Compact ECC Encodings for TLS 1.3

draft-mattsson-tls-compact-ecc-03

Existing ECDHE and ECDSA encodings in TLS 1.3

- **Two problems:** The encodings used in the ECDHE groups *secp256r1, secp384r1*, and *secp521r1* and the ECDSA signature algorithms *ecdsa_secp256r1_sha256, ecdsa_secp384r1_sha384*, and *ecdsa_secp521r1_sha512* have **significant overhead** and the ECDSA encoding produces **variable-length signatures**.
- The document defines new **optimal fixed-length encodings** and registers new ECDHE groups and ECDSA signature algorithms using these new encodings.
- The encoding are defined as a subset of the bytes in the current encodings. This makes interoperable implementations easy.
- **The new encodings have the same security properties and requirements as the old encodings.**
- The new encodings reduce the size of the ECDHE groups with 33, 49, and 67 bytes and the ECDSA algorithms with an average of 7 bytes.
- When *secp256r1_compact* and *ecdsa_secp256r1_sha256_compact* are used as a replacement for the the old encodings they **reduce the size of a mutually authenticated TLS handshake with on average 80 bytes.**
- These new encodings also work in DTLS 1.3 and are **especially useful in cTLS**.
  - The alternative to do something cTLS specific seems worse.
  - Many IoT devices want to continue using P-256 and ECDSA.

Compact ECDHE Encoding

Given a UncompressedPointRepresentation structure

```
struct {
    uint8 legacy_form = 4;
    opaque X[coordinate_length];
    opaque Y[coordinate_length];
} UncompressedPointRepresentation;
```

the legacy_form and Y field are omitted to create a CompactRepresentation structure.

```
struct {
    opaque X[coordinate_length];
} CompactRepresentation;
```

For secp256r1 the UncompressedPointRepresentation is 65 bytes and the CompactRepresentation is 32 bytes, saving of 33 bytes.

```
04 A6 DA 73 92 EC 59 1E 17 AB FD 53 59 64 B9 98
94 D1 3B EF B2 21 B3 DE F2 EB E3 83 0E AC 8F 01
51 81 26 77 C4 D6 D2 23 7E 85 CF 01 D6 91 0C FB
83 95 4E 76 BA 73 52 83 05 34 15 98 97 E8 06 57
80
```

```
A6 DA 73 92 EC 59 1E 17 AB FD 53 59 64 B9 98 94
D1 3B EF B2 21 B3 DE F2 EB E3 83 0E AC 8F 01 51
```

Section on Implementation Considerations

- The y-coordinate does not affect the shared secret but it is needed for point validation.
- The y-coordinate might also be needed for combability with APIs.
- Compact representation have no disadvantages compared to point compression where the sign bit is included.
- To my knowledge, there has never been any patents on compact representation.

3.2. Implementation Considerations for Compact Representation

For compatibility with APIs a compressed y-coordinate might be required. For validation or for compatibility with APIs that do not support the full [SECG] format an uncompressed y-coordinate might be required (using the notation in [SECG]):

- If a compressed y-coordinate is required, then the value ~yp set to zero can be used. The compact representation described above can in such a case be transformed into the SECG point compressed format by prepending X with the single byte 0x02 (i.e., M = 0x02 || X).
- If an uncompressed y-coordinate is required, then a y-coordinate has to be calculated following Section 2.3.4 of [SECG] or Appendix C of [RFC6090]. Any of the square roots (see [SECG] or [RFC6090]) can be used. The uncompressed SECG format is M = 0x04 || X || Y.

For example: The curve P-256 has the parameters (using the notation in [RFC6090])

- $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$
- a = -3
- b = 41058363725152142129326129780047268409114441015993725554835256314039467401291

Given an example x:

- x = 11579208918339630209554680715474055844340679510865333639897069777272788799766525

we can calculate y as the square root $w = (x^3 + a \cdot x + b)^{((p+1)/4)} \pmod p$

- y = 83438718007019280682007586491862600528145125996401575416632522940595860276856

Note that this does not guarantee that (x, y) is on the correct elliptic curve. A full validation according to Section 5.6.2.3.3 of [SP-800-56A] is done by also checking that $0 \le x < p$ and that $y^2 \equiv x^3 + a \cdot x + b \pmod p$. The implementation **MUST** perform public-key validation.

Compact ECDSA Encoding

Given a variable-length DER-encoded ECDSA-Sig-Value structure

```
Ecdsa-Sig-Value ::= SEQUENCE {
    r       INTEGER,
    s       INTEGER
}
```

```
30  69: SEQUENCE {
02  33:  INTEGER
          00 D7 A4 D3 4B D5 4F 55 FE E1 A8 96 25 67 8C 3D
          D5 E5 F6 0D AC 73 EC 94 0C 5C 7B 93 04 A0 20 84
          A9
02  32:  INTEGER
          28 9F 59 5E D4 88 B9 AC 68 9A 3D 19 2B 1A 8B B3
          8F 34 AF 78 74 C0 59 C9 80 6A 1F 38 26 93 53 E8
          }
```

the SEQUENCE type, INTEGER type, and length fields are omitted and if necessary, the two INTEGER value fields are truncated (at most a single zero byte) or left padded with zeroes to the fixed length L.

For secp256r1 the ecdsa_secp256r1_sha256 example is 71 bytes and the ecdsa_secp256r1_sha256_compact signature is 64 bytes, saving 7 bytes.

```
D7 A4 D3 4B D5 4F 55 FE E1 A8 96 25 67 8C 3D D5
E5 F6 0D AC 73 EC 94 0C 5C 7B 93 04 A0 20 84 A9
28 9F 59 5E D4 88 B9 AC 68 9A 3D 19 2B 1A 8B B3
8F 34 AF 78 74 C0 59 C9 80 6A 1F 38 26 93 53 E8
```

New IANA Registrations

— Three new ECDHE groups for P-256, P-384, and P-521

Value	Description	Recommended	Reference
TBD1	secp256r1_compact	Y	[This-Document]
TBD2	secp384r1_compact	Y	[This-Document]
TBD3	secp521r1_compact	Y	[This-Document]

Table 1: Compact ECDHE Groups

— Three new signature algorithms for P-256, P-384, and P-521

Value	Description	Recommended	Reference
TBD4	ecdsa_secp256r1_sha256_compact	Y	[This-Document]
TBD5	ecdsa_secp384r1_sha384_compact	Y	[This-Document]
TBD6	ecdsa_secp521r1_sha512_compact	Y	[This-Document]

Table 2: Compact ECDSA Signature Algorithms