Wikipedia
The Free Encyclopedia

Elliptic Curve Digital Signature Algorithm

In cryptography, the **Elliptic Curve Digital Signature Algorithm** (**ECDSA**) offers a variant of the Digital Signature Algorithm (DSA) which uses elliptic-curve cryptography.

Key and signature-size

As with elliptic-curve cryptography in general, the bit size of the private key believed to be needed for ECDSA is about twice the size of the security level, in bits.[1] For example, at a security level of 80 bits—meaning an attacker requires a maximum of about 2^{80} operations to find the private key—the size of an ECDSA private key would be 160 bits. On the other hand, the signature size is the same for both DSA and ECDSA: approximately $4t$ bits, where t is the exponent in the formula 2^t, that is, about 320 bits for a security level of 80 bits, which is equivalent to 2^{80} operations.

Signature generation algorithm

Suppose Alice wants to send a signed message to Bob. Initially, they must agree on the curve parameters $(\textrm{CURVE}, G, n)$. In addition to the field and equation of the curve, we need G, a base point of prime order on the curve; n is the multiplicative order of the point 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
n	integer order of *G*, means that $n \times G = O$, where O is the identity element.
d_A	the private key (randomly selected)
Q_A	the public key $d_A \times G$ (calculated by elliptic curve)
m	the message to send

The order n of the base point G **must be prime**. Indeed, we assume that every nonzero element of the ring $\mathbb{Z}/n\mathbb{Z}$ is invertible, so that $\mathbb{Z}/n\mathbb{Z}$ must be a field. It implies that n must be prime (cf. Bézout's identity).

Alice creates a key pair, consisting of a private key integer d_A, randomly selected in the interval $[1, n-1]$; and a public key curve point $Q_A = d_A \times G$. We use $\times$ to denote elliptic curve point multiplication by a scalar.

For Alice to sign a message m, she follows these steps:

1. Calculate $e = \textrm{HASH}(m)$. (Here HASH is a cryptographic hash function, such as SHA-2, with the output converted to an integer.)
2. Let z be the L_n leftmost bits of e, where L_n is the bit length of the group order n. (Note that z can be *greater* than n but not *longer*.[2])
3. Select a **cryptographically secure random** integer k from $[1, n-1]$.

4. Calculate the curve point $(x_1, y_1) = k \times G$.
5. Calculate $r = x_1 \bmod n$. If $r = 0$, go back to step 3.
6. Calculate $s = k^{-1}(z + rd_A) \bmod n$. If $s = 0$, go back to step 3.
7. The signature is the pair (r, s). (And $(r, -s \bmod n)$ is also a valid signature.)

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

This implementation failure was used, for example, to extract the signing key used for the PlayStation 3 gaming-console.[3]

Another way ECDSA signature may leak private keys is when k is generated by a faulty random number generator. Such a failure in random number generation caused users of Android Bitcoin Wallet to lose their funds in August 2013.[4]

To ensure that k is unique for each message, one may bypass random number generation completely and generate deterministic signatures by deriving k from both the message and the private key.[5]

Signature verification algorithm

For Bob to authenticate Alice's signature, he must have a copy of her public-key curve point Q_A. Bob can verify Q_A is a valid curve point as follows:

1. Check that Q_A is not equal to the identity element O, and its coordinates are otherwise valid.
2. Check that Q_A lies on the curve.
3. Check that $n \times Q_A = O$.

After that, Bob follows these steps:

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
3. Let z be the L_n leftmost bits of e.
4. Calculate $u_1 = zs^{-1} \bmod n$ and $u_2 = rs^{-1} \bmod n$.
5. Calculate the curve point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$. If $(x_1, y_1) = O$ then the signature is invalid.
6. The signature is valid if $r \equiv x_1 \pmod{n}$, invalid otherwise.

Note that an efficient implementation would compute inverse $s^{-1} \bmod n$ only once. Also, using Shamir's trick, a sum of two scalar multiplications $u_1 \times G + u_2 \times Q_A$ can be calculated faster than two scalar multiplications done independently.[6]

Correctness of the algorithm

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

$$C = u_1 \times G + u_2 \times Q_A$$

From the definition of the public key as $Q_A = d_A \times G$,

$$C = u_1 \times G + u_2 d_A \times G$$

Because elliptic curve scalar multiplication distributes over addition,

$$C = (u_1 + u_2 d_A) \times G$$

Expanding the definition of u_1 and u_2 from verification step 4,

$$C = (zs^{-1} + rd_A s^{-1}) \times G$$

Collecting the common term s^{-1},

$$C = (z + rd_A)s^{-1} \times G$$

Expanding the definition of s from signature step 6,

$$C = (z + rd_A)(z + rd_A)^{-1}(k^{-1})^{-1} \times G$$

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

$$C = k \times G$$

From the definition of r, this is verification step 6.

This shows only that a correctly signed message will verify correctly; other properties such as incorrectly signed messages failing to verify correctly and resistance to cryptanalytic attacks are required for a secure signature algorithm.

Public key recovery

Given a message m and Alice's signature r, s on that message, Bob can (potentially) recover Alice's public key:[7]

1. Verify that r and s are integers in $[1, n-1]$. If not, the signature is invalid.
2. Calculate a curve point $R = (x_1, y_1)$ where x_1 is one of r, $r + n$, $r + 2n$, etc. (provided x_1 is not too large for the field of the curve) and 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 R value results in a distinct recovered key.

3. Calculate $e = \mathrm{HASH}(m)$, where HASH is the same function used in the signature generation.
4. Let z be the L_n leftmost bits of e.
5. Calculate $u_1 = -zr^{-1} \mod n$ and $u_2 = sr^{-1} \mod n$.
6. Calculate the curve point $Q_A = (x_A, y_A) = u_1 \times G + u_2 \times R$.
7. The signature is valid if Q_A, matches Alice's public key.
8. The signature is invalid if all the possible 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

Start with the definition of Q_A from recovery step 6,

$$Q_A = (x_A, y_A) = u_1 \times G + u_2 \times R$$

From the definition $R = (x_1, y_1) = k \times G$ from signing step 4,

$$Q_A = u_1 \times G + u_2 k \times G$$

Because elliptic curve scalar multiplication distributes over addition,

$$Q_A = (u_1 + u_2 k) \times G$$

Expanding the definition of u_1 and u_2 from recovery step 5,

$$Q_A = (-zr^{-1} + skr^{-1}) \times G$$

Expanding the definition of s from signature step 6,

$$Q_A = (-zr^{-1} + k^{-1}(z + rd_A)kr^{-1}) \times G$$

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

$$Q_A = (-zr^{-1} + (zr^{-1} + d_A)) \times G$$

The first and second terms cancel each other out,

$$Q_A = d_A \times G$$

From the definition of $Q_A = d_A \times G$, 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 $R = (x_1, y_1)$ from signature value r.

Security

In December 2010, a group calling itself *fail0verflow* announced the recovery of the ECDSA private key used by Sony to sign software for the PlayStation 3 game console. However, this attack only worked because Sony did not properly implement the algorithm, because k was static instead of random. As pointed out in the Signature generation algorithm section above, this makes d_A solvable, rendering the entire algorithm useless.[8]

On March 29, 2011, two researchers published an IACR paper[9] demonstrating that it is possible to retrieve a TLS private key of a server using OpenSSL that authenticates with Elliptic Curves DSA over a binary field via a timing attack.[10] The vulnerability was fixed in OpenSSL 1.0.0e.[11]

In August 2013, it was revealed that bugs in some implementations of the Java class SecureRandom (https://docs.oracle.com/javase/10/docs/api/java/security/SecureRandom.html) sometimes generated collisions in the 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 app implementations, which use Java and rely on ECDSA to authenticate transactions.[12]

This issue can be prevented by deterministic generation of k, as described by RFC 6979.

Concerns

Some concerns expressed about ECDSA:

1. *Political concerns*: the trustworthiness of NIST-produced curves being questioned after revelations were made that the NSA willingly inserts backdoors into software, hardware components and published standards; well-known cryptographers[13] have expressed[14][15] doubts about how the NIST curves were designed, and voluntary tainting has already been proved in the past.[16][17] (See also the *libssh curve25519 introduction*.[18]) 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, its slowness, and design flaws which reduce security in insufficiently defensive implementations.[19]

Implementations

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

- Botan
- Bouncy Castle
- cryptlib
- Crypto++
- Crypto API (Linux)
- GnuTLS
- libgcrypt
- LibreSSL

- mbed TLS
- Microsoft CryptoAPI
- OpenSSL
- wolfCrypt

See also

- EdDSA
- RSA (cryptosystem)

References

1. Johnson, Don; Menezes, Alfred (1999). "The Elliptic Curve Digital Signature Algorithm (ECDSA)". CiteSeerX 10.1.1.38.8014 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.8014). {{cite journal}}: Cite journal requires |journal= (help)
2. NIST FIPS 186-4, July 2013, pp. 19 and 26 (http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf)
3. 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%5F27c3%5Fconsole%5Fhacking%5F2010.pdf) December 15, 2014, at the Wayback Machine, page 123–128
4. "Android Security Vulnerability" (https://bitcoin.org/en/alert/2013-08-11-android). Retrieved February 24, 2015.
5. Pornin, T. (2013). "RFC 6979 - Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)" (https://www.rfc-editor.org/rfc/rfc6979.html). doi:10.17487/RFC6979 (https://doi.org/10.17487%2FRFC6979). Retrieved February 24, 2015. {{cite journal}}: Cite journal requires |journal= (help)
6. "The Double-Base Number System in Elliptic Curve Cryptography" (http://www.lirmm.fr/~imbert/talks/laurent_Asilomar_08.pdf) (PDF). Retrieved April 22, 2014.
7. Daniel R. L. Brown SECG SEC 1: Elliptic Curve Cryptography (Version 2.0) https://www.secg.org/sec1-v2.pdf
8. 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.
9. "Cryptology ePrint Archive: Report 2011/232" (http://eprint.iacr.org/2011/232). Retrieved February 24, 2015.
10. "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*.
11. "ChangeLog" (http://www.openssl.org/news/changelog.html). OpenSSL Project. Retrieved April 22, 2014.
12. "Android bug batters Bitcoin wallets" (https://www.theregister.co.uk/2013/08/12/android_bug_batters_bitcoin_wallets/). The Register. August 12, 2013.
13. 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*.
14. "SafeCurves: choosing safe curves for elliptic-curve cryptography" (http://safecurves.cr.yp.to/rigid.html). October 25, 2013.

15. Bernstein, Daniel J.; Lange, Tanja (May 31, 2013). "Security dangers of the NIST curves" (https://www.hyperelliptic.org/tanja/vortraege/20130531.pdf) (PDF).
16. 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*.
17. 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.
18. "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*.
19. 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*.

Further reading

- Accredited Standards Committee X9 (http://www.x9.org), *ASC X9 Issues New Standard for Public Key Cryptography/ECDSA*, Oct. 6, 2020. Source (https://x9.org/asc-x9-issues-new-standard-for-public-key-cryptography-ecdsa/)
- 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, 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, editors, *Advances in Elliptic Curve Cryptography*, London Mathematical Society Lecture Note Series 317, Cambridge University Press, 2005.
- Hankerson, D.; Vanstone, S.; Menezes, A. (2004). *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. New York: Springer. doi:10.1007/b97644 (https://doi.org/10.1007%2Fb97644). ISBN 0-387-95273-X. S2CID 720546 (https://api.semanticscholar.org/CorpusID:720546).

External links

- Digital Signature Standard; includes info on ECDSA (https://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)

Retrieved from "https://en.wikipedia.org/w/index.php?title=Elliptic_Curve_Digital_Signature_Algorithm&oldid=1197349912"

-