TLS Working Group D. Taylor
Internet-Draft Forge Research Pty Ltd
Expires: December 16, 2003 T. Wu
Arcot Systems
N. Mavroyanopoulos
T. Perrin
June 17, 2003

Using SRP for TLS Authentication
draft-ietf-tls-srp-05

Status of this Memo

Copyright Notice

Abstract

This memo presents a technique for using the SRP [2] (Secure Remote Password) protocol as an authentication method for the TLS [1](Transport Layer Security) protocol.

Table of Contents

1. Introduction

At the time of writing TLS uses public key certificates, or Kerberos, for authentication.

These authentication methods do not seem well suited to the applications now being adapted to use TLS (IMAP [4], FTP [8], or TELNET [9], for example). Given that these protocols (and others like them) are designed to use the user name and password method of authentication, being able to safely use user names and passwords to authenticate the TLS connection provides a much easier route to additional security than implementing a public key infrastructure in certain situations.

SRP 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.

2. SRP Authentication in TLS

2.1 Modifications to the TLS Handshake Sequence

The advent of SRP-6 [3] allows the SRP protocol to be implemented using the standard sequence of handshake messages defined in [1].

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

2.1.1 Message Sequence

Handshake Message Flow for SRP Authentication

```
       Client                                       Server
         |                                            |
    Client Hello (I) ------------------------>       |
         |   <---------------------------- Server Hello
         |   <---------------------------- Certificate*
         |   <---------------------------- Server Key Exchange (N, g, s, B)
         |   <---------------------------- Server Hello Done
    Client Key Exchange (A) ----------------->       |
    [Change cipher spec]                              |
    Finished -------------------------------->       |
         |                              [Change cipher spec]
         |   <---------------------------- Finished
         |                                            |
    Application Data  <---------------> Application Data
```

* Indicates optional or situation-dependent messages that are not always sent.

Figure 1

The identifiers given after each message name refer to the SRP variables included in that message. The variables I, N, g, s, A, and B are defined in [3].

An extended client hello message, as defined in [10], is used to send the client identifier (the user name).

2.1.2 Session Re-use

The short handshake mechanism for re-using sessions for new connections, and renegotiating keys for existing connections will still work with the SRP authentication mechanism and handshake.

When a client attemps to re-use a session that uses SRP

authentication, it MUST include the SRP extension carrying the user name (I) in the client hello message, in case the server cannot or will not allow re-use of the session, meaning a full handshake sequence is required.

If the server does agree to re-use 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 attackers cannot replace the authenticated identity without supplying the proper authentication information.

2.2 Text Preparation

The user name and password strings shall be UTF-8 encoded Unicode, prepared using the "SASLprep" [7] profile of "stringprep" [6].

2.3 SRP Verifier Creation

The verifier is created by applying the SRP-SHA1 mechanism as described in RFC 2945 [2] to the user name and password.

2.4 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 2.7.

2.4.1 Client hello

The user name is appended to the standard client hello message using the hello message extension mechanism defined in [10].

The client may offer SRP ciphersuites in the hello message but omit the SRP extension. If the server would like to select an SRP ciphersuite in this case, the server will return a missing_srp_username alert (see Section 2.8) immediately after processing the client hello message. This alert signals the client to resend the hello message, this time with the SRP extension. Through this idiom, the client can 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.

If the server doesn't have a verifier for the given user name, the server MAY abort the handshake with an unknown_srp_username alert (see Section 2.8). 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 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 (g, N) 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. For B, the server can return a random value between 2 and N-2 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.

2.4.2 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 2.6 for details of which ciphersuites defined in this document require a server certificate to be sent.

Because the server's certificate is only used for generating a digital signature in SRP cipher suites, the certificate sent MUST contain a public key that can be used for verifying digital signatures.

2.4.3 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 value of (I) received in the client hello extension. The server key exchange message also contains the server's public value (B).

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

The group parameters (g, N) sent in this message MUST have N as a safe prime (a prime of the form N=2q+1, where q is also prime), and g as a generator % N. The SRP group parameters in Appendix A are proven to have these properties, so the client SHOULD accept any parameters from this Appendix which have large enough moduli to meet his security requirements. The client MAY accept other group parameters from the server, either by prior arrangement, or by checking the parameters himself.

To check that N is a safe prime, the client should use some method such as performing 64 iterations of the Miller-Rabin test with random bases (selected from 2 to N-2) on both N and q (by performing 64 iterations, the probability of a false positive is no more than

2^-128). To check that g is a generator % N, the client can check that g^q equals -1 % N. Performing these checks may be time-consuming: after checking new parameters, the client may want to add them to a known-good list.

Group parameters that are not accepted via one of the above methods MUST be rejected with an illegal_parameter alert.

The client MUST abort the handshake with an illegal_parameter alert if B % N is equal to zero.

2.4.4 Client key exchange

The client key exchange message carries the client's public value (A).

The server MUST abort the handshake with an illegal_parameter alert if A % N is equal to zero, 1, or -1.

2.5 Calculating the Pre-master Secret

The shared secret resulting from the SRP calculations (S) (defined in [2]) is used as the pre-master secret.

The finished messages perform the same function as the client and server evidence messages (M1 and M2) specified in [2]. If either the client or the server calculate an incorrect value, the finished messages will not be understood, and the connection will be dropped as specified in [1].

2.6 Cipher Suite Definitions

The following cipher suites are added by this draft. The usage of AES ciphersuites is as defined in [5].

```
CipherSuite TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA     = { 0x00,0x50 };

CipherSuite TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA = { 0x00,0x51 };

CipherSuite TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA = { 0x00,0x52 };

CipherSuite TLS_SRP_SHA_WITH_AES_128_CBC_SHA      = { 0x00,0x53 };

CipherSuite TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA  = { 0x00,0x54 };

CipherSuite TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA  = { 0x00,0x55 };

CipherSuite TLS_SRP_SHA_WITH_AES_256_CBC_SHA      = { 0x00,0x56 };
```

```
CipherSuite TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA  = { 0x00,0x57 };

CipherSuite TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA  = { 0x00,0x58 };
```

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

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.

Implementations conforming to this specification MUST implement the TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA ciphersuite, SHOULD implement the TLS_SRP_SHA_WITH_AES_128_CBC_SHA and TLS_SRP_SHA_WITH_AES_256_CBC_SHA ciphersuites, and MAY implement the remaining ciphersuites.

2.7 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 [1].

2.7.1 ExtensionType

A new value, "srp(6)", has been added to the enumerated ExtensionType, defined in [10]. This value MUST be used as the extension number for the SRP extension.

2.7.2 Client Hello

The "extension_data" field of the srp extension SHALL contain: opaque srp_I<1..2^8-1> where srp_I is the user name.

2.7.3 Server Key Exchange

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. The following table gives the SignatureAlgorithm value to be used for each ciphersuite.

Ciphersuite	SignatureAlgorithm
TLS_SRP_SHA_WITH_3DES_EDE_CBC_SHA	anonymous

```
      TLS_SRP_SHA_RSA_WITH_3DES_EDE_CBC_SHA      rsa

      TLS_SRP_SHA_DSS_WITH_3DES_EDE_CBC_SHA      dsa

      TLS_SRP_SHA_WITH_AES_128_CBC_SHA           anonymous

      TLS_SRP_SHA_RSA_WITH_AES_128_CBC_SHA       rsa

      TLS_SRP_SHA_DSS_WITH_AES_128_CBC_SHA       dsa

      TLS_SRP_SHA_WITH_AES_256_CBC_SHA           anonymous

      TLS_SRP_SHA_RSA_WITH_AES_256_CBC_SHA       rsa

      TLS_SRP_SHA_DSS_WITH_AES_256_CBC_SHA       dsa
```

```
   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 */
```

2.7.4 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 an ClientSRPPublic structure.

An extra value, srp, has been added to the enumerated KeyExchangeAlgorithm, originally defined in TLS [1].


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

enum { rsa, diffie_hellman, srp } KeyExchangeAlgorithm;

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

2.8 Error Alerts

Two new error alerts are defined:

- "unknown_srp_username" (120) - this alert MAY be sent by a server that receives an unknown user name. This message is always fatal.

- "missing_srp_username" (121) - this alert MUST be sent by a server which would like to select an offered SRP ciphersuite, if the SRP extension is absent from the client's hello message. This alert may be fatal or a warning. If it is a warning, the server MUST restart its handshake protocol without closing the TLS session, and the client MAY either treat the warning as fatal and close the session, or send the server a new hello message on the same session. By sending a new hello on the same session, the client can use the idiom described in 2.3.1 without terminating a current TLS session which might be protecting the handshake (and thus the user name).

3. Security Considerations

If an attacker is able to steal the SRP verifier file, the attacker can masquerade as the real server, and can also use dictionary attacks to recover client passwords. Filesystem based X.509 certificate installations are vulnerable to a similar attack unless the server's certificate is issued from a PKI that maintains revocation lists, and the client TLS code can both contact the PKI and make use of the revocation list.

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 session, then renegotiate an SRP-authenticated session with the handshake protected by the first session.

The checks described in Section 2.4.3 and Section 2.4.4 on the received values for A and B are crucial for security and MUST be performed.

The private exponentials (a and b in [2]) SHOULD be at least 256 bit random numbers, to give approximately 128 bits of security against certain methods of calculating discrete logarithms [12]. Increasing the length of these exponentials may increase security, but it also increases the computation cost."

References

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

[2] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September 2000.

[3] Wu, T., "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol", October 2002.

[4] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, June 1999.

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

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

[7] Zeilenga, K., "SASLprep: Stringprep profile for user names and passwords", draft-ietf-tn3270e-telnet-tls-06 (work in

progress), February 2003.

[8] Ford-Hutchinson, P., Carpenter, M., Hudson, T., Murray, E. and V. Wiegand, "Securing FTP with TLS", draft-murray-auth-ftp-ssl-09 (work in progress), April 2002.

[9] Boe, M. and J. Altman, "TLS-based Telnet Security", draft-ietf-sasl-saslprep-00 (work in progress), April 2002.

[10] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J. and T. Wright, "TLS Extensions", draft-ietf-tls-extensions-06 (work in progress), February 2003.

[11] Kivinen, T. and M. Kojo, "More Modular Exponentiation (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[12] van Oorschot, P. and M. Wiener, "On Diffie-Hellman Key Agreement with Short Exponents", 1996.

Authors' Addresses

David Taylor
Forge Research Pty Ltd

EMail: DavidTaylor@forge.com.au
URI: http://www.forge.com.au/

Tom Wu
Arcot Systems

EMail: tom@arcot.com
URI: http://www.arcot.com/

Nikos Mavroyanopoulos

EMail: nmav@gnutls.org
URI: http://www.gnutls.org/

Trevor Perrin

EMail: trevp@trevp.net
URI: http://trevp.net/

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 [11], but generators have been calculated that are primitive roots of N, unlike the generators in [11].

The 1024, 1536, and 2048-bit groups MUST be supported.

1. 1024-bit Group

   The hexadecimal value 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 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 E3A9BEB8 7F8A2FE9 B8B5292E 5A021FFF 5E91479E
8CE7A28C 2442C6F3 15180F93 499A234D CF76E3FE D135F9BB
```

   The generator is: 2.

3. 2048-bit Group

   The hexadecimal value is:

```
AC6BDB41 324A9A9B F166DE5E 1389582F AF72B665 1987EE07 FC319294
3DB56050 A37329CB B4A099ED 8193E075 7767A13D D52312AB 4B03310D
CD7F48A9 DA04FD50 E8083969 EDB767B0 CF609517 9A163AB3 661A05FB
D5FAAAE8 2918A996 2F0B93B8 55F97993 EC975EEA A80D740A DBF4FF74
7359D041 D5C33EA7 1D281E44 6B14773B CA97B43A 23FB8016 76BD207A
436C6481 F1D2B907 8717461A 5B9D32E6 88F87748 544523B5 24B0D57D
5EA77A27 75D2ECFA 032CFBDB F52FB378 61602790 04E57AE6 AF874E73
03CE5329 9CCC041C 7BC308D8 2A5698F3 A8D0C382 71AE35F8 E9DBFBB6
```



```
94B5C803 D89F7AE4 35DE236D 525F5475 9B65E372 FCD68EF2 0FA7111F
```

9E4AFF73

The generator is: 2.

4. 3072-bit Group

This prime is: 2^3072 - 2^3008 - 1 + 2^64 * { [2^2942 pi] + 1690314 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B E39E772C
180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D
04507A33 A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D
B3970F85 A6E1E4C7 ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226
1AD2EE6B F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31 43DB5BFC
E0FD108E 4B82D120 A93AD2CA FFFFFFFF FFFFFFFF
```

The generator is: 5.

5. 4096-bit Group

This prime is: 2^4096 - 2^4032 - 1 + 2^64 * { [2^3966 pi] + 240904 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B E39E772C
180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D
04507A33 A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D
B3970F85 A6E1E4C7 ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226
1AD2EE6B F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
```

```
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31 43DB5BFC
E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7 88719A10 BDBA5B26
99C32718 6AF4E23C 1A946834 B6150BDA 2583E9CA 2AD44CE8 DBBBC2DB
04DE8EF9 2E8EFC14 1FBECAA6 287C5947 4E6BC05D 99B2964F A090C3A2
233BA186 515BE7ED 1F612970 CEE2D7AF B81BDD76 2170481C D0069127
D5B05AA9 93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34063199
FFFFFFFF FFFFFFFF
```

The generator is: 5.

6. 6144-bit Group

This prime is: 2^6144 - 2^6080 - 1 + 2^64 * { [2^6014 pi] + 929484 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B E39E772C
180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D
04507A33 A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D
B3970F85 A6E1E4C7 ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226
1AD2EE6B F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31 43DB5BFC
E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7 88719A10 BDBA5B26
99C32718 6AF4E23C 1A946834 B6150BDA 2583E9CA 2AD44CE8 DBBBC2DB
04DE8EF9 2E8EFC14 1FBECAA6 287C5947 4E6BC05D 99B2964F A090C3A2
233BA186 515BE7ED 1F612970 CEE2D7AF B81BDD76 2170481C D0069127
D5B05AA9 93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34028492
36C3FAB4 D27C7026 C1D4DCB2 602646DE C9751E76 3DBA37BD F8FF9406
AD9E530E E5DB382F 413001AE B06A53ED 9027D831 179727B0 865A8918
DA3EDBEB CF9B14ED 44CE6CBA CED4BB1B DB7F1447 E6CC254B 33205151
2BD7AF42 6FB8F401 378CD2BF 5983CA01 C64B92EC F032EA15 D1721D03
F482D7CE 6E74FEF6 D55E702F 46980C82 B5A84031 900B1C9E 59E7C97F
BEC7E8F3 23A97A7E 36CC88BE 0F1D45B7 FF585AC5 4BD407B2 2B4154AA
CC8F6D7E BF48E1D8 14CC5ED2 0F8037E0 A79715EE F29BE328 06A1D58B
B7C5DA76 F550AA3D 8A1FBFF0 EB19CCB1 A313D55C DA56C9EC 2EF29632
387FE8D7 6E3C0468 043E8F66 3F4860EE 12BF2D5B 0B7474D6 E694F91E
6DCC4024 FFFFFFFF FFFFFFFF
```

The generator is: 5.

7. 8192-bit Group

This prime is: 2^8192 - 2^8128 - 1 + 2^64 * { [2^8062 pi] + 4743158 }

Its hexadecimal value is:

```
FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08
8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B
302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9
A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6
49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8
FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D
670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B E39E772C
180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718
3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D
04507A33 A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D
B3970F85 A6E1E4C7 ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226
1AD2EE6B F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C
BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31 43DB5BFC
E0FD108E 4B82D120 A9210801 1A723C12 A787E6D7 88719A10 BDBA5B26
99C32718 6AF4E23C 1A946834 B6150BDA 2583E9CA 2AD44CE8 DBBBC2DB
04DE8EF9 2E8EFC14 1FBECAA6 287C5947 4E6BC05D 99B2964F A090C3A2
233BA186 515BE7ED 1F612970 CEE2D7AF B81BDD76 2170481C D0069127
D5B05AA9 93B4EA98 8D8FDDC1 86FFB7DC 90A6C08F 4DF435C9 34028492
36C3FAB4 D27C7026 C1D4DCB2 602646DE C9751E76 3DBA37BD F8FF9406
AD9E530E E5DB382F 413001AE B06A53ED 9027D831 179727B0 865A8918
DA3EDBEB CF9B14ED 44CE6CBA CED4BB1B DB7F1447 E6CC254B 33205151
2BD7AF42 6FB8F401 378CD2BF 5983CA01 C64B92EC F032EA15 D1721D03
F482D7CE 6E74FEF6 D55E702F 46980C82 B5A84031 900B1C9E 59E7C97F
BEC7E8F3 23A97A7E 36CC88BE 0F1D45B7 FF585AC5 4BD407B2 2B4154AA
CC8F6D7E BF48E1D8 14CC5ED2 0F8037E0 A79715EE F29BE328 06A1D58B
B7C5DA76 F550AA3D 8A1FBFF0 EB19CCB1 A313D55C DA56C9EC 2EF29632
387FE8D7 6E3C0468 043E8F66 3F4860EE 12BF2D5B 0B7474D6 E694F91E
6DBE1159 74A3926F 12FEE5E4 38777CB6 A932DF8C D8BEC4D0 73B931BA
3BC832B6 8D9DD300 741FA7BF 8AFC47ED 2576F693 6BA42466 3AAB639C
5AE4F568 3423B474 2BF1C978 238F16CB E39D652D E3FDB8BE FC848AD9
22222E04 A4037C07 13EB57A8 1A23F0C7 3473FC64 6CEA306B 4BCBC886
2F8385DD FA9D4B7F A2C087E8 79683303 ED5BDD3A 062B3CF5 B3A278A6
```

```
6D2A13F8 3F44F82D DF310EE0 74AB6A36 4597E899 A0255DC1 64F31CC5
0846851D F9AB4819 5DED7EA1 B1D510BD 7EE74D73 FAF36BC3 1ECFA268
359046F4 EB879F92 4009438B 481C6CD7 889A002E D5EE382B C9190DA6
FC026E47 9558E447 5677E9AA 9E3050E2 765694DF C81F56E8 80B96E71
60C980DD 98EDD3DF FFFFFFFF FFFFFFFF
```

The generator is: 19 (decimal).

Appendix B. Acknowledgements

Thanks to all on the IETF tls mailing list for ideas and analysis.

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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