Network Working Group
Request for Comments: 4279
Category: Standards Track

P. Eronen, Ed.
Nokia
H. Tschofenig, Ed.
Siemens

December 2005

Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)

Status of This Memo

Copyright Notice

Abstract

This document specifies three sets of new ciphersuites for the Transport Layer Security (TLS) protocol to support authentication based on pre-shared keys (PSKs). These pre-shared keys are symmetric keys, shared in advance among the communicating parties. The first set of ciphersuites uses only symmetric key operations for authentication. The second set uses a Diffie-Hellman exchange authenticated with a pre-shared key, and the third set combines public key authentication of the server with pre-shared key authentication of the client.

Table of Contents

1. Introduction

Usually, TLS uses public key certificates [TLS] or Kerberos [KERB] 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, using pre-shared keys can, depending on the ciphersuite, avoid the need for public key operations. This is useful if TLS is used in performance-constrained environments with limited CPU power.

- 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 three sets of new ciphersuites for TLS. These ciphersuites use new key exchange algorithms, and reuse existing cipher and MAC algorithms from [TLS] and [AES]. A summary of these ciphersuites is shown below.

CipherSuite	Key Exchange	Cipher	Hash
TLS_PSK_WITH_RC4_128_SHA	PSK	RC4_128	SHA
TLS_PSK_WITH_3DES_EDE_CBC_SHA	PSK	3DES_EDE_CBC	SHA
TLS_PSK_WITH_AES_128_CBC_SHA	PSK	AES_128_CBC	SHA
TLS_PSK_WITH_AES_256_CBC_SHA	PSK	AES_256_CBC	SHA
TLS_DHE_PSK_WITH_RC4_128_SHA	DHE_PSK	RC4_128	SHA
TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA	DHE_PSK	3DES_EDE_CBC	SHA
TLS_DHE_PSK_WITH_AES_128_CBC_SHA	DHE_PSK	AES_128_CBC	SHA
TLS_DHE_PSK_WITH_AES_256_CBC_SHA	DHE_PSK	AES_256_CBC	SHA
TLS_RSA_PSK_WITH_RC4_128_SHA	RSA_PSK	RC4_128	SHA
TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA	RSA_PSK	3DES_EDE_CBC	SHA
TLS_RSA_PSK_WITH_AES_128_CBC_SHA	RSA_PSK	AES_128_CBC	SHA
TLS_RSA_PSK_WITH_AES_256_CBC_SHA	RSA_PSK	AES_256_CBC	SHA

The ciphersuites in Section 2 (with PSK key exchange algorithm) use only symmetric key algorithms and are thus especially suitable for performance-constrained environments.

The ciphersuites in Section 3 (with DHE_PSK key exchange algorithm) use a PSK to authenticate a Diffie-Hellman exchange. These ciphersuites protect against dictionary attacks by passive eavesdroppers (but not active attackers) and also provide Perfect Forward Secrecy (PFS).

The ciphersuites in Section 4 (with RSA_PSK key exchange algorithm) combine public-key-based authentication of the server (using RSA and certificates) with mutual authentication using a PSK.

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 Public-Key Infrastructures (PKIs), another possibility worth considering is using 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 the SRP (Secure Remote Password) ciphersuites for shared secret authentication [SRP]. SRP was designed to be used with passwords, and it incorporates protection against dictionary attacks. However, it is computationally more expensive than the PSK ciphersuites in Section 2.

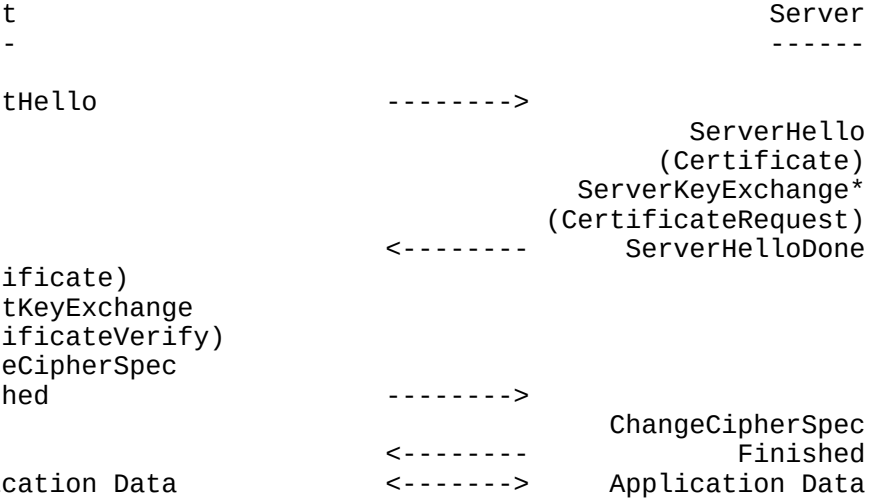
1.2. Conventions Used in This Document

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].

2. PSK Key Exchange Algorithm

This section defines the PSK key exchange algorithm and associated ciphersuites. These ciphersuites use only symmetric key algorithms.

It is assumed that the reader is familiar with the ordinary TLS handshake, shown below. The elements in parenthesis are not included when the PSK key exchange algorithm is used, and "*" indicates a situation-dependent message that is not always sent.



The client indicates its willingness to use pre-shared key authentication by including one or more PSK ciphersuites in the ClientHello message. 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 [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. If no hint is provided, the ServerKeyExchange message is omitted. See Section 5 for a more detailed description of these fields.

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

```
struct {
    select (KeyExchangeAlgorithm) {
        /* other cases for rsa, diffie_hellman, etc. */
        case psk:  /* NEW */
            opaque psk_identity_hint<0..2^16-1>;
    };
} ServerKeyExchange;

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

The premaster secret is formed as follows: if the PSK is N octets long, concatenate a uint16 with the value N, N zero octets, a second uint16 with the value N, and the PSK itself.

Note 1: All the ciphersuites in this document share the same general structure for the premaster secret, namely,

```
struct {
    opaque other_secret<0..2^16-1>;
    opaque psk<0..2^16-1>;
};
```

Here "other_secret" either is zeroes (plain PSK case) or comes from the Diffie-Hellman or RSA exchange (DHE_PSK and RSA_PSK, respectively). See Sections 3 and 4 for a more detailed description.

Note 2: Using zeroes for "other_secret" effectively means that only the HMAC-SHA1 part (but not the HMAC-MD5 part) of the TLS PRF

is used when constructing the master secret. This was considered more elegant from an analytical viewpoint than, for instance, using the same key for both the HMAC-MD5 and HMAC-SHA1 parts. See [KRAWCZYK] for a more detailed rationale.

The TLS handshake is authenticated using the Finished messages as usual.

If the server does not recognize the PSK identity, it MAY respond with an "unknown_psk_identity" alert message. Alternatively, if the server wishes to hide the fact that the PSK identity was not known, it MAY continue the protocol as if the PSK identity existed but the key was incorrect: that is, respond with a "decrypt_error" alert.

3. DHE_PSK Key Exchange Algorithm

This section defines additional ciphersuites that use a PSK to authenticate a Diffie-Hellman exchange. These ciphersuites give some additional protection against dictionary attacks and also provide Perfect Forward Secrecy (PFS). See Section 7 for discussion of related security considerations.

When these ciphersuites are used, the ServerKeyExchange and ClientKeyExchange messages also include the Diffie-Hellman parameters. The PSK identity and identity hint fields have the same meaning as in the previous section (note that the ServerKeyExchange message is always sent, even if no PSK identity hint is provided).

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

```
struct {
    select (KeyExchangeAlgorithm) {
        /* other cases for rsa, diffie_hellman, etc. */
        case diffie_hellman_psk:  /* NEW */
            opaque psk_identity_hint<0..2^16-1>;
            ServerDHParams params;
    };
} ServerKeyExchange;

struct {
    select (KeyExchangeAlgorithm) {
        /* other cases for rsa, diffie_hellman, etc. */
        case diffie_hellman_psk:   /* NEW */
            opaque psk_identity<0..2^16-1>;
            ClientDiffieHellmanPublic public;
    } exchange_keys;
} ClientKeyExchange;
```

The premaster secret is formed as follows. First, perform the Diffie-Hellman computation in the same way as for other Diffie-Hellman-based ciphersuites in [TLS]. Let Z be the value produced by this computation (with leading zero bytes stripped as in other Diffie-Hellman-based ciphersuites). Concatenate a uint16 containing the length of Z (in octets), Z itself, a uint16 containing the length of the PSK (in octets), and the PSK itself.

This corresponds to the general structure for the premaster secrets (see Note 1 in Section 2) in this document, with "other_secret" containing Z.

4. RSA_PSK Key Exchange Algorithm

The ciphersuites in this section use RSA and certificates to authenticate the server, in addition to using a PSK.

As in normal RSA ciphersuites, the server must send a Certificate message. The format of the ServerKeyExchange and ClientKeyExchange messages is shown below. If no PSK identity hint is provided, the ServerKeyExchange message is omitted.

```
struct {
    select (KeyExchangeAlgorithm) {
        /* other cases for rsa, diffie_hellman, etc. */
        case rsa_psk:  /* NEW */
            opaque psk_identity_hint<0..2^16-1>;
    };
} ServerKeyExchange;

struct {
    select (KeyExchangeAlgorithm) {
        /* other cases for rsa, diffie_hellman, etc. */
        case rsa_psk:   /* NEW */
            opaque psk_identity<0..2^16-1>;
            EncryptedPreMasterSecret;
    } exchange_keys;
} ClientKeyExchange;
```

The EncryptedPreMasterSecret field sent from the client to the server contains a 2-byte version number and a 46-byte random value, encrypted using the server's RSA public key as described in Section 7.4.7.1 of [TLS]. The actual premaster secret is formed by both parties as follows: concatenate a uint16 with the value 48, the 2-byte version number and the 46-byte random value, a uint16 containing the length of the PSK (in octets), and the PSK itself. (The premaster secret is thus 52 octets longer than the PSK.)

This corresponds to the general structure for the premaster secrets (see Note 1 in Section 2) in this document, with "other_secret" containing both the 2-byte version number and the 46-byte random value.

Neither the normal RSA ciphersuites nor these RSA_PSK ciphersuites themselves specify what the certificates contain (in addition to the RSA public key), or how the certificates are to be validated. In particular, it is possible to use the RSA_PSK ciphersuites with unvalidated self-signed certificates to provide somewhat similar protection against dictionary attacks, as the DHE_PSK ciphersuites define in Section 3.

5. Conformance Requirements

It is expected that different types of identities are useful for different applications running over TLS. This document does not therefore mandate the use of any particular type of identity (such as IPv4 address or Fully Qualified Domain Name (FQDN)).

However, the TLS client and server clearly have to agree on the identities and keys to be used. To improve interoperability, this document places requirements on how the identity is encoded in the protocol, and what kinds of identities and keys implementations have to support.

The requirements for implementations are divided into two categories, requirements for TLS implementations and management interfaces. In this context, "TLS implementation" refers to a TLS library or module that is intended to be used for several different purposes, while "management interface" would typically be implemented by a particular application that uses TLS.

This document does not specify how the server stores the keys and identities, or how exactly it finds the key corresponding to the identity it receives. For instance, if the identity is a domain name, it might be appropriate to do a case-insensitive lookup. It is RECOMMENDED that before looking up the key, the server processes the PSK identity with a stringprep profile [STRINGPREP] appropriate for the identity in question (such as Nameprep [NAMEPREP] for components of domain names or SASLprep for usernames [SASLPREP]).

5.1. PSK Identity Encoding

The PSK identity MUST be first converted to a character string, and then encoded to octets using UTF-8 [UTF8]. For instance,

- IPv4 addresses are sent as dotted-decimal strings (e.g., "192.0.2.1"), not as 32-bit integers in network byte order.

- Domain names are sent in their usual text form [DNS] (e.g., "www.example.com" or "embedded\.dot.example.net"), not in DNS protocol format.

- X.500 Distinguished Names are sent in their string representation [LDAPDN], not as BER-encoded ASN.1.

This encoding is clearly not optimal for many types of identities. It was chosen to avoid identity-type-specific parsing and encoding code in implementations where the identity is configured by a person using some kind of management interface. Requiring such identity-type-specific code would also increase the chances for interoperability problems resulting from different implementations supporting different identity types.

5.2. Identity Hint

In the absence of an application profile specification specifying otherwise, servers SHOULD NOT provide an identity hint and clients MUST ignore the identity hint field. Applications that do use this field MUST specify its contents, how the value is chosen by the TLS server, and what the TLS client is expected to do with the value.

5.3. Requirements for TLS Implementations

TLS implementations supporting these ciphersuites MUST support arbitrary PSK identities up to 128 octets in length, and arbitrary PSKs up to 64 octets in length. Supporting longer identities and keys is RECOMMENDED.

5.4. Requirements for Management Interfaces

In the absence of an application profile specification specifying otherwise, a management interface for entering the PSK and/or PSK identity MUST support the following:

- Entering PSK identities consisting of up to 128 printable Unicode characters. Supporting as wide a character repertoire and as long identities as feasible is RECOMMENDED.

- Entering PSKs up to 64 octets in length as ASCII strings and in hexadecimal encoding.

6. IANA Considerations

IANA does not currently have a registry for TLS ciphersuite or alert numbers, so there are no IANA actions associated with this document.

For easier reference in the future, the ciphersuite numbers defined in this document are summarized below.

```
CipherSuite TLS_PSK_WITH_RC4_128_SHA          = { 0x00, 0x8A };
CipherSuite TLS_PSK_WITH_3DES_EDE_CBC_SHA     = { 0x00, 0x8B };
CipherSuite TLS_PSK_WITH_AES_128_CBC_SHA      = { 0x00, 0x8C };
CipherSuite TLS_PSK_WITH_AES_256_CBC_SHA      = { 0x00, 0x8D };
CipherSuite TLS_DHE_PSK_WITH_RC4_128_SHA      = { 0x00, 0x8E };
CipherSuite TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA = { 0x00, 0x8F };
CipherSuite TLS_DHE_PSK_WITH_AES_128_CBC_SHA  = { 0x00, 0x90 };
CipherSuite TLS_DHE_PSK_WITH_AES_256_CBC_SHA  = { 0x00, 0x91 };
CipherSuite TLS_RSA_PSK_WITH_RC4_128_SHA      = { 0x00, 0x92 };
CipherSuite TLS_RSA_PSK_WITH_3DES_EDE_CBC_SHA = { 0x00, 0x93 };
CipherSuite TLS_RSA_PSK_WITH_AES_128_CBC_SHA  = { 0x00, 0x94 };
CipherSuite TLS_RSA_PSK_WITH_AES_256_CBC_SHA  = { 0x00, 0x95 };
```

This document also defines a new TLS alert message, unknown_psk_identity(115).

7. 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.

7.1. Perfect Forward Secrecy (PFS)

The PSK and RSA_PSK ciphersuites defined in this document do not provide Perfect Forward Secrecy (PFS). That is, if the shared secret key (in PSK ciphersuites), or both the shared secret key and the RSA private key (in RSA_PSK ciphersuites), is somehow compromised, an attacker can decrypt old conversations.

The DHE_PSK ciphersuites provide Perfect Forward Secrecy if a fresh Diffie-Hellman private key is generated for each handshake.

7.2. Brute-Force and Dictionary Attacks

Use of a fixed shared secret of limited entropy (for example, a PSK that is relatively short, or was chosen by a human and thus may contain less entropy than its length would imply) may allow 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 handshake messages) or an on-line attack where the attacker attempts to connect to the server and tries different keys.

For the PSK ciphersuites, an attacker can get the information required for an off-line attack by eavesdropping on a TLS handshake, or by getting a valid client to attempt connection with the attacker (by tricking the client to connect to the wrong address, or by intercepting a connection attempt to the correct address, for instance).

For the DHE_PSK ciphersuites, an attacker can obtain the information by getting a valid client to attempt connection with the attacker. Passive eavesdropping alone is not sufficient.

For the RSA_PSK ciphersuites, only the server (authenticated using RSA and certificates) can obtain sufficient information 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.

7.3. Identity Privacy

The PSK identity is sent in cleartext. Although 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 perform traffic analysis to identify the parties. It should be noted that this is no worse than client certificates, since they are also sent in cleartext.

7.4. Implementation Notes

The implementation notes in [TLS11] about correct implementation and use of RSA (including Section 7.4.7.1) and Diffie-Hellman (including Appendix F.1.1.3) apply to the DHE_PSK and RSA_PSK ciphersuites as well.

8. Acknowledgements

The protocol defined in this document is heavily based on work by Tim Dierks and Peter Gutmann, and borrows some text from [SHAREDKEYS] and [AES]. The DHE_PSK and RSA_PSK ciphersuites are based on earlier work in [KEYEX].

Valuable feedback was also provided by Bernard Aboba, Lakshminath Dondeti, Philip Ginzboorg, Peter Gutmann, Sam Hartman, Russ Housley, David Jablon, Nikos Mavroyanopoulos, Bodo Moeller, Eric Rescorla, and Mika Tervonen.

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

9. References

9.1. Normative References

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

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RANDOMNESS] Eastlake, D., 3rd, Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

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

[UTF8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

9.2. Informative References

[DNS] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

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

[KEYEX] Badra, M., Cherkaoui, O., Hajjeh, I. and A. Serhrouchni, "Pre-Shared-Key key Exchange methods for TLS", Work in Progress, August 2004.

[KRAWCZYK] 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.

[LDAPDN] Zeilenga, K., "LDAP: String Representation of Distinguished Names", Work in Progress, February 2005.

[NAMEPREP] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[PASSAUTH] Simon, D., "Addition of Shared Key Authentication to Transport Layer Security (TLS)", Work in Progress, November 1996.

[SASLPREP] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[SHAREDKEYS] Gutmann, P., "Use of Shared Keys in the TLS Protocol", Work in Progress, October 2003.

[SRP] Taylor, D., Wu, T., Mavroyanopoulos, N. and T. Perrin, "Using SRP for TLS Authentication", Work in Progress, March 2005.

[STRINGPREP] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[TLS11] Dierks, T. and E. Rescorla, "The TLS Protocol Version 1.1", Work in Progress, June 2005.

Authors' and Contributors' 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

Mohamad Badra
ENST Paris
46 rue Barrault
75634 Paris
France

EMail: Mohamad.Badra@enst.fr

Omar Cherkaoui
UQAM University
Montreal (Quebec)
Canada

EMail: cherkaoui.omar@uqam.ca

Ibrahim Hajjeh
ESRGroups
17 passage Barrault
75013 Paris
France

EMail: Ibrahim.Hajjeh@esrgroups.org

Ahmed Serhrouchni
ENST Paris
46 rue Barrault
75634 Paris
France

EMail: Ahmed.Serhrouchni@enst.fr

Full Copyright Statement

Intellectual Property

Acknowledgement

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