APPENDIX B: Key Pair Generation

Discrete logarithm cryptography (DLC) is divided into finite field cryptography (FFC) and elliptic curve cryptography (ECC); the difference between the two is the type of math that is used. DSA is an example of FFC; ECDSA is an example of ECC. Other examples of DLC are the Diffie-Hellman and MQV key agreement algorithms, which have both FFC and ECC forms.

The most common example of integer factorization cryptography (IFC) is RSA.

This appendix specifies methods for the generation of FFC and ECC key pairs and secret numbers, and the generation of IFC key pairs. All generation methods require the use of a properly instantiated random bit generator (RBG) as discussed in Appendix E; 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. See SP 800-57 [1] for guidance on security strengths and key sizes.

B.1 FFC Key Pair Generation

An FFC key pair x and y is generated for a set of domain parameters (p, q, g [, *domain_parameter_seed*, *counter*]). Two methods are provided for the generation of the FFC private key x and public key y; one of these two methods **shall** be used.

For DSA, the valid values of L and N are provided in Section 4.2.

B.1.1 Key Pair Generation Using Extra Random Bits

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

The following process or its equivalent may be used to generate an FFC key pair.

Input:

1. (p, q, g)	The subset of the domain parameters that are used for this process. p, q and g **shall** either be provided as integers during input, or **shall** be converted to integers prior to use.
2. *additional_input*	Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status*	The status returned from the key pair generation process. The status will indicate **SUCCESS** or an **ERROR**.
2. (x, y)	The generated private and public keys. If an error is encountered during the generation process, invalid values for x and y **should** be returned, as represented by *Invalid_x* and *Invalid_y* in the following specification. x

and y are returned as integers. The generated private key x is in the range [1, q-1].

Process:

1. N = **len** (q); L = **len** (p).
2. If (L, N) is invalid, then return an **ERROR**, *Invalid_x*, and *Invalid_y*.
3. *requested_security_strength* = the security strength associated with the (L, N) pair; see SP 800-57 [1].
4. Obtain a string of N+64 *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_x*, and *Invalid_y*.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix F.2.1).
6. $x = (c \bmod (q\text{-}1)) + 1$.
7. $y = g^x \bmod p$.
8. Return **SUCCESS**, x, and y.

B.1.2 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 x in the correct range. If x is out-of-range, another random number is obtained (i.e., the process is iterated until an acceptable value of x is obtained.

The following process or its equivalent may be used to generate an FFC key pair.

Input:

1. (p, q, g) — The subset of the domain parameters that are used for this process. p, q and g **shall** either be provided as integers during input, or **shall** be converted to integers prior to use.
2. *additional_input* — Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status* — The status returned from the key pair generation process. The status will indicate **SUCCESS** or an **ERROR**.
2. (x, y) — The generated private and public keys. If an error is encountered during the generation process, invalid values for x and y **should** be returned, as represented by *Invalid_x* and *Invalid_y* in the following specification. x and y are returned as integers. The generated private key x is in the range [1, q-1].

Process:

1. N = **len** (q); L = **len** (p).
2. If (L, N) is invalid, then return an **ERROR**, *Invalid_x*, and *Invalid_y*.
3. *requested_security_strength* = the security strength associated with the (L, N) pair; see SP 800-57 [1].
4. Obtain a string of N *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_x*, and *Invalid_y*.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix F.2.1).
6. If ($c > q$-2), then go to step 4.
7. $x = c + 1$.
8. $y = g^x \bmod p$.
9. Return **SUCCESS**, x, and y.

B.2 FFC Per-Message Secret Number Generation

DSA requires the generation of a new 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.

The valid values of N are provided in Section 4.2. Let **inverse** (k, q) be a function that computes the inverse of a (non-negative) integer k with respect to multiplication modulo the prime number q. If k is relatively prime to q, the value returned is a non-negative integer less than q; otherwise, the function returns an error message. A technique for computing the inverse is provided in Appendix F.1.

B.2.1 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 mod function in step 6 is not readily apparent.

The following process or its equivalent may be used to generate a per-message secret number.

Input:

1. (p, q, g)	DSA domain parameters that are generated as specified in Section 4.3.1.
2. *additional_input*	Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status*	The status returned from the secret number generation process. The status will indicate **SUCCESS** or an **ERROR**.

2. (k, k^{-1}) The per-message secret number k and its inverse, k^{-1}. If an error is encountered during the generation process, invalid values for k and k^{-1} **should** be returned, as represented by *Invalid_k* and *Invalid_k_inverse* in the following specification. k and k^{-1} are in the range [1, q-1].

Process:

1. N = **len** (q); L = **len** (p).
2. If (L, N) is invalid, then return an **ERROR**, *Invalid_k*, and *Invalid_k_inverse*.
3. *requested_security_strength* = the security strength associated with the (L, N) pair; see SP 800-57 [1].
4. Obtain a string of N+64 *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_k*, and *Invalid_k_inverse*.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix F.2.1).
6. $k = (c \bmod (q-1)) + 1$.
7. (*status*, *temp*) = **inverse** (k, q).
8. If an ERROR is returned from step 7, go to step 4.
9. $k^{-1} = temp \bmod q$.
10. Return **SUCCESS**, k, and k^{-1}.

B.2.2 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.

The following process or its equivalent may be used to generate a per-message secret number.

Input:

1. (p, q, g) DSA domain parameters that are generated as specified in Section 4.3.1.
2. *additional_input* Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status* The status returned from the secret number generation process. The status will indicate **SUCCESS** or an **ERROR**.

2. (k, k^{-1}) The per-message secret number k and its inverse, k^{-1}. If an error is encountered during the generation process, invalid values for k and k^{-1} **should** be returned, as represented by *Invalid_k* and *Invalid_k*$^{-1}$ in the following specification. k and k^{-1} are in the range [1, q-1].

Process:

1. N = **len** (q); L = **len** (p).
2. If (L, N) is invalid, then return an **ERROR**, *Invalid_k*, and *Invalid_k*$^{-1}$.
3. *requested_security_strength* = the security strength associated with the (L, N) pair; see SP 800-57 [1].
4. Obtain a string of N *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_k*, and *Invalid_k*$^{-1}$.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix F.2.1).
6. If ($c > q$-2), then go to step 4.
7. $k = c + 1$.
8. (*status*, *temp*) = **inverse** (k, q).
9. If an ERROR is returned from step 7, go to step 4.
10. k^{-1} = *temp* mod q.
11. Return **SUCCESS**, k, and k^{-1}.

B.3 IFC Key Pair Generation

RSA keys **shall** meet the following criteria in order to conform to FIPS 186-3:

1. The public exponent e **shall** be selected with the following constraints:
   (a) The public verification exponent e **shall** be selected prior to generating the private prime factors p and q.
   (b) The exponent e **shall** be an odd positive integer such that $65{,}537 \leq e < (2^{nlen-2S} - 1)$, where *nlen* is the length of the modulus n in bits, and S is the desired security strength.

   Note that the value of e may be any value that meets constraint 1b, and that p and q will be selected (in step 2) such that e is relatively prime to both (p-1) and (q-1).
2. Two secret and randomly generated positive primes p and q **shall** be selected with the following constraints:
   (a) (p-1) and (q-1) **shall** be relatively prime to the public exponent e.
   (b) The four numbers ($p \pm 1$) and ($q \pm 1$) **shall** have prime factors (denoted as p_1, p_2, q_1 and q_2) that are greater than 2^{S+20} and less than 2^{S+40}, such that:

Comment [ebb1]: Page: 1
It was suggested that S + 20 and S + 40 be used to accommodate the different security levels.

- $(p-1)$ has a prime factor p_1
- $(p+1)$ has a prime factor p_2
- $(q-1)$ has a prime factor q_1
- $(q+1)$ has a prime factor q_2

where the prime factors, p_1, p_2, q_1, and q_2, are randomly selected from the set of prime numbers between 2^{S+20} and 2^{S+40}.

Comment [ebb2]: Page: 1
See above comment.

(c) The private prime factor p **shall** be selected randomly from the primes that satisfy $(\sqrt{2})(2^{(nlen/2)-1}) \leq p \leq (2^{nlen/2}-1)$, where *nlen* is the length of the modulus n as specified in Table 2 for the desired security strength S.

(d) The private prime factor q **shall** be selected randomly from the primes that satisfy $(\sqrt{2})(2^{(nlen/2)-1}) \leq q \leq (2^{nlen/2}-1)$, where *nlen* is the length of the modulus n as specified in Table 2 for the desired security strength S.

(e) The difference between p and q **shall** be $> 2^{(nlen/2)-100}$.

An approved method for generating p and q with these constraints is provided in Appendix F.5.

3. The private signature exponent d **shall** be selected with the following constraints after the generation of p and q:

   (a) The exponent d **shall** be a positive integer value such that $d > 2^{nlen/2}$, and

   (b) $d = e^{-1} \bmod (\mathrm{LCM}((p-1), (q-1)))$.

   In the extremely rare event that $d \leq 2^{nlen/2}$, then new values for p, q and d **shall** be determined, and a different value of e may be used.

Note that there may be more than one private exponent d (satisfying $d > 2^{nlen/2}$) that corresponds to a public key (n, e). The smallest value of d **shall** be used.

B.4 ECC Key Pair Generation

An ECC key pair d and Q is generated for a set of domain parameters (q, *FR*, a, b, [*SEED*], G, n, h). Two methods are provided for the generation of the ECC private key d and public key Q; one of these two methods **shall** be used to generate d and Q.

For ECDSA, the valid bit-lengths of n are provided in Section 6.1.1. See ANS X9.62 for definitions of the elliptic curve math and the conversion routines.

B.4.1 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 mod function in step 6 is negligible.

The following process or its equivalent may be used to generate an ECC key pair.

Input:

1. (*q, FR, a, b,* [*SEED*], *G, n, h*): The domain parameters that are used for this process. *n* is a prime number, and *G* is a point on the elliptic curve.
2. *additional_input*: Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status* The status returned from the key pair generation procedure. The status will indicate **SUCCESS** or an **ERROR.**
2. (*d, Q*) The generated private and public keys. If an error is encountered during the generation process, invalid values for *d* and *Q* **should** be returned, as represented by *Invalid_d* and *Invalid_Q* in the following specification. *d* is an integer, and *Q* is an elliptic curve point. The generated private key *d* is in the range [1, *n*-1].

Process:

1. N = **len** (n).
2. If N is invalid, then return an **ERROR**, *Invalid_d*, and *Invalid_Q*.
3. *requested_security_strength* = the security strength associated with N; see SP 800-57 [1].
4. Obtain a string of N+64 *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_d*, and *Invalid_Q*.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix F.2.1).
6. $d = (c \bmod (n-1)) + 1$.
7. $Q = dG$.
8. Return **SUCCESS**, *d*, and *Q*.

B.4.2 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, another random number is obtained (i.e., the process is iterated until an acceptable value of *d* is obtained.

The following process or its equivalent may be used to generate an ECC key pair.

Input:

1. (*q, FR, a, b,* [*SEED*], *G, n, h*): The domain parameters that are used for this process. *n* is a prime number, and *G* is a point on the elliptic curve.

2. *additional_input*: Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status* The status returned from the key pair generation procedure. The status will indicate **SUCCESS** or an **ERROR**.
2. (d, Q) The generated private and public keys. If an error is encountered during the generation process, invalid values for d and Q **should** be returned, as represented by *Invalid_d* and *Invalid_Q* in the following specification. d is an integer, and Q is an elliptic curve point. The generated private key d is in the range $[1, n-1]$.

Process:

1. N = **len** (n).
2. If N is invalid, then return an **ERROR**, *Invalid_d*, and *Invalid_Q*.
3. *requested_security_strength* = the security strength associated with N; see SP 800-57 [1].
4. Obtain a string of N *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_d*, and *Invalid_Q*.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix F.2.1).
6. If $(c > n-2)$, then go to step 4.
7. $d = c + 1$.
8. $Q = dG$.
9. Return **SUCCESS**, d, and Q.

B.5 ECC Per-Message Secret Number Generation

ECDSA requires the generation of a new 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.

The valid values of n are provided in Section 6.1.1. See ANS X9.62 for definitions of the elliptic curve math and the conversion routines.

Let **inverse** (k, n) be a function that computes the inverse of a (non-negative) integer k with respect to multiplication modulo the prime number n. If k is relatively prime to n, the value returned is a non-negative integer less than n; otherwise, the function returns an error message. A technique for computing the inverse is provided in Appendix F.1.

B.5.1 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 mod function in step 6 is not readily apparent.

The following process or its equivalent may be used to generate a per-message secret number.

Input:

1. (*q, FR, a, b,* [*SEED*], *G, n, h*): The domain parameters that are used for this process. n is prime number, and G is a point on the elliptic curve.
2. *additional_input*: Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status* — The status returned from the key pair generation procedure. The status will indicate **SUCCESS** or an **ERROR**.
2. (k, k^{-1}) — The generated private and public keys. If an error is encountered during the generation process, invalid values for k and k^{-1} **should** be returned, as represented by *Invalid_k* and *Invalid_k_inverse* in the following specification. k and k^{-1} are integers. k and k^{-1} are integers in the range $[1, n-1]$.

Process:

1. N = **len** (q).
2. If N is invalid, then return an **ERROR**, *Invalid_k*, and *Invalid_k_inverse*.
3. *requested_security_strength* = the security strength associated with N; see SP 800-57 [1].
4. Obtain a string of N+64 *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_k*, and *Invalid_k_inverse*.
5. Convert *returned_bits* to the non-negative integer c (see Appendix F.2.1).
6. $k = (c \bmod (q-1)) + 1$.
7. (*status, temp*) = **inverse** (k, n).
8. If an ERROR is returned from step 7, go to step 4.
9. $k^{-1} = temp \bmod n$.
10. Return **SUCCESS**, k, and k^{-1}.

B.5.2 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.

The following process or its equivalent may be used to generate a per-message secret number.

Input:

1. (*q, FR, a, b,* [*SEED*], *G, n, h*): The domain parameters that are used for this process. n is a prime number, and G is a point on the elliptic curve.
2. *additional_input*: Optional additional input. The maximum length of the *additional_input* **shall** be $\leq 2^{35}$ bits.

Output:

1. *status* — The status returned from the key pair generation procedure. The status will indicate **SUCCESS** or an **ERROR.**
2. (k, k^{-1}) — The generated private and public keys. If an error is encountered during the generation process, invalid values for k and k^{-1} **should** be returned, as represented by *Invalid_k* and *Invalid_k_inverse* in the following specification. k and k^{-1} are integers. k and k^{-1} are integers in the range [1, n-1].

Process:

1. N = **len** (q).
2. If N is invalid, then return an **ERROR**, *Invalid_k*, and *Invalid_k_inverse*.
3. *requested_security_strength* = the security strength associated with N; see SP 800-57 [1].
4. Obtain a string of N *returned_bits* from an **RBG** with a security strength of *requested_security_strength* or more. Provide *additional_input*, if available. If an **ERROR** status is returned, then return an **ERROR**, *Invalid_k*, and *Invalid_k_inverse*.
5. Convert *returned_bits* to the (non-negative) integer c (see Appendix F.2.1).
6. If ($c > n$-2), then go to step 4.
7. $k = c + 1$.
8. (*status*, *temp*) = **inverse** (k, n).
9. If an ERROR is returned from step 7, go to step 4.
10. $k^{-1} = temp \bmod n$.
11. Return **SUCCESS**, k, and k^{-1}.

F.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^{-1} mod a for z with $0 < z < a$, where $0 < z^{-1} < a$, where a is a prime number. In this Standard, z is either k or s, and a is either q or n.

Input:

1. z Comment: The value to be inverted mod a (i.e., either k or s).
2. a Comment: The domain parameter and (prime) modulus (i.e., either q or n).

Output:

1. *status* Comment: The status returned from this function, where the *status* is either SUCCESS or ERROR.
2. z^{-1} Comment: The multiplicative inverse of z mod a, if it exists.

Process:

1. Verify that a and z are positive integers such that $z < a$; if not, return an ERROR.
2. Set $i = a, j = z, x_2 = 1, x_1 = 0, y_2 = 0$, and $y_1 = 1$.
3. *quotient* $= \lfloor i/j \rfloor$.
4. *remainder* $= i - (j \bullet quotient)$.
5. $x = x_2 - (x_1 \bullet quotient)$.
6. $y = y_2 - (y_1 \bullet quotient)$.
7. Set $i = j, j = remainder, x_2 = x_1, x_1 = x, y_2 = y_1$, and $y_1 = y$.
8. If $(j > 0)$, then go to step 3.
9. If $(i \neq 1)$, then return an ERROR.
10. Return SUCCESS and y_2.