Network Working Group
Request for Comments: 2875
Category: Standards Track
H. Prafullchandra
Critical Path Inc
J. Schaad
July 2000

Diffie-Hellman Proof-of-Possession Algorithms

Status of this Memo

Copyright Notice

Abstract

This document describes two methods for producing an integrity check value from a Diffie-Hellman key pair. This behavior is needed for such operations as creating the signature of a PKCS #10 certification request. These algorithms are designed to provide a proof-of-possession rather than general purpose signing.

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 proof-of-possession algorithms using the Diffie-Hellman key agreement process to provide a shared secret as the basis of an integrity check value. In the first algorithm, the value is constructed for a specific recipient/verifier by using a public key of that verifier. In the second algorithm, the value is constructed for arbitrary verifiers.

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. Static DH Proof-of-Possession Process

The steps for creating a DH POP are:

1. An entity (E) chooses the group parameters for a DH key agreement.

   This is done simply by selecting the group parameters from a certificate for the recipient of the POP process.

   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 Recipient key pair (Rpub and Rpriv).

   ```
   Rpub = g^x mod p          (where x=Rpriv, the private DH value and
                              ^ denotes exponentiation)
   ```

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 POP 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 = ZZ = g^xy mod p
   ```

[This is done by the entity E as Rpub^y and by the Recipient as Epub^x, where Rpub is retrieved from the Recipient's DH certificate (or is the one that was locally generated by the Entity) and Epub is retrieved from the actual certification request.]

c) A temporary key K is derived from the shared secret ZZ as follows:

```
K = SHA1(LeadingInfo | ZZ | TrailingInfo),
   where "|" means concatenation.

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

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 POP verification process requires the Recipient 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:

a) 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
b) Only the Recipient that the entity sent the request to could actually verify the request because they would require their own private key to compute the same shared secret. In the case where the recipient is a Certification Authority, this protects the Entity from rogue CAs.

ASN Encoding

The ASN.1 structures associated with the static Diffie-Hellman POP algorithm are:

```
id-dhPop-static-HMAC-SHA1 OBJECT IDENTIFIER ::= { id-pkix
   id-alg(6) 3}

DhPopStatic ::= 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.

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

4. Discrete Logarithm Signature

The use of a single set of parameters for an entire public key infrastructure allows all keys in the group to be attacked together.

For this reason we need to create a proof of possession for Diffie-Hellman keys that does not require the use of a common set of parameters.

This POP is based on the Digital Signature Algorithm, but we have removed the restrictions imposed by the [FIPS-186] standard. The use of this method does impose some additional restrictions on the set of keys that may be used, however if the key generation algorithm documented in [DH-X9.42] is used the required restrictions are met. The additional restrictions are the requirement for the existence of a q parameter. Adding the q parameter is generally accepted as a good practice as it allows for checking of small group attacks.

The following definitions are used in the rest of this section:

```
p is a large prime
g = h(p-1)/q mod p ,
   where h is any integer 1 < h < p-1 such that h(p-1) mod q > 1
   (g has order q mod p)
q is a large prime
j is a large integer such that p = qj + 1

x is a randomly or pseudo-randomly generated integer with
   1 < x < q
y = g^x mod p
```

Note: These definitions match the ones in [DH-X9.42].

4.1 Expanding the Digest Value

Besides the addition of a q parameter, [FIPS-186] also imposes size restrictions on the parameters. The length of q must be 160-bits (matching output of the SHA-1 digest algorithm) and length of p must be 1024-bits. The size restriction on p is eliminated in this document, but the size restriction on q is replaced with the requirement that q must be at least 160-bits. (The size restriction on q is identical with that in [DH-X9.42].)

Given that there is not a random length-hashing algorithm, a hash value of the message will need to be derived such that the hash is in the range from 0 to q-1. If the length of q is greater than 160-bits then a method must be provided to expand the hash length.

The method for expanding the digest value used in this section does not add any additional security beyond the 160-bits provided by SHA-1. The value being signed is increased mainly to enhance the difficulty of reversing the signature process.

This algorithm produces m the value to be signed.

Let L = the size of q (i.e. 2^L <= q < 2^(L+1)). Let M be the original message to be signed.

1. Compute d = SHA-1(M), the SHA-1 digest of the original message.

2. If L == 160 then m = d.

3. If L > 160 then follow steps (a) through (d) below.

   a) Set n = L / 160, where / represents integer division, consequently, if L = 200, n = 1.
   b) Set m = d, the initial computed digest value.
   c) For i = 0 to n - 1
      m = m | SHA(m), where "|" means concatenation.
   d) m = LEFTMOST(m, L-1), where LEFTMOST returns the L-1 left most bits of m.

Thus the final result of the process meets the criteria that 0 <= m < q.

4.2 Signature Computation Algorithm

The signature algorithm produces the pair of values (r, s), which is the signature. The signature is computed as follows:

Given m, the value to be signed, as well as the parameters defined earlier in section 5.

1. Generate a random or pseudorandom integer k, such that 0 < k^-1 < q.

2. Compute r = (g^k mod p) mod q.

3. If r is zero, repeat from step 1.

4. Compute s = (k^-1 (m + xr)) mod q.

5. If s is zero, repeat from step 1.

4.3 Signature Verification Algorithm

The signature verification process is far more complicated than is normal for the Digital Signature Algorithm, as some assumptions about the validity of parameters cannot be taken for granted.

Given a message m to be validated, the signature value pair (r, s) and the parameters for the key.

1. Perform a strong verification that p is a prime number.

2. Perform a strong verification that q is a prime number.

3. Verify that q is a factor of p-1, if any of the above checks fail then the signature cannot be verified and must be considered a failure.

4. Verify that r and s are in the range [1, q-1].

5. Compute w = (s^-1) mod q.

6. Compute u1 = m*w mod q.

7. Compute u2 = r*w mod q.

8. Compute v = ((g^u1 * y^u2) mod p) mod q.

9. Compare v and r, if they are the same then the signature verified correctly.

4.4 ASN Encoding

The signature is encoded using

```
id-alg-dhPOP OBJECT IDENTIFIER ::= {id-pkix id-alg(6) 4}
```

The parameters for id-alg-dhPOP are encoded as DomainParameters (imported from [PROFILE]). The parameters may be omitted in the signature, as they must exist in the associated key request.

The signature value pair r and s are encoded using Dss-Sig-Value (imported from [PROFILE]).

5. Security Considerations

In the static DH POP algorithm, an appropriate value can be produced by either party. Thus this algorithm only provides integrity and not origination service. The Discrete Logarithm algorithm provides both integrity checking and origination checking.

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 one 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 practices outlined in this document will lead to better selection of parameters.

6. References

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

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

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

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

[DH-X9.42] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

7. Authors' Addresses

Hemma Prafullchandra
Critical Path Inc.
5150 El Camino Real, #A-32
Los Altos, CA 94022

Phone: (640) 694-6812
EMail: hemma@cp.net

Jim Schaad

EMail: jimsch@exmsft.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) }

   Dss-Sig-Value, DomainParameters
   FROM PKIX1Explicit88 {iso(1) identified-organization(3) dod(6)
        internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
        id-pkix1-explicit-88(1)};

   id-dh-sig-hmac-sha1 OBJECT IDENTIFIER ::= {id-pkix id-alg(6) 3}

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

   id-alg-dh-pop OBJECT IDENTIFIER ::= {id-pkix id-alg(6) 4}

END
```

Appendix B. Example of Static DH Proof-of-Possession

The following example follows the steps described earlier in section 3.

Step 1: Establishing common Diffie-Hellman parameters. Assume the parameters are as in the DER encoded certificate. The certificate contains a DH public key signed by a CA with a DSA signing key.

```
  0 30 939: SEQUENCE {
  4 30 872:   SEQUENCE {
  8 A0   3:     [0] {
 10 02   1:       INTEGER 2
          :       }
 13 02   6:     INTEGER
          :       00 DA 39 B6 E2 CB
 21 30  11:     SEQUENCE {
 23 06   7:       OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
 32 05   0:       NULL
          :       }
 34 30  72:     SEQUENCE {
 36 31  11:       SET {
 38 30   9:         SEQUENCE {
 40 06   3:           OBJECT IDENTIFIER countryName (2 5 4 6)
 45 13   2:           PrintableString 'US'
          :           }
          :         }
 49 31  17:       SET {
 51 30  15:         SEQUENCE {
 53 06   3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
 58 13   8:           PrintableString 'XETI Inc'
          :           }
          :         }
 68 31  16:       SET {
 70 30  14:         SEQUENCE {
 72 06   3:           OBJECT IDENTIFIER organizationalUnitName (2 5 4
11)
 77 13   7:           PrintableString 'Testing'
          :           }
          :         }
 86 31  20:       SET {
 88 30  18:         SEQUENCE {
 90 06   3:           OBJECT IDENTIFIER commonName (2 5 4 3)
 95 13  11:           PrintableString 'Root DSA CA'
          :           }
          :         }
          :       }
108 30  30:     SEQUENCE {
```

```
110 17  13:       UTCTime '990914010557Z'
125 17  13:       UTCTime '991113010557Z'
          :       }
140 30  70:     SEQUENCE {
142 31  11:       SET {
144 30   9:         SEQUENCE {
146 06   3:           OBJECT IDENTIFIER countryName (2 5 4 6)
151 13   2:           PrintableString 'US'
          :           }
          :         }
155 31  17:       SET {
157 30  15:         SEQUENCE {
159 06   3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
164 13   8:           PrintableString 'XETI Inc'
          :           }
          :         }
174 31  16:       SET {
176 30  14:         SEQUENCE {
178 06   3:           OBJECT IDENTIFIER organizationalUnitName (2 5 4
11)
183 13   7:           PrintableString 'Testing'
          :           }
          :         }
192 31  18:       SET {
194 30  16:         SEQUENCE {
196 06   3:           OBJECT IDENTIFIER commonName (2 5 4 3)
201 13   9:           PrintableString 'DH TestCA'
          :           }
          :         }
          :       }
212 30 577:     SEQUENCE {
216 30 438:       SEQUENCE {
220 06   7:         OBJECT IDENTIFIER dhPublicKey (1 2 840 10046 2 1)
229 30 425:         SEQUENCE {
233 02 129:           INTEGER
          :             00 94 84 E0 45 6C 7F 69 51 62 3E 56 80 7C 68 E7
          :             C5 A9 9E 9E 74 74 94 ED 90 8C 1D C4 E1 4A 14 82
          :             F5 D2 94 0C 19 E3 B9 10 BB 11 B9 E5 A5 FB 8E 21
          :             51 63 02 86 AA 06 B8 21 36 B6 7F 36 DF D1 D6 68
          :             5B 79 7C 1D 5A 14 75 1F 6A 93 75 93 CE BB 97 72
          :             8A F0 0F 23 9D 47 F6 D4 B3 C7 F0 F4 E6 F6 2B C2
          :             32 E1 89 67 BE 7E 06 AE F8 D0 01 6B 8B 2A F5 02
          :             D7 B6 A8 63 94 83 B0 1B 31 7D 52 1A DE E5 03 85
          :             27
365 02 128:           INTEGER
          :             26 A6 32 2C 5A 2B D4 33 2B 5C DC 06 87 53 3F 90
          :             06 61 50 38 3E D2 B9 7D 81 1C 12 10 C5 0C 53 D4
          :             64 D1 8E 30 07 08 8C DD 3F 0A 2F 2C D6 1B 7F 57
```

```
          :              86 D0 DA BB 6E 36 2A 18 E8 D3 BC 70 31 7A 48 B6
          :              4E 18 6E DD 1F 22 06 EB 3F EA D4 41 69 D9 9B DE
          :              47 95 7A 72 91 D2 09 7F 49 5C 3B 03 33 51 C8 F1
          :              39 9A FF 04 D5 6E 7E 94 3D 03 B8 F6 31 15 26 48
          :              95 A8 5C DE 47 88 B4 69 3A 00 A7 86 9E DA D1 CD
496 02   33:            INTEGER
          :              00 E8 72 FA 96 F0 11 40 F5 F2 DC FD 3B 5D 78 94
          :              B1 85 01 E5 69 37 21 F7 25 B9 BA 71 4A FC 60 30
          :              FB
531 02   97:            INTEGER
          :              00 A3 91 01 C0 A8 6E A4 4D A0 56 FC 6C FE 1F A7
          :              B0 CD 0F 94 87 0C 25 BE 97 76 8D EB E5 A4 09 5D
          :              AB 83 CD 80 0B 35 67 7F 0C 8E A7 31 98 32 85 39
          :              40 9D 11 98 D8 DE B8 7F 86 9B AF 8D 67 3D B6 76
          :              B4 61 2F 21 E1 4B 0E 68 FF 53 3E 87 DD D8 71 56
          :              68 47 DC F7 20 63 4B 3C 5F 78 71 83 E6 70 9E E2
          :              92
630 30   26:            SEQUENCE {
632 03   21:              BIT STRING 0 unused bits
          :              1C D5 3A 0D 17 82 6D 0A 81 75 81 46 10 8E 3E DB
          :              09 E4 98 34
655 02    1:              INTEGER 55
          :              }
          :            }
          :          }
658 03 132:        BIT STRING 0 unused bits
          :          02 81 80 5F CF 39 AD 62 CF 49 8E D1 CE 66 E2 B1
          :          E6 A7 01 4D 05 C2 77 C8 92 52 42 A9 05 A4 DB E0
          :          46 79 50 A3 FC 99 3D 3D A6 9B A9 AD BC 62 1C 69
          :          B7 11 A1 C0 2A F1 85 28 F7 68 FE D6 8F 31 56 22
          :          4D 0A 11 6E 72 3A 02 AF 0E 27 AA F9 ED CE 05 EF
          :          D8 59 92 C0 18 D7 69 6E BD 70 B6 21 D1 77 39 21
          :          E1 AF 7A 3A CF 20 0A B4 2C 69 5F CF 79 67 20 31
          :          4D F2 C6 ED 23 BF C4 BB 1E D1 71 40 2C 07 D6 F0
          :          8F C5 1A
          :        }
793 A3   85:      [3] {
795 30   83:        SEQUENCE {
797 30   29:          SEQUENCE {
799 06    3:            OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29
14)
804 04   22:            OCTET STRING
          :              04 14 80 DF 59 88 BF EB 17 E1 AD 5E C6 40 A3 42
          :              E5 AC D3 B4 88 78
          :            }
828 30   34:          SEQUENCE {
830 06    3:            OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29
35)
```

```
835 01    1:             BOOLEAN TRUE
838 04   24:             OCTET STRING
           :               30 16 80 14 6A 23 37 55 B9 FD 81 EA E8 4E D3 C9
           :               B7 09 E5 7B 06 E3 68 AA
           :             }
864 30   14:           SEQUENCE {
866 06    3:             OBJECT IDENTIFIER keyUsage (2 5 29 15)
871 01    1:             BOOLEAN TRUE
874 04    4:             OCTET STRING
           :               03 02 03 08
           :             }
           :           }
           :         }
           :       }
880 30   11:   SEQUENCE {
882 06    7:     OBJECT IDENTIFIER dsaWithSha1 (1 2 840 10040 4 3)
891 05    0:     NULL
           :     }
893 03   48:   BIT STRING 0 unused bits
           :     30 2D 02 14 7C 6D D2 CA 1E 32 D1 30 2E 29 66 BC
           :     06 8B 60 C7 61 16 3B CA 02 15 00 8A 18 DD C1 83
           :     58 29 A2 8A 67 64 03 92 AB 02 CE 00 B5 94 6A
           :   }
```

Step 2. End Entity/User generates a Diffie-Hellman key-pair using the parameters from the CA certificate.

EE DH public key: SunJCE Diffie-Hellman Public Key:

```
Y: 13 63 A1 85 04 8C 46 A8 88 EB F4 5E A8 93 74 AE
   FD AE 9E 96 27 12 65 C4 4C 07 06 3E 18 FE 94 B8
   A8 79 48 BD 2E 34 B6 47 CA 04 30 A1 EC 33 FD 1A
   0B 2D 9E 50 C9 78 0F AE 6A EC B5 6B 6A BE B2 5C
   DA B2 9F 78 2C B9 77 E2 79 2B 25 BF 2E 0B 59 4A
   93 4B F8 B3 EC 81 34 AE 97 47 52 E0 A8 29 98 EC
   D1 B0 CA 2B 6F 7A 8B DB 4E 8D A5 15 7E 7E AF 33
   62 09 9E 0F 11 44 8C C1 8D A2 11 9E 53 EF B2 E8
```

EE DH private key:

```
X: 32 CC BD B4 B7 7C 44 26 BB 3C 83 42 6E 7D 1B 00
   86 35 09 71 07 A0 A4 76 B8 DB 5F EC 00 CE 6F C3
```

Step 3. Compute K and the signature.

LeadingInfo: DER encoded Subject/Requestor DN (as in the generated Certificate Signing Request)

```
  30 4E 31 0B 30 09 06 03 55 04 06 13 02 55 53 31
  11 30 0F 06 03 55 04 0A 13 08 58 45 54 49 20 49
  6E 63 31 10 30 0E 06 03 55 04 0B 13 07 54 65 73
  74 69 6E 67 31 1A 30 18 06 03 55 04 03 13 11 50
  4B 49 58 20 45 78 61 6D 70 6C 65 20 55 73 65 72
```

TrailingInfo: DER encoded Issuer/Recipient DN (from the certificate described in step 1)

```
  30 46 31 0B 30 09 06 03 55 04 06 13 02 55 53 31
  11 30 0F 06 03 55 04 0A 13 08 58 45 54 49 20 49
  6E 63 31 10 30 0E 06 03 55 04 0B 13 07 54 65 73
  74 69 6E 67 31 12 30 10 06 03 55 04 03 13 09 44
  48 20 54 65 73 74 43 41
```

K:

```
  F4 D7 BB 6C C7 2D 21 7F 1C 38 F7 DA 74 2D 51 AD
  14 40 66 75
```

TBS: the Ã´textÃ¶ for computing the SHA-1 HMAC.

```
30 82 02 98 02 01 00 30 4E 31 0B 30 09 06 03 55
04 06 13 02 55 53 31 11 30 0F 06 03 55 04 0A 13
08 58 45 54 49 20 49 6E 63 31 10 30 0E 06 03 55
04 0B 13 07 54 65 73 74 69 6E 67 31 1A 30 18 06
03 55 04 03 13 11 50 4B 49 58 20 45 78 61 6D 70
6C 65 20 55 73 65 72 30 82 02 41 30 82 01 B6 06
07 2A 86 48 CE 3E 02 01 30 82 01 A9 02 81 81 00
94 84 E0 45 6C 7F 69 51 62 3E 56 80 7C 68 E7 C5
A9 9E 9E 74 74 94 ED 90 8C 1D C4 E1 4A 14 82 F5
D2 94 0C 19 E3 B9 10 BB 11 B9 E5 A5 FB 8E 21 51
63 02 86 AA 06 B8 21 36 B6 7F 36 DF D1 D6 68 5B
79 7C 1D 5A 14 75 1F 6A 93 75 93 CE BB 97 72 8A
F0 0F 23 9D 47 F6 D4 B3 C7 F0 F4 E6 F6 2B C2 32
E1 89 67 BE 7E 06 AE F8 D0 01 6B 8B 2A F5 02 D7
B6 A8 63 94 83 B0 1B 31 7D 52 1A DE E5 03 85 27
02 81 80 26 A6 32 2C 5A 2B D4 33 2B 5C DC 06 87
53 3F 90 06 61 50 38 3E D2 B9 7D 81 1C 12 10 C5
0C 53 D4 64 D1 8E 30 07 08 8C DD 3F 0A 2F 2C D6
1B 7F 57 86 D0 DA BB 6E 36 2A 18 E8 D3 BC 70 31
7A 48 B6 4E 18 6E DD 1F 22 06 EB 3F EA D4 41 69
D9 9B DE 47 95 7A 72 91 D2 09 7F 49 5C 3B 03 33
51 C8 F1 39 9A FF 04 D5 6E 7E 94 3D 03 B8 F6 31
15 26 48 95 A8 5C DE 47 88 B4 69 3A 00 A7 86 9E
DA D1 CD 02 21 00 E8 72 FA 96 F0 11 40 F5 F2 DC
FD 3B 5D 78 94 B1 85 01 E5 69 37 21 F7 25 B9 BA
71 4A FC 60 30 FB 02 61 00 A3 91 01 C0 A8 6E A4
4D A0 56 FC 6C FE 1F A7 B0 CD 0F 94 87 0C 25 BE
```

```
   97 76 8D EB E5 A4 09 5D AB 83 CD 80 0B 35 67 7F
   0C 8E A7 31 98 32 85 39 40 9D 11 98 D8 DE B8 7F
   86 9B AF 8D 67 3D B6 76 B4 61 2F 21 E1 4B 0E 68
   FF 53 3E 87 DD D8 71 56 68 47 DC F7 20 63 4B 3C
   5F 78 71 83 E6 70 9E E2 92 30 1A 03 15 00 1C D5
   3A 0D 17 82 6D 0A 81 75 81 46 10 8E 3E DB 09 E4
   98 34 02 01 37 03 81 84 00 02 81 80 13 63 A1 85
   04 8C 46 A8 88 EB F4 5E A8 93 74 AE FD AE 9E 96
   27 12 65 C4 4C 07 06 3E 18 FE 94 B8 A8 79 48 BD
   2E 34 B6 47 CA 04 30 A1 EC 33 FD 1A 0B 2D 9E 50
   C9 78 0F AE 6A EC B5 6B 6A BE B2 5C DA B2 9F 78
   2C B9 77 E2 79 2B 25 BF 2E 0B 59 4A 93 4B F8 B3
   EC 81 34 AE 97 47 52 E0 A8 29 98 EC D1 B0 CA 2B
   6F 7A 8B DB 4E 8D A5 15 7E 7E AF 33 62 09 9E 0F
   11 44 8C C1 8D A2 11 9E 53 EF B2 E8
```

Certification Request:

```
  0 30 793: SEQUENCE {
  4 30 664:   SEQUENCE {
  8 02   1:     INTEGER 0
 11 30  78:     SEQUENCE {
 13 31  11:       SET {
 15 30   9:         SEQUENCE {
 17 06   3:           OBJECT IDENTIFIER countryName (2 5 4 6)
 22 13   2:           PrintableString 'US'
          :           }
          :         }
 26 31  17:       SET {
 28 30  15:         SEQUENCE {
 30 06   3:           OBJECT IDENTIFIER organizationName (2 5 4 10)
 35 13   8:           PrintableString 'XETI Inc'
          :           }
          :         }
 45 31  16:       SET {
 47 30  14:         SEQUENCE {
 49 06   3:           OBJECT IDENTIFIER organizationalUnitName (2 5 4
11)
 54 13   7:           PrintableString 'Testing'
          :           }
          :         }
 63 31  26:       SET {
 65 30  24:         SEQUENCE {
 67 06   3:           OBJECT IDENTIFIER commonName (2 5 4 3)
 72 13  17:           PrintableString 'PKIX Example User'
          :           }
          :         }
```

```
           :        }
 91 30 577:      SEQUENCE {
 95 30 438:        SEQUENCE {
 99 06   7:          OBJECT IDENTIFIER dhPublicKey (1 2 840 10046 2 1)
108 30 425:          SEQUENCE {
112 02 129:            INTEGER
           :              00 94 84 E0 45 6C 7F 69 51 62 3E 56 80 7C 68 E7
           :              C5 A9 9E 9E 74 74 94 ED 90 8C 1D C4 E1 4A 14 82
           :              F5 D2 94 0C 19 E3 B9 10 BB 11 B9 E5 A5 FB 8E 21
           :              51 63 02 86 AA 06 B8 21 36 B6 7F 36 DF D1 D6 68
           :              5B 79 7C 1D 5A 14 75 1F 6A 93 75 93 CE BB 97 72
           :              8A F0 0F 23 9D 47 F6 D4 B3 C7 F0 F4 E6 F6 2B C2
           :              32 E1 89 67 BE 7E 06 AE F8 D0 01 6B 8B 2A F5 02
           :              D7 B6 A8 63 94 83 B0 1B 31 7D 52 1A DE E5 03 85
           :              27
244 02 128:            INTEGER
           :              26 A6 32 2C 5A 2B D4 33 2B 5C DC 06 87 53 3F 90
           :              06 61 50 38 3E D2 B9 7D 81 1C 12 10 C5 0C 53 D4
           :              64 D1 8E 30 07 08 8C DD 3F 0A 2F 2C D6 1B 7F 57
           :              86 D0 DA BB 6E 36 2A 18 E8 D3 BC 70 31 7A 48 B6
           :              4E 18 6E DD 1F 22 06 EB 3F EA D4 41 69 D9 9B DE
           :              47 95 7A 72 91 D2 09 7F 49 5C 3B 03 33 51 C8 F1
           :              39 9A FF 04 D5 6E 7E 94 3D 03 B8 F6 31 15 26 48
           :              95 A8 5C DE 47 88 B4 69 3A 00 A7 86 9E DA D1 CD
375 02  33:            INTEGER
           :              00 E8 72 FA 96 F0 11 40 F5 F2 DC FD 3B 5D 78 94
           :              B1 85 01 E5 69 37 21 F7 25 B9 BA 71 4A FC 60 30
           :              FB
410 02  97:            INTEGER
           :              00 A3 91 01 C0 A8 6E A4 4D A0 56 FC 6C FE 1F A7
           :              B0 CD 0F 94 87 0C 25 BE 97 76 8D EB E5 A4 09 5D
           :              AB 83 CD 80 0B 35 67 7F 0C 8E A7 31 98 32 85 39
           :              40 9D 11 98 D8 DE B8 7F 86 9B AF 8D 67 3D B6 76
           :              B4 61 2F 21 E1 4B 0E 68 FF 53 3E 87 DD D8 71 56
           :              68 47 DC F7 20 63 4B 3C 5F 78 71 83 E6 70 9E E2
           :              92
509 30  26:            SEQUENCE {
511 03  21:              BIT STRING 0 unused bits
           :                1C D5 3A 0D 17 82 6D 0A 81 75 81 46 10 8E 3E
DB
           :                09 E4 98 34
534 02   1:              INTEGER 55
           :              }
           :            }
           :          }
537 03 132:        BIT STRING 0 unused bits
           :          02 81 80 13 63 A1 85 04 8C 46 A8 88 EB F4 5E A8
           :          93 74 AE FD AE 9E 96 27 12 65 C4 4C 07 06 3E 18
```

```
          :         FE 94 B8 A8 79 48 BD 2E 34 B6 47 CA 04 30 A1 EC
          :         33 FD 1A 0B 2D 9E 50 C9 78 0F AE 6A EC B5 6B 6A
          :         BE B2 5C DA B2 9F 78 2C B9 77 E2 79 2B 25 BF 2E
          :         0B 59 4A 93 4B F8 B3 EC 81 34 AE 97 47 52 E0 A8
          :         29 98 EC D1 B0 CA 2B 6F 7A 8B DB 4E 8D A5 15 7E
          :         7E AF 33 62 09 9E 0F 11 44 8C C1 8D A2 11 9E 53
          :         EF B2 E8
          :       }
          :     }
672 30   12:   SEQUENCE {
674 06    8:     OBJECT IDENTIFIER dh-sig-hmac-sha1 (1 3 6 1 5 5 7 6 3)
684 05    0:     NULL
          :     }
686 03 109:   BIT STRING 0 unused bits
          :     30 6A 30 52 30 48 31 0B 30 09 06 03 55 04 06 13
          :     02 55 53 31 11 30 0F 06 03 55 04 0A 13 08 58 45
          :     54 49 20 49 6E 63 31 10 30 0E 06 03 55 04 0B 13
          :     07 54 65 73 74 69 6E 67 31 14 30 12 06 03 55 04
          :     03 13 0B 52 6F 6F 74 20 44 53 41 20 43 41 02 06
          :     00 DA 39 B6 E2 CB 04 14 1B 17 AD 4E 65 86 1A 6C
          :     7C 85 FA F7 95 DE 48 93 C5 9D C5 24
          :   }
```

Signature verification requires CAÃs private key, the CA certificate and the generated Certification Request.

CA DH private key:

```
 x:  3E 5D AD FD E5 F4 6B 1B 61 5E 18 F9 0B 84 74 a7
     52 1E D6 92 BC 34 94 56 F3 0C BE DA 67 7A DD 7D
```

Appendix C. Example of Discrete Log Signature

Step 1. Generate a Diffie-Hellman Key with length of q being 256-bits.

```
p:
  94 84 E0 45 6C 7F 69 51 62 3E 56 80 7C 68 E7 C5
  A9 9E 9E 74 74 94 ED 90 8C 1D C4 E1 4A 14 82 F5
  D2 94 0C 19 E3 B9 10 BB 11 B9 E5 A5 FB 8E 21 51
  63 02 86 AA 06 B8 21 36 B6 7F 36 DF D1 D6 68 5B
  79 7C 1D 5A 14 75 1F 6A 93 75 93 CE BB 97 72 8A
  F0 0F 23 9D 47 F6 D4 B3 C7 F0 F4 E6 F6 2B C2 32
  E1 89 67 BE 7E 06 AE F8 D0 01 6B 8B 2A F5 02 D7
  B6 A8 63 94 83 B0 1B 31 7D 52 1A DE E5 03 85 27

q:
  E8 72 FA 96 F0 11 40 F5 F2 DC FD 3B 5D 78 94 B1
  85 01 E5 69 37 21 F7 25 B9 BA 71 4A FC 60 30 FB

g:
  26 A6 32 2C 5A 2B D4 33 2B 5C DC 06 87 53 3F 90
  06 61 50 38 3E D2 B9 7D 81 1C 12 10 C5 0C 53 D4
  64 D1 8E 30 07 08 8C DD 3F 0A 2F 2C D6 1B 7F 57
  86 D0 DA BB 6E 36 2A 18 E8 D3 BC 70 31 7A 48 B6
  4E 18 6E DD 1F 22 06 EB 3F EA D4 41 69 D9 9B DE
  47 95 7A 72 91 D2 09 7F 49 5C 3B 03 33 51 C8 F1
  39 9A FF 04 D5 6E 7E 94 3D 03 B8 F6 31 15 26 48
  95 A8 5C DE 47 88 B4 69 3A 00 A7 86 9E DA D1 CD

j:
  A3 91 01 C0 A8 6E A4 4D A0 56 FC 6C FE 1F A7 B0
  CD 0F 94 87 0C 25 BE 97 76 8D EB E5 A4 09 5D AB
  83 CD 80 0B 35 67 7F 0C 8E A7 31 98 32 85 39 40
  9D 11 98 D8 DE B8 7F 86 9B AF 8D 67 3D B6 76 B4
  61 2F 21 E1 4B 0E 68 FF 53 3E 87 DD D8 71 56 68
  47 DC F7 20 63 4B 3C 5F 78 71 83 E6 70 9E E2 92

y:
  5F CF 39 AD 62 CF 49 8E D1 CE 66 E2 B1 E6 A7 01
  4D 05 C2 77 C8 92 52 42 A9 05 A4 DB E0 46 79 50
  A3 FC 99 3D 3D A6 9B A9 AD BC 62 1C 69 B7 11 A1
  C0 2A F1 85 28 F7 68 FE D6 8F 31 56 22 4D 0A 11
  6E 72 3A 02 AF 0E 27 AA F9 ED CE 05 EF D8 59 92
  C0 18 D7 69 6E BD 70 B6 21 D1 77 39 21 E1 AF 7A
  3A CF 20 0A B4 2C 69 5F CF 79 67 20 31 4D F2 C6
  ED 23 BF C4 BB 1E D1 71 40 2C 07 D6 F0 8F C5 1A

seed:
```

```
     1C D5 3A 0D 17 82 6D 0A 81 75 81 46 10 8E 3E DB
     09 E4 98 34

   C:
     00000037

   x:
     3E 5D AD FD E5 F4 6B 1B 61 5E 18 F9 0B 84 74 a7
     52 1E D6 92 BC 34 94 56 F3 0C BE DA 67 7A DD 7D
```

Step 2. Form the value to be signed and hash with SHA1. The result of the hash for this example is:

```
     5f a2 69 b6 4b 22 91 22 6f 4c fe 68 ec 2b d1 c6
     d4 21 e5 2c
```

Step 3. The hash value needs to be expanded since |q| = 256. This is done by hashing the hash with SHA1 and appending it to the original hash. The value after this step is:

```
     5f a2 69 b6 4b 22 91 22 6f 4c fe 68 ec 2b d1 c6
     d4 21 e5 2c 64 92 8b c9 5e 34 59 70 bd 62 40 ad
     6f 26 3b f7 1c a3 b2 cb
```

Next the first 255 bits of this value are taken to be the resulting "hash" value. Note in this case a shift of one bit right is done since the result is to be treated as an integer:

```
     2f d1 34 db 25 91 48 91 37 a6 7f 34 76 15 e8 e3
     6a 10 f2 96 32 49 45 e4 af 1a 2c b8 5e b1 20 56
```

Step 4. The signature value is computed. In this case you get the values

```
   R:
     A1 B5 B4 90 01 34 6B A0 31 6A 73 F5 7D F6 5C 14
     43 52 D2 10 BF 86 58 87 F7 BC 6E 5A 77 FF C3 4B

   S:
     59 40 45 BC 6F 0D DC FF 9D 55 40 1E C4 9E 51 3D
     66 EF B2 FF 06 40 9A 39 68 75 81 F7 EC 9E BE A1
```

The encoded signature values is then:

```
   30 45 02 21 00 A1 B5 B4 90 01 34 6B A0 31 6A 73
   F5 7D F6 5C 14 43 52 D2 10 BF 86 58 87 F7 BC 6E
   5A 77 FF C3 4B 02 20 59 40 45 BC 6F 0D DC FF 9D
   55 40 1E C4 9E 51 3D 66 EF B2 FF 06 40 9A 39 68
   75 81 F7 EC 9E BE A1
```

Result:

```
  30 82 02 c2 30 82 02 67 02 01 00 30 1b 31 19 30
  17 06 03 55 04 03 13 10 49 45 54 46 20 50 4b 49
  58 20 53 41 4d 50 4c 45 30 82 02 41 30 82 01 b6
  06 07 2a 86 48 ce 3e 02 01 30 82 01 a9 02 81 81
  00 94 84 e0 45 6c 7f 69 51 62 3e 56 80 7c 68 e7
  c5 a9 9e 9e 74 74 94 ed 90 8c 1d c4 e1 4a 14 82
  f5 d2 94 0c 19 e3 b9 10 bb 11 b9 e5 a5 fb 8e 21
  51 63 02 86 aa 06 b8 21 36 b6 7f 36 df d1 d6 68
  5b 79 7c 1d 5a 14 75 1f 6a 93 75 93 ce bb 97 72
  8a f0 0f 23 9d 47 f6 d4 b3 c7 f0 f4 e6 f6 2b c2
  32 e1 89 67 be 7e 06 ae f8 d0 01 6b 8b 2a f5 02
  d7 b6 a8 63 94 83 b0 1b 31 7d 52 1a de e5 03 85
  27 02 81 80 26 a6 32 2c 5a 2b d4 33 2b 5c dc 06
  87 53 3f 90 06 61 50 38 3e d2 b9 7d 81 1c 12 10
  c5 0c 53 d4 64 d1 8e 30 07 08 8c dd 3f 0a 2f 2c
  d6 1b 7f 57 86 d0 da bb 6e 36 2a 18 e8 d3 bc 70
  31 7a 48 b6 4e 18 6e dd 1f 22 06 eb 3f ea d4 41
  69 d9 9b de 47 95 7a 72 91 d2 09 7f 49 5c 3b 03
  33 51 c8 f1 39 9a ff 04 d5 6e 7e 94 3d 03 b8 f6
  31 15 26 48 95 a8 5c de 47 88 b4 69 3a 00 a7 86
  9e da d1 cd 02 21 00 e8 72 fa 96 f0 11 40 f5 f2
  dc fd 3b 5d 78 94 b1 85 01 e5 69 37 21 f7 25 b9
  ba 71 4a fc 60 30 fb 02 61 00 a3 91 01 c0 a8 6e
  a4 4d a0 56 fc 6c fe 1f a7 b0 cd 0f 94 87 0c 25
  be 97 76 8d eb e5 a4 09 5d ab 83 cd 80 0b 35 67
  7f 0c 8e a7 31 98 32 85 39 40 9d 11 98 d8 de b8
  7f 86 9b af 8d 67 3d b6 76 b4 61 2f 21 e1 4b 0e
  68 ff 53 3e 87 dd d8 71 56 68 47 dc f7 20 63 4b
  3c 5f 78 71 83 e6 70 9e e2 92 30 1a 03 15 00 1c
  d5 3a 0d 17 82 6d 0a 81 75 81 46 10 8e 3e db 09
  e4 98 34 02 01 37 03 81 84 00 02 81 80 5f cf 39
  ad 62 cf 49 8e d1 ce 66 e2 b1 e6 a7 01 4d 05 c2
  77 c8 92 52 42 a9 05 a4 db e0 46 79 50 a3 fc 99
  3d 3d a6 9b a9 ad bc 62 1c 69 b7 11 a1 c0 2a f1
  85 28 f7 68 fe d6 8f 31 56 22 4d 0a 11 6e 72 3a
  02 af 0e 27 aa f9 ed ce 05 ef d8 59 92 c0 18 d7
  69 6e bd 70 b6 21 d1 77 39 21 e1 af 7a 3a cf 20
  0a b4 2c 69 5f cf 79 67 20 31 4d f2 c6 ed 23 bf
  c4 bb 1e d1 71 40 2c 07 d6 f0 8f c5 1a a0 00 30
  0c 06 08 2b 06 01 05 05 07 06 04 05 00 03 47 00
  30 44 02 20 54 d9 43 8d 0f 9d 42 03 d6 09 aa a1
  9a 3c 17 09 ae bd ee b3 d1 a0 00 db 7d 8c b8 e4
  56 e6 57 7b 02 20 44 89 b1 04 f5 40 2b 5f e7 9c
  f9 a4 97 50 0d ad c3 7a a4 2b b2 2d 5d 79 fb 38
  8a b4 df bb 88 bc
```

Decoded Version of result:

```
  0 30  707: SEQUENCE {
  4 30  615:   SEQUENCE {
  8 02    1:     INTEGER 0
 11 30   27:     SEQUENCE {
 13 31   25:       SET {
 15 30   23:         SEQUENCE {
 17 06    3:           OBJECT IDENTIFIER commonName (2 5 4 3)
 22 13   16:           PrintableString 'IETF PKIX SAMPLE'
           :           }
           :         }
           :       }
 40 30  577:     SEQUENCE {
 44 30  438:       SEQUENCE {
 48 06    7:         OBJECT IDENTIFIER dhPublicNumber (1 2 840 10046 2
1)
 57 30  425:         SEQUENCE {
 61 02  129:           INTEGER
           :            00 94 84 E0 45 6C 7F 69 51 62 3E 56 80 7C 68 E7
           :            C5 A9 9E 9E 74 74 94 ED 90 8C 1D C4 E1 4A 14 82
           :            F5 D2 94 0C 19 E3 B9 10 BB 11 B9 E5 A5 FB 8E 21
           :            51 63 02 86 AA 06 B8 21 36 B6 7F 36 DF D1 D6 68
           :            5B 79 7C 1D 5A 14 75 1F 6A 93 75 93 CE BB 97 72
           :            8A F0 0F 23 9D 47 F6 D4 B3 C7 F0 F4 E6 F6 2B C2
           :            32 E1 89 67 BE 7E 06 AE F8 D0 01 6B 8B 2A F5 02
           :            D7 B6 A8 63 94 83 B0 1B 31 7D 52 1A DE E5 03 85
           :            27
193 02  128:           INTEGER
           :            26 A6 32 2C 5A 2B D4 33 2B 5C DC 06 87 53 3F 90
           :            06 61 50 38 3E D2 B9 7D 81 1C 12 10 C5 0C 53 D4
           :            64 D1 8E 30 07 08 8C DD 3F 0A 2F 2C D6 1B 7F 57
           :            86 D0 DA BB 6E 36 2A 18 E8 D3 BC 70 31 7A 48 B6
           :            4E 18 6E DD 1F 22 06 EB 3F EA D4 41 69 D9 9B DE
           :            47 95 7A 72 91 D2 09 7F 49 5C 3B 03 33 51 C8 F1
           :            39 9A FF 04 D5 6E 7E 94 3D 03 B8 F6 31 15 26 48
           :            95 A8 5C DE 47 88 B4 69 3A 00 A7 86 9E DA D1 CD
324 02   33:           INTEGER
           :            00 E8 72 FA 96 F0 11 40 F5 F2 DC FD 3B 5D 78 94
           :            B1 85 01 E5 69 37 21 F7 25 B9 BA 71 4A FC 60 30
           :            FB
359 02   97:           INTEGER
           :            00 A3 91 01 C0 A8 6E A4 4D A0 56 FC 6C FE 1F A7
           :            B0 CD 0F 94 87 0C 25 BE 97 76 8D EB E5 A4 09 5D
           :            AB 83 CD 80 0B 35 67 7F 0C 8E A7 31 98 32 85 39
           :            40 9D 11 98 D8 DE B8 7F 86 9B AF 8D 67 3D B6 76
           :            B4 61 2F 21 E1 4B 0E 68 FF 53 3E 87 DD D8 71 56
           :            68 47 DC F7 20 63 4B 3C 5F 78 71 83 E6 70 9E E2
```

```
          :             92
458 30   26:            SEQUENCE {
460 03   21:              BIT STRING 0 unused bits
          :             1C D5 3A 0D 17 82 6D 0A 81 75 81 46 10 8E 3E DB
          :             09 E4 98 34
483 02    1:              INTEGER 55
          :              }
          :            }
          :          }
486 03  132:        BIT STRING 0 unused bits
          :          02 81 80 5F CF 39 AD 62 CF 49 8E D1 CE 66 E2 B1
          :          E6 A7 01 4D 05 C2 77 C8 92 52 42 A9 05 A4 DB E0
          :          46 79 50 A3 FC 99 3D 3D A6 9B A9 AD BC 62 1C 69
          :          B7 11 A1 C0 2A F1 85 28 F7 68 FE D6 8F 31 56 22
          :          4D 0A 11 6E 72 3A 02 AF 0E 27 AA F9 ED CE 05 EF
          :          D8 59 92 C0 18 D7 69 6E BD 70 B6 21 D1 77 39 21
          :          E1 AF 7A 3A CF 20 0A B4 2C 69 5F CF 79 67 20 31
          :          4D F2 C6 ED 23 BF C4 BB 1E D1 71 40 2C 07 D6 F0
          :          8F C5 1A
          :        }
621 A0    0:      [0]
          :      }
623 30   12:    SEQUENCE {
625 06    8:      OBJECT IDENTIFIER '1 3 6 1 5 5 7 6 4'
635 05    0:      NULL
          :      }
637 03   72:    BIT STRING 0 unused bits
          :      30 45 02 21 00 A1 B5 B4 90 01 34 6B A0 31 6A 73
          :      F5 7D F6 5C 14 43 52 D2 10 BF 86 58 87 F7 BC 6E
          :      5A 77 FF C3 4B 02 20 59 40 45 BC 6F 0D DC FF 9D
          :      55 40 1E C4 9E 51 3D 66 EF B2 FF 06 40 9A 39 68
          :      75 81 F7 EC 9E BE A1
          :    }
```

Full Copyright Statement

Acknowledgement

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