Introducing curve25519-dalek

longcpp

longcpp9@gmail.com

March 25, 2020

基于蒙哥马利曲线 Curve25519 的 x 坐标系可构建高效安全密钥交换协议 X25519, 基于与其双向有理等价的 Edwards25519 曲线上点群可以构建高效率的数字签名算法 Ed25519. 为了避免由于因子为 8 可能导致的基于 Edwards25519 点群构建的密码协议的安全隐患, Ristretto 技术利用蒙哥马利曲线, 扭曲爱德华曲线以及雅各比四次曲线之间的同源性和商群之间的同构特性, 可以从 Edwards25519 点群中萃取中素数阶点群 ristretto255, 在保留速度等优势的同时, 避开了余因子不为 1 的弊端, 为密码协议构建提供坚实的数学结构支撑, 例如 Polkadot 项目中正是采用了基于 ristretto255 点群的 Schnorr 签名算法, sr25519.

Curve25519, Edwards25519 以及 Ristretto255 所依赖的数学理论较为复杂, 高效安全实现也有难度. Rust 语言实现的 curve25519-dalek 库中提供了 3 种点群的快速实现, 基于该库可实现种类丰富的密码协议: x25519-dalek[1]中实现了 X25519, ed25519-dalek[2]中实现了 Ed25519, schnorrkel[3]中实现了 sr25519, zkp[4]中实现了 Schnorr 形式的零知识证明, bulletproofs[5]中实现了零知识证明系统 Bulletproof. 本次我们结合数学理论简要介绍 curve25519-dalek 库提供的接口和功能,

1 Curve25519 与 Edwards25519 曲线

先回顾下 Curve25519 和 Edwards25519 曲线相关的细节. 蒙哥马利形式的 Curve25519 曲线是定义在素数域 $\mathbb{F}_p, p = 2^{255} - 19$ 上的, 记为 $\mathcal{M}(\mathbb{F}_p)$ 其曲线方程为

$$v^2 = u^3 + Au^2 + u, A = 486662 \in \mathbb{F}_p,$$

[1] https://github.com/dalek-cryptography/x25519-dalek

[2] https://github.com/dalek-cryptography/ed25519-dalek

[3] https://github.com/w3f/schnorrkel

[4] https://github.com/dalek-cryptography/zkp

[5] https://github.com/dalek-cryptography/bulletproofs

其中 $\#\mathcal{M}(\mathbb{F}_p) = h \cdot \ell, h = 8, \ell = 2^{252} + 27742317777372353535851937790883648493$. 单位元为无穷远点 $\mathcal{O}_{\mathcal{M}}$, $(0,0)$ 为 2 阶点, $(1, \sqrt{A+2}), (1, -\sqrt{A+2})$ 为 4 阶点, 也即:

$$\mathcal{M}(\mathbb{F}_p)[4] = \{\mathcal{O}_{\mathcal{M}}, (0,0), (1, \sqrt{A+2}), (1, -\sqrt{A+2})\}.$$

$\mathcal{M}(\mathbb{F}_p)$ 专用于 X25519 协议, 而该协议依赖的点群运算仅依赖 u 坐标, 选定的基点为

$$u(G_{\mathcal{M}}) = \mathsf{9}.$$

$\mathcal{M}(\mathbb{F}_p)$ 有两个点满足 u 坐标为 9, 为了与 Edwards25519 点群的基点 $G_{\mathcal{E}}$ 相对应, RFC7748 中规定:

$$v(G_{\mathcal{M}}) = \mathsf{0x5f51e65e475f794b1fe122d388b72eb36dc2b28192839e4dd6163a5d81312c14}.$$

与 $\mathcal{M}(\mathbb{F}_p)$ 双向有理等价的 (Birational Equivalent) 扭曲爱德华形式 Edwards25519 曲线 $\mathcal{E}(\mathbb{F}_p)$ 的方程为

$$-x^2 + y^2 = 1 + dx^2y^2, d = -\frac{121665}{121666} \in \mathbb{F}_p,$$

并且有 $\#\mathcal{E}(\mathbb{F}_p) = \#\mathcal{M}(\mathbb{F}_p)$. 点 (x, y) 的逆元 $-(x, y) = (-x, y)$.

Listing 1: 验证 Curve25519 与 Edwards25519

```
fp25519 = FiniteField(2^255 - 19)
A, B = fp25519(486662), fp25519(1)
curve25519 = EllipticCurve(fp25519, [0, A, 0, B, 0])
ell = curve25519.cardinality() / 8

mont_G = curve25519.point((0x9,
0x5f51e65e475f794b1fe122d388b72eb36dc2b28192839e4dd6163a5d81312c14))

def curve25519_8_torsion():
    point = curve25519.random_element()
    while true:
        point = int(ell) * point;
        if point.order() == 8:
            break
        point = curve25519.random_element()

    torsion8 = [(i * point) for i in range(1, 9)]

    return torsion8

```

```
def print_point(point):
    print "order = %d" % point.order()
    if point.order() == 1:
        print "Inf"
    else:
        u, v = point.xy()
        print "(%064x,\n %064x)" % (u, v)

def mont_to_edwards25519(point):
    if point.order() == 1:
        return (fp25519(0), fp25519(1))

    u, v = point.xy()
    if u == 0 and v == 0:
        return (fp25519(0), fp25519(-1))

    x = sqrt(fp25519(-486664)) * u / v
    y = (u - 1) / (u + 1)
    return (x, y)

edwards_G = mont_to_edwards25519(mont_G)
print "basepoint for curve25519\n%064x\n%064x" % (mont_G.xy())
print "basepoint for edwards25519\n%064x\n%064x" % (edwards_G)

mont_torsion8 = curve25519_8_torsion()

print "8-torsion of mont curve: curve25519"
for point in mont_torsion8:
    print_point(point)

ed_torsion8 = [mont_to_edwards25519(p) for p in mont_torsion8]
ed_torsion8_order = [p.order() for p in mont_torsion8]

print "8-torsion of edwards curve: edwards25519"
for i in range(0, len(ed_torsion8)):
    print "order = %d" % ed_torsion8_order[i]
    (x, y) = ed_torsion8[i]
    print "(%064x,\n %064x)" % (x, y)
```

$\mathcal{E}(\mathbb{F}_p)$ 与 $\mathcal{M}(\mathbb{F}_p)$ 之间的双向有理映射为:

$$(u, v) = \left(\frac{1+y}{1-y}, \sqrt{-486664}\frac{u}{x}\right), (x, y) = \left(\sqrt{-486664}\frac{u}{v}, \frac{u-1}{u+1}\right)$$

```
sage: load("curve25519.sage")
basepoint for curve25519
0000000000000000000000000000000000000000000000000000000000000009
5f51e65e475f794b1fe122d388b72eb36dc2b28192839e4dd6163a5d81312c14
basepoint for edwards25519
216936d3cd6e53fec0a4e231fdd6dc5c692cc7609525a7b2c9562d608f25d51a
6666666666666666666666666666666666666666666666666666666666666658
8-torsion of mont curve: curve25519
order = 8
(00b8495f16056286fdb1329ceb8d09da6ac49ff1fae35616aeb8413b7c7aebe0,
 46ce3ed6a9617c5ad6b7d3eb19d74ba86cc403d6127fe4b29778eb7c6daf84d3)
order = 4
(0000000000000000000000000000000000000000000000000000000000000001,
 6be4f497f9a9c2afc21fa77ad7f4a6ef635a11c7284a9363e9a248ef9c884415)
order = 8
(57119fd0dd4e22d8868e1c58c45c44045bef839c55b1d0b1248c50a3bc959c5f,
 173a6c76c2ba719bce3935ffba04afeadf5bbcb971559722f0efc7bdfb7f9a36)
order = 2
(0000000000000000000000000000000000000000000000000000000000000000,
 0000000000000000000000000000000000000000000000000000000000000000)
order = 8
(57119fd0dd4e22d8868e1c58c45c44045bef839c55b1d0b1248c50a3bc959c5f,
 68c593893d458e6431c6ca0045fb501520a443468eaa68dd0f103842048065b7)
order = 4
(0000000000000000000000000000000000000000000000000000000000000001,
 141b0b6806563d503de05885280b59109ca5ee38d7b56c9c165db7106377bbd8)
order = 8
(00b8495f16056286fdb1329ceb8d09da6ac49ff1fae35616aeb8413b7c7aebe0,
 3931c129569e83a529482c14e628b457933bfc29ed801b4d6887148392507b1a)
order = 1
Inf
8-torsion of edwards curve: edwards25519
order = 8
(602a465ff9c6b5d716cc66cdc721b544a3e6c38fec1a1dc7215eb9b93aba2ea3,
 7a03ac9277fdc74ec6cc392cfa53202a0f67100d760b3cba4fd84d3d706a17c7)
order = 4
(2b8324804fc1df0b2b4d00993dfbd7a72f431806ad2fe478c4ee1b274a0ea0b0,
 0000000000000000000000000000000000000000000000000000000000000000)
order = 8
(602a465ff9c6b5d716cc66cdc721b544a3e6c38fec1a1dc7215eb9b93aba2ea3,
 05fc536d880238b13933c6d305acdfd5f098eff289f4c345b027b2c28f95e826)
order = 2
(0000000000000000000000000000000000000000000000000000000000000000,
 7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffec)
order = 8
(1fd5b9a006394a28e933993238de4abb5c193c7013e5e238dea14646c545d14a,
 05fc536d880238b13933c6d305acdfd5f098eff289f4c345b027b2c28f95e826)
order = 4
(547cdb7fb03e20f4d4b2ff66c2042858d0bce7f952d01b873b11e4d8b5f15f3d,
 0000000000000000000000000000000000000000000000000000000000000000)
order = 8
(1fd5b9a006394a28e933993238de4abb5c193c7013e5e238dea14646c545d14a,
 7a03ac9277fdc74ec6cc392cfa53202a0f67100d760b3cba4fd84d3d706a17c7)
order = 1
(0000000000000000000000000000000000000000000000000000000000000000,
 0000000000000000000000000000000000000000000000000000000000000001)
```

Figure 1: 代码 Listing 1 的执行结果

其中 $(0,1)$ 为单位元, $(0,-1)$ 为 2 阶点, $(1/\sqrt{-1},0),(-1/\sqrt{-1},0)$ 为 4 阶点, 也即

$$\mathcal{E}(\mathbb{F}_p)[4]=\{(0,1),(0,-1),(1/\sqrt{-1},0),(-1/\sqrt{-1},0)\}.$$

RFC7748 中为 $\mathcal{E}(\mathbb{F}_p)$ 选中的基点 $G_{\mathcal{E}}$ 为:

$$\begin{aligned} x(G_{\mathcal{E}}) &= \texttt{0x216936d3cd6e53fec0a4e231fdd6dc5c692cc7609525a7b2c9562d608f25d51a}, \\ y(G_{\mathcal{E}}) &= \texttt{0x6666666666666666666666666666666666666666666666666666666666666658}. \end{aligned}$$

$\mathcal{E}(\mathbb{F}_p)$ 与 $\mathcal{M}(\mathbb{F}_p)$ 在代数结构上都同构于 $\mathbb{Z}_{h\ell}$, 也即两个点群上的 8-torsion $\mathcal{E}(\mathbb{F}_p)[8]$ 和 $\mathcal{M}(\mathbb{F}_p)[8]$ 均为循环子群, 则有

$$[2]\mathcal{M}(\mathbb{F}_p)[8]=\mathcal{M}(\mathbb{F}_p)[4],\ [2]\mathcal{E}(\mathbb{F}_p)[8]=\mathcal{E}(\mathbb{F}_p)[4].$$

Curve25519 和 Edwards25519 之间的关系可以通过 Listing 1 中展示的 Sage 代码验证. 代码的执行结果, 参见 Figure 1, 从中可以看到 $G_{\mathcal{M}}$ 经过函数 `mont_to_edwards25519` 变换之后可以得到 $G_{\mathcal{E}}$.

2 curve25519-dalek 简介

curve25519-dalek 对外公开的模块有 `scalar`, `montgomery`, `edwards`, `ristretto`, `constants`, `traits` 等. `scalar` 中实现了 $\mathbb{Z}_\ell^*$ 上的标量运算, `montgomery` 中实现了 Curve25519 上的仅依赖 u 坐标的点群运算, `edwards` 中实现了 Edwards25519 上的点群运算, `ristretto` 中通过封装 Edwards25519 上的点群实现了点群 ristretto255 上的运算以及编解码功能, `constants` 中则定义了服务于各个模块的常量. crate 内部模块 `field` 完成 $\mathbb{F}_p$ 上各种运算, 而具体的运算逻辑则是由 `backend` 实现的. `backend` 是可插拔的针对不同平台的实现的 $\mathbb{F}_p$ 内的运算, 例如当存在 SIMD 指令时, 可以切换到`SIMD`后端以加速 $\mathbb{F}_p$ 内的运算. 内部模块 `window`则提供了基于固定窗口 (Fixed-Window) 和滑动窗口 (Sliding-Window) 策略的点倍乘运算.

`scalar` 中主要定义了 `Scalar` 结构体, 用于实现 $\mathbb{Z}_\ell^*$ 上的标量运算. `Scalar` 内部用 32 字节小端法表示的数组存储 $\mathbb{Z}_\ell^*$ 上的元素. 为何 `scalar` 模块对外公开可用, 而 `field` 模块仅对 crate 内部公开? 这是因为在各种密码协议中, 私钥为 $\mathbb{Z}_\ell^*$ 中的元素, 而公钥为点群中的点, $\mathbb{F}_p$ 相关逻辑隐藏在了点运算内部, 因此库的用户通常无需关心 $\mathbb{F}_p$ 相关接口.

点群的通用接口定义在 `traits` 模块中, 其中抽象了 $\mathcal{M}(\mathbb{F}_p)$, $\mathcal{E}(\mathbb{F}_p)$ 以及 ristretto255 这 3 个点群可以支持的方法:

1. 获得群上的单位元 `Identity`, 判断是否为单位元 `IsIdentity`

2. 判断一个点是否在曲线上的函数 `ValidityCheck`.
3. 没有预计算的常量时间多标量乘法运算 `MultiscalarMul`,
4. 没有预计算且不保证常量时间的多标量乘法运算 `VartimeMultiscalarMul`,
5. 有预计算且不保证常量时间的多标量乘法运算 `VartimePrecomputedMultiscalarMul`:
   (a) 仅有固定点时多标量乘法: `vartime_multiscalar_mul`
   (b) 计算在仅有固定点时的多标量乘法 `vartime_multiscalar_mul`
   (c) 有动态点且不一定是合法时的多标量乘法 `optional_mixed_multiscalar_mul`

`traits` 模块中点群方法的实例化是在 `montgomery`, `edwards`, `ristretto` 模块中完成的.

`montgomery` 中定义了 $\mathcal{M}(\mathbb{F}_p)$ 上的点群 `MontgomeryPoint` 和相应运算, 如前所述该点群中的运算仅依赖 u 坐标系, 无穷远点的 $\mathcal{O}_\mathcal{M}$ 对应的 u 坐标为 0. `MontgomeryPoint` 结构体内部为小端法表示的 32 字节数组, 这是因为 $u \in \mathbb{F}_p, p < 2^{256}$. 因此借助 `as_bytes` 或者 `to_bytes` 可以查看 `MontgomeryPoint` 的值. 关于点的倍乘方面, `MontgomeryPoint` 仅用来实现 X25519 协议, 而该协议仅需要单点的倍乘运算, 因此 `montgomery`不支持 `traits` 中多标量乘法运算.

值得注意的是, 通过函数 `to_edwards` 可以将 `MontgomeryPoint` 转换成 `edwards` 模块中定义的 `EdwardsPoint`, 不是所有的 $u \in \mathbb{F}_p$ 都位于曲线 Curve25519 上 (可能位于 Curve25519 的二次扭曲线上[6]) 也即不是所有的 `MontgomeryPoint` 都可以转换成 `EdwardsPoint` , 因此 `to_edwards` 的返回值为 `Option<EdwardsPoint>`. 另外一个 `MontgomeryPoint` 可以映射成 2 个 `EdwardsPoint`, 可以通过输入参数指定具体要映射到哪个点. `to_edwards` 具体实现可以参考 Listing 2. 从 `MontgomeryPoint` 转换到 `EdwardsPoint`的具体运算是按照从 $\mathcal{M}(\mathbb{F}_p)$ 到 $\mathcal{E}(\mathbb{F}_p)$ 的映射 $y = (u-1)/(u+1)$ 进行计算的. 这就要求 $u \neq -1$, 又注意到 $u = -1$ 时, 根据 $\mathcal{M}(\mathbb{F}_p)$ 方程有: $v^2 = -1 + 486662 - 1 = 486660$, 然而 48000 是 $\mathbb{F}_p$ 上的二次非剩余, 也即 $u = -1$ 时对应的点位于 Curve25519 的二次扭曲线上, 此时 `to_edwards` 直接返回 `None`. `to_edwards` 是借助 `edwards` 模块中定义的另一个结构体 `CompressedEdwardsY`来完成的到 `EdwardsPoint` 的转换.

Listing 2: `MontgomeryPoint`的 `to_edwards` 实现

```
pub fn to_edwards(&self, sign: u8) -> Option<EdwardsPoint> {
    let u = FieldElement::from_bytes(&self.0);

```

[6] longcpp. 深入理解 X25519. https://github.com/longcpp/CryptoInAction/blob/master/intro-ed25519/190902-intro-x25519.pdf

```
    if u == FieldElement::minus_one() { return None; } // u = -1

    let one = FieldElement::one();

    let y = &(&u - &one) * &(&u + &one).invert(); // u = (u-1) / (u+1)

    let mut y_bytes = y.to_bytes();
    y_bytes[31] ^= sign << 7; // 最高位保存输入参数sign

    CompressedEdwardsY(y_bytes).decompress()
}
```

edwards模块中的 CompressedEdwardsY 结构体是 EdwardsPoint 的压缩表示形式: 仅用 $y \in \mathbb{F}_p$ 以及 x 坐标正负来表示一个点. 由于 $p < 2^{255}$, 所以用 32 字节的数字来存储 y 的值, 并且最高位可以用来存储 x 的正负号, 因此 CompressedEdwardsY 仅需要 32 个字节来存储. EdwardsPoint 可以通过 compress 方法转换为 CompressedEdwardsY, 参见 Listing 3. 为了理解 Listing 3, 需要解释下 EdwardsPoint 的定义. 为了提高运算效率, curve25519-dalek 中为 EdwardsPoint 选取的坐标系为扩展的扭曲爱德华坐标系的坐标表示, 仿射坐标系下的点 (x, y) 对应该坐标系下的点 $(X : Y : Z : T)$, 其中 $x = X/Z, y = Y/Z, xy = T/Z$. 可以注意到 Listing 2 和 Listing 3 对符号位的处理是一致的. CompressedEdwardsY通过 decompress 方法转换为 EdwardsPoint ，此处不再展示相应实现. EdwardsPoint 可以通过 to_montgomery 方法转换成 MontgomeryPoint.

Listing 3: EdwardsPoint的 compress 实现

```
/// Compress this point to `CompressedEdwardsY` format.
pub fn compress(&self) -> CompressedEdwardsY {
    let recip = self.Z.invert(); // Z^-1
    let x = &self.X * &recip; // x = X/Z
    let y = &self.Y * &recip; // y = Y/Z
    let mut s: [u8; 32]

    s = y.to_bytes();
    s[31] ^= x.is_negative().unwrap_u8() << 7; // 在最高位保存x的符号
    CompressedEdwardsY(s)
}
```

当基于余因子不为 1 的点群构建密码协议时, 常会在不经意间引入安全隐患.[7] 方

[7] longcpp. Edwards25519 余因子与双花交易. https://github.com/longcpp/CryptoInAction/blob/master/intro-ed25519/200212-edwards25519-cofactor.pdf

便起见, `edwards` 模块提供了工具函数 `is_small_order` 来判断点是否属于 $\mathcal{E}(\mathbb{F}_p)[8]$, 也提供了函数 `is_torsion_free` 来判断点是否属于 $\mathcal{E}(\mathbb{F}_p)[\ell]$, 也即 "torsion-free". 为了理解 "torsion-free" 的概念, 考虑

$$\mathcal{E}(\mathbb{F}_p) \cong \mathcal{E}(\mathbb{F}_p)[8] \times \mathcal{E}(\mathbb{F}_p)[\ell],$$

并且 $H \in \mathcal{E}(\mathbb{F}_p)[8]$, $G \in \mathcal{E}(\mathbb{F}_p)[\ell]$ 为子群的生成元, 则所有的 $P \in \mathcal{E}(\mathbb{F}_p)$ 都可以表示为 $P = xG + yH, x \in \mathbb{Z}_\ell, y \in \mathbb{Z}_8$. 当 P 为 "torsion-free" 时, 意味着 $P = xG + yH$ 中 $y = 0$. 注意到 $\mathsf{gcd}(8, \ell) = 1$, 则 "torsion-free" 也等价于 $\ell P = (0, 1)$(注意 $(0, 1)$ 为单位元).

`ristretto` 模块利用 Ristretto 技术从基于 $\mathcal{E}(\mathbb{F}_p)$ 萃取出素数阶 ℓ 的点群 ristretto255. Ristretto 技术的数学原理参见[8] 以及 curve25519-dalek 的文档[9]. 依赖蒙哥马利曲线, 扭曲爱德华曲线以及雅各比四次曲线之间的同源关系以及某些商群之间的同构关系的 Ristretto 技术原理比较繁杂, 但是可以简单的概述为点扭曲爱德华曲线上的点 $P \in [2]\mathcal{E}(\mathbb{F}_p)$, 经过扭转 (Torquing), 曲线之间的同源关系 (Isogeny) 以及商群 (Quotient Group) 之间的同构 (Isomorphism), 最终被编码为雅各比四次曲线 (Jacobi Quartic Curve) 点群 $\mathcal{J}(\mathbb{F}_p)$ 与其 2-torsion$\mathcal{J}(\mathbb{F}_p)[2]$ 的商群 $\mathcal{J}(\mathbb{F}_p)/\mathcal{J}(\mathbb{F}_p)[2]$ 的规范表示 (Canonical Representation). 理解 ristretto255 并不需要理解前面这句话. 相比繁杂的 Ristretto 技术原理, 基于 $\mathcal{E}(\mathbb{F}_p)$ 实现点群 ristretto255 的逻辑相对清晰. 值得指出的是, 虽然根据 Ristretto 技术原理, ristretto255 经过快速转换之后可以参与蒙哥马利阶梯算法, 但是 curve25519-dalek 中没有实现这一过程. 另外值得注意的是, ristretto255 点群仅处理了 $P \in [2]\mathcal{E}(\mathbb{F}_p)$ 的点, 这涵盖了 $\mathcal{E}(\mathbb{F}_p)[4] \times \mathcal{E}(\mathbb{F}_p)[\ell]$ 的点.

正如 Mike Hamberg 在 Decaf 论文中提到的 (Ristretto 技术是通过扩展 Decaf 技术而得到的), 基于一个点群实现萃取素数阶点群, 只需要重新定义点的编解码以及点相等的判断逻辑. 具体的点群运算, 可以将萃取出来的点群上的点转化为适当的点格式基于已有的实现完成运算. curve25519-dalek 中的实现正是如此. `ristretto` 模块定义了两种点格式 `RistrettoPoint` 以及 `CompressedRistretto`, 从两种结构体的定义中可以看到具体的 curve25519-dalek 中 ristretto255 的实现策略, 参见 Listing 4. 由于

$$\mathcal{E}(\mathbb{F}_p)[4] = \{(0, 1), (0, -1), (1/\sqrt{-1}, 0), (-1/\sqrt{-1}, 0)\},$$

所以对于任意的 `EdwardsPoint` $P = (x, y) \in \mathcal{E}(\mathbb{F}_p)[\ell]$, 根据 Ristretto 编码, 点 P 关于 $\mathcal{E}(\mathbb{F}_p)[4]$ 的陪集中的四个点

$$P + \mathcal{E}(\mathbb{F}_p)[4] = \{(x, y), (-x, -y), (y/\sqrt{-1}, -\sqrt{-1}x), (-y/\sqrt{-1}, \sqrt{-1}x)\}.$$

[8]longcpp. Ristretto: 萃取素数阶点群. `https://github.com/longcpp/CryptoInAction/blob/master/intro-ed25519/200324-intro-ristretto.pdf`

[9]The Ristretto Group`TheRistrettoGroup`

在 Ristretto 编码下会得到相同的编码值, 也即相同的 `CompressedRistretto`.

Listing 4: `RistrettoPoint` 和 `CompressedRistretto` 定义

```
/// A `RistrettoPoint` represents a point in the Ristretto group for
/// Curve25519.  Ristretto, a variant of Decaf, constructs a
/// prime-order group as a quotient group of a subgroup of (the
/// Edwards form of) Curve25519.
///
/// Internally, a `RistrettoPoint` is implemented as a wrapper type
/// around `EdwardsPoint`, with custom equality, compression, and
/// decompression routines to account for the quotient.  This means that
/// operations on `RistrettoPoint`s are exactly as fast as operations on
/// `EdwardsPoint`s.
///
#[derive(Copy, Clone)]
pub struct RistrettoPoint(pub(crate) EdwardsPoint);


/// A Ristretto point, in compressed wire format.
///
/// The Ristretto encoding is canonical, so two points are equal if and
/// only if their encodings are equal.
#[derive(Copy, Clone, Eq, PartialEq)]
pub struct CompressedRistretto(pub [u8; 32]);
```

Listing 4 中可以看到 `RistrettoPoint` 结构体的定义其实是 `EdwardsPoint` 的简单封装. 这意味着 `RistrettoPoint` 的点群运算实际上是通过 `EdwardsPoint` 的点群运算完成的. 将 `RistrettoPoint` 转换为 `CompressedRistretto` 也就完成了点群 ristretto255 中点的编码, 相应逻辑实现在 `RistrettoPoint` 的 `compress` 方法中. `compress` 方法在扩展的扭曲爱德华坐标系下, 严格按照 Ristretto 编码方式完成. 解码方式则实现在 `CompressedRistretto` 的 `decompress` 方法中. 2 个 `RistrettoPoint` 相等的判断在 `RistrettoPoint` 的 `ct_eq` 方法中实现, 参见 Listing 5, 相关的逻辑解释请参考[10].

Listing 5: `RistrettoPoint` 的 `ct_eq` 方法

```
impl ConstantTimeEq for RistrettoPoint {
    /// Test equality between two `RistrettoPoint`s.
    ///
    /// # Returns
```

[10] longcpp. Ristretto: 萃取素数阶点群. https://github.com/longcpp/CryptoInAction/blob/master/intro-ed25519/200324-intro-ristretto.pdf

```
    ///
    /// * `Choice(1)` if the two `RistrettoPoint`s are equal;
    /// * `Choice(0)` otherwise.
    fn ct_eq(&self, other: &RistrettoPoint) -> Choice {
        let X1Y2 = &self.0.X * &other.0.Y;
        let Y1X2 = &self.0.Y * &other.0.X;
        let X1X2 = &self.0.X * &other.0.X;
        let Y1Y2 = &self.0.Y * &other.0.Y;

        X1Y2.ct_eq(&Y1X2) | X1X2.ct_eq(&Y1Y2)
    }
}
```

值得提及的是, `ristretto` 模块内部还提供了 Elligator 映射, 该映射本质上实现了哈希到点群的逻辑. Elligator 映射不是外部接口, 但可经由外部接口调用. `RistrettoPoint::random()` 用来从随机数发生器生成 ristretto255 中的随机点, `RistrettoPoint::from_hash()`和 `RistrettoPoint::hash_from_bytes()` 完成哈希到点群的逻辑. 至此 `curve25519-dalek` 中最为关键的三种点群已经介绍完, 他们之间的逻辑关系参见 Figure 2.

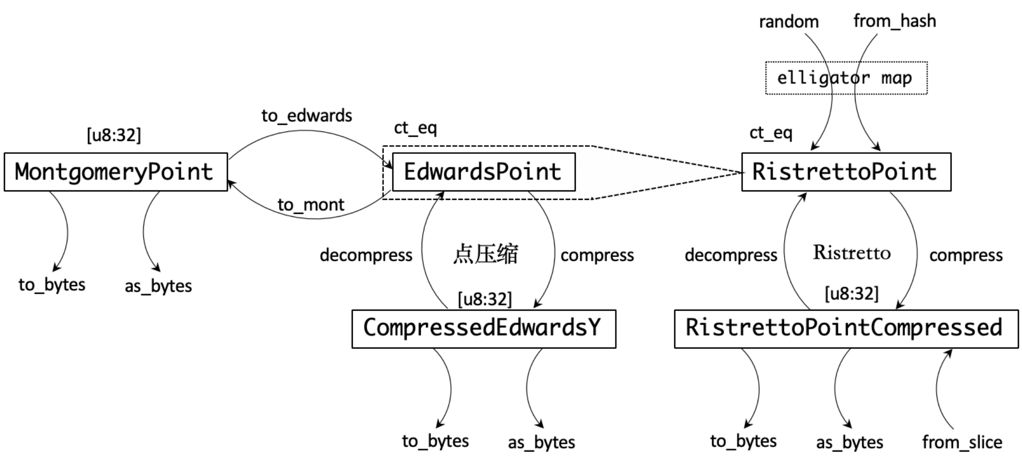

Figure 2: `curve25519-dalek`中点群间的关系

codeconstants 模块中定义了 3 个点群中的常量, 如 ℓ 用 `BASEPOINT_ORDER` 表示, `X25519_BASEPOINT` 是 X25519 协议选用的基点 $G_{\mathcal{M}}$, $\mathcal{E}(\mathbb{F}_p)$ 大素数阶子群 $\mathcal{E}(\mathbb{F}_p)[\ell]$ 的基点 $G_{\mathcal{E}}$ 及其压缩表示 `ED25519_BASEPOINT_POINT`, `ED25519_BASEPOINT_COMPRESSED`. `EdwardsPoint` 形式表示的 8 阶循环子群 $\mathcal{E}(\mathbb{F}_p)[8]$ 由 8 个元素的数组 `EIGHT_TORSION` 表示. 值得提及的是, `EIGHT_TORSION` 下标为 0,2,4,6 的 4 个 `EdwardsPoint` 记为 $\mathcal{E}(\mathbb{F}_p)[4]$. 由此可以理解 `RistrettoPoint` 的 `coset4` 方法返回的陪集, 参见 Listing 6. 点群 ristretto255 的基点

定义为 RISTRETTO_BASEPOINT_POINT, 其压缩表示形式为 RISTRETTO_BASEPOINT_COMPRESSED . 接下来介绍基于 curve25519-dalek 实现的 X25519, Ed25519 以及 sr25519 协议, 这 3 个协议分别用到了 MontgomeryPoint, EdwardsPoint 以及 RistrettoPoint.

Listing 6: RistrettoPoint 的 coset4 方法

```
/// Return the coset self + E[4], for debugging.
fn coset4(&self) -> [EdwardsPoint; 4] {
    [  self.0
    , &self.0 + &constants::EIGHT_TORSION[2]
    , &self.0 + &constants::EIGHT_TORSION[4]
    , &self.0 + &constants::EIGHT_TORSION[6]
    ]
}
```