Network Working Group
Request for Comments: 2631
Category: Standards Track

E. Rescorla
RTFM Inc.

June 1999

Diffie-Hellman Key Agreement Method

Status of this Memo

Copyright Notice

Abstract

This document standardizes one particular Diffie-Hellman variant, based on the ANSI X9.42 draft, developed by the ANSI X9F1 working group. Diffie-Hellman is a key agreement algorithm used by two parties to agree on a shared secret. An algorithm for converting the shared secret into an arbitrary amount of keying material is provided. The resulting keying material is used as a symmetric encryption key. The Diffie-Hellman variant described requires the recipient to have a certificate, but the originator may have a static key pair (with the public key placed in a certificate) or an ephemeral key pair.

Table of Contents

1. Introduction

In [DH76] Diffie and Hellman describe a means for two parties to agree upon a shared secret in such a way that the secret will be unavailable to eavesdroppers. This secret may then be converted into cryptographic keying material for other (symmetric) algorithms. A large number of minor variants of this process exist. This document describes one such variant, based on the ANSI X9.42 specification.

1.1. Requirements Terminology

Keywords "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT" and "MAY" that appear in this document are to be interpreted as described in [RFC2119].

2. Overview Of Method

Diffie-Hellman key agreement requires that both the sender and recipient of a message have key pairs. By combining one's private key and the other party's public key, both parties can compute the same shared secret number. This number can then be converted into cryptographic keying material. That keying material is typically used as a key-encryption key (KEK) to encrypt (wrap) a content-encryption key (CEK) which is in turn used to encrypt the message data.

2.1. Key Agreement

The first stage of the key agreement process is to compute a shared secret number, called ZZ. When the same originator and recipient public/private key pairs are used, the same ZZ value will result. The ZZ value is then converted into a shared symmetric cryptographic key. When the originator employs a static private/public key pair, the introduction of a public random value ensures that the resulting symmetric key will be different for each key agreement.

2.1.1. Generation of ZZ

X9.42 defines that the shared secret ZZ is generated as follows:

```
ZZ = g ^ (xb * xa) mod p
```

Note that the individual parties actually perform the computations:

```
ZZ = (yb ^ xa)  mod p  = (ya ^ xb)  mod p
```

where ^ denotes exponentiation

```
ya is party a's public key; ya = g ^ xa mod p
yb is party b's public key; yb = g ^ xb mod p
xa is party a's private key
xb is party b's private key
p is a large prime
q is a large prime
g = h^{(p-1)/q} mod p, where
h is any integer with 1 < h < p-1 such that h{(p-1)/q} mod p > 1
  (g has order q mod p; i.e. g^q mod p = 1 if g!=1)
j a large integer such that p=qj + 1
(See Section 2.2 for criteria for keys and parameters)
```

In [CMS], the recipient's key is identified by the CMS RecipientIdentifier, which points to the recipient's certificate. The sender's public key is identified using the OriginatorIdentifierOrKey field, either by reference to the sender's certificate or by inline inclusion of a public key.

2.1.2. Generation of Keying Material

X9.42 provides an algorithm for generating an essentially arbitrary amount of keying material from ZZ. Our algorithm is derived from that algorithm by mandating some optional fields and omitting others.

```
KM = H ( ZZ || OtherInfo)
```

H is the message digest function SHA-1 [FIPS-180] ZZ is the shared secret value computed in Section 2.1.1. Leading zeros MUST be preserved, so that ZZ occupies as many octets as p. For instance, if p is 1024 bits, ZZ should be 128 bytes long. OtherInfo is the DER encoding of the following structure:

```
OtherInfo ::= SEQUENCE {
  keyInfo KeySpecificInfo,
  partyAInfo [0] OCTET STRING OPTIONAL,
  suppPubInfo [2] OCTET STRING
```

```
}

KeySpecificInfo ::= SEQUENCE {
  algorithm OBJECT IDENTIFIER,
  counter OCTET STRING SIZE (4..4) }
```

Note that these ASN.1 definitions use EXPLICIT tagging. (In ASN.1, EXPLICIT tagging is implicit unless IMPLICIT is explicitly specified.)

algorithm is the ASN.1 algorithm OID of the CEK wrapping algorithm with which this KEK will be used. Note that this is NOT an AlgorithmIdentifier, but simply the OBJECT IDENTIFIER. No parameters are used.

counter is a 32 bit number, represented in network byte order. Its initial value is 1 for any ZZ, i.e. the byte sequence 00 00 00 01 (hex), and it is incremented by one every time the above key generation function is run for a given KEK.

partyAInfo is a random string provided by the sender. In CMS, it is provided as a parameter in the UserKeyingMaterial field (encoded as an OCTET STRING). If provided, partyAInfo MUST contain 512 bits.

suppPubInfo is the length of the generated KEK, in bits, represented as a 32 bit number in network byte order. E.g. for 3DES it would be the byte sequence 00 00 00 C0.

To generate a KEK, one generates one or more KM blocks (incrementing counter appropriately) until enough material has been generated. The KM blocks are concatenated left to right I.e. KM(counter=1) || KM(counter=2)...

Note that the only source of secret entropy in this computation is ZZ. Even if a string longer than ZZ is generated, the effective key space of the KEK is limited by the size of ZZ, in addition to any security level considerations imposed by the parameters p and q. However, if partyAInfo is different for each message, a different KEK will be generated for each message. Note that partyAInfo MUST be used in Static-Static mode, but MAY appear in Ephemeral-Static mode.

2.1.3. KEK Computation

Each key encryption algorithm requires a specific size key (n). The KEK is generated by mapping the left n-most bytes of KM onto the key. For 3DES, which requires 192 bits of keying material, the algorithm must be run twice, once with a counter value of 1 (to generate K1', K2', and the first 32 bits of K3') and once with a counter value of 2

(to generate the last 32 bits of K3). K1',K2' and K3' are then parity adjusted to generate the 3 DES keys K1,K2 and K3. For RC2-128, which requires 128 bits of keying material, the algorithm is run once, with a counter value of 1, and the left-most 128 bits are directly converted to an RC2 key. Similarly, for RC2-40, which requires 40 bits of keying material, the algorithm is run once, with a counter value of 1, and the leftmost 40 bits are used as the key.

2.1.4. Keylengths for common algorithms

Some common key encryption algorithms have KEKs of the following lengths.

3-key 3DES	192 bits
RC2-128	128 bits
RC2-40	40 bits

RC2 effective key lengths are equal to RC2 real key lengths.

2.1.5. Public Key Validation

The following algorithm MAY be used to validate a received public key y.

1. Verify that y lies within the interval [2,p-1]. If it does not, the key is invalid.
2. Compute y^q mod p. If the result == 1, the key is valid. Otherwise the key is invalid.

The primary purpose of public key validation is to prevent a small subgroup attack [LAW98] on the sender's key pair. If Ephemeral-Static mode is used, this check may not be necessary. See also [P1363] for more information on Public Key validation.

Note that this procedure may be subject to pending patents.

2.1.6. Example 1

ZZ is the 20 bytes 00 01 02 03 04 05 06 07 08 09
0a 0b 0c 0d 0e 0f 10 11 12 13

The key wrap algorithm is 3DES-EDE wrap.

No partyAInfo is used.

Consequently, the input to the first invocation of SHA-1 is:

00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 ; ZZ

```
   30 1d
      30 13
         06 0b 2a 86 48 86 f7 0d 01 09 10 03 06          ; 3DES wrap OID
         04 04
            00 00 00 01                                      ; Counter
      a2 06
         04 04
         00 00 00 c0                                      ; key length
```

And the output is the 20 bytes:

```
a0 96 61 39 23 76 f7 04 4d 90 52 a3 97 88 32 46 b6 7f 5f 1e
```

The input to the second invocation of SHA-1 is:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 ; ZZ
30 1d
   30 13
      06 0b 2a 86 48 86 f7 0d 01 09 10 03 06          ; 3DES wrap OID
      04 04
         00 00 00 02                                      ; Counter
   a2 06
      04 04
      00 00 00 c0                                      ; key length
```

And the output is the 20 bytes:

```
f6 3e b5 fb 5f 56 d9 b6 a8 34 03 91 c2 d3 45 34 93 2e 11 30
```

Consequently,
K1'=a0 96 61 39 23 76 f7 04
K2'=4d 90 52 a3 97 88 32 46
K3'=b6 7f 5f 1e f6 3e b5 fb

Note: These keys are not parity adjusted

2.1.7. Example 2

ZZ is the 20 bytes 00 01 02 03 04 05 06 07 08 09
0a 0b 0c 0d 0e 0f 10 11 12 13

The key wrap algorithm is RC2-128 key wrap, so we need 128 bits (16 bytes) of keying material.

The partyAInfo used is the 64 bytes

```
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
```

```
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
```

Consequently, the input to SHA-1 is:

```
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 ; ZZ
30 61
   30 13
      06 0b 2a 86 48 86 f7 0d 01 09 10 03 07           ; RC2 wrap OID
      04 04
         00 00 00 01                                        ; Counter
   a0 42
      04 40
         01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01 ; partyAInfo
         01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
         01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
         01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 01
   a2 06
      04 04
         00 00 00 80                                      ; key length
```

And the output is the 20 bytes:

```
48 95 0c 46 e0 53 00 75 40 3c ce 72 88 96 04 e0 3e 7b 5d e9
```

Consequently,
K=48 95 0c 46 e0 53 00 75 40 3c ce 72 88 96 04 e0

2.2. Key and Parameter Requirements

X9.42 requires that the group parameters be of the form p=jq + 1 where q is a large prime of length m and j>=2. An algorithm for generating primes of this form (derived from the algorithms in FIPS PUB 186-1[FIPS-186] and [X942]can be found in appendix A.

X9.42 requires that the private key x be in the interval [2, (q - 2)]. x should be randomly generated in this interval. y is then computed by calculating g^x mod p. To comply with this memo, m MUST be >=160 bits in length, (consequently, q MUST be at least 160 bits long). When symmetric ciphers stronger than DES are to be used, a larger m may be advisable. p must be a minimum of 512 bits long.

2.2.1. Group Parameter Generation

Agents SHOULD generate domain parameters (g,p,q) using the following algorithm, derived from [FIPS-186] and [X942]. When this algorithm is used, the correctness of the generation procedure can be verified by a third party by the algorithm of 2.2.2.

2.2.1.1. Generation of p, q

This algorithm generates a p, q pair where q is of length m and p is of length L.

1. Set m' = m/160 where / represents integer division with rounding upwards. I.e. 200/160 = 2.

2. Set L'= L/160

3. Set N'= L/1024

4. Select an arbitrary bit string SEED such that the length of SEED >= m

5. Set U = 0

6. For i = 0 to m' - 1

U = U + (SHA1[SEED + i] XOR SHA1[(SEED + m' + i)) * 2^(160 * i)

Note that for m=160, this reduces to the algorithm of [FIPS-186]

U = SHA1[SEED] XOR SHA1[(SEED+1) mod 2^160].

5. Form q from U by computing U mod (2^m) and setting the most significant bit (the 2^(m-1) bit) and the least significant bit to 1. In terms of boolean operations, q = U OR 2^(m-1) OR 1. Note that 2^(m-1) < q < 2^m

6. Use a robust primality algorithm to test whether q is prime.

7. If q is not prime then go to 4.

8. Let counter = 0

9. Set R = seed + 2*m' + (L' * counter)

10. Set V = 0

12. For i = 0 to L'-1 do

V = V + SHA1(R + i) * 2^(160 * i)

13. Set W = V mod 2^L

14. Set X = W OR 2^(L-1)

Note that 0 <= W < 2^(L-1) and hence X >= 2^(L-1)

15. Set p = X - (X mod (2*q)) + 1

6. If p > 2^(L-1) use a robust primality test to test whether p is prime. Else go to 18.

17. If p is prime output p, q, seed, counter and stop.

18. Set counter = counter + 1

19. If counter < (4096 * N) then go to 8.

20. Output "failure"

Note: A robust primality test is one where the probability of a non-prime number passing the test is at most 2^-80. [FIPS-186] provides a suitable algorithm, as does [X942].

2.2.1.2. Generation of g

This section gives an algorithm (derived from [FIPS-186]) for generating g.

1. Let j = (p - 1)/q.

2. Set h = any integer, where 1 < h < p - 1 and h differs from any value previously tried.

3. Set g = h^j mod p

4. If g = 1 go to step 2

2.2.2. Group Parameter Validation

The ASN.1 for DH keys in [PKIX] includes elements j and validationParms which MAY be used by recipients of a key to verify that the group parameters were correctly generated. Two checks are possible:

1. Verify that p=qj + 1. This demonstrates that the parameters meet the X9.42 parameter criteria.
2. Verify that when the p,q generation procedure of [FIPS-186] Appendix 2 is followed with seed 'seed', that p is found when 'counter' = pgenCounter.

This demonstrates that the parameters were randomly chosen and do not have a special form.

Whether agents provide validation information in their certificates is a local matter between the agents and their CA.

2.3. Ephemeral-Static Mode

In Ephemeral-Static mode, the recipient has a static (and certified) key pair, but the sender generates a new key pair for each message and sends it using the originatorKey production. If the sender's key is freshly generated for each message, the shared secret ZZ will be similarly different for each message and partyAInfo MAY be omitted, since it serves merely to decouple multiple KEKs generated by the same set of pairwise keys. If, however, the same ephemeral sender key is used for multiple messages (e.g. it is cached as a performance optimization) then a separate partyAInfo MUST be used for each message. All implementations of this standard MUST implement Ephemeral-Static mode.

In order to resist small subgroup attacks, the recipient SHOULD perform the check described in 2.1.5. If an opponent cannot determine success or failure of a decryption operation by the recipient, the recipient MAY choose to omit this check. See also [LL97] for a method of generating keys which are not subject to small subgroup attack.

2.4. Static-Static Mode

In Static-Static mode, both the sender and the recipient have a static (and certified) key pair. Since the sender's and recipient's keys are therefore the same for each message, ZZ will be the same for each message. Thus, partyAInfo MUST be used (and different for each message) in order to ensure that different messages use different KEKs. Implementations MAY implement Static-Static mode.

In order to prevent small subgroup attacks, both originator and recipient SHOULD either perform the validation step described in Section 2.1.5 or verify that the CA has properly verified the validity of the key. See also [LL97] for a method of generating keys which are not subject to small subgroup attack.

Acknowledgements

The Key Agreement method described in this document is based on work done by the ANSI X9F1 working group. The author wishes to extend his thanks for their assistance.

The author also wishes to thank Stephen Henson, Paul Hoffman, Russ Housley, Burt Kaliski, Brian Korver, John Linn, Jim Schaad, Mark Schertler, Peter Yee, and Robert Zuccherato for their expert advice and review.

References

[CMS] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[FIPS-46-1] Federal Information Processing Standards Publication (FIPS PUB) 46-1, Data Encryption Standard, Reaffirmed 1988 January 22 (supersedes FIPS PUB 46, 1977 January 15).

[FIPS-81] Federal Information Processing Standards Publication (FIPS PUB) 81, DES Modes of Operation, 1980 December 2.

[FIPS-180] Federal Information Processing Standards Publication (FIPS PUB) 180-1, "Secure Hash Standard", 1995 April 17.

[FIPS-186] Federal Information Processing Standards Publication (FIPS PUB) 186, "Digital Signature Standard", 1994 May 19.

[P1363] "Standard Specifications for Public Key Cryptography", IEEE P1363 working group draft, 1998, Annex D.

[PKIX] Housley, R., Ford, W., Polk, W. and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, January 1999.

[LAW98] L. Law, A. Menezes, M. Qu, J. Solinas and S. Vanstone, "An efficient protocol for authenticated key agreement", Technical report CORR 98-05, University of Waterloo, 1998.

[LL97] C.H. Lim and P.J. Lee, "A key recovery attack on discrete log-based schemes using a prime order subgroup", B.S. Kaliski, Jr., editor, Advances in Cryptology - Crypto ’97, Lecture Notes in Computer Science, vol. 1295, 1997, Springer-Verlag, pp. 249-263.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[X942] "Agreement Of Symmetric Keys Using Diffie-Hellman and MQV Algorithms", ANSI draft, 1998.

Security Considerations

All the security in this system is provided by the secrecy of the private keying material. If either sender or recipient private keys are disclosed, all messages sent or received using that key are compromised. Similarly, loss of the private key results in an inability to read messages sent using that key.

Static Diffie-Hellman keys are vulnerable to a small subgroup attack [LAW98]. In practice, this issue arises for both sides in Static-Static mode and for the receiver during Ephemeral-Static mode. Sections 2.3 and 2.4 describe appropriate practices to protect against this attack. Alternatively, it is possible to generate keys in such a fashion that they are resistant to this attack. See [LL97]

The security level provided by these methods depends on several factors. It depends on the length of the symmetric key (typically, a 2^l security level if the length is l bits); the size of the prime q (a 2^{m/2} security level); and the size of the prime p (where the security level grows as a subexponential function of the size in bits). A good design principle is to have a balanced system, where all three security levels are approximately the same. If many keys are derived from a given pair of primes p and q, it may be prudent to have higher levels for the primes. In any case, the overall security is limited by the lowest of the three levels.

Author’s Address

Eric Rescorla
RTFM Inc.
30 Newell Road, #16
East Palo Alto, CA 94303

EMail: ekr@rtfm.com

Full Copyright Statement

Acknowledgement

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