FIPS PUB 186-2

FEDERAL INFORMATION
PROCESSING STANDARDS PUBLICATION

2000 January 27

U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology

DIGITAL SIGNATURE STANDARD (DSS)

CATEGORY: COMPUTER SECURITY

U.S. DEPARTMENT OF COMMERCE, William M. Daley, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY,
Raymond G. Kammer, Director

Foreword

The Federal Information Processing Standards Publication Series of the National Institute of Standards and Technology (NIST) is the official series of publications relating to standards and guidelines adopted and promulgated under the provisions of Section 5131 of the Information Technology Management Reform Act of 1996 (Public Law 104-106), and the Computer Security Act of 1987 (Public Law 100-235). These mandates have given the Secretary of Commerce and NIST important responsibilities for improving the utilization and management of computer and related telecommunications systems in the Federal Government. The NIST, through its Information Technology Laboratory, provides leadership, technical guidance, and coordination of Government efforts in the development of standards and guidelines in these areas.

Comments concerning Federal Information Processing Standards Publications are welcomed and should be addressed to the Director, Information Technology Laboratory, National Institute of Standards and Technology, 100 Bureau Dr. Stop 8900, Gaithersburg, MD 20899-8900.

William Mehuron, Director
Information Technology Laboratory

Abstract

This standard specifies a suite of algorithms which can be used to generate a digital signature. Digital signatures are used to detect unauthorized modifications to data and to authenticate the identity of the signatory. In addition, the recipient of signed data can use a digital signature in proving to a third party that the signature was in fact generated by the signatory. This is known as nonrepudiation since the signatory cannot, at a later time, repudiate the signature.

Key words: ADP security, computer security, digital signatures, public-key cryptography, Federal Information Processing Standards.

Federal Information
Processing Standards Publication 186-2

2000 January 27

Announcing the

DIGITAL SIGNATURE STANDARD (DSS)

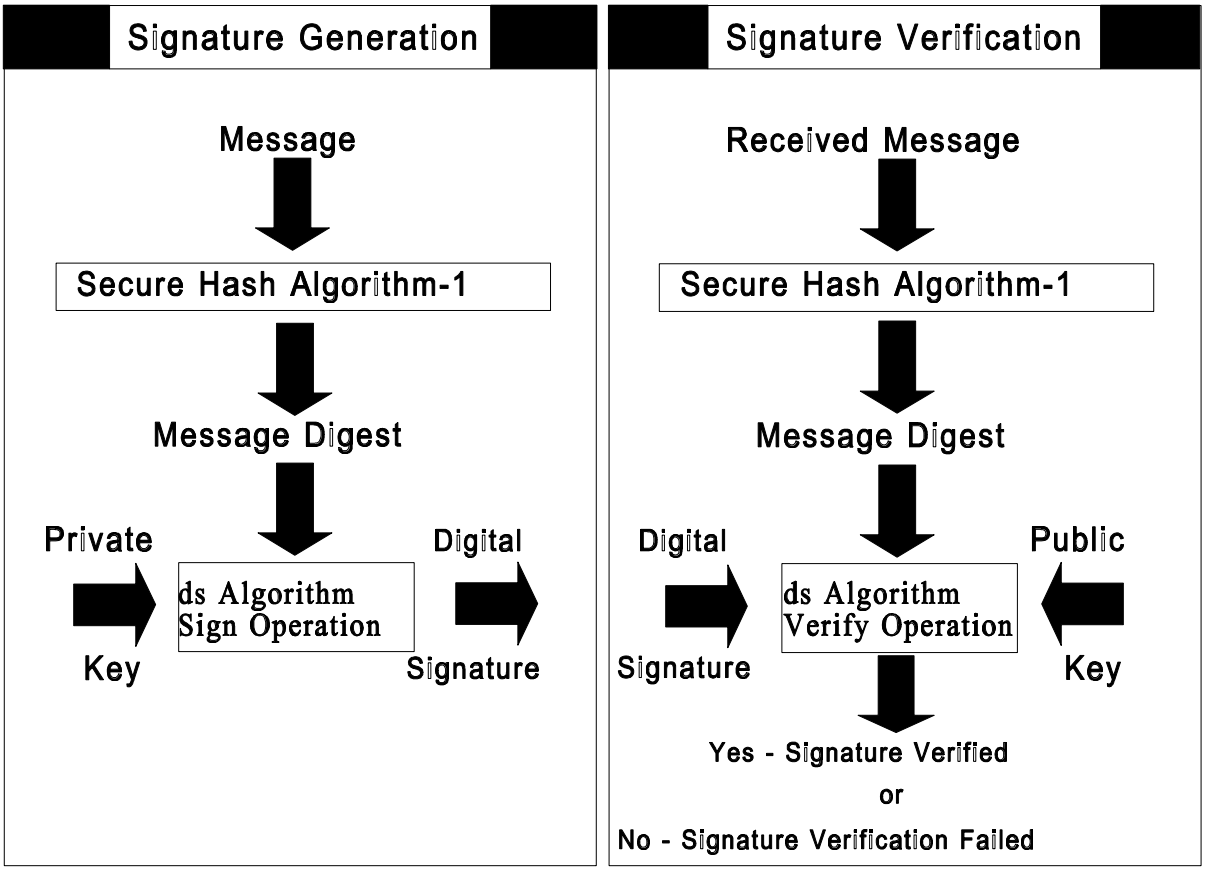
Federal Information Processing Standards Publications (FIPS PUBS) are issued by the National Institute of Standards and Technology (NIST) after approval by the Secretary of Commerce pursuant to Section 5131 of the Information Technology Management Reform Act of 1996 (Public Law 104-106), and the Computer Security Act of 1987 (Public Law 100-235).

Name of Standard: Digital Signature Standard (DSS).

Category of Standard: Computer Security, Cryptography.

Explanation: This Standard specifies algorithms appropriate for applications requiring a digital, rather than written, signature. A digital signature is represented in a computer as a string of binary digits. A digital signature is computed using a set of rules and a set of parameters such that the identity of the signatory and integrity of the data can be verified. An algorithm provides the capability to generate and verify signatures. Signature generation makes use of a private key to generate a digital signature. Signature verification makes use of a public key which corresponds to, but is not the same as, the private key. Each user possesses a private and public key pair. Public keys are assumed to be known to the public in general. Private keys are never shared. Anyone can verify the signature of a user by employing that user's public key. Signature generation can be performed only by the possessor of the user's private key.

A hash function is used in the signature generation process to obtain a condensed version of data, called a message digest (see Figure 1). The message digest is then input to the digital signature (ds) algorithm to generate the digital signature. The digital signature is sent to the intended verifier along with the signed data (often called the message). The verifier of the message and signature verifies the signature by using the sender's public key. The same hash function must also be used in the verification process. The hash function is specified in a separate standard, the Secure Hash Standard (SHS), FIPS 180-1. FIPS approved ds algorithms must be implemented with the SHS. Similar procedures may be used to generate and verify signatures for stored as well as transmitted data.



Approving Authority: Secretary of Commerce.

Maintenance Agency: U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Information Technology Laboratory (ITL).

Applicability: This standard is applicable to all Federal departments and agencies for the protection of sensitive unclassified information that is not subject to section 2315 of Title 10, United States Code, or section 3502(2) of Title 44, United States Code. This standard shall be used in designing and implementing public-key based signature systems that Federal departments and agencies operate or which are operated for them under contract. Adoption and use of this standard is available to private and commercial organizations.

Applications: A digital signature (ds) algorithm authenticates the integrity of the signed data and the identity of the signatory. A ds algorithm may also be used in proving to a third party that data was actually signed by the generator of the signature. A ds algorithm is intended for use in electronic mail, electronic funds transfer, electronic data interchange, software distribution, data storage, and

other applications that require data integrity assurance and data origin authentication. The techniques specified in ANSI X9.31 and ANSI X9.62 may be used in addition to the Digital Signature Algorithm (DSA) specified herein. (NIST editorial note: either DSA, RSA [ANSI X9.31], or ECDSA [ANSI X9.62] may be used; all three do not have to be implemented.)

Implementations: A ds algorithm may be implemented in software firmware, hardware or any combination thereof. NIST has developed a validation program to test implementations for conformance to DSA. Currently, conformance tests for ANSI X9.31 and ANSI X9.62 have not been developed. These tests will be developed and made available in the future. Information about the planned validation program can be obtained from the National Institute of Standards and Technology, Information Technology Laboratory, Attn: DSS Validation, 100 Bureau Drive Stop 8930, Gaithersburg, MD 20899-8930.

Agencies are advised that separate keys should be used for signature and confidentiality purposes when using the X9.31 standard. This is because the RSA algorithm can be used for both data encryption and digital signature purposes.

Export Control: Certain cryptographic devices and technical data regarding them are subject to Federal export controls. Applicable Federal government export controls are specified in Title 15, Code of Federal Regulations (CFR) Part 740.17; Title 15, CFR Part 742; and Title 15, CFR Part 774, Category 5, Part 2.

Patents: The algorithms in this standard may be covered by U.S. or foreign patents.

Implementation Schedule: This standard becomes effective July 27, 2000. A transition period from July 27, 2000 until July 27, 2001 is provided to enable all agencies to develop plans for the acquisition of equipment which implements the digital signature techniques adopted by FIPS 186-2. During the transition period, agencies may continue to use their existing digital signature systems and to acquire additional equipment that may be needed to interoperate with these legacy digital signature systems. Agencies without legacy digital signature systems should plan for the acquisition and use of equipment implementing the digital signature techniques that are adopted by FIPS 186-2. After the transition period, only equipment that implements FIPS 186-2 endorsed techniques should be acquired.

Specifications: Federal Information Processing Standard (FIPS) 186-2 Digital Signature Standard (affixed).

Cross Index:

a. FIPS PUB 46-3, Data Encryption Standard.

b. FIPS PUB 73, Guidelines for Security of Computer Applications.

c. FIPS PUB 140-1, Security Requirements for Cryptographic Modules.

d. FIPS PUB 171, Key Management Using ANSI X9.17.

e. FIPS PUB 180-1, Secure Hash Standard.

f. ANSI X9.31-1998, Digital Signatures Using Reversible Public Key Cryptography for the Financial Services Industry (rDSA).

g. ANSI X9.62-1998, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

Qualifications: The security of a digital signature system is dependent on maintaining the secrecy of users' private keys. Users must therefore guard against the unauthorized acquisition of their private keys. While it is the intent of this standard to specify general security requirements for generating digital signatures, conformance to this standard does not assure that a particular implementation is secure. The responsible authority in each agency or department shall assure that an overall implementation provides an acceptable level of security. This standard will be reviewed every five years in order to assess its adequacy.

Waiver Procedure: Under certain exceptional circumstances, the heads of Federal agencies, or their delegates, may approve waivers to Federal Information Processing Standards (FIPS). The head of such agency may redelegate such authority only to a senior official designated pursuant to section 3506(b) of Title 44, United States Code. Waiver shall be granted only when:

a. Compliance with a standard would adversely affect the accomplishment of the mission of an operator of a Federal computer system; or

b. Cause a major adverse financial impact on the operator which is not offset by Government wide savings.

Agency heads may act upon a written waiver request containing the information detailed above. Agency heads may also act without a written waiver request when they determine that conditions for meeting the standard cannot be met. Agency heads may approve waivers only by a written decision which explains the basis on which the agency head made the required finding(s). A copy of each such decision, with procurement sensitive or classified portions clearly identified, shall be sent to: National Institute of Standards and Technology; ATTN: FIPS Waiver Decisions, 100 Bureau Drive Stop 8970, Gaithersburg, MD 20899-8970.

In addition, notice of each waiver granted and each delegation of authority to approve waivers shall be sent promptly to the Committee on Government Operations of the House of Representatives and the Committee on Governmental Affairs of the Senate and shall be published promptly in the Federal Register.

When the determination on a waiver applies to the procurement of equipment and/or services, a notice of the waiver determination must be published in the Commerce Business Daily as a part of the notice of solicitation for offers of an acquisition or, if the waiver determination is made after that notice is published, by amendment to such notice.

A copy of the waiver, any supporting documents, the document approving the waiver and any supporting and accompanying documents, with such deletions as the agency is authorized and decides to make under 5 U.S.C. Sec. 552(b), shall be part of the procurement documentation and retained by the agency.

Where to Obtain Copies of the Standard: Copies of this publication are for sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. When ordering, refer to Federal Information Processing Standards Publication 186-2 (FIPSPUB186-2), and identify the title. When microfiche is desired, this should be specified. Prices are published by NTIS in current catalogs and other issuances. Payment may be made by check, money order, deposit account or charged to a credit card accepted by NTIS.

Federal Information
Processing Standards Publication 186-2

2000 January 27

Specifications for the

DIGITAL SIGNATURE STANDARD (DSS)

1. INTRODUCTION

This publication prescribes three algorithms suitable for digital signature (ds) generation and verification. The first algorithm, the Digital Signature Algorithm (DSA), is described in sections 4 - 6 and appendices 1 - 5. The second algorithm, the RSA ds algorithm, is discussed in section 7 and the third algorithm, the ECDSA algorithm, is discussed in section 8 and recommended elliptic curves in appendix 6.

2. GENERAL

When a message is received, the recipient may desire to verify that the message has not been altered in transit. Furthermore, the recipient may wish to be certain of the originator's identity. Both of these services can be provided by a ds algorithm. A digital signature is an electronic analogue of a written signature in that the digital signature can be used in proving to the recipient or a third party that the message was, in fact, signed by the originator. Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any later time.

This publication prescribes two algorithms suitable for digital signature generation and verification.

3. USE OF A DIGITAL SIGNATURE (ds) ALGORITHM

A ds algorithm is used by a *signatory* to generate a digital signature on data and by a *verifier* to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process and the public key is used in the signature verification process. For both signature generation and verification, the data which is referred to as a message,

M, is reduced by means of the Secure Hash Algorithm (SHA-1) specified in FIPS 180-1. An adversary, who does not know the private key of the signatory, cannot generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory's public key, anyone can verify a correctly signed message. A means of associating public and private key pairs to the corresponding users is required. That is, there must be a binding of a user's identity and the user's public key. This binding may be certified by a mutually trusted party. For example, a certifying authority could sign credentials containing a user's public key and identity to form a certificate. Systems for certifying credentials and distributing certificates are beyond the scope of this standard. NIST intends to publish separate document(s) on certifying credentials and distributing certificates.

4. DSA PARAMETERS

The DSA makes use of the following parameters:

1. p = a prime modulus, where $2^{L-1} < p < 2^{L}$ for $512 \leq L \leq 1024$ and L a multiple of 64

2. q = a prime divisor of p - 1, where $2^{159} < q < 2^{160}$

3. $g = h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < p - 1$ such that $h^{(p-1)/q} \bmod p > 1$
   (g has order q mod p)

4. x = a randomly or pseudorandomly generated integer with $0 < x < q$

5. $y = g^{x} \bmod p$

6. k = a randomly or pseudorandomly generated integer with $0 < k < q$

The integers p, q, and g can be public and can be common to a group of users. A user's private and public keys are x and y, respectively. They are normally fixed for a period of time. Parameters x and k are used for signature generation only, and must be kept secret. Parameter k must be regenerated for each signature.

Parameters p and q shall be generated as specified in Appendix 2, or using other FIPS approved security methods. Parameters x and k shall be generated as specified in Appendix 3, or using other FIPS approved security methods.

5. DSA SIGNATURE GENERATION

The signature of a message M is the pair of numbers r and s computed according to the equations below:

$r = (g^k \bmod p) \bmod q$ and
$s = (k^{-1}(\text{SHA-1}(M) + xr)) \bmod q$.

In the above, k^{-1} is the multiplicative inverse of k, mod q; i.e., $(k^{-1} k) \bmod q = 1$ and $0 < k^{-1} < q$. The value of SHA-1(M) is a 160-bit string output by the Secure Hash Algorithm specified in FIPS 180-1. For use in computing s, this string must be converted to an integer. The conversion rule is given in Appendix 2.2.

As an option, one may wish to check if r = 0 or s = 0. If either r = 0 or s = 0, a new value of k should be generated and the signature should be recalculated (it is extremely unlikely that r = 0 or s = 0 if signatures are generated properly).

The signature is transmitted along with the message to the verifier.

6. DSA SIGNATURE VERIFICATION

Prior to verifying the signature in a signed message, p, q and g plus the sender's public key and identity are made available to the verifier in an authenticated manner.

Let M′, r′, and s′ be the received versions of M, r, and s, respectively, and let y be the public key of the signatory. To verify the signature, the verifier first checks to see that $0 < r' < q$ and $0 < s' < q$; if either condition is violated the signature shall be rejected. If these two conditions are satisfied, the verifier computes

$w = (s')^{-1} \bmod q$

$u1 = ((\text{SHA-1}(M'))w) \bmod q$

$u2 = ((r')w) \bmod q$

$v = (((g)^{u1} (y)^{u2}) \bmod p) \bmod q$.

If v = r′, then the signature is verified and the verifier can have high confidence that the received message was sent by the party holding the secret key x corresponding to y. For a proof that v = r′ when M′ = M, r′ = r, and s′ = s, see Appendix 1.

If v does not equal r′, then the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message should be considered invalid.

7. RSA DIGITAL SIGNATURE ALGORITHM

The RSA ds algorithm is a FIPS approved cryptographic algorithm for digital signature generation and verification. This is described in ANSI X9.31.

8. ELLIPTIC CURVE DIGITAL SIGNATURE ALGORITHM (ECDSA)

The ECDSA ds algorithm is a FIPS approved cryptographic algorithm for digital signature generation and verification. ECDSA is the elliptic curve analogue of the DSA. ECDSA is described in ANSI X9.62. The recommended elliptic curves for Federal Government use are included in Appendix 6.

APPENDIX 1. A PROOF THAT v = r′ IN THE DSA

This appendix is for informational purposes only and is not required to meet the standard.

The purpose of this appendix is to show that in the DSA, if M′ = M, r′ = r and s′ = s in the signature verification then v = r′. We need the following easy result.

<u>LEMMA</u>. Let p and q be primes so that q divides p - 1, h a positive integer less than p, and $g = h^{(p-1)/q} \bmod p$. Then $g^q \bmod p = 1$, and if m mod q = n mod q, then $g^m \bmod p = g^n \bmod p$.

Proof: We have

$$g^q \bmod p = (h^{(p-1)/q} \bmod p)^q \bmod p$$

$$= h^{(p-1)} \bmod p$$

$$= 1$$

by Fermat's Little Theorem. Now let m mod q = n mod q, i.e., m = n + kq for some integer k. Then

$$g^m \bmod p = g^{n+kq} \bmod p$$

$$= (g^n g^{kq}) \bmod p$$

$$= ((g^n \bmod p)(g^q \bmod p)^k) \bmod p$$

$$= g^n \bmod p$$

since $g^q \bmod p = 1$. ■

We are now ready to prove the main result.

<u>THEOREM</u>. If M′ = M, r′ = r, and s′ = s in the signature verification, then v = r′.

Proof: We have

$$w = (s')^{-1} \bmod q = s^{-1} \bmod q$$

$$u1 = ((\text{SHA-1}(M'))w) \bmod q = ((\text{SHA-1}(M))w) \bmod q$$

$$u2 = ((r')w) \bmod q = (rw) \bmod q.$$

Now $y = g^x \bmod p$, so that by the lemma,

$$v = ((g^{u1}\ y^{u2}) \bmod p) \bmod q$$

$$= ((gSHA\text{-}1^{(M)w}\ y^{rw}) \bmod p) \bmod q$$

$$= ((gSHA\text{-}1^{(M)w}\ g^{xrw}) \bmod p) \bmod q$$

$$= ((g^{(}SHA\text{-}1^{(M)+xr)w}) \bmod p) \bmod q.$$

Also

$$s = (k^{-1}(SHA\text{-}1(M) + xr)) \bmod q.$$

Hence

$$w = (k(SHA\text{-}1(M) + xr)^{-1}) \bmod q$$

$$(SHA\text{-}1(M) + xr)w \bmod q = k \bmod q.$$

Thus by the lemma,

$$v = (g^k \bmod p) \bmod q$$

$$= r$$

$$= r'. \blacksquare$$

APPENDIX 2. GENERATION OF PRIMES FOR THE DSA

This appendix includes algorithms for generating the primes p and q used in the DSA. These algorithms require a random number generator (see Appendix 3), and an efficient modular exponentiation algorithm. Generation of p and q shall be performed as specified in this appendix, or using other FIPS approved security methods.

2.1. A PROBABILISTIC PRIMALITY TEST

In order to generate the primes p and q, a primality test is required.

There are several fast probabilistic algorithms available. The following algorithm is a simplified version of a procedure due to M.O. Rabin, based in part on ideas of Gary L. Miller. [See Knuth, The Art of Computer Programming, Vol. 2, Addison-Wesley, 1981, Algorithm P, page 379.] If this algorithm is iterated n times, it will produce a false prime with probability no greater than $1/4^n$. Therefore, $n \geq 50$ will give an acceptable probability of error. To test whether an integer is prime:

Step 1. Set $i = 1$ and $n \geq 50$.

Step 2. Set w = the integer to be tested, $w = 1 + 2^a m$, where m is odd and 2^a is the largest power of 2 dividing $w - 1$.

Step 3. Generate a random integer b in the range $1 < b < w$.

Step 4. Set $j = 0$ and $z = b^m \bmod w$.

Step 5. If $j = 0$ and $z = 1$, or if $z = w - 1$, go to step 9.

Step 6. If $j > 0$ and $z = 1$, go to step 8.

Step 7. $j = j + 1$. If $j < a$, set $z = z^2 \bmod w$ and go to step 5.

Step 8. w is not prime. Stop.

Step 9. If $i < n$, set $i = i + 1$ and go to step 3. Otherwise, w is probably prime.

2.2. GENERATION OF PRIMES

The DSA requires two primes, p and q, satisfying the following three conditions:

a. $2^{159} < q < 2^{160}$

b. $2^{L-1} < p < 2^L$ for a specified L, where $L = 512 + 64j$ for some $0 \leq j \leq 8$

c. q divides p - 1.

This prime generation scheme starts by using the SHA-1 and a user supplied SEED to construct a prime, q, in the range $2^{159} < q < 2^{160}$. Once this is accomplished, the same SEED value is used to construct an X in the range $2^{L-1} < X < 2^{L}$. The prime, p, is then formed by rounding X to a number congruent to 1 mod 2q as described below.

An integer x in the range $0 \leq x < 2^{g}$ may be converted to a g-long sequence of bits by using its binary expansion as shown below:

$$x = x_1*2^{g-1} + x_2*2^{g-2} + ... + x_{g-1}*2 + x_g \text{ -> } \{ x_1,...,x_g \}.$$

Conversely, a g-long sequence of bits $\{ x_1,...,x_g \}$ is converted to an integer by the rule

$$\{ x_1,...,x_g \} \text{ -> } x_1*2^{g-1} + x_2*2^{g-2} + ... + x_{g-1}*2 + x_g.$$

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

Let $L - 1 = n*160 + b$, where both b and n are integers and $0 \leq b < 160$.

Step 1. Choose an arbitrary sequence of at least 160 bits and call it SEED. Let g be the length of SEED in bits.

Step 2. Compute

$$U = \text{SHA-1}[\text{SEED}] \text{ XOR SHA-1}[(\text{SEED}+1) \bmod 2^{g}].$$

Step 3. Form q from U by setting the most significant bit (the 2^{159} bit) and the least significant bit to 1. In terms of boolean operations, q = U OR 2^{159} OR 1. Note that $2^{159} < q < 2^{160}$.

Step 4. Use a robust primality testing algorithm to test whether q is prime[1].

Step 5. If q is not prime, go to step 1.

Step 6. Let counter = 0 and offset = 2.

Step 7. For k = 0,...,n let

$$V_k = \text{SHA-1}[(\text{SEED} + \text{offset} + k) \bmod 2^{g}].$$

[1]A robust primality test is one where the probability of a non-prime number passing the test is at most 2^{-80}.

Step 8. Let W be the integer

$$W = V_0 + V_1*2^{160} + ... + V_{n-1}*2^{(n-1)*160} + (V_n \bmod 2^b) * 2^{n*160}$$

and let $X = W + 2^{L-1}$. Note that $0 \leq W < 2^{L-1}$ and hence $2^{L-1} \leq X < 2^L$.

Step 9. Let c = X mod 2q and set p = X - (c - 1). Note that p is congruent to 1 mod 2q.

Step 10. If $p < 2^{L-1}$, then go to step 13.

Step 11. Perform a robust primality test on p.

Step 12. If p passes the test performed in step 11, go to step 15.

Step 13. Let counter = counter + 1 and offset = offset + n + 1.

Step 14. If counter $\geq 2^{12} = 4096$ go to step 1, otherwise (i.e. if counter < 4096) go to step 7.

Step 15. Save the value of SEED and the value of counter for use in certifying the proper generation of p and q.

APPENDIX 3. RANDOM NUMBER GENERATION FOR THE DSA

Any implementation of the DSA requires the ability to generate random or pseudorandom integers. Such numbers are used to derive a user's private key, x, and a user's per message secret number, k. These randomly or pseudorandomly generated integers are selected to be between 0 and the 160-bit prime q (as specified in the standard). They shall be generated by the techniques given in this appendix, or using other FIPS approved security methods.

One FIPS approved pseudorandom integer generator is supplied in Appendix C of ANSI X9.17, "Financial Institution Key Management (Wholesale)."

Other pseudorandom integer generators are given in this appendix. These permit generation of pseudorandom values of x and k for use in the DSA. The algorithm in section 3.1 may be used to generate values for x. An algorithm for k and r is given in section 3.2. The latter algorithm allows most of the signature computation to be precomputed without knowledge of the message to be signed.

The algorithms employ a one-way function G(t,c), where t is 160 bits, c is b bits ($160 \leq b \leq 512$) and G(t,c) is 160 bits. One way to construct G is via the Secure Hash Algorithm (SHA-1), as defined in the Secure Hash Standard (SHS). The 160-bit message digest output of the SHA-1 algorithm when message M is input is denoted by SHA-1(M). A second method for constructing G is to use the Data Encryption Standard (DES). The construction of G by these techniques is discussed in sections 3.3 and 3.4 of this appendix.

In the algorithms in sections 3.1 and 3.2, a secret b-bit seed-key is used. The algorithm in section 3.1 optionally allows the use of a user provided input. If G is constructed via the SHA-1 as defined in section 3.3, then b is between 160 and 512. If DES is used to construct G as defined in section 3.4, then b is equal to 160.

3.1. ALGORITHM FOR COMPUTING m VALUES OF x

Let x be the signer's private key. The following may be used to generate m values of x:

Step 1. Choose a new, secret value for the seed-key, XKEY.

Step 2. In hexadecimal notation let

t = 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0.

This is the initial value for $H_0 \| H_1 \| H_2 \| H_3 \| H_4$ in the SHS.

Step 3. For j = 0 to m - 1 do

a. $XSEED_j$ = optional user input.
b. XVAL = (XKEY + $XSEED_j$) mod 2^b.

c. x_j = G(t,XVAL) mod q.

d. XKEY = (1 + XKEY + x_j) mod 2^b.

3.2. ALGORITHM FOR PRECOMPUTING ONE OR MORE k AND r VALUES

This algorithm can be used to precompute k, k^{-1}, and r for m messages at a time. Note that implementation of the DSA with precomputation may be covered by U.S. and foreign patents.

Algorithm:

Step 1. Choose a secret initial value for the seed-key, KKEY.

Step 2. In hexadecimal notation let

t = EFCDAB89 98BADCFE 10325476 C3D2E1F0 67452301.

This is a cyclic shift of the initial value for $H_0 \| H_1 \| H_2 \| H_3 \| H_4$ in the SHS.

Step 3. For j = 0 to m - 1 do

a. k = G(t,KKEY) mod q.

b. Compute $k_j^{-1} = k^{-1}$ mod q.

c. Compute $r_j = (g^k \bmod p) \bmod q$.

d. KKEY = (1 + KKEY + k) mod 2^b.

Step 4. Suppose M_0 , ... , M_{m-1} are the next m messages. For j = 0 to m - 1 do

a. Let h = SHA-1(M_j).

b. Let $s_j = (k_j^{-1}(h + xr_j)) \bmod q$.

c. The signature for M_j is (r_j,s_j).

Step 5. Let t = h.

Step 6. Go to step 3.

Step 3 permits precomputation of the quantities needed to sign the next m messages. Step 4 can begin whenever the first of these m messages is ready. The execution of step 4 can be suspended whenever the next of the m messages is not ready. As soon as steps 4 and 5 have completed, step 3 can be executed, and the results saved until the first member of the next group of m messages is ready.

In addition to space for KKEY, two arrays of length m are needed to store r_0 , ... r_{m-1} and k_0^{-1}, ... , k_{m-1}^{-1} when they are computed in step 3. Storage for s_0 , ... , s_{m-1} is only needed if the signatures for a group of messages are stored; otherwise s_j in step 4 can be replaced by s and a single space allocated.

3.3. CONSTRUCTING THE FUNCTION G FROM THE SHA-1

G(t,c) may be constructed using steps (a) - (e) in section 7 of the Specifications for the Secure Hash Standard. Before executing these steps, $\{H_j\}$ and M_1 must be initialized as follows:

i. Initialize the $\{H_j\}$ by dividing the 160 bit value t into five 32-bit segments as follows:

$$t = t_0 \parallel t_1 \parallel t_2 \parallel t_3 \parallel t_4$$

Then $H_j = t_j$ for j = 0 through 4.

ii. There will be only one message block, M_1, which is initialized as follows:

$$M_1 = c \parallel 0^{512-b}$$

(The first b bits of M_1 contain c, and the remaining (512-b) bits are set to zero).

Then steps (a) through (e) of section 7 are executed, and G(t,c) is the 160 bit string represented by the five words:

$$H_0 \parallel H_1 \parallel H_2 \parallel H_3 \parallel H_4$$

at the end of step (e).

3.4. CONSTRUCTING THE FUNCTION G FROM THE DES

Let a XOR b denote the bitwise exclusive-or of bit strings a and b. Suppose a1, a2, b1, b2 are 32-bit strings. Let b1' be the 24 least significant bits of b1. Let K = b1' || b2 and A = a1 || a2. Define

$$DES_{b1,b2}(a1,a2) = DES_K(A)$$

In the above, $DES_K(A)$ represents ordinary DES encryption of the 64-bit block A using the 56-bit

key K. Now suppose t and c are each 160 bits. To compute G(t,c):

Step 1. Write

$$t = t_1 \| t_2 \| t_3 \| t_4 \| t_5$$

$$c = c_1 \| c_2 \| c_3 \| c_4 \| c_5$$

In the above, each t_i and c_i is 32 bits.

Step 2. For i = 1 to 5 do

$$x_i = t_i \text{ XOR } c_i$$

Step 3. For i = 1 to 5 do

$$b1 = c_{((i+3) \bmod 5) + 1}$$

$$b2 = c_{((i+2) \bmod 5) + 1}$$

$$a1 = x_i$$

$$a2 = x_{(i \bmod 5) + 1} \text{ XOR } x_{((i+3) \bmod 5) + 1}$$

$$y_{i,1} \| y_{i,2} = DES_{b1,b2}(a1,a2) \quad (y_{i,1}, y_{i,2} = 32 \text{ bits})$$

Step 4. For i = 1 to 5 do

$$z_i = y_{i,1} \text{ XOR } y_{((i+1) \bmod 5)+1,2} \text{ XOR } y_{((i+2) \bmod 5)+1,1}$$

Step 5. Let

$$G(t,c) = z_1 \| z_2 \| z_3 \| z_4 \| z_5$$

APPENDIX 4. GENERATION OF OTHER QUANTITIES FOR THE DSA

This appendix is for informational purposes only and is not required to meet the standard.

The algorithms given in this appendix may be used to generate the quantities g, k^{-1}, and s^{-1} used in the DSA.

To generate g:

Step 1. Generate p and q as specified in Appendix 2.

Step 2. Let e = (p - 1)/q.

Step 3. Set h = any integer, where 1 < h < p - 1 and h differs from any value previously tried.

Step 4. Set $g = h^e \bmod p$.

Step 5. If g = 1, go to step 3.

To compute the multiplicative inverse $n^{-1} \bmod q$ for n with 0 < n < q, where $0 < n^{-1} < q$:

Step 1. Set i = q, h = n, v = 0, and d = 1.

Step 2. Let t = i DIV h, where DIV is defined as integer division.

Step 3. Set x = h.

Step 4. Set h = i - tx.

Step 5. Set i = x.

Step 6. Set x = d.

Step 7. Set d = v - tx.

Step 8. Set v = x.

Step 9. If h > 0, go to step 2.

Step 10. Let $n^{-1} = v \bmod q$.

Note that in step 10, v may be negative. The v mod q operation should yield a value between 1 and q - 1 inclusive.

APPENDIX 5. EXAMPLE OF THE DSA

This appendix is for informational purposes only and is not required to meet the standard.
Let L = 512 (size of p). The values in this example are expressed in hexadecimal notation. The p and q given here were generated by the prime generation standard described in appendix 2 using the 160-bit SEED:

```
d5014e4b 60ef2ba8 b6211b40 62ba3224 e0427dd3
```

With this SEED, the algorithm found p and q when the counter was at 105. x was generated by the algorithm described in appendix 3, section 3.1, using the SHA-1 to construct G (as in appendix 3, section 3.3) and a 160-bit XKEY:

XKEY =

```
bd029bbe 7f51960b cf9edb2b 61f06f0f eb5a38b6
```

t =

```
67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0
```

x = G(t,XKEY) mod q

k was generated by the algorithm described in appendix 3, section 3.2, using the SHA-1 to construct G (as in appendix 3, section 3.3) and a 160-bit KKEY:

KKEY =

```
687a66d9 0648f993 867e121f 4ddf9ddb 01205584
```

t =

```
EFCDAB89 98BADCFE 10325476 C3D2E1F0 67452301
```

k = G(t,KKEY) mod q

Finally:

h = 2

p =

```
8df2a494 492276aa 3d25759b b06869cb eac0d83a fb8d0cf7
cbb8324f 0d7882e5 d0762fc5 b7210eaf c2e9adac 32ab7aac
```

```
49693dfb f83724c2 ec0736ee 31c80291
```

q =

```
c773218c 737ec8ee 993b4f2d ed30f48e dace915f
```

g =

```
626d0278 39ea0a13 413163a5 5b4cb500 299d5522 956cefcb
3bff10f3 99ce2c2e 71cb9de5 fa24babf 58e5b795 21925c9c
c42e9f6f 464b088c c572af53 e6d78802
```

x =

```
2070b322 3dba372f de1c0ffc 7b2e3b49 8b260614
```

k =

```
358dad57 1462710f 50e254cf 1a376b2b deaadfbf
```

k^{-1} =

```
0d516729 8202e49b 4116ac10 4fc3f415 ae52f917
```

M = ASCII form of "abc" (See FIPS PUB 180-1, Appendix A)

(SHA-1)(M) =

```
a9993e36 4706816a ba3e2571 7850c26c 9cd0d89d
```

y =

```
19131871 d75b1612 a819f29d 78d1b0d7 346f7aa7 7bb62a85
9bfd6c56 75da9d21 2d3a36ef 1672ef66 0b8c7c25 5cc0ec74
858fba33 f44c0669 9630a76b 030ee333
```

r =

```
8bac1ab6 6410435c b7181f95 b16ab97c 92b341c0
```

s =

```
41e2345f 1f56df24 58f426d1 55b4ba2d b6dcd8c8
```

w =

```
9df4ece5 826be95f ed406d41 b43edc0b 1c18841b
```

u1 =

```
bf655bd0 46f0b35e c791b004 804afcbb 8ef7d69d
```

u2 =

```
821a9263 12e97ade abcc8d08 2b527897 8a2df4b0
```

g^{u1} mod p =

```
51b1bf86 7888e5f3 af6fb476 9dd016bc fe667a65 aafc2753
9063bd3d 2b138b4c e02cc0c0 2ec62bb6 7306c63e 4db95bbf
6f96662a 1987a21b e4ec1071 010b6069
```

y^{u2} mod p =

```
8b510071 2957e950 50d6b8fd 376a668e 4b0d633c 1e46e665
5c611a72 e2b28483 be52c74d 4b30de61 a668966e dc307a67
c19441f4 22bf3c34 08aeba1f 0a4dbec7
```

v =

```
8bac1ab6 6410435c b7181f95 b16ab97c 92b341c0
```

APPENDIX 6. RECOMMENDED ELLIPTIC CURVES FOR FEDERAL GOVERNMENT USE
July 1999

This collection of elliptic curves is recommended for Federal government use and contains choices of private key length and underlying fields.

1. Parameter Choices

1.1 Choice of Key Lengths

The principal parameters for elliptic curve cryptography are the elliptic curve E and a designated point G on E called the *base point*. The base point has order r, a large prime. The number of points on the curve is $n = fr$ for some integer f (the *cofactor*) not divisible by r. For efficiency reasons, it is desirable to take the cofactor to be as small as possible.

All of the curves given below have cofactors 1, 2, or 4. As a result, the private and public keys are approximately the same length. Each length is chosen to correspond to the cryptovariable length of a common symmetric cryptologic. In each case, the private key length is, at least, approximately twice the symmetric cryptovariable length.

1.2 Choice of Underlying Fields

For each cryptovariable length, there are given two kinds of fields.

- A *prime field* is the field $GF(p)$ which contains a prime number p of elements. The elements of this field are the integers modulo p, and the

field arithmetic is implemented in terms of the arithmetic of integers modulo p.

- A *binary field* is the field $GF(2^m)$ which contains 2^m elements for some m (called the *degree* of the field). The elements of this field are the bit strings of length m, and the field arithmetic is implemented in terms of operations on the bits.

The following table gives the sizes of the various underlying fields. By $||p||$ is meant the length of the binary expansion of the integer p.

Symmetric CV Length	Example Algorithm	Prime Field	Binary Field
80	SKIPJACK	$\|\|p\|\| = 192$	$m = 163$
112	Triple-DES	$\|\|p\|\| = 224$	$m = 233$
128	AES Small	$\|\|p\|\| = 256$	$m = 283$
192	AES Medium	$\|\|p\|\| = 384$	$m = 409$
256	AES Large	$\|\|p\|\| = 521$	$m = 571$

1.3 Choice of Basis

To describe the arithmetic of a binary field, it is first necessary to specify how a bit string is to be interpreted. This is referred to as choosing a *basis* for the field. There are two common types of bases: a *polynomial basis* and a *normal basis*.

- A polynomial basis is specified by an irreducible polynomial modulo 2, called the *field polynomial*. The bit string ($a_{m-1} \ldots a_2\ a_1\ a_0$) is taken to represent the polynomial

$$a_{m-1}\, t^{m-1} + \ldots + a_2\, t^2 + a_1\, t + a_0$$

over $GF(2)$. The field arithmetic is implemented as polynomial arithmetic modulo $p(t)$, where $p(t)$ is the field polynomial.

- A normal basis is specified by an element q of a particular kind. The bit string ($a_0\ \ a_1\ \ a_2\ \ldots a_{m-1}$) is taken to represent the element

$$a_0 q + a_1 q^{2} + a_2 q^{2^2} + a_{m-1} q^{2^{m-1}}.$$

Normal basis field arithmetic is not easy to describe or efficient to implement in general, but is for a special class called *Type T low-complexity* normal bases. For a given field degree m, the choice of T specifies the basis and the field arithmetic (see Appendix 6.2).

There are many polynomial bases and normal bases from which to choose. The following procedures are commonly used to select a basis representation.

- *Polynomial Basis*: If an irreducible *trinomial* $t^m + t^k + 1$ exists over $GF(2)$, then the field polynomial $p(t)$ is chosen to be the irreducible trinomial with the lowest-degree middle term t^k. If no irreducible trinomial exists, then one selects instead a *pentanomial* $t^m + t^a + t^b + t^c + 1$. The particular pentanomial chosen has the following properties: the second term t^a has the lowest degree m; the third term t^b has the lowest degree among all irreducible pentanomials of degree m and second term t^a; and the fourth term t^c has the lowest degree among all irreducible pentanomials of degree m, second term t^a, and third term t^b.

- *Normal Basis*: Choose the Type T low-complexity normal basis with the smallest T.

For each binary field, the parameters are given for the above basis representations.

1.4 Choice of Curves

Two kinds of curves are given:

- *Pseudo-random* curves are those whose coefficients are generated from the output of a seeded cryptographic hash. If the seed value is given along with the coefficients, it can be verified easily that the coefficients were indeed generated by that method.
- *Special curves* whose coefficients and underlying field have been selected to optimize the efficiency of the elliptic curve operations.

For each size, the following curves are given:

→ A pseudo-random curve over $GF(p)$.

→ A pseudo-random curve over $GF(2^m)$.

→ A special curve over $GF(2^m)$ called a *Koblitz curve* or *anomalous binary curve*.

The pseudo-random curves are generated via the SHA-1 based method given in the ANSI X9.62 and IEEE P1363 standards. (The generation and verification processes are given in Appendices 6-4 through 6-7.)

1.5 Choice of Base Points

Any point of order r can serve as the base point. Each curve is supplied with a sample base point $G = (G_x , G_y)$. Users may want to generate their own base points to ensure cryptographic separation of networks.

2. Curves over Prime Fields

For each prime p, a pseudo-random curve

$$E : \ y^2 \equiv x^3 - 3x + b \ (mod\ p)$$

of prime order r is listed1. (Thus, for these curves, the cofactor is always $f = 1$.) The following parameters are given:

- The prime modulus p
- The order r
- the 160-bit input seed s to SHA-1 based algorithm
- The output c of the SHA-1 based algorithm
- The coefficient b (satisfying $b^2 c \equiv -27$ ($mod\ p$))
- The base point x coordinate G_x
- The base point y coordinate G_y

The integers p and r are given in decimal form; bit strings and field elements are given in hex.

1 The selection $a \equiv -3$ for the coefficient of x was made for reasons of efficiency; see IEEE P1363.

Curve P-192

$p =$ 6277101735386680763835789423207666416083908700390324961279

$r =$ 6277101735386680763835789423176059013767194773182842284081

$s =$ 3045ae6f c8422f64 ed579528 d38120ea e12196d5

$c =$ 3099d2bb

bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65

$b =$ 64210519

e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1

$G_x =$ 188da80e

b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012

$G_y =$ 07192b95

ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811

Curve P-224

$p =$ 26959946667150639794667015087019630673557916\
26002630814351006629888 1

$r =$ 26959946667150639794667015087019625940457807\
7144243917216827223680 61

$s =$ bd713447 99d5c7fc dc45b59f a3b9ab8f 6a948bc5

$c =$ 5b056c7e 11dd68f4
0469ee7f 3c7a7d74 f7d12111 6506d031 218291fb

$b =$ b4050a85 0c04b3ab
f5413256 5044b0b7 d7bfd8ba 270b3943 2355ffb4

$G_x =$ b70e0cbd 6bb4bf7f
321390b9 4a03c1d3 56c21122 343280d6 115c1d21

$G_y =$ bd376388 b5f723fb
4c22dfe6 cd4375a0 5a074764 44d58199 85007e34

Curve P-256

$p =$ 115792089210356248762697446949407573530086143415290314195533631308867097853951

$r =$ 115792089210356248762697446949407573529996955224135760342422259061068512044369

$s =$ c49d3608 86e70493 6a6678e1 139d26b7 819f7e90

$c =$ 7efba166 2985be94 03cb055c 75d4f7e0 ce8d84a9 c5114abc af317768 0104fa0d

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

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

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

Curve P-384

p = 39402006196394479212279040100143613805079739\
27046544666794829340424572177149687032904726\
60882589380018616069731123 19

r = 39402006196394479212279040100143613805079739\
27046544666794690527962765939911326356939895\
63081522949135544336539426 43

s = a335926a a319a27a 1d00896a 6773a482 7acdac73

c = 79d1e655 f868f02f
ff48dcde e14151dd b80643c1 406d0ca1 0dfe6fc5
2009540a 495e8042 ea5f744f 6e184667 cc722483

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

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

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

Curve P-521

p = 6864797660130609714981900799081393217269435 3\
0014330540939446345918554318339765605212255 9\
6406614545549772963113914808580371219879997 1\
6643812574028291115057151

r = 6864797660130609714981900799081393217269435 3\
0014330540939446345918554318339765539424505 7\
7463332171975329639963713633211138647686124 4\
0380340372808892707005449

s = d09e8800 291cb853 96cc6717 393284aa a0da64ba

c = 0b4 8bfa5f42
0a349495 39d2bdfc 264eeeeb 077688e4 4fbf0ad8
f6d0edb3 7bd6b533 28100051 8e19f1b9 ffbe0fe9
ed8a3c22 00b8f875 e523868c 70c1e5bf 55bad637

b = 051 953eb961
8e1c9a1f 929a21a0 b68540ee a2da725b 99b315f3
b8b48991 8ef109e1 56193951 ec7e937b 1652c0bd
3bb1bf07 3573df88 3d2c34f1 ef451fd4 6b503f00

G_x = c6 858e06b7
0404e9cd 9e3ecb66 2395b442 9c648139 053fb521
f828af60 6b4d3dba a14b5e77 efe75928 fe1dc127
a2ffa8de 3348b3c1 856a429b f97e7e31 c2e5bd66

G_y = 118 39296a78
9a3bc004 5c8a5fb4 2c7d1bd9 98f54449 579b4468
17afbd17 273e662c 97ee7299 5ef42640 c550b901

3fad0761 353c7086 a272c240 88be9476 9fd16650

3. Curves over Binary Fields

For each field degree m, a pseudo-random curve is given, along with a Koblitz curve. The pseudo-random curve has the form

$$E: y^2 + xy = x^3 + x^2 + b,$$

and the Koblitz curve has the form

$$E_a: y^2 + xy = x^3 + ax^2 + 1$$

where $a = 0$ or 1.

For each pseudorandom curve, the cofactor is $f = 2$. The cofactor of each Koblitz curve is $f = 2$ if $a = 1$ and $f = 4$ if $a = 0$.

The coefficients of the pseudo-random curves, and the coordinates of the base points of both kinds of curves, are given in terms of both the polynomial and normal basis representations discussed in 1.3.

For each m, the following parameters are given:

Field Representation:

- The normal basis type T
- The field polynomial (a trinomial or pentanomial)

Koblitz Curve:

- The coefficient a
- The base point order r
- The base point x coordinate G_x
- The base point y coordinate G_y

Pseudo-random curve:

- The base point order r

Pseudo-random curve (Polynomial Basis representation):

- The coefficient b
- The base point x coordinate G_x
- The base point y coordinate G_y

Pseudo-random curve (Normal Basis representation):

- The 160-bit input seed s to the SHA-1 based algorithm
- The coefficient b (i.e., the output of the SHA-1 based algorithm)
- The base point x coordinate G_x
- The base point y coordinate G_y

Integers (such as T, m, and r) are given in decimal form; bit strings and field elements are given in hex.

Degree 163 Binary Field

$T = 4$

$p(t) = t^{163} + t^{7} + t^{6} + t^{3} + 1$

Curve K-163

$a = 1$

$r = 5846006549323611672814741753598448348329118574063$

Polynomial Basis:

$G_x =$ 2 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8

$G_y =$ 2 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9

Normal Basis:

$G_x =$ 0 5679b353 caa46825 fea2d371 3ba450da 0c2a4541

$G_y =$ 2 35b7c671 00506899 06bac3d9 dec76a83 5591edb2

Curve B-163

$r = 5846006549323611672814742442876390689256843201587$

Polynomial Basis:

$b =$ 2 0a601907 b8c953ca 1481eb10 512f7874 4a3205fd

$G_x =$ 3 f0eba162 86a2d57e a0991168 d4994637 e8343e36

$G_y =$ 0 d51fbc6c 71a0094f a2cdd545 b11c5c0c 797324f1

Normal Basis:

$s =$ 85e25bfe 5c86226c db12016f 7553f9d0 e693a268

$b =$ 6 645f3cac f1638e13 9c6cd13e f61734fb c9e3d9fb

$G_x =$ 0 311103c1 7167564a ce77ccb0 9c681f88 6ba54ee8

$G_y =$ 3 33ac13c6 447f2e67 613bf700 9daf98c8 7bb50c7f

Degree 233 Binary Field

$T = 2$

$p(t) = t^{233} + t^{74} + 1$

Curve K-233

$a = 0$

r =3450873173395281893717377931138512760570940988862252 1\
26328087024741343

Polynomial Basis:

G_x = 172 32ba853a 7e731af1
29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126

G_y = 1db 537dece8 19b7f70f
555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3

Normal Basis:

G_x = 0fd e76d9dcd 26e643ac
26f1aa90 1aa12978 4b71fc07 22b2d056 14d650b3

G_y = 064 3e317633 155c9e04
47ba8020 a3c43177 450ee036 d6335014 34cac978

Curve B-233

r = 6901746346790563787434755862277025555839812737345013 5\
55379383634485463

Polynomial Basis:

b = 066 647ede6c 332c7f8c
0923bb58 213b333b 20e9ce42 81fe115f 7d8f90ad

G_x = 0fa c9dfcbac 8313bb21
39f1bb75 5fef65bc 391f8b36 f8f8eb73 71fd558b

G_y = 100 6a08a419 03350678
e58528be bf8a0bef f867a7ca 36716f7e 01f81052

Normal Basis:

s = 74d59ff0 7f6b413d 0ea14b34 4b20a2db 049b50c3

b = 1a0 03e0962d 4f9a8e40
7c904a95 38163adb 82521260 0c7752ad 52233279

G_x = 18b 863524b3 cdfefb94
f2784e0b 116faac5 4404bc91 62a363ba b84a14c5

G_y = 049 25df77bd 8b8ff1a5
ff519417 822bfedf 2bbd7526 44292c98 c7af6e02

Degree 283 Binary Field

$T = 6$

$p(t) = t^{283} + t^{12} + t^{7} + t^{5} + 1$

Curve K-283

$a = 0$

$r = 38853377844514581418389238136470378132848117337930613\backslash$
$2429587499752981582970442260387 3$

Polynomial Basis:

$G_x =$ 503213f 78ca4488 3f1a3b81 62f188e5
53cd265f 23c1567a 16876913 b0c2ac24 58492836

$G_y =$ 1ccda38 0f1c9e31 8d90f95d 07e5426f
e87e45c0 e8184698 e4596236 4e341161 77dd2259

Normal Basis:

$G_x =$ 3ab9593 f8db09fc 188f1d7c 4ac9fcc3
e57fcd3b db15024b 212c7022 9de5fcd9 2eb0ea60

$G_y =$ 2118c47 55e7345c d8f603ef 93b98b10
6fe8854f feb9a3b3 04634cc8 3a0e759f 0c2686b1

Curve B-283

r = 77706755689029162836778476272940756265696259243769048\
891091965267700442777873786928 71

Polynomial Basis:

b = 27b680a c8b8596d a5a4af8a 19a0303f
ca97fd76 45309fa2 a581485a f6263e31 3b79a2f5

G_x = 5f93925 8db7dd90 e1934f8c 70b0dfec
2eed25b8 557eac9c 80e2e198 f8cdbecd 86b12053

G_y = 3676854 fe24141c b98fe6d4 b20d02b4
516ff702 350eddb0 826779c8 13f0df45 be8112f4

Normal Basis:

s = 77e2b073 70eb0f83 2a6dd5b6 2dfc88cd 06bb84be

b = 157261b 894739fb 5a13503f 55f0b3f1
0c560116 66331022 01138cc1 80c0206b dafbc951

G_x = 749468e 464ee468 634b21f7 f61cb700
701817e6 bc36a236 4cb8906e 940948ea a463c35d

G_y = 62968bd 3b489ac5 c9b859da 68475c31 5bafcdc4
ccd0dc90 5b70f624 46f49c05 2f49c08c

Degree 409 Binary Field

$T = 4$

$p(t) = t^{409} + t^{87} + 1$

Curve K-409

$a = 0$

$r = 3305279843951242994759576540163855199142023414821406 0\backslash$

$9642324395022880711289249191050673258457777458014096 3\backslash$

66590617731358671

Polynomial Basis:

$G_x =$ 060f05f 658f49c1 ad3ab189 0f718421 0efd0987 e307c84c 27accfb8 f9f67cc2 c460189e b5aaaa62 ee222eb1 b35540cf e9023746

$G_y =$ 1e36905 0b7c4e42 acba1dac bf04299c 3460782f 918ea427 e6325165 e9ea10e3 da5f6c42 e9c55215 aa9ca27a 5863ec48 d8e0286b

Normal Basis:

$G_x =$ 1b559c7 cba2422e 3affe133 43e808b5 5e012d72 6ca0b7e6 a63aeafb c1e3a98e 10ca0fcf 98350c3b 7f89a975 4a8e1dc0 713cec4a

$G_y =$ 16d8c42 052f07e7 713e7490 eff318ba 1abd6fef 8a5433c8 94b24f5c 817aeb79 852496fb ee803a47 bc8a2038 78ebf1c4 99afd7d6

Curve B-409

r = 661055968790248598951915308032771039828404682964281 21\
928464879830415777482737480520814372376217911096597 98\
67288366567526771

Polynomial Basis:

b = 021a5c2 c8ee9feb 5c4b9a75
3b7b476b 7fd6422e f1f3dd67 4761fa99 d6ac27c8
a9a197b2 72822f6c d57a55aa 4f50ae31 7b13545f

G_x = 15d4860 d088ddb3 496b0c60
64756260 441cde4a f1771d4d b01ffe5b 34e59703
dc255a86 8a118051 5603aeab 60794e54 bb7996a7

G_y = 061b1cf ab6be5f3 2bbfa783
24ed106a 7636b9c5 a7bd198d 0158aa4f 5488d08f
38514f1f df4b4f40 d2181b36 81c364ba 0273c706

Normal Basis:

s = 4099b5a4 57f9d69f 79213d09 4c4bcd4d 4262210b

b = 124d065 1c3d3772 f7f5a1fe
6e715559 e2129bdf a04d52f7 b6ac7c53 2cf0ed06
f610072d 88ad2fdc c50c6fde 72843670 f8b3742a

G_x = 0ceacbc 9f475767 d8e69f3b
5dfab398 13685262 bcacf22b 84c7b6dd 981899e7
318c96f0 761f77c6 02c016ce d7c548de 830d708f

G_y = 199d64b a8f089c6 db0e0b61
e80bb959 34afd0ca f2e8be76 d1c5e9af fc7476df

49142691 ad303902 88aa09bc c59c1573 aa3c009a

Degree 571 Binary Field

$T = 10$

$p(t) = t^{571} + t^{10} + t^{5} + t^{2} + 1$

Curve K-571

$a = 0$

$r =$ 193226876150862917234767594546599367214946366485321 74\
9932861762572575957114478021226813397852270671183470 6\
7128008253514612736749740666173119296824216170925035 5\
5733685276673

Polynomial Basis:

$G_x =$ 26eb7a8 59923fbc 82189631
f8103fe4 ac9ca297 0012d5d4 60248048 01841ca4
43709584 93b205e6 47da304d b4ceb08c bbd1ba39
494776fb 988b4717 4dca88c7 e2945283 a01c8972

$G_y =$ 349dc80 7f4fbf37 4f4aeade
3bca9531 4dd58cec 9f307a54 ffc61efc 006d8a2c
9d4979c0 ac44aea7 4fbebbb9 f772aedc b620b01a
7ba7af1b 320430c8 591984f6 01cd4c14 3ef1c7a3

Normal Basis:

$G_x =$ 04bb2db a418d0db 107adae0
03427e5d 7cc139ac b465e593 4f0bea2a b2f3622b
c29b3d5b 9aa7a1fd fd5d8be6 6057c100 8e71e484
bcd98f22 bf847642 37673674 29ef2ec5 bc3ebcf7

$G_y =$ 44cbb57 de20788d 2c952d7b
56cf39bd 3e89b189 84bd124e 751ceff4 369dd8da
c6a59e6e 745df44d 8220ce22 aa2c852c fcbbef49
ebaa98bd 2483e331 80e04286 feaa2530 50caff60

Curve B-571

r = 3864537523017258344695351890931987344298927329706434 9\
9865723525145151914228956042453614399938941577308313 3\
8811219269444862468724628168130702345282883033324113 9\
3191105285703

Polynomial Basis:

b = 2f40e7e 2221f295 de297117
b7f3d62f 5c6a97ff cb8ceff1 cd6ba8ce 4a9a18ad
84ffabbd 8efa5933 2be7ad67 56a66e29 4afd185a
78ff12aa 520e4de7 39baca0c 7ffeff7f 2955727a

G_x = 303001d 34b85629 6c16c0d4
0d3cd775 0a93d1d2 955fa80a a5f40fc8 db7b2abd
bde53950 f4c0d293 cdd711a3 5b67fb14 99ae6003
8614f139 4abfa3b4 c850d927 e1e7769c 8eec2d19

G_y = 37bf273 42da639b 6dccfffe
b73d69d7 8c6c27a6 009cbbca 1980f853 3921e8a6
84423e43 bab08a57 6291af8f 461bb2a8 b3531d2f
0485c19b 16e2f151 6e23dd3c 1a4827af 1b8ac15b

Normal Basis:

s = 2aa058f7 3a0e33ab 486b0f61 0410c53a 7f132310

b = 3762d0d 47116006 179da356
88eeaccf 591a5cde a7500011 8d9608c5 9132d434
26101a1d fb377411 5f586623 f75f0000 1ce61198
3c1275fa 31f5bc9f 4be1a0f4 67f01ca8 85c74777

G_x = 0735e03 5def5925 cc33173e

$G_y =$ b2a8ce77 67522b46 6d278b65 0a291612 7dfea9d2
d361089f 0a7a0247 a184e1c7 0d417866 e0fe0feb
0ff8f2f3 f9176418 f97d117e 624e2015 df1662a8
04a3642 0572616c df7e606f
ccadaecf c3b76dab 0eb1248d d03fbdfc 9cd3242c
4726be57 9855e812 de7ec5c5 00b4576a 24628048
b6a72d88 0062eed0 dd34b109 6d3acbb6 b01a4a97

APPENDIX 6.1: IMPLEMENTATION OF MODULAR ARITHMETIC

The prime moduli in the above examples are of a special type (called *generalized Mersenne numbers*) for which modular multiplication can be carried out more efficiently than in general. This appendix provides the rules for implementing this faster arithmetic, for each of the prime moduli appearing in the examples.

The usual way to multiply two integers (mod m) is to take the integer product and reduce it (mod m). One therefore has the following problem: given an integer A less than m^2, compute

$$B := A \bmod m.$$

In general, one must obtain B as the remainder of an integer division. If m is a generalized Mersenne number, however, then B can be expressed as a sum or difference (mod m) of a small number of terms. To compute this expression, one can evaluate the integer sum or difference and reduce the result modulo m. The latter reduction can be accomplished by adding or subtracting a few copies of m.

The prime moduli p for each of the five example curves is a generalized Mersenne number.

Curve P-192:

The modulus for this curve is $p = 2^{192} - 2^{64} - 1$. Every integer A less than p^2 can be written

$$A = A_5 \cdot 2^{320} + A_4 \cdot 2^{256} + A_3 \cdot 2^{192} + A_2 \cdot 2^{128} + A_1 \cdot 2^{64} + A_0,$$

where each A_i is a 64-bit integer. The expression for B is

$$B := T + S_1 + S_2 + S_3 \bmod p;$$

where the 192-bit terms are given by

$$T = A_2 \cdot 2^{128} + A_1 \cdot 2^{64} + A_0$$

$$S_1 = \qquad A_3 \cdot 2^{64} + A_3$$

$$S_2 = A_4 \cdot 2^{128} + A_4 \cdot 2^{64}$$

$$S_3 = A_5 \cdot 2^{128} + A_5 \cdot 2^{64} + A_5.$$

Curve P-224:

The modulus for this curve is $p = 2^{224} - 2^{96} + 1$. Every integer A less than p^2 can be written

$A = A_{13} \cdot 2^{416} + A_{12} \cdot 2^{384} + A_{11} \cdot 2^{352} + A_{10} \cdot 2^{320} + A_9 \cdot 2^{288} + A_8 \cdot 2^{256} + A_7 \cdot 2^{224} + A_6 \cdot 2^{192} + A_5 \cdot 2^{160} + A_4 \cdot 2^{128} + A_3 \cdot 2^{96} + A_2 \cdot 2^{64} + A_1 \cdot 2^{32} + A_0,$

where each A_i is a 32-bit integer. As a concatenation of 32-bit words, this can be denoted by

$$A = (A_{13} \,||\, A_{12} \,||\, \cdots \,||\, A_0).$$

The expression for B is

$$B := T + S_1 + S_2 - D_1 - D_2 \bmod p,$$

where the 224-bit terms are given by

$$T = (A_6 \,||\, A_5 \,||\, A_4 \,||\, A_3 \,||\, A_2 \,||\, A_1 \,||\, A_0)$$

$$S_1 = (A_{10} \,||\, A_9 \,||\, A_8 \,||\, A_7 \,||\, 0 \,||\, 0 \,||\, 0)$$

$$S_2 = (0 \,||\, A_{13} \,||\, A_{12} \,||\, A_{11} \,||\, 0 \,||\, 0 \,||\, 0)$$

$$D_1 = (A_{13} \,||\, A_{12} \,||\, A_{11} \,||\, A_{10} \,||\, A_9 \,||\, A_8 \,||\, A_7)$$

$$D_2 = (0 \,||\, 0 \,||\, 0 \,||\, 0 \,||\, A_{13} \,||\, A_{12} \,||\, A_{11}).$$

Curve P-256:

The modulus for this curve is $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$. Every integer A less than p^2 can be written

$$A = A_{15} \cdot 2^{480} + A_{14} \cdot 2^{448} + A_{13} \cdot 2^{416} + A_{12} \cdot 2^{384} + A_{11} \cdot 2^{352} +$$
$$A_{10} \cdot 2^{320} + A_9 \cdot 2^{288} + A_8 \cdot 2^{256} + A_7 \cdot 2^{224} + A_6 \cdot 2^{192} + A_5 \cdot 2^{160} +$$
$$A_4 \cdot 2^{128} + A_3 \cdot 2^{96} + A_2 \cdot 2^{64} + A_1 \cdot 2^{32} + A_0,$$

where each A_i is a 32-bit integer. As a concatenation of 32-bit words, this can be denoted by

$$A = (A_{15} \,||\, A_{14} \,||\, \cdots \,||\, A_0).$$

The expression for B is

$$B := T + 2S_1 + 2S_2 + S_3 + S_4 - D_1 - D_2 - D_3 - D_4 \bmod p,$$

where the 256-bit terms are given by

$$T = (A_7 \,||\, A_6 \,||\, A_5 \,||\, A_4 \,||\, A_3 \,||\, A_2 \,||\, A_1 \,||\, A_0)$$
$$S_1 = (A_{15} \,||\, A_{14} \,||\, A_{13} \,||\, A_{12} \,||\, A_{11} \,||\, 0 \,||\, 0 \,||\, 0)$$
$$S_2 = (0 \,||\, A_{15} \,||\, A_{14} \,||\, A_{13} \,||\, A_{12} \,||\, 0 \,||\, 0 \,||\, 0)$$
$$S_3 = (A_{15} \,||\, A_{14} \,||\, 0 \,||\, 0 \,||\, 0 \,||\, A_{10} \,||\, A_9 \,||\, A_8)$$
$$S_4 = (A_8 \,||\, A_{13} \,||\, A_{15} \,||\, A_{14} \,||\, A_{13} \,||\, A_{11} \,||\, A_{10} \,||\, A_9)$$
$$D_1 = (A_{10} \,||\, A_8 \,||\, 0 \,||\, 0 \,||\, 0 \,||\, A_{13} \,||\, A_{12} \,||\, A_{11})$$
$$D_2 = (A_{11} \,||\, A_9 \,||\, 0 \,||\, 0 \,||\, A_{15} \,||\, A_{14} \,||\, A_{13} \,||\, A_{12})$$
$$D_3 = (A_{12} \,||\, 0 \,||\, A_{10} \,||\, A_9 \,||\, A_8 \,||\, A_{15} \,||\, A_{14} \,||\, A_{13})$$
$$D_4 = (A_{13} \,||\, 0 \,||\, A_{11} \,||\, A_{10} \,||\, A_9 \,||\, 0 \,||\, A_{15} \,||\, A_{14}).$$

Curve P-384:

The modulus for this curve is $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$. Every integer A less than p^2 can be written

$$A = A_{23} \cdot 2^{736} + A_{22} \cdot 2^{704} + A_{21} \cdot 2^{672} + A_{20} \cdot 2^{640} + A_{19} \cdot 2^{608} + A_{18} \cdot 2^{576} + A_{17} \cdot 2^{544} + A_{16} \cdot 2^{512} + A_{15} \cdot 2^{480} + A_{14} \cdot 2^{448} + A_{13} \cdot 2^{416} + A_{12} \cdot 2^{384} + A_{11} \cdot 2^{352} + A_{10} \cdot 2^{320} + A_9 \cdot 2^{288} + A_8 \cdot 2^{256} + A_7 \cdot 2^{224} + A_6 \cdot 2^{192} + A_5 \cdot 2^{160} + A_4 \cdot 2^{128} + A_3 \cdot 2^{96} + A_2 \cdot 2^{64} + A_1 \cdot 2^{32} + A_0,$$

where each A_i is a 32-bit integer. As a concatenation of 32-bit words, this can be denoted by

$$A = (A_{23} \| A_{22} \| \cdots \| A_0).$$

The expression for B is

$$B := T + 2S_1 + S_2 + S_3 + S_4 + S_5 + S_6 - D_1 - D_2 - D_3 \bmod p,$$

where the 384-bit terms are given by

$$T = (A_{11} \| A_{10} \| A_9 \| A_8 \| A_7 \| A_6 \| A_5 \| A_4 \| A_3 \| A_2 \| A_1 \| A_0)$$

$$S_1 = (0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_{21} \| 0 \| 0 \| 0 \| 0)$$

$$S_2 = (A_{23} \| A_{22} \| A_{21} \| A_{20} \| A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12})$$

$$S_3 = (A_{20} \| A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{23} \| A_{22} \| A_{21})$$

$$S_4 = (A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{20} \| 0 \| A_{23} \| 0)$$

$$S_5 = (0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_{21} \| A_{20} \| 0 \| 0 \| 0 \| 0)$$

$$S_6 = (0 \| 0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_{21} \| 0 \| 0 \| A_{20})$$

$$D_1 = (A_{22} \| A_{21} \| A_{20} \| A_{19} \| A_{18} \| A_{17} \| A_{16} \| A_{15} \| A_{14} \| A_{13} \| A_{12} \| A_{23})$$

$$D_2 = (0 \| 0 \| 0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{22} \| A_2 \| A_{20} \| 0)$$

$$D_3 = (0 \| 0 \| 0 \| 0 \| 0 \| 0 \| 0 \| A_{23} \| A_{23} \| 0 \| 0 \| 0).$$

Curve P-521:

The modulus for this curve is $p = 2^{521} - 1$. Every integer A less than p^2 can be written

$$A = A_1 \cdot 2^{521} + A_0,$$

The expression for B is

$$B := A_0 + A_1 \bmod p$$

APPENDIX 6.2: NORMAL BASES

The elements of $GF(2^m)$ are expressed in terms of the type T normal *basis* 2 B for $GF(2^m)$, for some T. Each element has a unique representation as a bit string

$$(a_0\, a_1 \ldots\ a_{m-1})$$

The arithmetic operations are performed as follows.

<u>*Addition*</u>: addition of two elements is implemented by bitwise addition modulo 2. Thus, for example,

$$(1100111) + (1010010) = (0110101).$$

<u>*Squaring*</u>: if

$$a = (a_0\, a_1 \ldots\ a_{m-1})$$

then

$$a^2 = (a_{m-1}\, a_0\, a_1 \ldots\ a_{m-2})$$

<u>*Multiplication*</u>: to perform multiplication, one first constructs a function $F(\underline{u},\underline{v})$ on inputs

$$\underline{u} = (u_0\, u_1 \ldots\ u_{m-1}) \qquad \text{and} \qquad \underline{v} = (v_0\, v_1 \ldots\ v_{m-1})$$

as follows.

1. Set $p \neg Tm + 1$
2. Let u be an integer having order T modulo p

2 It is assumed in this section that m is odd and T is even, since this is the only case considered in this standard.

3. Compute the sequence $F(1); F(2),\ldots,F(p\text{-}1)$ as follows:

   3.1 Set $w \neg 1$

   3.2 For j from 0 to T-1 do

   Set $n \neg w$

   For i from 0 to m-1 do

   Set $F(n) \neg i$

   Set $n \neg 2n \bmod p$

   Set $w \neg uw \bmod p$

4. Output the formula

$$F(\underline{u}, \underline{v}) := \mathop{S}_{k=1}^{p-2} u_{F(k+1)} v_{F(p-k)}.$$

This computation need only be performed once per basis.

Given the function F for B, one computes the product

$$(c_0\, c_1 \ldots c_{m-1}) = (a_0\, a_1 \ldots a_{m-1}) \times (b_0\, b_1 \ldots b_{m-1})$$

as follows.

1. Set $(u_0\, u_1 \ldots u_{m-1}) \neg (a_0\, a_1 \ldots a_{m-1})$
2. Set $(v_0\, v_1 \ldots v_{m-1}) \neg (b_0\, b_1 \ldots b_{m-1})$
3. For k from 0 to m - 1 do

   3.1 Compute

   $$c_k := F(\underline{u}, \underline{v})$$

   3.2 Set $u \neg$ **LeftShift** (u) and $v \neg$ **LeftShift** (v), where **LeftShift** denotes the circular left shift operation.

4. Output $c := (c_0\, c_1 \ldots c_{m-1})$

EXAMPLE. For the type 4 normal basis for $GF(2^7)$, one has $p = 29$ and $u = 12$ or 17. Thus the values of F are given by

$F(1) = 0$	$F(8) = 3$	$F(15) = 6$	$F(22) = 5$
$F(2) = 1$	$F(9) = 3$	$F(16) = 4$	$F(23) = 6$
$F(3) = 5$	$F(10) = 2$	$F(17) = 0$	$F(24) = 1$
$F(4) = 2$	$F(11) = 4$	$F(18) = 4$	$F(25) = 2$
$F(5) = 1$	$F(12) = 0$	$F(19) = 2$	$F(26) = 5$
$F(6) = 6$	$F(13) = 4$	$F(20) = 3$	$F(27) = 1$
$F(7) = 5$	$F(14) = 6$	$F(21) = 3$	$F(28) = 0$

Therefore

$$\begin{aligned} F(\underline{u}; \underline{v}) = {} & u_0 v_1 + u_1 (v_0 + v_2 + v_5 + v_6) + u_2 (v_1 + v_3 + v_4 + v_5) \\ & + u_3 (v_2 + v_5) + u_4 (v_2 + v_6) + u_5 (v_1 + v_2 + v_3 + v_6) \\ & + u_6 (v_1 + v_4 + v_5 + v_6). \end{aligned}$$

Thus, if

$$a = (1\ 0\ 1\ 0\ 1\ 1\ 1) \text{ and } b = (1\ 1\ 0\ 0\ 0\ 0\ 1),$$

then

$$c_0 = F\,(\,(1\ 0\ 1\ 0\ 1\ 1\ 1), (1\ 1\ 0\ 0\ 0\ 0\ 1)\,) = 1,$$

$$c_1 = F\,(\,(0\ 1\ 0\ 1\ 1\ 1\ 1), (1\ 0\ 0\ 0\ 0\ 1\ 1)\,) = 0,$$

$$\vdots$$

$$c_6 = F\,(\,(1\ 1\ 0\ 1\ 0\ 1\ 1); (1\ 1\ 1\ 0\ 0\ 0\ 0)\,) = 1,$$

so that $c = ab = (1\ 0\ 1\ 1\ 0\ 0\ 1)$.

APPENDIX 6.3: SCALAR MULTIPLICATION ON KOBLITZ CURVES

This appendix describes a particularly efficient method of computing the scalar multiple nP on the Koblitz curve E_a over $GF(2^m)$.

The operation t is defined by

$$t\,(x, y) = (x^2, y^2)$$

When the normal basis representation is used, then the operation t is implemented by performing right circular shifts on the bit strings representing x and y.

Given m and a, define the following parameters:

- C is some integer greater than 5.
- $\mu := (-1)^{1-a}$
- For $i = 0$ and $i = 1$, define the sequence $s_i(m)$ by

$$s_i(0) = 0, \qquad s_i(1) = 1 - i,$$

$$s_i(m) = \mu \cdot s_i(m - 1) - 2\, s_i(m - 2) + (-1)^i$$

- Define the sequence $V(m)$

$$V(0) = 2, \qquad V(1) = \mu$$

$$V(m) = \mu \cdot v(m - 1) - 2V(m - 2).$$

For the example curves, the quantities $s_i(m)$ and $V(m)$ are as follows.

Curve K-163:

$s_0(163) = 2579386439110731650419537$

$s_1(163) = -7553600644762263754615 94$

$V(163) = -4845466632539410776804317$

Curve K-233:

$s_0(233) = -27859711741434429761757834964435883$

$s_1(233) = -44192136247082304936052160908934886$

$V(233) = -13738154601110823539498729965136 6779$

Curve K-283:

$s_0(283) = -66598153210904904110879553600159146928 0025$

$s_1(283) = 115586005490913677519228107259160991394 5968$

$V(283) = 77772448708728309992877919709628239775 69917$

Curve K-409:

$s_0(409) = -183075104560023821378103171987564613785905424875 5686\backslash$
9338419259

$s_1(409) = -889304852613830409719665324184421267962656610099 6606\backslash$
444816790

$V(409)=104572887373156259274476853870483207376387969576 8757\backslash$
5791173829

Curve K-571:

$s_0(571)$ = -3737319446876463692429385892476115567147293964596 13\
1024123406420235241916729983261305

$s_1(571)$ = -3191857706446416099583814595948959674131968912148564\
6586105651175898284851583261224875 2

$V(571)$=-1483809269816914138996191402970514903645425741804 93\
9362329123395342085168289731114598 43

The following algorithm computes the scalar multiple nP on the Koblitz curve E_a over $GF(2^m)$. The average number of elliptic additions and subtractions is at most ~ 1 + (m/3), and is at most ~ m/3 with probability at least 1 - 2^{5-C}.

For i = 0 to 1 do

$n¢ \leftarrow \lfloor n / 2^{a-C + (m-9) / 2} \rfloor$

$g¢ \leftarrow s_i(m) \cdot n¢$

$h¢ \leftarrow \lfloor g¢ / 2^m \rfloor$

$j¢ \leftarrow V(m) \cdot h¢$

$l¢ \leftarrow \mathrm{Round}((g¢ + j¢) / 2^{(m+5) / 2})$

$l_i \leftarrow l¢ / 2^C$

$f_i \leftarrow \mathrm{Round}(l_i)$

$h_i \leftarrow l_i - f_i$

$h_i \leftarrow 0$

$h \leftarrow 2 h_0 + m h_1$

If h ³ 1

then

if $h_o - 3\,mh_1 < -1$

then set $h_1 \leftarrow m$

else set $h_0 \leftarrow 1$

else

if $h_0 + 4\,m\,h_1$ ³ 2

then set $h_1 \leftarrow m$

If $h < -1$

then

if $h_0 - 3\,m\,h_1 \geq 1$

then set $h_1 \leftarrow -m$

else set $h_0 \leftarrow -1$

else

if $h_0 + 4\,m\,h_1 < -2$

then set $h_1 \leftarrow -m$

$q_0 \leftarrow f_0 + h_0$

$q_1 \leftarrow f_1 + h_1$

$r_0 \leftarrow n - (s_0 + m\,s_1)\,q_0 - 2s_1\,q_1$

$r_1 \leftarrow s_1\,q_0 - s_0\,q_1$

Set $Q \leftarrow O$

$P_0 \leftarrow P$

While r_0 ¹ 0 or r_1 ¹ 0

If r_0 odd then

set $u \leftarrow 2 - (r_0 - 2\,r_1 \bmod 4)$

set $r_0 \leftarrow r_0 - u$

if $u = 1$ then set $Q \leftarrow Q + P_0$

if $u = -1$ then set $Q \leftarrow Q - P_0$

Set $P_0 \leftarrow tP_0$

Set $(r_0, r_1) \leftarrow (r_1 + mr_0/2, -r_0/2)$

Endwhile

Output Q

APPENDIX 6.4: GENERATION OF PSEUDO-RANDOM CURVES (PRIME CASE)

Let l be the bit length of p, and define

$$v = \lfloor (l - 1)/160 \rfloor$$

$$w = l - 160v - 1$$

1. Choose an arbitrary 160-bit string s.
2. Compute h:= SHA-1(s).
3. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
4. Let z be the integer whose binary expansion is given by the 160-bit string s.
5. For i from 1 to v do:
   - 5.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$.
   - 5.2 Compute h_i :=SHA-1(s_i).
6. Let h be the bit string obtained by the concatenation of h_0 , h_1, . . . , h_v as follows:

$$h = h_0 \parallel h_1 \parallel \ldots \parallel h_v$$

7. Let c be the integer whose binary expansion is given by the bit string h.
8. If $c = 0$ or $4c + 27 \equiv 0 \pmod{p}$, then go to Step 1.
9. Choose integers $a, b \in GF(p)$ such that

$$c\, b^2 \equiv a^3 \pmod{p}.$$

(The simplest choice is $a = c$ and $b = c$. However, one may want to choose differently for performance reasons.)

10. Check that the elliptic curve E over $GF(p)$ given by $y^2 = x^3 + ax + b$ has suitable order. If not, go to Step 1.

APPENDIX 6.5: VERIFICATION OF CURVE PSEUDO-RANDOMNESS (PRIME CASE)

Given the 160-bit seed value s, one can verify that the coefficient b was obtained from s via the cryptographic hash function SHA-1 as follows.

Let l be the bit length of p, and define

$$v = \lfloor (l - 1)/160 \rfloor$$

$$w = l - 160v - 1$$

1. Compute h := SHA-1(s).
2. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
3. Let z be the integer whose binary expansion is given by the 160-bit string s.
4. For i from 1 to v do
   - 4.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$.
   - 4.2 Compute h_i :=SHA-1(s_i).
5. Let h be the bit string obtained by the concatenation of h_0 , h_1, . . . , h_v as follows:

$$h = h_0 \parallel h_1 \parallel \ldots \parallel h_v.$$

6. Let c be the integer whose binary expansion is given by the bit string h.
7. Verify that $b^2 c \equiv -27 \pmod{p}$.

APPENDIX 6.6: GENERATION OF PSEUDO-RANDOM CURVES (BINARY CASE)

Let:

$$v = \lfloor (m - 1) / B \rfloor$$

$$w = m - Bv$$

1. Choose an arbitrary 160-bit string s.
2, Compute h := SHA-1(s)
3. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
4. Let z be the integer whose binary expansion is given by the 160-bit string s.
5. For i from 1 to v do:
   5.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$.
   5.2 Compute h_i :=SHA-1(s_i).
6. Let h be the bit string obtained by the concatenation of h_0 , h_1, . . . , h_v as follows:

$$h = h_0 \parallel h_1 \parallel \ldots \parallel h_v.$$

7. Let b be the element of $GF(2^m)$ which binary expansion is given by the bit string h.
8. Choose an element a of $GF(2^m)$.
9. Check that the elliptic curve E over $GF(2^m)$ given by $y^2 + xy = x^3 + ax^2 + b$ has suitable order. If not, go to Step 1.

APPENDIX 6.7: VERIFICATION OF CURVE PSEUDO-RANDOMNESS (BINARY CASE)

Given the 160-bit seed value s, one can verify that the coefficient b was obtained from s via the cryptographic hash function SHA-1 as follows.

Define

$$v = \lfloor (m - 1) / 160 \rfloor$$

$$w = m - 160v$$

1. Compute h := SHA-1(s)
2. Let h_0 be the bit string obtained by taking the w rightmost bits of h.
3. Let z be the integer whose binary expansion is given by the 160-bit string s.
4. For i from 1 to v do

    4.1 Define the 160-bit string s_i to be binary expansion of the integer $(z + i) \bmod (2^{160})$

    4.2 Compute h_i :=SHA-1(s_i).
5. Let h be the bit string obtained by the concatenation of h_0 , h_1, . . . , h_v as follows:

$$h = h_0 \parallel h_1 \parallel \ldots \parallel h_v.$$

6. Let c be the element of $GF(2^m)$ which is represented by the bit string h.
7. Verify that $c = b$.

APPENDIX 6.8: POLYNOMIAL BASIS TO NORMAL BASIS CONVERSION

Suppose that a an element of the field $GF(2^m)$. Denote by **p** the bit string representing ′Y with respect to a given polynomial basis. It is desired to compute **n**, the bit string representing a with respect to a given normal basis. This is done via the matrix computation

$$\mathbf{p}\,\Gamma = \mathbf{n}$$

Where Γ is an m-by-m matrix with entries in $GF(2)$. The matrix Γ, which depends only on the bases, can be computed easily given its second-to-last row. The second-to-last row for each conversion is given in the table below.

Degree 163:

3 e173bfaf 3a86434d 883a2918 a489ddbd 69fe84e1

Degree 233:

0be 19b89595 28bbc490
038f4bc4 da8bdfc1 ca36bb05 853fd0ed 0ae200ce

Degree 283:

3347f17 521fdabc 62ec1551 acf156fb
0bceb855 f174d4c1 7807511c 9f745382 add53bc3

Degree 409:

0eb00f2 ea95fd6c 64024e7f
0b68b81f 5ff8a467 acc2b4c3 b9372843 6265c7ff

a06d896c ae3a7e31 e295ec30 3eb9f769 de78bef5

Degree 571:

7940ffa ef996513 4d59dcbf
e5bf239b e4fe4b41 05959c5d 4d942ffd 46ea35f3
e3cdb0e1 04a2aa01 cef30a3a 49478011 196bfb43
c55091b6 1174d7c0 8d0cdd61 3bf6748a bad972a4

Given the second-to-last row **r** of Γ, the rest of the matrix is computed as follows. Let b be the element of $GF(2^m)$ whose representation with respect to the normal basis is **r**. Then the rows of Γ, from top to bottom, are the bit strings representing the elements

$$b^{m-1}, b^{m-2}, \ldots, \upsilon^2, \upsilon, 1$$

with respect to the normal basis. (Note that the element 1 is represented by the all-1 bit string.)

Alternatively, the matrix is the inverse of the matrix described in Appendix 6.9.

More details of these computations can be found in Annex A.7 of the IEEE P1363 standard.

APPENDIX 6.9: NORMAL BASIS TO POLYNOMIAL BASIS CONVERSION

Suppose that a an element of the field $GF(2^m)$. Denote by **n** the bit string representing a with respect to a given normal basis. It is desired to compute **p**, the bit string representing a with respect to a given polynomial basis. This is done via the matrix computation

$$\mathbf{n}\,\Gamma = \mathbf{p}$$

where Γ is an m-by-m matrix with entries in $GF(2)$. The matrix Γ, which depends only on the bases, can be computed easily given its top row. The top row for each conversion is given in the table below.

Degree 163:

7 15169c10 9c612e39 0d347c74 8342bcd3 b02a0bef

Degree 233:

149 9e398ac5 d79e3685
59b35ca4 9bb7305d a6c0390b cf9e2300 253203c9

Degree 283:

31e0ed7 91c3282d c5624a72 0818049d
053e8c7a b8663792 bc1d792e ba9867fc 7b317a99

Degree 409:

0dfa06b e206aa97 b7a41fff
b9b0c55f 8f048062 fbe8381b 4248adf9 2912ccc8
e3f91a24 e1cfb395 0532b988 971c2304 2e85708d

Degree 571:

452186b bf5840a0 bcf8c9f0

2a54efa0 4e813b43 c3d41496 06c4d27b 487bf107

393c8907 f79d9778 beb35ee8 7467d328 8274caeb

da6ce05a eb4ca5cf 3c3044bd 4372232f 2c1a27c4

Given the top row **r** of Γ, the rest of the matrix is computed as follows. Let b be the element of $GF(2^m)$ whose representation with respect to the polynomial basis is **r**. Then the rows of Γ, from top to bottom, are the bit strings representing the elements

$$b, \upsilon^2, \upsilon^{2^2}, \ldots, \upsilon^{2^{m-1}}$$

with respect to the polynomial basis.

Alternatively, the matrix is the inverse of the matrix described in Appendix 6.8.

More details of these computations can be found in Annex A.7 of the IEEE P1363 standard.