Using the Secure Remote Password (SRP) Protocol for TLS Authentication

Status of This Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Abstract

This memo presents a technique for using the Secure Remote Password protocol as an authentication method for the Transport Layer Security protocol.
Table of Contents

1.  Introduction ................................................. 3
2.  SRP Authentication in TLS .................................... 3
   2.1.  Notation and Terminology .............................. 3
   2.2.  Handshake Protocol Overview ......................... 4
   2.3.  Text Preparation ...................................... 5
   2.4.  SRP Verifier Creation ................................ 5
   2.5.  Changes to the Handshake Message Contents .......... 5
      2.5.1.  Client Hello .................................... 6
      2.5.2.  Server Certificate ............................... 7
      2.5.3.  Server Key Exchange ............................ 7
      2.5.4.  Client Key Exchange ............................ 8
   2.6.  Calculating the Premaster Secret .................... 8
   2.7.  Ciphersuite Definitions .............................. 9
   2.8.  New Message Structures .............................. 9
      2.8.1.  Client Hello .................................... 10
      2.8.2.  Server Key Exchange ............................ 10
      2.8.3.  Client Key Exchange ............................ 11
   2.9.  Error Alerts .......................................... 11
3.  Security Considerations ..................................... 12
   3.1.  General Considerations for Implementors ............. 12
   3.2.  Accepting Group Parameters .......................... 12
   3.3.  Protocol Characteristics ............................. 12
   3.4.  Hash Function Considerations ....................... 13
4.  IANA Considerations ........................................ 13
5.  References ................................................ 14
   5.1.  Normative References ................................ 14
   5.2.  Informative References .............................. 15
Appendix A.  SRP Group Parameters ............................. 16
Appendix B.  SRP Test Vectors .................................. 21
Appendix C.  Acknowledgements ................................ 22
1. Introduction

At the time of writing TLS [TLS] uses public key certificates, pre-shared keys, or Kerberos for authentication.

These authentication methods do not seem well suited to certain applications now being adapted to use TLS ([IMAP], for example). Given that many protocols are designed to use the user name and password method of authentication, being able to safely use user names and passwords provides an easier route to additional security.

SRP ([SRP], [SRP-6]) is an authentication method that allows the use of user names and passwords over unencrypted channels without revealing the password to an eavesdropper. SRP also supplies a shared secret at the end of the authentication sequence that can be used to generate encryption keys.

This document describes the use of the SRP authentication method for TLS.

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 RFC 2119 [REQ].

2. SRP Authentication in TLS

2.1. Notation and Terminology

The version of SRP used here is sometimes referred to as "SRP-6" [SRP-6]. This version is a slight improvement over "SRP-3", which was described in [SRP] and [SRP-RFC]. For convenience, this document and [SRP-RFC] include the details necessary to implement SRP-6; [SRP-6] is cited for informative purposes only.
This document uses the variable names defined in [SRP-6]:

N, g: group parameters (prime and generator)

s: salt

B, b: server’s public and private values

A, a: client’s public and private values

I: user name (aka "identity")

P: password

v: verifier

k: SRP-6 multiplier

The | symbol indicates string concatenation, the ^ operator is the exponentiation operation, and the % operator is the integer remainder operation.

Conversion between integers and byte-strings assumes the most significant bytes are stored first, as per [TLS] and [SRP-RFC]. In the following text, if a conversion from integer to byte-string is implicit, the most significant byte in the resultant byte-string MUST be non-zero. If a conversion is explicitly specified with the operator PAD(), the integer will first be implicitly converted, then the resultant byte-string will be left-padded with zeros (if necessary) until its length equals the implicitly-converted length of N.

2.2. Handshake Protocol Overview

The advent of [SRP-6] allows the SRP protocol to be implemented using the standard sequence of handshake messages defined in [TLS].

The parameters to various messages are given in the following diagram.
**Text Preparation**

The user name and password strings SHALL be UTF-8 encoded Unicode, prepared using the [SASLPREP] profile of [STRINGPREP].

**SRP Verifier Creation**

The verifier is calculated as described in Section 3 of [SRP-RFC]. We give the algorithm here for convenience.

The verifier (v) is computed based on the salt (s), user name (I), password (P), and group parameters (N, g). The computation uses the [SHA1] hash algorithm:

\[
    x = \text{SHA1}(s \mid \text{SHA1}(I \mid ":" \mid P))
\]

\[
    v = g^x \mod N
\]

**Changes to the Handshake Message Contents**

This section describes the changes to the TLS handshake message contents when SRP is being used for authentication. The definitions of the new message contents and the on-the-wire changes are given in Section 2.8.
2.5.1. Client Hello

The user name is appended to the standard client hello message using the extension mechanism defined in [TLSEXT] (see Section 2.8.1). This user name extension is henceforth called the "SRP extension". The following subsections give details of its use.

2.5.1.1. Session Resumption

When a client attempts to resume a session that uses SRP authentication, the client MUST include the SRP extension in the client hello message, in case the server cannot or will not allow session resumption, meaning a full handshake is required.

If the server does agree to resume an existing session, the server MUST ignore the information in the SRP extension of the client hello message, except for its inclusion in the finished message hashes. This is to ensure that attackers cannot replace the authenticated identity without supplying the proper authentication information.

2.5.1.2. Missing SRP Extension

The client may offer SRP cipher suites in the hello message but omit the SRP extension. If the server would like to select an SRP cipher suite in this case, the server SHOULD return a fatal "unknown_psk_identity" alert (see Section 2.9) immediately after processing the client hello message.

A client receiving this alert MAY choose to reconnect and resend the hello message, this time with the SRP extension. This allows the client to advertise that it supports SRP, but not have to prompt the user for his user name and password, nor expose the user name in the clear, unless necessary.

2.5.1.3. Unknown SRP User Name

If the server doesn’t have a verifier for the user name in the SRP extension, the server MAY abort the handshake with an "unknown_psk_identity" alert (see Section 2.9). Alternatively, if the server wishes to hide the fact that this user name doesn’t have a verifier, the server MAY simulate the protocol as if a verifier existed, but then reject the client’s finished message with a "bad_record_mac" alert, as if the password was incorrect.

To simulate the existence of an entry for each user name, the server must consistently return the same salt (s) and group (N, g) values for the same user name. For example, the server could store a secret "seed key" and then use HMAC-SHA1(seed_key, "salt" | user_name) to
generate the salts [HMAC]. For B, the server can return a random value between 1 and N-1 inclusive. However, the server should take care to simulate computation delays. One way to do this is to generate a fake verifier using the "seed key" approach, and then proceed with the protocol as usual.

2.5.2. Server Certificate

The server MUST send a certificate if it agrees to an SRP cipher suite that requires the server to provide additional authentication in the form of a digital signature. See Section 2.7 for details of which cipher suites defined in this document require a server certificate to be sent.

2.5.3. Server Key Exchange

The server key exchange message contains the prime (N), the generator (g), and the salt value (s) read from the SRP password file based on the user name (I) received in the client hello extension.

The server key exchange message also contains the server’s public value (B). The server calculates this value as $B = k*v + g^b \mod N$, where $b$ is a random number that SHOULD be at least 256 bits in length and $k = SHA1(N \mid PAD(g))$.

If the server has sent a certificate message, the server key exchange message MUST be signed.

The group parameters (N, g) sent in this message MUST have N as a safe prime (a prime of the form $N=2q+1$, where q is also prime). The integers from 1 to N-1 will form a group under multiplication $\mod N$, and g MUST be a generator of this group. In addition, the group parameters MUST NOT be specially chosen to allow efficient computation of discrete logarithms.

The SRP group parameters in Appendix A satisfy the above requirements, so the client SHOULD accept any parameters from this appendix that have large enough N values to meet her security requirements.

The client MAY accept other group parameters from the server, if the client has reason to believe that these parameters satisfy the above requirements, and the parameters have large enough N values. For example, if the parameters transmitted by the server match parameters on a "known-good" list, the client may choose to accept them. See Section 3 for additional security considerations relevant to the acceptance of the group parameters.
Group parameters that are not accepted via one of the above methods MUST be rejected with an "insufficient_security" alert (see Section 2.9).

The client MUST abort the handshake with an "illegal_parameter" alert if \( B \% N = 0 \).

### 2.5.4. Client Key Exchange

The client key exchange message carries the client’s public value (A). The client calculates this value as \( A = g^a \% N \), where a is a random number that SHOULD be at least 256 bits in length.

The server MUST abort the handshake with an "illegal_parameter" alert if \( A \% N = 0 \).

### 2.6. Calculating the Premaster Secret

The premaster secret is calculated by the client as follows:

\[
I, P = \text{<read from user>} \\
N, g, s, B = \text{<read from server>} \\
a = \text{random()} \\
A = g^a \% N \\
u = \text{SHA1(PAD(A) | PAD(B))} \\
k = \text{SHA1(N | PAD(g))} \\
x = \text{SHA1(s | SHA1(I | "." | P))} \\
\text{<premaster secret> = (B - (k \cdot g^x)) ^ (a + (u \cdot x)) \% N}
\]

The premaster secret is calculated by the server as follows:

\[
N, g, s, v = \text{<read from password file>} \\
b = \text{random()} \\
k = \text{SHA1(N | PAD(g))} \\
B = k^v + g^b \% N \\
A = \text{<read from client>} \\
u = \text{SHA1(PAD(A) | PAD(B))} \\
\text{<premaster secret> = (A \cdot v^u) ^ b \% N}
\]

The finished messages perform the same function as the client and server evidence messages (M1 and M2) specified in [SRP-RFC]. If either the client or the server calculates an incorrect premaster secret, the finished messages will fail to decrypt properly, and the other party will return a "bad_record_mac" alert.

If a client application receives a "bad_record_mac" alert when performing an SRP handshake, it should inform the user that the entered user name and password are incorrect.
2.7. Ciphersuite Definitions

The following cipher suites are added by this document. The usage of Advanced Encryption Standard (AES) cipher suites is as defined in [AESCIPH].

CipherSuite TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA = { 0xC0,0x1A };
CipherSuite TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA = { 0xC0,0x1B };
CipherSuite TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA = { 0xC0,0x1C };
CipherSuite TLS_SRP_SHA_WITH_AES_128_CBC_SHA = { 0xC0,0x1D };
CipherSuite TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA = { 0xC0,0x1E };
CipherSuite TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA = { 0xC0,0x1F };
CipherSuite TLS_SRP_SHA_WITH_AES_256_CBC_SHA = { 0xC0,0x20 };
CipherSuite TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA = { 0xC0,0x21 };
CipherSuite TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA = { 0xC0,0x22 };

Cipher suites that begin with TLS_SRP_SHA_RSA or TLS_SRP_SHA_DSS require the server to send a certificate message containing a certificate with the specified type of public key, and to sign the server key exchange message using a matching private key.

Cipher suites that do not include a digital signature algorithm identifier assume that the server is authenticated by its possession of the SRP verifier.

Implementations conforming to this specification MUST implement the TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA cipher suite, SHOULD implement the TLS_SRP_SHA_WITH_AES_128_CBC_SHA and TLS_SRP_SHA_WITH_AES_256_CBC_SHA cipher suites, and MAY implement the remaining cipher suites.

2.8. New Message Structures

This section shows the structure of the messages passed during a handshake that uses SRP for authentication. The representation language used is the same as that used in [TLS].
2.8.1. Client Hello

A new extension "srp", with value 12, has been added to the enumerated ExtensionType defined in [TLSEXT]. This value MUST be used as the extension number for the SRP extension.

The "extension_data" field of the SRP extension SHALL contain:

    opaque srp_I<1..2^8-1>;

where srp_I is the user name, encoded per Section 2.3.

2.8.2. Server Key Exchange

A new value, "srp", has been added to the enumerated KeyExchangeAlgorithm originally defined in [TLS].

When the value of KeyExchangeAlgorithm is set to "srp", the server’s SRP parameters are sent in the server key exchange message, encoded in a ServerSRPParams structure.

If a certificate is sent to the client, the server key exchange message must be signed.

    enum { rsa, diffie_hellman, srp } KeyExchangeAlgorithm;

    struct {
        select (KeyExchangeAlgorithm) {
            case diffie_hellman:
                ServerDHParams params;
                Signature signed_params;
            case rsa:
                ServerRSAParams params;
                Signature signed_params;
            case srp: /* new entry */
                ServerSRPParams params;
                Signature signed_params;
        }
    } ServerKeyExchange;

    struct {
        opaque srp_N<1..2^16-1>;
        opaque srp_g<1..2^16-1>;
        opaque srp_s<1..2^8-1>;
        opaque srp_B<1..2^16-1>;
    } ServerSRPParams;   /* SRP parameters */
2.8.3. Client Key Exchange

When the value of KeyExchangeAlgorithm is set to "srp", the client’s public value (A) is sent in the client key exchange message, encoded in a ClientSRPPublic structure.

```
struct {
    select (KeyExchangeAlgorithm) {
        case rsa: EncryptedPreMasterSecret;
        case diffie_hellman: ClientDiffieHellmanPublic;
        case srp: ClientSRPPublic; /* new entry */
    } exchange_keys;
} ClientKeyExchange;
```

struct {
    opaque srp_A<1..2^16-1>;
} ClientSRPPublic;

2.9. Error Alerts

This document introduces four new uses of alerts:

- "unknown_psk_identity" (115) - this alert MAY be sent by a server that would like to select an offered SRP cipher suite, if the SRP extension is absent from the client’s hello message. This alert is always fatal. See Section 2.5.1.2 for details.

- "unknown_psk_identity" (115) - this alert MAY be sent by a server that receives an unknown user name. This alert is always fatal. See Section 2.5.1.3 for details.

- "insufficient_security" (71) - this alert MUST be sent by a client that receives unknown or untrusted (N, g) values. This alert is always fatal. See Section 2.5.3 for details.

- "illegal_parameter" (47) - this alert MUST be sent by a client or server that receives a key exchange message with A % N = 0 or B % N = 0. This alert is always fatal. See Section 2.5.3 and Section 2.5.4 and for details.

The "insufficient_security" and "illegal_parameter" alerts are defined in [TLS]. The "unknown_psk_identity" alert is defined in [PSK].

The "unknown_psk_identity" and "illegal_parameter" alerts are defined in [TLS]. The "unknown_psk_identity" alert is defined in [PSK].
3. Security Considerations

3.1. General Considerations for Implementors

The checks described in Section 2.5.3 and Section 2.5.4 on the received values for A and B are CRUCIAL for security and MUST be performed.

The private values a and b SHOULD be at least 256-bit random numbers, to give approximately 128 bits of security against certain methods of calculating discrete logarithms. See [TLS], Section D.1, for advice on choosing cryptographically secure random numbers.

3.2. Accepting Group Parameters

An attacker who could calculate discrete logarithms % N could compromise user passwords, and could also compromise the confidentiality and integrity of TLS sessions. Clients MUST ensure that the received parameter N is large enough to make calculating discrete logarithms computationally infeasible.

An attacker may try to send a prime value N that is large enough to be secure, but that has a special form for which the attacker can more easily compute discrete logarithms (e.g., using the algorithm discussed in [TRAPDOOR]). If the client executes the protocol using such a prime, the client’s password could be compromised. Because of the difficulty of checking for such primes in real time, clients SHOULD only accept group parameters that come from a trusted source, such as those listed in Appendix A, or parameters configured locally by a trusted administrator.

3.3. Protocol Characteristics

If an attacker learns a user’s SRP verifier (e.g., by gaining access to a server’s password file), the attacker can masquerade as the real server to that user, and can also attempt a dictionary attack to recover that user’s password.

An attacker could repeatedly contact an SRP server and try to guess a legitimate user’s password. Servers SHOULD take steps to prevent this, such as limiting the rate of authentication attempts from a particular IP address or against a particular user name.

The client’s user name is sent in the clear in the Client Hello message. To avoid sending the user name in the clear, the client could first open a conventional anonymous or server-authenticated connection, then renegotiate an SRP-authenticated connection with the handshake protected by the first connection.
If the client receives an "unknown_pskIdentity" alert in response to a client hello, this alert may have been inserted by an attacker. The client should be careful about making any decisions, or forming any conclusions, based on receiving this alert.

It is possible to choose a (user name, password) pair such that the resulting verifier will also match other, related, (user name, password) pairs. Thus, anyone using verifiers should be careful not to assume that only a single (user name, password) pair matches the verifier.

3.4. Hash Function Considerations

This protocol uses SHA-1 to derive several values:

- $u$ prevents an attacker who learns a user’s verifier from being able to authenticate as that user (see [SRP-6]).

- $k$ prevents an attacker who can select group parameters from being able to launch a 2-for-1 guessing attack (see [SRP-6]).

- $x$ contains the user’s password mixed with a salt.

Cryptanalytic attacks against SHA-1 that only affect its collision-resistance do not compromise these uses. If attacks against SHA-1 are discovered that do compromise these uses, new cipher suites should be specified to use a different hash algorithm.

In this situation, clients could send a Client Hello message containing new and/or old SRP cipher suites along with a single SRP extension. The server could then select the appropriate cipher suite based on the type of verifier it has stored for this user.

4. IANA Considerations

This document defines a new TLS extension "srp" (value 12), whose value has been assigned from the TLS ExtensionType Registry defined in [TLSExt].

This document defines nine new cipher suites, whose values have been assigned from the TLS Ciphersuite registry defined in [TLS].

CipherSuite TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA = { 0xC0,0x1A };
CipherSuite TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA = { 0xC0,0x1B };
CipherSuite TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA = { 0xC0,0x1C };

CipherSuite TLS_SRP_SHA_WITH_AES_128_CBC_SHA = { 0xC0,0x1D }
CipherSuite TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA = { 0xC0,0x1E }
CipherSuite TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA = { 0xC0,0x1F }
CipherSuite TLS_SRP_SHA_WITH_AES_256_CBC_SHA = { 0xC0,0x20 }
CipherSuite TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA = { 0xC0,0x21 }
CipherSuite TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA = { 0xC0,0x22 }

5. References

5.1. Normative References


5.2. Informative References


Appendix A. SRP Group Parameters

The 1024-, 1536-, and 2048-bit groups are taken from software developed by Tom Wu and Eugene Jhong for the Stanford SRP distribution, and subsequently proven to be prime. The larger primes are taken from [MODP], but generators have been calculated that are primitive roots of N, unlike the generators in [MODP].

The 1024-bit and 1536-bit groups MUST be supported.

1. 1024-bit Group

The hexadecimal value for the prime is:

   EEAF0AB9 ADB38DD6 9C33F80A FA8FC5E8 60726187 75FF3C0B 9EA2314C
   9C256576 D674DF74 96EA81D3 383B4813 D692C6E0 E0D5D8E2 50B98BE4
   8E495C1D 6089DAD1 5DC7D7B4 6154D6B6 CE8EF4AD 69B15D49 82559B29
   7BCF1885 C529F566 660E57EC 68EDBC3C 05726CC0 2FD4CBF4 976EAA9A
   FD5138FE 8376435B 9FC61D2F C0EB06E3

   The generator is: 2.

2. 1536-bit Group

The hexadecimal value for the prime is:

   9DEF3CAF B939277A B1F12A86 17A47BBB DBA51DF4 99AC4C80 BEEEA961
   4B19CC4D 5F4F5F55 6E27CBDE 51C6A94B E4607A29 1558903B A0D0F843
   80B655BB 9A22E8DC DF028A7C EC67F0D0 8134B1C8 B9798914 9B609E0B
   E3BAB63D 47548381 DBC5B1FC 764E3F4B 53DD9DA1 158BFD3E 2B9C8CF5
   6EDF0195 39349627 DB2FD53D 24B7C486 65772E43 7D6C7F8C E442734A
   F7CCB7AE 837C264A E3A9EBE8 7F8A2F9E B8B5292E 5A021FFF 5E91479E
   8CE7A28C 2442C6F3 15180F93 499A234D CF76E3FE D135F9BB

   The generator is: 2.
3. 2048-bit Group

The hexadecimal value for the prime is:

\[
\begin{align*}
AC6BDB41 & \text{ 324A99B } \text{ F166DE5E } 1389582F \text{ AF72B665 } 1987EE0F \text{ FC319294} \\
3DB56050 & \text{ A37329CB } \text{ B4A099ED } 8193E075 \text{ F166DE5E } 1389582F \text{ AF72B665} \\
CD7F48A9 & \text{ DA04FD50 } \text{ E8083969 } \text{ EDB767B0 } \text{ CF609517 } \text{ 9A163AB3 } 661A05F8
\end{align*}
\]

The generator is: 2.

4. 3072-bit Group

This prime is: 

\[
2^{3072} - 2^{3008} - 1 + 2^{64} \times \left\{ \left[ 2^{2942} \pi \right] + 1690314 \right\}
\]

Its hexadecimal value is:

\[
\begin{align*}
\text{FFFFFFFF} & \text{ FFFFFFFF} \text{ C90FDAA2 } 2168C234 \text{ C466628B} \text{ 80DC1CD1 } 29024E08 \\
8A677C74 & \text{ 02BEEA6} \text{ 3B13B22} \text{ 514A0879} \text{ 8E3404DD} \text{ EF9519B3} \text{ CD3A431B}
\end{align*}
\]

The generator is: 5.
5. 4096-bit Group

This prime is: $2^{4096} - 2^{4032} - 1 + 2^{64} \times \{2^{3966} \pi\} + 240904$

Its hexadecimal value is:

```
00000000 00000000 FF000000 00000000 C90FDAA2 C46626B B0DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386F82 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 83655D23 DCA3AD96 1C62F536 208552BB 9ED52907 7096966D
670C53AD 4ABC9804 F1746C08 CA1B217C 32905E46 2E36CE3B E39E772C
180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D
04507A33 A8521ABD DF1C6A46 ECFB8504 58DBEFOA 8AAE7157 5D060C7D
B3970F85 A6E1E4C7 ABF5AEB0 DB093DD7 1E8C94E0 4A25619D CEE3D226
1AD2E66B F12FFA06 D98A086D D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31 43DB5BFC
E0FD010E 4B82D120 A9210801 1A723C12 A787ED67 88719A10 BDBA5B26
99C32718 6AF4E23C 1A846384 B6150BDA 2583E9CA 2AD4CE8D DDBBC2DB
04DE8EF9 2E8EFC14 1FBECA6 287C5947 4E6BC05D 99B2964F A9093CA2
233BA186 515B7ED1 F612970 CEE2D7AF B81BDD76 2170481C D0069127
D5B05AA9 93B4EA98 8DBFDDC1 86FF87DC 90A6C08F 4DF435C9 34063199
```

The generator is: 5.
6. 6144-bit Group

This prime is: $2^{6144} - 2^{6080} - 1 + 2^{64} \cdot (\pi + 929484)$

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEAE 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 4E85BF76 6257EC6F 4A4C42E9
A637ED6B 0BBF5CB6 F4067BED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE45B3D C007CB8 A163BF05 98DA4836 1C55D39A 6916F3A8
FD24CF5F 8365DFD3 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B E39E772C
180E8603 9B2783AE EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22B61 98FA0510 15728E5A 8AAAC42D AD33170D
04507A33 A85521AB DF1CBA64 ECFB8504 58DBEFOA 8AAE7157 50D60C7D
B3970F85 A6E1E4C7 ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226
1AD2EE6B F12FFAO6 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E2FA0 74E5AB31 43DB5BF7
E0FD108E 4B82D120 A9210801 1A723C12 A7876E07 88719A10 BDBA5B62
99C32718 6AF4E23C 1A946834 B6150BDA 2583E9CA 2AD44CE8 DBBC2DB
04DE8EF9 2E8EFC14 1FBECA6A 287C5947 4E6B0C5D 99B2964F A909CC3A2
233BA186 515BE7ED 1F612970 CEE2D7AF B81BDD76 2170481C D0069127
D5B05AA9 93B4EA98 8BDFDCC1 86FFBBDC A9G6C08E 4DF435C9 34028492
36C3FAB4 D27C0726 C1D4D2B2 602646DE C9751E76 3DBA37BD F8FF9406
A9E530E E5DB382F 431001AE B06A53ED 9027D831 179727B0 856A8918
DA3EDBEB CF9B14ED 44CE5B0A CED4BB1B DB7F1447 E6CC254B 33205151
28D7AF42 6FB8F401 378CB2BF 59B3C0A1 C64822EC F032EA15 D1721DD3
F482D7CE 6E74FF66 D55E702F 46980C82 B5A84031 9081B8FE 59E7C97F
BEC7EBF3 23A97A7E 36C888BE 0FD4D57F FF585AC5 4B407B2 2B4154AA
CC8F6D7E BF48E1D8 14C55ED2 0F8037E0 A79715EE F29BE328 06A1D58B
B7C5DA76 F550A3D 8A1FF8F0 EB19CCB1 A313D55C DA56C9EC 2EF29632
387FE8D7 6E3C0468 043EBF6E 3F4860EE 12BF2D5B 0B747D6D E694F91E
60D4024 FFFFFFFF FFFFFFFF
```

The generator is: 5.
7. 8192-bit Group

This prime is: 2^8192 - 2^8128 - 1 + 2^64 \times [2^8062 \pi] + 4743158

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBAE6 3B139BB2 51A0A879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BF5CB5F 406BB7ED EE386BFB 5A899FA5 AE9F2411 7C4BF1E6
49286561 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 836552B7 DCA3AD96 1C62F536 20B552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CEB3 E97E772C
180E8E03 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD3170D
04507A33 A85251AB DF1CB5A4 ECFB8504 58DBEFOA 8AE66777 500D6C70
B3970F85 A6E1E4C7 ABF5AEB8 DB0933D7 E8C94E0 4A25619D CEE3D226
1AD2EE6B F12FFA06 D98A0864 D8760273 3EC86A4D 521F2B18 777B200C
B8E11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31 43DB5BFC
E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7 88719A10 BDDA5B26
99C32718 6AF4E23C 1A946834 B6150BDA 2583E9CA 2AD44CE8 DBBC2DB
04DEBEF9 2E8EFC14 1FBECAAE 287C5947 4E6BC05D 99B2964F A090C3A2
23BA186 515B8E76 1F612970 CEE2D7AF B81BDD76 2170481C D0069127
D5B05AA9 93B4EA98 8BDFFDD1 86FFBD7C 90A6C08F 4DF435C9 34028492
36C3FAB4 D27C7026 CD4DCB2 602646DE C9751E76 3DBA37BD F8B8F906
A9DE530E E5D8B3E2 F413001E B06A53ED 9027D831 1977287B 865A8918
DA33EDBE CF9B14ED 44ECE6BC C64B92EC E6C254B 33205151
28D7AF42 6FB8F401 378C82BF 5983CA01 C64B92EC F032EA15 D1721DD3
F482D7CE 6E74FEF6 D55E702F 46980C82 B5A84031 900B1C9E 597E979F
BEC7E8BF 32A997A7 36C88B8E 0F1D45B7 FF585AC5 4BD047B2 24B1AA54
CC8F6D7E BF4E81D8 14CC5ED2 0F8037E0 A79715EE F29BEB32 06A1D58B
B7C5DA76 F55A0A3D 8A1BFBBF EB19CCB1 A313D5C3 DA56C9EC 2EF29632
387F8E7D 6E3C0468 03EBF66 3F4860EE 12BF2D56 B8747D6E 694F991E
60BE1159 7A34296F 09FEE5E4 38777CB6 A932DF8C D8EEC00D 73B9531B
3BC83B26 BD9DD300 741F87AF 8AFC47ED 2576F693 6BA42466 3AA6B719
5AE4F568 3423B474 2BF1C978 23F816C3 E936D52D E3FD88BE FC8489AD
222220E0 A4037C07 13EB57A8 1A23FC07 3473FC64 66E3A05B 4BCB886
2F8385DD FA9D4B7F A2C0878E 796B3203 06D5B3A3 0623BC5F B3A278A9
6D2A13F8 3F4F82FD DF310EE0 74AB636 4597E899 A02555DC 6F431C5C
0846851D F9A8B419 5DEDE7EA1 B1D510BD 7EE74D73 FAF3B6C3 1EFCF4268
359046F4 EB879F92 4004943B 481C6CD7 889A002E D5EE382B C9190D6A
FC026E47 9558E447 5677E9AA 9E3050E2 765694DF C81F56E8 80B96E71
60C980DD 98EDD3DF FFFFFFFFFF
```

The generator is: 19 (decimal).
Appendix B. SRP Test Vectors

The following test vectors demonstrate calculation of the verifier and premaster secret.

I = "alice"

P = "password123"

s = BEB25379 D1A8581E B5A72767 3A2441EE

N, g = <1024-bit parameters from Appendix A>

k = 7556AA04 5AEF2CDD 07ABAFOF 665C3E81 8913186F

x = 94B7555A ABE9127C C58CCF49 93DB6CF8 4D16C124

v =

7E273DE8 696FCC4F 4E337D05 B4B375BE B0DDE156 9E8FA00A 9886D812
9BADA1F1 822223CA 1A605B53 0E379BA4 729FDC59 F105B478 7E518EF5
C671085A 1447B52A 48CF1970 B4FB6F84 00BBF4CE BFB81B61 52E08AB5
EA53D15C 1AFF87B2 B9DA6E04 E058AD51 CC72BFC9 033B5648 26480D78
E955A5E2 9E7AB245 DB2BE315 E2099AFB

a =

60975527 035CF2AD 1989806F 0407210B C81EDCO4 E2762A56 AFD529DD
DA2D4393

b =

E487CB59 D31AC550 471E81F0 0F6928E0 1DDA08E9 74A004F4 9E61F5D1
05284D20

A =

61D5E490 F6F1B795 47B0704C 436F523D D0E56OF0 C64115BB 72557EC4
4352E890 3211C046 92272D8B 2D1A5358 A2CF1B6E 0BFCF99F 921530EC
8E393561 79EAE45E 42BA92AE ACED8251 71E1E8B9 AF6D9CO3 E1327F44
BE087EF0 6530E69F 66615261 EEF54073 CA11CF58 58F0EDFD FE15EFEA
B349EF5D 76988A36 72FAC47B 0769447B
B =

BD0C6151 2C692C0C B6D041FA 01BB152D 4916A1E7 7AF46AE1 05393011
BAF38964 DC46A067 0DD125B9 5A981652 236F99D9 B681CBF8 7837EC99
6C6DA044 53728610 D0C6DDB5 8B318885 D7D82C7F 8DEB75CE 7BD4FBA
37089E6F 9C6059F3 88838E7A 00030B33 1EB76840 910440B1 B27AAEAE
EB4012B7 D7665238 A8E3FB00 4B117B58

u =

CE38B959 3487DA98 554ED47D 70A7AE5F 462EF019

<premaster secret> =

B0DC82BA BCF30674 AE450C02 87745E79 90A3381F 63B387AA F271A10D
233861E3 59B48220 F7C4693C 9AE12B0A 6F67809F 0876E2D0 13800D6C
41BB59B6 D59795B5 00A172B4 A2A5903A 0BDCAF8A 70958EB 2AFAFA8F
3499B200 210DCC1F 10EB3394 3CD67FC8 8A2F39A4 BE5BEC4E C0A3212D
C346D7E4 74B29EDE 8A469FFE CA686E5A

Appendix C. Acknowledgements

Thanks to all on the IETF TLS mailing list for ideas and analysis.
Authors’ Addresses

David Taylor
Independent
EMail: dtaylor@gnutls.org

Tom Wu
Cisco
EMail: thomwu@cisco.com

Nikos Mavrogiannopoulos
Independent
EMail: nmav@gnutls.org
URI: http://www.gnutls.org/

Trevor Perrin
Independent
EMail: trevp@trevp.net
URI: http://trevp.net/
Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM 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.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.