Internet Engineering Task Force (IETF)
Request for Comments: 6278
Category: Informational
ISSN: 2070-1721

J. Herzog
R. Khazan
MIT Lincoln Laboratory

June 2011

Use of Static-Static Elliptic Curve Diffie-Hellman Key Agreement in Cryptographic Message Syntax

Abstract

This document describes how to use the 'static-static Elliptic Curve Diffie-Hellman key-agreement scheme (i.e., Elliptic Curve Diffie-Hellman where both participants use static Diffie-Hellman values) with the Cryptographic Message Syntax. In this form of key agreement, the Diffie-Hellman values of both the sender and receiver are long-term values contained in certificates.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6278.

Table of Contents

1. Introduction

This document describes how to use the static-static Elliptic Curve Diffie-Hellman key-agreement scheme (i.e., Elliptic Curve Diffie-Hellman [RFC6090] where both participants use static Diffie-Hellman values) in the Cryptographic Message Syntax (CMS) [RFC5652]. The CMS is a standard notation and representation for cryptographic messages. The CMS uses ASN.1 notation [X.680] [X.681] [X.682] [X.683] to define

a number of structures that carry both cryptographically protected information and key-management information regarding the keys used. Of particular interest here are three structures:

- EnvelopedData, which holds encrypted (but not necessarily authenticated) information [RFC5652],
- AuthenticatedData, which holds authenticated (MACed) information [RFC5652], and
- AuthEnvelopedData, which holds information protected by authenticated encryption: a cryptographic scheme that combines encryption and authentication [RFC5083].

All three of these types share the same basic structure. First, a fresh symmetric key is generated. This symmetric key has a different name that reflects its usage in each of the three structures. EnvelopedData uses a content-encryption key (CEK); AuthenticatedData uses an authentication key; AuthEnvelopedData uses a content-authenticated-encryption key. The originator uses the symmetric key to cryptographically protect the content. The symmetric key is then wrapped for each recipient; only the intended recipient has access to the private keying material necessary to unwrap the symmetric key. Once unwrapped, the recipient uses the symmetric key to decrypt the content, check the authenticity of the content, or both. The CMS supports several different approaches to symmetric key wrapping, including:

- key transport: the symmetric key is encrypted using the public encryption key of some recipient,
- key-encryption key: the symmetric key is encrypted using a previously distributed symmetric key, and
- key agreement: the symmetric key is encrypted using a key-encryption key (KEK) created using a key-agreement scheme and a key-derivation function (KDF).

One such key-agreement scheme is the Diffie-Hellman algorithm [RFC2631], which uses group theory to produce a value known only to its two participants. In this case, the participants are the originator and one of the recipients. Each participant produces a private value and a public value, and each participant can produce the shared secret value from their own private value and their counterpart’s public value. There are some variations on the basic algorithm:

- The basic algorithm typically uses the group 'Z mod p', meaning the set of integers modulo some prime p. One can also use an elliptic curve group, which allows for shorter messages.

- Over elliptic curve groups, the standard algorithm can be extended to incorporate the 'cofactor' of the group. This method, called 'cofactor Elliptic Curve Diffie-Hellman' [SP800-56A] can prevent certain attacks possible in the elliptic curve group.

- The participants can generate fresh new public/private values (called ephemeral values) for each run of the algorithm, or they can re-use long-term values (called static values). Ephemeral values add randomness to the resulting private value, while static values can be embedded in certificates. The two participants do not need to use the same kind of value: either participant can use either type. In 'ephemeral-static' Diffie-Hellman, for example, the sender uses an ephemeral public/private pair value while the receiver uses a static pair. In 'static-static' Diffie-Hellman, on the other hand, both participants use static pairs. (Receivers cannot use ephemeral values in this setting, and so we ignore ephemeral-ephemeral and static-ephemeral Diffie-Hellman in this document.)

Several of these variations are already described in existing CMS standards; for example, [RFC3370] contains the conventions for using ephemeral-static and static-static Diffie-Hellman over the 'basic' (Z mod p) group. [RFC5753] contains the conventions for using ephemeral-static Diffie-Hellman over elliptic curves (both standard and cofactor methods). It does not, however, contain conventions for using either method of static-static Elliptic Curve Diffie-Hellman, preferring to discuss the Elliptic Curve Menezes-Qu-Vanstone (ECMQV) algorithm instead.

In this document, we specify the conventions for using static-static Elliptic Curve Diffie-Hellman (ECDH) for both standard and cofactor methods. Our motivation stems from the fact that ECMQV has been removed from the National Security Agency's Suite B of cryptographic algorithms and will therefore be unavailable to some participants. These participants can use ephemeral-static Elliptic Curve Diffie-Hellman, of course, but ephemeral-static Diffie-Hellman does not provide source authentication. The CMS does allow the application of digital signatures for source authentication, but this alternative is available only to those participants with certified signature keys. By specifying conventions for static-static Elliptic Curve Diffie-Hellman in this document, we present a third alternative for source authentication, available to those participants with certified Elliptic Curve Diffie-Hellman keys.

We note that like ephemeral-static ECDH, static-static ECDH creates a secret key shared by the sender and receiver. Unlike ephemeral-static ECDH, however, static-static ECDH uses a static key pair for the sender. Each of the three CMS structures discussed in this document (EnvelopedData, AuthenticatedData, and AuthEnvelopedData) uses static-static ECDH to achieve different goals:

- EnvelopedData uses static-static ECDH to provide data confidentiality. It will not necessarily, however, provide data authenticity.
- AuthenticatedData uses static-static ECDH to provide data authenticity. It will not provide data confidentiality.
- AuthEnvelopedData uses static-static ECDH to provide both confidentiality and data authenticity.

1.1. 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].

2. EnvelopedData Using Static-Static ECDH

If an implementation uses static-static ECDH with the CMS EnvelopedData, then the following techniques and formats MUST be used. The fields of EnvelopedData are as in [RFC5652]; as static-static ECDH is a key-agreement algorithm, the RecipientInfo 'kari' choice is used. When using static-static ECDH, the EnvelopedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key.

2.1. Fields of the KeyAgreeRecipientInfo

When using static-static ECDH with EnvelopedData, the fields of KeyAgreeRecipientInfo [RFC5652] are as follows:

- version MUST be 3.
- originator identifies the static EC public key of the sender. It MUST be either issuerAndSerialNumber or subjectKeyIdentifier, and it MUST point to one of the sending agent's certificates.
- ukm MAY be present or absent. However, message originators SHOULD include the ukm and SHOULD ensure that the value of ukm is unique to the message being sent. As specified in [RFC5652], implementations MUST support ukm message recipient processing, so

interoperability is not a concern if the ukm is present or absent. The use of a fresh value for ukm will ensure that a different key is generated for each message between the same sender and receiver. The ukm, if present, is placed in the entityUInfo field of the ECC-CMS-SharedInfo structure [RFC5753] and therefore used as an input to the key-derivation function.

- keyEncryptionAlgorithm MUST contain the object identifier of the key-encryption algorithm, which in this case is a key-agreement algorithm (see Section 5). The parameters field contains KeyWrapAlgorithm. The KeyWrapAlgorithm is the algorithm identifier that indicates the symmetric encryption algorithm used to encrypt the content-encryption key (CEK) with the key-encryption key (KEK) and any associated parameters (see Section 5).

- recipientEncryptedKeys contains an identifier and an encrypted CEK for each recipient. The RecipientEncryptedKey KeyAgreeRecipientIdentifier MUST contain either the issuerAndSerialNumber identifying the recipient's certificate or the RecipientKeyIdentifier containing the subject key identifier from the recipient's certificate. In both cases, the recipient's certificate contains the recipient's static ECDH public key. RecipientEncryptedKey EncryptedKey MUST contain the content-encryption key encrypted with the static-static ECDH-generated pairwise key-encryption key using the algorithm specified by the KeyWrapAlgorithm.

2.2. Actions of the Sending Agent

When using static-static ECDH with EnvelopedData, the sending agent first obtains the EC public key(s) and domain parameters contained in the recipient's certificate. It MUST confirm the following at least once per recipient-certificate:

- that both certificates (the recipient's certificate and its own) contain public-key values with the same curve parameters, and

- that both of these public-key values are marked as appropriate for ECDH (that is, marked with algorithm identifiers id-ecPublicKey or id-ecDH [RFC5480]).

The sender then determines whether to use standard or cofactor Diffie-Hellman. After doing so, the sender then determines which hash algorithms to use for the key-derivation function. It then chooses the keyEncryptionAlgorithm value that reflects these choices. It then determines:

- an integer "keydatalen", which is the KeyWrapAlgorithm symmetric key size in bits, and

- the value of ukm, if used.

The sender then determines a bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 7.2 of [RFC5753]). The sending agent then performs either the Elliptic Curve Diffie-Hellman operation of [RFC6090] (for standard Diffie-Hellman) or the Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive of [SP800-56A] (for cofactor Diffie-Hellman). The sending agent then applies the simple hash-function construct of [X963] (using the hash algorithm identified in the key-agreement algorithm) to the results of the Diffie-Hellman operation and the SharedInfo string. (This construct is also described in Section 3.6.1 of [SEC1].) As a result, the sending agent obtains a shared secret bit string "K", which is used as the pairwise key-encryption key (KEK) to wrap the CEK for that recipient, as specified in [RFC5652].

2.3. Actions of the Receiving Agent

When using static-static ECDH with EnvelopedData, the receiving agent retrieves keyEncryptionAlgorithm to determine the key-agreement algorithm chosen by the sender, which will identify:

- the domain parameters of the curve used,

- whether standard or cofactor Diffie-Hellman was used, and

- which hash function was used for the KDF.

The receiver then retrieves the sender’s certificate identified in the rid field and extracts the EC public key(s) and domain parameters contained therein. It MUST confirm the following at least once per sender certificate:

- that both certificates (the sender’s certificate and its own) contain public-key values with the same curve parameters, and

- that both of these public-key values are marked as appropriate for ECDH (that is, marked with algorithm identifiers id-ecPublicKey or id-ecDH [RFC5480]).

The receiver then determines whether standard or cofactor Diffie-Hellman was used. The receiver then determines a bit string "SharedInfo", which is the DER encoding of ECC-CMS-SharedInfo (see Section 7.2 of [RFC5753]). The receiving agent then performs either the Elliptic Curve Diffie-Hellman operation of [RFC6090] (for

standard Diffie-Hellman) or the Elliptic Curve Cryptography Cofactor Diffie-Hellman (ECC CDH) Primitive of [SP800-56A] (for cofactor Diffie-Hellman). The receiving agent then applies the simple hash-function construct of [X963] (using the hash algorithm identified in the key-agreement algorithm) to the results of the Diffie-Hellman operation and the SharedInfo string. (This construct is also described in Section 3.6.1 of [SEC1].) As a result, the receiving agent obtains a shared secret bit string "K", which it uses as the pairwise key-encryption key to unwrap the CEK.

3. AuthenticatedData Using Static-Static ECDH

This section describes how to use the static-static ECDH key-agreement algorithm with AuthenticatedData. When using static-static ECDH with AuthenticatedData, the fields of AuthenticatedData are as in [RFC5652], but with the following restrictions:

- macAlgorithm MUST contain the algorithm identifier of the message authentication code (MAC) algorithm. This algorithm SHOULD be one of the following -- id-hmacWITHSHA224, id-hmacWITHSHA256, id-hmacWITHSHA384, or id-hmacWITHSHA512 -- and SHOULD NOT be hmac-SHA1. (See Section 5.)

- digestAlgorithm MUST contain the algorithm identifier of the hash algorithm. This algorithm SHOULD be one of the following -- id-sha224, id-sha256, id-sha384, or id-sha512 -- and SHOULD NOT be id-sha1. (See Section 5.)

As static-static ECDH is a key-agreement algorithm, the RecipientInfo kari choice is used in the AuthenticatedData. When using static-static ECDH, the AuthenticatedData originatorInfo field MAY include the certificate(s) for the EC public key(s) used in the formation of the pairwise key.

3.1. Fields of the KeyAgreeRecipientInfo

The AuthenticatedData KeyAgreeRecipientInfo fields are used in the same manner as the fields for the corresponding EnvelopedData KeyAgreeRecipientInfo fields of Section 2.1 of this document. The authentication key is wrapped in the same manner as is described there for the content-encryption key.

3.2. Actions of the Sending Agent

The sending agent uses the same actions as for EnvelopedData with static-static ECDH, as specified in Section 2.2 of this document.

3.3. Actions of the Receiving Agent

The receiving agent uses the same actions as for EnvelopedData with static-static ECDH, as specified in Section 2.3 of this document.

4. AuthEnvelopedData Using Static-Static ECDH

When using static-static ECDH with AuthEnvelopedData, the fields of AuthEnvelopedData are as in [RFC5083]. As static-static ECDH is a key-agreement algorithm, the RecipientInfo kari choice is used. When using static-static ECDH, the AuthEnvelopedData originatorInfo field MAY include the certificate(s) for the EC public key used in the formation of the pairwise key.

4.1. Fields of the KeyAgreeRecipientInfo

The AuthEnvelopedData KeyAgreeRecipientInfo fields are used in the same manner as the fields for the corresponding EnvelopedData KeyAgreeRecipientInfo fields of Section 2.1 of this document. The content-authenticated-encryption key is wrapped in the same manner as is described there for the content-encryption key.

4.2. Actions of the Sending Agent

The sending agent uses the same actions as for EnvelopedData with static-static ECDH, as specified in Section 2.2 of this document.

4.3. Actions of the Receiving Agent

The receiving agent uses the same actions as for EnvelopedData with static-static ECDH, as specified in Section 2.3 of this document.

5. Comparison to RFC 5753

This document defines the use of static-static ECDH for EnvelopedData, AuthenticatedData, and AuthEnvelopedData. [RFC5753] defines ephemeral-static ECDH for EnvelopedData only.

With regard to EnvelopedData, this document and [RFC5753] greatly parallel each other. Both specify how to apply Elliptic Curve Diffie-Hellman and differ only on how the sender's public value is to be communicated to the recipient. In [RFC5753], the sender provides the public value explicitly by including an OriginatorPublicKey value in the originator field of KeyAgreeRecipientInfo. In this document, the sender includes a reference to a (certified) public value by including either an IssuerAndSerialNumber or SubjectKeyIdentifier value in the same field. Put another way, [RFC5753] provides an interpretation of a KeyAgreeRecipientInfo structure where:

- the keyEncryptionAlgorithm value indicates Elliptic Curve Diffie-Hellman, and

- the originator field contains an OriginatorPublicKey value.

This document, on the other hand, provides an interpretation of a KeyAgreeRecipientInfo structure where:

- the keyEncryptionAlgorithm value indicates Elliptic Curve Diffie-Hellman, and

- the originator field contains either an IssuerAndSerialNumber value or a SubjectKeyIdentifier value.

AuthenticatedData or AuthEnvelopedData messages, on the other hand, are not given any form of ECDH by [RFC5753]. This is appropriate: that document only defines ephemeral-static Diffie-Hellman, and this form of Diffie-Hellman does not (inherently) provide any form of data authentication or data-origin authentication. This document, on the other hand, requires that the sender use a certified public value. Thus, this form of key agreement provides implicit key authentication and, under some limited circumstances, data-origin authentication. (See Section 7.)

This document does not define any new ASN.1 structures or algorithm identifiers. It provides new ways to interpret structures from [RFC5652] and [RFC5753], and it allows previously defined algorithms to be used under these new interpretations. Specifically:

- The ECDH key-agreement algorithm identifiers from [RFC5753] define only how Diffie-Hellman values are processed, and not where these values are created. Therefore, they can be used for static-static ECDH with no changes.

- The key-wrap, MAC, and digest algorithms referenced in [RFC5753] describe how the secret key is to be used but not created. Therefore, they can be used with keys from static-static ECDH without modification.

6. Requirements and Recommendations

It is RECOMMENDED that implementations of this specification support AuthenticatedData and EnvelopedData. Support for AuthEnvelopedData is OPTIONAL.

Implementations that support this specification MUST support standard Elliptic Curve Diffie-Hellman, and these implementations MAY also support cofactor Elliptic Curve Diffie-Hellman.

In order to encourage interoperability, implementations SHOULD use the elliptic curve domain parameters specified by [RFC5480].

Implementations that support standard static-static Elliptic Curve Diffie-Hellman:

- MUST support the dhSinglePass-stdDH-sha256kdf-scheme key-agreement algorithm;
- MAY support the dhSinglePass-stdDH-sha224kdf-scheme, dhSinglePass-stdDH-sha384kdf-scheme, and dhSinglePass-stdDH-sha512kdf-scheme key-agreement algorithms; and
- SHOULD NOT support the dhSinglePass-stdDH-sha1kdf-scheme algorithm.

Other algorithms MAY also be supported.

Implementations that support cofactor static-static Elliptic Curve Diffie-Hellman:

- MUST support the dhSinglePass-cofactorDH-sha256kdf-scheme key-agreement algorithm;
- MAY support the dhSinglePass-cofactorDH-sha224kdf-scheme, dhSinglePass-cofactorDH-sha384kdf-scheme, and dhSinglePass-cofactorDH-sha512kdf-scheme key-agreement algorithms; and
- SHOULD NOT support the dhSinglePass-cofactorDH-sha1kdf-scheme algorithm.

In addition, all implementations:

- MUST support the id-aes128-wrap key-wrap algorithm and the id-aes128-cbc content-encryption algorithm;
- MAY support:
  - the id-aes192-wrap and id-aes256-wrap key-wrap algorithms;
  - the id-aes128-CCM, id-aes192-CCM, id-aes256-CCM, id-aes128-GCM, id-aes192-GCM, and id-aes256-GCM authenticated-encryption algorithms; and
  - the id-aes192-cbc and id-aes256-cbc content-encryption algorithms.

- SHOULD NOT support the id-alg-CMS3DESwrap key-wrap algorithm or the des-ede3-cbc content-encryption algorithms.

(All algorithms above are defined in [RFC3370], [RFC3565], [RFC5084], and [RFC5753].) Unless otherwise noted above, other algorithms MAY also be supported.

7. Security Considerations

All security considerations in Section 9 of [RFC5753] apply.

Extreme care must be used when using static-static Diffie-Hellman (either standard or cofactor) without the use of some per-message value in the ukm. As described in [RFC5753], the ukm value (if present) will be embedded in an ECC-CMS-SharedInfo structure, and the DER encoding of this structure will be used as the 'SharedInfo' input to the key-derivation function of [X963]. The purpose of this input is to add a message-unique value to the key-distribution function so that two different sessions of static-static ECDH between a given pair of agents result in independent keys. If the ukm value is not used or is re-used, on the other hand, then the ECC-CMS-SharedInfo structure (and 'SharedInfo' input) will likely not vary from message to message. In this case, the two agents will re-use the same keying material across multiple messages. This is considered to be bad cryptographic practice and may open the sender to attacks on Diffie-Hellman (e.g., the 'small subgroup' attack [MenezesUstaoglu] or other, yet-undiscovered attacks).

It is for these reasons that Section 2.1 states that message senders SHOULD include the ukm and SHOULD ensure that the value of ukm is unique to the message being sent. One way to ensure the uniqueness of the ukm is for the message sender to choose a 'sufficiently long' random string for each message (where, as a rule of thumb, a 'sufficiently long' string is one at least as long as the keys used by the key-wrap algorithm identified in the keyEncryptionAlgorithm field of the KeyAgreeRecipientInfo structure). However, other methods (such as a counter) are possible. Also, applications that cannot tolerate the inclusion of per-message information in the ukm (due to bandwidth requirements, for example) SHOULD NOT use static-static ECDH for a recipient without ascertaining that the recipient knows the private value associated with their certified Diffie-Hellman value.

Static-static Diffie-Hellman, when used as described in this document, does not necessarily provide data-origin authentication. Consider, for example, the following sequence of events:

- Alice sends an AuthEnvelopedData message to both Bob and Mallory. Furthermore, Alice uses a static-static DH method to transport the content-authenticated-encryption key to Bob, and some arbitrary method to transport the same key to Mallory.

- Mallory intercepts the message and prevents Bob from receiving it.

- Mallory recovers the content-authenticated-encryption key from the message received from Alice. Mallory then creates new plaintext of her choice, and encrypts it using the same authenticated-encryption algorithm and the same content-authenticated-encryption key used by Alice.

- Mallory then replaces the EncryptedContentInfo and MessageAuthenticationCode fields of Alice's message with the values Mallory just generated. She may additionally remove her RecipientInfo value from Alice's message.

- Mallory sends the modified message to Bob.

- Bob receives the message, validates the static-static DH values, and decrypts/authenticates the message.

At this point, Bob has received and validated a message that appears to have been sent by Alice, but whose content was chosen by Mallory. Mallory may not even be an apparent receiver of the modified message. Thus, this use of static-static Diffie-Hellman does not necessarily provide data-origin authentication. (We note that this example does not also contradict either confidentiality or data authentication: Alice's message was not received by anyone not intended by Alice, and Mallory's message was not modified before reaching Bob.)

More generally, the data origin may not be authenticated unless:

- it is a priori guaranteed that the message in question was sent to exactly one recipient, or

- data-origin authentication is provided by some other mechanism (such as digital signatures).

However, we also note that this lack of authentication is not a product of static-static ECDH per se, but is inherent in the way key-agreement schemes are used in the AuthenticatedData and AuthEnvelopedData structures of the CMS.

When two parties are communicating using static-static ECDH as described in this document, and either party's asymmetric keys have been centrally generated, it is possible for that party's central

infrastructure to decrypt the communication (for application-layer network monitoring or filtering, for example). By way of contrast: were ephemeral-static ECDH to be used instead, such decryption by the sender's infrastructure would not be possible (though it would remain possible for the infrastructure of any recipient).

8. Acknowledgements and Disclaimer

This work is sponsored by the United States Air Force under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

The authors would like to thank Jim Schaad, Russ Housley, Sean Turner, Brian Weis, Rene Struik, Brian Carpenter, David McGrew, and Stephen Farrell for their helpful comments and suggestions. We would also like to thank Jim Schaad for describing to us the attack described in Section 7.

9. References

9.1. Normative References

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

[RFC3370] Housley, R., "Cryptographic Message Syntax (CMS) Algorithms", RFC 3370, August 2002.

[RFC3565] Schaad, J., "Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS)", RFC 3565, July 2003.

[RFC5083] Housley, R., "Cryptographic Message Syntax (CMS) Authenticated-Enveloped-Data Content Type", RFC 5083, November 2007.

[RFC5084] Housley, R., "Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)", RFC 5084, November 2007.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC5753] Turner, S. and D. Brown, "Use of Elliptic Curve Cryptography (ECC) Algorithms in Cryptographic Message Syntax (CMS)", RFC 5753, January 2010.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[SP800-56A] Barker, E., Johnson, D., and M. Smid, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography (Revised)", NIST Special Publication (SP) 800-56A, March 2007.

[X963] "Public Key Cryptography for the Financial Services Industry, Key Agreement and Key Transport Using Elliptic Curve Cryptography", ANSI X9.63, 2001.

9.2. Informative References

[MenezesUstaoglu] Menezes, A. and B. Ustaoglu, "On Reusing Ephemeral Keys in Diffie-Hellman Key Agreement Protocols", International Journal of Applied Cryptography, Vol. 2, No. 2, pp. 154-158, 2010.

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

[SEC1] Standards for Efficient Cryptography Group (SECG), "SEC 1: Elliptic Curve Cryptography", Version 2.0, May 2009.

[X.680] ITU-T, "Information Technology - Abstract Syntax Notation One: Specification of Basic Notation", Recommendation X.680, ISO/IEC 8824-1:2002, 2002.

[X.681] ITU-T, "Information Technology - Abstract Syntax Notation One: Information Object Specification", Recommendation X.681, ISO/IEC 8824-2:2002, 2002.

[X.682] ITU-T, "Information Technology - Abstract Syntax Notation One: Constraint Specification", Recommendation X.682, ISO/IEC 8824-3:2002, 2002.

[X.683] ITU-T, "Information Technology - Abstract Syntax Notation One: Parameterization of ASN.1 Specifications", Recommendation X.683, ISO/IEC 8824-4:2002, 2002.

Authors' Addresses

Jonathan C. Herzog
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02144
USA

EMail: jherzog@ll.mit.edu

Roger Khazan
MIT Lincoln Laboratory
244 Wood St.
Lexington, MA 02144
USA

EMail: rkh@ll.mit.edu