Elliptic Curve Digital Signature Algorithm

From Wikipedia, the free encyclopedia

[Jump to navigation](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#mw-head)[Jump to search](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#searchInput)

In [cryptography](https://en.wikipedia.org/wiki/Cryptography), the **Elliptic Curve Digital Signature Algorithm** (**ECDSA**) offers a variant of the [Digital Signature Algorithm](https://en.wikipedia.org/wiki/Digital_Signature_Algorithm) (DSA) which uses [elliptic curve cryptography](https://en.wikipedia.org/wiki/Elliptic_curve_cryptography).



**Contents**

* [1Key and signature-size](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Key_and_signature-size)
* [2Signature generation algorithm](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Signature_generation_algorithm)
* [3Signature verification algorithm](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Signature_verification_algorithm)
  + [3.1Correctness of the algorithm](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Correctness_of_the_algorithm)
* [4Public key recovery](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Public_key_recovery)
  + [4.1Correctness of the recovery algorithm](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Correctness_of_the_recovery_algorithm)
* [5Security](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Security)
  + [5.1Concerns](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Concerns)
* [6Implementations](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Implementations)
* [7Example usage](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Example_usage)
* [8See also](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#See_also)
* [9References](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#References)
* [10Further reading](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Further_reading)
* [11External links](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#External_links)

Key and signature-size[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=1)]

As with elliptic-curve cryptography in general, the bit [size](https://en.wikipedia.org/wiki/Key_size) of the [public key](https://en.wikipedia.org/wiki/Public_key) believed to be needed for ECDSA is about twice the size of the [security level](https://en.wikipedia.org/wiki/Security_level), in bits. For example, at a security level of 80 bits (meaning an attacker requires a maximum of about {\displaystyle 2^{80}} operations to find the private key) the size of an ECDSA public key would be 160 bits, whereas the size of a DSA public key is at least 1024 bits. On the other hand, the signature size is the same for both DSA and ECDSA: approximately {\displaystyle 4t} bits, where {\displaystyle t} is the security level measured in bits, that is, about 320 bits for a security level of 80 bits.

Signature generation algorithm[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=2)]

Suppose [Alice](https://en.wikipedia.org/wiki/Alice_and_Bob) wants to send a signed message to [Bob](https://en.wikipedia.org/wiki/Alice_and_Bob). Initially, they must agree on the curve parameters {\displaystyle ({\textrm {CURVE}},G,n)}. In addition to the field and equation of the curve, we need {\displaystyle G}, a base point of prime order on the curve; {\displaystyle n} is the multiplicative order of the point {\displaystyle G}.

|  |  |
| --- | --- |
| **Parameter** |  |
| CURVE | the elliptic curve field and equation used |
| *G* | elliptic curve base point, a point on the curve that generates a [subgroup of large prime order n](https://en.wikipedia.org/wiki/Elliptic-curve_cryptography#Domain_parameters) |
| *n* | integer order of *G*, means that {\displaystyle n\times G=O}, where {\displaystyle O} is the identity element. |
| {\displaystyle d\_{A}} | the private key (randomly selected) |
| {\displaystyle Q\_{A}} | the public key (calculated by elliptic curve) |
| *m* | the message to send |

The order {\displaystyle n} of the base point {\displaystyle G} **must be prime**. Indeed, we assume that every nonzero element of the ring {\displaystyle \mathbb {Z} /n\mathbb {Z} } is invertible, so that {\displaystyle \mathbb {Z} /n\mathbb {Z} } must be a field. It implies that {\displaystyle n} must be prime (cf. [Bézout's identity](https://en.wikipedia.org/wiki/B%C3%A9zout%27s_identity)).

Alice creates a key pair, consisting of a private key integer {\displaystyle d\_{A}}, randomly selected in the interval {\displaystyle [1,n-1]}; and a public key curve point {\displaystyle Q\_{A}=d\_{A}\times G}. We use {\displaystyle \times } to denote [elliptic curve point multiplication by a scalar](https://en.wikipedia.org/wiki/Elliptic_curve_point_multiplication).

For Alice to sign a message {\displaystyle m}, she follows these steps:

1. Calculate {\displaystyle e={\textrm {HASH}}(m)}. (Here HASH is a [cryptographic hash function](https://en.wikipedia.org/wiki/Cryptographic_hash_function), such as [SHA-2](https://en.wikipedia.org/wiki/SHA-2), with the output converted to an integer.)
2. Let {\displaystyle z} be the {\displaystyle L\_{n}} leftmost bits of {\displaystyle e}, where {\displaystyle L\_{n}} is the bit length of the group order {\displaystyle n}. (Note that {\displaystyle z} can be *greater* than {\displaystyle n} but not *longer*.[[1]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-1))
3. Select a **cryptographically secure random** integer {\displaystyle k} from {\displaystyle [1,n-1]}.
4. Calculate the curve point {\displaystyle (x\_{1},y\_{1})=k\times G}.
5. Calculate {\displaystyle r=x\_{1}\,{\bmod {\,}}n}. If {\displaystyle r=0}, go back to step 3.
6. Calculate {\displaystyle s=k^{-1}(z+rd\_{A})\,{\bmod {\,}}n}. If {\displaystyle s=0}, go back to step 3.
7. The signature is the pair {\displaystyle (r,s)}. (And {\displaystyle (r,-s\,{\bmod {\,}}n)} is also a valid signature.)

As the standard notes, it is not only required for {\displaystyle k} to be secret, but it is also crucial to select different {\displaystyle k} for different signatures, otherwise the equation in step 6 can be solved for {\displaystyle d\_{A}}, the private key: given two signatures {\displaystyle (r,s)} and {\displaystyle (r,s')}, employing the same unknown {\displaystyle k} for different known messages {\displaystyle m} and {\displaystyle m'}, an attacker can calculate {\displaystyle z} and {\displaystyle z'}, and since {\displaystyle s-s'=k^{-1}(z-z')} (all operations in this paragraph are done modulo {\displaystyle n}) the attacker can find {\displaystyle k={\frac {z-z'}{s-s'}}}. Since {\displaystyle s=k^{-1}(z+rd\_{A})}, the attacker can now calculate the private key {\displaystyle d\_{A}={\frac {sk-z}{r}}}.

This implementation failure was used, for example, to extract the signing key used for the [PlayStation 3](https://en.wikipedia.org/wiki/PlayStation_3) gaming-console.[[2]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-2)

Another way ECDSA signature may leak private keys is when {\displaystyle k} is generated by a faulty [random number generator](https://en.wikipedia.org/wiki/Random_number_generator). Such a failure in random number generation caused users of Android Bitcoin Wallet to lose their funds in August 2013.[[3]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-3)

To ensure that {\displaystyle k} is unique for each message, one may bypass random number generation completely and generate deterministic signatures by deriving {\displaystyle k} from both the message and the private key.[[4]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-4)

Signature verification algorithm[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=3)]

For Bob to authenticate Alice's signature, he must have a copy of her public-key curve point {\displaystyle Q\_{A}}. Bob can verify {\displaystyle Q\_{A}} is a valid curve point as follows:

1. Check that {\displaystyle Q\_{A}} is not equal to the identity element {\displaystyle O}, and its coordinates are otherwise valid
2. Check that {\displaystyle Q\_{A}} lies on the curve
3. Check that {\displaystyle n\times Q\_{A}=O}

After that, Bob follows these steps:

1. Verify that {\displaystyle r} and {\displaystyle s} are integers in {\displaystyle [1,n-1]}. If not, the signature is invalid.
2. Calculate {\displaystyle e={\textrm {HASH}}(m)}, where HASH is the same function used in the signature generation.
3. Let {\displaystyle z} be the {\displaystyle L\_{n}} leftmost bits of {\displaystyle e}.
4. Calculate {\displaystyle u\_{1}=zs^{-1}\,{\bmod {\,}}n} and {\displaystyle u\_{2}=rs^{-1}\,{\bmod {\,}}n}.
5. Calculate the curve point {\displaystyle (x\_{1},y\_{1})=u\_{1}\times G+u\_{2}\times Q\_{A}}. If {\displaystyle (x\_{1},y\_{1})=O} then the signature is invalid.
6. The signature is valid if {\displaystyle r\equiv x\_{1}{\pmod {n}}}, invalid otherwise.

Note that an efficient implementation would compute inverse {\displaystyle s^{-1}\,{\bmod {\,}}n} only once. Also, using Shamir's trick, a sum of two scalar multiplications {\displaystyle u\_{1}\times G+u\_{2}\times Q\_{A}} can be calculated faster than two scalar multiplications done independently.[[5]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-5)

**Correctness of the algorithm**[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=4)]

It is not immediately obvious why verification even functions correctly. To see why, denote as {\displaystyle C} the curve point computed in step 5 of verification,

{\displaystyle C=u\_{1}\times G+u\_{2}\times Q\_{A}}

From the definition of the public key as {\displaystyle Q\_{A}=d\_{A}\times G},

{\displaystyle C=u\_{1}\times G+u\_{2}d\_{A}\times G}

Because elliptic curve scalar multiplication distributes over addition,

{\displaystyle C=(u\_{1}+u\_{2}d\_{A})\times G}

Expanding the definition of {\displaystyle u\_{1}} and {\displaystyle u\_{2}} from verification step 4,

{\displaystyle C=(zs^{-1}+rd\_{A}s^{-1})\times G}

Collecting the common term {\displaystyle s^{-1}},

{\displaystyle C=(z+rd\_{A})s^{-1}\times G}

Expanding the definition of {\displaystyle s} from signature step 6,

{\displaystyle C=(z+rd\_{A})(z+rd\_{A})^{-1}(k^{-1})^{-1}\times G}

Since the inverse of an inverse is the original element, and the product of an element's inverse and the element is the identity, we are left with

{\displaystyle C=k\times G}

From the definition of {\displaystyle r}, this is verification step 6.

This shows only that a correctly signed message will verify correctly; many other properties[[*which?*](https://en.wikipedia.org/wiki/Wikipedia:Avoid_weasel_words)] are required for a secure signature algorithm.

Public key recovery[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=5)]

Given a message {\displaystyle m} and Alice's signature {\displaystyle r,s} on that message, Bob can (potentially) recover Alice's public key:[[6]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-6)

1. Verify that {\displaystyle r} and {\displaystyle s} are integers in {\displaystyle [1,n-1]}. If not, the signature is invalid.
2. Calculate a curve point {\displaystyle R=(x\_{1},y\_{1})} where {\displaystyle x\_{1}} is one of {\displaystyle r}, {\displaystyle r+n}, {\displaystyle r+2n}, etc. (provided {\displaystyle x\_{1}} is not too large for a field element) and {\displaystyle y\_{1}} is a value such that the curve equation is satisfied. Note that there may be several curve points satisfying these conditions, and each different {\displaystyle R} value results in a distinct recovered key.
3. Calculate {\displaystyle e={\textrm {HASH}}(m)}, where HASH is the same function used in the signature generation.
4. Let {\displaystyle z} be the {\displaystyle L\_{n}} leftmost bits of {\displaystyle e}.
5. Calculate {\displaystyle u\_{1}=-zr^{-1}\,{\bmod {\,}}n} and {\displaystyle u\_{2}=sr^{-1}\,{\bmod {\,}}n}.
6. Calculate the curve point {\displaystyle Q\_{A}=(x\_{A},y\_{A})=u\_{1}\times G+u\_{2}\times R}.
7. The signature is valid if {\displaystyle Q\_{A}}, matches Alice's public key.
8. The signature is invalid if all the possible {\displaystyle R} points have been tried and none match Alice's public key.

Note that an invalid signature, or a signature from a different message, will result in the recovery of an incorrect public key. The recovery algorithm can only be used to check validity of a signature if the signer's public key (or its hash) is known beforehand.

**Correctness of the recovery algorithm**[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=6)]

Start with the definition of {\displaystyle Q\_{A}} from recovery step 6,

{\displaystyle Q\_{A}=(x\_{A},y\_{A})=u\_{1}\times G+u\_{2}\times R}

From the definition {\displaystyle R=(x\_{1},y\_{1})=k\times G} from signing step 4,

{\displaystyle Q\_{A}=u\_{1}\times G+u\_{2}k\times G}

Because elliptic curve scalar multiplication distributes over addition,

{\displaystyle Q\_{A}=(u\_{1}+u\_{2}k)\times G}

Expanding the definition of {\displaystyle u\_{1}} and {\displaystyle u\_{2}} from recovery step 5,

{\displaystyle Q\_{A}=(-zr^{-1}+skr^{-1})\times G}

Expanding the definition of {\displaystyle s} from signature step 6,

{\displaystyle Q\_{A}=(-zr^{-1}+k^{-1}(z+rd\_{A})kr^{-1})\times G}

Since the product of an element's inverse and the element is the identity, we are left with

{\displaystyle Q\_{A}=(-zr^{-1}+(zr^{-1}+d\_{A}))\times G}

The first and second terms cancel each other out,

{\displaystyle Q\_{A}=d\_{A}\times G}

From the definition of {\displaystyle Q\_{A}=d\_{A}\times G}, this is Alice's public key.

This shows that a correctly signed message will recover the correct public key, provided additional information was shared to uniquely calculate curve point {\displaystyle R=(x\_{1},y\_{1})} from signature value {\displaystyle r}.

Security[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=7)]

In December 2010, a group calling itself *fail0verflow* announced recovery of the ECDSA private key used by [Sony](https://en.wikipedia.org/wiki/Sony) to sign software for the [PlayStation 3](https://en.wikipedia.org/wiki/PlayStation_3) game console. However, this attack only worked because Sony did not properly implement the algorithm, because {\displaystyle k} was static instead of random. As pointed out in the [Signature generation algorithm](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#Signature_generation_algorithm) section above, this makes {\displaystyle d\_{A}} solvable and the entire algorithm useless.[[7]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-7)

On March 29, 2011, two researchers published an [IACR](https://en.wikipedia.org/wiki/International_Association_for_Cryptologic_Research) paper[[8]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-8) demonstrating that it is possible to retrieve a TLS private key of a server using [OpenSSL](https://en.wikipedia.org/wiki/OpenSSL) that authenticates with Elliptic Curves DSA over a binary [field](https://en.wikipedia.org/wiki/Field_(mathematics)) via a [timing attack](https://en.wikipedia.org/wiki/Timing_attack).[[9]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-9) The vulnerability was fixed in OpenSSL 1.0.0e.[[10]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-10)

In August 2013, it was revealed that bugs in some implementations of the [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) class [SecureRandom](https://docs.oracle.com/javase/10/docs/api/java/security/SecureRandom.html) sometimes generated collisions in the {\displaystyle k} value. This allowed hackers to recover private keys giving them the same control over bitcoin transactions as legitimate keys' owners had, using the same exploit that was used to reveal the PS3 signing key on some [Android](https://en.wikipedia.org/wiki/Android_(operating_system)) app implementations, which use Java and rely on ECDSA to authenticate transactions.[[11]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-11)

This issue can be prevented by an unpredictible generation of {\displaystyle k}, e.g., a deterministic procedure as described by [RFC 6979](https://tools.ietf.org/html/rfc6979).

**Concerns**[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=8)]

There exist two sorts of concerns with ECDSA:

1. *Political concerns*: the trustworthiness of [NIST](https://en.wikipedia.org/wiki/National_Institute_of_Standards_and_Technology)-produced curves being questioned after revelations that the [NSA](https://en.wikipedia.org/wiki/National_Security_Agency) willingly inserts [backdoors](https://en.wikipedia.org/wiki/Backdoor_(computing)) into software, hardware components and published standards were made; well-known cryptographers[[12]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-12) have expressed[[13]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-13)[[14]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-14) doubts about how the NIST curves were designed, and voluntary tainting has already been proved in the past.[[15]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-15)[[16]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-16). Nevertheless, a proof, that the named NIST curves exploit a rare weakness, is missing yet.
2. *Technical concerns*: the difficulty of properly implementing the standard,[[17]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-17) its slowness, and design flaws which reduce security in insufficiently defensive implementations of the [Dual\_EC\_DRBG](https://en.wikipedia.org/wiki/Dual_EC_DRBG) random number generator.[[18]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-18)

Both of those concerns are summarized in *libssh*[*curve25519*](https://en.wikipedia.org/wiki/Curve25519)*introduction*.[[19]](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_note-19)

Implementations[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=9)]

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

* [Botan](https://en.wikipedia.org/wiki/Botan_(programming_library))
* [Bouncy Castle](https://en.wikipedia.org/wiki/Bouncy_Castle_(cryptography))
* [cryptlib](https://en.wikipedia.org/wiki/Cryptlib)
* [Crypto++](https://en.wikipedia.org/wiki/Crypto%2B%2B)
* [libgcrypt](https://en.wikipedia.org/wiki/Libgcrypt)
* [GnuTLS](https://en.wikipedia.org/wiki/GnuTLS)
* [OpenSSL](https://en.wikipedia.org/wiki/OpenSSL)
* [wolfCrypt](https://en.wikipedia.org/wiki/WolfCrypt)
* [LibreSSL](https://en.wikipedia.org/wiki/LibreSSL)
* [mbed TLS](https://en.wikipedia.org/wiki/Mbed_TLS)

Example usage[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=10)]

Wikipedia.org uses ECDSA in a TLS ciphersuite to authenticate itself to web browsers, which the following abbreviated transcript shows.

**$** date

Wed Mar 4 10:24:52 EST 2020

**$** openssl s\_client -connect wikipedia.org:443 *# output below has DELETIONS for brevity*

CONNECTED(00000003)

depth=2 O = Digital Signature Trust Co., CN = DST Root CA X3

verify return:1

depth=1 C = US, O = Let's Encrypt, CN = Let's Encrypt Authority X3

verify return:1

depth=0 CN = \*.wikipedia.org

verify return:1

---

Certificate chain

0 s:/CN=\*.wikipedia.org

i:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3

1 s:/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3

i:/O=Digital Signature Trust Co./CN=DST Root CA X3

---

Server certificate

-----BEGIN CERTIFICATE-----

MIIHOTCCBiGgAwIBAgISA4srJU6bpT7xpINN6bbGO2/mMA0GCSqGSIb3DQEBCwUA

... many lines DELETED ....

kTOXMoKzBkJCU8sCdeziusJtNvWXW6p8Z3UpuTw=

-----END CERTIFICATE-----

subject=/CN=\*.wikipedia.org

issuer=/C=US/O=Let's Encrypt/CN=Let's Encrypt Authority X3

---

No client certificate CA names sent

Peer signing digest: SHA256

Server Temp Key: ECDH, P-256, 256 bits

---

SSL handshake has read 3353 bytes and written 431 bytes

---

New, TLSv1/SSLv3, Cipher is ECDHE-ECDSA-AES256-GCM-SHA384

Server public key is 256 bit

Secure Renegotiation IS supported

Compression: NONE

Expansion: NONE

No ALPN negotiated

SSL-Session:

Protocol : TLSv1.2

Cipher : ECDHE-ECDSA-AES256-GCM-SHA384

Session-ID: ... DELETED ...

Session-ID-ctx:

Master-Key: ... DELETED ...

Key-Arg : None

PSK identity: None

PSK identity hint: None

SRP username: None

Start Time: 1583335210

Timeout : 300 (sec)

Verify return code: 0 (ok)

---

DONE

See also[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=11)]

* [EdDSA](https://en.wikipedia.org/wiki/EdDSA)
* [RSA (cryptosystem)](https://en.wikipedia.org/wiki/RSA_(cryptosystem))

References[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=12)]

* 1. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-1) [NIST FIPS 186-4, July 2013, pp. 19 and 26](http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf)
  2. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-2) [Console Hacking 2010 - PS3 Epic Fail](https://events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf) [Archived](https://web.archive.org/web/20141215140847/http:/events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf) December 15, 2014, at the [Wayback Machine](https://en.wikipedia.org/wiki/Wayback_Machine), page 123–128
  3. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-3) [*"Android Security Vulnerability"*](https://bitcoin.org/en/alert/2013-08-11-android)*. Retrieved February 24, 2015.*
  4. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-4) [*"RFC 6979 - Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)"*](http://tools.ietf.org/html/rfc6979)*. Retrieved February 24, 2015.*
  5. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-5) [*"The Double-Base Number System in Elliptic Curve Cryptography"*](http://www.lirmm.fr/~imbert/talks/laurent_Asilomar_08.pdf)*(PDF). Retrieved April 22, 2014.*
  6. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-6) Daniel R. L. Brown [SECG](https://en.wikipedia.org/wiki/SECG) SEC 1: Elliptic Curve Cryptography (Version 2.0) <https://www.secg.org/sec1-v2.pdf>
  7. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-7) *Bendel, Mike (December 29, 2010).*[*"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 January 5, 2011.*
  8. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-8) [*"Cryptology ePrint Archive: Report 2011/232"*](http://eprint.iacr.org/2011/232)*. Retrieved February 24, 2015.*
  9. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-9) [*"Vulnerability Note VU#536044 - OpenSSL leaks ECDSA private key through a remote timing attack"*](https://www.kb.cert.org/vuls/id/536044)*. www.kb.cert.org.*
  10. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-10) [*"ChangeLog"*](https://www.openssl.org/news/changelog.html)*. OpenSSL Project. Retrieved April 22, 2014.*
  11. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-11) [*"Android bug batters Bitcoin wallets"*](https://www.theregister.co.uk/2013/08/12/android_bug_batters_bitcoin_wallets/)*. The Register. August 12, 2013.*
  12. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-12) *Schneier, Bruce (September 5, 2013).*[*"The NSA Is Breaking Most Encryption on the Internet"*](https://www.schneier.com/blog/archives/2013/09/the_nsa_is_brea.html#c1675929)*. Schneier on Security.*
  13. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-13) [*"SafeCurves: choosing safe curves for elliptic-curve cryptography"*](http://safecurves.cr.yp.to/rigid.html)*. October 25, 2013.*
  14. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-14) *Bernstein, Daniel J.;*[*Lange, Tanja*](https://en.wikipedia.org/wiki/Tanja_Lange)*(May 31, 2013).*[*"Security dangers of the NIST curves"*](https://www.hyperelliptic.org/tanja/vortraege/20130531.pdf)*(PDF).*
  15. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-15) *Schneier, Bruce (November 15, 2007).*[*"The Strange Story of Dual\_EC\_DRBG"*](https://www.schneier.com/blog/archives/2007/11/the_strange_sto.html)*. Schneier on Security.*
  16. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-16) *Greenemeier, Larry (September 18, 2013).*[*"NSA Efforts to Evade Encryption Technology Damaged U.S. Cryptography Standard"*](http://www.scientificamerican.com/article/nsa-nist-encryption-scandal/)*. Scientific American.*
  17. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-17) *Bernstein, Daniel J. (March 23, 2014).*[*"How to design an elliptic-curve signature system"*](http://blog.cr.yp.to/20140323-ecdsa.html)*. The cr.yp.to blog.*
  18. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-18) [*"New key type (ed25519) and private key format"*](http://www.gossamer-threads.com/lists/openssh/dev/57162#57162)*.*
  19. [**^**](https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm#cite_ref-19) [*"curve25519-sha256@libssh.org.txt\doc - projects/libssh.git"*](https://git.libssh.org/projects/libssh.git/tree/doc/curve25519-sha256@libssh.org.txt#n4)*. libssh shared repository.*

Further reading[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=13)]

* Accredited Standards Committee [X9](http://www.x9.org/), *American National Standard X9.62-2005, Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*, November 16, 2005.
* Certicom Research, [*Standards for efficient cryptography, SEC 1: Elliptic Curve Cryptography*](https://www.secg.org/sec1-v2.pdf), Version 2.0, May 21, 2009.
* López, J. and Dahab, R. [*An Overview of Elliptic Curve Cryptography*](http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.2771), Technical Report IC-00-10, State University of Campinas, 2000.
* [Daniel J. Bernstein](https://en.wikipedia.org/wiki/Daniel_J._Bernstein), [Pippenger's exponentiation algorithm](http://cr.yp.to/papers/pippenger.pdf), 2002.
* Daniel R. L. Brown, *Generic Groups, Collision Resistance, and ECDSA*, Designs, Codes and Cryptography, **35**, 119–152, 2005. [ePrint version](http://eprint.iacr.org/2002/026)
* Ian F. Blake, Gadiel Seroussi, and [Nigel Smart](https://en.wikipedia.org/wiki/Nigel_Smart_(cryptographer)), editors, *Advances in Elliptic Curve Cryptography*, London Mathematical Society Lecture Note Series 317, Cambridge University Press, 2005.
* *Hankerson, D.;*[*Vanstone, S.*](https://en.wikipedia.org/wiki/Scott_Vanstone)*;*[*Menezes, A.*](https://en.wikipedia.org/wiki/Alfred_Menezes)*(2004). Guide to Elliptic Curve Cryptography. Springer Professional Computing. New York:*[*Springer*](https://en.wikipedia.org/wiki/Springer_Science%2BBusiness_Media)*.*[*doi*](https://en.wikipedia.org/wiki/Doi_(identifier))*:*[*10.1007/b97644*](https://doi.org/10.1007%2Fb97644)*.*[*ISBN*](https://en.wikipedia.org/wiki/ISBN_(identifier))[*0-387-95273-X*](https://en.wikipedia.org/wiki/Special:BookSources/0-387-95273-X)*.*

External links[[edit](https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&action=edit&section=14)]

* [Digital Signature Standard; includes info on ECDSA](http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf)
* [The Elliptic Curve Digital Signature Algorithm (ECDSA); provides an in-depth guide on ECDSA](https://web.archive.org/web/20160304101319/http:/cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf). [Wayback link](https://web.archive.org/web/20100627011540/http:/cs.ucsb.edu/~koc/ccs130h/notes/ecdsa-cert.pdf)