INTERNET DRAFT November 11, 1998
Expires in six months

Diffie-Hellman Signing Algorithm
<draft-schaad-dhsign-00.txt>

Status of this Memo

Abstract

This document describes a method for producing a signature from a Diffie-Hellman key pair. This behavior is needed for such operations as creating a signature of a PKCS #10 certification request. This document describes two different flavors of the signature algorithm, one using a D-H key from a CA and the other using a temporary key created by the signer.

1. Introduction

PKCS #10 [RFC2314] defines a syntax for certification requests. It assumes that the public key being requested for certification corresponds to an algorithm that is capable of signing/encrypting. Diffie-Hellman (DH) is a key agreement algorithm and as such cannot be directly used for signing or encryption.

This document describes two new signing algorithms using the Diffie-Hellman key agreement process to provide a shared secret as the basis of the signature. In the first signature algorithm, the signature is constructed for a specific recipient/verifier by using a public key of that verifier. In the second signature algorithm, the signature is constructed for arbitrary verifiers. This is done by creating an appropriate D-H key pair and encoding them as part of the signature value.

2. Terminology

The following definitions will be used in this document

DH certificate = a certificate whose SubjectPublicKey is a DH public value and is signed with any signature algorithm (e.g. rsa or dsa).

3. DH Signature Process

The steps for creating a DH signature are:

1. **An entity (E) chooses the group parameters for a DH key agreement. In many cases this is done simply by selecting the group parameters from** a certificate for the recipient of the signature process (static DH signatures) but they may be computed for other methods (ephemeral DH signatures).

   In the ephemeral DH signature scheme, a temporary DH key-pair is generated using the group parameters. In the static DH signature scheme, a certificate with the correct group parameters has to be available. Let these common DH parameters be g and p; and let this DH key-pair be known as the CA key pair (CApub and CApriv).

   CApub = g^x mod p   (where x=CApriv, the private DH value)

2. **The entity generates a DH public/private key-pair using the parameters from step 1.**

   For an entity E:
   Epriv = DH private value = y
   Epub = DH public value = g^y mod p

3. **The signature computation process will then consist of:**

   a) The value to be signed is obtained. (For a RFC2314 object, the value is the DER encoded certificationRequestInfo field represented as an octet string.) This will be the `text' referred to in [RFC2104], the data to which HMAC-SHA1 is applied.

   b) A shared DH secret is computed, as follows,
      shared secret = Kec = g^xy mod p

      [This is done by the entity E as g^(y.CApub) and by the CA as g^(x.Epub), where CApub is retrieved from the CA's DH certificate and Epub is retrieved from the actual certification request.]

   c) A temporary key K is derived from the shared secret Kec as follows:
      K = SHA1(LeadingInfo | Kec | TrailingInfo)
      where "|" means concatenation.

d) Compute HMAC-SHA1 over the data `text' as per [RFC2104] as:
SHA1(K XOR opad, SHA1(K XOR ipad, text))

where,
opad (outer pad) = the byte 0x36 repeated 64 times
and
ipad (inner pad) = the byte 0x5C repeated 64 times.

Namely,
(1) Append zeros to the end of K to create a 64 byte string (e.g., if K is of length 16 bytes it will be appended with 48 zero bytes 0x00).
(2) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with ipad.
(3) Append the data stream `text' to the 64 byte string resulting from step (2).
(4) Apply SHA1 to the stream generated in step (3).
(5) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with opad.
(6) Append the SHA1 result from step (4) to the 64 byte string resulting from step (5).
(7) Apply SHA1 to the stream generated in step (6) and output the result.

Sample code is also provided in [RFC2104].

e) The output of (d) is encoded as a BIT STRING (the Signature value).

The signature verification process requires the CA to carry out steps (a) through (d) and then simply compare the result of step (d) with what it received as the signature component. If they match then the following can be concluded:

1) The Entity possesses the private key corresponding to the public key in the certification request because it needed the private key to calculate the shared secret; and
2) For the static signature scheme, that only the CA, the entity sent the request to could actually verify the request because the CA would require its own private key to compute the same shared secret. This protects from rogue CAs.

4. Static DH Signature

In the static DH Signature scheme, the public key used in the key agreement process of step 2 is obtained from the entity that will be verifying the signature (i.e. the recipient). In the case of a certification request, the public key would normally be extracted from a certificate issued to the CA with the appropriate key parameters.

The values used in step 3c for "LeadingInfo" and the "TrailingInfo" are:

```
LeadingInfo ::= Subject Distinguished Name from certificate
TrailingInfo ::= Issuer Distinguished Name from certificate
```

The ASN.1 structures associated with the static Diffie-Hellman signature algorithms are:

```
id-dhSig-static-HMAC-SHA1 ::= <TBD>

DhSigStatic ::= SEQUENCE {
    issuerAndSerial      IssuerAndSerialNumber OPTIONAL,
    hashValue                 MessageDigest
}
```

issuerAndSerial is the issuer name and serial number of the certificate from which the public key was obtained. The issuerAndSerial field is omitted if the public key did not come from a certificate.

hashValue contains the result of the SHA-1 HMAC operation in step 3d.

DhSigStatic is encoded as a BIT STRING and is the signature value (i.e. encodes the above sequence instead of the raw output from 3d).

5. Ephemeral DH Signature

There are occasions when it is difficult to obtain the CA's certificate, or the group parameters of the CA are inappropriate. In these cases the following variation on what is presented in section 3 can be used.

A temporary Diffie-Hellman key pair is generated using the same group parameters as the DH key pair to be used in the signing operation. The private half of the key pair is encoded as part of the signature value and sent in the clear to be used by the signature verifier. The public half of the key pair is not transmitted, as it provides no useful information to the signature verifier. After the signature has been performed the temporary key pair is discarded.

The values used in step 3c for "LeadingInfo" and the "TrailingInfo" are:

```
LeadingInfo ::= DER encoded p
TrailingInfo ::= DER encoded g
```

The ASN.1 structures used for ephemeral DH signatures are:

```
id-dhSig-ephermal-HMAC-SHA1 ::= <TBD>

DhSigEphemeral ::== SEQUENCE {
    domainParameters     DomainParams,
    tempPrivateKey       DHPrivateKey,
    hashValue            MessageDigest
}

DomainParameters ::= SEQUENCE {
    p       INTEGER, -- odd prime, p=jq +1
    g       INTEGER, -- generator, g
    q       INTEGER, -- factor of p-1
    j       INTEGER OPTIONAL, -- subgroup factor
    validationParms  ValidationParms OPTIONAL
}

ValidationParms ::= SEQUENCE {
    seed             BIT STRING,
    pgenCounter      INTEGER
}
```

The fields of type DomainParameters have the following meanings:

p identifies the prime p defining the Galois field;

g specifies the generator of the mutiplicative subgroup of order g;

q specifies the prime factor of p-1;

j optionally specifies the value that satisfies the equation p=jq+1 to support the optional verification of group parameters;

seed optionally specifies the bit string parameter used as the seed for the system parameter generation process; and

pgenCounter optionally specifies the integer value output as part of the of the system parameter prime generation process.

If either of the parameter generation components (pgencounter or seed) is provided, the other must be present as well. The presence of q, j, seed and pgenCounter can be used to do parameter validation. The validation procedures can be found in [FIPS-186] Appendix 2.

```
DHPrivateKey ::= INTEGER
```

DhSigEphermeral is DER encoded as a BIT STRING and is the signature value.

5. 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.

Selection of parameters can be of paramount importance. In the selection of parameters once must take into account the community/group of entities that one wishes to be able to communicate with. In choosing a set of parameters one must also be sure to avoid small groups. [FIPS-186] Appendixes 2 and 3 contain information on the selection of parameters. The paractices outlined in this document will lead to better selection of parameters.

6. Open Issues

Need to add some text about the validations of parameters that MUST be done as part of the ephemeral signature validation process. (Minimum would be checks on value of $1 < x < q-2$)

7. References

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

[RFC2314] B. Kaliski, "PKCS #10: Certification Request Syntax v1.5", RFC 2314, October 1997

[RFC2104] H. Krawczyk, M. Bellare, R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

8. Author's Addresses

Hemma Prafullchandra
XETI Inc.
5150 El Camino Real, #A-32
Los Altos, CA 94022
(640) 694-6812
hemma@xeti.com

Jim Schaad
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

(425) 936-3101
jimsch@microsoft.com

Appendix A. ASN.1 Module

```
DH-Sign DEFINITIONS IMPLICIT TAGS ::=

BEGIN

--EXPORTS ALL
-- The types and values defined in this module are exported for use in
-- the other ASN.1 modules. Other applications may use them for their
-- own purposes.

IMPORTS
    IssuerAndSerialNumber, MessageDigest
     FROM CryptographicMessageSyntax { iso(1) member-body(2)
          us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16)
          modules(0) cms(1) }

     id-dhSig-static-HMAC-SHA1 ::= <TBD>

     DhSigStatic ::= SEQUENCE {
         issuerAndSerial      IssuerAndSerialNumber OPTIONAL,
         hashValue            MessageDigest
     }

     id-dhSig-ephermal-HMAC-SHA1 ::= <TBD>

     DhSigEphemeral ::== SEQUENCE {
         domainParameters     DomainParams,
         tempprivateKey       DHPrivateKey,
         hashValue            MessageDigest
     }

     DomainParameters ::= SEQUENCE {
         p       INTEGER, -- odd prime, p=jq +1
         g       INTEGER, -- generator, g
         q       INTEGER, -- factor of p-1
         j       INTEGER OPTIONAL, -- subgroup factor
         validationParms  ValidationParms OPTIONAL }

     ValidationParms ::= SEQUENCE {
         seed             BIT STRING,
         pgenCounter      INTEGER }

     DHPrivateKey ::= INTEGER
```

END