```
Network Working Group                                 Markus Friedl
INTERNET-DRAFT                                        Niels Provos
Expires in six months                            William A. Simpson
                                                           May 2000
```

Diffie-Hellman Group Exchange for the SSH Transport Layer Protocol
draft-provos-secsh-dh-group-exchange-00.txt

Abstract

This memo describes a new key exchange method for the SSH protocol. It allows the SSH server to propose to the client new groups on which to perform the Diffie-Hellman key exchange. The proposed groups need not be fixed and can change with time.

Overview and Rational

SSH is a de-facto standard for secure remote login on the Internet. Currently, SSH performs the initial key exchange using the "diffie-hellman-group1-sha1" method. This method prescribes a fixed group on which all operations are performed. The security of the Diffie-Hellman key exchange is based on the difficulty of solving the Discrete Logarithm Problem (DLP). Since we expect that the SSH protocol will be in use for many years in the future, we fear that extensive precomputation and more efficient alogorithms to compute the Discrete Logarithm might pose a security threat to the SSH protocol.

The ability to propose new moduli will reduce the possibility to use precomputation for more efficient calculation of the DL. The server can constantly compute new moduli in the background.

Diffie-Hellman Group and Key Exchange

The Diffie-Hellman key exchange provides a shared secret that can not be determined by either party alone. The key exchange is combined with a signature with the host key to provide host authentication.

The server keeps a list of safe primes and corresponding generators that it can select from. A prime p is safe, if p = 2q + 1, and q is prime. New primes can be generated in the background. The server SHOULD know at least one safe prime that has 1024 or more bits.

The generator g should be chosen such that the order of the generated subgroup does not factor into small primes, i.e., with p = 2q + 1, the order has to be either q or p - 1. If the order is p - 1, then the exponents generate all possible public-values, evenly distributed throughout the range of the modulus p, without cycling through a smaller subset. Such a generator is called a "primitive root" (which is trivial to find when p is "safe").

Implementation Notes:

One useful technique is to select the generator, and then limit the modulus selection sieve to primes with that generator:

```
  2   when p (mod 24) = 11.
  5   when p (mod 10) = 3 or 7.
```

It is recommened to to use 2 as generator, because it improves efficiency in multiplication performance. It is usable even when it is not a primitive root, as it still covers half of the space of possible residues.

The client requests a minimum modulus size from the server. In the following description (C is the client, S is the server; n is the minimal number of bits the subgroup the server replies with should have; p is a large safe prime and g is a generator for a subgroup of

GF(p); V_S is S's version string; V_C is C's version string; K_S is S's public host key; I_C is C's KEXINIT message and I_S S's KEXINIT message which have been exchanged before this part begins):

1. C sends n, the minimal number of bits the subgroup the server replies with should have.

2. S finds a group that matches the clients request the closest and sends p and g to C.

3. C generates a random number x (1 < x < (p-1)/2). It computes e = g^x mod p, and sends "e" to S.

4. S generates a random number y (0 < y < (p-1)/2) and computes f = g^y mod p. S receives "e". It computes K = e^y mod p, H = hash(V_C || V_S || I_C || I_S || K_S || n || p || g || e || f || K) (these elements are encoded according to their types; see below), and signature s on H with its private host key. S sends "K_S || f || s" to C. The signing operation may involve a second hashing operation.

5. C verifies that K_S really is the host key for S (e.g. using certificates or a local database). C is also allowed to accept the key without verification; however, doing so will render the protocol insecure against active attacks (but may be desirable for practical reasons in the short term in many environments). C then computes K = f^x mod p, H = hash(V_C || V_S || I_C || I_S || K_S || n || p || g || e || f || K), and verifies the signature s on H.

   Either side MUST NOT send or accept e or f values that are not in the range [1, p-1]. If this condition is violated, the key exchange fails.

This is implemented with the following messages. The hash algorithm for computing the exchange hash is defined by the method name, and is called HASH. The public key algorithm for signing is negotiated with the KEXINIT messages.

First, the client sends:

```
  byte      SSH_MSG_KEY_DH_GEX_REQUEST
  uint32    n, number of bits the subgroup should have at least
```

The server responds with

```
  byte      SSH_MSG_KEX_DH_GEX_GROUP
  mpint     p, safe prime
  mpint     g, generator for subgroup in GF(p)
```

The client responds with:

```
  byte      SSH_MSG_KEX_DH_GEX_INIT
  mpint     e
```

The server responds with:

```
  byte      SSH_MSG_KEX_DH_GEX_REPLY
  string    server public host key and certificates (K_S)
  mpint     f
  string    signature of H
```

The hash H is computed as the HASH hash of the concatenation of the following:

```
  string    V_C, the client's version string (CR and NL excluded)
  string    V_S, the server's version string (CR and NL excluded)
  string    I_C, the payload of the client's SSH_MSG_KEXINIT
  string    I_S, the payload of the server's SSH_MSG_KEXINIT
  string    K_S, the host key
  uint32    n, number of bits the client requested
  mpint     p, safe prime
  mpint     g, generator for subgroup
  mpint     e, exchange value sent by the client
  mpint     f, exchange value sent by the server
  mpint     K, the shared secret
```

This value is called the exchange hash, and it is used to authenticate the key exchange.

diffie-hellman-group-exchange-sha1

The "diffie-hellman-group-exchange-sha1" method specifies Diffie-Hellman Group and Key Exchange with SHA-1 as HASH.

Summary of Message numbers

The following message numbers have been defined in this document.

```
  #define SSH_MSG_KEX_DH_GEX_REQUEST      30
  #define SSH_MSG_KEX_DH_GEX_GROUP        31
  #define SSH_MSG_KEX_DH_GEX_INIT         32
  #define SSH_MSG_KEX_DH_GEX_REPLY        33
```

The numbers 30-49 are key exchange specific and may be redefined by other kex methods.

Security Considerations

The use of multiple moduli inhibits a determined attacker from pre-calculating moduli exchange values, and discourages dedication of resources for analysis of any particular modulus.

It is important to only employ safe primes as moduli. Oorshot and Wiener note that using short private exponents with a random prime modulus p makes the computation of the discrete logarithm easy [1]. However, they also state that this problem does not apply to safe primes.

The least significant bit of the private exponent can be recovered, when the modulus is a safe prime [2]. However, this is not a problem, if the size of the private exponent is big enough. Related to this, Waldvogel and Massey note: When private exponents are chosen independently and uniformly at random from {0,...,p-2}, the key entropy is less than 2 bits away from the maximum, lg(p-1) [3].

Acknowledgments

The document is derived in part from "SSH Transport Layer Protocol" by T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne and S. Lehtinen.

Markku-Juhani Saarinen pointed out that the least significant bit of the private exponent can be recovered efficiently when using safe primes and a subgroup with an order divisible by two.

Bodo Moeller suggested that the server sends only one group reducing the complexity of the implementation and the amount of data that needs to be exchanged between client and server.

Bibliography

[1] P. C. van Oorschot and M. J. Wiener, On Diffie-Hellman key agreement with short exponents, In Advances in Cryptology - EUROCRYPT'96, LNCS 1070, Springer-Verlag, 1996, pp.332-343.

[2] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied Cryptography. CRC Press, 1996.

[3] C. P. Waldvogel and J. L. Massey, The probability distribution of the Diffie-Hellman key, in Proceedings of AUSCRYPT 92, LNCS 718, Springer-Verlag, 1993, pp. 492-504.

Appendix A: Generation of safe primes

The Handbook of Applied Cryptography [2] lists the following algorithm to generate a k-bit safe prime p. It has been modified so that 2 is a generator for the multiplicative group mod p.

```
1. Do the following:
  1.1 Select a random (k-1)-bit prime q, so that q mod 12 = 5.
  1.2 Compute p := 2q + 1, and test whether p is prime, (using, e.g.
      trial division and the Rabin-Miller test.)
  Repeat until p is prime.
```

If an implementation uses the OpenSSL libraries, a group consisting of a 1024-bit safe prime and 2 as generator can be created as follows:

```
DH *d = NULL;
d = DH_generate_parameters(1024, DH_GENERATOR_2, NULL, NULL);
BN_print_fp(stdout, d->p);
```

The order of the subgroup generated by 2 is q = p - 1.

Author's Address

Niels Provos
CITI
519 W. William Street
Ann Arbor, MI, 48103

Phone: (734) 764-5207

EMail: provos@citi.umich.edu