lamps
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2021

R. Struik
Struik Security Consultancy
March 11, 2021

ECDSA Signatures in Verification-Friendly Format
draft-struik-lamps-verification-friendly-ecdsa-01

Abstract

This document specifies how to represent ECDSA signatures so as to facilitate accelerated verification of single signatures and fast batch verification. We demonstrate that this representation technique can be applied retroactively by any device (rather than only by the signer), thereby facilitating transitioning to always generating ECDSA signatures in this way, without changing standardized ECDSA specifications. This facilitates verifying devices to reap the significant speed-up potential (ranging from ~1.3x to more than 2x) fast verification techniques afford.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Status of This Memo

Table of Contents

1. Fostering Fast Verification with ECDSA

ECDSA is one of the most widely used elliptic-curve digital signature algorithms. It has been standardized in FIPS Pub 186-4, ANSI X9.62, BSI, SECG, and IETF, and is widely deployed by a plethora of internet protocols specified by the Internet Engineering Task Force (IETF), with industry specifications in the areas of machine-to-machine communication, such as ZigBee, ISA, and Thread, with wireless communication protocols, such as IEEE 802.11, with payment protocols, such as EMV, with vehicle-to-vehicle (V2V) specifications, as well as with electronic travel documents and other specifications developed under a more stringent regulatory oversight regime, such as, e.g., ICAO and PIV. ECDSA is the only elliptic-curve based signature scheme endorsed by regulatory bodies in both the United States and the European Union.

While methods for accelerated verification of ECDSA signatures and for combining this with key computations have been known for over 1 1/2 decade (see, e.g., [SAC2005] and [SAC2010]), these have been commonly described in technical papers in terms of ECDSA*, a slightly modified version of ECDSA, where their use with standardized ECDSA seems less well known. It is the purpose of this document to bridge this gap and describe how ECDSA signatures can be easily generated to facilitate more efficient verification, without failing. We emphasize that this does not require changes to standardized specifications of ECDSA, thereby allowing reuse of existing standards and easy integration with existing implementations. We exemplify this for ECDSA certificates.

2. Review of ECDSA and ECDSA*

In this section, we summarize the properties of the signature scheme ECDSA and of the modified signature scheme ECDSA* that are relevant for our exposition. The signature schemes are defined in terms of a suitable elliptic curve E, hash function H, and several representation functions, where n is the (prime) order of the base point G of this curve, and where E is an elliptic curve in short-Weierstrass form. For full details, we refer to the relevant standards.

With the ECDSA signature scheme, the signature over a message m provided by a signing entity with static private key d is an ordered pair (r,s) of integers in the interval [1,n-1], where the value r is derived from a so-called ephemeral signing key R:=k*G generated by the signer via a fixed public conversion function and where the value s is a function of the ephemeral private key k, the static private key d, the value r and the value e derived from message m via hash function H and representation hereof in the interval [0,n-1]. (More specifically, one has e=s*k-d*r (mod n), where r is a function of the x-coordinate of R.) A signature (r,s) over message m purportedly signed by an entity with public key Q:=d*G is accepted if Q is indeed a valid public key, if both signature components r and s are integers in the interval [1,n-1] and if the reconstructed value R' derived from the purported signature, message, and public key yields r, via the same fixed conversion function as used during the signing operation. (More specifically, one computes R':=(1/s)*(e*G+r*Q) and checks that r is the same function of the x-coordinate of R'.)

With the ECDSA* signature scheme, one follows the same signing operation, except that one outputs as signature the ordered pair (R,s), rather than the pair (r,s), where R is the ephemeral signing key; one accepts a signature (R,s) over message m purportedly signed by an entity with public key Q by first computing the value r derived from signature component R via the conversion function, checking that

Q is indeed a valid public key and that both r and s are integers in the interval [1,n-1], computing R’:=(1/s)*(e*G+r*Q) and checking whether, indeed, R’=R.

It is known that ECDSA signatures and the corresponding ECDSA* signatures have the same success/failure conditions (i.e., ECDSA and ECDSA* are equally secure): if (r,s) is a valid ECDSA signature for message m purportedly signed by an entity with public key Q, then (R’,s) is a valid corresponding ECDSA* signature, where R’:=(1/s)(e*G+r*Q) is a point for which the conversion function yields r. Conversely, if (R,s) is a valid ECDSA* signature for message m purportedly signed by an entity with public key Q, then (r,s) is a valid corresponding ECDSA signature, where r is obtained from R via the conversion function.

It is well-known that if an ECDSA signature (r,s) is valid for a particular message m and public key Q, then so is (r,-s) -- the so-called malleability -- and that, similarly, if an ECDSA* signature (R,s) is valid, then so is (-R,-s), where this relies on the fact that the conversion function only depends on the x-coordinate of R.

3. Signature Verification with ECDSA and ECDSA*

In this section, we more closely scrutinize ECDSA and ECDSA* verification processes.

With ECDSA*, signature verification primarily involves checking an elliptic curve equation, viz. checking whether R = (1/s)*(e*G+r*Q), which lends itself to accelerated signature verification techniques and the ability to use batch verification techniques, with significant potential for accelerated verification (with ~1.3x and up and more than 2x speed-up potential, respectively). Here, speed-ups are due to the availability of the point R, which effectively allows checking an equation of the form -s*R + (e*G+r*Q)=O instead (where O is the identity element of the curve). Similarly to the case with EdDSA [RFC8032] (which natively represents the ephemeral signing key R as part of the signature), this offers the potential for batch verification, by checking a randomized linear combination of this equation instead (thereby sharing the so-called point doubling operations amongst all individual verifications and, potentially, sharing scalars for signers of more than one message). In the case of single verifications, efficient tricks allow reducing the bit-size of the scalars involved in evaluating this expression (thereby effectively halving the required point doubling operations).

With ECDSA itself, these techniques are generally not available, since one cannot uniquely (and efficiently) reconstruct R from r: both R and -R yield the same r value. If the conversion function only has two pre-images, though, one can use malleability to remove ambiguity altogether.

The modified ECDSA signing procedure is as follows:

a. Generate ECDSA signature (r,s) of message m;

b. If the ephemeral signing key R has odd parity of the y-coordinate, change (r,s) to (r,-s).

Note that this modified signing procedure removes the ambiguity in the reconstruction of R from r if the conversion function would otherwise only have two preimages, since R and -R have different parity of the y-coordinate. In practice, this is the case for all prime-order curves, including the NIST prime curves P-256, P-384, P-521, all standardized Brainpool curves, and, e.g., the "BitCoin" curve secp256k1. (This follows from the observation that, for prime-order curves, r generally uniquely represents the x-coordinate of R.)

NOTE: With ECDSA, any party (not just the signer) can recompute the ephemeral signing key R’ from a valid signature, since R’:=(1/s)(e*G+r*Q). In particular, any party can retroactively put the ECDSA signature in the required form above, thereby allowing subsequent unique reconstruction of the R value from r by verifying entities that know this modified signing procedure was indeed followed (again, subject to the assumption that r would only have two preimages otherwise, as is generally the case with prime-order curves).

One can extend this technique to also apply to curves that have a small co-factor h, e.g., h=4 or h=8 (rather than h=1, as is the case with prime-order curves). This extension is out of scope for the current document.

4. Transitionary Considerations

The modified signing procedure described in Section 3 facilitates the use of accelerated ECDSA verification techniques by devices that wish to do so, provided these know that this modified signing procedure was indeed followed. This can be realized explicitly via a new "fast-verification-friendly" label (e.g., OID) indicating that this was indeed the case. This has the following consequences:

a. New device: accept both old and new label and apply speed-ups with new label if possible (and desired);

b. Old device: implement flimsy parser that replaces new label by old label and proceed as with traditional ECDSA verification.

Note that this parser "label replacement" step is a public operation, so any interface can implement this step.

A label can also be realized implicitly (e.g., by stipulating the modified signing procedure in protocol specifications that use ECDSA signatures), where the benefit of not having to introduce a new label explicitly should be weighed against potential disadvantages of implicit labels, such as requiring extra care with specification work to avoid confusion and the likely need to reintroduce an explicit label if ECDSA signatures are processed outside the original context (e.g., using a generic crypographic token).

As suggested before, any device can implement the modified ECDSA signing procedure retroactively, so one could conceivably implement this once for all existing ECDSA signatures and only use "new" labels once this task has been completed (i.e., old labels could be mothballed from then on).

NOTE: the above labeling procedures assume that old and new labels are not part of the message to be signed. If they are, one may not be able to mothball old labels. In this case, signing devices should always use the old label during ECDSA signing and only change this to the corresponding new label afterwards, whereby verifying devices always replace the new label (since simply a pseudonym) by the corresponding old label before processing the ECDSA signature. This ensures that the signature semantics are not impacted and that old devices' ECDSA verification implementations (after reinstating old labels) work as is, while still being able to flag verification-friendly ECDSA signature formatting.

5. Implementation Status

[Note to the RFC Editor] Please remove this entire section before publication, as well as the reference to [RFC7942].

The ECDSA* signature scheme has been implemented in V2V specifications [P1609.2], where ECDSA is used with the NIST curves P-224 and P-256.

6. Informal Comparison with Speed-ups for EdDSA Signatures

The main message of this draft is as follows (no crypto required, except believing that the third step below works):

a. EdDSA [RFC8032] does allow speedy signature verification and batch verification, since the signature is (R,s), i.e., it represents the ephemeral signing key R as part of the signature;

b. With ECDSA, the signature is (r,s), where r is derived from the signing key R (essentially, r is the x-coordinate of R if the curve has co-factor h=1). However, generally, one cannot go back and get (r,s) --> (R,s), at least not efficiently;

c. If one uses the modified ECDSA signing procedure of Section 3, one can, though, thereby allowing similar accelerations (30% and up) for signature verification as EdDSA does. This can be viewed as "point compression" (since it determines which of R and -R apply);

d. The rest is detail, where the ideas underlying the speed-ups informally described in Section 3 are described in detail in the papers [SAC2005] and [SAC2010].

7. Security Considerations

The signature representation change described in this document is publicly known and, therefore, does not affect security provisions. Obviously, any adversary could change the signature value in a malicious way, so as to make signature verification fail. This does, however, not extend capabilities the adversary already had.

8. Privacy Considerations

The signature representation change described in this document is publicly known and, therefore, does not affect privacy provisions.

9. IANA Considerations

This section requests the following IANA code point assignments.

Editorial Note: the approach below is simply one way of realizing ECDSA* functionality. Other options to consider include, e.g., introducing a non-critical extension as label, where old devices can simple ignore this. This will be elaborated upon further in next versions of this draft, after feedback.

9.1. OIDs for Use with PKIX and CMS

This section registers the following object identifiers for the verification-friendly version of ECDSA introduced in this document:

a. id-ecdsa-star-with-sha256 ::= {iso(1) identified-organization(3) thawte (101) (100) 81};

b. id-ecdsa-star-with-sha384 ::= {iso(1) identified-organization(3) thawte (101) (100) 82};

c. id-ecdsa-star-with-sha512 ::= {iso(1) identified-organization(3) thawte (101) (100) 83};

d. id-ecdsa-star-with-shake128 ::= {iso(1) identified-organization(3) thawte (101) (100) 84};

e. id-ecdsa-star-with-shake256 ::= {iso(1) identified-organization(3) thawte (101) (100) 85}.

Each of these object identifiers indicates the use of ECDSA with the indicated hash function, as the corresponding object identifiers without the "-star-" substring specified in [RFC5480] (for ECDSA with SHA2-hash family members) and in [RFC8692] (for ECDSA with SHAKE family members) do, where the "-star-" substring simply indicates that the modified signing procedure specified in Section 3 of this document was indeed used.

These new object identifiers are used with PKIX certificates and CMS in the same way as the corresponding object identifiers without the "-star-" substring, except that verifying devices now have the option to implement ECDSA signature verification as if ECDSA* signatures had been used, since the new object identifiers indicate the modified signing operation was followed, as illustrated in Section 3 of this document.

As mentioned in Section 4, any ECDSA signature with the old object identifier can be changed retroactively to one with the corresponding new object identifier, provided one has assurance that the modified ECDSA signing procedure was indeed followed and, conversely, any ECDSA signature with the new object identifier can be changed to one with the corresponding old object identifier, without change in semantics (assuming these object identifiers are not part of the message that is signed).

With [RFC5280], the signature algorithm is indicated twice: once as signatureAlgorithm field of the Certificate and once as the Signature field of the sequence tbsCertificate, where the former is not part of the message to be signed, whereas the latter is. Moreover, these two fields are stipulated to be the same (see Sections 4.1.1.2 and 4.1.2.3 of [RFC5280]). In this case, old and new labels MUST be used as indicated in the NOTE of Section 3, where the two fields indicating the signature algorithm are always both changed at the same time (thereby, strictly complying with MUST behavior of PKIX that these two fields should be the same).

10. Acknowledgements

Thanks to Rich Salz for suggesting to informally compare speed-ups with ECDSA* with those of EdDSA (now in Section 6).

11. References

11.1. Normative References

[FIPS-186-4]
FIPS 186-4, "Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186-4", US Department of Commerce/National Institute of Standards and Technology, Gaithersburg, MD, July 2013.

[I-D.ietf-lwig-curve-representations]
Struik, R., "Alternative Elliptic Curve Representations", draft-ietf-lwig-curve-representations-19 (work in progress), December 2020.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8692] Kampanakis, P. and Q. Dang, "Internet X.509 Public Key Infrastructure: Additional Algorithm Identifiers for RSASSA-PSS and ECDSA Using SHAKEs", RFC 8692, DOI 10.17487/RFC8692, December 2019, <https://www.rfc-editor.org/info/rfc8692>.

[SEC1] SEC1, "SEC 1: Elliptic Curve Cryptography, Version 2.0", Standards for Efficient Cryptography, , June 2009.

[SEC2] SEC2, "SEC 2: Elliptic Curve Cryptography, Version 2.0", Standards for Efficient Cryptography, , January 2010.

11.2. Informative References

[ECC] I.F. Blake, G. Seroussi, N.P. Smart, "Elliptic Curves in Cryptography", Cambridge University Press, Lecture Notes Series 265, July 1999.

[GECC] D. Hankerson, A.J. Menezes, S.A. Vanstone, "Guide to Elliptic Curve Cryptography", New York: Springer-Verlag, 2004.

[P1609.2] IEEE 1609.2-2013, "IEEE Standard for Wireless Access in Vehicular Environments-Security Services for Applications and Management Messages", IEEE Vehicular Technology Society, New York: IEEE, 2013.

[SAC2005] A. Antipa, D.R. Brown, R. Gallant, R. Lambert, R. Struik, S.A. Vanstone, "Accelerated Verification of ECDSA Signatures", SAC 2005, B. Preneel, S. Tavares, Eds., Lecture Notes in Computer Science, Vol. 3897, pp. 307-318, Berlin: Springer, 2006.

[SAC2010] R. Struik, "Batch Computations Revisited: Combining Key Computations and Batch Verifications", SAC 2010, A. Biryukov, G. Gong, D.R. Stinson, Eds., Lecture Notes in Computer Science, Vol. 6544, pp. 130-142, Berlin-Heidelberg: Springer, 2011.

Author’s Address

Rene Struik
Struik Security Consultancy

Email: rstruik.ext@gmail.com