRF(z, \text{“}password\ authentication\ key\text{”}, w|w')$ and $PRF(z, \text{“}password\ verifier\text{”}, w|w')$, respectively. PRF is the pseudo-random function as defined in Section 5 of Dierks and Allen [1999]. It takes as input a secret, an identifying label and a seed, and produces an output of arbitrary length.

4.2 Protocol Flow Processing

In the following, we assume that in `ClientHello` the client proposes some of the DH-EKE cipher suites (see Figure 8 in the Appendix) and the server agrees on one of them. We also omit all standard processing as defined in TLS and refer the reader to Dierks and Allen [1999]. The definition of the new or modified TLS data structures (always written in `typewriter font`) mentioned below can be found in the Appendix in Figure 9.

1. **Client → Server** The client prepares the `ClientHello` as usual.
2. **Server → Client** The server chooses $y \in_R \mathbb{Z}_q$ and computes $g^y \pmod p$. Additionally, the server also chooses $y' \in_R \mathbb{Z}_q$ and computes $g_*^{y'} \pmod p$.
   The server completes the `ServerDHEKEParams` field in `ServerKeyExchange` with g^y and $g_*^{y'}$. If the server's group parameters are not a priori fixed, the server also prepares `ServerDHParamsProof` to allow optimized parameter verification for the client as described in Section 4.3. The server sends the `ServerHello`, `ServerKeyExchange` and `ServerHelloDone` messages to the client.
3. **Client → Server** The Client verifies the parameters of the group: If they are not installed and well-defined, the client performs the tests as outlined in Section 4.3.
   Then, the client verifies that the g^y and $g_*^{y'}$ contained in `ServerKeyExchange` are of the right size and order, i.e., $(g^y)^q \pmod p = 1 \wedge g^y \pmod p \neq 1$. The client aborts if the above conditions are not fulfilled.

The client (software) ask the user for her password and derives the authentication key as $k_{auth} = \mathcal{H}_1(\boldsymbol{pwd}, ID_C, ID_S)$ and the verifier key as $k_{vrfy} = \mathcal{H}_2(\boldsymbol{pwd}, ID_C, ID_S)$. The client chooses $x \in_R \mathbb{Z}_{p-1}$ and computes $h^x \pmod p$. Then, the client encrypts h^x as defined in Section 4.4, enters the resulting value $\hat{E}_{k_{auth}}(h^x)$ as well as the user's identity in the `ClientDHEKEParams` field of the `ClientKeyExchange` message and sends the message to the server.

4. **Server → Client** The server extracts the identity of the client from the `ClientKeyExchange` message and retrieves the client's record from the user database. The server verifies that the account is not locked and decrypts the client's half-key h^x as defined in Section 4.4 using the authentication key k_{auth} stored in the record.
   The server computes the premaster secret pms as $\mathcal{H}_3((h^x)^{(y(p-1/q))}, v^{y'})$ (with $\mathcal{H}_3(z, w)$ defined as $PRF(z, \text{“}DH\ premastersecret\text{”}, w)$ and generates the server's `Finished` message as defined in the TLS specifications, i.e., a message authentication code over all previously sent handshake messages, and idealized in Figure 7 as $\text{MAC}_{\mathcal{G}_i(pms)}(\dots)$. The server performs a `ChangeCipherSpec` and sends the `Finished` message to the client.
5. **Client → Server** The client computes $pms = \mathcal{H}_3((g^y)^{(x \pmod q)}, (g_*^{y'})^{k_{vrfy}})$ to obtain the premaster secret and verifies the server's `Finished` message. If the verification fails, the client aborts.
   The client generates the `Finished` message (again a MAC over all previously exchanged handshake messages), proceeds with the `ChangeCipherSpec` and sends the `Finished` message to the server. Note that contrary to the standard case the client can start sending data immediately after the `Finished` message (and thus retains the original handshake overhead of four flows).
6. **Server → Client** The server verifies the client's `Finished` message. If the verification fails, the server aborts, increments the 'potential online attack' counter in the client's database record and locks the account if the 'potential online attack' counter reaches a threshold (a reasonable number for the threshold might be five. Note that in addition more elaborate policies with exponential retry delays might be used). If the verification is correct, the 'potential online attack' counter is updated (exact procedure depends on local policy: Possibilities are setting it to 0, decrementing it by 1, etc.).

Note: To reduce the risk of password exposure, implementors are advised to throw away (zero out) all traces of the password and all critical random values used (e.g., the DH parameters x, y, g^y and the premaster secret) as soon as possible.

4.3 Group Verification

The group parameters p, q, h and g should preferably be fixed at system startup. If not, they may be chosen by the server and passed to the client in `ServerKeyExchange`. In this case, the client has to verify them. It is of particular importance to make sure that p and q are prime, n and m are sufficiently large and h and g are indeed generators of their respective group. As in the ephemeral case the parameters might be chosen by an adversary, it is not possible to use optimization techniques that drastically reduce the number of Miller-Rabin tests such as the one described in Table 4.3 of Menezes et al. [1997]. Instead we can only rely on

$1/4^t$ as the upper bound of the probability that a candidate is prime after t Miller-Rabin tests: Therefore, at least 40 to 50 tests per prime, i.e., q and p, are required to render the probability negligible that we accept a composite number falsely as prime. The test bases a should be chosen at random and not be predictable by the adversary.

These tests are rather expensive, in particular if we assume light-weight clients. A more efficient way of verification is to let the server send further verification information together with the group parameters. This can help establish the correctness of the parameters more efficiently. The approach chosen here is quite simple. To show the randomness of the prime selection, the server sends, together with the prime, also a pre-image of it taken from a one-way function, i.e., the `ServerDHParamsProof` field. This requires only a small change in the server's prime generation process but should prevent an adversary from choosing weak or special primes. Therefore this randomization allows the use of the optimization techniques described in Menezes et al. [1997], and the number of Miller-Rabin tests on the client side can be reduced down to at most five tests with the given range of n as defined above.

4.4 Encryption using Weak Secrets

In Section 2.4 we described the principles of the encryption function $\hat{E}_P(z)$. In the following, we instantiate that function based on building blocks that already exist in TLS. On input P, a weak secret, and z, an element of $\mathbb{Z}_p^*$, we perform the following steps:

Key Derivation We derive the encryption key k as $\mathcal{H}_0(P, salt)$. The input parameters are the weak secret P and the concatenation of the two challenges found in `ClientHello` and `ServerHello` as *salt*. The function $\mathcal{H}_0(z, w)$ is computed as the first *keylength* bits of $PRF(z, \text{“}Password - derived\ key\text{”}, w)$. *keylength* equals 8 for DES, 16 for 3DES, IDEA and RC4-128, and 5 for RC2. For DES (3DES) the key should be considered as a 64-bit (192-bit) encoding of a 56-bit (168-bit) DES key, with parity bits ignored.

Expansion To prevent dictionary attacks on the encrypted elements (see Section 2.4 for more details) we uniformly expand the element z from an n-bit number to a $(n + \alpha)$-bit number. We form a block b of $(n + \alpha)$ bits as follows:

$\alpha = 50$;
$n = \lceil \log_2 p \rceil$;
$\alpha' := \lfloor (2^{\alpha+n})/p \rfloor$;
$c \in_{\mathcal{R}} \{0, \ldots, \alpha' - 1\}$;
$b := z + cp$; {Note that this calculation is in $\mathbb{Z}$ and not in $\mathbb{Z}_p^*$, i.e., no reduction modulo p.}

Padding If the block length *len* of the encryption scheme does not divide $(n + \alpha)$ then b is padded with $(len - (n + \alpha \pmod{len}))$ random bits to form b'.

Encryption The b' is encrypted using the derived key k. The used shared-key cipher is defined by the agreed cipher suite. It is encoded in the cipher suite name after `TLS_DH_EKE_` and is basically the agreed session encryption cipher if existing (e.g., we would encrypt with RC4/128 if the cipher suite agreed upon is

`TLS_DH_EKE_RC4_128_WITH_RC4_128_SHA`). The list of proposed additional cipher suites is given in Figure 8 in the Appendix. For block ciphers in chaining mode, the Initialization Vector (IV) will be set to all 0.

Decryption An encrypted value is decrypted using the key k derived as defined above in the "Key derivation" step. From the decrypted text, the random padding (if existing) is removed and the resulting value is then reduced (mod p) to undo the expansion.

5. RATIONALES AND EXPLANATIONS

The protocol proposed above takes into account all known attacks [Bellovin and Merrit 1992; Steiner et al. 1995; Jablon 1996; Patel 1997]. In addition, it provides semantic security and at the same time improves the performance. Let us now give more detailed rationales and explanations of certain choices taken during the protocol design.

5.1 Flows

The `ClientHello` message cannot carry the client's identity information.[9] Therefore, the server cannot initiate the key exchange protocol by encrypting its DH half-key as described (with inverted roles) in the protocol in Section 2 and the best possible alternative is to send the half-keys unencrypted.

To prevent dictionary attacks, the party that encrypts with the password should be very careful. That party must never use keys derived from the DH key before it knows that the other party explicitly confirmed knowledge of the password by proving knowledge of the DH key or implicitly by encrypting its own half-key with the password. This rules out using the standard TLS flows. The client, which is the first party to be able to encrypt with the password, cannot send `Finished` before getting a "proof of knowledge of password" from the server, i.e., it is of paramount importance that the client does not use any key derived from the premaster secret *pms* in the Record or Handshake Protocol before the client has successfully received and verified the server's `Finished` message.

Any other approach of not swapping the `Finished` messages would increase the number of flows and deviate even further from the standard TLS messages. The changes in the overall protocol state-machine can nevertheless be kept to a minimum. Note also that there is no penalty in communication delay due to the additional fifth flow in the Handshake Protocol: The client can start to send application data immediately after sending the `Finished` message.

If we exclude altering or misusing `ClientHello` we can actually extend this reasoning and show that it is impossible to build a secure mutually-authenticated key exchange in four flows that relies only on weak secrets. The server, not knowing the client's identity after the first flow, cannot produce any implicit or explicit demonstration of knowledge of the password in the second flow. Consequently the client cannot send any key confirmation in the third flow, and the only way to

[9] At least if we want to retain compatibility with standard TLS and do not resort to changes of `ClientHello` or unacceptable ad-hoc measures such as encoding the identity in the nonce field of the `ClientHello`.

complete client authentication is to send such a message in an additional fifth flow. Thus our protocol is optimal in terms of the number of flows.

5.2 Algebraic Structure

The algebraic group of choice of TLS and also the original basis for DH-EKE is $\mathbb{Z}_p^*$. However, instead of $\mathbb{Z}_p^*$ alternative cyclic groups might be worth considering in the future.

One interesting alternative are the multiplicative groups $GF(2^m)^*$. Computation is quite efficient. Additionally, the problem discussed in Section 2.4.1 disappears: The cardinality of $GF(2^m)^*$ is $2^m - 1$ and a straightforward mapping to m bits would leave only negligible probability of decrypting the only illegal value, i.e., 0, with a wrong password during a dictionary attack. However, further study is necessary to find specific parameters and compare the security and performance with the solution for $\mathbb{Z}_p^*$.

Elliptic curves are another promising alternative. Their advantage is that, for comparable security, the group operations are more efficiently computable and the group elements are smaller, hence bandwidth can be saved. Elliptic curves are also discussed in the context of the IETF TLS working group, but so far no corresponding cipher suites have been adopted. Additionally, the parameter choice and verification are more difficult, and the password encryption function $\hat{E}(\cdot)$ would have to be redesigned. The points on an elliptic curve cannot be mapped bijectively onto a continuous range of integers and therefore expansion cannot be used to circumvent the problem related to the domain size of ordinary cipher discussed in Section 2.4.1. However, a recent proposal by Black and Rogaway [2000] on encrypting finite subsets of arbitrary size together with a dense representation of points such as the one described by Seroussi [1998] might open the door to a more efficient protocol based on elliptic curves.

5.3 Verifiable Parameter Generation

The verification of ephemeral group parameter is based on heuristics. There still remains some degree of freedom for an opponent to find pseudo-primes through pre-computational search. A safer alternative might be to use provable primes generated using the prime-number generation algorithm of Maurer [1995]. The server generates p based on Maurer's algorithm. The primality of q can then be shown as part of the primality proof for p. One drawback of this approach is that messages increase in size and the code becomes more complicated (the current approach can be built on components already existing in most TLS toolkits). Additionally, we can expect a considerable performance impact for this approach.

5.4 Password Encryption

The following design decisions are worth commenting on the choice of the key derivation algorithm and the number of expansion bits.

The key derivation mechanism approximates the recent Version 2.0 of PKCS #5 [RSA 1999] reusing basic TLS building blocks. The salt guarantees that for each protocol run we get independent keys and addresses concerns about interactions between multiple usage of the same key.

Based on the formula $\alpha \approx k + log_2(t_{max})$ given in Section 2.4.1, we choose 2^{30}

for 2^k, the maximally tolerable success probability of a guessing attack, and 2^{20} for t_{max}, the upper bound for the number of protocol runs. This gives us $\alpha = 50$ required expansion bits. With these values we have a wide safety margin in all practical applications: On the one hand, no user will enter his or her password and connect to the server more than 2^{20} times, and the server that tracks failed connection request in its 'potential online attack' counter will foil all attempts to get more samples with active attacks. On the other hand, already $k = 1$ means that an attacker reduces the number of possible passwords at most by half, which in many cases could already be acceptable.

5.5 Why EKE?

We also investigated alternatives to EKE. Whereas many of them have various advantages over DH-EKE, none could match DH-EKE with its minimal impact on TLS: Two additions in `ClientKeyExchange` and `ServerKeyExchange` and a minimal and unavoidable change in the protocol state machine (reversion of the two finished flows) seems to be the smallest change possible to integrate secure password-based protocols into TLS. Below are some more detailed explanations why we rejected other protocols.

5.5.1 *SPEKE.* An alternative protocol is the Simple Password Encrypted Key Exchange (SPEKE) [Jablon 1996]. The protocol is also based on a DH key exchange but instead of encrypting the half-keys with the password it uses the password to derive a generator for a large prime-order subgroup.

It has two main advantages over DH-EKE. On the one hand, the problem due to non-uniform distribution of encrypted elements does not occur and, on the other hand, the adoption of elliptic curves to improve performance is more straightforward.

Unfortunately, integrating SPEKE into TLS cannot be done as easily: As already explained, the `ClientHello` message cannot carry identity information. As the identity of the peer has to be known before anybody can start the protocol, we require more radical changes in the flows, in particular it would require two more messages or changes in the `Finished` messages.

5.5.2 *SRP.* Another prominent proposal is the Secure Remote Password protocol (SRP) [Wu 1998]. While it seems the most efficient system that reduces also the risk when the server database is stolen, it has similar problems with integration as SPEKE does. The protocol cannot be started in flow 2, which means that the handshake would require an additional request-response pair. Taking into account current network delays and the performance of computers today as well as projections on how they will change in the future[10] led us to trade performance for reduced flows.

5.5.3 *What about Protocols relying on Server Public Keys?.* The responder side in TLS is quite often a stand-alone server capable of keeping strong public key pairs. You might wonder if this cannot be exploited to achieve easier and more efficient protocols? Indeed, various protocols [Halevi and Krawczyk 1999; Gong et al. 1993]

[10] Moore's Law still seems valid for the foreseeable future yet latency is already closely bounded by the speed of light.

show how to do this in a provably secure and simple manner. While these protocols are definitely suitable in many applications, there is one major drawback: The client has to obtain the proper public key of the server. One solution is to ask the user for confirmation of a fingerprint as suggested by Halevi and Krawczyk [1999]. This is definitely preferable over fixing the public key in the software, but is quite cumbersome for the user. You might argue that current web browsers already manage root certificates and adding one more is not such a big deal. This is true, but there is the problem of key revocation. Additionally, one should not ignore the fact that it is not very difficult to trick ignorant users into installing bogus root keys to their key ring: Generate your own root CA, build a fancy web site and require https using certificates relying on your own root CA to access it. The likelihood that some user will install this key is rather large. Even worse, you can tell who has installed your root CA certificate if you track user access to the site and the certificate. This allows you to target that user for a man-in-the-middle attack. In fact, an anecdotal incident with a similar man-in-the-middle attack has happened in mid-1998 to a Dutch web banking site. As EKE-like protocols rely less on the user's awareness of the such involved risks, they clearly are a more robust and secure approach.

5.5.4 *Others.* We also considered the protocol proposed by Lucks [1997] and protocols based on collisionful hash [Anderson and Lomas 1994; Bakhtiari et al. 1996]. However, none of their feature was able to outweigh the simplicity of the integration of DH-EKE in TLS.[11]

6. CONCLUSION

We outlined a number of situations in which the current cipher suites of TLS are not completely satisfactory, such as home banking over the web. Secure password-based authenticated key-exchange protocols can improve the situation and can be integrated into TLS in an efficient and non-intrusive manner. We validated our approach by integrating the cipher suite into a in-house toolkit providing the complete SSL3.0 protocol suite. Because of our careful protocol design relying on existing building blocks and the non-intrusive integration of the protocol flows, the adaption of the protocol engine required only few and small changes. Measurements of the performance showed that our cipher suite compares well with other cipher suites. DH-EKE outperformed comparable cipher suites providing mutual authentication and perfect forward secrecy by a factor of up to two (`SSL_DHE_DSS_WITH_DES_CBC_SHA`) and was only slightly slower than the commonly used cipher suite `SSL_RSA_WITH_RC4_128_SHA`.

Moreover, in a modification to the original DH-EKE protocol we showed that the session keys not only can but also should be computed in subgroups of prime order: We achieve better security and as a side effect also improve the performance of DH-EKE. In compliance with the security analysis given in the Appendix of Steiner et al. [1995], we obtain reasonable assurance that the security of our protocols can be founded on the hardness of the Decisional Diffie-Hellman problem.

[11] Since the time of our first publication, MacKenzie, Patel, and Swaminathan [2000] also have found a serious attack on Lucks [1997].

Since the time of building our prototype and our first publication [Buhler et al. 2000], considerable progress has been made regarding the security of password-based key-agreement protocols: Bellare, Pointcheval, and Rogaway [2000] and Boyko, MacKenzie, and Patel [2000] proposed protocols that can be formally proven secure in stronger and more rigid models adapted from Bellare and Rogaway [1995b], Bellare et al. [1998] and Shoup [1999]. As it turns out AuthA [Bellare et al. 2000; Bellare and Rogaway 2000] corresponds to a large extent to our adoption of DH-EKE for TLS, which further validates our approach.

It remains to be seen whether password-based protocols will find a wider adoption in the context of TLS. However, given the problems identified in the introduction and how they can be alleviated using password-based protocols, one can only hope so.

ACKNOWLEDGMENTS

We thank Victor Shoup, Luke O'Connor and Birgit Pfitzmann for their invaluable discussions on cryptographic and algorithmic issues related to this document, and André Adelsbach, N. Asokan and Ahmad-Reza Sadeghi for their detailed comments.

REFERENCES

Anderson, R. J. and Lomas, T. M. A. 1994. Fortifying key negotiation schemes with poorly chosen passwords. *Electronics Letters 30*, 13 (June), 1040–1041.

Bakhtiari, S., Safavi-Naini, R., and Pieprzyk, J. 1996. On password-based authenticated key exchange using collisionful hash functions. In *1st Australasian Conference on Information Security and Privacy (ACISP '96)*, Number 1172 in Lecture Notes in Computer Science (1996), pp. 299–310. Springer-Verlag, Berlin Germany.

Bellare, M., Canetti, R., and Krawczyk, H. 1998. A modular approach to the design and analysis of authentication and key exchange protocols. In *30th Annual Symposium on Theory Of Computing (STOC)* (Dallas, TX, USA, May 1998), pp. 419–428. ACM Press.

Bellare, M., Pointcheval, D., and Rogaway, P. 2000. Authenticated key exchange secure against dictionary attacks. In B. Preneel Ed., *Advances in Cryptology – EUROCRYPT '2000*, Number 1807 in Lecture Notes in Computer Science (Brugge, Belgium, 2000), pp. 139–155. Springer-Verlag, Berlin Germany. Appeared also as Cryptology ePrint Archive Report 2000/014, 28 April, 2000.

Bellare, M. and Rogaway, P. 1993. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby Ed., *1st ACM Conference on Computer and Communications Security* (Fairfax, Virginia, Nov. 1993), pp. 62–73. ACM Press. Appeared also (in identical form) as IBM RC 19619 (87000) 6/22/94.

Bellare, M. and Rogaway, P. 1995a. Optimal asymmetric encryption — how to encrypt with RSA. In A. D. Santis Ed., *Advances in Cryptology – EUROCRYPT '94*, Number 950 in Lecture Notes in Computer Science (1995), pp. 92–111. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany. Final (revised) version appeared November 19, 1995. Available from http://www-cse.ucsd.edu/users/mihir/papers/oaep.html.

Bellare, M. and Rogaway, P. 1995b. Provably secure session key distribution — the three party case. In *27th Annual Symposium on Theory of Computing (STOC)* (May 1995), pp. 57–66. ACM Press.

Bellare, M. and Rogaway, P. 2000. The AuthA protocol for password-based authenticated key exchange. Technical report (March), Contribution to the IEEE P1363 Study Group for Future Public-Key Cryptography Standards.

Bellovin, S. and Merrit, M. 1993. Augmented encrypted key exchange: a password-based protocol secure against dictionary atttacks and password file compromise. In V. Ashby Ed.,

1st ACM Conference on Computer and Communications Security (Fairfax, Virginia, Nov. 1993), pp. 244–250. ACM Press.

Bellovin, S. M. and Merrit, M. 1992. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, CA, May 1992), pp. 72–84. IEEE Computer Society, Technical Committee on Security and Privacy: IEEE Computer Society Press.

Bellovin, S. M. and Merritt, M. 1991. Limitations of the Kerberos authentication system. In *USENIX Conference Proceedings* (Dallas, TX, Winter 1991), pp. 253–267. USENIX. An earlier version of this paper was published in the October, 1990 issue of Computer Communications Review.

Berners-Lee, T., Fielding, R. T., Nielsen, H. F., Gettys, J., and Mogul, J. 1997. Hypertext Transfer Protocol — HTTP/1.1. Internet Request for Comment RFC 2068 (Jan.), Internet Engineering Task Force.

Bishop, M. and Klein, D. V. 1995. Improving system security via proactive password checking. *Computers & Security 14*, 3, 233–249.

Black, J. and Rogaway, P. 2000. Ciphers with arbitrary finite domains. Manuscript. Available from http://www.cs.unr.edu/~jrb/papers.html.

Bleichenbacher, D. 1998. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS. In H. Krawczyk Ed., *Advances in Cryptology – CRYPTO '98*, Number 1462 in Lecture Notes in Computer Science (1998), pp. 1–12. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany.

Boyko, V., MacKenzie, P., and Patel, S. 2000. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel Ed., *Advances in Cryptology – EUROCRYPT '2000*, Number 1807 in Lecture Notes in Computer Science (Brugge, Belgium, 2000), pp. 156–171. Springer-Verlag, Berlin Germany.

Buhler, P., Eirich, T., Steiner, M., and Waidner, M. 2000. Secure password-based cipher suite for TLS. In *Symposium on Network and Distributed Systems Security (NDSS '00)* (San Diego, CA, Feb. 2000), pp. 129–142. Internet Society.

Chappell, D. 1999. Exploring Kerberos, the protocol for distributed security in Windows 2000. *Microsoft Systems Journal 14*, 8 (Aug.).

Dierks, T. and Allen, C. 1999. The TLS protocol version 1.0. Internet Request for Comment RFC 2246 (Jan.), Internet Engineering Task Force. Proposed Standard.

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

Freier, A. O., Kariton, P., and Kocher, P. C. 1996. The SSL protocol: Version 3.0. Internet draft, Netscape Communications.

Gong, L., Lomas, M., Needham, R., and Saltzer, J. 1989. Protecting poorly chosen secrets from guessing attacks. In *Proceedings of the Twelth ACM Symposium on Operating Systems Principles* (Dec. 1989). The Wigwam, Litchfield Park, Arizona. A revised journal version appeared as [Gong et al. 1993].

Gong, L., Lomas, M., Needham, R., and Saltzer, J. 1993. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications 11*, 5 (June), 648–656.

Halevi, S. and Krawczyk, H. 1999. Public-key cryptography and password protocols. *ACM Transactions on Information and System Security 2*, 3, 25–60. Preliminary version in Proc. of the 5th ACM Conference on Computer and Communications Security, 1998, pp. 122-131.

Jablon, D. P. 1996. Strong password-only authenticated key exchange. *Computer Communication Review 26*, 5 (Sept.), 5–26.

Jablon, D. P. 1997. Extended password key exchange protocols immune to dictionary attack. In *Proceedings of the WETICE'97 Workshop on Enterprise Security* (Cambridge, MA, USA, June 1997).

Kaliski, B. and Staddon, J. 1998. PKCS #1: RSA cryptography specifications. Technical note (Sept.), RSA Laboratories. Version 2.0. Published in October 1998 as Internet RFC

2437.

Kohl, J. T. and Neuman, B. C. 1993. The Kerberos network authentication service (V5). Internet Request for Comment RFC 1510, Internet Engineering Task Force.

Lim, C. H. and Lee, P. J. 1997. A key recovery attack on discrete log-based schemes using a prime order subgroup. In B. S. Kaliski, Jr. Ed., *Advances in Cryptology – CRYPTO '97*, Number 1294 in Lecture Notes in Computer Science (1997), pp. 249–263. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany.

Lucks, S. 1997. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Security Protocol Workshop'97* (Ecole Normale Suprieure, Paris, April 1997).

MacKenzie, P., Patel, S., and Swaminathan, R. 2000. Password-authenticated key exchange based on RSA. In T. Okamoto Ed., *Advances in Cryptology – ASIACRYPT '2000*, Number 1976 in Lecture Notes in Computer Science (Kyoto, Japan, 2000), pp. 599–613. International Association for Cryptologic Research: Springer-Verlag, Berlin Germany.

Maurer, U. M. 1995. Fast generation of prime numbers and secure public-key cryptographic parameters. *Journal of Cryptology 8*, 3, 123–155.

Medvinsky, A. and Hur, M. 1999. Addition of Kerberos cipher suites to Transport Layer Security (TLS). Internet Request for Comment RFC 2712 (Oct.), Internet Engineering Task Force.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. 1997. *Handbook of applied cryptography*. CRC Press series on discrete mathematics and its applications. CRC Press. ISBN 0-8493-8523-7.

Mitchell, J., Shmatikov, V., and Stern, U. 1998. Finite-state analysis of SSL 3.0. In *7th USENIX Security Symposium* (San Antonio, Texas, USA, Jan. 1998). USENIX.

Morris, R. and Thompson, K. 1979. Password security: A case history. *Communications of the ACM 22*, 11 (Nov.), 594–597.

Patel, S. 1997. Number theoretic attacks on secure password schemes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy* (Oakland, CA, May 1997), pp. 236–247. IEEE Computer Society, Technical Committee on Security and Privacy: IEEE Computer Society Press.

RSA. 1999. PKCS #5: Password-based cryptography standard. Version 2.0 (March), RSA Laboratories.

Seroussi, G. 1998. Compact representations of elliptic curve points over GF(2^n). Research Contribution to IEEE P1363.

Shoup, V. 1999. On formal models for secure key exchange. Research Report RZ 3120 (#93166) (April), IBM Research. A revised version 4, dated November 15, 1999, is available from http://www.shoup.net/papers/.

Steiner, M., Tsudik, G., and Waidner, M. 1995. Refinement and extension of Encrypted Key Exchange. *ACM Operating Systems Review 29*, 3 (July), 22–30.

Tygar, J. and Whitten, A. 1996. WWW electronic commerce and Java Trojan horses. In *Second USENIX Workshop on Electronic Commerce* (Oakland, California, Nov. 1996), pp. 243–250. USENIX.

Wagner, D. and Schneier, B. 1996. Analysis of the SSL 3.0 protocol. In *Second USENIX Workshop on Electronic Commerce* (Oakland, California, Nov. 1996), pp. 29–40. USENIX.

Wu, T. 1998. The secure remote password protocol. In *Symposium on Network and Distributed Systems Security (NDSS '98)* (San Diego, California, March 1998), pp. 97–111. Internet Society.

Wu, T. 1999. A real-world analysis of Kerberos password security. In *Symposium on Network and Distributed Systems Security (NDSS '99)* (San Diego, CA, Feb. 1999). Internet Society.

Zimmermann, P. R. 1995. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA. ISBN 0-262-74017-6.

APPENDIX

A. DATA STRUCTURES AND DEFINITIONS

In addition to the logical flows and their processing, a standardization of TLS extension also requires the definition of the identifiers of the cipher suites and the necessary additional data structures.

Possible cipher suites for the DH-EKE protocol are proposed in Figure 8, but for obvious reasons no codes have been assigned yet. The nomenclature follows TLS tradition and encodes the involved algorithm in the name: A cipher suite of the form `TLS_DH_EKE_pwdencrypt_WITH_cipher_hash` means that that cipher `pwdencrypt` is used to encrypt the password in the handshake protocol (see Section 4.4) whereas `cipher` and `hash` are used in the record layer as cipher and hash, respectively. Given that the effective entropy of a password is not very high, the use of 3DES for the password-based encryption may seem an overkill. However, keeping `pwdencrypt` equal to `cipher` (in the case it is not `NULL`) is the simpler and more consistent approach than fixing a single cipher or defining all possible permutations.

Figure 9 defines the necessary additional data structures for the `ClientKeyExchange` and `ServerKeyExchange` messages.

CipherSuite	TLS_DH_EKE_DES_CBC_WITH_NULL_SHA
CipherSuite	TLS_DH_EKE_RC4_128_WITH_NULL_MD5
CipherSuite	TLS_DH_EKE_DES_CBC_WITH_DES_CBC_SHA
CipherSuite	TLS_DH_EKE_3DES_EDE_CBC_WITH_3DES_EDE_CBC_SHA
CipherSuite	TLS_DH_EKE_RC4_128_WITH_RC4_128_MD5
CipherSuite	TLS_DH_EKE_IDEA_CBC_WITH_IDEA_CBC_SHA
CipherSuite	TLS_DH_EKE_RC4_128_WITH_NULL_SHA
CipherSuite	TLS_DH_EKE_DES_CBC_WITH_NULL_MD5
CipherSuite	TLS_DH_EKE_DES_CBC_WITH_DES_CBC_MD5
CipherSuite	TLS_DH_EKE_3DES_EDE_CBC_WITH_3DES_EDE_CBC_MD5
CipherSuite	TLS_DH_EKE_RC4_128_WITH_RC4_128_SHA
CipherSuite	TLS_DH_EKE_IDEA_CBC_WITH_IDEA_CBC_MD5

Fig. 8. Proposed Cipher Suites for DH-EKE/TLS.

```
struct {
    select (KeyExchangeAlgorithm) {
        case dh_eke:      /* new option */
            ServerDHEKEParams   params;
        case diffie_hellman:
            ServerDHParams   params;
            Signature   signed_params;
        case rsa:
            ServerRSAParams   params;
            Signature   signed_params;
        } ;
} ServerKeyExchange;

struct {
    ServerDHParams   key_params;
    ServerDHParams   verifier_params;
    ServerDHParamsProof   proof;     /* optional */
} ServerDHEKEParams;     /* new type */

struct {
    seed <0 .. 2^16 − 1>;
} ServerDHParamsProof;     /* new type */

struct {
    select (KeyExchangeAlgorithm) {
        case dh_eke:      /* new option */
            ClientDHEKEParams   params;
        case rsa:
            EncryptedPreMasterSecret;
        case diffie_hellman:
            ClientDiffieHellmanPublic;
    } exchange_keys;
} ClientKeyExchange;

struct {
    String   clientIdentity;
    EncryptedDHParams   params;
} ClientDHEKEParams;     /* new type */

struct {
    password-encrypted   dh_Xs<1 .. 2^16 − 1>;
} EncryptedDHParams;     /* new addition */
```

Fig. 9. Adding DH-EKE/TLS to data structures of TLS.