

PKIX  
Internet-Draft  
Obsoletes: [2875](#) (if approved)  
Intended status: Standards Track  
Expires: September 28, 2013

J. Schaad  
Soaring Hawk Consulting  
H. Prafullchandra  
Hy-Trust  
March 27, 2013

Diffie-Hellman Proof-of-Possession Algorithms  
draft-schaad-pkix-rfc2875-bis-08

## Abstract

This document describes two methods for producing an integrity check value from a Diffie-Hellman key pair and one method for producing an integrity check value from an Elliptic Curve 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 of the private key and not to be a general purpose signing algorithm.

This document obsoletes [RFC 2875](#).

## Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 28, 2013.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

Internet-Draft

DH POP Algorithms

March 2013

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

<a href="#">1.</a>	Introduction . . . . .	<a href="#">2</a>
<a href="#">1.1.</a>	Changes since <a href="#">RFC2875</a> . . . . .	<a href="#">4</a>
<a href="#">1.2.</a>	Requirements Terminology . . . . .	<a href="#">4</a>
<a href="#">2.</a>	Terminology . . . . .	<a href="#">4</a>
<a href="#">3.</a>	Notation . . . . .	<a href="#">5</a>
<a href="#">4.</a>	Static DH Proof-of-Possession Process . . . . .	<a href="#">5</a>
<a href="#">4.1.</a>	ASN.1 Encoding . . . . .	<a href="#">7</a>
<a href="#">5.</a>	Discrete Logarithm Signature . . . . .	<a href="#">10</a>
<a href="#">5.1.</a>	Expanding the Digest Value . . . . .	<a href="#">11</a>
<a href="#">5.2.</a>	Signature Computation Algorithm . . . . .	<a href="#">12</a>
<a href="#">5.3.</a>	Signature Verification Algorithm . . . . .	<a href="#">12</a>
<a href="#">5.4.</a>	ASN.1 Encoding . . . . .	<a href="#">13</a>
<a href="#">6.</a>	Static ECDH Proof-of-Possession Process . . . . .	<a href="#">15</a>
<a href="#">6.1.</a>	ASN.1 Encoding . . . . .	<a href="#">17</a>
<a href="#">7.</a>	Security Considerations . . . . .	<a href="#">19</a>
<a href="#">8.</a>	IANA Considerations . . . . .	<a href="#">19</a>
<a href="#">9.</a>	References . . . . .	<a href="#">19</a>
<a href="#">9.1.</a>	Normative References . . . . .	<a href="#">19</a>
<a href="#">9.2.</a>	Informative References . . . . .	<a href="#">20</a>
<a href="#">Appendix A.</a>	ASN.1 Modules . . . . .	<a href="#">20</a>
<a href="#">A.1.</a>	2008 ASN.1 Module . . . . .	<a href="#">21</a>
<a href="#">A.2.</a>	1988 ASN.1 Module . . . . .	<a href="#">25</a>
<a href="#">Appendix B.</a>	Example of Static DH Proof-of-Possession . . . . .	<a href="#">27</a>

<a href="#">Appendix C. Example of Discrete Log Signature</a>	<a href="#">35</a>
Authors' Addresses	<a href="#">40</a>

## [1.](#) Introduction

Schaad & PrafullchandraExpires September 28, 2013

[Page 2]

Internet-Draft

DH POP Algorithms

March 2013

Among the responsibilities of a Certificate Authority in issuing certificates is a requirement that it verifies the identity for the entity to which it is issuing a certificate and that it verifies that the private key for the public key to be placed in the certificate is in the possession of that entity. The process of validating that the private key is held by the requester of the certificate is called Proof-of-Possession(POP). Further details on why POP is important can be found in [Appendix C of RFC 4211](#) [[CRMF](#)].

This document is designed to deal with the problem of how to support POP for encryption-only keys. PKCS #10 [[RFC2986](#)] and the Certificate Request Message Format (CRMF) [[CRMF](#)] both define syntaxes for certification requests. However, while CRMF supports an alternative method to support POP for encryption-only keys, PKCS #10 does not. PKCS #10 assumes that the public key being requested for certification corresponds to an algorithm that is capable of producing a POP by a signature operation. Diffie-Hellman (DH) and Elliptic Curve Diffie-Hellman (ECDH) are key agreement algorithms and, as such, cannot be directly used for signing or encryption.

This document describes a set of three proof-of-possession algorithms. Two methods use the key agreement process (one for Diffie-Hellman and one for Elliptic-Curve DH) to provide a shared secret as the basis of an integrity check value. For these methods, the value is constructed for a specific recipient/verifier by using a public key of that verifier. The third method uses a modified signature algorithm (for Diffie-Hellman). This method allows for arbitrary verifiers.

It should be noted that we did not create an algorithm that parallels ECDSA (Elliptical Curve Digital Signature Algorithm) as was done for DSA (Digital Signature Algorithm). When using ECDH, the common practice is to use one of a set of predefined curves, each of these curves has been designed to be paired with one of the commonly used hash algorithm. This differs in practice from the Diffie-Hellman case where the common practice is to generate a set of group

parameters either on a single machine or for a given community and are aligned to encryption algorithms rather than hash algorithms. The implication is that, if a key has the ability to perform the modified DSA algorithm for ECDSA, it should be able to use the correct hash algorithm and perform the regular ECDSA signature algorithm with the correctly sized hash.

### 1.1. Changes since [RFC2875](#)

The following changes have been made:

- o The Static DH Proof-of-Possession algorithm has been re-written for parameterization of the hash algorithm and the message authentication code (MAC) algorithm.
- o New instances of the static DH POP algorithm have been created using HMAC paired with the SHA-224, SHA-256, SHA-384 and SHA-512 hash algorithms. However the current SHA-1 algorithm remains identical.
- o The Discrete Logarithm Signature algorithm has been re-written for parameterization of the hash algorithm.
- o New instances of the Discrete Logarithm Signature have been created for the SHA-224, SHA-256, SHA-384, and SHA-512 hash functions. However the current SHA-1 algorithm remains identical.
- o A new Static ECDH Proof-of-Possession algorithm has been added.
- o New instances of the Static ECDH POP algorithm has been created using HMAC paired with the SHA-224, SHA-256, SHA-384, and SHA-512 hash functions.

### 1.2. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",

"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)].

When the words are in lower case they have their natural language meaning.

## 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).

ECDH certificate = a certificate whose SubjectPublicKey is an ECDH public value and is signed with any signature algorithm (e.g., RSA or ECDSA).

Proof-of-Possession (POP) is a means that provides a method for a second party to perform an algorithm to establish with some degree of

assurance that the first party does possess and has the ability to use a private key. The reasoning behind doing POP can be found in [Appendix C](#) in [[CRME](#)].

## 3. Notation

This section describes mathematical notations, conventions and symbols used throughout this document.

$a \parallel b$	: Concatenation of a and b
$a^b$	: a raised to the power of b
$a \bmod b$	: a modulo b
$a / b$	: a divided by b using integer division
$a * b$	: a times b depending on context multiplication may be within an Elliptic Curve or normal multiplication
$KDF(a)$	: Key Derivation Function producing a value from a.
$MAC(a, b)$	: Message Authentication Code function where a is the key and b is the text
$LEFTMOST(a, b)$	: Return the b left most bits of a
$FLOOR(a)$	: Return n where n is the largest integer such that

$$n \leq a$$

Details on how to implement the HMAC version of a MAC function used in this document can be found in [RFC 2104](#) [[RFC2104](#)], [RFC 6234](#) [[RFC6234](#)] and [RFC 4231](#) [[RFC4231](#)].

#### 4. Static DH Proof-of-Possession Process

The Static DH POP algorithm is set up to use a key derivation function (KDF) and a message authentication code (MAC). This algorithm requires that a common set of group parameters be used by both the creator and verifier of the POP value.

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 the common DH parameters be  $g$  and  $p$ ; and let the DH key-pair from the certificate be known as the Recipient key pair ( $R_{pub}$  and  $R_{priv}$ ).

$R_{pub} = g^x \bmod p$  (where  $x=R_{priv}$ , the private DH value)

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

For an entity  $E$ :

$E_{priv} = \text{DH private value} = y$

$E_{pub} = \text{DH public value} = g^y \bmod p$

3. The POP computation process will then consist of:

- a) The value to be signed (text) is obtained. (For a PKCS #10 object, the value is the DER encoded certificationRequestInfo field represented as an octet string.)
- b) A shared DH secret is computed, as follows,  
  

$$\text{shared secret} = ZZ = g^{(x*y)} \bmod p$$

[This is done by the entity E as  $R_{\text{pub}}^y$  and by the Recipient as  $E_{\text{pub}}^x$ , where  $R_{\text{pub}}$  is retrieved from the Recipient's DH certificate (or is provided in the protocol) and  $E_{\text{pub}}$  is retrieved from the certification request.]
- c) A temporary key K is derived from the shared secret ZZ as follows:  
  

$$K = \text{KDF}(\text{LeadingInfo} \mid ZZ \mid \text{TrailingInfo})$$

LeadingInfo ::= Subject Distinguished Name from recipient's certificate

TrailingInfo ::= Issuer Distinguished Name from recipient's certificate
- d) Using the defined MAC function, compute  $\text{MAC}(K, \text{text})$ .

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 it would require its 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.

#### 4.1. ASN.1 Encoding

The algorithm outlined above allows for the use of an arbitrary hash function in computing the temporary key and the MAC algorithm. In this specification we define object identifiers for the SHA-1, SHA-256, SHA-384 and SHA-512 hash values and use HMAC for the MAC algorithm. The ASN.1 structures associated with the static Diffie-Hellman POP algorithm are:

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

sa-dhPop-static-sha1-hmac-sha1 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-dhPop-static-sha1-hmac-sha1
    VALUE      DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-dh }
}

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

id-dhPop-static-sha1-hmac-sha1 OBJECT IDENTIFIER ::=
    id-dh-sig-hmac-sha1

sa-dhPop-static-sha224-hmac-sha224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-static-sha224-hmac-sha224
    VALUE      DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-static-sha224-hmac-sha224 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 15
}
```

```
sa-dhPop-static-sha256-hmac-sha256 SIGNATURE-ALGORITHM ::= {
```



```

        IDENTIFIER id-alg-dhPop-static-sha256-hmac-sha256
        VALUE DhSigStatic
        PARAMS ARE absent
        PUBLIC-KEYS { pk-dh }
    }

    id-alg-dhPop-static-sha256-hmac-sha256 OBJECT IDENTIFIER ::= {
        id-pkix id-alg(6) 16
    }

    sa-dhPop-static-sha384-hmac-sha384 SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-alg-dhPop-static-sha384-hmac-sha384
        VALUE DhSigStatic
        PARAMS ARE absent
        PUBLIC-KEYS { pk-dh }
    }

    id-alg-dhPop-static-sha384-hmac-sha384 OBJECT IDENTIFIER ::= {
        id-pkix id-alg(6) 17
    }

    sa-dhPop-static-sha512-hmac-sha512 SIGNATURE-ALGORITHM ::= {
        IDENTIFIER id-alg-dhPop-static-sha512-hmac-sha512
        VALUE DhSigStatic
        PARAMS ARE absent
        PUBLIC-KEYS { pk-dh }
    }

    id-alg-dhPop-static-sha512-hmac-sha512 OBJECT IDENTIFIER ::= {
        id-pkix id-alg(6) 18
    }

```

In the above ASN.1 the following items are defined:

#### DhSigStatic

This ASN.1 type structure holds the information describing the signature. The structure has the following fields:

#### issuerAndSerial

This field contains 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

This field contains the result of the MAC operation in step 3d.

sa-dhPop-static-sha1-hmac-sha1

An ASN.1 SIGNATURE-ALGORITHM object which associates together the information describing a signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-dhPop-static-sha1-hmac-sha1

This OID identifies the Static DH POP algorithm that uses SHA-1 as the KDF and HMAC-SHA1 as the MAC function. The new OID was created for naming consistency with the other OIDs defined here. The value of the OID is the same value as id-dh-sig-hmac-sha1 which was defined in the previous version of this document [[RFC2875](#)].

sa-dhPop-static-sha224-hmac-sha224

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-dhPop-static-sha224-hmac-sha224

This OID identifies the Static DH POP algorithm that uses SHA-224 as the KDF and HMAC-SHA224 as the MAC function.

sa-dhPop-static-sha256-hmac-sha256

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-dhPop-static-sha256-hmac-sha256

This OID identifies the Static DH POP algorithm that uses SHA-256 as the KDF and HMAC-SHA256 as the MAC function.

sa-dhPop-static-sha384-hmac-sha384

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-dhPop-static-sha384-hmac-sha384

This OID identifies the Static DH POP algorithm that uses SHA-384

as the KDF and HMAC-SHA384 as the MAC function.

sa-dhPop-static-sha512-hmac-sha512

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-dhPop-static-sha512-hmac-sha512

This OID identifies the Static DH POP algorithm that uses SHA-512 as the KDF and HMAC-SHA512 as the MAC function.

## [5.](#) Discrete Logarithm Signature

When a single set of parameters is used for a large group of keys, the chances that a collision will occur in the set of keys either by accident or design increases as the number of keys used increases. A large number of keys from a single parameter set also encourages the use of brute force methods of attack as the entire set of keys in the parameters can be attacked in a single operation rather than having to attack each key parameter set individually.

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 dealing with the hash and key sizes 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 [\[RFC2631\]](#) 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 subgroup attacks.

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

$p$  is a large prime

$g = h^{((p-1)/q)} \bmod p$  ,

where  $h$  is any integer  $1 < h < p-1$  such that  $h^{((p-1)/q)} \bmod p > 1$

(g has order  $q \bmod p$ )  
q is a large prime  
j is a large integer such that  $p = q*j + 1$   
x is a randomly or pseudo-randomly generated integer with  $1 < x < q$   
 $y = g^x \bmod p$   
HASH is a hash function such that  
b = the output size of HASH in bits

Note: These definitions match the ones in [\[RFC2631\]](#).

### [5.1.](#) 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 the output length of the SHA-1 digest algorithm) and the 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 b bits in length. (If the hash function is SHA-1, then  $b=160$  bits and the size restriction on b is identical with that in [\[FIPS-186\]](#).)

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 b then a method must be provided to expand the hash.

The method for expanding the digest value used in this section does not add any additional security beyond the b bits provided by the hash algorithm. For this reason the hash algorithm should be the largest size possible to match q. 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.,  $2^L \leq q < 2^{(L+1)}$ ).  
Let M be the original message to be signed.  
Let b be the length of HASH output

1. Compute  $d = \text{HASH}(M)$ , the digest of the original message.
2. If  $L == b$  then  $m = d$ .

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

a) Set  $n = \text{FLOOR}(L / b)$

b) Set  $m = d$ , the initial computed digest value.

c) For  $i = 0$  to  $n - 1$   
     $m = m \parallel \text{HASH}(m)$

d)  $m = \text{LEFTMOST}(m, L-1)$

Thus the final result of the process meets the criteria that  $0 \leq m < q$ .

### [5.2.](#) 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  $r = (g^k \bmod p) \bmod q$ .
3. If  $r$  is zero, repeat from step 1.
4. Compute  $s = ((k^{-1}) * (m + x*r)) \bmod q$ .
5. If  $s$  is zero, repeat from step 1.

### [5.3.](#) 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 value  $m$  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  $w = (s^{-1}) \bmod q$ .
6. Compute  $u_1 = m \cdot w \bmod q$ .
7. Compute  $u_2 = r \cdot w \bmod q$ .
8. Compute  $v = ((g^{u_1} * y^{u_2}) \bmod p) \bmod q$ .
9. Compare  $v$  and  $r$ , if they are the same then the signature verified correctly.

#### [5.4.](#) ASN.1 Encoding

The signature algorithm is parameterized by the hash algorithm. The ASN.1 structures associated with the Discrete Logarithm Signature algorithm are:

```
sa-dhPop-SHA1 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-alg-dh-pop
  VALUE DSA-Sig-Value
  PARAMS TYPE DomainParameters ARE preferredAbsent
  HASHES { mda-sha1 }
  PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-sha1 OBJECT IDENTIFIER ::= id-alg-dh-pop

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

```

sa-dhPop-sha224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha224
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha224 }
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-sha224 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 5
}

sa-dhPop-sha256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha256
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha256 }
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-sha256 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 6
}

sa-dhPop-sha384 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha384
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha384 }
    PUBLIC-KEYS { pk-dh }
}

```

```

}

id-alg-dhPop-sha384 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 7
}

sa-dhPop-sha512 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha512
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha512 }
}

```

```

    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-sha512 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 8
}

```

In the above ASN.1 the following items are defined:

#### sa-dhPop-sha1

A SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DSA-Sig-Value represents the signature value and the parameters DomainParameters SHOULD be omitted in the signature, but MUST be present in the associated key request.

#### id-alg-dhPop-sha1

This OID identifies the discrete logarithm signature using SHA-1 as the hash algorithm. The new OID was created for naming consistency with the others defined here. The value of the OID is the same as id-alg-dh-pop which was defined in the previous version of this document [[RFC2875](#)].

#### sa-dhPop-sha224

A SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DSA-Sig-Value represents the signature value and the parameters DomainParameters SHOULD be omitted in the signature, but MUST be present in the associated key request.

#### id-alg-dhPop-sha224

This OID identifies the discrete logarithm signature using SHA-224 as the hash algorithm.

#### sa-dhPop-sha256

A SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DSA-Sig-Value represents the signature value and the parameters DomainParameters SHOULD be omitted in the signature, but MUST be



present in the associated key request.

**id-alg-dhPop-sha256**

This OID identifies the discrete logarithm signature using SHA-256 as the hash algorithm.

**sa-dhPop-sha384**

A SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DSA-Sig-Value represents the signature value and the parameters DomainParameters SHOULD be omitted in the signature, but MUST be present in the associated key request.

**id-alg-dhPop-sha384**

This OID identifies the discrete logarithm signature using SHA-384 as the hash algorithm.

**sa-dhPop-sha512**

A SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DSA-Sig-Value represents the signature value and the parameters DomainParameters SHOULD be omitted in the signature, but MUST be present in the associated key request.

**id-alg-dhPop-sha512**

This OID identifies the discrete logarithm signature using SHA-512 as the hash algorithm.

## **6. Static ECDH Proof-of-Possession Process**

The Static ECDH POP algorithm is set up to use a key derivation function (KDF) and a message authentication code (MAC). This algorithm requires that a common set of group parameters be used by both the creator and verifier of the POP value. Full details of how Elliptic Curve Cryptography works can be found in [RFC 6090](#) [[RFC6090](#)].

The steps for creating an ECDH POP are:

1. An entity (E) chooses the group parameters for an ECDH 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.

The ECDH parameters can be identified either by a named group or by a set of curve parameters. [Section 2.3.5 of RFC 3279 \[RFC3279\]](#) documents how the parameters are encoded for PKIX certificates. For PKIX-based applications, the parameters will almost always be defined by a named group. Designate  $G$  as the group from the ECDH parameters. Let the ECDH key-pair associated with the certificate be known as the Recipient key pair ( $R_{pub}$  and  $R_{priv}$ ).

$$R_{pub} = R_{priv} * G$$

2. The entity generates an ECDH public/private key-pair using the parameters from step 1.

For an entity  $E$ :

$$\begin{aligned} E_{priv} &= \text{Entity private value} \\ E_{pub} &= \text{ECDH public point} = E_{priv} * G \end{aligned}$$

3. The POP computation process will then consist of:
  - a) The value to be signed (text) is obtained. (For a PKCS #10 object, the value is the DER encoded certificationRequestInfo field represented as an octet string.)

- b) A shared ECDH secret is computed, as follows,

$$\text{shared secret point } (x, y) = E_{priv} * R_{pub} = R_{priv} * E_{pub}$$

shared secret value  $ZZ$  is the  $x$  coordinate of the computed point

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

$$K = \text{KDF}(\text{LeadingInfo} \mid ZZ \mid \text{TrailingInfo})$$

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

- d) Compute  $\text{MAC}(K, \text{text})$ .

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:

Internet-Draft

DH POP Algorithms

March 2013

- 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 it would require its 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.

#### [6.1.](#) ASN.1 Encoding

The algorithm outlined above allows for the use of an arbitrary hash function in computing the temporary key and the MAC value. In this specification we defined object identifiers for the SHA-1 and SHA-256 hash values. The ASN.1 structures associated with the static ECDH POP algorithm are:

```
id-alg-ecdhPop-static-sha224-hmac-sha224 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 25
}

sa-ecdhPop-sha224-hmac-sha224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha224-hmac-sha224
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-ec }
}

id-alg-ecdhPop-static-sha256-hmac-sha256 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 26
}

sa-ecdhPop-sha256-hmac-sha256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha256-hmac-sha256
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-ec }
}
```

```
id-alg-ecdhPop-static-sha384-hmac-sha384 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 27
}
```

```
sa-ecdhPop-sha384-hmac-sha384 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha384-hmac-sha384
    VALUE DhSigStatic
    PARAMS ARE absent
}
```

```
    PUBLIC-KEYS { pk-ec }
}
```

```
id-alg-ecdhPop-static-sha512-hmac-sha512 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 28
}
```

```
sa-ecdhPop-sha512-hmac-sha512 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha512-hmac-sha512
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-ec }
}
```

In the above ASN.1 the following items are defined:

sa-ecdhPop-static-sha224-hmac-sha224

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-ecdhPop-static-sha224-hmac-sha224

This OID identifies the Static ECDH POP algorithm that uses SHA-224 as the KDF and HMAC-SHA224 as the MAC function.

sa-ecdhPop-static-sha256-hmac-sha256

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-ecdhPop-static-sha256-hmac-sha256

This OID identifies the Static ECDH POP algorithm that uses SHA-256 as the KDF and HMAC-SHA256 as the MAC function.

sa-ecdhPop-static-sha384-hmac-sha384

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-ecdhPop-static-sha384-hmac-sha384

This OID identifies the Static ECDH POP algorithm that uses SHA-384 as the KDF and HMAC-SHA384 as the MAC function.

sa-ecdhPop-static-sha512-hmac-sha512

An ASN.1 SIGNATURE-ALGORITHM object that associates together the information describing this signature algorithm. The structure DhSigStatic represents the signature value and the parameters MUST be absent.

id-ecdhPop-static-sha512-hmac-sha512

This OID identifies the Static ECDH POP algorithm that uses SHA-512 as the KDF and HMAC-SHA512 as the MAC function.

## [7.](#) Security Considerations

None of the algorithms defined in this document are meant for use in general purpose situations. These algorithms are designed and purposed solely for use in doing Proof-of-Possession with PKCS#10 and CRMF constructs.

In the static DH POP and static ECDH POP algorithms, an appropriate value can be produced by either party. Thus these algorithms only provide 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 for DH. [RFC6090] Section 10 contains information on the selection of parameter for ECC. The practices outlined in these documents will lead to better selection of parameters.

## 8. IANA Considerations

This document contains no IANA considerations.

## 9. References

### 9.1. Normative References

- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), February 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", [RFC 2631](#), June 1999.
- [RFC2986] Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", [RFC 2986](#), November 2000.
- [RFC4231] Nystrom, M., "Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512", [RFC 4231](#), December 2005.
- [RFC6234] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms

(SHA and SHA-based HMAC and HKDF)", [RFC 6234](#), May 2011.

## [9.2.](#) Informative References

- [CRMF]        Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", [RFC 4211](#), September 2005.
- [FIPS-186]        , "Digital Signature Standard", Federal Information Processing Standards Publication 186, May 1994.
- [RFC2875]        Prafullchandra, H. and J. Schaad, "Diffie-Hellman Proof-of-Possession Algorithms", [RFC 2875](#), July 2000.
- [RFC3279]        Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 3279](#), April 2002.
- [RFC5912]        Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", [RFC 5912](#), June 2010.
- [RFC6090]        McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", [RFC 6090](#), February 2011.

## [Appendix A.](#) ASN.1 Modules

Schaad & Prafullchandra Expires September 28, 2013

[Page 20]

---

Internet-Draft

DH POP Algorithms

March 2013

### [A.1.](#) 2008 ASN.1 Module

This appendix contains an ASN.1 module which is conformant with the 2008 version of ASN.1. This module references the object classes defined by [\[RFC5912\]](#) to more completely describe all of the associations between the elements defined in this document. Where a difference exists between the module in this section and the 1988 module, the 2008 module is the definitive module.

DH-Sign

```
{ iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0)
```

```

        id-mod-dhSign-2012-08(80) }
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
    SIGNATURE-ALGORITHM
    FROM AlgorithmInformation-2009
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-algorithmInformation-02(58) }

    IssuerAndSerialNumber, MessageDigest
    FROM CryptographicMessageSyntax-2010
        { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
          pkcs-9(9) smime(16) modules(0) id-mod-cms-2009(58) }

    DSA-Sig-Value, DomainParameters, ECDSA-Sig-Value,
    mda-sha1, mda-sha224, mda-sha256, mda-sha384, mda-sha512,
    pk-dh, pk-ec
    FROM PKIXAlgs-2009
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-pkix1-algorithms2008-02(56) }

    id-pkix
    FROM PKIX1Explicit-2009
        { iso(1) identified-organization(3) dod(6) internet(1)
          security(5) mechanisms(5) pkix(7) id-mod(0)
          id-mod-pkix1-explicit-02(51) };

    DhSigStatic ::= SEQUENCE {

```

```

        issuerAndSerial IssuerAndSerialNumber OPTIONAL,
        hashValue      MessageDigest
    }

```

```

sa-dhPop-static-sha1-hmac-sha1 SIGNATURE-ALGORITHM ::= {

```



```

    IDENTIFIER id-dhPop-static-sha1-hmac-sha1
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-dh }
}

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

id-dhPop-static-sha1-hmac-sha1 OBJECT IDENTIFIER ::=
    id-dh-sig-hmac-sha1

sa-dhPop-static-sha224-hmac-sha224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-static-sha224-hmac-sha224
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-static-sha224-hmac-sha224 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 15
}

sa-dhPop-static-sha256-hmac-sha256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-static-sha256-hmac-sha256
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-static-sha256-hmac-sha256 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 16
}

sa-dhPop-static-sha384-hmac-sha384 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-static-sha384-hmac-sha384
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-static-sha384-hmac-sha384 OBJECT IDENTIFIER ::= {

```

```

        id-pkix id-alg(6) 17
    }

sa-dhPop-static-sha512-hmac-sha512 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-static-sha512-hmac-sha512
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-static-sha512-hmac-sha512 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 18
}

```

```

sa-dhPop-SHA1 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dh-pop
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha1 }
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-sha1 OBJECT IDENTIFIER ::= id-alg-dh-pop

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

```

```

sa-dhPop-sha224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha224
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha224 }
    PUBLIC-KEYS { pk-dh }
}

id-alg-dhPop-sha224 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 5
}

```

```

sa-dhPop-sha256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha256
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha256 }
    PUBLIC-KEYS { pk-dh }
}

```

Internet-Draft

DH POP Algorithms

March 2013

```
id-alg-dhPop-sha256 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 6
}
```

```
sa-dhPop-sha384 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha384
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha384 }
    PUBLIC-KEYS { pk-dh }
}
```

```
id-alg-dhPop-sha384 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 7
}
```

```
sa-dhPop-sha512 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-dhPop-sha512
    VALUE DSA-Sig-Value
    PARAMS TYPE DomainParameters ARE preferredAbsent
    HASHES { mda-sha512 }
    PUBLIC-KEYS { pk-dh }
}
```

```
id-alg-dhPop-sha512 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 8
}
```

```
id-alg-ecdhPop-static-sha224-hmac-sha224 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 25
}
```

```
sa-ecdhPop-sha224-hmac-sha224 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha224-hmac-sha224
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-ec }
}
```

```
id-alg-ecdhPop-static-sha256-hmac-sha256 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 26
}
```

```

sa-ecdhPop-sha256-hmac-sha256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha256-hmac-sha256
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-ec }

```

```

}

id-alg-ecdhPop-static-sha384-hmac-sha384 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 27
}

sa-ecdhPop-sha384-hmac-sha384 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha384-hmac-sha384
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-ec }
}

id-alg-ecdhPop-static-sha512-hmac-sha512 OBJECT IDENTIFIER ::= {
    id-pkix id-alg(6) 28
}

sa-ecdhPop-sha512-hmac-sha512 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-alg-ecdhPop-static-sha512-hmac-sha512
    VALUE DhSigStatic
    PARAMS ARE absent
    PUBLIC-KEYS { pk-ec }
}

```

END

## [A.2.](#) 1988 ASN.1 Module

This appendix contains an ASN.1 module which is conformant with the 1988 version of ASN.1 represents an informational version of the ASN.1 module for this document. Where a difference exists between the module in this section and the 2008 module, the 2008 module is the definitive module.

```

DH-Sign
{ iso(1) identified-organization(3) dod(6) internet(1)
  security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-dhSign-2012-88(79) }
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 CryptographicMessageSyntax2004
    { iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)
      pkcs-9(9) smime(16) modules(0) cms-2004(24) }

  id-pkix
  FROM PKIX1Explicit88
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-pkix1-explicit(18) }

  Dss-Sig-Value, DomainParameters
  FROM PKIX1Algorithms88
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-algorithms(17) };

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 }

id-dhPop-static-sha1-hmac-sha1 OBJECT IDENTIFIER ::=

```

id-dh-sig-hmac-sha1

id-alg-dhPop-static-sha224-hmac-sha224 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 15 }

id-alg-dhPop-static-sha256-hmac-sha256 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 16 }

id-alg-dhPop-static-sha384-hmac-sha384 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 17 }

id-alg-dhPop-static-sha512-hmac-sha512 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 18 }

id-alg-dhPop-sha1 OBJECT IDENTIFIER ::= id-alg-dh-pop

id-alg-dhPop-sha224 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 5 }

id-alg-dhPop-sha256 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 6 }

id-alg-dhPop-sha384 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 7 }

id-alg-dhPop-sha512 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 8 }

id-alg-ecdhPop-static-sha224-hmac-sha224 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 25 }

id-alg-ecdhPop-static-sha256-hmac-sha256 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 26 }

id-alg-ecdhPop-static-sha384-hmac-sha384 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 27 }

id-alg-ecdhPop-static-sha512-hmac-sha512 OBJECT IDENTIFIER ::= {  
id-pkix id-alg(6) 28 }

END

## [Appendix B](#). Example of Static DH Proof-of-Possession

The following example follows the steps described earlier in [section 4](#).

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
:      }

```

Internet-Draft

DH POP Algorithms

March 2013

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

EE DH 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 the shared secret ZZ

```
56 b6 01 39 42 8e 09 16 30 b0 31 4d 12 90 af 03
c7 92 65 c2 9c ba 88 bb 0a d5 94 02 ed 6f 54 cb
22 e5 94 b4 d6 60 72 bc f6 a5 2b 18 8d df 28 72
ac e0 41 dd 3b 03 2a 12 9e 5d bd 72 a0 1e fb 6b
ee c5 b2 16 59 ee 12 00 3b c8 e0 cb c5 08 8e 2d
40 5f 2d 37 62 8c 4f bb 49 76 69 3c 9e fc 2c f7
f9 50 c1 b9 f7 01 32 4c 96 b9 c3 56 c0 2c 1b 77
3f 2f 36 e8 22 c8 2e 07 76 d0 4f 7f aa d5 c0 59
```

Step 4. Compute K and the signature.

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

```
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
```

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

Schaad & PrafullchandraExpires September 28, 2013

[Page 31]

Internet-Draft

DH POP Algorithms

March 2013

```
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
```

K:

```
B1 91 D7 DB 4F C5 EF EF AC 9A C5 44 5A 6D 42 28
DC 70 7B DA
```

TBS: the "text" 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 2D 05 77 FE 5E 8F 65 F5
:        AF AD C9 5C 9B 02 C0 A8 88 29 61 63
:      }

```

Signature verification requires CA's 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:

```
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 value 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 {

Internet-Draft

DH POP Algorithms

March 2013

```
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
:      }

```

#### Authors' Addresses

Jim Schaad  
Soaring Hawk Consulting

Email: [ietf@augustcellars.com](mailto:ietf@augustcellars.com)

Hemma Prafullchandra  
Hy-Trust

