Network Working Group
Request for Comments: 2785
Category: Informational

R. Zuccherato
Entrust Technologies

March 2000

Methods for Avoiding the "Small-Subgroup" Attacks on the Diffie-Hellman Key Agreement Method for S/MIME

Status of this Memo

Copyright Notice

Abstract

In some circumstances the use of the Diffie-Hellman key agreement scheme in a prime order subgroup of a large prime p is vulnerable to certain attacks known as "small-subgroup" attacks. Methods exist, however, to prevent these attacks. This document will describe the situations relevant to implementations of S/MIME version 3 in which protection is necessary and the methods that can be used to prevent these attacks.

1. Introduction

This document will describe those situations in which protection from "small-subgroup" type attacks is necessary when using Diffie-Hellman key agreement [RFC2631] in implementations of S/MIME version 3 [RFC2630, RFC2633]. Thus, the ephemeral-static and static-static modes of Diffie-Hellman will be focused on. Some possible non-S/MIME usages of CMS are also considered, though with less emphasis than the cases arising in S/MIME. The situations for which protection is necessary are those in which an attacker could determine a substantial portion (i.e. more than a few bits) of a user’s private key.

Protecting oneself from these attacks involves certain costs. These costs may include additional processing time either when a public key is certified or a shared secret key is derived, increased parameter generation time, and possibly the licensing of encumbered

technologies. All of these factors must be considered when deciding whether or not to protect oneself from these attacks, or whether to engineer the application so that protection is not necessary.

We will not consider "attacks" where the other party in the key agreement merely forces the shared secret value to be "weak" (i.e. from a small set of possible values) without attempting to compromise the private key. It is not worth the effort to attempt to prevent these attacks since the other party in the key agreement gets the shared secret and can simply make the plaintext public.

The methods described in this memo may also be used to provide protection from similar attacks on elliptic curve based Diffie-Hellman.

1.1 Notation

In this document we will use the same notation as in [RFC2631]. In particular 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; xa is in the interval [2, (q - 2)]
xb is Party B's private key; xb is in the interval [2, (q - 2)]
p 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)
q is a large prime
j a large integer such that p=q*j + 1
```

In this discussion, a "static" public key is one that is certified and is used for more than one key agreement, and an "ephemeral" public key is one that is not certified but is used only one time.

The order of an integer y modulo p is the smallest value of x greater than 1 such that y^x mod p = 1.

1.2 Brief Description of Attack

For a complete description of these attacks see [LAW] and [LIM].

If the other party in an execution of the Diffie-Hellman key agreement method has a public key not of the form described above, but of small order (where small means less than q) then he/she may be able to obtain information about the user's private key. In particular, if information on whether or not a given decryption was successful is available, if ciphertext encrypted with the agreed upon key is available, or if a MAC computed with the agreed upon key is available, information about the user's private key can be obtained.

Assume Party A has a valid public key ya and that Party B has a public key yb that is not of the form described in Section 1.1, rather yb has order r, where r is much less than q. Thus yb^r=1 mod p. Now, when Party A produces ZZ as yb^xa mod p, there will only be r possible values for ZZ instead of q-3 possible values. At this point Party B does not know the value ZZ, but may be able to exhaustively search for it.

If Party A encrypts plaintext with this value and makes that ciphertext available to Party B, Party B only needs to exhaustively search through r possibilities to determine which key produced the ciphertext. When the correct one is found, this gives information about the value of xa modulo r. Similarly, if Party A uses ZZ to decrypt a ciphertext and Party B is able to determine whether or not decryption was performed correctly, then information about xa can be obtained. The actual number of messages that must be sent or received for these attacks to be successful will depend on the structure of the prime p. However, it is not unreasonable to expect that the entire private key could be determined after as few as one hundred messages.

A similar attack can be mounted if Party B chooses a public key of the form yb=g^xb*f, where f is an element of small order. In this situation Party A will compute ZZ=yb^xa=g^(xa*xb)*f^xa mod p. Again, Party B can compute g^(xa*xb) and can therefore exhaust the small number of possible values of f^xa mod p to determine information about xa.

An attack is also possible if Party B has a public key yb of order r where r factors into small integers but is not necessarily a small integer itself. In this case, the attacker needs to know the value ZZ computed by Party A. From this value Party B can solve for Party A's private key modulo r using the Pohlig-Hellman [PH] algorithm.

However, this attack is not as practical as the cases already presented, where information about the private key is recovered from the *use* of ZZ, rather than ZZ itself, by exhaustive search.

2. Situations Where Protection Is Necessary

This section describes the situations in which the sender of a message should obtain protection against this type of attack and also those situations in which the receiver of a message should obtain protection. Each entity may decide independently whether it requires protection from these attacks.

This discussion assumes that the recipient's key pair is static, as is always the case in [RFC2631].

2.1 Message Sender

This section describes situations in which the message sender should be protected.

If the sender's key is ephemeral, (i.e. ephemeral-static Diffie-Hellman is being used), then no protection is necessary. In this situation only the recipients of the message can obtain the plaintext and corresponding ciphertext and therefore determine information about the private key using the "small-subgroup" attacks. However, the recipients can always decrypt the message and since the sender's key is ephemeral, even if the recipient can learn the entire private key no other messages are at risk. Notice here that if two or more recipients have selected the same domain parameters (p,q,g) then the same ephemeral public key can be used for all of them. Since the key is ephemeral and only associated with a message that the recipients can already decrypt, no interesting attacks are possible.

If the sender's key is static (i.e. static-static Diffie-Hellman is being used), then protection is necessary because in this situation a recipient mounting a small-subgroup attack may be able to obtain the plaintext from another recipient (perhaps one with a valid public key also controlled by the recipient) and therefore could obtain information about the private key. Moreover, the attacker does not need to know the plaintext to test whether a key is correct, provided that the plaintext has sufficient redundancy (e.g., ASCII). This information could then be used to attack other messages protected with the same static key.

2.2 Message Recipient

This section describes situations in which the message recipient should be protected.

If absolutely no information on the decryption of the ciphertext is available to any other party than the recipient, then protection is not necessary because this attack requires information on whether the decryption was successful to be sent to the attacker. So, no protective measures are necessary if the implementation ensures that no information about the decryption can leak out. However, protection may be warranted if human users may give this information to the sender via out of band means (e.g. through telephone conversations).

If information on the decryption is available to any other party, then protection is necessary. In particular, protection is necessary if any protocol event allows any other party to conclude that decryption was successful. Such events include replies and returning signed receipts.

3. Methods Of Protection

This section describes five protective measures that senders and recipients of messages can use to protect themselves from "small-subgroup" attacks.

Implementers should note that some of the procedures described in this section may be the subject of patents or pending patents.

3.1 Public Key Validation

This method is described in Section 2.1.5 of [RFC2631], and its description is repeated here. If this method is used, it should be used to validate public keys of the other party prior to computing the shared secret ZZ. The public key to be validated is 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.

3.2 CA Performs Public Key Validation

The Certification Authority (CA) could perform the Public Key Validation method described in Section 3.1 prior to signing and issuing a certificate containing a Diffie-Hellman public key. In this way, any party using the public key can be assured that a

trusted third party has already performed the key validation process. This method is only viable for static public keys. When Static-Static Diffie-Hellman is employed, both the sender and recipient are protected when the CA has performed public key validation. However, when Ephemeral-Static Diffie-Hellman is employed, only the sender can be protected by having the CA perform public key validation. Since the sender generates an ephemeral public key, the CA cannot perform the validation on that public key.

In the case of a static public key a method must exist to assure the user that the CA has actually performed this verification. The CA can notify certificate users that it has performed the validation by reference to the CA's Certificate Policy (CP) and Certification Practice Statement (CPS) [RFC2527] or through extensions in the certificate.

3.3 Choice of Prime p

The prime p could be chosen such that p-1=2*q*k where k is a large prime or is the product of large primes (large means greater than or equal to q). This will prevent an attacker from being able to find an element (other than 1 and p-1) of small order modulo p, thus thwarting the small-subgroup attack. One method to produce primes of this form is to run the prime generation algorithm multiple times until an appropriate prime is obtained. As an example, the value of k could be tested for primality. If k is prime, then the value of p could be accepted, otherwise the prime generation algorithm would be run again, until a value of p is produced with k prime.

However, since with primes of this form there is still an element of order 2 (i.e. p-1), one bit of the private key could still be lost. Thus, this method may not be appropriate in circumstances where the loss of a single bit of the private key is a concern.

Another method to produce primes of this form is to choose the prime p such that p = 2*q*k + 1 where k is small (i.e. only a few bits). In this case, the leakage due to a small subgroup attack will be only a few bits. Again, this would not be appropriate for circumstances where the loss of even a few bits of the private key is a concern. In this approach, q is large. Note that in DSA, q is limited to 160 bits for performance reasons, but need not be the case for Diffie-Hellman.

Additionally, other methods (i.e. public key validation) can be combined with this method in order to prevent the loss of a few bits of the private key.

3.4 Compatible Cofactor Exponentiation

This method of protection is specified in [P1363] and [KALISKI]. It involves modifying the computation of ZZ by including j (the cofactor) in the computations and is compatible with ordinary Diffie-Hellman when both parties' public keys are valid. If a party's public key is invalid, then the resulting ZZ will either be 1 or an element of order q; the small subgroup elements will either be detected or cancelled. This method requires that gcd(j,q)=1.

Instead of computing ZZ as ZZ=yb^xa mod p, Party A would compute it as ZZ=(yb^j)^c mod p where c=j^(-1)*xa mod q. (Similarly for Party B.)

If the resulting value ZZ satisfies ZZ==1, then the key agreement should be abandoned because the public key being used is invalid.

Note that when j is larger than q, as is usually the case with Diffie-Hellman, this method is less efficient than the method of Section 3.1.

3.5 Non-compatible Cofactor Exponentiation

This method of protection is specified in [P1363]. Similar to the method of Section 3.4, it involves modifying the computation of ZZ by including j (the cofactor) in the computations. If a party's public key is invalid, then the resulting ZZ will either be 1 or an element of order q; the small subgroup elements will either be detected or cancelled. This method requires that gcd(j,q)=1.

Instead of computing ZZ as ZZ=yb^xa mod p, Party A would compute it as ZZ=(yb^j)^xa mod p. (Similarly for Party B.) However, with this method the resulting ZZ value is different from what is computed in [RFC2631] and therefore is not interoperable with implementations conformant to [RFC2631].

If the resulting value ZZ satisfies ZZ==1, then the key agreement should be abandoned because the public key being used is invalid.

Note that when j is larger than q, as is usually the case with Diffie-Hellman, this method is less efficient than the method of Section 3.1.

4. Ephemeral-Ephemeral Key Agreement

This situation is when both the sender and recipient of a message are using ephemeral keys. While this situation is not possible in S/MIME, it might be used in other protocol environments. Thus we will briefly discuss protection for this case as well.

Implementers should note that some of the procedures described in this section may be the subject of patents or pending patents.

Ephemeral-ephemeral key agreement gives an attacker more flexibility since both parties' public keys can be changed and they can be coerced into computing the same key from a small space. However, in the ephemeral-static case, only the sender's public key can be changed, and only the recipient can be coerced by an outside attacker into computing a key from a small space.

Thus, in some ephemeral-ephemeral key agreements protection may be necessary for both entities. One possibility is that the attacker could modify both parties' public key so as to make their shared key predictable. For example, the attacker could replace both ya and yb with some element of small order, say -1. Then, with a certain probability, both the sender and receiver would compute the same shared value that comes from some small, easily exhaustible set.

Note that in this situation if protection was obtained from the methods of Section 3.3, then each user must ensure that the other party's public key does not come from the small set of elements of small order. This can be done either by checking a list of such elements, or by additionally applying the methods of Sections 3.1, 3.4 or 3.5.

Protection from these attacks is not necessary however if the other party's ephemeral public key has been authenticated. The authentication may be in the form of a signature, MAC, or any other integrity protection mechanism. An example of this is in the Station-To-Station protocol [STS]. Since the owner authenticates the public key, a third party cannot modify it and therefore cannot mount an attack. Thus, the only person that could attack an entity's private key is the other authenticated entity in the key agreement. However, since both public keys are ephemeral, they only protect the current session that the attacker would have access to anyway.

5. Security Considerations

This entire document addresses security considerations in the implementation of Diffie-Hellman key agreement.

6. Intellectual Property Rights

7. References

[KALISKI] B.S. Kaliski, Jr., "Compatible cofactor multiplication for Diffie-Hellman primitives", Electronics Letters, vol. 34, no. 25, December 10, 1998, pp. 2396-2397.

[LAW] 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.

[LIM] 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.

[P1363] IEEE P1363, Standard Specifications for Public Key Cryptography, 1998, work in progress.

[PH] S.C Pohlig and M.E. Hellman, "An improved algorithm for computing logarithms over GF(p) and its cryptographic significance", IEEE Transactions on Information Theory, vol. 24, 1972, pp. 106-110.

[RFC2527] Chokhani, S. and W. Ford, "Internet X.509 Public Key Infrastructure, Certificate Policy and Certification Practices Framework", RFC 2527, March 1999.

[RFC2630] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[RFC2633] Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[STS] W. Diffie, P.C. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges", Designs, Codes and Cryptography, vol. 2, 1992, pp. 107-125.

8. Author's Address

Robert Zuccherato
Entrust Technologies
750 Heron Road
Ottawa, Ontario
Canada K1V 1A7

EMail: robert.zuccherato@entrust.com

9. Full Copyright Statement

Acknowledgement

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