Internet Research Task Force
Internet-Draft
Intended status: Informational
Expires: July 28, 2018

D. Harkins
HP Enterprise

January 24, 2018

Public Key Exchange
draft-harkins-pkex-05

Abstract

This memo describes a password-authenticated protocol to allow two devices to exchange "raw" (uncertified) public keys and establish trust that the keys belong to their respective identities.

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 July 28, 2018.

Copyright Notice

Copyright (c) 2018 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 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.

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
1.2. Notation 3
2. Properties 4
3. Assumptions 5
4. Cryptographic Primitives 6
5. Protocol Definition 6
5.1. Authentication Phase 7
5.2. Reveal Phase 8
6. IANA Considerations 9
7. Security Considerations 10
8. Acknowledgements 10
9. Changes/Author Notes 11
10. References 11
10.1. Normative References 11
10.2. Informative References 12
Appendix A. Role-specific Elements 13
A.1. ECC Role-specific Elements 14
A.1.1. Role-specific Elements for NIST p256 14
A.1.2. Role-specific Elements for NIST p384 14
A.1.3. Role-specific Elements for NIST p521 15
A.1.4. Role-specific Elements for brainpool p256r1 17
A.1.5. Role-specific Elements for brainpool p384r1 17
A.1.6. Role-specific Elements for brainpool p512r1 18
A.2. FFC Role-specific Elements 19
A.2.1. Role-specific Elements for 2048-bit FFC group 19
A.2.2. Role-specific Elements for 3072-bit FFC group 21
A.2.3. Role-specific Elements for 4096-bit FFC group 23
A.2.4. Role-specific Elements for 8192-bit FFC group 26
Author's Address 31

1. Introduction

Many authenticated key exchange protocols allow for authentication using uncertified, or "raw", public keys. Usually these specifications-- e.g. [RFC7250] for TLS and [RFC7670] for IKEv2-- assume keys are exchanged in some out-of-band mechanism.

[RFC7250] further states that "the main security challenge [to using 'raw' public keys] is how to associate the public key with a specific entity. Without a secure binding between identifier and key, the protocol will be vulnerable to man-in-the- middle attacks."

The Public Key Exchange (PKEX) is designed to fill that gap: it establishs a secure binding between exchanged public keys and identifiers, it provides proof-of-possession of the exchanged public

keys to each peer, and it enables the establishment of trust in public keys that can subsequently be used to faccilitate authentication in other authentication and key exchange protocols. At the end of a successful run of PKEX the two peers will have trust in each others exchanged public keys and also share an authenticated symmetric key which may be discarded or used for another purpose.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

1.2. Notation

This memo describes a cryptographic exchange using sets of elements called groups. Groups can be based on elliptic curves (hereafter, ECC groups) or on the multiplicative group of the field of integers modulo an odd prime (hereafter FFC groups). The public keys exchanged by PKEX are elements in a group. Elements in groups are denoted in upper-case and scalar values are denoted with lower-case. There is a distinguised generator of the group, denoted G. The order of the sub-group formed by G is q which itself must be a large prime.

When both the initator and responder use a similar, but unique, datum it is denoted by appending an "i" for initiator or "r" for responder, e.g. if each side needs an element C then the initiator's is Ci and the responder's is Cr.

During the exchange, one side will generate data and the other side will attempt to reconstruct it. The reconstructed data is "primed". That is, if the initiator generates C then when responder tries to reconstruct it, the responder will refer to it as C'. Data that is directly sent and received is not primed.

The following notation is used in this memo:

C = A + B
: The "group operation" on two elements, A and B, that produces a third element, C. For finite field cryptography this is the modular multiplication, for elliptic curve cryptography this is point addition.

C = A - B
: The "group operation" on element A and the inverse of element B to produce a third element, C. Inversion is defined such that the group operation on an element and its inverse results in the

identity element, the value one (1) for finite field cryptography and the "point at infinity" for elliptic curve cryptography.

C = a * B
This denotes repeated application of the group operation to B-- i.e. B + B-- (a - 1) times.

a = H(b)
A cryptographic hash function that takes data b of indeterminate length and returns a fixed sized digest a.

a = F(B)
A mapping function that takes an element and returns a scalar. For elliptic curve cryptography, F() returns the x-coordinate of the point B. For finite field cryptography, F() is the identity function.

a = KDF-b(c, d)
A key derivation function that derives an output key a of length b from an input key c and context d.

a = HMAC(b, c)
A keyed MAC function that produces a digest a using key b and text c.

a | b
Concatentation of data a with data b.

{a}b[c]
Authenticated-encryption of data (a), with a key (b), and associated data (c) that is authenticated but not encrypted. The result of this is a ciphertext that includes authentication information dependent on both a and c, whether c is actually transmitted or is somehow reconstructed. The receipent can decrypt a if it knows b and neither {a}b nor c have been altered during transmission, otherwise an error will be flagged.

2. Properties

Subversion of PKEX involves an adversary being able to insert its own public key into the exchange without the exchange failing, resulting in one of the parties to the exchange believing the adversary's public key actually belongs to the protocol peer.

PKEX has the following properties:

- An adversary is unable to subvert the exchange without knowing the password.

- o An adversary is unable to discover the password through passive attack.

- o The only information exposed by an active attack is whether a single guess of the password is correct or not.

- o Proof-of-possession of the private key is provided.

- o At the end of the protocol, either trust is established in the peer's public key and the public key is bound to the peer's identity, or the exchange fails.

3. Assumptions

Due to the nature of the exchange, only DSA ([DSS]) and ECDSA ([X9.62]) keys can be exchanged with PKEX.

PKEX requires fixed elements that are unique to the particular role in the protocol, an initiator-specific element and a responder-specific element. They need not be secret. It is assumed that both parties know the role-specific elements for the particular group in which their key pairs were derived. Techniques to generate role-specific elements, and generated elements for popular groups, are listed in Appendix A.1 and Appendix A.2.

The generator used in PKEX SHALL be obtained from the domain parameter set defining the group, for example [DSS] for the NIST elliptic curves and [RFC5639] for brainpool curves.

The authenticated-encryption algorithm provides deterministic "key wrapping". To achieve this the AE scheme used in PKEX is AES-SIV as defined in [RFC5297].

The KDF provides for the generation of a cryptographically strong secret key from an "imperfect" source of randomness. To achieve this the KDF used in PKEX is the unsalted version of [RFC5869].

The keyed MAC function is HMAC per [RFC2104].

The following assumptions are made on PKEX:

- o Only the peers involved in the exchange know the password.

- o The peers' public keys are from the same group.

- o The discrete logarithms of the public role-specific elements are unknown, and determining them is computationally infeasible.

4. Cryptographic Primitives

HKDF and HMAC require an underlying hash function and AES-SIV requires a key length. To provide for consistent security the hash algorithm and key length depend on the group chosen to use with PKEX.

For ECC, the hash algorithm and key length depends on the size of the prime defining the curve, p:

- SHA-256 and 256 bits: when len(p) <= 256

- SHA-384 and 384 bits: when 256 < len(p) <= 384

- SHA-512 and 512 bits: when 384 < len(p)

For FFC, the hash algorithm depends on the prime, p, defining the finite field:

- SHA-256 and 256 bits: when len(p) <= 2048

- SHA-384 and 384 bits: when 2048 < len(p) <= 3072

- SHA-512 and 512 bits: when 3072 < len(p)

5. Protocol Definition

PKEX is a balanced PAKE. The identical version of the password is used by both parties.

PKEX consists of two phases: authentication and reveal. It is described using the popular protocol participants, Alice (an initiator of PKEX), and Bob (a responder of PKEX).

We denote Alice's role-specific element a Pi and Bob's as Pr. The password is pw. For simplicity, Alice's identity is "Alice" and Bob's identity is "Bob". Alice's public key she wants to share with Bob is A and her private key is a, while Bob's public key he wants to share with Alice is B and his private key is b.

While both Alice and Bob expose their identities to passive eavesdroppers, the public keys they exchange (and ultimately gain trust in) are not. Once PKEX has finished, Alice and Bob can identifiy each other using their trusted public keys and thereby provide a level of anonymity to subsequent communications.

Implementations SHALL maintain a counter of unsuccessful exchanges for each password in order to defend against repeated active attacks to determine the password. This counter SHALL be set to zero when a

password is provisioned and incremented each time PKEX finishes unsuccessfully for that password. When the counter reaches a value of five (5) the password SHALL be irretrievably removed from the implementation.

5.1. Authentication Phase

The Authenticaiton phase is essentially the SPAKE2 key exchange. The peers derive ephemeral public keys, encrypt, and exchange them. Each party hashes a concatentation of his or her identity and the password and operates on the role-specific element to obtain a secret encrypting element. The group operation is then performed with the ephemeral key and the secret encrypting element to produce an encrypted ephmeral key. The ephemeral private keys MUST be generated with high quality (pseudo-)randomness and SHALL never be re-used.

```
      Alice:                                Bob:
      ------                                ----
  x, X = x*G                            y, Y = y*G
  Qa = H(Alice|pw)*Pi                   Qb = H(Bob|pw)*Pr
  M = X + Qa
                     Alice, M ------>
                                        Qa = H(Alice|pw)*Pi
                                        X' = M - Qa
                                        N = Y + Qb
                                        z = KDF-n(F(y*X'),
                                                Alice | Bob |
                                                F(M) | F(N) | pw)
                            <------ Bob, N
  Qb = H(Bob|pw)*Pr
  Y' = N - Qb
  z = KDF-n(F(x*Y'),
            Alice | Bob |
            F(M) | F(N) | pw)
```

where n is the key length and KDF uses the hash algorithm from Section 4.

Both M and N MUST be verified to be valid elements in the selected group. For ECC groups this means they MUST be valid points on the curve, for FFC groups they MUST be between one and the prime minus one, and the group operation on the element and the order of the sub-group, q, MUST equal one. If either element is not valid the protocol fails.

At this point the peers have exchanged ephemeral elements that will be unknown except by someone with knowledge of the password. Given

our assumptions that means only Alice and Bob can know the elements X and Y, and the secret key, z.

The secret encrypting elements Qa and Qb SHALL be irretrievably deleted at this point. The password MAY be irretrievably deleted at this time.

5.2. Reveal Phase

In the Reveal phase the peers commit to the particular public key they wish to exchange and reveal it to the peer. Proof-of-possession of the private key is accomplished by "signing" the public key, the identity to which the public key is bound, the receipient's ephemeral public key, and the sender's ephemeral public key.

The messages exchanged in the Reveal phase are encrypted and authenticated with AES-SIV using a key derived from the SPAKE2 key exchange in Section 5.1. Successful construction and validation of these messages authenticates the SPAKE2 exchange by proving possession of the SPAKE2 shared secret and therefore knowledge of the password. A single octet of the value zero (0) is used as associated data when encrypting Alice's message to Bob and a single octet of the value one (1) is used as associated data when constructing Bob's response. The associated data is not transferred as part of the either message.

The received public keys MUST be verified to be valid elements in the selected group using the same technique as above: for ECC groups they MUST be valid points on the curve, for FFC groups they MUST be between one and the prime minus one, and the group operation on the element and the order of the group, q, MUST equal one. If a received public key is not valid the protocol fails.

```
        Alice:                              Bob:
        ------                              ----
    u = HMAC(F(a*Y'), Alice | F(A) |
             F(Y') | F(X))

                     {A, u}z[0] ------>

                                   if (SIV-decrypt returns fail) fail
                                   if (A not valid element) fail
                                   u' = HMAC(F(y*A), Alice | F(A) |
                                             F(Y) | F(X'))
                                   if (u' != u) fail
                                   v = HMAC(F(b*X'), Bob | F(B) |
                                            F(X') | F(Y))

                          <------ {B, v}z[1]

    if (SIV-decrypt returns fail) fail
    if (B not valid element) fail
    v' = HMAC(F(x*B), Bob | F(B) |
              F(X) | F(Y))
    if (v'!= v) fail
```

where 0 and 1 are single octets of the value zero and one, respectively, and HMAC uses the hash algorithm from Section 4.

If the parties didn't fail they have each other's public key, knowledge that the peer possesses the corresponding private key, and trust that the public key belongs to the peer's identity that was authenticated in the Authentication Phase.

If the parties fail, the counter that protects against active attack (see Section 5) SHALL be incremented. If the value of the counter is five (5) the password SHALL be irretrievably deleted.

All ephemeral state created during the PKEX exchange SHALL be irretrievably deleted at this point. Once PKEX successfully completes the password MAY be deleted (or even exposed, with no loss of security). The authenticated and secret symmetric key, z, MAY be used for further key derivation with a different context but if not it SHOULD be irretrievably deleted.

6. IANA Considerations

This memo could create a registry of the fixed public elements for a nice cross section of popular groups. Or not. Once published this document will be a stable reference and a registry might not be needed.

7. Security Considerations

The encrypted shares exchanged in the Authentication phase MUST be ephemeral. Reuse of these keys, even with a different password, voids the security of the exchange.

If fixed elements other than those in Appendix A.1 and Appendix A.2 are used, their discrete logarithm MUST not be known. Knowledge of of the discrete logarithm of either of the fixed elements voids the security of the exchange.

The public keys exchanged in PKEX are never disclosed to an attacker, either passive or active. While they are, as the name implies, public, PKEX provides for secrecy of the exchanged keys for any protocol that might need such a capability.

PKEX has forward secrecy in the sense that exposure of the password used in a previous run of the protocol will not affect the security of that run. This also means that once PKEX has finished, the password can be exposed to a third party with out loss of security-- the public keys exchanged are still trusted and still bound to the entities that performed the exchange originally.

The Authentication Phase of PKEX is SPAKE2. The SPAKE2 security proof guarantees that if both sides bind the same password to each other's identity they will derive the same secret. This means that the public key sent in the Reveal phase is guaranteed to be sent by the identified peer-- it is sent in a message that is integrity protected and encrypted by a key, z, derived from the SPAKE2 shared secret. This binds the peer's public key to its authenticated identity. Proof-of-possession of the private key is provided by also sending a digest keyed by the result of a function of the private key and the peer's ephemeral share from the Authentication Phase. Since the sender is not able to predict what random ephemeral share will be received in the Authentication Phase, it is unable to generate a keyed digest without knowing the private analog to the public key it is sending.

There is no proof of security of PKEX at this time.

8. Acknowledgements

The author wishes to thank Liliya Ruslanovna Ahmetzyanova, Stanislav Smyshlyaev, and Greg Rose for their detailed reviews, helpful comments, and patience in answering questions.

9. Changes/Author Notes

[RFC Editor: Please remove this section before publication]

00-04

Initial version recorded is -04

04-05

Accepted comments from Liliya Ruslanovna Ahmetzyanova and Stanislav Smyshlyaev.

Accepted comments from Greg Rose

Indicated G is from group definitions, added normative reference to brainpool curves.

Added a counter to deal with repeated active attack.

Mention in introduction that the result of PKEX is trust in the public key and an authenticated symmetric key

Make group description more formal and accurate.

Add some assumptions to the notational definition for authenticated encryption.

Send identities during auth phase instead of asssuming identities are somehow learned a priori. Add mention that public key is not exposed to passive attackers.

Note that ephemeral private keys must be generated with high quality randomness and never be reused.

Define what it means to verify M and N for both FFC and ECC.

Fix the technique used to generate the fixed elements for both FFC and ECC.

10. References

10.1. Normative References

[DSS] U.S. Department of Commerce/National Institute of Standards and Technology, "Digital Signature Standard (DSS)", Federal Information Processing Standards FIPS PUB 186-4, July 2013.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

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

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, DOI 10.17487/RFC3526, May 2003, <http://www.rfc-editor.org/info/rfc3526>.

[RFC5297] Harkins, D., "Synthetic Initialization Vector (SIV) Authenticated Encryption Using the Advanced Encryption Standard (AES)", RFC 5297, DOI 10.17487/RFC5297, October 2008, <http://www.rfc-editor.org/info/rfc5297>.

[RFC5639] Lochter, M. and J. Merkle, "Elliptic Curve Cryptography (ECC) Brainpool Standard Curves and Curve Generation", RFC 5639, DOI 10.17487/RFC5639, March 2010, <https://www.rfc-editor.org/info/rfc5639>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[X9.62] American National Standards Institute, "X9.62-2005", Public Key Cryptography for the Financial Services Industry (ECDSA), 2005.

10.2. Informative References

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <http://www.rfc-editor.org/info/rfc7250>.

[RFC7670] Kivinen, T., Wouters, P., and H. Tschofenig, "Generic Raw Public-Key Support for IKEv2", RFC 7670, DOI 10.17487/RFC7670, January 2016, <http://www.rfc-editor.org/info/rfc7670>.

Appendix A. Role-specific Elements

Role-specific elements for six popular elliptic curves and four popular modp groups from [RFC3526] have been generated using the following technique which guarantees that their discrete logarithm will be unknown.

A loop is performed to generate role-specific elements by generating a candidate, testing the candidate, and exiting the loop once the test succeeds. A single octet counter is incremented each time through the loop (first time through the loop, the counter is one).

To find a candidate, a hash of an identifier (the concatenation of the ASN.1 of the OID of the curve or the name of the FFC group), a constant string, and the counter is produced. If the length of the hash's digest is less than the desired bits, the digest is pre-pended to the inputs and the result is fed back into the hash (this time it is a hash of a concatentation of the old digest, identifier, constant string, counter) to produce the next length-of-digest bits. This is repeated until the number of bits has been produced. Excess octets are stripped off. The resulting string is interpreted as an integer with the first octet of the (first) hash being the high-order octet of the integer. If the prime defining the group (the modulo of all operations in an FFC group or the prime defining the curve for ECC groups) is not an integral number of octets, the bitstring is right-shifted, pre-pending with zero bits, in order to make a big-endian bitstring of the appropriate length. If that resulting big-endian number is larger than the prime defining group, the counter is incremented and the loop continues. Otherwise, the integer is checked to see whether it is in the correct sub-group. This process is different for ECC and FFC.

For ECC, the integer is treated as an x-coordinate and checked whether it produces a valid point on the curve. If a solution to the equation of the curve does not exist for that x-coordinate, the counter is incremented and a new candidate integer is calculated. If a solution to the equation of the curve exists for that x-coordinate, the polarity of the counter is used to select a y-coordinate-- if the counter is odd then use y-p, if the counter is even use y. This point is in the correct sub-group and looping terminates.

For FFC, the co-factor is calculated as (p-1)/q, where p is the prime modulus and q is the order of the sub-group. The integer is taken to the power of the co-factor modulo p. If the result is equal to one, the counter is increased and a new candidate integer is calculated. If the result is not equal to one then the result is in the correct sub-group and it becomes the element; looping terminates.

The hash algorithm used to generate candidates is determined using the criteria in Section 4.

The loop is performed twice for each elliptic curve and FFC group to produce initiator- and responder-specific elements. The string passed for the initiator-specific element is "PKEX Initiator", the string passed for the responder-specific element is "PKEX Responder".

For FFC groups, the identifier is "group X" (including the space character and excluding the quotation marks) where X is the id assigned to the group, e.g. the 2048-bit group is named "group 14". For ECC groups, the identifier is the DER-encoded ASN.1 representation of the OID of the curve.

A.1. ECC Role-specific Elements

A.1.1. Role-specific Elements for NIST p256

```
unsigned char nist_p256_initiator_x_coord[32] = {
0x56, 0x26, 0x12, 0xcf, 0x36, 0x48, 0xfe, 0x0b,
0x07, 0x04, 0xbb, 0x12, 0x22, 0x50, 0xb2, 0x54,
0xb1, 0x94, 0x64, 0x7e, 0x54, 0xce, 0x08, 0x07,
0x2e, 0xec, 0xca, 0x74, 0x5b, 0x61, 0x2d, 0x25
};
unsigned char nist_p256_initiator_y_coord[32] = {
0x3e, 0x44, 0xc7, 0xc9, 0x8c, 0x1c, 0xa1, 0x0b,
0x20, 0x09, 0x93, 0xb2, 0xfd, 0xe5, 0x69, 0xdc,
0x75, 0xbc, 0xad, 0x33, 0xc1, 0xe7, 0xc6, 0x45,
0x4d, 0x10, 0x1e, 0x6a, 0x3d, 0x84, 0x3c, 0xa4
};
unsigned char nist_p256_responder_x_coord[32] = {
0x1e, 0xa4, 0x8a, 0xb1, 0xa4, 0xe8, 0x42, 0x39,
0xad, 0x73, 0x07, 0xf2, 0x34, 0xdf, 0x57, 0x4f,
0xc0, 0x9d, 0x54, 0xbe, 0x36, 0x1b, 0x31, 0x0f,
0x59, 0x91, 0x52, 0x33, 0xac, 0x19, 0x9d, 0x76
};
unsigned char nist_p256_responder_y_coord[32] = {
0xd9, 0xfb, 0xf6, 0xb9, 0xf5, 0xfa, 0xdf, 0x19,
0x58, 0xd8, 0x3e, 0xc9, 0x89, 0x7a, 0x35, 0xc1,
0xbd, 0xe9, 0x0b, 0x77, 0x7a, 0xcb, 0x91, 0x2a,
0xe8, 0x21, 0x3f, 0x47, 0x52, 0x02, 0x4d, 0x67
};
```

A.1.2. Role-specific Elements for NIST p384

```
unsigned char nist_p384_initiator_x_coord[48] = {
0x95, 0x3f, 0x42, 0x9e, 0x50, 0x7f, 0xf9, 0xaa,
0xac, 0x1a, 0xf2, 0x85, 0x2e, 0x64, 0x91, 0x68,
0x64, 0xc4, 0x3c, 0xb7, 0x5c, 0xf8, 0xc9, 0x53,
0x6e, 0x58, 0x4c, 0x7f, 0xc4, 0x64, 0x61, 0xac,
0x51, 0x8a, 0x6f, 0xfe, 0xab, 0x74, 0xe6, 0x12,
0x81, 0xac, 0x38, 0x5d, 0x41, 0xe6, 0xb9, 0xa3
};
unsigned char nist_p384_initiator_y_coord[48] = {
0x76, 0x2f, 0x68, 0x84, 0xa6, 0xb0, 0x59, 0x29,
0x83, 0xa2, 0x6c, 0xa4, 0x6c, 0x3b, 0xf8, 0x56,
0x76, 0x11, 0x2a, 0x32, 0x90, 0xbd, 0x07, 0xc7,
0x37, 0x39, 0x9d, 0xdb, 0x96, 0xf3, 0x2b, 0xb6,
0x27, 0xbb, 0x29, 0x3c, 0x17, 0x33, 0x9d, 0x94,
0xc3, 0xda, 0xac, 0x46, 0xb0, 0x8e, 0x07, 0x18
};
unsigned char nist_p384_responder_x_coord[48] = {
0xad, 0xbe, 0xd7, 0x1d, 0x3a, 0x71, 0x64, 0x98,
0x5f, 0xb4, 0xd6, 0x4b, 0x50, 0xd0, 0x84, 0x97,
0x4b, 0x7e, 0x57, 0x70, 0xd2, 0xd9, 0xf4, 0x92,
0x2a, 0x3f, 0xce, 0x99, 0xc5, 0x77, 0x33, 0x44,
0x14, 0x56, 0x92, 0xcb, 0xae, 0x46, 0x64, 0xdf,
0xe0, 0xbb, 0xd7, 0xb1, 0x29, 0x20, 0x72, 0xdf
};
unsigned char nist_p384_responder_y_coord[48] = {
0xab, 0xa7, 0xdf, 0x52, 0xaa, 0xe2, 0x35, 0x0c,
0xe3, 0x75, 0x32, 0xe6, 0xbf, 0x06, 0xc8, 0x7c,
0x38, 0x29, 0x4c, 0xec, 0x82, 0xac, 0xd7, 0xa3,
0x09, 0xd2, 0x0e, 0x22, 0x5a, 0x74, 0x52, 0xa1,
0x7e, 0x54, 0x4e, 0xfe, 0xc6, 0x29, 0x33, 0x63,
0x15, 0xe1, 0x7b, 0xe3, 0x40, 0x1c, 0xca, 0x06
};
```

A.1.3. Role-specific Elements for NIST p521

```
unsigned char nist_p521_initiator_x_coord[66] = {
0x00, 0x16, 0x20, 0x45, 0x19, 0x50, 0x95, 0x23,
0x0d, 0x24, 0xbe, 0x00, 0x87, 0xdc, 0xfa, 0xf0,
0x58, 0x9a, 0x01, 0x60, 0x07, 0x7a, 0xca, 0x76,
0x01, 0xab, 0x2d, 0x5a, 0x46, 0xcd, 0x2c, 0xb5,
0x11, 0x9a, 0xff, 0xaa, 0x48, 0x04, 0x91, 0x38,
0xcf, 0x86, 0xfc, 0xa4, 0xa5, 0x0f, 0x47, 0x01,
0x80, 0x1b, 0x30, 0xa3, 0xae, 0xe8, 0x1c, 0x2e,
0xea, 0xcc, 0xf0, 0x03, 0x9f, 0x77, 0x4c, 0x8d,
0x97, 0x76
};
unsigned char nist_p521_initiator_y_coord[66] = {
0x00, 0xb3, 0x8e, 0x02, 0xe4, 0x2a, 0x63, 0x59,
0x12, 0xc6, 0x10, 0xba, 0x3a, 0xf9, 0x02, 0x99,
0x3f, 0x14, 0xf0, 0x40, 0xde, 0x5c, 0xc9, 0x8b,
0x02, 0x55, 0xfa, 0x91, 0xb1, 0xcc, 0x6a, 0xbd,
0xe5, 0x62, 0xc0, 0xc5, 0xe3, 0xa1, 0x57, 0x9f,
0x08, 0x1a, 0xa6, 0xe2, 0xf8, 0x55, 0x90, 0xbf,
0xf5, 0xa6, 0xc3, 0xd8, 0x52, 0x1f, 0xb7, 0x02,
0x2e, 0x7c, 0xc8, 0xb3, 0x20, 0x1e, 0x79, 0x8d,
0x03, 0xa8
};
unsigned char nist_p521_responder_x_coord[66] = {
0x00, 0x79, 0xe4, 0x4d, 0x6b, 0x5e, 0x12, 0x0a,
0x18, 0x2c, 0xb3, 0x05, 0x77, 0x0f, 0xc3, 0x44,
0x1a, 0xcd, 0x78, 0x46, 0x14, 0xee, 0x46, 0x3f,
0xab, 0xc9, 0x59, 0x7c, 0x85, 0xa0, 0xc2, 0xfb,
0x02, 0x32, 0x99, 0xde, 0x5d, 0xe1, 0x0d, 0x48,
0x2d, 0x71, 0x7d, 0x8d, 0x3f, 0x61, 0x67, 0x9e,
0x2b, 0x8b, 0x12, 0xde, 0x10, 0x21, 0x55, 0x0a,
0x5b, 0x2d, 0xe8, 0x05, 0x09, 0xf6, 0x20, 0x97,
0x84, 0xb4
};
unsigned char nist_p521_responder_y_coord[66] = {
0x00, 0x46, 0x63, 0x39, 0xbe, 0xcd, 0xa4, 0x2d,
0xca, 0x27, 0x74, 0xd4, 0x1b, 0x91, 0x33, 0x20,
0x83, 0xc7, 0x3b, 0xa4, 0x09, 0x8b, 0x8e, 0xa3,
0x88, 0xe9, 0x75, 0x7f, 0x56, 0x7b, 0x38, 0x84,
0x62, 0x02, 0x7c, 0x90, 0x51, 0x07, 0xdb, 0xe9,
0xd0, 0xde, 0xda, 0x9a, 0x5d, 0xe5, 0x94, 0xd2,
0xcf, 0x9d, 0x4c, 0x33, 0x91, 0xa6, 0xc3, 0x80,
0xa7, 0x6e, 0x7e, 0x8d, 0xf8, 0x73, 0x6e, 0x53,
0xce, 0xe1
};
```

A.1.4. Role-specific Elements for brainpool p256r1

```
unsigned char brainpool_p256_initiator_x_coord[32] = {
0x46, 0x98, 0x18, 0x6c, 0x27, 0xcd, 0x4b, 0x10,
0x7d, 0x55, 0xa3, 0xdd, 0x89, 0x1f, 0x9f, 0xca,
0xc7, 0x42, 0x5b, 0x8a, 0x23, 0xed, 0xf8, 0x75,
0xac, 0xc7, 0xe9, 0x8d, 0xc2, 0x6f, 0xec, 0xd8
};
unsigned char brainpool_p256_initiator_y_coord[32] = {
0x93, 0xca, 0xef, 0xa9, 0x66, 0x3e, 0x87, 0xcd,
0x52, 0x6e, 0x54, 0x13, 0xef, 0x31, 0x67, 0x30,
0x15, 0x13, 0x9d, 0x6d, 0xc0, 0x95, 0x32, 0xbe,
0x4f, 0xab, 0x5d, 0xf7, 0xbf, 0x5e, 0xaa, 0x0b
};
unsigned char brainpool_p256_responder_x_coord[32] = {
0x90, 0x18, 0x84, 0xc9, 0xdc, 0xcc, 0xb5, 0x2f,
0x4a, 0x3f, 0x4f, 0x18, 0x0a, 0x22, 0x56, 0x6a,
0xa9, 0xef, 0xd4, 0xe6, 0xc3, 0x53, 0xc2, 0x1a,
0x23, 0x54, 0xdd, 0x08, 0x7e, 0x10, 0xd8, 0xe3
};
unsigned char brainpool_p256_responder_y_coord[32] = {
0x2a, 0xfa, 0x98, 0x9b, 0xe3, 0xda, 0x30, 0xfd,
0x32, 0x28, 0xcb, 0x66, 0xfb, 0x40, 0x7f, 0xf2,
0xb2, 0x25, 0x80, 0x82, 0x44, 0x85, 0x13, 0x7e,
0x4b, 0xb5, 0x06, 0xc0, 0x03, 0x69, 0x23, 0x64
};
```

A.1.5. Role-specific Elements for brainpool p384r1

```
unsigned char brainpool_p384_initiator_x_coord[48] = {
0x0a, 0x2c, 0xeb, 0x49, 0x5e, 0xb7, 0x23, 0xbd,
0x20, 0x5b, 0xe0, 0x49, 0xdf, 0xcf, 0xcf, 0x19,
0x37, 0x36, 0xe1, 0x2f, 0x59, 0xdb, 0x07, 0x06,
0xb5, 0xeb, 0x2d, 0xae, 0xc2, 0xb2, 0x38, 0x62,
0xa6, 0x73, 0x09, 0xa0, 0x6c, 0x0a, 0xa2, 0x30,
0x99, 0xeb, 0xf7, 0x1e, 0x47, 0xb9, 0x5e, 0xbe
};
unsigned char brainpool_p384_initiator_y_coord[48] = {
0x54, 0x76, 0x61, 0x65, 0x75, 0x5a, 0x2f, 0x99,
0x39, 0x73, 0xca, 0x6c, 0xf9, 0xf7, 0x12, 0x86,
0x54, 0xd5, 0xd4, 0xad, 0x45, 0x7b, 0xbf, 0x32,
0xee, 0x62, 0x8b, 0x9f, 0x52, 0xe8, 0xa0, 0xc9,
0xb7, 0x9d, 0xd1, 0x09, 0xb4, 0x79, 0x1c, 0x3e,
0x1a, 0xbf, 0x21, 0x45, 0x66, 0x6b, 0x02, 0x52
};
unsigned char brainpool_p384_responder_x_coord[48] = {
0x03, 0xa2, 0x57, 0xef, 0xe8, 0x51, 0x21, 0xa0,
0xc8, 0x9e, 0x21, 0x02, 0xb5, 0x9a, 0x36, 0x25,
0x74, 0x22, 0xd1, 0xf2, 0x1b, 0xa8, 0x9a, 0x9b,
0x97, 0xbc, 0x5a, 0xeb, 0x26, 0x15, 0x09, 0x71,
0x77, 0x59, 0xec, 0x8b, 0xb7, 0xe1, 0xe8, 0xce,
0x65, 0xb8, 0xaf, 0xf8, 0x80, 0xae, 0x74, 0x6c
};
unsigned char brainpool_p384_responder_y_coord[48] = {
0x2f, 0xd9, 0x6a, 0xc7, 0x3e, 0xec, 0x76, 0x65,
0x2d, 0x38, 0x7f, 0xec, 0x63, 0x26, 0x3f, 0x04,
0xd8, 0x4e, 0xff, 0xe1, 0x0a, 0x51, 0x74, 0x70,
0xe5, 0x46, 0x63, 0x7f, 0x5c, 0xc0, 0xd1, 0x7c,
0xfb, 0x2f, 0xea, 0xe2, 0xd8, 0x0f, 0x84, 0xcb,
0xe9, 0x39, 0x5c, 0x64, 0xfe, 0xcb, 0x2f, 0xf1
};
```

A.1.6. Role-specific Elements for brainpool p512r1

```
unsigned char brainpool_p512_initiator_x_coord[64] = {
0x4c, 0xe9, 0xb6, 0x1c, 0xe2, 0x00, 0x3c, 0x9c,
0xa9, 0xc8, 0x56, 0x52, 0xaf, 0x87, 0x3e, 0x51,
0x9c, 0xbb, 0x15, 0x31, 0x1e, 0xc1, 0x05, 0xfc,
0x7c, 0x77, 0xd7, 0x37, 0x61, 0x27, 0xd0, 0x95,
0x98, 0xee, 0x5d, 0xa4, 0x3d, 0x09, 0xdb, 0x3d,
0xfa, 0x89, 0x9e, 0x7f, 0xa6, 0xa6, 0x9c, 0xff,
0x83, 0x5c, 0x21, 0x6c, 0x3e, 0xf2, 0xfe, 0xdc,
0x63, 0xe4, 0xd1, 0x0e, 0x75, 0x45, 0x69, 0x0f
};
unsigned char brainpool_p512_initiator_y_coord[64] = {
0x50, 0xb5, 0x9b, 0xfa, 0x45, 0x67, 0x75, 0x94,
0x44, 0xe7, 0x68, 0xb0, 0xeb, 0x3e, 0xb3, 0xb8,
0xf9, 0x99, 0x05, 0xef, 0xae, 0x6c, 0xbc, 0xe3,
0xe1, 0xd2, 0x51, 0x54, 0xdf, 0x59, 0xd4, 0x45,
0x41, 0x3a, 0xa8, 0x0b, 0x76, 0x32, 0x44, 0x0e,
0x07, 0x60, 0x3a, 0x6e, 0xbe, 0xfe, 0xe0, 0x58,
0x52, 0xa0, 0xaa, 0x8b, 0xd8, 0x5b, 0xf2, 0x71,
0x11, 0x9a, 0x9e, 0x8f, 0x1a, 0xd1, 0xc9, 0x99
};
unsigned char brainpool_p512_responder_x_coord[64] = {
0x2a, 0x60, 0x32, 0x27, 0xa1, 0xe6, 0x94, 0x72,
0x1c, 0x48, 0xbe, 0xc5, 0x77, 0x14, 0x30, 0x76,
0xe4, 0xbf, 0xf7, 0x7b, 0xc5, 0xfd, 0xdf, 0x19,
0x1e, 0x0f, 0xdf, 0x1c, 0x40, 0xfa, 0x34, 0x9e,
0x1f, 0x42, 0x24, 0xa3, 0x2c, 0xd5, 0xc7, 0xc9,
0x7b, 0x47, 0x78, 0x96, 0xf1, 0x37, 0x0e, 0x88,
0xcb, 0xa6, 0x52, 0x29, 0xd7, 0xa8, 0x38, 0x29,
0x8e, 0x6e, 0x23, 0x47, 0xd4, 0x4b, 0x70, 0x3e
};
unsigned char brainpool_p512_responder_y_coord[64] = {
0x80, 0x1f, 0x43, 0xd2, 0x17, 0x35, 0xec, 0x81,
0xd9, 0x4b, 0xdc, 0x81, 0x19, 0xd9, 0x5f, 0x68,
0x16, 0x84, 0xfe, 0x63, 0x4b, 0x8d, 0x5d, 0xaa,
0x88, 0x4a, 0x47, 0x48, 0xd4, 0xea, 0xab, 0x7d,
0x6a, 0xbf, 0xe1, 0x28, 0x99, 0x6a, 0x87, 0x1c,
0x30, 0xb4, 0x44, 0x2d, 0x75, 0xac, 0x35, 0x09,
0x73, 0x24, 0x3d, 0xb4, 0x43, 0xb1, 0xc1, 0x56,
0x56, 0xad, 0x30, 0x87, 0xf4, 0xc3, 0x00, 0xc7
};
```

A.2. FFC Role-specific Elements

A.2.1. Role-specific Elements for 2048-bit FFC group

```
unsigned char group 14_initiator[256] = {
0x01, 0x1f, 0x33, 0x72, 0x90, 0x86, 0x76, 0x68,
0x9d, 0x29, 0x9c, 0x42, 0xd2, 0x43, 0x1b, 0xeb,
```

```
0x99, 0x53, 0x3e, 0x5c, 0x3e, 0xe5, 0x15, 0xa1,
0x06, 0x01, 0xb5, 0x8b, 0xac, 0x33, 0xd8, 0xc7,
0x30, 0x4d, 0xec, 0x84, 0x0d, 0xb1, 0x13, 0xd0,
0xb3, 0x44, 0xeb, 0xbe, 0x6f, 0x70, 0x21, 0x8b,
0xd7, 0xe2, 0x86, 0x9f, 0xfc, 0x03, 0xc6, 0x34,
0xd2, 0x08, 0xdb, 0x1d, 0x6e, 0x57, 0xe2, 0xe0,
0xa8, 0x0c, 0xbb, 0xb8, 0x37, 0xa5, 0x73, 0x75,
0x31, 0x48, 0x49, 0x43, 0x24, 0xdb, 0x96, 0x71,
0x40, 0xc6, 0xfa, 0xe7, 0x12, 0x13, 0xb4, 0x20,
0x89, 0x46, 0x63, 0xff, 0x38, 0xc3, 0x72, 0x82,
0xf6, 0xa1, 0x23, 0xd2, 0x2c, 0x25, 0xf9, 0x46,
0x80, 0x76, 0x82, 0xb7, 0xed, 0x6c, 0x21, 0x28,
0x79, 0xdb, 0xaa, 0xdd, 0x69, 0x84, 0xd7, 0x09,
0x20, 0xac, 0x5f, 0x94, 0xf4, 0x10, 0x86, 0x98,
0x0a, 0x69, 0xc4, 0x62, 0xb7, 0x48, 0xea, 0xa5,
0xdf, 0x41, 0xce, 0xfa, 0xb6, 0x00, 0x41, 0xbd,
0x9e, 0x35, 0xfa, 0x15, 0x41, 0x3e, 0xa8, 0x7f,
0x4f, 0x44, 0xae, 0x14, 0x48, 0x53, 0xf2, 0x7c,
0x4d, 0x69, 0xe4, 0xa0, 0x44, 0x32, 0x78, 0xbf,
0x7a, 0x59, 0xe3, 0xd9, 0x6a, 0x32, 0xb1, 0x5a,
0x63, 0xd7, 0x7d, 0x47, 0xb6, 0xe6, 0x00, 0x00,
0xea, 0x70, 0x91, 0x4b, 0xde, 0x0e, 0xf5, 0x76,
0x0b, 0x45, 0x1b, 0xa8, 0xee, 0x99, 0xb0, 0xd2,
0x34, 0x4e, 0x7a, 0x95, 0x46, 0xbb, 0xf6, 0x51,
0xba, 0xfa, 0x15, 0x90, 0xf9, 0x88, 0xc0, 0x49,
0x3f, 0x5d, 0x98, 0x4e, 0x36, 0xcb, 0x96, 0xa9,
0xcd, 0x47, 0x7f, 0x21, 0xff, 0x32, 0xde, 0xb3,
0x65, 0xc3, 0xe1, 0xe9, 0x88, 0x8e, 0xbd, 0x3e,
0xc1, 0x84, 0x63, 0x77, 0x26, 0xf9, 0x90, 0x64,
0x66, 0x3a, 0x5c, 0xfc, 0x44, 0xbd, 0x6f, 0xd0
};
unsigned char group 14_responder[256] = {
0x7a, 0x9e, 0x5f, 0xa9, 0xcb, 0x6e, 0x36, 0xe1,
0x66, 0x75, 0x95, 0x42, 0xe8, 0x86, 0x44, 0xf0,
0xe5, 0xe5, 0x4e, 0x7f, 0xb0, 0x63, 0x5c, 0x38,
0xd3, 0x25, 0x02, 0xd3, 0x2a, 0x72, 0x92, 0xfa,
0x17, 0xa1, 0x93, 0xc2, 0x9a, 0x15, 0xf9, 0x81,
0xa6, 0x16, 0xfc, 0x72, 0xaf, 0xfa, 0xe6, 0x71,
0x08, 0x96, 0x26, 0x49, 0x7a, 0x4d, 0xc8, 0xc2,
0xc1, 0xdb, 0x63, 0x9d, 0xc3, 0x22, 0x3c, 0x9f,
0xb4, 0x00, 0x3e, 0xe7, 0x02, 0x89, 0x0c, 0xb1,
0x65, 0x97, 0x55, 0x0a, 0x74, 0x83, 0x0d, 0xe9,
0x77, 0x5f, 0xc4, 0x00, 0x1c, 0xaf, 0x24, 0xca,
0xb1, 0xcc, 0x31, 0x1c, 0x2d, 0x53, 0x8a, 0x79,
0x01, 0xe1, 0x00, 0x62, 0x61, 0x1c, 0xa8, 0xf7,
0x76, 0x73, 0x24, 0xad, 0x8b, 0xb0, 0x6f, 0xd6,
0x83, 0x3d, 0x06, 0x9f, 0x9d, 0xf0, 0x84, 0x64,
0xb2, 0xba, 0x11, 0xec, 0x1e, 0xfb, 0x21, 0x96,
```

```
0x0a, 0xab, 0x4c, 0x70, 0x79, 0x47, 0x7b, 0x6e,
0xce, 0x22, 0xd5, 0x22, 0x82, 0x06, 0xa8, 0x81,
0x0d, 0x39, 0x03, 0xca, 0x5f, 0x54, 0x67, 0x79,
0x20, 0xa7, 0xde, 0xd6, 0xba, 0x1e, 0x33, 0xe8,
0x85, 0xa0, 0x39, 0x5f, 0x8d, 0x8a, 0x91, 0x28,
0xb2, 0x63, 0xe6, 0x9b, 0xd1, 0x68, 0xff, 0xd8,
0x57, 0x7d, 0x85, 0x43, 0x70, 0xe1, 0xab, 0x55,
0x13, 0xc7, 0x02, 0x23, 0xfa, 0x8f, 0xf7, 0x9c,
0x25, 0x8e, 0xc1, 0x0e, 0xd4, 0xab, 0xf4, 0x81,
0x38, 0x86, 0x22, 0x16, 0x24, 0x06, 0x7f, 0x37,
0xbb, 0x2d, 0x16, 0x2b, 0xc7, 0x82, 0xe4, 0x93,
0xf6, 0x6b, 0x8f, 0x1f, 0xb6, 0x6f, 0x63, 0x66,
0x4d, 0xa4, 0x39, 0xd5, 0x57, 0x3b, 0x73, 0x69,
0x22, 0xf1, 0x62, 0xb3, 0xf4, 0x8c, 0x5c, 0x3f,
0xc8, 0xb1, 0x94, 0x76, 0x2b, 0x7f, 0x6b, 0x8d,
0xc6, 0xa5, 0x5f, 0xc6, 0x06, 0x74, 0x36, 0xea
};
```

A.2.2. Role-specific Elements for 3072-bit FFC group

```
unsigned char group 15_initiator[384] = {
0x2a, 0xfc, 0x6e, 0xa4, 0x33, 0x54, 0xa1, 0xba,
0x34, 0x25, 0x84, 0x6c, 0xe3, 0x54, 0x2d, 0x52,
0xdd, 0x59, 0x9c, 0xef, 0xa6, 0x96, 0x2d, 0x1d,
0x53, 0xd4, 0xd4, 0x2e, 0xe9, 0x18, 0xb3, 0x2d,
0x75, 0x11, 0xeb, 0x3f, 0x1d, 0x3d, 0xac, 0x67,
0x62, 0x99, 0xa6, 0xe0, 0x22, 0xa1, 0xa5, 0xd6,
0x07, 0xfb, 0xe0, 0x76, 0x29, 0x8f, 0xf7, 0x3d,
0xa1, 0x99, 0x64, 0x44, 0xb5, 0xe4, 0xfa, 0x69,
0x00, 0x3c, 0x46, 0x56, 0x99, 0xf1, 0xb6, 0xc1,
0xa4, 0x2d, 0x54, 0xf4, 0x4e, 0x2c, 0xdc, 0x14,
0x27, 0xf5, 0xbb, 0x55, 0x61, 0xda, 0x36, 0x0d,
0x46, 0xa6, 0xd7, 0xe9, 0x9e, 0xcc, 0x7e, 0x35,
0x87, 0x32, 0xa1, 0xb9, 0x80, 0x07, 0x16, 0xaa,
0x74, 0xa5, 0x0f, 0xe0, 0x96, 0xb1, 0x25, 0x88,
0x6d, 0xda, 0x64, 0xc9, 0xa9, 0x5e, 0x6d, 0xb8,
0x7a, 0xf4, 0x42, 0xf3, 0xba, 0x37, 0xe8, 0xbd,
0x23, 0x36, 0x7b, 0xdc, 0x60, 0x93, 0x94, 0x5a,
0xb2, 0x99, 0x2a, 0x22, 0x1d, 0x50, 0xd6, 0x1d,
0xb7, 0xbc, 0xb9, 0xd1, 0x99, 0x3c, 0x06, 0x11,
0x79, 0x06, 0x21, 0x58, 0x60, 0x45, 0x3a, 0x00,
0xb6, 0x43, 0x0d, 0xcd, 0xa7, 0x60, 0x83, 0x3a,
0x7d, 0x9c, 0x35, 0x58, 0xc4, 0x0d, 0xcc, 0xef,
0x66, 0x55, 0xa9, 0xd2, 0xce, 0xe2, 0x80, 0x73,
0x26, 0xab, 0x7c, 0x8a, 0xf9, 0x1b, 0x3e, 0xf7,
0x75, 0x31, 0xea, 0x7f, 0x4a, 0x57, 0x15, 0x9a,
0x71, 0x92, 0xc3, 0x8f, 0xca, 0xab, 0x4b, 0x98,
0x11, 0xbe, 0x58, 0x8c, 0x20, 0x3d, 0x73, 0x4e,
```

```
0x39, 0xad, 0x17, 0x10, 0x99, 0x4f, 0x2e, 0x70,
0xae, 0xb6, 0xb8, 0x54, 0x2a, 0x37, 0x12, 0xf1,
0x85, 0x64, 0x9d, 0x97, 0x79, 0x8d, 0x69, 0x8c,
0x27, 0xd4, 0xf3, 0x65, 0x8a, 0xf3, 0x41, 0x42,
0x3e, 0x89, 0xf0, 0xa5, 0xbe, 0x71, 0x40, 0xb6,
0x56, 0x65, 0xb1, 0x62, 0x1f, 0x09, 0x76, 0xa3,
0xad, 0xb1, 0x16, 0x61, 0x87, 0x85, 0xfc, 0x1d,
0xa3, 0x1a, 0xf9, 0xa2, 0x4b, 0x25, 0x1c, 0x9f,
0x6d, 0x9b, 0xcd, 0x02, 0xc4, 0x0f, 0x64, 0x54,
0x97, 0x83, 0x2c, 0x41, 0xd6, 0x7b, 0x59, 0x0d,
0xcf, 0xdd, 0xa4, 0xd0, 0x75, 0xeb, 0xd9, 0x1c,
0xb8, 0xcb, 0x6c, 0x80, 0x00, 0x24, 0xf6, 0xf8,
0x62, 0x82, 0x97, 0x75, 0x0a, 0x4c, 0xfa, 0xbb,
0xbb, 0xe0, 0x87, 0x25, 0x86, 0x80, 0xc3, 0xb0,
0xc6, 0xb2, 0xfb, 0xe2, 0x8f, 0xb4, 0xd2, 0xc3,
0xbb, 0x78, 0xf4, 0xef, 0x9c, 0x1f, 0xd3, 0xa5,
0xab, 0xcf, 0xc2, 0xbd, 0x63, 0xc4, 0x5b, 0x2c,
0x9c, 0x3d, 0xa3, 0xed, 0xae, 0x97, 0xcc, 0x54,
0xdb, 0x3c, 0x04, 0x38, 0x1b, 0xaf, 0x22, 0x27,
0x53, 0xa4, 0xc1, 0xd6, 0x4a, 0x8f, 0xe9, 0x77,
0x13, 0x86, 0xf8, 0x0e, 0x1b, 0x2a, 0xdc, 0x6f
};
unsigned char group 15_responder[384] = {
0xbe, 0xfa, 0x77, 0xff, 0x9c, 0xa4, 0x21, 0x86,
0x6f, 0x22, 0x42, 0xf2, 0x86, 0x12, 0x70, 0x57,
0x7b, 0x1e, 0x00, 0x82, 0x0a, 0x10, 0xad, 0x84,
0x52, 0xe6, 0x3c, 0x39, 0x5e, 0x0d, 0xcc, 0x13,
0xfc, 0x82, 0x32, 0x58, 0x1d, 0x74, 0xab, 0x6e,
0xfa, 0xf1, 0xc2, 0x2f, 0x80, 0x55, 0xd0, 0x1e,
0x8a, 0x6d, 0x75, 0x8e, 0x80, 0x24, 0x64, 0x0e,
0x66, 0xc2, 0xf5, 0xbf, 0x89, 0x1c, 0x6b, 0xee,
0x35, 0x4c, 0x44, 0x16, 0x12, 0xe9, 0x26, 0x44,
0x74, 0xdd, 0x24, 0x84, 0x36, 0xfe, 0x5a, 0x66,
0x8a, 0xb6, 0x7c, 0xab, 0xf2, 0x8c, 0xc3, 0x98,
0xe7, 0xb0, 0xd1, 0x45, 0x22, 0xbf, 0x49, 0xa4,
0x09, 0x0e, 0xf0, 0xdf, 0xb5, 0xc4, 0xf7, 0xc9,
0x2d, 0x9e, 0x65, 0x93, 0x5d, 0x84, 0x1b, 0x93,
0xec, 0x5e, 0xdc, 0xb6, 0x8b, 0xee, 0x84, 0x3e,
0x0d, 0xf8, 0x81, 0x00, 0x60, 0x55, 0x8d, 0xab,
0x51, 0x31, 0x2c, 0xf4, 0x85, 0xbe, 0x4b, 0xe0,
0x61, 0xc2, 0x9a, 0xd1, 0xdb, 0xb2, 0x32, 0x11,
0x01, 0xca, 0xa3, 0x23, 0x28, 0xf8, 0x5a, 0x40,
0xe2, 0xaf, 0x65, 0xd5, 0xa1, 0x4f, 0xae, 0xa2,
0x1e, 0x3c, 0x23, 0xd3, 0x53, 0xb7, 0x59, 0xe6,
0x02, 0x5f, 0xb1, 0x81, 0xd9, 0xd9, 0x41, 0x02,
0x2d, 0xf3, 0x7f, 0xbe, 0x08, 0x9c, 0xa8, 0x58,
0x4f, 0x72, 0x7a, 0x71, 0xc8, 0x34, 0xb4, 0xbe,
0xd6, 0x46, 0x55, 0x47, 0x15, 0x29, 0x95, 0x01,
```

```
0x19, 0x1f, 0xbb, 0xfe, 0x0d, 0xce, 0xb6, 0x41,
0xf7, 0x22, 0x19, 0x8b, 0x57, 0x2a, 0x42, 0x05,
0xbc, 0xba, 0x08, 0xb5, 0xd3, 0x5b, 0xab, 0xc5,
0x34, 0xb5, 0xe4, 0x2e, 0xf4, 0x23, 0x69, 0x63,
0x0c, 0x0e, 0xfd, 0x9d, 0xf1, 0x3f, 0xee, 0x14,
0xad, 0x9b, 0x2c, 0x61, 0x09, 0xb0, 0xea, 0x46,
0x3c, 0x16, 0xca, 0xcd, 0x72, 0x53, 0xe9, 0xf7,
0x87, 0x96, 0x6f, 0xec, 0xbf, 0x93, 0x36, 0x43,
0x66, 0x60, 0x48, 0xfe, 0x3f, 0xb0, 0x47, 0x26,
0x87, 0x86, 0x07, 0x08, 0xb4, 0x7d, 0xab, 0x60,
0xad, 0xf1, 0x84, 0xd9, 0x5a, 0xeb, 0xbb, 0xdb,
0x15, 0x69, 0x42, 0x62, 0x2c, 0x82, 0x6a, 0x24,
0xcb, 0xce, 0x7d, 0xd9, 0xd3, 0xcb, 0x10, 0x55,
0x73, 0x36, 0x15, 0xe2, 0x05, 0x91, 0xc9, 0x68,
0x09, 0x76, 0xcb, 0xcf, 0x6c, 0xd2, 0x06, 0x34,
0xcd, 0xb5, 0x69, 0x44, 0x66, 0x33, 0x37, 0xec,
0x24, 0x17, 0x73, 0x79, 0x74, 0xdd, 0xba, 0x04,
0xad, 0xb9, 0xd6, 0xef, 0x60, 0xcb, 0x58, 0xfd,
0x71, 0xac, 0x6e, 0xb8, 0x78, 0xd7, 0x4d, 0x6e,
0x72, 0xa1, 0x78, 0x68, 0xbd, 0x9c, 0x56, 0x81,
0x94, 0x69, 0xc7, 0x63, 0xe3, 0x2b, 0xda, 0x76,
0xe5, 0x2f, 0xf8, 0xaa, 0xb2, 0x4b, 0xf6, 0xa1,
0xe5, 0xa7, 0xa2, 0xbc, 0xf9, 0x0a, 0xb9, 0x63
};
```

A.2.3. Role-specific Elements for 4096-bit FFC group

```
unsigned char group 16_initiator[512] = {
0x2e, 0xf4, 0x19, 0xbc, 0x45, 0x4b, 0x5a, 0x16,
0x05, 0x38, 0xc0, 0x82, 0x6e, 0xab, 0x66, 0xcc,
0xe5, 0xd1, 0xd8, 0x64, 0xdc, 0x5a, 0x8d, 0xae,
0x90, 0x00, 0x1a, 0x72, 0xb9, 0xd5, 0xbb, 0xfa,
0xc1, 0x91, 0xe3, 0xde, 0x50, 0xed, 0x31, 0x31,
0x4b, 0xf2, 0xb7, 0x2e, 0xbe, 0xa0, 0x31, 0x9b,
0xce, 0xbf, 0x35, 0xd8, 0xde, 0xb6, 0x38, 0xd3,
0x2c, 0xfc, 0xf5, 0x7b, 0x5f, 0x60, 0xef, 0x11,
0x08, 0x44, 0x0a, 0x68, 0x6c, 0x07, 0x40, 0x3b,
0xdc, 0xc8, 0x1d, 0xdd, 0xd0, 0xc3, 0x15, 0x19,
0xcf, 0x85, 0x43, 0xc0, 0xab, 0x65, 0x75, 0x48,
0x75, 0x54, 0x5d, 0x9d, 0x73, 0xd6, 0x57, 0x08,
0x78, 0x0c, 0x3b, 0xfd, 0x22, 0x90, 0xdf, 0x5b,
0x13, 0x90, 0x17, 0x61, 0xb3, 0x18, 0x67, 0x14,
0x1e, 0xaa, 0x81, 0xea, 0x9e, 0xd0, 0xe7, 0x4e,
0x8b, 0x69, 0xc8, 0xef, 0xe4, 0x58, 0x9e, 0xf5,
0x86, 0xd1, 0x3b, 0xd2, 0x94, 0x7d, 0x8a, 0x95,
0xca, 0xdc, 0x04, 0x80, 0x60, 0x66, 0x4f, 0x2c,
0xf5, 0x69, 0xb4, 0xd6, 0x9e, 0xe6, 0xf9, 0x88,
0x0a, 0x0b, 0x5e, 0x01, 0xc7, 0x50, 0xad, 0xe8,
```

```
0x4f, 0x1d, 0x0c, 0xcd, 0x6c, 0x92, 0x46, 0x2e,
0x06, 0x4f, 0x7d, 0x18, 0x1b, 0xb8, 0x03, 0xef,
0xff, 0x85, 0x59, 0x16, 0x44, 0xd3, 0x28, 0x80,
0x58, 0x91, 0xf0, 0x9c, 0x08, 0x83, 0x87, 0x63,
0xf8, 0x6d, 0xc9, 0x3a, 0x17, 0x9d, 0xb0, 0x50,
0x4f, 0x1f, 0xa5, 0x6a, 0x88, 0xda, 0xf2, 0xab,
0x93, 0x36, 0x58, 0x9b, 0xf3, 0xe7, 0x93, 0xac,
0x28, 0xd9, 0x62, 0xf3, 0xc5, 0xe0, 0x2c, 0xe1,
0x23, 0x38, 0xb9, 0xd7, 0xfc, 0x54, 0x0e, 0x8e,
0x28, 0xf3, 0x88, 0x32, 0x81, 0x8b, 0x45, 0x47,
0xe9, 0x54, 0xff, 0x7f, 0x8b, 0x45, 0xc2, 0xc5,
0xa1, 0xe3, 0x9a, 0x02, 0xd4, 0x8b, 0x91, 0x44,
0x90, 0xfa, 0xb0, 0x86, 0x27, 0xac, 0x09, 0x7b,
0x93, 0x75, 0x86, 0xfd, 0x46, 0x99, 0xbf, 0xd9,
0xbd, 0xe2, 0xf2, 0x79, 0x24, 0x9a, 0x84, 0x5c,
0x12, 0x67, 0xf8, 0xe1, 0xa8, 0xd6, 0x60, 0x31,
0x0f, 0xd9, 0x7f, 0xb3, 0xba, 0x0c, 0x92, 0x55,
0x6a, 0x5c, 0x8a, 0xca, 0x98, 0x78, 0xbc, 0x0d,
0x9f, 0x6c, 0x26, 0xab, 0xfb, 0x80, 0xed, 0xa8,
0xb3, 0x08, 0x15, 0xaa, 0x46, 0x09, 0x6a, 0x55,
0x76, 0xd4, 0xbf, 0xc0, 0x84, 0xf6, 0xf0, 0x41,
0xc0, 0xf8, 0xdb, 0xb7, 0x46, 0xe2, 0xa0, 0xf3,
0xde, 0x6a, 0xdc, 0x44, 0x9c, 0x79, 0xb2, 0xff,
0xc9, 0xaa, 0x42, 0x4d, 0x75, 0x53, 0x39, 0xaf,
0x91, 0x84, 0x51, 0x9b, 0xc7, 0x5b, 0xbc, 0x36,
0x5b, 0xbe, 0x47, 0xd4, 0x8b, 0x25, 0x4c, 0xa1,
0xf6, 0x0f, 0x8a, 0x35, 0x45, 0x24, 0x23, 0x48,
0x1a, 0xda, 0x84, 0xf9, 0x33, 0x67, 0x55, 0x24,
0xf1, 0xff, 0xe0, 0x28, 0x5c, 0x8c, 0x85, 0x28,
0xf1, 0xfc, 0x3d, 0x31, 0xb2, 0x38, 0x24, 0x79,
0x1b, 0x80, 0x44, 0xca, 0xd9, 0x25, 0x87, 0xa1,
0xba, 0x7f, 0xba, 0x40, 0x01, 0x1c, 0x7a, 0xc1,
0x09, 0xe6, 0x37, 0xc0, 0xd3, 0x8d, 0xc5, 0xc4,
0x81, 0xad, 0xc9, 0xa2, 0x86, 0x93, 0xb5, 0x50,
0x29, 0xd8, 0x03, 0x8b, 0x76, 0xd7, 0x94, 0x65,
0x7a, 0x8c, 0x85, 0xad, 0xd6, 0xf7, 0x83, 0x64,
0x86, 0x5a, 0x53, 0xc4, 0xa8, 0x56, 0x87, 0xa1,
0xb3, 0xd9, 0x8c, 0xed, 0xb8, 0x10, 0x91, 0xdf,
0xbc, 0xb4, 0x64, 0xa8, 0x7c, 0x51, 0xf6, 0xaa,
0x47, 0x62, 0xbe, 0x01, 0xa4, 0x10, 0x4d, 0x4a,
0x9a, 0xf1, 0x0c, 0xb6, 0xd0, 0xde, 0xb2, 0x78,
0x5b, 0xd8, 0x65, 0x6f, 0x6e, 0xf8, 0x12, 0x20,
0xac, 0x3f, 0x1b, 0x6e, 0x3a, 0x0d, 0xed, 0x84,
0xde, 0x5e, 0x23, 0x83, 0x9e, 0xd9, 0x6d, 0x05
};
unsigned char group 16_responder[512] = {
0xe1, 0x1e, 0x9b, 0x32, 0x93, 0x44, 0xc0, 0xac,
0xc2, 0x27, 0x6c, 0x08, 0xdc, 0x7f, 0xe7, 0x7b,
```

```
0xa5, 0x21, 0xaa, 0x31, 0xc3, 0xd5, 0x45, 0xe2,
0x8c, 0xd5, 0x01, 0x4f, 0x1c, 0x33, 0xba, 0x5e,
0x28, 0x4e, 0x85, 0xd8, 0x2a, 0x88, 0x2f, 0x93,
0x1d, 0x5e, 0x00, 0x2f, 0xc1, 0x4a, 0xc2, 0x17,
0x76, 0x0f, 0xfb, 0xfd, 0x8b, 0xf7, 0xbc, 0x52,
0x3a, 0x04, 0x32, 0x9e, 0xd7, 0xdf, 0xf2, 0x32,
0x57, 0x15, 0xe1, 0xd4, 0x36, 0xaf, 0xd0, 0xb7,
0xfc, 0xb3, 0x00, 0x11, 0x57, 0xf4, 0x85, 0xed,
0x85, 0x47, 0xc9, 0xe8, 0xca, 0x89, 0x67, 0x95,
0x44, 0x7b, 0x98, 0x38, 0xed, 0x6e, 0xb8, 0xc6,
0x8b, 0xb0, 0xf5, 0x15, 0xde, 0xaf, 0x5b, 0x19,
0xe2, 0x8f, 0xde, 0x85, 0x47, 0xdd, 0x36, 0xd2,
0xf4, 0x49, 0xbd, 0x06, 0x75, 0xaa, 0x74, 0x7c,
0xc9, 0x0d, 0xc2, 0x10, 0x3c, 0xf6, 0x0d, 0xe1,
0x7a, 0x3f, 0x06, 0x8d, 0x98, 0x98, 0x9b, 0x21,
0xdf, 0x30, 0xf6, 0xa6, 0x3d, 0x72, 0x59, 0x69,
0x3b, 0x9f, 0xad, 0x82, 0x60, 0x8e, 0xf0, 0xa6,
0x2c, 0xa6, 0x3c, 0x94, 0x1e, 0x1c, 0xe6, 0x8a,
0xf2, 0xef, 0x66, 0xa9, 0x89, 0x80, 0x82, 0x5e,
0x41, 0x51, 0xf4, 0x6e, 0xdd, 0xeb, 0x23, 0x67,
0x42, 0x80, 0x28, 0xab, 0x8a, 0xf5, 0x04, 0xa1,
0xae, 0x63, 0xdf, 0xa7, 0x8f, 0xdf, 0x91, 0x50,
0x1d, 0x38, 0x52, 0xb3, 0x8b, 0x45, 0x9d, 0x91,
0x91, 0xba, 0x07, 0x0b, 0xce, 0xd6, 0xb2, 0xa2,
0xfc, 0xca, 0x16, 0x43, 0xae, 0x63, 0xf6, 0xc2,
0xb3, 0xac, 0x44, 0x78, 0x88, 0xbe, 0xd1, 0x69,
0xbb, 0x93, 0x40, 0x6f, 0x11, 0x83, 0xfa, 0x33,
0xc4, 0xb4, 0x4b, 0x66, 0xda, 0xa3, 0x30, 0x1b,
0x5d, 0x21, 0xc8, 0x3f, 0xde, 0xc5, 0xce, 0x2b,
0x01, 0xd1, 0x4e, 0xcb, 0xa5, 0xe5, 0x42, 0x12,
0xea, 0x48, 0xd1, 0x5c, 0x27, 0xf9, 0x94, 0x82,
0x52, 0x8d, 0xe6, 0xbf, 0x67, 0x3e, 0xbd, 0xbb,
0xea, 0xe7, 0x3c, 0x85, 0xf3, 0xcf, 0x8a, 0xd8,
0x1f, 0x5c, 0x33, 0x90, 0x9b, 0x2c, 0x2a, 0xf1,
0x29, 0x89, 0x1e, 0x42, 0x39, 0xef, 0xc0, 0xca,
0x96, 0x3a, 0x8e, 0xc9, 0x73, 0xb2, 0xa8, 0x95,
0xcb, 0x61, 0xc7, 0xa6, 0xac, 0x55, 0xb4, 0xef,
0x71, 0x3c, 0x6e, 0xfd, 0x40, 0xe8, 0x19, 0x5b,
0x2d, 0x66, 0x90, 0x8b, 0xa0, 0x8c, 0x56, 0xe4,
0xaa, 0x10, 0xd5, 0xca, 0xed, 0x5e, 0x41, 0x19,
0x57, 0x2b, 0xbd, 0x93, 0xf4, 0xc5, 0xaf, 0x47,
0xf1, 0x61, 0xd1, 0xdd, 0xef, 0x3a, 0x73, 0xdd,
0x28, 0xd0, 0xa9, 0xf1, 0x3b, 0x59, 0x85, 0x34,
0x4a, 0xda, 0x1d, 0xa4, 0xf6, 0x57, 0x76, 0x04,
0x88, 0x75, 0x68, 0xb2, 0x1b, 0xc4, 0xef, 0x1a,
0x2c, 0x2d, 0x72, 0xa4, 0xbf, 0xfc, 0x62, 0x6e,
0xe8, 0x3f, 0x07, 0x6f, 0x49, 0x62, 0x2d, 0x3b,
0xca, 0x61, 0xeb, 0x9a, 0x85, 0xb0, 0x1f, 0x2b,
```

```
0x00, 0xb6, 0x59, 0x21, 0x9d, 0xc1, 0x91, 0xd8,
0x20, 0x0d, 0x83, 0x2c, 0xfa, 0x67, 0xd5, 0x5a,
0x1d, 0xa5, 0xdf, 0xc5, 0xae, 0x4b, 0x79, 0x41,
0x34, 0x18, 0x5b, 0xff, 0xa9, 0x07, 0x3c, 0x35,
0x92, 0x5a, 0x2b, 0x1a, 0x13, 0x5a, 0x2c, 0x88,
0x4c, 0x5b, 0x87, 0xee, 0x19, 0xc5, 0xca, 0xf9,
0x2b, 0x4c, 0x44, 0x59, 0x8b, 0x1f, 0x65, 0x48,
0x49, 0xbc, 0xb5, 0xf0, 0x02, 0x96, 0xb5, 0x18,
0xc5, 0x58, 0x01, 0x5c, 0xf3, 0x26, 0x39, 0x7b,
0x35, 0x74, 0x20, 0x0c, 0xcb, 0x86, 0x8d, 0x70,
0xfc, 0xce, 0xdf, 0x88, 0x7f, 0xe9, 0x6b, 0xc7,
0x08, 0x2d, 0x17, 0xea, 0x72, 0x6e, 0xbc, 0xdd,
0xc8, 0xfe, 0x62, 0x7c, 0x8f, 0x9a, 0x5e, 0xad,
0x47, 0x60, 0xb6, 0xa1, 0x82, 0x1a, 0xf9, 0xcc
};
```

A.2.4. Role-specific Elements for 8192-bit FFC group

```
unsigned char group 18_initiator[1024] = {
0x42, 0x5b, 0x57, 0x35, 0x57, 0xed, 0x1c, 0x14,
0xcd, 0x91, 0x34, 0x61, 0x75, 0x67, 0x88, 0x52,
0xf9, 0x10, 0x44, 0xad, 0x3c, 0xbf, 0x83, 0x2b,
0xd7, 0x94, 0x70, 0x7e, 0xe2, 0x79, 0x72, 0xfd,
0x44, 0xdb, 0xc2, 0xb5, 0x21, 0xae, 0x4a, 0x78,
0xad, 0x45, 0x09, 0xa6, 0x3c, 0x79, 0x07, 0x09,
0x65, 0x57, 0xf2, 0xac, 0x81, 0x90, 0xe9, 0x77,
0x3e, 0x7a, 0xd4, 0xbf, 0x56, 0x81, 0x35, 0x41,
0x31, 0x21, 0x8a, 0xfb, 0x03, 0xa2, 0xe0, 0x01,
0x27, 0x9b, 0x07, 0x45, 0x35, 0xcc, 0x84, 0x7e,
0xcc, 0x7b, 0x01, 0xb6, 0x80, 0xd9, 0x2e, 0x1e,
0xa3, 0x09, 0xf3, 0x15, 0x47, 0xf5, 0x37, 0x0d,
0xb0, 0x22, 0x39, 0x5a, 0xd1, 0xb3, 0xf5, 0x11,
0x5c, 0x63, 0x08, 0x8e, 0x80, 0xde, 0x08, 0xe2,
0xf5, 0xbc, 0xbb, 0xae, 0x21, 0xb5, 0xed, 0x2c,
0x7b, 0xa9, 0xdf, 0x54, 0xf3, 0x3a, 0xd5, 0x0e,
0x34, 0x33, 0x97, 0xae, 0x7f, 0x35, 0x67, 0x4e,
0x29, 0xca, 0x1d, 0xe6, 0xea, 0x04, 0x23, 0xad,
0x8f, 0x1e, 0xe3, 0xeb, 0xd2, 0x55, 0xc1, 0x02,
0x2e, 0x95, 0x4f, 0xd9, 0x97, 0x17, 0xd9, 0x7f,
0x31, 0xca, 0xf7, 0xa8, 0xa6, 0x59, 0x44, 0x44,
0xd2, 0x3f, 0xbe, 0x71, 0xb6, 0x87, 0xe8, 0x07,
0x84, 0x0d, 0x46, 0x7b, 0x24, 0x56, 0x74, 0x06,
0xcd, 0x46, 0x34, 0x6a, 0x73, 0x18, 0xbc, 0xbb,
0x57, 0x5e, 0xb1, 0x8d, 0xf5, 0xc7, 0xb6, 0x85,
0xdd, 0x14, 0x8a, 0x74, 0x15, 0xf1, 0x23, 0xda,
0xd3, 0xac, 0xfc, 0x59, 0x61, 0x0a, 0x78, 0x3b,
0x5a, 0x93, 0x8e, 0x10, 0x59, 0x74, 0x7c, 0xa8,
0x2c, 0x97, 0x50, 0xf0, 0x44, 0x73, 0x03, 0xad,
```

```
0xb4, 0xb8, 0x1a, 0x2b, 0xa7, 0x54, 0xdb, 0x33,
0xd3, 0x82, 0xda, 0x8b, 0x93, 0x39, 0x70, 0xe4,
0x1a, 0x3a, 0x88, 0xc4, 0x9f, 0x62, 0x90, 0x3a,
0xeb, 0x32, 0x07, 0x80, 0x64, 0x95, 0xf2, 0x9e,
0xf1, 0xb5, 0xed, 0xcf, 0x78, 0x1a, 0x44, 0x96,
0x48, 0xb5, 0x40, 0xc9, 0x0a, 0x46, 0xa6, 0xcb,
0xb5, 0x85, 0xf2, 0x63, 0x5c, 0x0c, 0x4e, 0x77,
0x06, 0xc1, 0x44, 0x5f, 0xd9, 0x0b, 0xeb, 0x14,
0xa4, 0x74, 0x14, 0x57, 0x55, 0x5b, 0xad, 0xb2,
0x92, 0x53, 0x0a, 0x10, 0x54, 0x61, 0xae, 0x95,
0x2f, 0x54, 0x83, 0xfe, 0x22, 0x67, 0x3e, 0xe3,
0x99, 0x06, 0x4e, 0x0c, 0x6a, 0x4d, 0xd0, 0xcb,
0x82, 0xc1, 0x67, 0xfe, 0xb2, 0x9b, 0xde, 0xc4,
0x0e, 0x19, 0x97, 0x59, 0xc8, 0xe3, 0x75, 0xc2,
0xf1, 0xd6, 0xff, 0xf6, 0xca, 0x84, 0x4c, 0x40,
0xa4, 0x41, 0xd0, 0xf6, 0x09, 0x99, 0x6d, 0x94,
0x14, 0x80, 0xe2, 0x0a, 0x44, 0x5f, 0xf4, 0xce,
0xe3, 0xc9, 0x3f, 0xff, 0x13, 0xdc, 0x90, 0xce,
0x35, 0x21, 0xa9, 0x83, 0x59, 0xa3, 0x67, 0x3c,
0xa4, 0x84, 0x3f, 0x82, 0x70, 0x19, 0xf1, 0x84,
0x1a, 0x3f, 0xba, 0x71, 0xa3, 0x73, 0x0b, 0xa2,
0x80, 0x1d, 0x45, 0xa9, 0xf0, 0xba, 0x4b, 0x72,
0x7b, 0xc0, 0x16, 0x36, 0x37, 0x3e, 0x39, 0x2e,
0xcb, 0x5e, 0x1b, 0x03, 0x6d, 0xab, 0xd1, 0xd0,
0x24, 0x6f, 0x0f, 0x35, 0x83, 0x8f, 0xd1, 0x1b,
0x0b, 0xb2, 0x69, 0xcf, 0x78, 0x50, 0xeb, 0x52,
0xd6, 0x66, 0x01, 0xc9, 0x50, 0xa8, 0x11, 0x4d,
0x2b, 0xf7, 0x95, 0x43, 0xe1, 0x44, 0x2c, 0x19,
0xa3, 0x9e, 0xc6, 0x71, 0xdc, 0x76, 0x47, 0x1d,
0xb8, 0x53, 0xab, 0xed, 0x28, 0x01, 0xdd, 0x6b,
0x3b, 0xe2, 0x19, 0xce, 0xf9, 0x9b, 0xac, 0x9b,
0xba, 0x50, 0x93, 0x6f, 0x90, 0xb3, 0x5a, 0x58,
0xbf, 0x92, 0x2a, 0x30, 0x35, 0xe8, 0x0f, 0x23,
0xc1, 0x8c, 0x86, 0x94, 0x89, 0xd2, 0x7c, 0x64,
0x60, 0x32, 0xa6, 0x30, 0xdd, 0x50, 0xf3, 0x47,
0xc2, 0x59, 0xb9, 0xa1, 0xf0, 0xa7, 0x7e, 0xb1,
0x08, 0xea, 0xfb, 0x72, 0x7e, 0x24, 0xe0, 0x75,
0x8e, 0x0e, 0xbf, 0xa0, 0x89, 0xa0, 0x73, 0x23,
0x85, 0x37, 0xd6, 0xad, 0x67, 0x08, 0x8d, 0x4e,
0x1c, 0x81, 0xdc, 0x3c, 0xd9, 0x69, 0x4a, 0x26,
0x81, 0x4d, 0xb6, 0x4c, 0x70, 0x3b, 0xf9, 0x43,
0xf9, 0x2e, 0xd7, 0xba, 0x24, 0x82, 0xc7, 0x67,
0xac, 0xc4, 0xbe, 0x14, 0xf7, 0xdf, 0xd0, 0x6e,
0xa0, 0x70, 0x0d, 0xff, 0x31, 0x59, 0xc7, 0xf6,
0x43, 0x5f, 0x32, 0x94, 0xd1, 0xf5, 0x9c, 0x7c,
0xff, 0x55, 0xc4, 0xf0, 0x43, 0x22, 0xe2, 0xb1,
0x58, 0x83, 0xa7, 0x7e, 0x00, 0x15, 0xee, 0xe1,
0xff, 0xe8, 0x81, 0xbc, 0xb1, 0xfc, 0x3d, 0xc6,
```

```
0x5e, 0x12, 0xbe, 0xa2, 0x71, 0x82, 0x34, 0xbc,
0xb9, 0x7a, 0xe5, 0x22, 0xc9, 0xe6, 0x13, 0x62,
0x7a, 0xb3, 0x85, 0xec, 0xe3, 0x6d, 0xd0, 0xb6,
0x44, 0x8c, 0x62, 0x3e, 0x06, 0x49, 0x77, 0x9d,
0x90, 0x62, 0x19, 0x0f, 0x1e, 0xd3, 0x6a, 0x2b,
0x9b, 0xe3, 0xf1, 0x9b, 0xc5, 0xc3, 0x81, 0x38,
0x3f, 0x40, 0x26, 0x08, 0x0c, 0x4e, 0xfa, 0x0e,
0x41, 0xb0, 0x4a, 0x6f, 0x85, 0x6f, 0x49, 0x94,
0x01, 0x90, 0x8d, 0x02, 0x4b, 0x22, 0x54, 0x71,
0xbb, 0x2b, 0xab, 0xb6, 0x95, 0xf7, 0x51, 0x53,
0x27, 0x6c, 0x5a, 0x8e, 0x10, 0x03, 0x64, 0x63,
0xf4, 0x2f, 0x40, 0x94, 0x66, 0xc7, 0x59, 0xa9,
0xba, 0xd1, 0x4d, 0xa6, 0x8c, 0x55, 0x82, 0x25,
0xa6, 0x3b, 0xb5, 0x92, 0xc1, 0x81, 0xf6, 0x2f,
0x9d, 0x6d, 0xc0, 0x4c, 0x98, 0xd0, 0x82, 0x78,
0xa5, 0xac, 0xba, 0xee, 0x33, 0x47, 0xa3, 0x49,
0x00, 0xdd, 0x13, 0x09, 0x41, 0xaf, 0x52, 0xe3,
0x5f, 0xe8, 0xc2, 0x66, 0x22, 0x53, 0x3c, 0xd9,
0x17, 0xe6, 0x57, 0x0c, 0x49, 0xc0, 0xda, 0x45,
0xc0, 0x61, 0x1c, 0x25, 0xd2, 0xa9, 0x90, 0x82,
0x7e, 0x6b, 0x4a, 0xc1, 0xd2, 0xa3, 0x86, 0x0a,
0x73, 0x8b, 0x42, 0x3f, 0xc9, 0x29, 0x70, 0xda,
0xff, 0x29, 0x50, 0xa5, 0x25, 0x9f, 0xdd, 0xef,
0x03, 0x2a, 0x79, 0x62, 0xf6, 0x55, 0xe8, 0x01,
0xc0, 0x15, 0xb3, 0xb6, 0xb3, 0xad, 0x53, 0xd8,
0x7e, 0xea, 0xef, 0xa5, 0x0e, 0xbd, 0x97, 0xfc,
0xac, 0x15, 0xa5, 0x91, 0x4a, 0xc8, 0x9a, 0x4f,
0x60, 0x0e, 0x64, 0xa4, 0x85, 0x8d, 0x85, 0x0a,
0x6a, 0xd5, 0xe4, 0x67, 0x0a, 0x3a, 0x5b, 0x0e,
0xe7, 0xc3, 0xf5, 0x76, 0x34, 0x8e, 0x47, 0x15,
0x7b, 0x0f, 0xd0, 0x3b, 0xee, 0xd5, 0x9b, 0xa3,
0x9a, 0x01, 0xb5, 0x90, 0x9d, 0xe1, 0xf2, 0xa2,
0x35, 0xdd, 0x0b, 0xd8, 0x1d, 0xd7, 0xd6, 0x8f,
0x34, 0x5d, 0x69, 0x9b, 0xc3, 0xae, 0x29, 0xcf,
0x99, 0xf4, 0x94, 0x7f, 0x35, 0x92, 0x09, 0x21,
0x93, 0x35, 0xba, 0x25, 0x97, 0x9a, 0xc5, 0x6c,
0x64, 0xbe, 0xb1, 0x0a, 0x90, 0x4f, 0x3c, 0x16,
0xe5, 0x59, 0x07, 0xb4, 0x6e, 0x4b, 0x50, 0x54,
0x97, 0x53, 0xa5, 0x87, 0x9c, 0x4d, 0xb5, 0x96,
0x76, 0xd1, 0x7e, 0x3d, 0xd1, 0x60, 0xb0, 0x14,
0xc3, 0x43, 0xba, 0xdb, 0x2d, 0x16, 0x94, 0xdf,
0xc0, 0x97, 0x84, 0x58, 0xb4, 0xdc, 0xd3, 0x02,
0x83, 0xb8, 0x04, 0x94, 0xda, 0x66, 0x2f, 0x1e,
0xf6, 0xe1, 0x82, 0x59, 0x1d, 0xfe, 0x93, 0xe6,
0x92, 0xf7, 0x7d, 0xb9, 0x25, 0x97, 0xe2, 0x1c,
0x2f, 0x3a, 0x42, 0xb6, 0xac, 0x46, 0x1a, 0x37,
0xe8, 0xfd, 0x86, 0x51, 0xf2, 0x07, 0x46, 0xb3,
0xb2, 0x71, 0xf6, 0xd0, 0x3d, 0x7e, 0x0a, 0x09,
```

```
0x91, 0x96, 0xbb, 0xb6, 0x13, 0xa1, 0x04, 0xf4,
0x5b, 0x82, 0xe9, 0x69, 0xf1, 0xfd, 0xab, 0x06,
0x42, 0xda, 0xa9, 0x6a, 0xb7, 0x64, 0x30, 0x91
};
unsigned char group 18_responder[1024] = {
0xdc, 0x33, 0x0f, 0xaf, 0x4a, 0x8f, 0x0d, 0x35,
0xad, 0x20, 0x14, 0xfb, 0x37, 0x88, 0xee, 0xeb,
0xdb, 0x71, 0x4d, 0x4b, 0x2a, 0x1d, 0xff, 0x5e,
0xe7, 0x78, 0x2c, 0xa4, 0xc7, 0x6d, 0x4d, 0xb0,
0x39, 0xb3, 0xbf, 0xc4, 0x2e, 0xe6, 0x30, 0x66,
0x3d, 0x52, 0x2d, 0xf1, 0xce, 0x44, 0x6e, 0x1e,
0x89, 0x85, 0x97, 0x4b, 0xdc, 0x50, 0x0c, 0xaf,
0x59, 0xc8, 0xaf, 0x7f, 0xbb, 0x67, 0xb3, 0x68,
0xf3, 0xf9, 0x10, 0x30, 0xde, 0x27, 0x2d, 0x83,
0xc5, 0x9a, 0xa9, 0xc1, 0x06, 0x36, 0xb0, 0xdd,
0xe5, 0x55, 0x1b, 0xbf, 0x7c, 0xbf, 0x2d, 0xca,
0x8d, 0x84, 0x4d, 0x55, 0x44, 0x28, 0xe4, 0xcc,
0xb4, 0xb8, 0x26, 0xd5, 0x11, 0x91, 0xdb, 0xf8,
0x1b, 0x57, 0x17, 0xad, 0x3e, 0x2b, 0xee, 0x7a,
0x7a, 0xdc, 0xc5, 0x46, 0xc9, 0x88, 0xf1, 0x39,
0x9e, 0xf3, 0xca, 0x6d, 0x09, 0x11, 0xe4, 0xcc,
0x31, 0x03, 0x80, 0xd1, 0x7d, 0xc8, 0xf7, 0xc5,
0x10, 0x78, 0x5d, 0x15, 0xa8, 0xe3, 0x55, 0xd2,
0x01, 0x54, 0x96, 0x99, 0xaf, 0x18, 0x33, 0x8c,
0x71, 0xf9, 0x25, 0x7a, 0xc0, 0xa5, 0xfd, 0x61,
0xf1, 0x04, 0xc5, 0x22, 0xfa, 0xa6, 0xdd, 0xc1,
0x13, 0xf4, 0x2b, 0x39, 0xa8, 0x17, 0x8d, 0x68,
0xb0, 0xe9, 0x70, 0x12, 0xda, 0x60, 0x47, 0x2b,
0x17, 0xf4, 0x14, 0x53, 0xad, 0x29, 0x1c, 0xa5,
0x07, 0x43, 0x91, 0xe9, 0xfb, 0xf4, 0x5d, 0x4c,
0xe5, 0xb8, 0x07, 0x27, 0x37, 0x03, 0x39, 0xf8,
0x28, 0x2d, 0xab, 0x2f, 0x5a, 0x1d, 0x41, 0xa3,
0x38, 0x2e, 0x42, 0xe6, 0xe2, 0x32, 0xf9, 0x75,
0xca, 0x19, 0x80, 0x0d, 0xd1, 0x15, 0x73, 0x45,
0xda, 0x8a, 0x67, 0x7a, 0x3c, 0xfd, 0x6b, 0x2d,
0x46, 0xa4, 0xd0, 0xd2, 0x8d, 0x12, 0x2d, 0x54,
0x5d, 0x1d, 0xa7, 0xc3, 0x44, 0x98, 0x4f, 0x6d,
0x83, 0xbf, 0x33, 0xf8, 0x51, 0xf2, 0x29, 0xa3,
0x48, 0x26, 0x43, 0x26, 0xfa, 0x3a, 0x4a, 0x48,
0x6a, 0xac, 0x0d, 0x2c, 0xb5, 0x89, 0xec, 0xff,
0xc3, 0x6f, 0x28, 0x54, 0xe6, 0x54, 0x35, 0x5f,
0x93, 0xb7, 0x9e, 0xfa, 0x04, 0x1b, 0x31, 0x5d,
0xe0, 0x58, 0x4a, 0x8d, 0x54, 0xb9, 0x63, 0x72,
0x4a, 0x68, 0x5f, 0x9d, 0x1b, 0xde, 0xdf, 0xbb,
0xae, 0x9b, 0xb4, 0x65, 0x91, 0x93, 0x91, 0x9f,
0xd9, 0xb5, 0xbc, 0x41, 0x32, 0xd3, 0x37, 0x95,
0xb1, 0x0e, 0xec, 0xe5, 0x18, 0x38, 0xf9, 0xbe,
0xc9, 0xf9, 0xc1, 0x5c, 0x18, 0x9e, 0xd0, 0x56,
```

```
0x35, 0xe1, 0xf2, 0xd1, 0xeb, 0x09, 0x18, 0xc4,
0xe3, 0x56, 0x10, 0x47, 0x3c, 0xb4, 0x8b, 0xcc,
0xf0, 0xab, 0x4c, 0xcd, 0xdb, 0x6c, 0xa2, 0x39,
0xb9, 0x32, 0xee, 0x57, 0x33, 0x9e, 0x9b, 0xc8,
0x30, 0xa1, 0x60, 0x3f, 0xd0, 0xdb, 0xf3, 0xb5,
0x14, 0x9f, 0xab, 0x9e, 0xaf, 0xd6, 0x88, 0x12,
0x26, 0xaf, 0xf6, 0x3b, 0x4c, 0x20, 0xf9, 0xae,
0xa4, 0x85, 0x7b, 0x07, 0x8a, 0x1e, 0x10, 0x90,
0x29, 0x9e, 0xba, 0x63, 0x54, 0x3f, 0x0e, 0xbe,
0x95, 0x39, 0x22, 0x11, 0xe8, 0xff, 0xda, 0x08,
0xd6, 0x6e, 0xb3, 0xd5, 0xcc, 0xbd, 0x4f, 0x8a,
0x1d, 0x37, 0x0f, 0x9b, 0x6d, 0xda, 0x9d, 0xd0,
0x46, 0x4d, 0x0f, 0x57, 0x06, 0x9f, 0x15, 0x53,
0x52, 0x1b, 0xfe, 0x2c, 0x52, 0xc4, 0xab, 0x80,
0x29, 0xce, 0x64, 0x1a, 0x7a, 0xd2, 0x98, 0x56,
0xcc, 0x90, 0xed, 0x6d, 0x1e, 0x9c, 0xe8, 0x5d,
0xcf, 0x55, 0x89, 0x14, 0x3d, 0x38, 0x46, 0x68,
0x67, 0xbe, 0x09, 0x9c, 0x9a, 0xdd, 0x74, 0xeb,
0xde, 0xc8, 0x65, 0x71, 0x1a, 0xd7, 0x35, 0xa5,
0xee, 0xf9, 0xbd, 0xe0, 0xf9, 0x04, 0xb5, 0x8e,
0xc9, 0x42, 0xc4, 0xa4, 0x5e, 0x2c, 0xa6, 0x4e,
0x19, 0x8d, 0x45, 0x24, 0x28, 0x02, 0xbe, 0x7f,
0xea, 0xd7, 0xc1, 0x99, 0x04, 0x72, 0xf5, 0xb3,
0x06, 0x2d, 0xaf, 0xf6, 0x37, 0x08, 0x22, 0x82,
0xd7, 0xbb, 0x5a, 0xea, 0x72, 0x23, 0xce, 0xa8,
0x2e, 0x73, 0x76, 0x9c, 0x3b, 0xa7, 0x23, 0xfa,
0x82, 0x9f, 0xd8, 0x6b, 0x75, 0x57, 0xab, 0x5e,
0x49, 0xcb, 0x29, 0x55, 0xe5, 0x55, 0xb1, 0x40,
0x41, 0xa0, 0x36, 0xdc, 0xff, 0xc5, 0xd3, 0x8a,
0xa8, 0x1d, 0xd2, 0x15, 0x9b, 0xc5, 0x84, 0xb7,
0xea, 0x88, 0x85, 0xfa, 0x82, 0x75, 0x16, 0xbf,
0x7f, 0xc9, 0x8f, 0xa0, 0x76, 0x90, 0x8c, 0x99,
0x7e, 0xd9, 0x8d, 0xd6, 0x88, 0x08, 0x43, 0xd3,
0x50, 0x98, 0x06, 0xda, 0x78, 0x9a, 0xdd, 0x71,
0x7a, 0x61, 0xd2, 0x4c, 0xc8, 0xf0, 0xc1, 0x52,
0x1c, 0x09, 0x7f, 0xe7, 0xba, 0x4e, 0x11, 0x39,
0x06, 0xb8, 0xe9, 0xa1, 0xb0, 0x12, 0x9b, 0x6b,
0xda, 0x90, 0x5e, 0x24, 0x54, 0x54, 0x87, 0xbb,
0x69, 0x07, 0x5d, 0xf2, 0x65, 0xb3, 0xf8, 0x7e,
0xea, 0xa5, 0xe5, 0x3c, 0xe9, 0x4b, 0x31, 0x47,
0x79, 0x44, 0x74, 0x3b, 0x96, 0x1e, 0x1c, 0xbd,
0x8a, 0xb2, 0x6f, 0x89, 0xd1, 0x60, 0xb8, 0x06,
0xa7, 0x05, 0x1e, 0xe0, 0x6b, 0x26, 0xa1, 0xd2,
0x85, 0xe9, 0x0a, 0x7d, 0x92, 0x26, 0xdd, 0xb0,
0xa9, 0x51, 0x9b, 0x5d, 0xfa, 0x0a, 0x19, 0xe4,
0x0d, 0xdb, 0xfb, 0x94, 0x60, 0x89, 0x3e, 0x49,
0x10, 0x75, 0x5c, 0xa0, 0x1e, 0x52, 0xad, 0xf9,
0x06, 0x04, 0x15, 0x91, 0xb0, 0x42, 0x67, 0x3a,
```

```
0x1e, 0x43, 0x87, 0xdc, 0x06, 0x4c, 0x54, 0x37,
0x41, 0xbf, 0x6f, 0x5b, 0xd5, 0xc1, 0xd9, 0x7b,
0xdc, 0x25, 0x78, 0xa6, 0x6c, 0x56, 0x91, 0x0d,
0x57, 0x4e, 0x6c, 0x1f, 0xc7, 0xab, 0xec, 0x53,
0xbd, 0x4a, 0x1e, 0xee, 0x3e, 0x5e, 0x74, 0xe8,
0x4c, 0x37, 0x46, 0xe3, 0xa7, 0x55, 0xce, 0x16,
0x35, 0x6b, 0xbe, 0x43, 0x9d, 0xef, 0x7c, 0x6b,
0x77, 0xb9, 0xc7, 0xda, 0x16, 0x6d, 0x7e, 0x4a,
0x95, 0x09, 0x33, 0x70, 0x37, 0xe0, 0xe2, 0x44,
0xad, 0x6b, 0xa2, 0x44, 0xe7, 0x96, 0x07, 0x18,
0x19, 0xf1, 0x88, 0x2d, 0x47, 0x36, 0x6e, 0x57,
0x48, 0xa3, 0x7a, 0x3b, 0x00, 0x70, 0x76, 0x52,
0xe3, 0xd4, 0xa5, 0xac, 0x8d, 0x37, 0x2c, 0xd1,
0xbe, 0x5c, 0x7c, 0x6c, 0xda, 0x86, 0x0b, 0x02,
0x86, 0xcc, 0xbf, 0x44, 0x46, 0x69, 0xca, 0x86,
0xa6, 0x9d, 0x98, 0xd5, 0xd7, 0x84, 0x57, 0xa5,
0x90, 0x63, 0x49, 0x14, 0x48, 0x03, 0x25, 0x33,
0x8f, 0xd5, 0xc2, 0x71, 0xfa, 0x6c, 0xf0, 0x9b,
0x9e, 0xf0, 0x2a, 0xb2, 0x0c, 0x2d, 0x31, 0x5e,
0xda, 0x33, 0x88, 0x72, 0xe7, 0x5a, 0x56, 0xbc,
0xfd, 0x63, 0x70, 0xf6, 0xa3, 0xc2, 0x46, 0xec,
0x57, 0x53, 0xfa, 0x09, 0x7d, 0x61, 0xc1, 0x56,
0xa5, 0xee, 0x57, 0x47, 0x81, 0x9b, 0x7f, 0x7b,
0x33, 0xdf, 0x20, 0xf6, 0x84, 0x24, 0x6e, 0xb6,
0x29, 0xcc, 0xe5, 0x9c, 0x3e, 0x23, 0xd3, 0x0a,
0x29, 0xf4, 0x46, 0x68, 0xb2, 0x88, 0xcc, 0x22,
0x95, 0x70, 0xd3, 0x36, 0x0f, 0x60, 0xcd, 0xf1,
0x0d, 0xfa, 0xcd, 0x22, 0x22, 0x8a, 0x30, 0x6b,
0xc4, 0x44, 0x46, 0xe8, 0xf6, 0xad, 0x8f, 0x32,
0x9a, 0x05, 0x30, 0x94, 0xbb, 0x47, 0x52, 0xd3,
0x6f, 0x42, 0xe1, 0x6f, 0x50, 0xd1, 0x63, 0x0c,
0x74, 0x6f, 0x75, 0x02, 0x40, 0x29, 0xab, 0x74,
0xeb, 0x61, 0xb8, 0x09, 0xe2, 0x16, 0xf0, 0x48,
0x8e, 0xd0, 0xf3, 0x9e, 0x11, 0x6c, 0x6e, 0xf1,
0xe8, 0x73, 0x8f, 0x38, 0xba, 0x9c, 0x9b, 0x16,
0xc0, 0xdf, 0x33, 0xb7, 0x29, 0x7b, 0xa6, 0x39,
0xa5, 0x86, 0x7d, 0xda, 0xb7, 0xf9, 0x9e, 0x88
};
```

Author's Address

Dan Harkins
HP Enterprise
3333 Scott boulevard
Santa Clara, California 95054
United States of America

Phone: +1 415 997 9834
Email: dharkins@lounge.org