Alternative
Elliptic Curve Representations

draft-struik-lwig-curve-representations-00

René Struik
Struik Security Consultancy
E-mail: rstruik.ext@gmail.com

IETF 101 – London, UK, March 2018
Outline

1. The ECC Algorithm Zoo
   – NIST curve P-256, ECDSA
   – Curve25519
   – Ed25519

2. Implementation Detail

3. How to Reuse Code

4. How to Reuse Existing Standards

5. Conclusions
ECC Algorithm Zoo (1)

NIST curves:
Curve model: Weierstrass curve
Curve equation: \( y^2 = x^3 + ax + b \pmod{p} \)
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)

Examples: NIST P-256 (ANSI X9.62, NIST SP 800-56a, SECG, etc.);
Brainpool256r1 (RFC 5639)

ECDSA:
Signature: \( R || s \) in most-significant-bit/octet first order
Signing equation: \( e = s \cdot k + d \cdot r \pmod{n} \), where \( e=\text{Hash}(m) \)
Example: ECDSA, w/ P-256 and SHA-256 (FIPS 186-4, ANSI X9.62, etc.)
Note: message \( m \) pre-hashed

draft-struik-lwig-curve-representations-00
ECC Algorithm Zoo (2)

CFRG curves:
Curve model: Montgomery curve
Curve equation: \( B \cdot y^2 = x^3 + A \cdot x^2 + x \mod p \)
Base point: \( G = (G_x, G_y) \)
Scalar multiplication: Montgomery ladder, using projective coordinates \([X: :Z]\)
Point representation: x-coordinate of point \( P = (X, Y) \) (x-coordinate-only)

\[ X \text{ in least-significant-octet, most-significant-bit first order} \]
Examples: Curve25519, Curve448 (RFC 7748)

DH Key agreement:
Key shares: x-coordinates of \( X = xG \) (Party A) and \( Y = yG \) (Party B)
Shared key: x-coordinate of \( K_A = (h \cdot x)Y = K_B = (h \cdot y)X \), where \( h \) co-factor
Notes: Montgomery ladder can provide y-coordinate as well
Examples: X25519, X448 (RFC 7748)

\(^1\)Not in RFC 7748 or most code (NaCl, etc.)
ECC Algorithm Zoo (3)

CFRG curves:

Curve model: twisted Edwards curve
Curve equation: \( a \cdot x^2 + y^2 = 1 + d \cdot x^2 \cdot y^2 \pmod{p} \)
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 \( X'=\text{lsb}(X) \)
Examples: Edwards25519, Edwards448 (RFC 7748)

EdDSA:

Signature: \( R \ | \ | \ s \) in least significant bit/octet first order
Signing equation: \( s = k + e \cdot d \pmod{n} \), where \( e=\text{Hash}(Q \ | \ | R \ | \ | m) \)
Example: Ed25519-SHA-512, Ed448-SHAKE-256
Notes: Deterministic signature, where \( k=\text{Hash}(d' \ | \ | m) \)
Variant w/ pre-hashing uses \( \text{Hash}(m) \) instead of \( m \)
# Implementation Detail

<table>
<thead>
<tr>
<th>Aspect</th>
<th>NIST P-256</th>
<th>Curve25519</th>
<th>Edwards25519</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve model:</td>
<td>Weierstrass</td>
<td>Montgomery</td>
<td>twisted Edwards</td>
</tr>
<tr>
<td>Base point:</td>
<td>affine</td>
<td>x-coord only</td>
<td>affine</td>
</tr>
<tr>
<td>Internal coord:</td>
<td>Jacobian</td>
<td>x-projective</td>
<td>extended</td>
</tr>
<tr>
<td>Formulae:</td>
<td>Jacobian</td>
<td>Montgomery</td>
<td>Dawson</td>
</tr>
<tr>
<td>Wire format:</td>
<td>affine</td>
<td>x-coord only</td>
<td>compressed</td>
</tr>
<tr>
<td>Bit/Octet ordering:</td>
<td>MSB, msb</td>
<td>LSB, msb</td>
<td>LSB, lsb</td>
</tr>
</tbody>
</table>

Implementation drawback:
different arithmetic, different point format, different bit/octet encoding

*Lots of code to implement*...

a) key agreement co-factor ECDH using NIST P-256 + Curve25519 (TLS1.3);
b) key agreement + sign/verify Curve25519 + Ed25519 (JOSE [RFC 8037]);
c) key agreement + sign/verify P-256 + ECDSA & Curve25519 + Ed25519
## How to Reuse Code (1)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>NIST P-256</th>
<th>Curve25519</th>
<th>Edwards25519</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve model:</td>
<td>Weierstrass</td>
<td>Montgomery</td>
<td>(t)Edwards</td>
</tr>
<tr>
<td>Base point:</td>
<td>affine</td>
<td>affine</td>
<td>affine</td>
</tr>
<tr>
<td>Internal coord:</td>
<td>Jacobian</td>
<td>x-projective</td>
<td>extended</td>
</tr>
<tr>
<td>Formulae:</td>
<td>Jacobian</td>
<td>Montgomery</td>
<td>Dawson</td>
</tr>
<tr>
<td>Wire format:</td>
<td>affine</td>
<td>affine</td>
<td>affine</td>
</tr>
<tr>
<td>Bit/Octet ordering:</td>
<td>MSB, msb</td>
<td>MSB, msb</td>
<td>MSB, msb</td>
</tr>
</tbody>
</table>

Implementation drawback:

different arithmetic, different point format, different bit/octet encoding

we will deal with the encoding mess later on ...

---

draft-struik-lwig-curve-representations-00
How to Reuse Code (2)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Wei25519</th>
<th>Curve25519</th>
<th>Edwards25519</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve model:</td>
<td>Weierstrass</td>
<td>Montgomery</td>
<td>(t)Edwards</td>
</tr>
<tr>
<td>Base point:</td>
<td>affine</td>
<td>affine</td>
<td>affine</td>
</tr>
<tr>
<td>Internal coord:</td>
<td>Jacobian</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Formulae:</td>
<td>Jacobian</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>Wire format:</td>
<td>affine</td>
<td>affine</td>
<td>affine</td>
</tr>
<tr>
<td>Bit/Octet ordering:</td>
<td>MSB, msb</td>
<td>MSB, msb</td>
<td>MSB, msb</td>
</tr>
</tbody>
</table>

Implementation can use mappings between different curve models:

These mappings are so-called isomorphisms, so group structure remains the same. **Mappings easy to implement at virtually zero incremental cost**

draft-struik-lwig-curve-representations-00
How to Reuse Code (3)

<table>
<thead>
<tr>
<th>Aspect:</th>
<th>Wei25519</th>
<th>Curve25519</th>
<th>Edwards25519</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve model:</td>
<td>Weierstrass</td>
<td>Montgomery</td>
<td>(t)Edwards</td>
</tr>
</tbody>
</table>

**Common parms:**
- prime $p$: $p=2^{255}-19$
- co-factor $h$: $h=8$
- group size $n$: $n=2^{252}+ 0x14def9de a2f79cd6 5812631a 5cf5d3ed$

**Specific parms:**
- Base point $G$: $(9+A/3, \ldots)$, $(9, \ldots)$, $(c\cdot4/5, \ldots)$
- Constants:
  - $a=\ldots$, $A=486662$, $a=-1$
  - $b=\ldots$, $B=1$
  - $d=-121665/121666$

\[
(x, y) \xrightarrow{\text{Wei25519}} (u, v) : = (x-A/3, y) \xrightarrow{\text{Curve25519}} (x, y) \xrightarrow{\text{Edwards25519}} (x+A/3, v)
\]
# How to Reuse Code (4)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>NIST P-256</th>
<th>Wei25519</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve model:</td>
<td>Weierstrass</td>
<td>Weierstrass</td>
</tr>
<tr>
<td>Base point:</td>
<td>affine</td>
<td>affine</td>
</tr>
<tr>
<td>Internal coord:</td>
<td>Jacobian</td>
<td>Jacobian</td>
</tr>
<tr>
<td>Formulae:</td>
<td>Jacobian</td>
<td>Jacobian</td>
</tr>
<tr>
<td>Wire format:</td>
<td>affine</td>
<td>affine</td>
</tr>
<tr>
<td>Bit/Octet ordering:</td>
<td>MSB, msb</td>
<td>MSB, msb</td>
</tr>
</tbody>
</table>

Implementations can **re-use all code for Weierstrass curves** under the hood, using the mappings between different curve models.
How to Reuse Code (5)

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Wei25519</th>
<th>Curve25519</th>
<th>Edwards25519</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curve model:</td>
<td>Weierstrass</td>
<td>Montgomery</td>
<td>twisted Edwards</td>
</tr>
<tr>
<td>Base point:</td>
<td>affine</td>
<td>x-coord only</td>
<td>affine</td>
</tr>
<tr>
<td>Internal coord:</td>
<td>Jacobian</td>
<td>x-projective</td>
<td>extended</td>
</tr>
<tr>
<td>Formulae:</td>
<td>Jacobian</td>
<td>Montgomery</td>
<td>Dawson</td>
</tr>
<tr>
<td>Wire format:</td>
<td>affine</td>
<td>x-coord only</td>
<td>compressed</td>
</tr>
<tr>
<td>Bit/Octet ordering:</td>
<td>MSB, msb</td>
<td>LSB, msb</td>
<td>LSB, lsb</td>
</tr>
</tbody>
</table>

Implementations can re-use all code for Weierstrass curves under the hood, using the mapping between different curve models. One can deal with encoding conversion as pre- or postprocessing operation (or, fix encoding if one can).
Efficiency:
– Conversion between different curve models comes at negligible cost (only +0.1%)
– Point representation can impact efficiency (if no affine coordinates available):
  Example: TLS1.3 purposely uses x-coordinate only representation Curve25519, thus handicapping conversion to Weierstrass format (+12% cost due to point decompression [affine format would have done away with this])
– Encoding mess re endianness octet/bit ordering [RFC 7748, RFC 8032] does not help, but comes at negligible computational cost
How to Reuse Existing Standards (1)

Lots of code to implement... and ways to alleviate this
a) key agreement co-factor ECDH using NIST P-256 + Curve25519 (TLS1.3)
   – Use co-factor ECDH with Wei25519 curve and NIST encodings (@NIST SP 800-56a)
b) key agreement + sign/verify Curve25519 + Ed25519 (JOSE [RFC 8037])
   – Use ECDSA/Wei25519/SHA-256 (@FIPS 186-4)
   – Use Schnorr/Wei25519/SHA-256 and NIST encodings (@BSI ECC2.0 spec)
c) key agreement + sign/verify P-256+ ECDSA & Curve25519 + Ed25519
   – As above, but now Wei25519 + P-256 with NIST encodings allow single
     implementation (simply invoke with different domain parms)

Alternatively, one can reuse Weierstrass code under the hood if implementing
Curve25519 or Ed25519, provided one deals with encoding conversion as pre- and
post-processing operation.

NOTE: if one uses Ed25519 with SHA-256 one can reuse curve arithmetic and hash
function ECDSA/P-256/SHA-256, but not secure computation of (e, s) values.
How to Reuse Existing Standards (2)

Defining new schemes using existing standards
a) MQV key agreement using NIST P-256 + Curve25519 (DTLS1.3)
   – Use MQV with Wei25519 curve and NIST encodings (@NIST SP 800-56a)
b) X509 Certificate Infrastructure with CA key on Curve25519 [RFC 5280]
   – Use curve Wei25519 with X509 and NIST encodings (@RFC 5280)
c) Implicit certificates using NIST P-256 + Curve25519
   – Use curve Wei25519 with NIST encodings (@SEC4, P1609.2, Autonomous Vehicles, US Dept of Transport)
Conclusions

1. Different curve models can be implemented using the same code if one uses the short-Weierstrass model.
2. One can thereby reuse not just code, but also existing standards, thus significantly reducing standards development cycles.
3. Encoding format issues may negatively impact code reuse and reuse of existing standards, since can be used as artificial “moat around a solution”, making code reuse or algorithm agility economically less viable than these could/should be.
4. Representation conventions require more careful considerations by IETF in the future than has happened so far (in TLS1.3, CFRG).