Suite B Implementer's Guide to FIPS 186-3 (ECDSA)

February 3, 2010

1 Introduction

This document specifies the Elliptic Curve Digital Signature Algorithm (ECDSA) from the "Digital Signature Standard" [FIPS186-3] that will be used in future and existing cryptographic protocols for Suite B products. It also includes the Suite B elliptic curve domain parameters, (NIST curves P-256 and P-384), along with example data for the ECDSA signature algorithm on these curves and auxiliary functions that are necessary for ECDSA implementations to be in compliance with [FIPS186-3] and Suite B.

[FIPS186-3] defines methods for digital signature generation that can be used for the authentication of binary data (commonly called a message), and for the verification and validation of those digital signatures. One of the approved techniques is the Elliptic Curve Digital Signature Algorithm (ECDSA) but additional requirements are specified.

This document includes requirements for obtaining the assurances necessary for valid digital signatures. Methods for obtaining these assurances are provided in the NIST Special Publication [SP800-89].

2 Glossary of Terms, Acronymns and Mathematical Symbols

These definitions, acronyms and symbols are from [FIPS186-3]. Definitions, acronyms and symbols from [FIPS186-3] that are not used in this guide have been omitted. In some cases, modifications have been made that are specific to Suite B.

2.1 Terms and definitions

Approved	FIPS-approved and/or NIST-recommended. An algorithm or technique that is either 1. specified in a FIPS or NIST Recommendation, or 2. adopted in a FIPS or NIST Recommendation, or 3. specified in a list of NIST approved security functions.
Assurance of public key validity	Confidence that the public key is arithmetically correct.
Bit string	An ordered sequence of 0's and 1's. The leftmost bit is the most significant bit of the string. The rightmost bit is the least significant bit of the string.
Certificate	A set of data that uniquely identifies a key pair and an owner that is authorized to use the key pair. The certificate contains the owner's public key and possibly other information, and is digitally signed by a Certification Authority (i.e., a trusted party), thereby binding the public key to the owner.

Certification Authority (CA)	The entity in a Public Key Infrastructure (PKI) that is responsible for issuing certificates and exacting compliance with a PKI policy.
Digital signature	The result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity and signatory non-repudiation.
Domain parameters	Parameters used with cryptographic algorithms that are usually common to a domain of users. An ECDSA cryptographic key pair is associated with a specifc set of domain parameters.
Entity	An individual (person), organization, device or process. Used interchangeably with "party".
Hash function	A function that maps a bit string of arbitrary length to a fixed length bit string. Approved hash functions satisfy the following properties: 1. (one-way) it is computationally infeasible to find any input that maps to any new pre-specified output, and 2. (collision resistant) it is computationally infeasible to find any two distinct inputs that map to the same output. Approved hash functions for Suite B are SHA-256 and SHA-384, which are specified in [FIPS180-3].
Hash value	See "message digest".

Key	A parameter used in conjunction with a cryptographic algorithm that determines its operation. Examples applicable to this document include: 1. the computation of a digital signature from data, and 2. the verification of a digital signature.
Key pair	A public key and its corresponding private key.
Message	The data that is signed. Also known as "signed data" during the signature verification and validation process.
Message digest	The result of applying a hash function to a message. Also known as "hash value".
Non-repudiation	A service that is used to provide assurance of the integrity and origin of data in such a way that the integrity and origin can be verified and validated by a third party as having originated from a specific entity in possession of the private key (i.e., the signatory).
Owner	A key pair owner is the entity that is authorized to use the private key of a key pair.
Party	An individual (person), organization, device or process. Used interchangeably with "entity".
Per-message secret number	A secret random number that is generated prior to the generation of each digital signature.

Private key	A cryptographic key that is used with an asymmetric (public key) cryptographic algorithm. For digital signatures, the private key is uniquely associated with the owner and is not made public. The private key is used to compute a digital signature that may be verified using the corresponding public key.
Public key	A cryptographic key that is used with an asymmetric (public key) cryptographic algorithm and is associated with a private key. The public key is associated with an owner and may be made public. In the case of digital signatures, the public key is used to verify a digital signature that was signed using the corresponding private key.
Random number generator	A device or algorithm that can produce a sequence of random numbers that appears to be statistically independent and unbiased.
Security strength	A number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system. Sometimes referred to as a security level.
Shall	Used to indicate a requirement of [FIPS186-3].
Should	Used to indicate a strong recommendation, but not a requirement of [FIPS186-3].
Signatory	The entity that generates a digital signature on data using a private key.
Signature generation	The process of using a digital signature algorithm and a private key to generate a digital signature on data.
Signature verification	The process of using a digital signature algorithm and a public key to verify a digital signature on data.

Signed data	The data or message upon which a digital signature has been computed. Also, see "message".
Trusted third party	An entity other than the owner and verifier that is trusted by the owner or the verifier or both. Sometimes shortened to "trusted party".
Verifier	The entity that verifies the authenticity of a digital signature using the public key.

2.2 Acronyms

CA	Certification Authority.
EC	Elliptic curve.
ECC	Elliptic curve cryptography.
ECDSA	Elliptic Curve Digital Signature Algorithm; specified in [ANS-X9.62-2005].
FIPS	Federal Information Processing Standard.
NIST	National Institute of Standards and Technology.
RBG	Random Bit Generator; specified in [SP800-90].
SHA	Secure Hash Algorithm; specified in [FIPS180-3].
SP	NIST Special Publication.

2.3 Mathematical Symbols

$[i, j]$	The interval of integers between and including i and j. For example, $[1, 4]$ consists of the integers $1, 2, 3$ and 4.
q	The size of the underlying field of the elliptic curve. It is a prime number for Suite B elliptic curves.

FR	The field representation indicator, which is $NULL$ for Suite B elliptic curves.
a	An elliptic curve domain parameter, which is equal to the integer $q - 3$ for Suite B elliptic curves.
b	An elliptic curve domain parameter, which for Suite B curves is an integer in $[0, q - 1]$, generated using the elliptic curve domain parameter $SEED$.
$SEED$	A bit string used to generate elliptic curve domain parameter b via the method defined in [ANS-X9.62-2005].
$G = (x_G, y_G)$	The base point on the elliptic curve. The coordinates x_G and y_G are integers in the interval $[0, q - 1]$.
$\mathcal{O}$	The point at infinity of the elliptic curve.
n	The order of the base point G of the elliptic curve; $nG = \mathcal{O}$. The bit length of n is considered to be the key size, and the security strength associated with the use of a base point of order n is one-half the bit length of n.
h	The order of the elliptic curve group divided by the order n of the base point G. For a Suite B elliptic curve, $h = 1$; that is, the base point G generates the entire elliptic curve group.
d	The ECDSA private key, which is an integer in the interval $[1, n - 1]$.
$Q = (x_Q, y_Q)$	An ECDSA public key. The coordinates x_Q and y_Q are integers in the interval $[0, q - 1]$, and $Q = dG$.
k	The ECDSA per-message secret number, which is an integer in the interval $[1, n - 1]$.

$R = (x_R, y_R)$	The public value from the ECDSA ephemeral key pair. ($R = kG$). The coordinates x_R and y_R are integers in the interval $[0, q-1]$.
r	One component of an ECDSA digital signature. It is an integer in $[1, n-1]$. See the definition of (r, s).
s	One component of an ECDSA digital signature. It is an integer in $[1, n-1]$. See the definition of (r, s).
(r, s)	An ECDSA digital signature, where r and s are the digital signature components.
M	The message that is signed using the digital signature algorithm.
Hash(M)	The result of a hash computation (message digest or hash value) on message M using an approved hash function.
$x \bmod m$	The unique integer remainder r, satisfying $0 \leq r \leq (m-1)$, when integer x is divided by the positive integer m. For example, $23 \bmod 7 = 2$.
Compute $y = x \bmod m$	Compute the value of $x \bmod m$ and assign that value to y. For example, compute $y = 23 \bmod 7$ will assign the value 2 to y.
$x^{-1} \bmod m$	The multiplicative inverse of the integer x modulo the positive integer m. Defined only when x is relatively prime to m and $m > 1$, $x^{-1} \bmod m$ is the unique integer y satisfying $0 < y < m$ and $(x * y) \bmod m = 1$. For example, $4^{-1} \bmod 19 = 5$. (See Appendix B.1.) In this document, when the context is clear (and the value of m is apparent), the simpler notation x^{-1} may be used instead of $x^{-1} \bmod m$. This should not cause confusion, since we will never use x^{-1} to indicate the rational number $1/x$.

len(x)	The length (in bits) of the (shortest) binary representation of the integer x.
$\lfloor x \rfloor$	The floor of x; the largest integer that is less than or equal to x For example, $\lfloor 5 \rfloor = 5$, $\lfloor 5.3 \rfloor = 5$, and $\lfloor -2.1 \rfloor = -3$.

3 The Elliptic Curve Digital Signature Algorithm (ECDSA)

The standard [ANS-X9.62-2005] defines methods for digital signature generation and verification using the Elliptic Curve Digital Signature Algorithm (ECDSA). These methods are reproduced here, specialized to the Suite B curves P-256 and P-384.

3.1 ECDSA Domain Parameters

ECDSA requires that the private/public key pairs used for digital signature generation and verification be generated with respect to a particular set of elliptic curve domain parameters. These domain parameters may be common to a group of users and may be public. Domain parameters may remain fixed for an extended time period.

ECDSA Domain parameters are given by

- q, the size of the underlying field,
- FR, a field representation indicator, which is $NULL$ for Suite B curves,
- a, elliptic curve parameter, which is equal to $q - 3$ for Suite B curves,
- b, elliptic curve parameter,
- $SEED$, used to generate parameter b,
- $G = (x_G, y_G)$, a point on the curve, known as the base point,

- n, the order of the base point G, and
- h, the order of the elliptic curve group divided by the order n of G; h=1 for Suite B curves.

Suite B allows for two possible sets of elliptic curve domain parameters: the curves P-256 and P-384. These parameters are given in Appendix C.

3.1.1 Domain Parameter Management

Each ECDSA key pair **shall** be correctly associated with one specific set of domain parameters (e.g., by a public key certificate that identifies the domain parameters associated with the public key). The domain parameters **shall** be protected from unauthorized modification until the set is deactivated (if and when the set is no longer needed). The same domain parameters may be used for more than one purpose (e.g., the same domain parameters may be used for both digital signatures and key establishment).

3.2 Private/Public Keys

An ECDSA key pair, (d, Q), consists of a private key d and a public key Q that is associated with a specific set of EC domain parameters; d, Q and the domain parameters are mathematically related to each other via the relation $Q = dG$, where dG indicates the sum (using elliptic curve arithmetic) of d copies of the domain parameter G (see 3.1), also known as the elliptic curve scalar multiply of G by d. The private key d is normally used for a limited period of time (i.e., the cryptoperiod); the public key Q may continue to be used as long as digital signatures that have been generated using the associated private key need to be verified (i.e., the public key may continue to be used beyond the cryptoperiod of the associated private key). See [SP800-57] for further guidance.

ECDSA keys **shall** only be used for the generation and verification of ECDSA digital signatures. ECDSA keys **shall not** be used for any other purpose, e.g., key establishment.

3.2.1 Key Pair Generation

A digital signature key pair (d, Q) is generated using a specific set of domain parameters. Methods for the generation of (d, Q) are provided in Appendix A.1.

3.2.2 Key Pair Validation

Prior to or during the signature verification process, the verifier **shall** obtain the public key Q in a trusted manner. After obtaining the public key the verifier **shall** obtain assurance of the validity of the signer's public key Q as specified in Appendix A.3 using the appropriate set of domain parameters.

3.2.3 Key Pair Management

Guidance on the protection of key pairs is provided in [SP800-57]. The secure use of digital signatures depends on the management of an entity's digital signature key pair as follows:

1. The validity of the domain parameters **shall** be assured prior to the generation of the key pair, or the verification and validation of a digital signature. For Suite B this means that the entity **shall** have an authentic copy of appropriate domain parameters.

2. Each key pair **shall** be associated with the domain parameters under which the key pair was generated.

3. A key pair **shall** only be used to generate and verify signatures using the domain parameters associated with that key pair.

4. The private key **shall** be used only for signature generation as specified in this document and **shall** be kept secret; the public key **shall** be used only for signature verification and may be made public.

5. An intended signatory **shall** have assurance of possession of the private key prior to or concurrently with using it to generate a digital signature (see [SP800-89] section 6).

6. A private key **shall** be protected from unauthorized access, disclosure and modification.

7. A public key **shall** be protected from unauthorized modification (including substitution). For example, public key certificates that are signed by a CA may provide such protection.

8. A verifier **shall** be assured of a binding between the public key, its associated domain parameters and the key pair owner. For example, public key certificates that are signed by a CA may provide such protection.

9. A verifier **shall** obtain public keys in a trusted manner (e.g., from a certificate signed by a CA that the entity trusts, or directly from the intended or claimed signatory, provided that the entity is trusted by the verifier and can be authenticated as the source of the signed information that is to be verified).

10. Verifiers **shall** be assured that the claimed signatory is the key pair owner and that, at the time that the signature was generated, the owner possessed the private key that was used to generate the signature (i.e., the private key that is associated with the public key that will be used to verify the digital signature) (see [SP800-89] Section 6).

11. A signatory and a verifier **shall** have assurance of the validity of the public key (see Appendix A.3).

3.3 Per-Message Secret-Number Generation

A new nonzero secret random number k **shall** be generated prior to the generation of each digital signature for use during the signature generation process. This secret number **shall** be protected from unauthorized disclosure and modification. Methods for the generation of the per-message secret number are provided in Appendix A.2.

The number $k^{-1} \bmod n$, referred to as k^{-1}, is also required for the signature generation process. A technique is provided in Appendix B.1 for deriving k^{-1} from k.

The values k and k^{-1} may be pre-computed, since knowledge of the message to be signed is not required for the computations. When k and k^{-1} are pre-computed, their confidentiality and integrity **shall** be protected.

3.4 ECDSA Digital Signature Generation and Verification

An ECDSA digital signature (r, s) **shall** be generated as specified in 3.4.1 below, using:

1. An authentic copy of the domain parameters for the curve P-256 or P-384 (see Appendix C),
2. a private key d that is generated (along with a public key Q) as specified in Appendix A.1,
3. a per-message secret number k that is generated as specified in Appendix A.2,
4. an approved hash function as discussed below, and
5. an approved random number generator as specified in [SP800-90].

An ECDSA digital signature **shall** be verified as specified in 3.4.2 below, using the same domain parameters and hash function that were used during signature generation.

Suite B requires that SHA-256 be the hash used when the parameter set is P-256, and that SHA-384 be the hash used when the parameter set is P-384. This is in conformance with the requirements of [FIPS186-3].

3.4.1 ECDSA Signature Generation

Prior to generating an ECDSA signature, the signatory **shall** obtain

1. an authentic copy of the domain parameters,

2. a digital signature key pair (d, Q), either generated by a method from Appendix A.1, or obtained from a trusted third party,

3. assurance of the validity of the public key Q (see Appendix A.3), and

4. assurance that he/she/it actually possesses the associated private key d (see [SP800-89] Section 6).

The signatory **shall** use the process specified below to sign a message:

Input:

1. a message M, which is the bit string to be signed,

2. the elliptic curve domain parameters,

3. the identity of the hash function selected by the signatory, as determined by the elliptic curve domain parameters, and

4. the elliptic curve private key d employed by the signatory, as generated in Appendix A.1.

Output: The pair of integers (r, s), each in the interval $[1, n-1]$.

Process:

1. Use one of the routines in Appendix A.2 to generate (k, k^{-1}), a per-message secret number and its inverse modulo n. Since n is prime, the output will be invalid only if there is a failure in the RBG.

2. Compute the elliptic curve point $R = kG = (x_R, y_R)$ using EC scalar multiplication (see [Routines]), where G is the base point included in the set of domain parameters.

3. Compute $r = x_R \bmod n$. If $r = 0$, then return to Step 1. [1].

4. Use the selected hash function to compute $H = \mathbf{Hash}(M)$.

5. Convert the bit string H to an integer e as described in Appendix B.2.

[1]The probability that $r = 0$ is approximately $1/n$.

6. Compute $s = (k^{-1} * (e + d * r)) \bmod n$. If $s = 0$, return to Step 1.[2].

7. Return (r, s).

3.4.2 ECDSA Signature Verification

Prior to accepting a verified digital signature as valid the verifier **shall** have

1. assurance of the signatory's claimed identity,

2. an authentic copy of the domain parameters, $(q, FR, a, b, SEED, G, n, h)$,

3. assurance of the validity of the public key (d, Q), and

4. assurance that the claimed signatory actually possessed the private key that was used to generate the digital signature at the time that the signature was generated.

Methods for the verifier to obtain these assurances are provided in [SP800-89].

The following process **shall** be used to verify a purported signature if the verifier is not the purported signatory, and may be used if the verifier is the purported signatory:

Input:

1. M': the received message, which is a bit string,

2. (r', s'): the received signature on M', which is a pair of integers

3. the EC domain parameters used by the signatory,

4. the identity of the hash function selected by the signatory, as determined by the EC domain parameters, and

5. Q: the EC public key chosen by the signatory.

[2]The probability that $s = 0$ is approximately $1/n$

Output: An indication as to whether the purported signature is valid or not – either VALID or INVALID.

Process:

1. If r' and s' are not both integers in the interval $[1, n-1]$, output INVALID.
2. Use the selected hash function to compute $H' = \mathbf{Hash}(M')$.
3. Convert the bit string H' to an integer e' as described in Appendix B.2.
4. Compute $w = (s')^{-1} \bmod n$, using the routine in Appendix B.1.
5. Compute $u_1 = (e' * w) \bmod n$, and compute $u_2 = (r' * w) \bmod n$.
6. Compute the elliptic curve point $R = (x_R, y_R) = u_1 G + u_2 Q$, using EC scalar multiplication and EC addition (see [Routines]). If R is equal to the point at infinity $\mathcal{O}$, output INVALID.
7. Compute $v = x_R \bmod n$.
8. Compare v and r'. If $v = r'$, output VALID; otherwise, output INVALID.

This concludes the presentation of the ECDSA signature algorithms for the Suite B curves. Additional material is presented in the following appendices:

Appendix A: Key Pair Generation and Validation
Appendix B: Generation of Other Quantities
Appendix C: The NIST Curves P-256 and P-384
Appendix D: Example Data

A Key Pair Generation and Validation

This appendix specifies methods for the generation of ECC key pairs and secret numbers. All generation methods require the use of an approved, properly instantiated random bit generator (RBG) as specified in [SP800-90]; the RBG **shall** have a security strength equal to or greater than the security

strength associated with the key pairs and secret numbers to be generated. Security strengths for Suite B are 128 bits for P-256 and 192 bits for P-384.

An approved process to provide assurance of public key validity is also given. Assurance of public key validity is inherent in both key generation methods, hence additional steps to provide assurance of public-key validation immediately after generation are not required.

This appendix does not indicate the required conversions between bit strings and integers. When required by a process in this appendix, the conversion shall be accomplished as specified in Appendix B.2.

See [Routines] for definitions of elliptic curve arithmetic and appropriate algorithms.

In addition to their desired output, each of the generation routines in this appendix returns a value *status*, which will be set to either SUCCESS or ERROR, where, in practice, SUCCESS will be some appropriate indication of successful completion of the routine, and ERROR will be some appropriate indication of an error condition. For example, if an error indication is returned by the call to the RBG, then *status* is set to ERROR, where ERROR is an appropriate indication of the RBG failure.

The routines presented in this appendix are are based on [FIPS186-3] Appendix B, sections B.4 and B.5.

A.1 ECC Key Pair Generation

Given a set of domain parameters $(q, FR, a, b, SEED, G, n, h)$, one can generate an ECC key pair (d, Q). Two methods are provided for the generation of the ECC key pairs (d, Q) (from Section 3.4.2); one of these two methods **shall** be used to generate (d, Q). Prior to generating ECC key pairs, an authentic copy of the domain parameters $(q, FR, a, b, SEED, G, n, h)$ **shall** be obtained.

A.1.1 ECC Key Pair Generation Using Extra Random Bits

In this method, 64 more bits are requested from the RBG than are needed for d so that bias produced by the modular reduction in step 6 is negligible.

Input: $(q, FR, a, b, SEED, G, n, h)$: domain parameters.

Output:

1. *status*: the status returned, either SUCCESS or ERROR, from the key pair generation procedure, and
2. (d, Q): the generated private and public keys. If an error is encountered during the generation process, pair of invalid values for d and Q, represented by $(Invalid\_d, Invalid\_Q)$, **should** be returned. The generated private key d is an integer in the range $[1, n-1]$, and Q is an elliptic curve point.

Process:

1. Set $N = \mathbf{len}(n)$. Check that N is valid, that is, $N = 256$ or $N = 384$ (these are the only valid lengths for Suite B).
2. If N is invalid, then return ERROR; the pair $(Invalid\_d, Invalid\_Q)$ **should** also be returned.
3. Set *requested_security_strength* to be the security strength associated with N, which is 128 when using P-256 or 192 when using P-384.
4. Obtain a string of $N + 64$ *returned_bits* from an RBG with a security strength of *requested_security_strength* or more. If an error indication is returned by the RBG, then return ERROR; the pair $(Invalid\_d, Invalid\_Q)$ **should** also be returned.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix B.2).
6. Set $d = (c \bmod (n-1)) + 1$.
7. Compute $Q = dG$ using EC scalar multiplication (see [Routines]).
8. Return SUCCESS and (d, Q).

A.1.2 ECC Key Pair Generation by Testing Candidates

In this method, a random number is obtained and tested to determine that it will produce a value of d in the correct range. If d is out-of-range, the process is iterated until an acceptable value of d is obtained.

Input: $(q, FR, a, b, SEED, G, n, h)$: domain parameters.

Output:

1. *status*: the status returned, either SUCCESS or ERROR, from the key pair generation procedure, and
2. (d, Q): the generated private and public keys. If an error is encountered during the generation process, a pair of invalid values for d and Q, represented by $(Invalid\_d, Invalid\_Q)$, **should** be returned. The generated private key d is an integer in the range $[1, n-1]$, and Q is an elliptic curve point.

Process:

1. Set $N = \mathbf{len}(n)$. Check that N is valid, that is, $N = 256$ or $N = 384$ (these are the only valid lengths for Suite B).
2. If N is invalid, then return ERROR; the pair $(Invalid\_d, Invalid\_Q)$ **should** also be returned.
3. Set *requested_security_strength* to be the security strength associated with N, which is 128 when using P-256 and 192 when using P-384.
4. Obtain a string of N *returned_bits* from an RBG with a security strength of *requested_security_strength* or more. If an error indication is returned by the RBG, then return ERROR; the pair $(Invalid\_d, Invalid\_Q)$ **should** also be returned.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix B.2).
6. If $(c > n-2)$, then go to step 4.

7. $d = c + 1$.

8. Compute $Q = dG$ using EC scalar multiplication (see [Routines]).

9. Return SUCCESS and (d, Q).

A.2 ECC Per-Message Secret-Number Generation

ECDSA requires the generation of a new non-zero secret random number k for each message to be signed. Two methods are provided for the generation of k; one of these two methods **shall** be used.

Let **inverse**(k, m) denote a function that computes the inverse of a positive integer k with respect to multiplication modulo the number m.

A.2.1 ECC Per-Message Secret Number Generation Using Extra Random Bits

In this method, 64 more bits are requested from the RBG than are needed for k so that bias produced by the modular reduction in step 6 is not readily apparent.

Input: $(q, FR, a, b, SEED, G, n, h)$: domain parameters.

Output:

1. *status*: the status returned, either SUCCESS or ERROR, from the secret number generation procedure, and

2. (k, k^{-1}): the generated secret number k and its inverse k^{-1}, which are integers in the range $[1, n - 1]$. If an error is encountered during the generation process, a pair of invalid values for k and k^{-1}, represented by $(Invalid\_k, Invalid\_k\_inverse)$, **should** be returned.

Process:

1. Set $N =$ **len**(n). Check that N is valid, that is, $N = 256$ or $N = 384$ (these are the only valid lengths for Suite B).

2. If N is invalid, then return ERROR; the pair $(Invalid\_k, Invalid\_k\_inverse)$ **should** also be returned.

3. Set *requested_security_strength* to be the security strength associated with N, which is 128 when using P-256 and 192 when using P-384.

4. Obtain a string of $N + 64$ *returned_bits* from an RBG with a security strength of *requested_security_strength* or more. If an error indication is returned by the RBG, then return ERROR; the pair $(Invalid\_k, Invalid\_k\_inverse)$ **should** also be returned.

5. Convert *returned_bits* to the (non-negative) integer c (see Appendix B.2).

6. $k = (c \bmod (n-1)) + 1$.

7. $(status, k^{-1}) = \textbf{inverse}(k, n)$.

8. Return *status* and (k, k^{-1}).

A.2.2 ECC Per-Message Secret Number Generation by Testing Candidates

In this method, a random number is obtained and tested to determine that it will produce a value of k in the correct range. If k is out-of-range, another random number is obtained (i.e., the process is iterated until an acceptable value of k is obtained).

Input: $(q, FR, a, b, SEED, G, n, h)$: domain parameters.

Output:

1. *status*: the status returned, either SUCCESS or ERROR, from the secret number generation procedure, and

2. (k, k^{-1}): the generated secret number k and its inverse k^{-1}, which are integers in the range $[1, n-1]$. If an error is encountered during the generation process, a pair of invalid values for k and k^{-1}, represented by $(Invalid\_k, Invalid\_k\_inverse)$, **should** be returned.

Process:

1. Set $N = \mathbf{len}(n)$. Check that N is valid, that is, $N = 256$ or $N = 384$ (these are the only valid lengths for Suite B).
2. If N is invalid, then return ERROR; the pair $(Invalid\_k, Invalid\_k\_inverse)$ **should** also be returned.
3. Set *requested_security_strength* to be the security strength associated with N, which is 128 when using P-256 and 192 when using P-384.
4. Obtain a string of N *returned_bits* from an RBG with a security strength of *requested_security_strength* or more. If an error indication is returned by the RBG, then return ERROR; the pair $(Invalid\_k, Invalid\_k\_inverse)$ **should** also be returned.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix B.2).
6. If $(c > n - 2)$ then go to step 4.
7. $k = c + 1$.
8. $(status, k^{-1}) = \mathbf{inverse}(k, n)$.
9. Return *status* and (k, k^{-1}).

A.3 Assurance of Public Key Validity

The following routine, ECC Partial Public-Key Validation, from [SP800-56A] **shall** be executed prior to executing a routine that performs elliptic curve computations on a public key.

Note: ECC *Full* Public-Key Validation includes an additional check to ensure that the point has the correct order. This check is not necessary for curves having prime order (and cofactor $h = 1$), such as P-256 and P-384. As long as the implementation under testing claims to support only the Suite B subset of the NIST curves, the partial validation routine will be sufficient to satisfy FIPS 140 CAVP testing of both full and partial public key validation capabilities.

Input:

1. $(q, FR, a, b, SEED, G, n, h)$: domain parameters, and
2. $Q = (x_Q, y_Q)$, a candidate ECC public key.

Process:

1. Verify that Q is not the point at infinity $\mathcal{O}$. This can be done by inspection if the point is entered in the standard affine representation. If the point can be represented in the standard affine representation, then the point is not the point at infinity.
2. Verify that x_Q and y_Q are integers in the interval $[0, q-1]$. This ensures that each coordinate of the public key has the unique correct representation of an element in the underlying field.
3. Verify that

$$((y_Q)^2) \bmod q = ((x_Q)^3 + a * x_Q + b) \bmod q$$

where a, b, q are the values indicated in the domain parameters used. This ensures that the public key is on the correct elliptic curve.

Note that in [SP800-56A] the interval in step 2 is specified as $[0, p-1]$ and the modulus in step 3 is specified as p. For Suite B curves, $q = p$.

Output: If any of the above checks fail, then output an error indicator. Otherwise, output an indication of validation success.

B Generation of Other Quantities

This appendix contains routines for supplementary processes required for the implementation of this document. Appendix B.1 provides a method to produce the inverse of the per-message secret k (see Section 3.4.1 and Appendices A.2.1 and A.2.2) and the inverse of the signature portion s' that is used during signature verification (see Section 3.4.2). The routine Appendix B.2 is required to convert between bit strings and integers in Appendices A.1, A.2 and Section 3.4.

B.1 Computation of the Inverse Value

This algorithm or an algorithm that produces an equivalent result **shall** be used to compute the multiplicative inverse $z' = z^{-1} \bmod m$, where $1 \leq z \leq (m-1)$, $1 \leq z' \leq (m-1)$, and both z and z' are relatively prime to m. In this document, z is either k in the case of signature generation or s' in the case of signature verification, and m is the EC domain parameter n.

In addition to the inverse value this algorithm returns a value *status*, which will be set to either SUCCESS or ERROR, where, in practice, SUCCESS will be some appropriate indication of successful completion of the routine, and ERROR will be some appropriate indication of an error condition. If an error is encountered during the routine, an invalid value for z', represented by $Invalid\_z\_prime$, **should** be returned.

Input:

1. z: the value to be inverted modulo m (i.e., either k or s'), and
2. m: the modulus.

Output:

1. *status*: the status returned, either SUCCESS or ERROR, and
2. z': the multiplicative inverse of z modulo m, if it exists.

Process:

1. Verify that m and z are positive integers such that $z < m$; if not, return ERROR; the value $Invalid\_z\_prime$ **should** also be returned.
2. Set $i = m, j = z, y_2 = 0$ and $y_1 = 1$.
3. $quotient = \lfloor i/j \rfloor$.
4. $remainder = i - (j * quotient)$.
5. $y = y_2 - (y_1 * quotient)$.

6. Set $i = j$, $j = remainder$, $y_2 = y_1$, and $y_1 = y$.
7. If $j > 0$, then go to step 3.
8. If $i \neq 1$, then return ERROR; the value $Invalid\_z\_prime$ **should** also be returned.
9. Compute $z' = y_2 \bmod m$.
10. Return SUCCESS and z'.

B.2 Conversion of a Bit String to an Integer

An M-long sequence of bits $x_1, \ldots, x_M$ is converted to an integer by the rule

$$x_1, \ldots, x_M \rightarrow (x_1 * 2^{M-1}) + (x_2 * 2^{M-2}) + \cdots + (x_{M-1} * 2) + x_M$$

Note that the first bit of a sequence corresponds to the most significant bit of the corresponding integer, and the last bit corresponds to the least significant bit.

Input:

1. $x_1, x_2, ..., x_M$, which is the bit string to be converted.

Output:

1. C, the requested integer representation of the bit string.

Process:

1. Let $x_1, x_2, ..., x_M$ be the bits in the bit string to be converted, from leftmost to rightmost.
2. $C = \sum_{i=1}^{M} 2^{M-i} * x_i$.
3. Return C.

C The NIST Curves P-256 and P-384

Domain parameters for ECC schemes are of the form

$$(q, FR, a, b, SEED, G, n, h),$$

where q is the field size; FR is an indication of the basis used; a and b are two field elements that define the equation of the curve; $SEED$ is an optional bit string that is present if the elliptic curve was randomly generated in a verifiable fashion; G is a generating point of prime order on the curve (i.e., $G = (x_G, y_G)$); n is the order of the point G, and h is the cofactor (which is equal to the order of the curve divided by n).

The equation of the curve is generally given as

$$y^2 = x^3 + ax + b \mod q.$$

For the Suite B curves, $a = q - 3$, and with this value of a, the equation is equivalent to the one given in [FIPS186-3], namely

$$y^2 = x^3 - 3x + b \mod q.$$

Suite B requires the use of one of the following two sets of domain parameters. All values are given in hexadecimal notation.

C.1 Domain Parameters for Curve P-256

Field size:

$q =$ `ffffffff 00000001 00000000 00000000 00000000 ffffffff`
`ffffffff ffffffff`

Field Representation indicator:

$FR =$ `NULL`

Curve parameter:

$a =$ `ffffffff 00000001 00000000 00000000 00000000 ffffffff ffffffff fffffffc`

Curve parameter:

$b =$ `5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b`

Seed used to generate parameter b:

$SEED =$ `c49d3608 86e70493 6a6678e1 139d26b7 819f7e90`

x-coordinate of base point G:

$x_G =$ `6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296`

y-coordinate of base point G:

$y_G =$ `4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5`

Order of the point G:

$n =$ `ffffffff 00000000 ffffffff ffffffff bce6faad a7179e84 f3b9cac2 fc632551`

Cofactor (order of the elliptic curve group divided by the order of the point G):

$h =$ `1`

C.2 Domain Parameters for Curve P-384

Field size:

$q =$ `ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff`
`ffffffff fffffffe ffffffff 00000000 00000000 ffffffff`

Field Representation indicator:

$FR =$ `NULL`

Curve parameter:

$a =$ `ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff`
`ffffffff fffffffe ffffffff 00000000 00000000 fffffffc`

Curve parameter:

$b =$ `b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112`
`0314088f 5013875a c656398d 8a2ed19d 2a85c8ed d3ec2aef`

Seed used to generate parameter b:

$SEED =$ `a335926a a319a27a 1d00896a 6773a482 7acdac73`

x-coordinate of base point G:

$x_G =$ `aa87ca22 be8b0537 8eb1c71e f320ad74 6e1d3b62 8ba79b98`
`59f741e0 82542a38 5502f25d bf55296c 3a545e38 72760ab7`

y-coordinate of base point G:

$y_G =$ `3617de4a 96262c6f 5d9e98bf 9292dc29 f8f41dbd 289a147c`
`e9da3113 b5f0b8c0 0a60b1ce 1d7e819d 7a431d7c 90ea0e5f`

Order of the point G:

$n =$ `ffffffff ffffffff ffffffff ffffffff ffffffff ffffffff`
`c7634d81 f4372ddf 581a0db2 48b0a77a ecec196a ccc52973`

Cofactor (order of the elliptic curve group divided by the order of the point G):

$h =$ `1`

D Example Data

This section gives example data for the ECDSA signature generation and verification for the Suite B curves P-256 and P-384. All values are given in hexadecimal notation, with the exception of the output values of the conversion of hash output bit strings (denoted H, H') into integers (denoted e, e'), which are given in base ten notation.

D.1 Example ECDSA Signature for P-256

D.1.1 Key Generation

The private key d:

$d =$ `70a12c2d b16845ed 56ff68cf c21a472b 3f04d7d6 851bf634 9f2d7d5b 3452b38a`

The public key $Q = dG = (x_Q, y_Q)$:

$x_Q =$ `8101ece4 7464a6ea d70cf69a 6e2bd3d8 8691a326 2d22cba4 f7635eaf f26680a8`

$y_Q =$ `d8a12ba6 1d599235 f67d9cb4 d58f1783 d3ca43e7 8f0a5aba a6240799 36c0c3a9`

D.1.2 Signature Generation

1. Generate (k, k^{-1}), a per-message secret and its inverse modulo n:

   $k =$ `580ec00d 85643433 4cef3f71 ecaed496 5b12ae37 fa47055b 1965c7b1 34ee45d0`

   $k^{-1} =$ `6a664fa1 15356d33 f16331b5 4c4e7ce9 67965386 c7dcbf29 04604d0c 132b4a74`

2. Set $R = (x_R, y_R) = kG$ (elliptic curve arithmetic):

$x_R =$ `7214bc96 47160bbd 39ff2f80 533f5dc6 ddd70ddf 86bb8156 61e805d5 d4e6f27c`

$y_R =$ `8b81e3e9 77597110 c7cf2633 435b2294 b7264298 7defd3d4 007e1cfc 5df84541`

3. Compute $r = x_R \bmod n$:

   $r =$ `7214bc96 47160bbd 39ff2f80 533f5dc6 ddd70ddf 86bb8156 61e805d5 d4e6f27c`

4. Message M="This is only a test message. It is 48 bytes long". $H =$ **Hash**(M):

   $M =$ `54686973 20697320 6f6e6c79 20612074 65737420 6d657373 6167652e 20497420 69732034 38206279 74657320 6c6f6e67`

   $H =$ `7c3e883d dc8bd688 f96eac5e 9324222c 8f30f9d6 bb59e9c5 f020bd39 ba2b8377`

5. Conversion of bit string H to integer e (shown in base ten):

   $e =$ `56197278047627432394583341962843287937266210957576322469816113796290471232375`

6. Compute $s = (k^{-1} * (e + d * r)) \bmod n$:

   $s =$ `7d1ff961 980f961b daa3233b 6209f401 3317d3e3 f9e14935 92dbeaa1 af2bc367`

The signature on the message M is the pair (r, s).

D.1.3 Signature Verification

The receiver receives a message M', along with a signature (r', s').

1. Validation checking on r', s':

$r' =$ `7214bc96 47160bbd 39ff2f80 533f5dc6 ddd70ddf 86bb8156 61e805d5 d4e6f27c`

$s' =$ `7d1ff961 980f961b daa3233b 6209f401 3317d3e3 f9e14935 92dbeaa1 af2bc367`

r', s' are in the range $[1, n-1]$.

2. $H' = \mathbf{Hash}(M')$

   $H' =$ `7c3e883d dc8bd688 f96eac5e 9324222c 8f30f9d6 bb59e9c5 f020bd39 ba2b8377`

3. Integer e' derived from H' (shown in base ten):

   $e' =$ `56197278047627432394583341962843287937266210957576322469816113796290471232375`

4. Compute $w = (s')^{-1} \bmod n$:

   $w =$ `d69be75f 67ee5394 cabb6c28 6f3610cf 62d722cb a9eea70f aee770a6 b2ed72dc`

5. Compute $u_1 = (e' * w) \bmod n$, and compute $u_2 = (r' * w) \bmod n$:

   $u_1 =$ `bb252401 d6fb322b b747184c f2ac52bf 8d54b95a 1515062a 2f6141f2 e2092ed8`

   $u_2 =$ `aae7d1c7 f2c232df c641948a f3dba141 d4de8634 e571cf84 c486301b 510cfc04`

6. $R = (x_R, y_R) = u_1 G + u_2 Q$ (elliptic curve arithmetic):

   $x_R =$ `7214bc96 47160bbd 39ff2f80 533f5dc6 ddd70ddf 86bb8156 61e805d5 d4e6f27c`

   $y_R =$ `8b81e3e9 77597110 c7cf2633 435b2294 b7264298 7defd3d4 007e1cfc 5df84541`

7. Compute $v = x_R \bmod n$:

   $v =$ `7214bc96 47160bbd 39ff2f80 533f5dc6 ddd70ddf 86bb8156 61e805d5 d4e6f27c`

8. Compare v and r': Successfully Verified.

D.2 Example ECDSA Signature for P-384

D.2.1 Key Generation

The private key d:

$d =$ `c838b852 53ef8dc7 394fa580 8a518398 1c7deef5 a69ba8f4`
`f2117ffe a39cfcd9 0e95f6cb c854abac ab701d50 c1f3cf24`

The public key $Q = dG = (x_Q, y_Q)$:

$x_Q =$ `1fbac8ee bd0cbf35 640b39ef e0808dd7 74debff2 0a2a329e`
`91713baf 7d7f3c3e 81546d88 3730bee7 e48678f8 57b02ca0`

$y_Q =$ `eb213103 bd68ce34 3365a8a4 c3d4555f a385f533 0203bdd7`
`6ffad1f3 affb9575 1c132007 e1b24035 3cb0a4cf 1693bdf9`

D.2.2 Signature Generation

1. Generate (k, k^{-1}), a per-message secret and its inverse modulo n:

   $k =$ `dc6b4403 6989a196 e39d1cda c000812f 4bdd8b2d b41bb33a`
   `f5137258 5ebd1db6 3f0ce827 5aa1fd45 e2d2a735 f8749359`

   $k^{-1} =$ `7436f030 88e65c37 ba8e7b33 887fbc87 757514d6 11f7d1fb`
   `df6d2104 a297ad31 8cdbf740 4e4ba37e 599666df 37b8d8be`

2. Set $R = (x_R, y_R) = kG$ (elliptic curve arithmetic):

   $x_R =$ `a0c27ec8 93092dea 1e1bd2cc fed3cf94 5c8134ed 0c9f8131`
   `1a0f4a05 942db8db ed8dd59f 267471d5 462aa14f e72de856`

   $y_R =$ `85564940 9815bb91 424eaca5 fd76c973 75d575d1 422ec53d`
   `343bd33b 847fdf0c 11569685 b528ab25 49301542 8d7cf72b`

3. Compute $r = x_R \bmod n$:

   $r =$ `a0c27ec8 93092dea 1e1bd2cc fed3cf94 5c8134ed 0c9f8131`
   `1a0f4a05 942db8db ed8dd59f 267471d5 462aa14f e72de856`

4. Message M="This is only a test message. It is 48 bytes long". $H = \mathbf{Hash}(M)$:

   $M =$ `54686973 20697320 6f6e6c79 20612074 65737420 6d657373`
   `6167652e 20497420 69732034 38206279 74657320 6c6f6e67`

   $H =$ `b9210c9d 7e20897a b8659726 6a9d5077 e8db1b06 f7220ed6`
   `ee75bd8b 45db3789 1f8ba555 03040041 59f4453d c5b3f5a1`

5. Conversion of bit string H to integer e (shown in base ten):

   $e =$ `284939761554504754043024822430664637691 8`
   `062062946200867579388439388940182880066 3`
   `731864240088367206094074919580333473`

6. Compute $s = (k^{-1} * (e + d * r)) \bmod n$:

   $s =$ `20ab3f45 b74f10b6 e11f96a2 c8eb694d 206b9dda 86d3c7e3`
   `31c26b22 c987b753 77265776 67adadf1 68ebbe80 3794a402`

The signature on the message M is the pair (r, s).

D.2.3 Signature Verification

The receiver receives a message M', along with a signature (r', s').

1. Validation checking on r', s':

   $r' =$ `a0c27ec8 93092dea 1e1bd2cc fed3cf94 5c8134ed 0c9f8131`
   `1a0f4a05 942db8db ed8dd59f 267471d5 462aa14f e72de856`

   $s' =$ `20ab3f45 b74f10b6 e11f96a2 c8eb694d 206b9dda 86d3c7e3`
   `31c26b22 c987b753 77265776 67adadf1 68ebbe80 3794a402`

   r', s' are in the range $[1, n-1]$.

2. $H' = \mathbf{Hash}(M')$

$H' =$ `b9210c9d 7e20897a b8659726 6a9d5077 e8db1b06 f7220ed6`
`ee75bd8b 45db3789 1f8ba555 03040041 59f4453d c5b3f5a1`

3. Integer e' derived from H' (shown in base ten):

$e' =$ `284939761554504754043024822430664637691 8`
`062062946200867579388439388940182880066 3`
`731864240088367206094074919580333473`

4. Compute $w = (s')^{-1} \bmod n$:

$w =$ `1798845c d0a6cea5 327c501a 71a4baf2 f7be882c fbc30375`
`0a7c861a f8fe8225 467a257f 5bf91a4a aa5a79a8 637d218a`

5. Compute $u_1 = (e' * w) \bmod n$, and compute $u_2 = (r' * w) \bmod n$:

$u_1 =$ `6ce25649 d42d223e 020c1114 0fe77232 6612bb11 b686d35e`
`e98ed455 0e0635d9 dd3a2afb ca0cf2c4 baedcd23 313b189e`

$u_2 =$ `f3b24075 1d5d8ed3 94a4b5bf 8e2a4c0e 1e21aa51 f2620a08`
`b8c55a2b c334c968 99231626 48f06e5f 4659fc52 6d9c1fd6`

6. $R = (x_R, y_R) = u_1G + u_2Q$ (elliptic curve arithmetic):

$x_R =$ `a0c27ec8 93092dea 1e1bd2cc fed3cf94 5c8134ed 0c9f8131`
`1a0f4a05 942db8db ed8dd59f 267471d5 462aa14f e72de856`

$y_R =$ `85564940 9815bb91 424eaca5 fd76c973 75d575d1 422ec53d`
`343bd33b 847fdf0c 11569685 b528ab25 49301542 8d7cf72b`

7. Compute $v = x_R \bmod n$:

$v =$ `a0c27ec8 93092dea 1e1bd2cc fed3cf94 5c8134ed 0c9f8131`
`1a0f4a05 942db8db ed8dd59f 267471d5 462aa14f e72de856`

8. Compare v and r': Successfully Verified.

References

[ANS-X9.62-2005] ANSI X9.62–2005, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)". American National Standards Institute, November 2005.

[FIPS180-3] "Secure Hash Standard (SHS)". Federal Information Standards Processing Publication 180-3, National Institute of Standards and Technology, October 2008.

[FIPS186-3] "Digital Signature Standard (DSS)". Federal Information Standards Processing Publication 186-3, National Institute of Standards and Technology, June 2009.

[SP800-56A] "Recommendation for Pair-Wise Key-establishment Schemes Using Discrete Logarithm Cryptography". NIST Special Publication 800-56A, National Institute of Standards and Technology, March 2007.

[SP800-57] "Recommendation for Key Management – Part 1: General (Revised)". NIST Special Publication 800-57, National Institute of Standards and Technology, March 2007.

[SP800-89] "Recommendation for Assurances for Digital Signature Applications". NIST Special Publication 800-89, National Institute of Standards and Technology, November 2006.

[SP800-90] "Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)". NIST Special Publication 800-90, National Institute of Standards and Technology, March 2007.

[Routines] "Mathematical Routines for NIST Prime Elliptic Curves", dated May 2008, available as a companion document on `<http://www.nsa.gov/ia/programs/suiteb_cryptography>`,