Standards for Efficient Cryptography

SEC 3: Elliptic Curve Signature Schemes with Partial Message Recovery: ECPVS and ECAOS

Certicom Research

Contact: Koray Karabina (kkarabina@certicom.com)

Working Draft
June 1, 2011
Version 0.5

Contents

1 Introduction

This section gives an overview of this standard, its use, its aims, and its development.

1.1 Overview

This document specifies public-key cryptographic schemes based on elliptic curve cryptography (ECC). In particular, it specifies:

- signature schemes providing partial message recovery.

It also describes cryptographic primitives which are used to construct the schemes, and ASN.1 syntax for identifying the schemes.

The schemes are intended for general application within computer and communications systems.

1.2 Aim

The aim of this document is threefold:

- First, to facilitate deployment of ECC by completely specifying efficient, well-established, and well-understood public-key cryptographic signature schemes providing partial message recovery based on ECC.
- Second, to encourage deployment of interoperable implementations of ECC by profiling standards such as ANSI X9.92 [X9.92] and IEEE 1363A [1363A], but restricting the options allowed in these standards to increase the likelihood of interoperability and to ensure conformance with as many standards as possible.
- Third, to help ensure ongoing detailed analysis of ECC by cryptographers by clearly, completely, and publicly specifying baseline techniques.

1.3 Compliance

Implementations may claim compliance with the cryptographic schemes specified in this document provided the external interface (input and output) to the schemes is equivalent to the interface specified here. Internal computations may be performed as specified here, or may be performed via an equivalent sequence of operations.

Note that this compliance definition implies that conformant implementations must perform all the cryptographic checks included in the scheme specifications in this document. This is important because the checks are essential for the prevention of subtle attacks.

1.4 Document Evolution

This document will be reviewed every five years to ensure it remains up to date with cryptographic advances. The document is now undergoing its first scheduled review.

This current draft, version 0.5, is a work in progress and is subject to change without notice. Its purpose is for review at the annual SECG meeting in June, 2011.

Additional intermittent reviews may also be performed occasionally, as deemed necessary by the Standards for Efficiency Cryptography Group.

1.5 Intellectual Property

The reader's attention is called to the possibility that compliance with this document may require use of an invention covered by patent rights. By the publication of this document, no position is taken with respect to the validity of this claim or of any patent rights in connection therewith. The patent holder(s) may have filed with the SECG a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. Additional details may be obtained from the patent holder and from the SECG website, `http://www.secg.org`.

1.6 Organization

This document is organized as follows.

The main body of the document focuses on the specification of public-key cryptographic signature schemes based on ECC and providing partial message recovery. Section 2 describes the mathematical foundations fundamental to the operation of all the schemes. Section 3 provides the cryptographic components used to build the schemes. Section 4 specifies signature schemes providing partial message recovery.

The appendices to the document provide additional relevant material. Appendix A gives a glossary of the acronyms and notation used, as well as an explanation of the terms used. Appendix B elaborates some of the details of the main body — discussing implementation guidelines, making security remarks, and attributing references. Appendix C provides reference ASN.1 syntax for implementations to use in identifying the schemes, and Appendix D lists the references cited in the document.

2 Mathematical Foundations

This section gives an overview of the mathematical foundations necessary for describing the signature schemes in this document.

2.1 Finite Fields

Abstractly, a finite field consists of a finite set of objects called field elements together with the description of two operations — addition and multiplication — that can be performed on pairs of field elements. These operations must possess certain properties.

There is a finite field containing q field elements if and only if q is a power of a prime number, and furthermore, for each such q there is precisely one finite field. The finite field containing q elements is denoted by $\mathbb{F}_q$.

In this document two types of finite fields $\mathbb{F}_q$ are used:

- Finite fields $\mathbb{F}_p$ where $q = p$ for some prime $p > 2$, which are called prime finite fields,
- Finite fields $\mathbb{F}_{2^m}$ where $q = 2^m$ for some integer $m \geq 1$, which are called characteristic-2 finite fields,

The rest of this section refers to [SEC 1, Section 2.1] for more details on prime finite fields and characteristic-2 finite fields.

2.2 Elliptic Curves

An elliptic curve over $\mathbb{F}_q$ is defined in terms of the solutions to an equation in $\mathbb{F}_q$. The form of the equation defining an elliptic curve over $\mathbb{F}_q$ differs depending on whether the field is a prime finite field, or a characteristic-2 finite field.

The rest of this section refers to [SEC 1, Section 2.2] and Section 3.2 which describe elliptic curves defined over prime finite fields and characteristic 2 finite fields.

2.3 Data Types and Conversions

The schemes specified in this document involve operations using several different data types and it is necessary to convert some data types into other data types. The following is a list of the data types and conversions that are used in ECPVS and ECAOS:

- Octet-String-to-Bit-String Conversion (OS2BS): The input to the OS2BS is an octet string M of length $mlen$ octets, and the output of OS2BS is a bit string B of length $blen = 8(mlen)$.
- Bit-String-to-Octet-String Conversion (BS2OS): The input to the BS2OS is a bit string B of length $blen$ bits, and the output of BS2OS is an octet string M of length $mlen = \lceil blen/8 \rceil$ octets.

- Octet-String-to-Integer Conversion (OS2I): The input to the OS2I is an octet string M of length *mlen* octets, and the output of OS2I is an integer x.
- Integer-to-Octet-String Conversion (I2OS): The input to the I2OS is a non-negative integer x together with the desired length *mlen* of the octet string such that $2^{8mlen} > x$, and the output of I2OS is an octet string M of length *mlen* octets.
- Field-Element-to-Octet-String Conversion (FE2OS): The input to the FE2OS is an element a of the field $\mathbb{F}_q$, where $q = p$ for some prime p, or $q = 2^m$ for some integer $m > 1$, and the output of FE2OS is an octet string M of length $mlen = \lceil \log_2 q/8 \rceil$ octets.
- Field-Element-to-Integer Conversion (FE2I): The input to the FE2I is an element a of the field $\mathbb{F}_q$, where $q = p$ for some prime p, or $q = 2^m$ for some integer $m > 1$, and the output of FE2I is an integer x.
- Elliptic-Curve-Point-to-Octet-String Conversion (EC2OS): The input to the EC2OS is a point P on an elliptic curve defined over the finite field $\mathbb{F}_q$ (where $q = p$ for some prime p, or $q = 2^m$ for some integer $m > 1$) and a parameter to specify whether the point compression is being used. The output of the EC2OS is an octet string M of length *mlen* octets, where $mlen = 1$ if $P = \mathcal{O}$, $mlen = \lceil (\log_2 q)/8 \rceil + 1$ if $P \neq \mathcal{O}$ and point compression is used, and $mlen = 2\lceil (\log_2 q)/8 \rceil + 1$ if $P \neq \mathcal{O}$ and point compression is not used.

The rest of this section refers to [SEC 1, Section 2.3], which describes how the above conversions should be performed.

3 Cryptographic Components

This section describes the various cryptographic components that are used to build signature schemes providing partial message recovery.

3.1 Security Levels

This standard follows NIST and ANSI in fixing five security levels (in bits): 80, 112, 128, 192, and 256.

According to [800-57], the use of 80-bit security level in the years 2011 through 2013 is *deprecated*, which means that the use of an algorithm or key length that provides the indicated security strength may be used if risk is accepted; and the use of 80-bit security level in 2014 and beyond is *disallowed*, which means that an algorithm or key length that provides the indicated security level shall not be used for applying cryptographic protection.

3.2 Elliptic Curve Domain Parameters and Validation

Elliptic curve domain parameters in this document are either defined over a prime finite field $\mathbb{F}_p$ of size prime p, or over a characteristic 2 finite field $\mathbb{F}_{2^m}$ of size 2^m. In the first case, elliptic curve domain parameters are given by a tuple

$$T = (p, a, b, G, n, h),$$

where p specifies the finite field $\mathbb{F}_p$, two elements $a, b \in \mathbb{F}_p$ specify an elliptic curve $E(\mathbb{F}_p)$ defined by the equation:

$$E: \ y^2 \equiv x^3 + ax + b \pmod{p},$$

a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_p)$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_p)/n$.

In the second case, when elliptic curve domain parameters are defined over $\mathbb{F}_{2^m}$, they are given by a tuple:

$$T = (m, f(x), a, b, G, n, h)$$

consisting of an integer m specifying the finite field $\mathbb{F}_{2^m}$, an irreducible binary polynomial $f(x)$ of degree m specifying the representation of $\mathbb{F}_{2^m}$, two elements $a, b \in \mathbb{F}_{2^m}$ specifying the elliptic curve $E(\mathbb{F}_{2^m})$ defined by the equation:

$$E: \ y^2 + xy = x^3 + ax^2 + b \text{ in } \mathbb{F}_{2^m},$$

a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_{2^m})$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_{2^m})/n$.

The rest of this section refers to [SEC 1, Section 3.1], where it is described how to choose elliptic curve domain parameters T, and how to check the validity of the chosen elliptic curve domain parameters.

More details on finite fields and elliptic curves can be found in [SEC 1, Sections 2.1, 2.2].

3.3 Elliptic Curve Key Pairs and Validation

Given some elliptic curve domain parameters $T = (p, a, b, G, n, h)$, or $T = (m, f(x), a, b, G, n, h)$ an elliptic curve key pair (d, Q) associated with T consists of an elliptic curve secret key d, which is an integer in the interval $[1, n-1]$, and an elliptic curve public key $Q = (x_Q, y_Q)$ which is the point $Q = dG$.

The rest of this section refers to [SEC 1, Section 3.2] where it is described, given elliptic curve domain parameters T, how to establish an elliptic curve key pair (d, Q) associated with T, and how to check the validity of the elliptic curve public key Q.

3.4 Hash Functions

Hash functions are used in signing and verify operations in ECPVS and ECAOS. In particular, in ECPVS, hash functions are used in the key derivation function *KDF*, in Step 6 in Section 4.1.3 and in Step 2 in Section 4.1.4. In ECAOS, hash functions are used in the mask generating function MGF. Hash functions are also used in verifiably random elliptic curve domain parameter generation and validation, and in random number generators.

The security level associated with a hash function depends on its application. Where collision resistance is necessary, the security level is at most half the output length (in bits) of the hash function. Where collision resistance is not necessary, the security level is at most the length (in bits) of the hash function.

It is recommended that hash functions used in ECPVS and in ECAOS are chosen to be collision resistant. That is, the output length (in bits) of the hash function should be at least twice the security level.

The list of supported hash functions in this standard at this time is:

SHA-1

SHA-224

SHA-256

SHA-384

SHA-512

Let *hashlen* denote the length in octets of hash values computed using a hash function. Then the values of *hashlen*, for the hash functions SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512 are, respectively, 20, 28, 32, 48, and 64. The hash functions SHA-224, SHA-256, SHA-384 and SHA-512 are believed to achieve 112, 128, 192 and 256-bit security levels with respect to collision resistance, respectively. We note that, in 2005, an attack [WYY05b] was announced that finds a collision in SHA-1 in about 2^{69} hash operations. Subsequently, attacks using 2^{63} hash operations were announced [WYY05a]. These attacks decrease the security of SHA-1 against collision resistance. Therefore, the use of SHA-1 in ECPVS and ECAOS may not be suitable at the 80-bit security level.

The rest of this section refers to [SEC 1, Section 3.5], for a list of the supported hash functions and for more details on hash functions.

3.5 Mask Generation Function

ECAOS uses mask generation function MGF that takes as input octet string x and octet length l and outputs an octet string $\mathrm{MGF}(x, l)$ of length l. We denote by L^{in}_{MGF} the maximal octet length of input x, and by L^{out}_{MGF} the maximal octet length of output $\mathrm{MGF}(x, l)$.

The following implementation of mask generation function MGF is recommended. It uses the mask generation function MGF1 specified in ISO9796-3[9796-3], and with $L^{in}_{\mathrm{MGF}} = L^{in}_{\mathrm{Hash}} - 8$ and $L^{out}_{\mathrm{MGF}} = 2^{64} L^{out}_{\mathrm{Hash}}$. Here, L^{in}_{Hash} and L^{out}_{Hash} are maximal input and output octet lengths, respectively, of the hash function used in MGF1.

$$\mathrm{MGF}(x, l) = \mathrm{MGF1}(x||\mathrm{I2OS}(0, 4), 2^{32} L^{out}_{\mathrm{Hash}})||\mathrm{MGF1}(x||\mathrm{I2OS}(1, 4), 2^{32} L^{out}_{\mathrm{Hash}})||$$

$$\ldots||\mathrm{MGF1}(x||\mathrm{I2OS}(b-1, 4), 2^{32} L^{out}_{\mathrm{Hash}}) \in \{0,1\}^{8l},$$

where $b = \lceil l/(2^{32} L^{out}_{\mathrm{Hash}}) \rceil$. If $l \leq 2^{32} L^{out}_{\mathrm{Hash}}$, it can be written as follows:

$$\mathrm{MGF}(x, l) = \mathrm{MGF1}(x||\mathrm{I2OS}(0, 4), l) \in \{0,1\}^{8l}.$$

3.6 Key Derivation Functions

A key derivation function, denoted by *KDF*, is used in signing and verify operations in ECPVS; see Section 4.1.3 and 4.1.4.

In general, *KDF* is used to derive keying data K, and may be expressed as

$$K = KDF(keydatalen, Z, OtherInput),$$

where *keydatalen* is an integer that specifies the octet length of K to be generated, Z is a *shared secret* value, and *OtherInput* is some (optional) additional input. *OtherInput* might consist of some private information mutually-known to the parties involved in key derivation process. *OtherInput* might also consist of some other data such as the identifiers of the parties involved in key derivation.

In ECPVS, the signer and *any* verifier should be able to compute the keying data derived from the *KDF*. More precisely, the input string Z to the *KDF* in ECPVS is the octet string representation of the x-coordinate of an elliptic curve point R. The elliptic curve point R is initially computed by the signer during the signing operation; see Step 2 in Section 4.1.3. During the verify operation, any user, who has access to the public key of the user and the signature on the message, can recover the elliptic curve point R, and so can compute the octet string Z. Hence, the input string Z to the *KDF* is not necessarily secret and any user can compute the keying data K; see Steps 4–6 in Section 4.1.4.

Remark 1 *In ECPVS, the signer and a certain set of users may agree on* ***SharedInfo****, which is an octet string consisting of some data shared only by these users, as a part of the input data*

OtherInput *to the key derivation function. In this case, the third parties who do not know* ***SharedInfo*** *would not be able to verify the ECPVS signatures.*

Identifiers for the signer and verifier may also be used as a part of the input data ***OtherInput*** *to the key derivation function. In the case that the verifier is known to the signer in advance, a specific identifier can be used for the verifier's identifier. Otherwise, the identifier for the verifier might be the encoding of the string "****Verifier****".*

The list of supported key derivation functions in this standard at this time is:

ANSI-X9.63-KDF

IKEv2-KDF

TLS-KDF

NIST-800-56-Concatenation-KDF

The rest of this section refers to [SEC 1, Section 3.6] for a list of supported key derivation functions and more details.

3.7 Symmetric Encryption Schemes

Symmetric encryption schemes are described in terms of an encryption operation, denoted by ENC_K, and a decryption operation, denoted by DEC_K, where K is the keying data derived from the associated key derivation function; see Section 3.6. The symmetric encryption schemes are used in signing and verify operations in ECPVS; see Step 5 in Section 4.1.3 and Step 6 in Section 4.1.4.

In general, symmetric encryption schemes are used to provide data confidentiality. However, in ECPVS, the use of the symmetric encryption scheme should not be assumed to provide data confidentiality for the portion of the message that is encrypted since the keying data K can be derived by the signer and by *any* user unless a particular set of users are designated to be verifiers, in which case some confidentiality is obtained; see Remark 1 in Section 3.6.

The list of supported symmetric encryption schemes in this standard at this time is:

3-key TDES in CBC mode

XOR encryption scheme

AES–128 in CBC mode

AES–192 in CBC mode

AES–256 in CBC mode

AES–128 in CTR mode

AES–192 in CTR mode

AES–256 in CTR mode

The symmetric encryption scheme in ECPVS should be chosen at the desired security level. In particular, at the κ-bit security level, if the AES block cipher is chosen, then the octet length *keydatalen* of the keying data K should satisfy $8 \cdot keydatalen \geq \kappa$; if the XOR encryption scheme is chosen, then the keying data K should be chosen such that the bit length of K is equal to the bit length of the data input to the encryption function.

The encryption scheme 3-key TDES as supported in this document uses keying data K of length $keydatalen = 24$ octets, and achieves 112-bit security level. The encryption schemes AES–128, AES–192, and AES–256 are believed to achieve 128, 192 and 256-bit security levels, respectively. The XOR encryption scheme with keying data K achieves κ-bit security level, where κ is the bit length of K.

The rest of this section refers to [SEC 1, Section 3.8], for a list of the supported symmetric encryption schemes and for more details on symmetric encryption schemes.

3.8 Random Number Generation

Random numbers are used for key generation in ECPVS and ECAOS. All random numbers in this standard must be generated using a random number generator that must comply with ANSI X9.82 [X9.82] or corresponding NIST Special Publication 800-90 [800-90].

The rest of this section refers to [SEC 1, Section 3.10] for a description of one random number generator and for more details on random number generators.

3.9 Message Redundancy Criteria and Message Padding

ECPVS computes a signature on a message (M_{rec}, M_{vis}), which is a pair of octet strings, where M_{rec} is the recoverable message part and M_{vis} is the visible message part. The recoverable message part M_{rec} contains an *inherent* redundancy The inherent redunduncy shall be agreed upon in advance by the users. We denote the bit length of the inherent redundancy in M_{rec} by *inhredBitlen*. For example, the inherent redundancy may come from the ASCII encoding of a proper English text. In such a case, since the first bit of each octet in the ASCII encoding of a text is zero, *inhredBitlen* would be the number of octets in the ASCII encoding of the text.

If the users cannot agree upon the inherent redundancy in advance, then it may be assumed that M_{rec} has zero bits of inherent redundancy, i.e. $inhredBitlen = 0$.

During the signing operation in ECPVS, M_{rec} is padded with an octet string P_2 of length *padOctlen* octets. The octet length *padOctlen* must satisfy $1 \leq padOctlen \leq 255$.

The users should agree upon *padOctlen* in advance to their communication. Otherwise, *padOctlen* is chosen by the signer to achieve the desired security level, and *padOctlen* can be derived by the verifier during the verify operation.

The inherent redundancy in M_{rec} and the redundancy in the padding P_2 determines the *total* redundancy of the (padded) message pair (M_{rec}, M_{vis}) to be signed. The amount of total redundancy determines the security level of ECPVS against existentially forgery under adaptive chosen message attacks; see Section B.1.1. In particular, for κ-bit security level, *padOctlen* should be chosen to be

a positive integer such that $8 \cdot padOctlen + inhredBitlen \geq \kappa$.

In ECPVS, the message redundancy criteria should also specify one of the followings in order to achieve the security objectives (see Section B.1.1.1):

- The recoverable message component M_{rec} has a fixed length.
- The recoverable message component M_{rec} *begins* with a fixed-length representation of its length.
- The visible message component M_{vis} has a fixed length.
- The visible message component M_{vis} *ends* with a fixed-length representation of its length.
- DER encoding of ASN.1 types is used for the recoverable message part M_{rec}.

If the users could not agree upon the message redundancy in advance, then it may be assumed by default that the recoverable message part M_{rec} begins with an 8-octet representation of its length.

4 Signature Schemes

This section specifies two signature schemes providing partial message recovery based on ECC.

Signature schemes are designed to be used by two entities — a signer U and a verifier V — when U wants to send a message M in an authentic manner and V wants to verify the authenticity of M. In fact, once a message is signed, any entity V having a copy of U's public key may verify the signature. In particular, the verifier may not be the entity to whom U originally sent the message.

Here, signature schemes are described in terms of a signing operation, a verify operation, and associated setup and key deployment (key generation and authentic distribution) procedures. Entities U and V should use the schemes as follows when they want to communicate. First U and V should use the setup procedure to establish which scheme options will be used, then U should use the key deployment procedure to select a key pair and V should obtain U's authentic public key — U will use the key pair to control the signing operation, and V will use the public key to control the verify operation. Then, each time U wants to send a message M, entity U should apply the signing operation to M under its key pair to obtain a signature S on M, form a signed message from M and S, and convey the signed message to V. Finally, when V receives the signed message, entity V should apply the verify operation to the signed message under U's public key to verify its authenticity. If the verify operation outputs "valid", entity V concludes the signed message is indeed authentic.

Loosely speaking, signature schemes are designed so that it is hard for an adversary who does not know U's secret key to forge valid signed messages.

There are three types of signature schemes, depending on the form of the signed message that U must convey to V: 1) Signature schemes with appendix, in which U must convey both M and S to V; 2) Signature schemes with message recovery, in which M can be recovered from S, so U need to convey only S to V; and 3) Signature schemes with partial message recovery, in which part of M can be recovered from S, so U need to convey only S and the visible (non-recoverable) part of M to V. Signatures with partial message recovery may be viewed as intermediate between signatures with appendix and signatures with (full) message recovery.

In this specification, the signature schemes with partial message recovery supported are the Elliptic Curve Pintsov-Vanstone signature scheme (ECPVS) and the Elliptic Curve Abe-Okamoto-Suzuki signature scheme (ECAOS).

See Appendix B for a commentary on the contents of this section, including implementation discussion, security discussion, and references.

4.1 Elliptic Curve Pintsov-Vanstone Signatures

The Elliptic Curve Pintsov-Vanstone Signature scheme (ECPVS) is a signature scheme with partial message recovery based on ECC. It is designed to be existentially unforgeable, even in the presence of an adversary capable of launching chosen-message attacks.

The setup procedure for ECPVS is specified in Section 4.1.1, the key deployment procedure is specified in Section 4.1.2, the signing operation is specified in Section 4.1.3, and the verify operation is specified in Section 4.1.4.

4.1.1 Scheme Setup

Entities U and V must perform the following setup procedure to prepare to use ECPVS:

1. Entity U should establish which of the hash functions supported in Section 3.4 to use when generating signatures. Let *Hash* denote the hash function chosen, and *hashlen* denote the length in octets of the hash values produced using *Hash*.

2. Entity U should establish elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $T = (m, f(x), a, b, G, n, h)$ at the desired security level. The elliptic curve domain parameters T should be generated using the primitive specified in Section 3.2. Entity U should receive assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.2.

3. Entity U should establish which of the symmetric encryption primitive supported in Section 3.7 to use when generating signatures.

4. Entity U should establish which message redundancy criteria should be used. See Sections 3.9, B.1.1 and B.1.1.1.

5. Entity V should obtain in an authentic manner the hash function *Hash*, the message redundancy criteria, the symmetric encryption primitive, and elliptic curve domain parameters T established by U.

Entity V must receive assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.2.

4.1.2 Key Deployment

Entities U and V must perform the following key deployment procedure to prepare to use ECPVS:

1. Entity U should establish an elliptic curve key pair (d_U, Q_U) associated with T to use with the signature scheme. The key pair should be generated using the primitive specified in Section 3.3.

2. Entity V should obtain in an authentic manner the elliptic curve public key Q_U selected by U.

Entity V must receive assurance that the elliptic curve public key Q_U is valid using one of the methods specified in Section 3.3.

4.1.3 Signing Operation

Entity U must sign messages using ECPVS using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: A message (M_{rec}, M_{vis}), which is a pair of octet strings to be signed, where M_{rec} is intended to be a recoverable message part and M_{vis} is intended to be a visible message.

Output: A message representative octet string r and an integer s, or "invalid".

Actions: Generate the message representative octet string r and an integer s as follows:

1. Verify that the recoverable message part M_{rec} meets the established redundancy criteria; see Section 3.9. If it does not, output "invalid".

2. Select an ephemeral elliptic curve key pair, associated with the elliptic curve domain parameters T established during the setup procedure, (k, R) with $R = (x_R, y_R)$ using the key pair generation primitive specified in Section 3.3.

3. Convert the field element x_R to an octet string Z using the conversion routine FE2OS specified in Section 2.3.

4. Compute a padding octet string P_2 as follows:

   4.1. Using the established redundancy criteria for M_{rec}, determine the number *padOctlen* of octets with which to pad M_{rec}; see Sections 3.9 and B.1.2.

   4.2. Convert *padOctlen* to an octet string P_1 of length one octet.

   4.3. Form an octet string P_2 of length *padOctlen* octets, with each octet in the string equaling P_1. For example, if $padOctlen = 3$, then $P_1 = 03$ and $P_2 = \texttt{030303}$.

5. Compute the message representative r as follows:

   5.1. Determine from the symmetric encryption algorithm ENC_K, and possibly also from the length of the plaintext octet string $P_2 \| M_{rec}$, the length *keydatalen* of the symmetric encryption key needed.

   5.2. Compute $K = KDF(keydatalen, Z)$.

   5.3. Compute $r = ENC_K(P_2 \| M_{rec})$.

6. Use the hash function selected during the setup procedure to compute the hash value:

   $$H = Hash(r \| M_{vis})$$

   of length *hashlen* octets.

7. Derive an integer e from H as follows:

   7.1. Convert the octet string H to a bit string $\overline{H}$ using the conversion routine OS2BS specified in Section 2.3.

   7.2. Set $\overline{E} = \overline{H}$ if $\lceil \log_2 n \rceil \geq 8(hashlen)$, and set $\overline{E}$ equal to the leftmost $\lceil \log_2 n \rceil$ bits of $\overline{H}$ if $\lceil \log_2 n \rceil < 8(hashlen)$.

   7.3. Convert the bit string $\overline{E}$ to an octet string E using the conversion BS2OS routine specified in Section 2.3.

7.4. Convert the octet string E to an integer e using the conversion routine OS2I specified in Section 2.3.

8. Compute:

$$s = k - ed_U \bmod n.$$

4.1.4 Verify Operation

Entity V must verify signed messages from entity U using ECPVS using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The verify operation takes as input:

1. An octet string M_{vis}, which is the visible (non-recoverable) message part.
2. An integer s, which is part of the ECPVS signed message.
3. An octet string r, which is the purported message representative.

Output: The verify operation yields as output:

1. An indication of whether the purported ECPVS signed message is valid or not — either "valid" or "invalid".
2. The recoverable message part M_{rec}, if the purported ECPVS signature is "valid".

Actions: Verify the purported ECPVS signed message as follows:

1. If s is not an integer in the interval $[1, n-1]$, output "invalid" and stop.
2. Use the hash function established during the setup procedure to compute the hash value:

$$H = Hash(r \| M_{vis})$$

of length *hashlen* octets as specified in Section 3.4. If the hash function outputs "invalid", output "invalid" and stop.

3. Derive an integer e from H as follows:

3.1. Convert the octet string H to a bit string $\overline{H}$ using the conversion OS2BS routine specified in Section 2.3.

3.2. Set $\overline{E} = \overline{H}$ if $\lceil \log_2 n \rceil \geq 8(hashlen)$, and set $\overline{E}$ equal to the leftmost $\lceil \log_2 n \rceil$ bits of $\overline{H}$ if $\lceil \log_2 n \rceil < 8(hashlen)$.

3.3. Convert the bit string $\overline{E}$ to an octet string E using the conversion routine BS2OS specified in Section 2.3.

3.4. Convert the octet string E to an integer e using the conversion OS2I routine specified in Section 2.3.

4. Compute the elliptic curve point:

$$R = (x_R, y_R) = sG + eQ_U.$$

If $R = \mathcal{O}$, output "invalid" and stop.

5. Convert the field element x_R to an octet string Z using the conversion routine FE2OS specified in Section 2.3.

6. Compute padded message representative N as follows:

6.1. Determine from the symmetric decryption algorithm DEC_K, and possibly also from the length of the message representative r, the length *keydatalen* of the symmetric encryption key needed.

6.2. Compute $K = KDF(keydatalen, Z)$.

6.3. Compute $N = DEC_K(r)$.

7. Compute the recoverable message part M_{rec} as follows:

7.1. Let P_1 be the first octet of N.

7.2. Convert P_1 to an integer *padOctlen*.

7.3. For an octet string P_2 of length *padOctlen* octets, with each octet in the string equaling P_1. Note for example, if $padOctlen = 3$, then $P_1 = 03$ and $P_2 = \texttt{030303}$.

7.4. If the first *padOctlen* octets of N do not match P_2, then stop and output "invalid".

7.5. If the first *padOctlen* octets of N match P_2, then output the remaining octets of N after the first *padOctlen* as the recoverable message part M_{rec}. Note that $N = P_2 \| M_{rec}$ will hold.

8. Using the established redundancy criteria, determine if the number *padOctlen* of octets with which M_{rec} was padded is acceptable. If it is not acceptable, stop and output "invalid".

9. Determine if M_{rec} meets the established redundancy criteria. If M_{rec} meets the redundancy criteria, then output "valid" and the recovered message M_{rec}. Otherwise, stop and output "invalid".

4.2 Elliptic Curve Abe-Okamoto-Suzuki Signatures

The Elliptic Curve Abe-Okamoto-Suzuki Message Recovery Signature Scheme (ECAOS) [AOS08] is a signature scheme with partial message recovery based on ECC. It is designed to be existentially unforgeable, even in the presence of an adversary capable of launching chosen-message attacks.

The setup procedure for ECAOS is specified in Section 4.2.1, the key deployment procedure is specified in Section 4.2.2, the signing operation is specified in Section 4.2.3, and the verify operation is specified in Section 4.2.4.

4.2.1 Scheme Setup

Entities U and V must perform the following setup procedure to prepare to use ECAOS:

1. Entity U should establish the mask generation function MGF with maximum input octet length L^{in}_{MGF} and maximum output octet length L^{out}_{MGF}, as specified in Section 3.5, to use when generating signatures.

2. Entity U should establish the octet length domain parameters as below:

   - L_n: octet length of prime integer n,
   - L_F: octet length of an element of finite field F,
   - K: octet length of the extended part of a mask string,
   - L_{red}: octet length of added redundancy, and
   - L^{min}_{rec}: octet length of minimum recoverable message.

   These length domain parameters are required to satisfy the following conditions w.r.t. the security parameter $\kappa > 0$ to ensure κ-bit security: $L_n \geq \lceil 2\kappa/8 \rceil$, $L_F \geq \lceil 2\kappa/8 \rceil$, $K \geq \lceil \kappa/8 \rceil$, $L_{red} \geq \lceil \kappa/8 \rceil$, and $L_{red} + L^{min}_{rec} \geq \lceil 2\kappa/8 \rceil$.

   L_{red} and L^{min}_{rec} are required to satisfy the following conditions because of the maximum output octet length L^{out}_{MGF} of mask generation function MGF: $L_{red} \leq L^{out}_{\mathrm{MGF}}$ and $L^{min}_{rec} \leq L^{out}_{\mathrm{MGF}}$.

3. Entity U should establish elliptic curve domain parameters $T = (p, a, b, G, n, h)$ or $T = (m, f(x), a, b, G, n, h)$ at the desired security level. The elliptic curve domain parameters T should be generated using the primitive specified in Section 3.2. Entity U should receive assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.2.

4. Entity V should obtain in an authentic manner the mask generation function MGF, the octet length domain parameters, and elliptic curve domain parameters T established by U.

Entity V must receive assurance that the elliptic curve domain parameters T are valid using one of the methods specified in Section 3.2.

Throughout this section $L(s)$ will denote the octet length of octet string s.

4.2.2 Key Deployment

Entities U and V must perform the following key deployment procedure to prepare to use ECAOS:

1. Entity U should establish an elliptic curve key pair (x, Y) associated with T to use with the signature scheme. The key pair should be generated using the primitive specified in Section 3.3.

2. Entity V should obtain in an authentic manner the elliptic curve public key Y selected by U.

Entity V must receive assurance that the elliptic curve public key Y is valid using one of the methods specified in Section 3.3.

4.2.3 Signing Operation

Entity U must sign messages with ECAOS using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: A message (M_{rec}, M_{vis}), which is a pair of octet strings to be signed, where M_{rec} is intended to be a recoverable message part and M_{vis} is intended to be a visible message. M_{rec} is of octet length $L(M_{rec})$, where $0 \leq L(M_{rec}) \leq min(L^{in}_{\mathrm{MGF}} - (L_F + 11), L^{out}_{\mathrm{MGF}} - 1)$; see [1], and M_{vis} is of octet length $L(M_{vis})$, where $0 \leq L(M_{vis}) \leq L^{in}_{\mathrm{MGF}} - (L_{red} + \widetilde{L}_{rec} + 1)$; see [2].

Output: A message representative octet string $r \in \{0,1\}^{8(L_{red}+\widetilde{L}_{rec})}$ and an integer $s \in [1, n-1]$.

Actions: Generate the message representative octet string r and an integer s as follows:

1. Compute $L_{rec} = L(M_{rec})$ and $L_{vis} = L(M_{vis})$.
2. Compute $\widetilde{L}_{rec} = \max(L^{min}_{rec}, L_{rec} + 1)$ and $\widetilde{M}_{rec} = \mathrm{I2OS}(1, \widetilde{L}_{rec} - L_{rec}) || M_{rec} \in \{0,1\}^{8\widetilde{L}_{rec}}$; see [3].
3. Compute $L'_{rec} = \mathrm{I2OS}(L_{rec}, 8) \in \{0,1\}^{64}$,
4. Select an ephemeral elliptic curve key pair (k, R) with $R = (x_R, y_R)$ associated with the elliptic curve domain parameters T established during the setup procedure using the key pair generation primitive specified in Section 3.3.
5. Compute $R' = \mathrm{EC2OS}_E(R, \mathit{compressed}) \in \{0,1\}^{8(L_F+1)}$.
6. Compute $h_0 = \mathrm{MGF}(\widetilde{M}_{rec} || L'_{rec} || R' || \mathrm{I2OS}(0,1), L_{red}) \in \{0,1\}^{8L_{red}}$.
7. Compute $h_1 = \mathrm{MGF}(h_0 || R' || \mathrm{I2OS}(1,1), \widetilde{L}_{rec}) \in \{0,1\}^{8\widetilde{L}_{rec}}$.
8. Compute $r = h_0 || (\widetilde{M}_{rec} \oplus h_1) \in \{0,1\}^{8(L_{red}+\widetilde{L}_{rec})}$.
9. Compute $u = \mathrm{MGF}(M_{vis} || r || \mathrm{I2OS}(2,1), L_n + K) \in \{0,1\}^{8(L_n+K)}$.
10. Compute $t = \mathrm{OS2I}(u) \mod n \in [0, n-1]$.
11. If $t = 0$, go to step 4.
12. Compute integer $s = k - xt \mod n$.
13. If $s = 0$, go to step 4.
14. Output signature $(r, s) \in \{0,1\}^{8(L_{red}+\widetilde{L}_{rec})} \times [1, n-1]$.

[1] See step 6 and step 8 Notice that $L(\widetilde{M}_{rec}) = L(M_{rec}) + 1$ for long M_{rec}, and the length of input of MGF in step 6 is $(L(M_{rec}) + 1) + 8 + (L_F + 1) + 1$.

[2] See step 9

[3] Add padding to M_{rec} to satisfy $L_{red} + L(\widetilde{M}_{rec}) \geq 2\kappa$.

4.2.4 Verify Operation

Entity V must verify signed messages from entity U using ECAOS using the keys and parameters established during the setup procedure and the key deployment procedure as follows:

Input: The verify operation takes as input:

1. An octet string M_{vis}, which is the visible (non-recoverable) message part.
2. An integer s, which is part of the ECAOS signed message.
3. An octet string r, which is the purported message representative.

Output: The verify operation yields as output:

1. An indication of whether the purported ECAOS signed message is valid or not — either "valid" or "invalid".
2. The recoverable message part M_{rec}, if the purported ECAOS signature is "valid".

Actions: Verify the purported ECAOS signed message as follows:

1. Compute $L_r = L(r)$, $L_{vis} = L(M_{vis})$, and $\widetilde{L}_{rec} = L_r - L_{red}$.
2. Check $L_{red} + L_{rec}^{min} \leq L_r \leq L_{red} + min(L_{\mathrm{MGF}}^{in} - (L_F + 10), L_{\mathrm{MGF}}^{out})$; see [4] , $1 \leq s \leq n-1$, and $0 \leq L_{vis} \leq L_{\mathrm{MGF}}^{in} - (L_{red} + \widetilde{L}_{rec} + 1)$; see [5] .
3. If one of the checks in step 2 does not hold, output 'invalid' and stop.
4. Compute $u = \mathrm{MGF}(M_{vis}||r||\mathrm{I2OS}(2,1), L_n + K) \in \{0,1\}^{8(L_n+K)}$.
5. Compute $t = \mathrm{OS2I}(u) \mod n \in [0, n-1]$.
6. If $t = 0$, output 'invalid' and stop.
7. Compute $R = sG + tY \in \langle G \rangle$.
8. If R is the point at infinity, output 'invalid' and stop.
9. Compute $R' = \mathrm{EC2OS}_E(R, compressed) \in \{0,1\}^{8(L_F+1)}$.
10. Decompose $r = r_0||r_1 \in \{0,1\}^{8(L_{red}+\widetilde{L}_{rec})}$ s.t. $r_0 \in \{0,1\}^{8L_{red}}$ and $r_1 \in \{0,1\}^{8\widetilde{L}_{rec}}$.
11. Compute $h_1 = \mathrm{MGF}(r_0||R'||\mathrm{I2OS}(1,1), \widetilde{L}_{rec}) \in \{0,1\}^{8\widetilde{L}_{rec}}$.
12. Compute $\widetilde{M}_{rec} = r_1 \oplus h_1 \in \{0,1\}^{8\widetilde{L}_{rec}}$.

[4] See step 17 and step 12. Notice that $L_r = L_{red} + L(\widetilde{M}_{rec})$, and the length of input of MGF in step 17 is $L(\widetilde{M}_{rec}) + 8 + (L_F + 1) + 1$.

[5] See step 4.

13. Decompose $\widetilde{M}_{rec} = \mathrm{I2OS}(1, i)||M_{rec}$ s.t. $1 \leq i \leq L_{rec}^{min}$ if there exists such unique i.
14. If there exists no such i, output 'invalid' and stop.
15. Compute $L_{rec} = \widetilde{L}_{rec} - i$.
16. Compute $L'_{rec} = \mathrm{I2OS}(L_{rec}, 8) \in \{0,1\}^{64}$.
17. Compute $h_0 = \mathrm{MGF}(\widetilde{M}_{rec}||L'_{rec}||R'||\mathrm{I2OS}(0,1), L_{red}) \in \{0,1\}^{8L_{red}}$.
18. If $r_0 \neq h_0$, output 'invalid' and stop.
19. Output "valid" and the recovered message $M_{rec} \in \{0,1\}^{8L_{rec}}$.

A Glossary

This section supplies a glossary to the terms and notation used in this document.

The section is organized as follows: Section A.1 lists the terms, Section A.2 the acronyms, and Section A.3 notation used in this document.

A.1 Terms

Terms used in this document include:

Message representative	A signature component from which all or part of a signed message can be recovered and subsequently verified in a signature scheme providing partial message recovery.
recoverable message part	Part of a signed message which must be recovered from the message representative and which also may contain some intrinsic redundancy, thereby reducing the need for padding and bandwidth expansion.
visible message part	Part of a signed message which cannot be recovered from the message representative, but rather must be made directly available to the verifier. Intrinsic redundancy in the visible message part does not reduce the need for padding and bandwidth expansion.

A.2 Acronyms, Initialisms and Other Abbreviations

The acronyms, initialisms and other abbreviations used in this document include:

AES	Advanced Encryption Standard.
ANSI	American National Standards Institute.
ASN.1	Abstract Syntax Notation One.
CBC	Cipher Block Chaining.
CTR	Counter (block cipher mode of operation)
DEC	Decryption Function.
DER	Distinguished Encoding Rules.
EC	Elliptic Curve.
ECC	Elliptic Curve Cryptography.
ECAOS	Elliptic Curve Abe-Okamoto-Suzuki Signature Scheme.

ECPVS	Elliptic Curve Pintsov-Vanstone Signature Scheme.
ENC	Encryption function.
IEEE	Institute of Electrical and Electronics Engineers.
KDF	Key Derivation Function.
MGF	Mask Generation Function.
NIST	(U.S.) National Institute of Standards and Technology.
SEC	Standard for Efficient Cryptography
SHA	Secure Hash Algorithm
TDES	Triple Data Encryption Standard.
XOR	Exclusive Or

A.3 Notation

The notation adopted in this document is:

$[X]$	Indicates that the inclusion of X is optional.
$[x, y]$	The interval of integers between and including x and y.
$\lceil x \rceil$	Ceiling: the smallest integer $\geq x$. For example, $\lceil 5 \rceil = 5$, $\lceil 5.3 \rceil = 6$, and $\lceil -5.3 \rceil = -5$.
$\lfloor x \rfloor$	Floor: the largest integer $\leq x$. For example, $\lfloor 5 \rfloor = 5$, $\lfloor 5.3 \rfloor = 5$, and $\lfloor -5.3 \rfloor = -6$.
$x \bmod n$	The unique remainder r, $0 \leq r \leq n-1$, when x is divided by n. For example, $23 \bmod 7 = 2$.
C	Ciphertext.
d	An EC private key.
E	An elliptic curve over the field $\mathbb{F}_q$ defined by a and b.
$E(\mathbb{F}_q)$	The set of all points on an elliptic curve E defined over $\mathbb{F}_q$ and including the point at infinity $\mathcal{O}$.
$\#E(\mathbb{F}_q)$	If E is defined over $\mathbb{F}_q$, then $\#E(\mathbb{F}_q)$ denotes the number of points on the curve (including the point at infinity $\mathcal{O}$). $\#E(\mathbb{F}_q)$ is called the order of the curve E.
$\mathbb{F}_{2^m}$	The finite field containing 2^m elements, where m is a positive integer.
$\mathbb{F}_p$	The finite field containing p elements, where p is a prime.
$\mathbb{F}_{p^m}$	The finite field containing p^m elements, where p is a prime and $m \geq 1$ is an integer.

$\mathbb{F}_q$	The finite field containing q elements. In this document attention is restricted to the cases where q is an odd prime number (p) or a power of 2 (2^m).
G	A distinguished point on an elliptic curve called the base point or generating point.
$\gcd(x, y)$	The greatest common divisor of integers x and y.
h	$h = \#E(\mathbb{F}_q)/n$, where n is the order of the base point G. h is called the co-factor.
k	An EC private key specific to one particular instance of a cryptographic scheme.
K	Keying data.
$\log_2 x$	The logarithm of x to the base 2.
m	The extension degree of the finite field $\mathbb{F}_{p^m}$ over $\mathbb{F}_p$, where p is prime.
M	A message.
M_{rec}	Recoverable message part.
M_{vis}	Visible message part.
mod	Modulo.
mod $f(x)$	Arithmetic modulo the polynomial $f(x)$.
mod n	Arithmetic modulo n.
n	The order of the base point G.
$\mathcal{O}$	A special point on an elliptic curve, called the point at infinity. This is the additive identity of the elliptic curve group.
p	An odd prime number.
P	An EC point.
q	The number of elements in the field $\mathbb{F}_q$.
Q	An EC public key.
R	An EC public key specific to one particular instance of a cryptographic scheme.
S	A digital signature.
T	Elliptic curve domain parameters.
U, V	Entities.
$\|X\|$	Length in octets of the octet string X.
$X \| Y$	Concatenation of two strings X and Y. X and Y are either both bit strings or both octet strings.

$X \oplus Y$	Bitwise exclusive-or of two bit strings X and Y of the same bit length.
x_P	The x-coordinate of a point P.
y_P	The y-coordinate of a point P.
$\tilde{y}_P$	The representation of the y-coordinate of a point P when point compression is used.
z, or Z	A shared secret value. z denotes a shared secret integer or field element, and Z a shared secret bit string or octet string.

B Commentary

This section provides commentary on the main body of this document, including implementation discussion, security discussion, and references.

The aim of this section is to supply implementers with relevant guidance. However the section does not attempt to provide exhaustive information but rather focuses on giving basic information and including pointers to references which contain additional material.

B.1 Commentary on ECPVS

B.1.1 Security Considerations

It was proved in [BJ01, Theorem 1] that in the combined random oracle model and ideal cipher model, ECPVS is asymptotically secure against existential forgeries under adaptive chosen message attacks, where the recoverable message part M_{rec} in (M_{rec}, M_{vis}) is assumed to have some fixed length with total redundancy of length *totred* bits (see Section B.1.1.1), if the elliptic curve discrete logarithm problem is intractable and $1/2^{totred}$ is negligible. In particular, in this model and under these assumptions, a forgery in ECPVS is possible only after one performs 2^{totred} (signature verification) operations. See also [BJ01, Theorem 2 and Theorem 3] for other security proofs of ECPVS. Theorem 2 in [BJ01] proves the security of ECPVS in the combined ideal cipher and generic group model, assuming that the underlying hash function is a *strong hash function*. Theorem 3 in [BJ01] proves the security of ECPVS in the combined random oracle and generic group model, assuming that the underlying encryption function satisfies *weak uniform decipherment* property.

It follows that, ECPVS signatures have two different security levels: primary and secondary. The primary security level measures the resistance to attacks on the private key, and the secondary security level measures the resistance to signature forgery. The primary security level, say κ_p-bits, is achieved by selecting cryptographic components (elliptic curve domain parameters, the hash function and the encryption function) at κ_p-bit security level; and the secondary security level, say κ_s-bits, is achieved by selecting the total amount of the redundancy to be (at least) κ_s-bits. The secondary security level by definition is at most the primary security level, and depending on the desired security level against signature forgery, κ_s might be chosen to be smaller than κ_p. The overall security level against signature forgery is determined by the minimum of κ_p and κ_s.

As an example, let (M_{rec}, M_{vis}) be the message pair to be signed under ECPVS and assume that the cryptographic components (elliptic curve domain parameters, the hash function and the encryption function) in ECPVS are chosen in accordance with the 128-bit security level, which is the primary security level. If M_{rec} has 10-bits of inherent redundancy, then choosing $padOctlen = 15$ yields a total redundancy of length $10 + 15 \cdot 8 = 130$-bits in the padded message. Hence, ECPVS achieves 128-bit security against signature forgery. In the same setting, if the message was padded so that $padOctlen = 5$ then the secondary security level in ECPVS would be $10 + 5 \cdot 8 = 50$-bits even though the primary security level is still 128-bits.

It is very plausible to assume that the recommended elliptic curve domain parameters, the hash functions and the encryption functions in this document satisfy the necessary requirements imposed by the security proofs in [BJ01]. Hence, the specifications given in this document for implementing

ECPVS should satisfy the claimed security levels.

B.1.1.1 Protecting the Integrity of the Boundary between r and M_{vis} In the security proofs in [BJ01], it is assumed that the recoverable message part M_{rec} in (M_{rec}, M_{vis}) is of some fixed length. If this assumption is violated then an adversary can get a verifier to accept a false signature as follows. Capturing a valid signature (r, M_{vis}, s) which was previously signed by a verifier on the message (M_{rec}, M_{vis}), the adversary can form a fake signature (r', M'_{vis}, s), where r' and M'_{vis} are such that $r' \neq r$ and $r' \,\|\, M'_{vis} = r \,\|\, M_{vis}$. However, this attack can be prevented if one of the following specifications is used in ECPVS:

- The recoverable message part M_{rec} has a fixed length,
- The recoverable message part M_{rec} *begins* with a fixed-length representation of its length,
- The visible message part M_{vis} has a fixed length,
- The visible message part M_{vis} *ends* with a fixed-length representation of its length,
- DER encoding of ASN.1 types is used for the recoverable message part M_{rec}.

B.1.2 Recommended Parameters

Note that the bit length of the inherent redundancy *inhredBitlen* in the recoverable message part M_{rec} in the message pair (M_{rec}, M_{vis}) to be signed is determined by the established message redundancy criteria. If no such redundancy criteria was established then *inhredBitlen* is zero.

The octet length *padOctlen* should be chosen to be the minimum positive integer such that $8 \cdot padOctlen + inhredBitlen \geq \kappa$, where κ is the desired bit security level. It is also required that $1 \leq padOctlen \leq 255$. Therefore, we recommend

$$padOctlen = \min\{\max\{1, \lceil(\kappa - inhredBitlen)/8\rceil\}, 255\}.$$

Also, the elliptic curve domain parameters ($T = (p, a, b, G, n, h)$ or $T = (m, f(x), a, b, G, n, h)$), the hash function, and the symmetric encryption scheme should be chosen to achieve the desired security level from the list of recommended elliptic curve domain parameters, hash functions, and symmetric encryption schemes. See [SEC 2] for recommended elliptic curve domain parameters. See Sections 3.7 and 3.4 for recommended hash functions and symmetric encryption schemes.

B.2 Commentary on ECAOS

B.2.1 Security Considerations

It was proved in [AOS08] that, in the random oracle model, ECAOS is asymptotically secure against existential forgeries under adaptive chosen message attacks if the elliptic curve discrete logarithm problem is intractable and $\max(1/2^{8L_{red}}, 1/2^{8\tilde{L}_{rec}})$ is negligible. In this model and under

these assumptions, a forgery in ECAOS is possible only after one performs $\min(2^{8L_{red}}, 2^{8\tilde{L}_{rec}}) > \min(2^{8L_{red}}, 2^{8\tilde{L}_{rec}^{min}})$ (signature verification) operations.

It is very plausible to assume that the recommended elliptic curve domain parameters and the hash functions in this document satisfy the necessary requirements imposed by the security proofs in [AOS08]. Hence, the specifications given in this document for implementing ECAOS should satisfy the claimed security levels.

B.2.2 Recommended Parameters

Length parameters L_n, L_F, K, L_{red}, and L_{rec}^{min} are required to satisfy the following conditions.

- $L_n \geq 20$,
- $L_F \geq 20$,
- $K \geq 10$,
- $L_{red} \geq 10$,
- $L_{rec}^{min} \geq 0$ and $L_{red} + L_{rec}^{min} \geq 20$.

For κ-bit security level, length parameters L_n, L_F, K, L_{red}, and L_{rec}^{min} are recommended to take the following values:

- $L_n = \lceil 2\kappa/8 \rceil$, $L_F = \lceil 2\kappa/8 \rceil$, $K = \lceil \kappa/8 \rceil$, $L_{red} = \lceil \kappa/8 \rceil$, $L_{rec}^{min} = \lceil \kappa/8 \rceil$.

Also, the elliptic curve domain parameters ($T = (p, a, b, G, n, h)$ or $T = (m, f(x), a, b, G, n, h)$), and the hash function should be chosen to achieve the desired security level from the list of recommended elliptic curve domain parameters and hash functions. See [SEC 2] for recommended elliptic curve domain parameters, and Section 3.4 for recommended hash functions.

C ASN.1 Syntax

This section specifies the ASN.1 syntax that should be used when ASN.1 syntax is used to convey parts of signature information. Generally, the ASN.1 syntax needs to be suitably encoded via, for example, DER [X.690]. Several different types of information may need to be conveyed during the operation of the schemes specified in this document.

C.1 Syntax for Message Redundancy Criteria

An ASN.1 syntax for indicating the message redundancy criteria inherent to the visible message part is beyond the scope of this document. The signatory and verifier must agree in an authentic manner upon such an ASN.1 syntax or other means to indicate these message redundancy criteria.

C.2 Syntax for Hash Functions

The syntax for indicating the hash function is inherited from SEC1 [SEC 1]. In particular, the ASN.1 module in this standart imports the type `HashFunction` from the ASN.1 module in [SEC 1], where

```
HashAlgorithm ::= AlgorithmIdentifier {{HashFunctions}}

HashFunctions ALGORITHM ::= {
      {OID sha-1 PARMS NULL} |
      {OID id-sha224 PARMS NULL} |
      {OID id-sha256 PARMS NULL} |
      {OID id-sha384 PARMS NULL} |
      {OID id-sha512 PARMS NULL} ,
      ...  -- Additional hash functions may be added in the future
}
```

and

```
sha-1 OBJECT IDENTIFIER ::= {iso(1) identified-organization(3)
      oiw(14) secsig(3) algorithm(2) 26}
id-sha OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) country(16) us(840)
      organization(1) gov(101) csor(3) nistalgorithm(4) hashalgs(2)}
id-sha224 OBJECT IDENTIFIER ::= {id-sha 4}
id-sha256 OBJECT IDENTIFIER ::= {id-sha 1}
id-sha384 OBJECT IDENTIFIER ::= {id-sha 2}
id-sha512 OBJECT IDENTIFIER ::= {id-sha 3}
```

C.3 Syntax for Encryption and Key Derivation Function

The syntax for indicating the symmetric encryption function is inherited from SEC 1 [SEC 1]. In particular, the ASN.1 module in this standart imports the type `SymmetricEncryption` from the

ASN.1 module in [SEC 1], where

```
SymmetricEncryption ::= AlgorithmIdentifier {{SYMENCSet}}

SYMENCSet ALGORITHM ::= {
      {OID xor-in-ecies} |
      {OID tdes-cbc-in-ecies} |
      {OID aes128-cbc-in-ecies} |
      {OID aes192-cbc-in-ecies} |
      {OID aes256-cbc-in-ecies} |
      {OID aes128-ctr-in-ecies} |
      {OID aes192-ctr-in-ecies} |
      {OID aes256-ctr-in-ecies} ,
      ...  -- Future combinations may be added
}

xor-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 18}
tdes-cbc-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 19}
aes128-cbc-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 20 0}
aes192-cbc-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 20 1}
aes256-cbc-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 20 2}
aes128-ctr-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 21 0}
aes192-ctr-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 21 1}
aes256-ctr-in-ecies OBJECT IDENTIFIER ::= {secg-scheme 21 2}
```

and

```
secg-scheme OBJECT IDENTIFIER ::= {iso(1)
      identified-organization(3) certicom(132) schemes(1)}
```

The syntax for indicating the key derivation function is inherited from ANSI X9.92 [X9.92]. In particular, the ASN.1 module in this standart imports the type `KdfAlgorithm` from the ASN.1 module in [X9.92], where

```
KdfAlgorithm ::= AlgorithmIdentifier {{KdfAlgorithms}}

KdfAlgorithms ALGORITHM ::= {
{OID ecpvs-recommended-kdf} |
{OID ecpvs-recommended-kdf PARMS NULL} |
{OID ecpvs-recommended-kdf PARMS HashAlgorithm},
-- Additional key derivation functions may be added.
}

ecpvs-recommended-kdf OBJECT IDENITIFIER ::= {ansi-X9-92 kdf(2) recommended(0)}
```

and

```
ansi-X9-92 OBJECT IDENITIFIER ::= {iso(1) identified-organization(3) tc68(133)
      country(16) x9(840) x9Standards(9) x9-92(92)}
```

C.4 Syntax for Identifying ECPVS Signatures and Its Parameters

To indicate that an ECPVS signature has been employed, for example in an X.509 certificate or CRL, the following object identifier `ecpvs`, which is inherited from ANSI X9.92 [X9.92], shall be used:

```
ecpvs OBJECT IDENTIFIER ::= {ansi-X9-92 signatures(4) 1}
```

where the object identifier `ansi-X9-92` represents the root of the tree containing all object identifiers defined in ANSI X9.92 [X9.92], and has the following value

```
ansi-X9-92 OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) tc68(133)
      country(16) x9(840) x9Standards(9) x9-92(92)}
```

When the ecpvs OID appears in the algorithm field of the ASN.1 type `AlgorithmIdentifier`, and the parameters field is absent or a value of type `NULL`, the ECPVS parameters for signature verification must be obtained from other sources, such as the `subjectPublicKeyInfo` field of the certificate of the issuer. Otherwise, the parameters field should have the following ASN.1 type:

```
ECPVSParameters ::= SEQUENCE {
hash [0] HashAlgorithm OPTIONAL,
sym [1] MemAlgorithm,
kdf [2] KdfAlgorithm DEFAULT {OID ecpvs-recommended-kdf},
padOctlen [3] INTEGER OPTIONAL,
redundancy [4] OCTET STRING OPTIONAL
}
```

Certain fields within the type `ECPVSParameters` are optional. The absence of these fields shall be interpreted according to the following rules:

- The absence of the field `hash` indicates that the hash function to be used is the Approved hash function whose security level matches the security level of the public key.
- The absence of the field `padOctlen` indicates that `padOctlen` is determined by another source of information, if such exists, and otherwise, that `padOctlen` is determined implicitly by the first padding octet obtained during the decyption (decoding) operation.
- The absence of the field `redundancy` indicates that the inherent redundancy in the recoverable message part is determined by another source of information, if such a source exists, and otherwise indicates that no inherent redundancy is required.

C.5 Syntax for Identifying ECAOS Signatures and Its Parameters

To indicate that an ECAOS signature has been employed, for example in an X.509 certificate or CRL, the following object identifier `ecaos` shall be used:

```
ecaos OBJECT IDENTIFIER ::= {id-SigType aos(0)}
```

where the object identifier `id-SigType` has the following value

```
id-sigType OBJECT IDENTIFIER ::= {sec3 signatures(0)}
```

and where

```
sec3 OBJECT IDENTIFIER ::= {certicom-arc sec3Arc(12)}


certicom-arc OBJECT IDENTIFIER ::= {
      iso(1) identified-organization(3) certicom(132)
}
```

When the `ecaos` OID appears in the algorithm field of the ASN.1 type `AlgorithmIdentifier`, and the parameters field is absent or a value of type `NULL`, the ECAOS parameters for signature verification must be obtained from other sources, such as the `subjectPublicKeyInfo` field of the certificate of the issuer. Otherwise, the parameters field should have the following ASN.1 type:

```
ECAOSParameters ::= SEQUENCE{
      hash [0] HashAlgorithm OPTIONAL,
      K [1] INTEGER OPTIONAL, -- octet length of the extended part of a mask string
--
      Lred [2] INTEGER OPTIONAL, -- octet length of added redundancy --
      Lrecmin [3] INTEGER OPTIONAL-- octet length of min.  recoverable message --
}
```

Certain fields within the type `ECAOSParameters` are optional. The absence of these fields shall be interpreted according to the following rules:

- The absence of the field `hash` indicates that the hash function to be used is the Approved hash function whose security level matches the security level of the public key.
- The absence of the field `K` indicates that the the integer K should be chosen such that its length matches the security level of the signature scheme.
- The absence of the field `Lred` indicates that the the integer L_{red} should be chosen such that its length matches the security level of the signature scheme.
- The absence of the field `Lrecmin` indicates that the the integer L_{rec}^{min} should be chosen such that its length matches the security level of the signature scheme.

C.6 Syntax for ECPVS Signature Values

When a digital signature is identified by the OID `ecpvs`, the digital signature shall be ASN.1 encoded using the following syntax:

```
ECPVS-Sig-Value ::= SEQUENCE {
r OCTET STRING,
Mvis OCTET STRING,
s INTEGER
}
```

C.7 Syntax for ECAOS Signature Values

When a digital signature is identified by the OID `ecaos`, the digital signature shall be ASN.1 encoded using the following syntax:

```
ECAOS-Sig-Value ::= SEQUENCE{
      r OCTET STRING,
      Mvis OCTET STRING,
      s INTEGER
}
```

C.8 ASN.1 Module

The following comprises the ASN.1 module for all the items specified in this standard, including those that may have been defined in other modules.

```
SEC3-v1-0 {
      iso(1) identified-organization(3) certicom(132) module(1) ver(3) sec3(0)
}
DEFINITIONS EXPLICIT TAGS ::= BEGIN
      --
      -- EXPORTS ALL;
      --


IMPORTS SymmetricEncryption, HashAlgorithm FROM
      SEC1-v1-9{iso(1) identified-organization(3) certicom(132) module(1) ver(2)};


IMPORTS KdfAlgorithm FROM
      ANSI-X9-92{iso(1) identified-organization(3) tc68(133) country(16)
      x9(840) x9Standards(9) x9-92(92) module(0) 1};


ALGORITHM ::= CLASS{
      &id OBJECT IDENTIFIER UNIQUE
      &Type OPTIONAL
}
WITH SYNTAX{OID &id [PARMS &Type]}


AlgorithmIdentifier{ALGORITHM:IOSet } ::= SEQUENCE {
      algorithm ALGORITHM.&id({IOSet}),
      parameters ALGORITHM.&Type({IOSet}{@algorithm}) OPTIONAL
}


ECPVSAlgorithm ::= AlgorithmIdentifier{{ECPVSAlgorithmSet}}
```

```
ECPVSAlgorithmSet ALGORITHM ::= {
      {OID ecpvs PARMS ECPVSParameters},
      ...  -- future extensions may be possible }

-- This is (ecpvs = id-sigType 1) in ANSI-X9.92 --
ecpvs OBJECT IDENTIFIER ::= {ansi-X9-92 signatures(4) 1}

-- This is ansi-X9-92 in ANSI-X9.92 --
ansi-X9-92 OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) tc68(133)
      country(16) x9(840) x9Standards(9) x9-92(92)}

ECPVSParameters ::= SEQUENCE{
      hash [0] HashAlgorithm OPTIONAL,
      sym [1] SymmetricEncryption,
      kdf [2] KdfAlgorithm DEFAULT {OID ecpvs-recomended-kdf},
      padOctlen [3] INTEGER OPTIONAL,
      redundancy [4] OCTET STRING OPTIONAL
}
-- This is ecpvs-recomended-kdf in ANSI-X9.92 --
ecpvs-recomended-kdf OBJECT IDENTIFIER ::= {ansi-X9-92 kdf(2) recommended(0)}

ECPVS-Sig-Value ::= SEQUENCE{
      r OCTET STRING,
      Mvis OCTET STRING,
      s INTEGER
}
ECAOSAlgorithm ::= AlgorithmIdentifier{{ECAOSAlgorithmSet}}

ECAOSAlgorithmSet ALGORITHM ::= {
      {OID ecaos PARMS ECAOSParameters},
      ...  -- future extensions may be possible }

ecaos OBJECT IDENTIFIER ::= {id-sigType aos(0)}

id-SigType OBJECT IDENTIFIER ::= {sec3 signatures(0)}

sec3 OBJECT IDENTIFIER ::= {certicom-arc sec3Arc(12)}

-- This is certicom-arc from SEC2 --
certicom-arc OBJECT IDENTIFIER ::= {
      iso(1) identified-organization(3) certicom(132)
```

```
}

ECAOSParameters ::= SEQUENCE{
      hash [0] HashAlgorithm OPTIONAL,
      K [1] INTEGER OPTIONAL, -- octet length of the extended part of a mask string
--
      Lred [2] INTEGER OPTIONAL, -- octet length of added redundancy --
      Lrecmin [3] INTEGER OPTIONAL-- octet length of min.  recoverable message --
}

ECAOS-Sig-Value ::= SEQUENCE{
      r OCTET STRING,
      Mvis OCTET STRING,
      s INTEGER
}

END
```

D References

[800-57] E. Barker, W. Barker, W. Burr, W. Polk and M. Smid. *DRAFT Recommendation for Key Management: Part 1: General*, Special Publication 800-57. National Institute of Standards and Technology, May 2011. `http://csrc.nist.gov/publications/`.

[800-90] E. Barker and J. Kelsey. *Recommendation for Random Number Generation Using Deterministic Bit Generators*, Special Publication 800-90. National Institute of Standards and Technology, Mar. 2007. `csrc.nist.gov/groups/ST/toolkit/random_number.html`.

[1363A] Institute of Electrical and Electronics Engineers. *Specifications for Public-Key Cryptography — Amendment 1: Additional Techniques*, IEEE Standard 1363A-2004, Oct. 2004. `http://standards.ieee.org/catalog/olis/busarch.html`.

[9796-3] International Standards Organization. *Information technology — Security techniques Algorithms — Digital signature schemes giving message recovery — Part 3: Discrete logarithm based mechanisms*, International Standard 9796-3, 2006.

[SEC 1] Standards for Efficient Cryptography Group. *SEC 1: Elliptic Curve Cryptography*, Sep. 2000. Version 2.0. `http://www.secg.org/download/aid-780/sec1-v2.pdf`.

[SEC 2] ———. *SEC 2: Recommended Elliptic Curve Domain Parameters*, Sep. 2000. Version 1.0. `http://www.secg.org/download/aid-386/sec2_final.pdf`.

[X9.82] American National Standards Institute. *Random Number Generation*, Draft American National Standard X9.82, 2005. Tentative organization: Part 1: Overview; Part 2: Entropy Sources; Part 3: Deterministic Algorithms; Part 4: Complete Systems.

[X9.92] ———. *Public-Key Cryptography for the Financial Services Industry: Digital Signature Algorithms Providing Partial Message Recovery: Part 1: Elliptic Curve Pintsov-Vanstone Signatures (ECPVS)*, Draft American National Standard X9.92-2002, 2002.

[X.690] International Telecommunications Union. *ITU-T Recommendation X.690: Information Technology — ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER)*, Dec. 1997. Equivalent to ISO/IEC 8825-1.

[AOS08] M. Abe, T. Okamoto and K. Suzuki. *Message recovery signature schemes from sigma-protocols.* In *NTT Technical Review 2008 January.* 2008. `https://www.ntt-review.jp/archive/ntttechnical.php?contents=ntr200801sp2.html`.

[BJ01] D. R. L. Brown and D. B. Johnson. *Formal security proofs for a signature scheme with partial message recovery.* In D. Naccache (ed.), *Topics in Cryptology — CT-RSA 2001*, Lecture Notes in Computer Science 2020, pp. 126–142. Springer, Apr. 2001.

[WYY05a] X. Wang, A. C. Yao and F. Yao. *Cryptanalysis of SHA-1 hash function*. In National Institute of Standards and Technology (ed.), *1st Cryptographic Hash Workshop*. Oct. 2005. `http://www.csrc.nist.gov/pki/HashWorkshop/2005/program.htm`.

[WYY05b] X. Wang, Y. L. Yin and H. Yu. *Finding collisions in the full SHA-1*. In V. Shoup (ed.), *Advances in Cryptology — CRYPTO 2005*, Lecture Notes in Computer Science 3621, pp. 17–36. International Association for Cryptologic Research, Springer, Aug. 2005.