Verification-Friendly ECDSA

draft-struik-secdispatch-verify-friendly-ecdsa-00

René Struik

Struik Security Consultancy

E-mail: rstruik.ext@gmail.com

IETF 111 – virtual San Francisco, USA, July 2021

Outline

1. ECC Signature Schemes:
   – ECDSA, EdDSA
   – Implementation details
   – ECDSA*
2. Speed-ups:
   – Verification with ECDSA vs. with ECDSA*
   – How to get from ECDSA to ECDSA*?
4. ECDSA* with reuse of <u>existing ECDSA standards</u>
5. Conclusions, next steps

ECC Signature Algorithms (1)

NIST curves:

Curve model:	Weierstrass curve
Curve equation:	$y^2 = x^3 + a \cdot x + b$ (**mod** p)
Base point:	$G=(G_x, G_y)$
Scalar multiplication:	addition formulae using, e.g., mixed Jacobian coordinates
Point representation:	both coordinates of point $P=(X, Y)$ (affine coordinates)
	0x04 || $\underline{X}$ || $\underline{Y}$ in most-significant-bit/octet first order
Examples:	NIST P-256 (ANSI X9.62, NIST SP 800-56a, SECG, etc.);
	Brainpool256r1 (RFC 5639)

ECDSA:

Signature:	$\underline{r}$ || $\underline{s}$ in most-significant-bit/octet first order
Signing equation:	$e = s \cdot k - d \cdot r$ (**mod** n), where e=Hash(m), $R=k\, G$, $R \to r$
Verification:	$R' = (e/s)\, G + (r/s)\, Q$, where $Q= d\, G$; check that $R' \to r$
Example:	ECDSA, w/ P-256 and SHA-256 (FIPS 186-4, ANSI X9.62, etc.)
Note:	message m pre-hashed

ECC Signature Algorithms (2)

CFRG curves:

Curve model: twisted Edwards curve

Curve equation: $a \cdot x^2 + y^2 = 1 + d \cdot x^2 \cdot y^2 \; (\mathbf{mod}\; p)$

Base point: $G=(G_x, G_y)$

Scalar multiplication: Dawson formulae, using extended coordinates $(X: Y: T: Z)$

Point representation: compressed point $P=(Y, X')$, where $X'=\mathrm{lsb}(X)$

$\underline{Y} \,||\, \underline{X}'$

Examples: Edwards25519, Edwards448 (RFC 7748)

EdDSA:

Signature: $\underline{R} \,||\, \underline{s}$

Signing equation: $s = k + e \cdot d \; (\mathbf{mod}\; n)$, where $e=\mathrm{Hash}(\underline{Q} \,||\, \underline{R} \,||\, m)$, $R=k\,G$

Verification: $s\,G = R + e\,Q$, where $Q=d\,G$

Example: Ed25519-SHA-512, Ed448-SHAKE-256 (RFC 8032)

Notes: Deterministic Schnorr signature, where $k=\mathrm{Hash}\,(d' \,||\, m)$

Variant w/ pre-hashing uses Hash(m) instead of m

Implementation Detail (1)

Aspect:	ECDSA	EdDSA
Curve model:	Weierstrass	Edwards
Base point:	affine	affine
Internal coord:	Jacobian	extended
Formulae:	Jacobian	Dawson
Wire format:	(r, s)	(R, s)
@signing:		
#message passes:	once	twice
eph. signing key R:	offline	inline
inversions mod n:	once	none
@verification:		
single verification	no speed-ups	speed-ups
batch verification	no speed-ups	speed-ups

NOTE:
EdDSA is full-Schnorr signature, which are also defined for Weierstrass curves

- Not standardized with IETF ☹
- Standardized with BSI (as short-Schnorr Signature (e,s))

APPLICATION NOTE:
- Batch verification of certificate chains (and any other batch)
- Batch sanity checks

Implementation Detail (2)

Aspect:	ECDSA	EdDSA	ECDSA*
Curve model:	Weierstrass	Edwards	Weierstrass
Base point:	affine	affine	affine
Internal coord:	Jacobian	extended	Jacobian
Formulae:	Jacobian	Dawson	Jacobian
Wire format:	($\underline{r}$, $\underline{s}$)	($\underline{R}$, $\underline{s}$)	($\underline{R}$, $\underline{s}$)
@signing:			
#message passes:	once	twice	once
eph. signing key *R*:	offline	inline	offline
inversions mod *n*:	once	none	once
@verification:			
single verification	no speed-ups	speed-ups	speed-ups ☺
batch verification	no speed-ups	speed-ups	speed-ups ☺

Verification Detail (1)

ECDSA:

Signature: $\underline{r} \,||\, \underline{s}$ in most-significant-bit/octet first order

Signing equation: $e = s \cdot k - d \cdot r \;(\mathbf{mod}\; n)$, where e=Hash(m), R=$k\,G$, $R \rightarrow r$

Verification: compute $R' = (e/s)\,G + (r/s)\,Q$;
check that $R' \rightarrow r$

ECDSA*:

Signature: $\underline{R} \,||\, \underline{s}$ in most-significant-bit/octet first order

Signing equation: $e = s \cdot k - d \cdot r \;(\mathbf{mod}\; n)$, where e=Hash(m), R=$k\,G$, $R \rightarrow r$

Verification: compute $\underline{R} \rightarrow r$;
compute $\underline{R} \rightarrow R$
check that $R = (e/s)\,G + (r/s)\,Q$, where $Q = d\,G$

Alternative verify: $\boldsymbol{\lambda\,(- R + (e/s)\,G + (r/s)\,Q) = O}$ ***for any*** $\boldsymbol{\lambda \neq 0}$

speed-ups: ~1.3x make scalars small, which halves ECC doubles (single verify)

up to ~ 6x amortize ECC doubles and common terms (batch verify)

ECDSA and ECDSA* the same if one could reverse $R' \rightarrow r$ mapping, but $\pm R' \rightarrow r$

How to Get from ECDSA to ECDSA*?

ECDSA and ECDSA* the same if one could reverse $R' \rightarrow r$ mapping, but $\pm R' \rightarrow r$

This follows from the fact that $R' \rightarrow r$ is defined as $r := x(R)$ (**mod** n)

For all prime-order curves, these pre-images come in pairs $\{R, -R\}$ in practice

Modified ECDSA signing procedure:
– Step 1: Generate ECDSA signature (r, s) of message m, as usual;
– Step 2: Change (r, s) to $(r, -s)$ if ephemeral key R has y-coordinate with odd parity

Notes:
– If (r, s) is a valid ECDSA signature, then so is $(r, -s)$ –– the so-called malleability
– Any party can perform Step 2, since for valid signatures $R := (e/s)\, G + (r/s)\, Q$
This party does not have to be the signer and this can be done retroactively
– If verifyer knows that modified signing produre was used, $R' \rightarrow r$ has unique preimage in practice for all prime-order curves (implicit point compression R)

Transitioning towards ECDSA* (1)

ECDSA with modified signing procedure allows implementation of ECDSA* with existing ECDSA standards (for prime-order curves), provided the verifying device knows this modified signing procedure was indeed used

Option #1: "Big Bang"

– Implement modified signing procedure retroactively for all existing ECDSA signatures;

– Generate all new ECDSA signatures with the modified signing procedure (i.e., mothball the old way of generating ECDSA signatures)

Option #2: mandate in specifications

– This has same effect as Option #1, for a particular protocol

Question: does this entice implementors enough to adopt speed-ups en masse?

Option #3: define new label for ECDSA*

– New devices who recognize label can uniquely recover R from r

– Old devices that have parser that replaces label ECDSA* with label ECDSA as pre-processing step can still process ECDSA signatures as usual

NOTE: no changes to old ECDSA processing of triples ($h(m)$, (r,s), Q)

Transitioning towards ECDSA* (2)

Applications with IETF protocols:

Everywhere, e.g., PKIX, CMS, Certificate Transparency, OpenPGP, COSE, JOSE, lake, etc.

Example w/ PKIX:

include `id-ecdsa-star-with-sha256 ::=`

```
      {iso(1) identified-organization(3)thawte (101) (100) 81}
```

(for consideration of old devices (if any), see draft, Section 4)

Example w/ OpenPGP:

include ECDSA* as Suite #25 in Table 15 of `draft-ietf-openpgp-crypto-refresh`

Example w/ lake:

use ECDSA* instead of ECDSA with `draft-ietf-lake-edhoc-08`

What about other deployed signature schemes similar to ECDSA?

Richer definition allows speed-ups to apply also to other signature schemes,
e.g., Chinese SM2, German ECGDSA scheme, Russian GOST R34.10-2012 (RFC 7091)

Conclusions & Question to Group

Summary:

– ECDSA verification can take advantage of ECDSA* speed-ups, similarly to EdDSA, both in single verify and batch verify case
– Techniques trivial to use with all prime-order curves (roughly all existing deployments), for those verifying devices that wish this
– Techniques compatible with existing ECDSA for prime-order curves
– Speed-ups deployed in V2V (P1609.2); useful for servers with more widespread use client certificates

Techniques known since 1994 (batch), resp. since Jan 2005 (single)

Question to Group:

– Discussed w/ lamps @IETF-110, but not yet in revised charter
– Useful throughout IETF; do in lamps, elsewhere?
– Should be quick project (mainly definition of code points)