Network Working Group
Internet-Draft
Expires: August 6, 2004

P. Eronen
Nokia
H. Tschofenig
Siemens
February 6, 2004

Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)
draft-eronen-tls-psk-00.txt

Status of this Memo

Copyright Notice

Abstract

This document specifies new ciphersuites for the Transport Layer Security (TLS) protocol to support authentication based on pre-shared keys. These pre-shared keys are symmetric keys, shared in advance among the communicating parties, and do not require any public key operations.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [KEYWORDS].

1. Introduction

Usually TLS uses public key certificates [TLS] or Kerberos [TLS-KRB] for authentication. This document describes how to use symmetric keys (later called pre-shared keys or PSKs), shared in advance among the communicating parties, to establish a TLS connection.

There are basically two reasons why one might want to do this:

- First, TLS may be used in performance-constrained environments where the CPU power needed for public key operations is not available.

- Second, pre-shared keys may be more convenient from a key management point of view. For instance, in closed environments where the connections are mostly configured manually in advance, it may be easier to configure a PSK than to use certificates. Another case is when the parties already have a mechanism for setting up a shared secret key, and that mechanism could be used to "bootstrap" a key for authenticating a TLS connection.

This document specifies a number of new ciphersuites for TLS. These ciphersuites use a new authentication and key exchange algorithm for PSKs, and re-use existing cipher and MAC algorithms from [TLS] and [TLS-AES].

1.1 Applicability statement

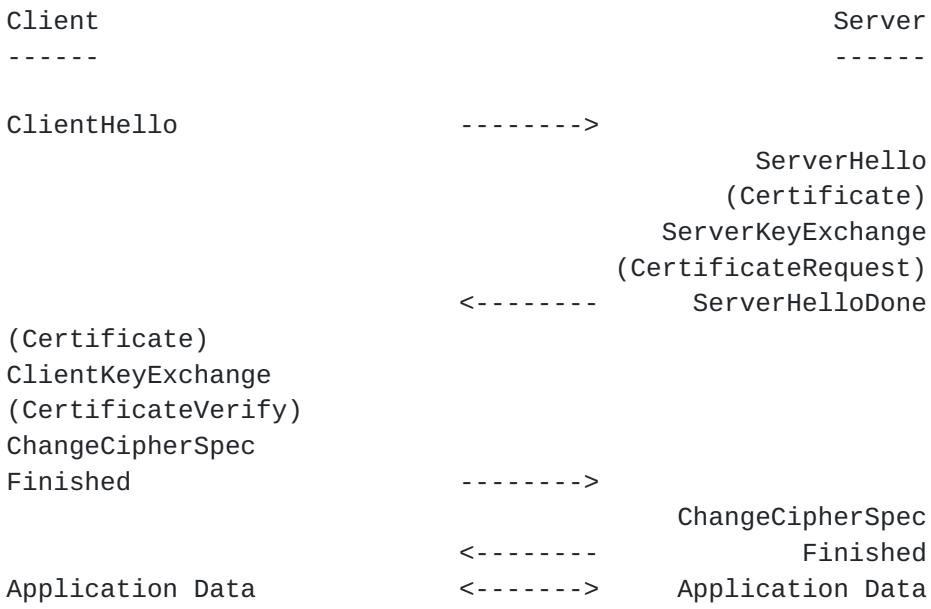
The ciphersuites defined in this document are intended for a rather limited set of applications, usually involving only a very small number of clients and servers. Even in such environments, other alternatives may be more appropriate.

If the main goal is to avoid PKIs, another possibility worth considering is to use self-signed certificates with public key fingerprints. Instead of manually configuring a shared secret in, for instance, some configuration file, a fingerprint (hash) of the other party's public key (or certificate) could be placed there instead.

It is also possible to use SRP for shared secret authentication [TLS-SRP]. However, SRP requires more computational resources and may have some IPR issues. However, it does provide protection against dictionary attacks.

2. Protocol

It is assumed that the reader is familiar with ordinary TLS handshake, shown below. The elements in parenthesis are not included in PSK-based ciphersuites.



The client indicates its willingness to use pre-shared key authentication by including one or more PSK-based ciphersuites in the ClientHello message. The following ciphersuites are defined in this document:

```
CipherSuite TLS_PSK_WITH_RC4_128_SHA        = { 0x00, 0xTBD };
CipherSuite TLS_PSK_WITH_3DES_EDE_CBC_SHA   = { 0x00, 0xTBD };
CipherSuite TLS_PSK_WITH_AES_128_CBC_SHA    = { 0x00, 0xTBD };
CipherSuite TLS_PSK_WITH_AES_256_CBC_SHA    = { 0x00, 0xTBD };
```

Note that this document defines only a new authentication and key exchange algorithm; see [TLS] and [TLS-AES] for description of the cipher and MAC algorithms.

If the TLS server also wants to use pre-shared keys, it selects one of the PSK ciphersuites, places the selected ciphersuite in the ServerHello message, and includes an appropriate ServerKeyExchange message (see below). The Certificate and CertificateRequest payloads are omitted from the response.

Both clients and servers may have pre-shared keys with several different parties. The client indicates which key to use by including a "PSK identity" in the ClientKeyExchange message (note that unlike

in [TLS-SHAREDKEYS], the session_id field in ClientHello message keeps its usual meaning). To help the client in selecting which identity to use, the server can provide a "PSK identity hint" in the ServerKeyExchange message (note that if no hint is provided, a ServerKeyExchange message is still sent).

This document does not specify the format of the PSK identity or PSK identity hint; neither is specified how exactly the client uses the hint (if it uses it at all). The parties have to agree on the identities when the shared secret is configured (however, see Section 4 for related security considerations).

The format of the ServerKeyExchange and ClientKeyExchange messages is shown below.

```
   struct {
       select (KeyExchangeAlgorithm) {
           case diffie_hellman:
               ServerDHParams params;
               Signature signed_params;
           case rsa:
               ServerRSAParams params;
               Signature signed_params;
           case psk:  /* NEW */
               opaque psk_identity_hint<0..2^16-1>;
       };
   } ServerKeyExchange;

   struct {
       select (KeyExchangeAlgorithm) {
           case rsa: EncryptedPreMasterSecret;
           case diffie_hellman: ClientDiffieHellmanPublic;
           case psk: opaque psk_identity<0..2^16-1>;  /* NEW */
       } exchange_keys;
   } ClientKeyExchange;
```

The premaster secret is formed as follows: concatenate 24 zero octets, followed by SHA-1 hash [FIPS180-2] of the PSK itself, followed by 4 zero octets.

Note: This effectively means that only the HMAC-SHA1 part of the TLS PRF is used, and the HMAC-MD5 part is not used. See [Krawczyk20040113] for a more detailed rationale. The PSK is first hashed so that PSKs longer than 24 octets can be used; this is similar to what is done in [HMAC] if the key length is longer than the hash block size.

If the server does not recognize the PSK identity, it SHOULD respond

with a decrypt_error alert message. This alert is also sent if validating the Finished message fails. The use of the same alert message makes it more difficult to find out which PSK identities are known to the server.

3. IANA considerations

This document does not define any new namespaces to be managed by IANA. It does require assignment of several new ciphersuite numbers, but it is unclear how this is done, since the TLS spec does not say who is responsible for assigning them :-)

4. Security Considerations

As with all schemes involving shared keys, special care should be taken to protect the shared values and to limit their exposure over time.

The ciphersuites defined in this document do not provide Perfect Forward Secrecy (PFS). That is, if the shared secret key is somehow compromised, an attacker can decrypt old conversations. (Note that the most popular TLS key exchange algorithm, RSA, does not provide PFS either.)

Use of a fixed shared secret of limited entropy (such as a password) allows an attacker to perform a brute-force or dictionary attack to recover the secret. This may be either an off-line attack (against a captured TLS conversation), or an on-line attack where the attacker tries to connect to the server and tries different keys. Note that the protocol requires the client to prove it knows the key first, so just attempting to connect to a server does not reveal information required for an off-line attack. It is RECOMMENDED that implementations that allow the administrator to manually configure the PSK also provide a functionality for generating a new random PSK, taking [RANDOMNESS] into account.

The PSK identity is sent in cleartext. While using a user name or other similar string as the PSK identity is the most straightforward option, it may lead to problems in some environments since an eavesdropper is able to identify the communicating parties. Even when the identity does not reveal any information itself, reusing the same identity over time may eventually allow an attacker to use traffic analysis to the identify parties. It should be noted that this is no worse than client certificates, since they are also sent in cleartext.

5. Acknowledgments

The protocol defined in this document is heavily based on work by Tim Dierks and Peter Gutmann, and borrows some text from [TLS-SHAREDKEYS] and [TLS-AES]. Valuable feedback was also provided by Peter Gutmann and Mika Tervonen.

When the first version of this draft was almost ready, the authors learned that something similar had been proposed already in 1996 [TLS-PASSAUTH]. However, this draft is not intended for web password authentication, but rather other uses of TLS.

Normative References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[TLS-AES] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.

[TLS] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RANDOMNESS] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[FIPS180-2] National Institute of Standards and Technology, "Specifications for the Secure Hash Standard", Federal Information Processing Standard (FIPS) Publication 180-2, August 2002.

Informative References

[TLS-SHAREDKEYS] Gutmann, P., "Use of Shared Keys in the TLS Protocol", draft-ietf-tls-sharedkeys-02 (work in progress), October 2003.

[HMAC] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[Krawczyk20040113] Krawczyk, H., "Re: TLS shared keys PRF", message on

ietf-tls@lists.certicom.com mailing list 2004-01-13, http://www.imc.org/ietf-tls/mail-archive/msg04098.html.

[TLS-KRB] Medvinsky, A. and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712, October 1999.

[TLS-PASSAUTH] Simon, D., "Addition of Shared Key Authentication to Transport Layer Security (TLS)", draft-ietf-tls-passauth-00 (expired), November 1996.

[TLS-SRP] Taylor, D., Wu, T., Mavroyanopoulos, N. and T. Perrin, "Using SRP for TLS Authentication", draft-ietf-tls-srp-06 (work in progress), January 2004.

Authors' Addresses

Pasi Eronen
Nokia Research Center
P.O. Box 407
FIN-00045 Nokia Group
Finland

EMail: pasi.eronen@nokia.com

Hannes Tschofenig
Siemens
Otto-Hahn-Ring 6
Munich, Bayern 81739
Germany

EMail: Hannes.Tschofenig@siemens.com

Appendix A. Comparison with draft-ietf-tls-sharedkeys-02 (informative)

[TLS-SHAREDKEYS] presents another way to use shared keys with TLS. Instead of defining new ciphersuites, it re-uses the TLS session cache and session resumption functionality.

The approach presented in this document is, in our opinion, more elegant and better in line with the design of TLS. However, it does probably require more changes to existing TLS implementations. Nevertheless, these changes should be rather straightforward, especially for implementations that already support multiple key exchange algorithms and have a modular architecture.

The changes required are roughly the following:

1. An API to pass psk_identities and keys around from the application to the TLS library. Most likely, both push-style interface (use this psk_identity and key) and callbacks (given a psk_identity, fetch corresponding shared secret) would be useful.

2. An API to determine which psk_identity was used for a session.

3. PSK ciphersuite identifiers must be added to the list of supported ciphersuites.

4. Processing of PSK messages in the handshake code.

The session-cache based approach probably requires the following changes (depending on details of the TLS implementation):

1. Most TLS implementations do not expose an API that allows detailed modification of the session cache, so some modifications are required (especially if the implementation is done in some reasonably type-safe language, the application cannot just use some pointer tricks to access private data structures).

   On the client side, we need an API to communicate session_id, key and whatever is used to look up entries from the session cache (for instance, some implementations use IP address and port number) to the TLS implementation (and initialize other session cache fields to some sensible values).

   On the server side, we need to communicate session_id and key. Most likely, both push-style interface (use this session_id and key) and pull callbacks (given a session_id, fetch corresponding shared secret) would be useful (but callbacks may require more changes).

2. An API to determine which session_id was used (and to determine if shared secret or normal RSA was used).

3. The session resumption code normally checks that resumed sessions use the same ciphersuite as the original session. Unless a single ciphersuite is hardcoded to the session cache, this code has to be modified (and the session cache needs a flag indicating which entries were created using ordinary handshake and which using shared-secret API--unless the check is omitted for all sessions, breaking TLS 1.0 rules).

4. If the TLS implementation supports compression, resumed sessions must use the same compression method as the original. Either compressions has to be disabled or this code modified.

5. TLS implementation should also check that the resumed session uses the same protocol version; this needs changes as well, unless a single version number is hardcoded.

6. The session cache code may need modifications to ensure the stored entries actually stay there long enough to be useful. Currently implementations are free to discard entries whenever they want. However, probably most implementations would not require any changes.

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.