OTP Methods for TLS

Draft 22~~11~~

RSA Laboratories

~~January 20~~March 27, 2006

Editor's note: This is the ~~first~~ second draft of this document. Please send comments and suggestions to: otps@rsasecurity.com or otps-editor@rsasecurity.com.

TABLE OF CONTENTS

1 Introduction

1.1 Scope

This document describes means for applying One-Time Password (OTP) methods to authenticate Transport Layer Security (TLS) [DiRe05] sessions, operating in conjunction with Pre-Shared Key (PSK) [ErTs05] ciphersuites defined for use with TLS.

1.2 Background

The TLS protocol is widely deployed and used to provide secure sessions, not only for web browsing but also for many other purposes. It supports a broad range of cryptographic methods, through definition and use of different ciphersuites. Today, the majority of TLS deployments authenticate servers using certificate-based public-key techniques. While certificate-based authentication of clients is also supported within the protocol, most deployments authenticate clients by passing other data (e.g., passwords) to servers across the protected channel that TLS establishes. A recent specification [ErTs05] defines new ciphersuites, where clients and servers are authenticated to each other based on common possession of a shared secret. The current document leverages these ciphersuites by using shared secrets that are based on OTPs. As such, the OTP becomes the basis for authentication of a TLS session.

1.3 Document organization

The organization of this document is as follows:

- Section 0~~1~~ is an introduction.
- Section 2 defines acronyms and notation used in this document.
- Section 3 defines methods for use of OTPs within TLS.
- Section 4 discusses security considerations.
- Appendix A provides example messages.
- Appendices B, C, and D cover intellectual property considerations, give references to other publications and standards, and provide general information about the One-Time Password Specifications.

2 Acronyms and notation

2.1 Acronyms

PSK	Pre-Shared Key
TLS	Transport Layer Security

2.2 Notation

TLS presentation language declarations are made in the `Courier` typeface. Function parameter names and structure components are written in *italic*.

3 OTP TLS elements and protocola~~Approaches~~

3.1 PSK e~~E~~stablishment

3.1.1 Introduction

We define two classes of approaches for establishing a PSK based on an OTP value:

- The first choice applies the OTP directly as the PSK. Given that the entropy of the value space produced by many OTP methods is insufficient to preclude exhaustive search by an attacker, this class is recommended only for use with PSK ciphersuites that incorporate additional random elements in PSK construction; currently-defined ciphersuites with this characteristic employ public-key methods.
- The second choice applies "hardening" techniques (i.e. techniques that do not make it computationally impossible, but economically unattractive for an attacker to search for a given key) to the OTP in order to derive a resulting PSK that is more resistant to search by an attacker. A PSK of this form is suitable for use

with arbitrary PSK ciphersuites, even those based purely on symmetric-key operations.

When the OTP method requires a server-issued challenge, we rely on the TLS extension feature of [Blak03] to request and provide such challenges. We also rely on the TLS extension feature of [Blak03] to negotiate OTP hardening parameters.

It is important to recognize that some underlying OTP methods employ and transfer user-entered PINs in conjunction with OTP values. For purposes of this discussion, we ~~[ISSUE: We are evaluating possible alternate approaches for use of PINs in conjunction with TLS-PSK, and solicit OTPS community discussion on this topic. (We~~ are concerned ~~here~~ with PINs that users provide for distributed processing purposes, not those consumed locally to unlock a token device.~~)~~ Two basic alternatives exist:

- Apply a function to combine the PIN with the OTP value, and use the result as input to PSK derivation. With this approach, successful establishment of a TLS-PSK session implies that both peers have been independently able to provide matching OTP and PIN values. As such, it offers two-factor mutual authentication. This approach increases the input entropy to PSK derivation, but may be inconsistent with operational models where users' PINs are maintained independently from secrets associated with their token devices.
- Omit the PIN from PSK derivation, instead retaining it to be separately transferred for validation in a subsequent step outside the scope of TLS, though the newly-established TLS channel may be used as a means to transfer it securely. This approach decouples the PIN from TLS processing, which may be appropriate for environments where PINs and token device secrets are managed separately (e.g., if PINs are to be accepted and processed by applications above the TLS layer).

This specification allows either alternative to be supported, but leaves the decision as a matter to be profiled for particular OTP methods. For a method that combines PINs with OTPs, the method profile must also define the specific combination function used to yield a single element as input to TLS-PSK operation. If appropriate to support different scenarios and requirements, more than one method profile may be defined for a given underlying OTP algorithm and technology.~~ We envision possible uses for both of these methods; if both are to be supported, this implies either that clients must know (implicitly and/or via configuration) which mode of processing to perform, or that a signaling path between client and server must be defined in order to reach common agreement about whether or not a PIN is to be incorporated in the derivation of a PSK. Both methods also assume support by the underlying authentication method.]~~

~~3.1.2[ISSUE: It may be appropriate to define new TLS error alerts to indicate conditions specific to OTP methods. For compatibility reasons, any such alerts should only be presented to a client issuing a corresponding extension, and thereby implying an ability to process such alerts. Possible definition of OTP-specific alerts is a matter for future study.]In both cases, when the OTP method requires a server-issued challenge, we rely on the TLS extension feature of [Blak03] to request and provide such challenges. In the~~

~~latter approach, we also rely on the TLS extension feature of [Blak03] to negotiate OTP hardening parameters.~~

3.1.2 Direct u~~U~~se with e~~E~~ntropy-e~~E~~nhancing PSK c~~C~~iphersuites

For this case, the OTP value is used directly as a PSK for a TLS-PSK ciphersuite. No extension to the TLS handshake is required except in the case of OTP methods that must transfer and process a server-provided challenge value before an OTP value can be generated. For this, the OTP Challenge Data extension defined in Section 3.3.2 may be used.

This approach can be applied in conjunction with the DHE_PSK or RSA_PSK key exchange algorithms defined in [ErTs05], although use of the DHE_PSK key exchange algorithm without OTP hardening will expose clients to man-in-the-middle (MITM) attacks (see further Section 4). For this reason, the alternative of directly using an OTP as a PSK is not recommended for the DHE_PSK key exchange algorithm, unless MITM attacks are prevented within the operational environment by separate server authentication methods or other means.

3.1.3 Deriving a PSK through OTP hardening

In this approach, a PSK is derived from a given OTP in such a way that an attack based on searching for the OTP becomes economically unattractive. This approach can be applied in conjunction with any of the key exchange algorithms defined in [ErTs05]. The key derivation mechanism recommended in this document builds on the PBKDF2 key derivation function defined in PKCS #5 v2.0 [RSA99].

3.2 Structure of the PSK_Identity e~~E~~lement

For purposes of this specification, the `PSK_Identity` field of the `ClientKeyExchange` message defined in [ErTs05] may carry ~~reflect~~ either a user's name or the identity of the key associated with the user's token device. At least one of user and/or key identifier is required. We adopt the method of [Wahl97], Section 2.4 to represent these values and other data related to OTP-based authentication textually, defining the following prefixes:

- ~~with~~ P~~p~~refix "UI=" signifies~~ying~~ a user identifier (e.g., username).
- Prefix ~~and~~ "KI=" signifies~~ying~~ a key identifier.
- An "OM=" prefix is used to indicate that an OTP method is being used in conjunction with TLS-PSK. This prefix may be followed by a textual identifier of the method[1], or an "OM=" element with an empty value can be used to signify that some OTP method is being used in conjunction with TLS-PSK but that the method must be identified through other means. ~~Optionally, a client may also add information pertaining to the OTP method to this element.~~

[1] This means there may be a need for a registry of OTP method identifiers.

- To provide the time associated with the OTP used in the PSK computations, the client may~~shall~~ use the prefix "T=" and, if used, shall encode the time value as YYYYMMDDhhmmss, where YYYY denotes the year, MM the month, DD the day, and hh, mm, and ss, the hour, minutes and seconds, respectively. The time shall always be provided as UTC.
- To provide a counter value associated with the OTP used in the PSK computations, the client may~~shall~~ use the prefix "C=" and, if used, shall provide the counter value as a UTF-8 string of decimal digits.

In this notation, assuming a user with the user identifier "J. Random User" and a time-based OTP calculated based on the time 11:42:04 (UTC) December 22, 2005, but without an explicit OTP method identifier, the `PSK_Identity` value would become:

```
psk_identity = "UI=J. Random User, T=20051222114204,O
        ~~O~~M=";
```

OTP-based authentication of a user with OTP key identifier 142857 and the explicitly-identified "OTP-Counter" method with counter value 285714 would be represented with the following `PSK_Identity` value:

```
psk_identity = "KI=142857, C=285714, OM=OTP-Counter";
```

The order of elements within a `PSK_Identity` field is not significant.

3.3 TLS e~~E~~xtension d~~D~~efinitions

3.3.1 Extension t~~T~~ypes

The extensions defined in this document are:

```
challenge_data(X) /* To be defined */
otp_hardening(Y) /* To be defined */
```

3.3.2 OTP c~~C~~hallenge d~~D~~ata

This extension may be used when a TLS client wants to make use of an OTP as a PSK (whether hardened or not), and the OTP algorithm requires a challenge as input. In order to request a challenge from the TLS server, the client may include an extension of type `challenge_data` in the (extended) Client Hello message. The `extension_data` field of this extension shall contain a `ChallengeData` where:

```
struct {
   ChallengeDataType challenge_data_type;
   select (ChallengeDataType) {
        case request: ChallengeRequestData;
        case response: ChallengeResponseData;
   } challenge_data;
} ChallengeData;

enum {
   request(0), response(1), (255)
} ChallengeDataType;
```

```
struct {
   opaque otp_algorithm<0..2^16-1>;
   opaque otp_user_id<0..2^16-1>;
   opaque otp_key_id<0..2^16-1>;
} ChallengeRequestData;

struct {
   opaque otp_challenge<1..2^16-1>;
} ChallengeResponseData;
```

In a Client Hello message, the `request` alternative of `ChallengeData` shall be used and shall provide information that the server may need in order to generate a challenge:

- The `otp_algorithm` field of the `ChallengeRequestData` structure shall, when non-empty, contain a URL identifying the OTP algorithm that needs the challenge.
- The `otp_user_id` field of the `ChallengeRequestData` structure shall, when non-empty, contain a user identifier (e.g., username) that enables the server to generate a correct challenge. Note that use of this field may disclose the user identifier to eavesdroppers.
- The `otp_key_id` field of the `ChallengeRequestData` structure shall, when non-empty, contain an identifier of the OTP generation key with which a requested challenge is to be used. As for the `otp_user_id` field, this field may be subject to eavesdropping.

At least one of the `otp_algorithm`, `otp_user_id`, or `otp_key_id` fields must be non-empty in a `ChallengeRequestData` value; when feasible, both `otp_algorithm` and at least one of `otp_user_id` and `otp_key_id` should be provided. The `otp_key_id` field shall refer to a key for the given user when both the `otp_key_id` and the `otp_user_id` fields are non-empty. If the `otp_user_id` or the `otp_key_id` (or both) alternatives of `ChallengeRequestData` are non-empty, then a subsequently sent `psk_identity` value must match these values.

Servers that receive a Client Hello containing the `challenge_data` extension may use the information contained in the extension to generate an appropriate challenge to return to the client. In this event, the server shall include an extension of type `challenge_data` in the (extended) Server Hello. The `extension_data` field of this extension shall contain a `ChallengeData` value and the `response` alternative of the `ChallengeData` shall be used to provide the challenge to the client:

- The `otp_challenge` field of the `ChallengeResponseData` structure shall contain the requested challenge.

Servers that receive a Client Hello containing the `challenge_data` extension but are unable to generate an appropriate challenge based on the information provided by the client shall abort the handshake with an `illegal_parameter` alert.

3.3.3 OTP h~~H~~ardening e~~E~~xtension

This extension may be used when a TLS client wants to make use of an OTP as a PSK and the OTP needs to be hardened before being used as a PSK.

OTP hardening is achieved by applying the PBKDF2 from [RSA99] on the OTP, using a negotiated iteration count.

A TLS client may include an extension of type `otp_hardening` in the (extended) Client Hello message in order to establish hardening parameters with the TLS server. The `extension_data` field of this extension shall contain an `OTPHardeningData` where:

```
struct {
   uint16 iteration_count;
} OTPHardeningData;
```

The client ~~suggests values~~indicates its highest supported value for the iteration count *c* in PBKDF2 through the `iteration_count` field of the `OTPHardeningData`.

Servers that receive a Client Hello containing the `otp_hardening` extension may use the information contained in the extension to provide a selected iteration count in return to the client. In this event, the server shall include an extension of type `otp_hardening` in the (extended) Server Hello. The `extension_data` field of this extension shall contain an `OTPHardeningData` value with the selected iteration count.

Clients that receive a Server Hello containing the `otp_hardening` extension must do a sanity and policy check on the provided iteration_count value. If the check passes, the client shall compute a PSK using PBKDF2 from [RSA99] as follows ("||" denotes string concatenation):

$\text{PSK} = \text{PBKDF2}\ (OTP, R_S \,||\, R_C, iterationCount, keyLen)$

Where:

- OTP is the current one-time password,
- R_S and R_C are the `server_random` value from the Server Hello message and the `client_random` value from the Client Hello message, respectively,
- $iterationCount$ is the `iteration_count` value from the `otp_hardening` extension in the Server Hello message, and
- $keyLen$ shall be set to 16 (128 bits).

The PBKDF2 PRF algorithm shall be the TLS PRF, using the US-ASCII string "OTP hardening" as the label.

Servers that receive a Client Hello containing the `otp_hardening` extension but are unwilling or unable to engage in an OTP hardening operation shall ignore the extension. In this case, TLS client (and TLS server) policy will determine whether the handshake will continue or not.

3.4 Error alerts

This section defines new error alerts for use with the OTP TLS method defined in this document. For compatibility reasons, these alerts must not be sent unless the sending party has received, from the party they are communicating with, an extended hello message or a key exchange message indicating use of OTPs in TLS-PSK as defined by this document.

- "unsupported_otp_algorithm" – this alert is sent by servers that receive a key exchange message (or a "challenge_data" extension) which indicates use of an OTP algorithm that is not supported by the server. This message is always fatal.
- "otp_key_lost" – this alert is sent by servers that receive a key exchange message (or a "challenge_data" extension) which indicates use of an OTP key that has been reported as lost. This message may or may not be fatal depending on server policy~~ This message is always fatal. This message is always fatal~~
- "otp_key_expired" – this alert is sent by servers that receive a key exchange message (or a "challenge_data" extension) which indicates use of an OTP key that has expired. This message may or may not be fatal depending on server policy.
- "otp_key_blocked" – this alert is sent by servers that receive a key exchange message (or a "challenge_data" extension) which indicates use of an OTP key that has been blocked. This message is always fatal.
- "otp_key_unknown" – this alert is sent by servers that receive a key exchange message (or a "challenge_data" extension) which indicates use of an OTP key unknown to the server. This message is always fatal.
- "pin_change_required" – this alert is sent by servers that also maintain user PINs associated with OTP keys when they receive a key exchange message (or a "challenge_data" extension) based on a key for which the user PIN needs to be replaced. This message may or may not be fatal depending on server policy.

These new error alerts are conveyed using the following syntax:

```
enum {
   /* Only listing error alerts defined in */
   /* this document here */
   unsupported_otp_algorithm(TBD),
   otp_key_lost(TBD),
   otp_key_expired(TBD),
   otp_key_blocked(TBD),
   otp_key_unknown(TBD),
   pin_change_required(TBD)
} AlertDescription;
```

4 Security considerations

This document is a profile of [ErTs05] and all security considerations discussed in [ErTs05] apply. In particular, since OTPs usually are relatively short, the risk of PSK compromise due to brute-force searching applies when an OTP is used directly as the PSK and the PSK key exchange algorithm or (in the case of an MITM attack) the DHE_PSK key exchange algorithm is used. This document therefore recommends OTP hardening whenever the PSK key exchange algorithm is suggested by the client, or when there is a risk for a MITM attack and the DHE_PSK key exchange algorithm is suggested by the client.

When, per Section 3.1.3 of this document and Section 2 of [ErTs05], a symmetric PSK is established through hardening of an OTP value, an attacker obtaining that PSK would become able to decrypt data from its protected TLS session. If the encrypted session was recorded, its underlying plaintext could be revealed once the PSK was obtained. It is important, therefore, that the level of hardening applied to protect the PSK against searching attacks on the OTP space be consistent with the data's value, its useful lifetime, and the anticipated level of computational resources to be applied against the PSK. If a suitable hardening level cannot be achieved, use of the entropy-enhancing ciphersuites as discussed in Section 3.1.2 of this document and Sections 3 and 4 of [ErTs05] is recommended. It should be noted, however, that the use of OTPs provides perfect forward secrecy: Even if a particular OTP is compromised, an attacker will not be able to apply its value to decrypt any other conversation than the one where the OTP was used as a basis for a PSK.

A. Example messages

A.1 Example syntax

The syntax of the examples in this appendix loosely follows the presentation language syntax defined in [DiRe05].

A.2 Client Hello

In this example, the client suggests use of either the `TLS_DHE_PSK_WITH_AES_128_CBC_SHA` or the `TLS_RSA_PSK_WITH_AES_128_CBC_SHA` ciphersuite, both defined in [ErTs05]. The client also asks the server for a suitable iteration count for the OTP hardening, and ~~suggests~~ indicates it cannot support an~~the~~ iteration count ~~to be~~above 2~~1~~0,000.

```
      message_type = client_hello;
      length = ...;
      body = {
           client_version = 3.1;
           random = {
               gmt_unix_time = 1135069786;
               random_bytes = 0xa243edfeed12adef...;
               };
           session_id = ;     /* Empty */
```

```
            cipher_suites = {{0x00, 0x8F},{0x00,0x94}};
            compression_methods = {null};
            client_hello_extension_list = {
                {
                     extension_type = otp_hardening;
                     extension_data = {
                          iteration_count = 2~~1~~0000;
                          }
                     }
                }
            };
```

A.3 Server Hello

Continuing with the previous example, the server responds positively to the client hello message, selects the Diffie-Hellman PSK cipher suite, and sets the iteration count to 20,000:

```
       message_type = server_hello;
       length = ...;
       body = {
            server_version = 3.1;
            random = {
                gmt_unix_time = 1135069787;
                random_bytes = 0x2143287432987321dbc321...;
                };
            session_id = 0x4321defeadcbbdbe213...;
            cipher_suite = {0x00, 0x8F};
            compression_method = null;
            server_hello_extension_list = {
                {
                     extension_type = otp_hardening;
                     extension_data = {
                          iteration_count = 20000;
                          }
                     }
                }
            };
```

B. Intellectual property considerations

RSA Security makes no patent claims on the general constructions described in this document, although specific underlying techniques may be covered. RSA Security has US patents (and foreign counterparts) on certain underlying techniques, in particular US Patent Nos. 4,885,778; 4,856,062; 5,097,505; 5,168,520; 5,367,572 and 5,657,388. Additional patents are pending. As this specification can be implemented without the use of these underlying techniques, it is RSA Security's position that the technology covered by these patents and applications is not required to implement this specification.

C. References

[Blak03] S. Blake-Wilson, M. Nyström, D. Hopwood, and J. Mikkelsen, "Transport Layer Security (TLS) Extensions." Internet RFC-3546, June 2003. URL: http://ietf.org/rfc/rfc3546.txt.

[DiRe05] T. Dierks and E. Rescorla, "The TLS Protocol Version 1.1", work in progress, IETF TLS Working Group, June 2005. (Note: As of ~~December 2005~~March 2006, this document had been approved for publication as a Proposed Standard and was awaiting RFC Editor processing.)

[ErTs05] P. Eronen and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", Internet RFC-4279, December 2005. URL: http://ietf.org/rfc/rfc4279.txt.

[RSA99] RSA Laboratories. "PKCS #5 v2.0: Password-Based Cryptography Standard," April, 1999. URL: http://www.rsasecurity.com/rsalabs/pkcs/.

[Wahl97] M. Wahl, S. Kille, and T. Howes, "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", Internet RFC-2253, December 1997. URL: http://ietf.org/rfc/rfc2253.txt.

D. About OTPS

The *One-Time Password Specifications* are documents produced by RSA Laboratories in cooperation with secure systems developers for the purpose of simplifying integration and management of strong authentication technology into secure applications, and to enhance the user experience of this technology.

RSA Laboratories plans further development of the OTPS series through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. As for our PKCS documents, results may also be submitted to standards forums. For more information, contact:

OTPS Editor
RSA Laboratories
174 Middlesex Turnpike
Bedford, MA 01730 USA
otps-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/