FIPS PUB 186

FEDERAL INFORMATION
PROCESSING STANDARDS PUBLICATION

1994 May 19

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

DIGITAL SIGNATURE STANDARD (DSS)

CATEGORY: COMPUTER SECURITY

U.S. DEPARTMENT OF COMMERCE, Ronald Brown, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY,
Arati Prabhakar, 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 111(d) of the Federal Property and Administrative Services Act of 1949 as amended by 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 the Computer Systems 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, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899.

James H. Burrows, Director
Computer Systems Laboratory

Abstract

This standard specifies a Digital Signature Algorithm (DSA) 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 Standard.

Federal Information
Processing Standards Publication 186

1994 May 19

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 111(d) of the Federal Property and Administrative Services Act of 1949 as amended by 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 a Digital Signature Algorithm (DSA) appropriate for applications requiring a digital, rather than written, signature. The DSA digital signature is a pair of large numbers represented in a computer as strings of binary digits. The digital signature is computed using a set of rules (i.e., the DSA) and a set of parameters such that the identity of the signatory and integrity of the data can be verified. The DSA 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 DSA 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. Similar procedures may be used to generate and verify signatures for stored as well as transmitted data.

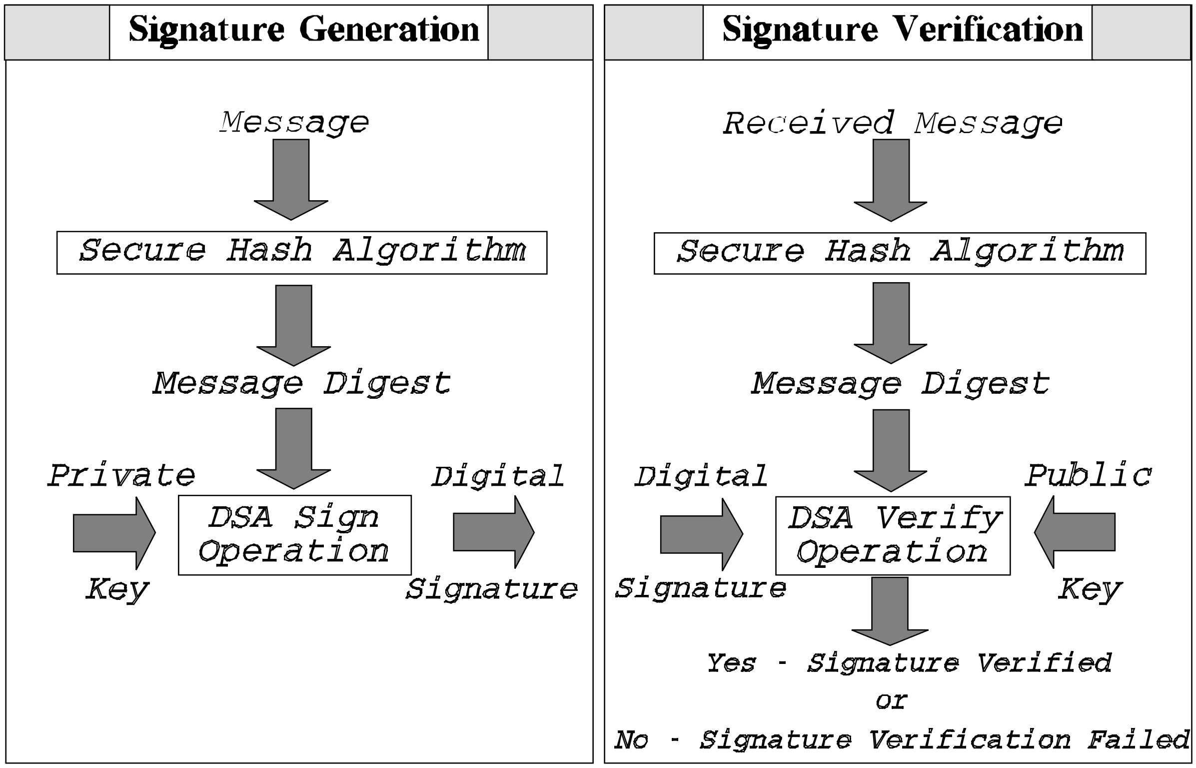

Figure 1: Using the SHA with the DSA

Approving Authority: Secretary of Commerce.

Maintenance Agency: U.S. Department of Commerce, National Institute of Standards and Technology (NIST), Computer Systems Laboratory (CSL).

Applicability: This standard is applicable to all Federal departments and agencies for the protection of 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 which 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: The DSA authenticates the integrity of the signed data and the identity of the signatory. The DSA may also be used in proving to a third party that data was actually signed by the generator of the signature. The DSA is intended for use in electronic mail, electronic funds transfer, electronic data interchange, software distribution, data storage, and other applications which require data integrity assurance and data origin authentication.

Implementations: The DSA may be implemented in software, firmware, hardware, or any combination thereof. NIST is developing a validation program to test implementations for conformance to this standard. Information about the planned validation program can be obtained from the National Institute of Standards and Technology, Computer Systems Laboratory, Attn: DSS Validation, Gaithersburg, MD 20899.

Export Control: Implementations of this standard are subject to Federal Government export controls as specified in Title 15, Code of Federal Regulations, Parts 768 through 799. Exporters are advised to contact the Department of Commerce, Bureau of Export Administration for more information.

Patents: The Department of Commerce is not aware of any patents that would be infringed by this standard.

Implementation Schedule: This standard becomes effective December 1, 1994.

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

Cross Index:

a. Federal Information Resources Management Regulations (FIRMR) subpart 201.20.303, Standards, and subpart 201.39.1002, Federal Standards.

b. FIPS PUB 46-2, Data Encryption Standard.

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

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

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

f. FIPS PUB 180, Secure Hash Standard.

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 departments and agencies 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 with 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, Technology Building, Room B-154, Gaithersburg, MD 20899.

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 (FIPS PUB 186), 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

1994 May 19

Specifications for the

DIGITAL SIGNATURE STANDARD (DSS)

1. INTRODUCTION

This publication prescribes the Digital Signature Algorithm (DSA) for digital signature generation and verification. Additional information is provided in Appendices 1 through 5.

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 the DSA. 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 the DSA for digital signature generation and verification. In addition, the criteria for the public and private keys required by the algorithm are provided.

3. USE OF THE DSA ALGORITHM

The DSA 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) specified in FIPS 180. 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 $512 \le L \le 1024$ 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. 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

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

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(M) is a 160-bit string output by the Secure Hash Algorithm specified in FIPS 180. 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. 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

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

$$u1 = ((SHA(M'))w) \bmod q$$

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

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

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.

APPENDIX 1. A PROOF THAT $v = r'$

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

The purpose of this appendix is to show that 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 $m \bmod q = n \bmod q$, then $g^m \bmod p = g^n \bmod p$.

Proof: We have

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

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

$$\begin{aligned} g^m \bmod p &= g^{n+kq} \bmod p \\ &= (g^n\, g^{kq}) \bmod p \\ &= ((g^n \bmod p)\,(g^q \bmod p)^k) \bmod p \\ &= g^n \bmod p \end{aligned}$$

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

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

$$u1 = ((SHA(M'))w) \bmod q = ((SHA(M))w) \bmod q$$

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

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

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

$$= ((g^{SHA(M)w}\ y^{rw}) \bmod p) \bmod q$$

$$= ((g^{SHA(M)w}\ g^{xrw}) \bmod p) \bmod q$$

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

Also

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

Hence

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

$$(SHA(M) + xr)w \bmod q = k \bmod q.$$

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 DSS 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 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

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

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

$$V_k = SHA[(SEED + offset + k) \bmod 2^g].$$

[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 $c = X \bmod 2q$ 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 ($160 \le b \le 512$) and G(t,c) is 160 bits. One way to construct G is via the Secure Hash Algorithm (SHA), as defined in the Secure Hash Standard (SHS). The 160-bit message digest output of the SHA algorithm when message M is input is denoted by SHA(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 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. 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(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

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:

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

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:

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

(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:

$$H_0 \ \| \ H_1 \ \| \ H_2 \ \| \ H_3 \ \| \ H_4$$

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

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

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

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

Step 2. For i = 1 to 5 do

$x_i = t_i$ XOR c_i

Step 3. For i = 1 to 5 do

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

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

$a1 = x_i$

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

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

Step 4. For i = 1 to 5 do

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

Step 5. Let

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

APPENDIX 4. GENERATION OF OTHER QUANTITIES

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 DSS.

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 $t = i \text{ DIV } h$, 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 e0427dbd

With this SEED, the algorithm found p and q when the counter was at 38.

x was generated by the algorithm described in appendix 3, section 3.1, using the SHA to construct G (as in appendix 3, section 3.3) and a 160-bit XSEED:

XSEED =

bd029bbe 7f51960b cf9edb2b 61f06f0f eb5a38b6

t =

67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0

x = G(t,XSEED) mod q

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

KSEED =

687a66d9 0648f993 867e121f 4ddf9ddb 01205584

t =

EFCDAB89 98BADCFE 10325476 C3D2E1F0 67452301

k = G(t,KSEED) mod q

Finally:

h = 2

p =

d411a4a0 e393f6aa b0f08b14 d1845866 5b3e4dbd ce254454
3fe365cf 71c86224 12db6e7d d02bbe13 d88c58d7 263e9023
6af17ac8 a9fe5f24 9cc81f42 7fc543f7

q =

b20db0b1 01df0c66 24fc1392 ba55f77d 577481e5

g =

b3085510 021f9990 49a9e7cd 3872ce99 58186b50 07e7adaf
25248b58 a3dc4f71 781d21f2 df89b717 47bd54b3 23bbecc4
43ec1d3e 020dadab bf782257 8255c104

x =

6b2cd935 d0192d54 e2c942b5 74c80102 c8f8ef67

k =

79577ddc aafddc03 8b865b19 f8eb1ada 8a2838c6

kinv =

2784e3d6 72d972a7 4e22c67f 4f4f726e cc751efa

M = ASCII form of "abc" (See FIPS PUB YY, Appendix A)

SHA(M) =

0164b8a9 14cd2a5e 74c4f7ff 082c4d97 f1edf880

y =

b32fbec0 3175791d f08c3f86 1c81df7d e7e0cba7 f1c4f726
9bb12d6c 628784fb 742e66ed 315754df e38b5984 e94d3725
37f655cb 3ea4767c 878cbd2d 783ee662

r =

9b77f705 4c81531c 4e46a469 2fbfe0f7 7f7ebff2 s =

95b4f608	1f8f890e	4b5a199e	f10ffe21	f52b2d68

w =

0ceb5f6b	875f6b67	7e093134	df70b0d4	3226680c

u1 =

347089a2	9897273b	fc7a774f	a70e0e0e	153bcc95

u2 =

793d9312	a41b88af	aa2c1bd9	49ec3bee	2e75d2f5

g^{u1} mod p =

57a198ab	2c8ea0b6	4810767a	ff732fb2	da5fcafb	278889f1
96b60b9c	1285b848	1d08505e	201a5c68	523a15ee	2fb62a56
d141dc4d	71925ef0	6acde0a5	b89c5671		

y^{u2} mod p =

5d983d20	be604e23	fb19bec8	7860490a	41b865dc	0f5623f4
0724a795	021bcd8c	93a39ddf	51cae380	fb6d682a	676608f7
65227ff0	5e44ccf4	9767e4a6	0832d33f		

v =

9b77f705	4c81531c	4e46a469	2fbfe0f7	7f7ebff2