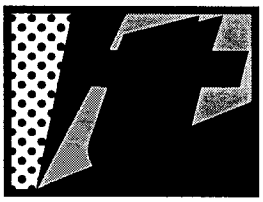
The Digital Signature Standard

Proposed by NIST



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 a computer using a set of rules (i.e., the DSA) and a set of parameters enabling it to be used to verify the identity of the originator and integrity of the data.

The DSA includes signature generation and verification. Generation makes use of a private key to generate a digital signature. Verification of the signature makes use of a public key that corresponds to, but is not the same as, the private key used to generate the signature. Each user possesses a private and public key pair. Public keys are assumed to be known to all members of a group of users or to the public in general. Private keys must be known only by their creators. 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. The message digest is then signed. The digital signature is sent to the intended recipient along with the signed data (often called the message). The recipient 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 will be specified in a separate standard. Similar procedures may be used to generate and verify signatures for stored as well as transmitted data.

This standard is applicable to all federal departments and agencies for the protection of unclassified information not subject to section 2315 of Title 10 or section 3502(2) of Title 44, U.S. Code. This standard shall be used in designing and implementing public key-based signature systems operated by federal departments and agencies under contract. Private and commercial organizations are encouraged to adopt and use this standard.

The DSA authenticates the integrity of the signed data and the identity of the signer. 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.

Implementation and Qualifications

The DSA may be implemented in software, firmware

or hardware; only implementations of the DSA that are validated by NIST will be considered as complying with this standard. Implementations of the DSA in this standard may be covered by U.S. and foreign patents. This standard is effective six months after publication in the Federal Register announcing approval by the Secretary of Commerce.

In terms of qualifications, the security of a digital signature system is dependent on maintaining the secrecy of users' private keys. Therefore, users must 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.

Specifications for a DSS

When a message is transmitted from one party to another, the recipient may desire to know that the message has not been altered in transit. Furthermore, the recipient may wish to be certain of the origin of the message. 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 a third party that the message was, in fact, signed by the originator. Unlike their written counterparts, digital signatures also verify the integrity of messages. It is also desirable to be able to generate digital signatures for stored data and programs, allowing the integrity of the data and programs to be verified at any later time.

This section 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.

Use of the DSA Algorithm

A private and public key pair is used to generate and verify a digital signature. These keys are employed in conjunction with a hash function H, which is not specified in this standard. The holder of a private key can generate a signature for data, referred to as the message, m. A holder of the corresponding public key can verify the signature. Both signature generation and verification use H. An adversary, who does not know the private key of a user, cannot generate that user's signature for a message. Thus, signatures cannot be forged: an adversary cannot generate a correct signature for another person for any message. By using the appropriate public key, however, anyone can check that a given signature is valid.

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.

DSA Parameters

Let:

p = a prime modulus where $2^{511} < p < 2^{512}$

Decoding Cryptographic Terminology

Cryptography provides both privacy and authentication; namely, protection against someone listening in and protection against someone injecting a message.

A conventional cryptographic system uses a single secret key, known only to the legitimate sender and receiver. Privacy results because an eavesdropper, not knowing the key, cannot decipher the intercepted message to learn what is being said. Authentication results because an eavesdropper, not knowing the key, cannot encipher the message he/she wants to interject. (The receiver only accepts properly enciphered messages as authentic.)

Conventional cryptosystems are sometimes called *symmetric cryptosystems* because neither enciphering nor deciphering is possible without knowledge of the secret key.

In a public key system each user has one public and one private key. The public key is used to scramble the message and the secret key is used to unscramble, ensuring only the intended receiver can read the message. In practice, however, the public key system is used to send a preamble that specifies a secret key to be used in a conventional cryptosystem. The conventional system is used to send the entire message for many reasons, primarily because it is much faster than public keys systems (MB per sec. vs. KB per sec.).

When a public key system is used for authentication, the sender's secret key is used to sign a message and the public key is used to verify the signature. Therefore, no one except the sender can sign messages.

Public key systems are sometimes called *asymmetric cryptosystems* because one operation—enciphering or deciphering—is possible without knowledge of the secret key, while the other is impossible. Some public key systems, particularly the RSA system, provide both privacy and authentication automatically. Other systems, notably the Diffie-Hellman key exchange and ElGamal signature systems, provide protections for one or the other.

q = a prime divisor of $p - 1$ where $2^{159} < q < 2^{160}$.
$g = h^{(p-1)/q} \bmod p$, where h is any integer with $0 < h < p$ such that $h^{(p-1)/q} \bmod p > 1$.
x = an integer with $0 < x < q$.
$y = g^x \bmod p$.
m = the message to be signed and transmitted.
k = a random integer with $0 < k < q$.
H = a one-way hash function.

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. x and k must be secret. k must be changed for each signature. H is not specified in this standard. H however, must be chosen so that it is computationally infeasible to create a message that results in a given hash value and it must also be computationally infeasible to find any two different messages that result in the same hash value.

Signature Generation and Verification

To send a signed message m the user chooses a random k and computes

$$r = (g^k \bmod p) \bmod q$$
$$s = (k^{-1}(H(m) + xr)) \bmod q.$$

where k^{-1} is the multiplicative inverse of k, mod q; i.e., $(k^{-1}\, k) \bmod q = 1$ and $0 < k^{-1} < q$.

The values r and s constitute the signature of the message. These are transmitted along with the message m to the recipient.

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 recipient 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 sender. To verify the signature, the recipient first checks to see that $0 < r' < q$ and $0 < s' < q$; if either condition is violated the signature is rejected. If these two conditions are satisfied, the recipient computes

$$w = (s')^{-1} \bmod q$$
$$u1 = ((H(m'))w) \bmod q$$
$$u2 = ((r')w) \bmod q$$
$$v = (((g)^{u1}\, (y)^{u2}) \bmod p) \bmod q.$$

If $v = r'$, the signature is verified and the receiver 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', the message may have been modified, the message may have been incorrectly signed by the sender, or the message may have been signed by an impostor. The message should be considered invalid.

APPENDIX 1. A Proof that $v = r'$

The purpose of this appendix is to provide a rigorous proof that in the signature verification, we have $v = r'$ when $m' = m$, $r' = r$, and $s' = s$. The proof is given by the following Theorem; it is preceded by four lemmas.

LEMMA 1. *For any nonnegative integer t, if $g = h^{(p-1)/q} \bmod p$, then $g^t \bmod p = g^{t \bmod q} \bmod p$.*

Proof: By the Euler/Fermat theorem, since h is relatively prime to p, we have $h^{p-1} \bmod p = 1$. Hence for any nonnegative integer n,

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

Thus, for any nonnegative integers n and z we have

$$\begin{aligned} g^{nq+z} \bmod p &= (g^{nq}\, g^z) \bmod p \\ &= ((g^{nq} \bmod p)\, (g^z \bmod p)) \bmod p \\ &= g^z \bmod p. \end{aligned}$$

Any nonnegative integer t can be represented uniquely as $t = nq + z$ where n and z are nonnegative integers and $0 \le z < q$. Then

$$g^t \bmod p = g^z \bmod p.$$

Also, $z = t \bmod q$. The result follows. QED.

LEMMA 2. *For any nonnegative integers a and b, $g^{a \bmod q + b \bmod q} \bmod p = g^{(a+b) \bmod q} \bmod p$.*

Proof: By Lemma 1 we have

$$\begin{aligned} g^{a \bmod q + b \bmod q} \bmod p &= g^{(a \bmod q + b \bmod q) \bmod q} \bmod p \\ &= g^{(a+b) \bmod q} \bmod p. \end{aligned}$$

QED.

LEMMA 3. $y^{(rw) \bmod q} \bmod p = g^{(xrw) \bmod q} \bmod p$.

Proof: Since $y = g^x \bmod p$, using Lemma 1 we have

$$\begin{aligned} y^{(rw) \bmod q} \bmod p &= (g^x \bmod p)^{(rw) \bmod q} \bmod p \\ &= g^{x((rw) \bmod q)} \bmod p \\ &= g^{(x((rw) \bmod q)) \bmod q} \bmod p \\ &\quad \text{(by Lemma 1)} \\ &= g^{(xrw) \bmod q} \bmod p. \end{aligned}$$

QED.

LEMMA 4. *((H(m) + xr)w) mod q = k.*
Proof: We have

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

Since $(k\ k^{-1}) \bmod q = 1$,

$$\begin{aligned}(ks) \bmod q &= (k((k^{-1}(H(m) + xr)) \bmod q)) \bmod q \\ &= ((k(k^{-1}(H(m) + xr)))) \bmod q \\ &= (((k\ k^{-1}) \bmod q)\ ((H(m) + xr) \bmod q)) \bmod q \\ &= (H(m) + xr) \bmod q.\end{aligned}$$

Since $w = s^{-1} \bmod q$ we have $(ws) \bmod q = 1$, and thus

$$\begin{aligned}((H(m) + xr)w) \bmod q &= (((H(m) + xr) \bmod q)\ (w \bmod q)) \bmod q \\ &= (((ks) \bmod q)\ (w \bmod q)) \bmod q \\ &= (kws) \bmod q \\ &= ((k \bmod q)\ ((ws) \bmod q)) \bmod q \\ &= k \bmod q.\end{aligned}$$

Since $0 < k < q$ we have $k \bmod q = k$. QED.

THEOREM. *If $m' = m$, $r' = r$, and $s' = s$, then $v = r'$.*
Proof: Using Lemmas 2, 3 and 4 we find

$$\begin{aligned}v &= ((g^{u1}\ y^{u2}) \bmod p) \bmod q \\ &= ((g^{(H(m)w) \bmod q}\ y^{(rw) \bmod q}) \bmod p) \bmod q \\ &= ((g^{(H(m)w) \bmod q}\ g^{(xrw) \bmod q}) \bmod p) \bmod q \\ &\quad \text{(by Lemma 3)} \\ &= ((g^{(H(m)w) \bmod q + (xrw) \bmod q}) \bmod p) \bmod q \\ &= ((g^{((H(m)w) \bmod q + (xrw)} \bmod q) \bmod q) \\ &\quad \bmod p) \bmod q \\ &= ((g^{(H(m)w + xrw) \bmod q}) \bmod p) \bmod q \\ &\quad \text{(by Lemma 2)} \\ &= ((g^{((H(m) + xr)w) \bmod q}) \bmod p) \bmod q \\ &= (g^k \bmod p) \bmod q \text{ (by Lemma 4)} \\ &= r \\ &= r'.\end{aligned}$$

QED.

APPENDIX 2. Generation of Parameters for the DSA

This appendix includes suggestions for generating the parameters and performing the functions needed to implement the DSA. These algorithms require a random number generator and an efficient modular exponentiation algorithm. 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 M.O. Rabin's procedure based in part on ideas of Gary L. Miller.[1] If this algorithm is iterated n times, it will produce a false prime with probability no greater than $1/4^n$. Therefore, $n = 50$ should give an acceptable probability of error.
To test whether an integer is prime:

1. Set $i = 1$ and $n = 50$.
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 $w - 1$.
3. Generate a random integer b in the range $1 < b < w$.
4. Set $j = 0$ and $z = b^m \bmod w$.
5. If $j = 0$ and $z = 1$, or if $z = w - 1$, go to step 9.
6. If $j > 0$ and $z = 1$, go to step 8.
7. $j = j + 1$. If $j < a$, set $z = z^2 \bmod w$ and go to step 6.
8. w is not prime. Stop.
9. If $i < n$, set $i = i + 1$ and go to step 3. Otherwise, w is probably prime.

To generate a prime q where $2^{159} < q < 2^{160}$:

1. Set n = a random number between 2^{158} and $2^{159} - 1$, inclusive.
2. Set $t = 2n + 1$.
3. If $t < 2^{160}$, test whether t is prime. Otherwise, go to step 1.
4. If t is prime, set $q = t$. Otherwise, set $t = t + 2$ and go to step 3.

To generate a prime p where $2^{511} < p < 2^{512}$ and q divides $p - 1$:

1. Generate q as specified earlier.
2. Generate a random integer n, where n is between $(2^{511} - 1)/(2q)$ and $(2^{512} - 1)/(2q)$.
3. Set $t = 2nq + 1$.
4. Test whether t is prime.
5. If t is prime, set $p = t$. Otherwise, go to step 2.

To generate an element g of order $q \bmod p$:

1. Generate p and q as specified earlier.
2. Set h = a random number, where $1 < h < p - 1$.
3. Set $t = h^{(p-1)/q} \bmod p$.
4. If $t = 1$ then go to step 2. Otherwise, set $g = t$.

To generate integers x and k: Set x and k equal to random integers between 0 and q.
To generate y: Set $y = g^x \bmod p$.
To compute the signature (r,s) of a message m:

1. Set $t = g^k \bmod p$.
2. Set $r = t \bmod q$.

3. Set $h = H(m)$ where H is a hash function.
4. Calculate k^{-1} mod q as described.
5. Set $s = ((k^{-1})\ (h + xr))$ mod q.

To verify the signature (r,s) of a message m:

1. m, r, and s are received as m', r', and s', respectively.
2. If $r' <= 0$ or $r' >= q$ then reject the signature as invalid.
3. If $s' <= 0$ or $s' >= q$ then reject the signature as invalid.
4. Set $h = H(m')$.
5. Calculate $w = s'^{-1}$ mod q as described.
6. Set $u1 = (hw)$ mod q.
7. Set $u2 = (r'w)$ mod q.
8. Set $a = g^{u1}$ mod p.
9. Set $b = y^{u2}$ mod p.
10. Set $t = (ab)$ mod p.
11. Set $v = t$ mod q.
12. If $v = r'$, signature is verified. Otherwise, reject signature as invalid.

To compute the multiplicative inverse n^{-1} mod q for n with $0 < n < q$:

1. Set $i = q$, $h = n$, $c = 1$, $z = 0$, $v = 0$ and $d = 1$.
2. While $h > 0$ repeat steps 3 thru 5.
3. $t = i$ DIV h where DIV is defined as integer division.
4. Set $x = h$, $h = i - tx$ and $i = x$.
5. Set $x = d$, $d = v - tx$ and $v = x$.
6. $n^{-1} = v$ mod q.

APPENDIX 3. Random Number Generation for the DSA

Any implementation of the DSA requires the ability to generate random numbers. Random numbers are used to derive a user's private key and a user's per-message-random-secret-number. These random numbers are selected to be between 0 and the 160-bit prime q (as specified in the standard). The numbers can be generated by either a true noise hardware randomizer or via a pseudorandom function. Such a function would employ a user-generated and secret "seed" key to initialize the number generator. The generator then would produce a stream of bits or number that could be converted into the integers mod q.

APPENDIX 4. Modular Arithmetic for the DSA

One key to efficient implementation of the DSS is the development of a modular arithmetic package suited to the processor one intends to use. For purposes of this discussion, we will consider two types of processors: hardware and software and suggest some techniques that may allow for efficient implementation of the DSA in those systems.

For hardware implementations an excellent reference is "VLSI Implementation of Public Key Encryption Algorithms" by Orton, Roy, Scott, Peppard and Tavares from the *Proceedings of CRYPTO-86*. That paper and the references it includes will provide a good basis for anyone looking into hardware implementations of the DSA. It is also worth noting that several commercial firms have custom processors to do modular arithmetic on the market.

For software implementations, there are many ways to proceed and a rich literature containing various techniques for different computing environments. For instance, the algorithms one chooses for 32-bit processors may be substantially different from those appropriate for 8-bit processors. Likewise, one can make trade-offs on computation vs. memory in various algorithms. Good starting places for modular arithmetic algorithms in software are found in "A Cryptographic Library for the Motorola DSP56000" by Dusse and Kaliski from the *Proceedings of EUROCRYPT-90* and "Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor" by Barrett from the *Proceedings of CRYPTO-86*. These papers contain pseudocode for some of the more popular techniques used in modular arithmetic.

NIST's proposed Digital Signature Standard includes two sections—announcement information and specifications—both of which have been edited here for publication. To obtain a complete copy of the proposal, please write to Standards Processing Coordinator (ADP), National Institute of Standards and Technology, Technology Building, Room B-64, Gaithersburg, MD 20899.

Appendices 1, 2, 3, 4 are for informational purposes only and are not required to meet the standard.

---

[1]See "The Art of Computer Programming," by Knuth, D. Algorithm P, 379.