WIKIPEDIA
The Free Encyclopedia

Digital Signature Algorithm

The **Digital Signature Algorithm** (**DSA**) is a public-key cryptosystem and Federal Information Processing Standard for digital signatures, based on the mathematical concept of modular exponentiation and the discrete logarithm problem. In a digital signature system, there is a keypair involved, consisting of a private and a public key. In this system a signing entity that declared their public key can generate a signature using their private key, and a verifier can assert the source if it verifies the signature correctly using the declared public key. DSA is a variant of the Schnorr and ElGamal signature schemes.[1]:486

The National Institute of Standards and Technology (NIST) proposed DSA for use in their Digital Signature Standard (DSS) in 1991, and adopted it as FIPS 186 in 1994.[2] Five revisions to the initial specification have been released. The newest specification is: FIPS 186-5 (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf) from February 2023.[3] DSA is patented but NIST has made this patent available worldwide royalty-free. Specification FIPS 186-5 (https://doi.org/10.6028/NIST.FIPS.186-5) indicates DSA will no longer be approved for digital signature generation, but may be used to verify signatures generated prior to the implementation date of that standard.

Overview

The DSA works in the framework of public-key cryptosystems and is based on the algebraic properties of modular exponentiation, together with the discrete logarithm problem, which is considered to be computationally intractable. The algorithm uses a key pair consisting of a public key and a private key. The private key is used to generate a digital signature for a message, and such a signature can be verified by using the signer's corresponding public key. The digital signature provides message authentication (the receiver can verify the origin of the message), integrity (the receiver can verify that the message has not been modified since it was signed) and non-repudiation (the sender cannot falsely claim that they have not signed the message).

History

In 1982, the U.S government solicited proposals for a public key signature standard. In August 1991 the National Institute of Standards and Technology (NIST) proposed DSA for use in their Digital Signature Standard (DSS). Initially there was significant criticism, especially from software companies that had already invested effort in developing digital signature software based on the RSA cryptosystem.[1]:484 Nevertheless, NIST adopted DSA as a Federal standard (FIPS 186) in 1994. Five revisions to the initial specification have been released: FIPS 186–1 in 1998,[4] FIPS 186–2 in 2000,[5] FIPS 186–3 in 2009,[6] FIPS 186–4 in 2013,[3] and FIPS 186–5 in 2023.[7] Standard FIPS 186-5 forbids signing with DSA, while allowing verification of signatures generated prior to the implementation date of the standard as a document. It is to be replaced by newer signature schemes such as EdDSA.[8]

DSA is covered by U.S. patent 5,231,668 (https://patents.google.com/patent/US5231668), filed July 26, 1991 and now expired, and attributed to David W. Kravitz,[9] a former NSA employee. This patent was given to "The United States of America as represented by the Secretary of Commerce, Washington, D.C.", and NIST has made this patent available worldwide royalty-free.[10] Claus P. Schnorr claims that his U.S. patent 4,995,082 (https://patents.google.com/patent/US4995082) (also now expired) covered DSA; this claim is disputed.[11]

In 1993, Dave Banisar managed to get confirmation, via a FOIA request, that the DSA algorithm hasn't been designed by the NIST, but by the NSA.[12]

OpenSSH announced that DSA was going to be removed in 2025. The support was entirely dropped in version 10.0.[13][14]

Operation

The DSA algorithm involves four operations: key generation (which creates the key pair), key distribution, signing and signature verification.

1. Key generation

Key generation has two phases. The first phase is a choice of *algorithm parameters* which may be shared between different users of the system, while the second phase computes a single key pair for one user.

Parameter generation

- Choose an approved cryptographic hash function H with output length $|H|$ bits. In the original DSS, H was always SHA-1, but the stronger SHA-2 hash functions are approved for use in the current DSS.[3][15] If $|H|$ is greater than the modulus length N, only the leftmost N bits of the hash output are used.
- Choose a key length L. The original DSS constrained L to be a multiple of 64 between 512 and 1024 inclusive. NIST 800-57 recommends lengths of 2048 (or 3072) for keys with security lifetimes extending beyond 2010 (or 2030).[16]
- Choose the modulus length N such that $N < L$ and $N \leq |H|$. FIPS 186-4 specifies L and N to have one of the values: (1024, 160), (2048, 224), (2048, 256), or (3072, 256).[3]
- Choose an N-bit prime q.
- Choose an L-bit prime p such that $p - 1$ is a multiple of q.
- Choose an integer h randomly from $\{2 \ldots p-2\}$.
- Compute $g := h^{(p-1)/q} \mod p$. In the rare case that $g = 1$ try again with a different h. Commonly $h = 2$ is used. This modular exponentiation can be computed efficiently even if the values are large.

The algorithm parameters are (p, q, g). These may be shared between different users of the system.

Per-user keys

Given a set of parameters, the second phase computes the key pair for a single user:

- Choose an integer x randomly from $\{1 \ldots q-1\}$.
- Compute $y := g^x \mod p$.

x is the private key and y is the public key.

2. Key distribution

The signer should publish the public key y. That is, they should send the key to the receiver via a reliable, but not necessarily secret, mechanism. The signer should keep the private key x secret.

3. Signing

A message m is signed as follows:

- Choose an integer k randomly from $\{1 \ldots q-1\}$
- Compute $r := \left(g^k \bmod p\right) \bmod q$. In the unlikely case that $r = 0$, start again with a different random k.
- Compute $s := \left(k^{-1}\left(H(m) + xr\right)\right) \bmod q$. In the unlikely case that $s = 0$, start again with a different random k.

The signature is (r, s)

The calculation of k and r amounts to creating a new per-message key. The modular exponentiation in computing r is the most computationally expensive part of the signing operation, but it may be computed before the message is known. Calculating the modular inverse $k^{-1} \bmod q$ is the second most expensive part, and it may also be computed before the message is known. It may be computed using the extended Euclidean algorithm or using Fermat's little theorem as $k^{q-2} \bmod q$.

4. Signature Verification

One can verify that a signature (r, s) is a valid signature for a message m as follows:

- Verify that $0 < r < q$ and $0 < s < q$.
- Compute $w := s^{-1} \bmod q$.
- Compute $u_1 := H(m) \cdot w \bmod q$.
- Compute $u_2 := r \cdot w \bmod q$.
- Compute $v := \left(g^{u_1} y^{u_2} \bmod p\right) \bmod q$.
- The signature is valid if and only if $v = r$.

Correctness of the algorithm

The signature scheme is correct in the sense that the verifier will always accept genuine signatures. This can be shown as follows:

First, since $g = h^{(p-1)/q} \bmod p$, it follows that $g^q \equiv h^{p-1} \equiv 1 \mod p$ by Fermat's little theorem. Since $g > 0$ and q is prime, g must have order q.

The signer computes

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

Thus

$$\begin{aligned} k &\equiv H(m)s^{-1} + xrs^{-1} \\ &\equiv H(m)w + xrw \pmod{q} \end{aligned}$$

Since g has order q we have

$$\begin{aligned} g^k &\equiv g^{H(m)w} g^{xrw} \\ &\equiv g^{H(m)w} y^{rw} \\ &\equiv g^{u_1} y^{u_2} \pmod{p} \end{aligned}$$

Finally, the correctness of DSA follows from

$$\begin{aligned} r &= (g^k \bmod p) \bmod q \\ &= (g^{u_1} y^{u_2} \bmod p) \bmod q \\ &= v \end{aligned}$$

Sensitivity

With DSA, the entropy, secrecy, and uniqueness of the random signature value k are critical. It is so critical that violating any one of those three requirements can reveal the entire private key to an attacker.[17] Using the same value twice (even while keeping k secret), using a predictable value, or leaking even a few bits of k in each of several signatures, is enough to reveal the private key x.[18]

This issue affects both DSA and Elliptic Curve Digital Signature Algorithm (ECDSA) – in December 2010, the group *failoverflow* announced the recovery of the ECDSA private key used by Sony to sign software for the PlayStation 3 game console. The attack was made possible because Sony failed to generate a new random k for each signature.[19]

This issue can be prevented by deriving k deterministically from the private key and the message hash, as described by RFC 6979 (https://www.rfc-editor.org/rfc/rfc6979). This ensures that k is different for each $H(m)$ and unpredictable for attackers who do not know the private key x.

In addition, malicious implementations of DSA and ECDSA can be created where k is chosen in order to subliminally leak information via signatures. For example, an offline private key could be leaked from a perfect offline device that only released innocent-looking signatures.[20]

Implementations

Below is a list of cryptographic libraries that provide support for DSA:

- Botan
- Bouncy Castle
- cryptlib
- Crypto++
- libgcrypt
- Nettle
- OpenSSL
- wolfCrypt
- GnuTLS

See also

- Modular arithmetic
- RSA (cryptosystem)
- ECDSA

References

1. Schneier, Bruce (1996). *Applied Cryptography* (https://archive.org/details/Applied_Cryptography_2nd_ed._B._Schneier). Wiley. ISBN 0-471-11709-9.
2. "FIPS PUB 186: Digital Signature Standard (DSS), 1994-05-19" (https://web.archive.org/web/20131213131144/http://www.itl.nist.gov/fipspubs/fip186.htm). *qcsrc.nist.gov*. Archived from the original (http://www.itl.nist.gov/fipspubs/fip186.htm) on 2013-12-13.
3. "FIPS PUB 186-4: Digital Signature Standard (DSS), July 2013" (http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf) (PDF). *csrc.nist.gov*.
4. "FIPS PUB 186-1: Digital Signature Standard (DSS), 1998-12-15" (https://web.archive.org/web/20131226115544/http://csrc.nist.gov/publications/fips/fips1861.pdf) (PDF). *csrc.nist.gov*. Archived from the original (http://csrc.nist.gov/publications/fips/fips1861.pdf) (PDF) on 2013-12-26.
5. "FIPS PUB 186-2: Digital Signature Standard (DSS), 2000-01-27" (http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf) (PDF). *csrc.nist.gov*.
6. "FIPS PUB 186-3: Digital Signature Standard (DSS), June 2009" (http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf) (PDF). *csrc.nist.gov*.
7. "FIPS PUB 186-5: Digital Signature Standard (DSS), February 2023" (https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf) (PDF). *csrc.nist.gov*.
8. "Digital Signature Standard (DSS)" (https://csrc.nist.gov/publications/detail/fips/186/5/draft). U.S. Department of Commerce. 31 October 2019. Retrieved 21 July 2020.
9. Dr. David W. Kravitz (http://www.certicom.com/index.php/dr-david-kravitz) Archived (https://web.archive.org/web/20130109092551/http://www.certicom.com/index.php/dr-david-kravitz) January 9, 2013, at the Wayback Machine
10. Werner Koch. "DSA and patents" (https://lists.gnupg.org/pipermail/gnupg-devel/1997-December/014123.html)

11. "1994 Annual Report of CSSPAB" (https://web.archive.org/web/20090826042831/http://csrc.nist.gov/groups/SMA/ispab/documents/94-rpt.txt). 26 August 2009. Archived from the original (http://csrc.nist.gov/groups/SMA/ispab/documents/94-rpt.txt) on 26 August 2009.
12. Neumann, Peter G. (2020-02-29). "The RISKS Digest Volume 14 Issue 59" (https://web.archive.org/web/20200229145033/https://catless.ncl.ac.uk/Risks/14/59). Archived from the original on 2020-02-29. Retrieved 2023-10-03.
13. "OpenSSH announces DSA-removal timeline [LWN.net]" (https://lwn.net/Articles/958048/). *lwn.net*. Retrieved 11 January 2024.
14. "OpenSSH version 10.0. release notes" (https://www.openssh.com/txt/release-10.0). Retrieved 21 April 2025.
15. "FIPS PUB 180-4: Secure Hash Standard (SHS), March 2012" (http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf) (PDF). *csrc.nist.gov*.
16. "NIST Special Publication 800-57" (https://web.archive.org/web/20140606050814/http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf) (PDF). *csrc.nist.gov*. Archived from the original (http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf) (PDF) on 2014-06-06.
17. "The Debian PGP disaster that almost was" (http://rdist.root.org/2009/05/17/the-debian-pgp-disaster-that-almost-was/). *root labs rdist*. 18 May 2009.
18. DSA k-value Requirements (https://rdist.root.org/2010/11/19/dsa-requirements-for-random-k-value/)
19. Bendel, Mike (2010-12-29). "Hackers Describe PS3 Security As Epic Fail, Gain Unrestricted Access" (http://exophase.com/20540/hackers-describe-ps3-security-as-epic-fail-gain-unrestricted-access/). Exophase.com. Retrieved 2011-01-05.
20. Verbücheln, Stephan (2 January 2015). "How Perfect Offline Wallets Can Still Leak Bitcoin Private Keys". arXiv:1501.00447 (https://arxiv.org/abs/1501.00447) [cs.CR (https://arxiv.org/archive/cs.CR)].

External links

- FIPS PUB 186-4: Digital Signature Standard (DSS) (http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf), the fourth (and current) revision of the official DSA specification.
- Recommendation for Key Management -- Part 1: general (https://web.archive.org/web/20140606050814/http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf), NIST Special Publication 800-57, p. 62–63