

TLS Working Group
Internet-Draft
Expires: December 2003

P.Gutmann
University of Auckland
June 2003

Use of Shared Keys in the TLS Protocol
draft-ietf-tls-sharedkeys-01

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of [Section 10 of RFC2026](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet- Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

1. Abstract

The TLS handshake requires the use of CPU-intensive public-key algorithms with a considerable overhead in resource-constrained environments or ones such as mainframes where users are charged for CPU time. This document describes a means of employing TLS using symmetric keys or passwords shared in advance among communicating parties. No modifications or alterations to the TLS protocol are required for this process.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document (in uppercase, as shown) are to be interpreted as described in [[RFC 2119](#)].

2. Problem analysis

TLS is frequently used with devices with little CPU power available, for example mobile and embedded devices. In these situations the initial TLS handshake can take as long as half a minute with a 1Kbit RSA key. In many cases a fully general public-key-based handshake is unnecessary, since the

device is only syncing to a host PC or contacting a fixed base station, which would allow a pre-shared symmetric key to be used instead.

In a slight variation of this case, CPU power is available but is too expensive to devote to public-key operations. This situation is common in mainframe environments, where users are charged for CPU time. As with mobile devices, mainframe-to-mainframe or client-to-mainframe communications are generally fixed in advance, allowing shared symmetric keys to be employed.

In order to solve these problems, we require a means of eliminating the expensive public-key operations in the TLS handshake, while providing an equivalent level of security using shared symmetric keys. The solution is fairly straightforward. Observe that after the initial handshake phase, TLS is operating with a quantity of symmetric keying material derived from the information exchanged during the initial handshake. Using shared symmetric keys involves explicitly deriving the TLS master secret from the shared key, rather than sharing it implicitly via the public-key-based key agreement process. TLS already contains such a mechanism built into the protocol in the form of the session cacheing mechanism, which allows a TLS session to be resumed without requiring a full public-key-based re-handshake.

The solution to the problem then is obvious: We need to seed the TLS session cache with the shared symmetric key. When the client connects, the session cacheing mechanism takes over and the client and server "resume" the phantom session created by seeding the cache. This mechanism requires an absolute minimum of code changes to existing TLS implementations (it can be bolted onto any existing TLS engine without needing to change the engine itself), and no changes to the TLS protocol itself.

[2.1](#) Design considerations

In order to work within the existing TLS protocol design, we require a means of identifying a particular session (the session ID in TLS terminology), and the keying material required to protect the session. The { ID, key } combination is analogous to the { user name, password } combination traditionally used to secure access to computer systems.

In TLS, the session ID is a variable-length value of up to 32 bytes, but is typically 32 or less frequently 16 bytes long. For our use we don't really care about its form. A (somewhat unsound) practice would be to use the user name as the session ID. A more secure alternative would be to employ a value derived from the user name in such a way that it can't be directly connected to it, for example a MAC of the user name.

Normally the exact format of the session ID is determined explicitly by the server and remembered by the client for use during session resumption. However, when "resuming" a phantom session in the manner described here, both the client and the server must be able to implicitly generate identical session ID values in order to identify the phantom session to be resumed. To create a canonical session ID value, we pad the variable-length value out to a fixed length by appending zero bytes.

The TLS master secret is a 48-byte value, which is unlikely to correspond to the value of the shared symmetric key or password, which would typically be a 128-bit key or a text password/passphrase. In order to transform this into the type of keying material required by TLS, we need to apply the TLS pseudorandom function (PRF) to produce the master secret with which we seed the session cache. The shared secret thus takes the place of the 48-byte premaster secret usually used to derive the master secret. As with the variable-length session ID, we need to canonicalise the variable-length secret by zero-padding it to the standard 48-byte length for the premaster secret.

Finally, we need a means of injecting the resulting session ID and master secret into the session cache. This is the only modification required to existing TLS implementations. Once the cache is seeded, all further details are handled automatically by the TLS protocol.

It should be noted that this mechanism is best suited for situations where a small number of clients/servers are communicating. While seeding a session cache with IDs and keys for 10,000 different users is certainly possible, this is rather wasteful of server resources, not to mention the accompanying key management nightmare involved in handling such a large number of shared symmetric keys.

[3. TLS using shared keys](#)

[Note: The following is phrased fairly informally, since it's really an application note rather than a standards-track RFC]

Before any exchange takes place, the client and server session caches are seeded with a session ID identifying the user/session, and a master secret derived from the shared secret key or password/passphrase. The exact form of the data used to create the session ID is application specific (but see the comment in the security considerations). The data used to create the session ID is zero-padded to 16 bytes (128 bits) if necessary to meet the requirements given in [section 2.1](#). In C this may be expressed as:

```
memset( session_id, 0, 16 );
memcpy( session_id, input_data, min( input_data_length, 16 ) );
```

The master secret used to seed the cache is computed in the standard manner using the TLS PRF:

```
master_secret = PRF(shared_secret, "shared secret", "")[0..47];
```

The shared secret or password/passphrase takes the place of the premaster secret that is normally used at this point, padded with zero bytes if necessary to the standard length of 48 bytes. In C this may be expressed as:

```
memset( premaster_secret, 0, 48 );
memcpy( premaster_secret, shared_secret,
        min( shared_secret_length, 48 ) );
```

This formats the shared secret in a manner that allows it to be used directly in place of the standard premaster secret derived from the public-key-based key agreement process.

The 'seed' component of the calculation (normally occupied by the client and server nonces) is empty in this case, however applications may choose to use an application or system-specific value to ensure that the same shared secret used with another application or system yields a different master secret. Note that the use of the client and server nonces will always produce different keys for each session, even if the same master secret is employed.

The final step involves injecting the session ID and master secret into the session cache. This is an implementation-specific issue beyond the scope of this document. All further steps are handled automatically by the TLS protocol, which will "resume" the phantom session created by the above steps without going through the full public-key-based handshake.

Session cache entries are normally expired after a given amount of time, or overwritten on an LRU basis. In order to prevent shared secret-based entries from vanishing after a certain amount of time, these cache entries will need to be distinguished from standard cache entries and made more persistent than the latter, for example by giving them a longer expiry time when they are added or periodically touching them to update their last-access time. Again, this is an implementation issue beyond the scope of this document.

[3.1](#) Use of shared keys with SSLv3

If this key management mechanism is used with an implementation that supports SSLv3 alongside TLS (as most do), the TLS PRF must be used for both SSLv3 and TLS. This is required in order to allow the mechanism to function for both SSLv3 and TLS, since using different PRFs would require a different session ID for each PRF used.

[Note: Is this a good idea? It makes administration simpler, but also makes some rollback attacks possible. If rollback attacks are a major concern we could also force the first 2 bytes of the pseudo-premaster secret to the version number as for the real premaster secret]

[3.2](#) Test vectors

The following test vectors are derived from the transformation of the password "test" into a master_secret value to be added to the session cache:

Shared secret:

74 65 73 74 ("test")

Shared secret zero-padded to 48-byte premaster secret size:

[74](#) 65 73 74 00 00 00 00 ... 00 00

Master secret added to session cache:

```
F9 74 60 1A F7 02 49 3B
C5 A1 06 C0 C3 11 0F 18
60 FD E0 4D 15 29 60 FE
9F 83 61 84 1E 91 5C AB
8A CB F5 75 C4 E2 33 1D
4B 2F F9 F4 C3 9B 4B C1
```

4. Security considerations

The session ID used to identify sessions is visible to observers. While using a user name as the session ID is the most straightforward option, it may lead to problems with traffic analysis, with an attacker being able to track the identities of communicating parties. In addition since the session ID is reused over time, traffic analysis may eventually allow an attacker to identify parties even if an opaque session ID is used. [RFC 2246] contains a similar warning about the contents of session IDs with TLS in general. It should be noted though that even a worse-case non-opaque session ID results in no more exposure than the use of client certificates during a handshake.

As with all schemes involving shared keys, special care should be taken to protect the shared values and to limit their exposure over time. Documents covering other shared-key protocols such as Kerberos [RFC 1510] contain various security suggestions in this regard.

Use of a fixed shared secret of limited entropy (for example a password) allows an attacker to perform an online password-guessing attack by trying to resume a session with a master secret derived from each possible password. This results in a fatal `decrypt_error` alert (or some equivalent such as `handshake_failure` or `bad_record_mac`) which makes the session non-resumable (that is, it clears the phantom session from the session cache). Implementations should limit the enthusiasm with which they re-seed the session cache after such an event; standard precautions against online password-guessing attacks apply.

References (Normative)

[RFC 2119] "Key words for use in RFCs to Indicate Requirement Levels", Scott Bradner, [RFC 2119](#), March 1997.

[RFC 2246] "The TLS Protocol", [RFC 2246](#), Tim Dierks and Christopher Allen, January 1999.

References (Informative)

[RFC 1510] "The Kerberos Network Authentication Service (V5)", [RFC 1510](#), John Kohl and B. Clifford Neuman, September 1993.

Author's Address

Peter Gutmann
University of Auckland
Private Bag 92019
Auckland, New Zealand

Email: pgut001@cs.auckland.ac.nz

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

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