Edwards25519 余因子与双花交易

longcpp

longcpp9@gmail.com

February 12, 2020

1 余因子为 8 的现实挑战与对策

蒙哥马利曲线 Curve25519 以及其双向有理等价的扭曲爱德华曲线 Edwards25519 因为其速度以及易于安全实现等特点, 在区块链领域内外逐渐被广泛采用, 例如基于 Curve25519 的 Diffie-Hellman 密钥交换协议 X25519 被 TLS 1.3 协议采纳, Monero 中则使用了基于 Edwards25519 的 Schnorr 签名机制, Tendermint Core 的共识投票过程则采用了基于 Edwards25519 的 EdDSA 签名机制 Ed25519. 对比基于 secp256k1/secp256r1 等 NIST 推荐曲线的 ECDH 协议/ECDSA/Schnorr 等签名机制, 基于 Curve25519/Edwards25519 的密钥交换协议或者签名机制在运算速度与实现安全等方面都胜出. 也因此很多工程项目倾向于基于 Curve25519/Edwards25519 构建新的密码学协议, 然而由于 Edwards25519 曲线参数余因子不为 1 的事实, 导致了 CryptoNote 协议中基于该曲线构建 RingCT 交易时引入了安全漏洞, 使得双花甚至多花成为可能, 影响了所有基于 CryptoNote 协议的数字货币, 例如 Monero, Bytecoin 等项目. 值得庆幸的是, 在该漏洞被利用之前 Monero 团队就填补了这个安全漏洞. 对比之下 Bytecoin 项目就没有这么幸运了, 有攻击者利用这一漏洞构建双花交易, 凭空创建了更多的数字货币.

广泛应用的 secp256r1/secp256k1 曲线的余因子为 1, 天然规避了与余因子相关联的安全隐患, 因此基于这两条曲线的 ECDH 或者签名机制通常无需考虑余因子的影响. 由于 Curve25519/Edwards25519 的余因子为 8, 在设计密码协议时必须将余因子不为 1 这个事实纳入考量. 例如 X25519 协议中解码点的倍乘中用到的标量参数时总是会将该标量的最低 3 比特清零, 而 Ed25519 签名机制中, 从种子 (Seed) 派生出私钥后也会将私钥的最低 3 比特清零之后再参与后续计算. 当椭圆曲线点群的余因子不为 1 时, 设计上层密码协议设计需要将底层点群的这一因素纳入考量, 并采取相应的措施. 对于复杂的密码协议, 这种方式是一个巨大的安全挑战, 这点可参考 CryptoNote 中发现的相关安全漏洞. 在上

层协议设计中, 不断加入为底层点群的数学结构采取防范措施, 也会使得协议的安全性难以论述. 另外在复杂的密码学协议设计中考虑应对措施本身也是一个技术挑战, 参考零知识证明系统 Bulletproof. 另外, 为了应对余因子的影响 Ed25519 签名算法的私钥空间不是连续的整数值, 也导致 Ed25519 签名机制适配分层钱包的机制时面临新的技术挑战.

是否存在方法能够既享用 Curve25519/Edwards25519 的优势, 又能够解放上层密码协议的设计? 答案是肯定的. Mike Hamberg 提出的 Decaf 方法[1] 能够在特定条件下从余因子为 4 的非素数阶的点群上" 萃取" 出素数阶的点群. 基于新的素数阶的点群, 余因子不为 1 相关的安全隐患与技术障碍都得以规避. Decaf 技术无需引入新的安全假设, 对既有的椭圆曲线点运算的实现的改动也很少. 然而 Decaf 技术无法直接应用于 Curve25519/Edwards25519, 因为此处的余因子为 8. Isis Agora Lovecruft 和 Henry de Valence 提出 Ristretto 技术通过扩展 Decaf 技术可以从余因子为 8 的非素数阶点群萃取出素数阶点群, 作用于 Curve25519 得到的素数阶点群记为 ristretto255. 由此上层协议可以安心利用曲线的各项优势而不再被余因子不为 1 的事实所羁绊. Polkadot 项目基于该技术实现了 sr25519 签名机制, MuSig 以及可验证随机函数 (Verifiable Random Function)[2], 其中 sr25519 表示基于 ristretto255 点群的 Schnorr 签名机制. 而由 Interstellar 赞助 Henry de Valence, Cathie Yun 和 Oleg Andreev 所完成的基于 Ristretto 技术所实现的 Bulletproof[3] (Rust 语言实现) 是目前效率最高的 Bulletproof 实现, 比基于 secp256k1 曲线的 Bulletproof 实现快 2 倍左右.

本次, 我们首先回顾 CryptoNote 中相关的安全漏洞, Monero 中的补救措施以及 Bytecoin 中的双花和多花; 关于 Ristretto 技术是如何在非素数阶的点群中抽象出素数阶点群, 以及 Ed25519 与 BIP32 等规范的分层钱包机制的适配, 留作后续讨论.

2 Monero 隐患与 Bytecoin 双花

Monero 是目前市值排名最高的提供链上隐私保护特性的数字货币 (根据 20191015 的 coinmarketcap.com 数据显示 Monero 以 9 亿美金的市值在所有种类的数字货币中排第 14 位, Dash 排 18 位, Zcash 排 30 位). 基于 UTXO 模型的 Monero 通过组合多项技术达到同时保护交易发起方, 接收方和交易金额. 利用 Diffie-Hellman 密钥交换协议为交易的每个输出都创建唯一的一次性地址 (One-Time Address/Stealth Address) 可以隐

[1] Mike Hamburg. Decaf: Eliminating cofactors through point compression
https://www.shiftleft.org/papers/decaf/decaf.pdf

[2] Polkadot Wiki. Polkadot Keys
https://wiki.polkadot.network/docs/en/learn-keys#account-keys

[3] A pure-Rust implementation of Bulletproofs using Ristretto.
https://github.com/dalek-cryptography/bulletproofs

藏交易的接收方; 利用 Pedersen 承诺及范围证明 (一开始基于 Borromean 环签名后升级为 Bulletproof 技术) 可隐藏交易金额; 利用可链接的环签名 (Linkable Spontaneous Anonymous Group signatures, LSAG, 包括 Back's LSAG 以及 Multilayer LSAG) 技术可以将交易的发起方隐藏在一个群组当中保护交易发起方的信息. 其中双花的预防是通过可链接特性来保证的, 具体来说是为用来签名的私钥计算 key image. 如果有两笔交易相关联的 key image 相同, 则意味着这两笔交易是由同一个私钥产生的, 而一次性地址使用导致每个可花费的 UTXO 对应的私钥值唯一, 因此两个相同的 key image 意味着这两笔交易是尝试对同一个 UTXO 的双花. 由于 Edwards25519 带来的速度与安全优势, Monero 中基于该曲线上的点群实例化了所需的密码协议, 包括 Schnorr 签名, Diffie-Hellman 协议, 环签名机制, Pedersen 承诺, 一次性地址及范围证明等. 值得指出的是, Monero 中没有使用基于 Edwards25519 曲线的 EdDSA 签名机制 Ed25519[4].

Edwards25519 曲线的采用旨在利用其速度与安全方面优势, 然而在基于该曲线构建上层密码协议时, 却由于 Edwards25519 曲线上的椭圆曲线点群的余因子 (Cofactor) 不为 1 的事实, 引发了安全漏洞, 考虑到 Edwards25519 曲线是 Bernstein 等人为了 EdDSA 签名机制而构造的椭圆曲线, Monero 中的漏洞也展示了密码学领域的经典错误: 为一个协议设计的底层组件在挪作他用时常在不经意间引发安全漏洞. 这个安全漏洞在所有基于 CryptoNote 协议的数字货币项目中均存在, 包括 Monero, Bytecoin, DashCoin 以及 DigitalNote 等. 2017 年 5 月在 Monero 项目通过硬分叉修正了自身的安全漏洞之后在网站上公布了安全漏洞细节[5]. 根据相关信息, 该安全漏洞是由 Monero 的研究人员在 17 年 2 月份参与 XEdDSA 签名机制[6]的细节讨论时发现的 (XEdDSA 签名机制是与 EdDSA 签名机制兼容的签名机制, 但是公私钥是 X25519 协议的形式, 我们后续专门讨论 X25519 与 Ed25519 之间共用同一个私钥的机制设计, 并讨论 XEdDSA 签名机制). 随后 Monero 项目扫描了所有的历史交易确认了该漏洞未被利用过 (由于漏洞的特性, 利用该漏洞的双花交易可以被甄别出来, 另外由于区块链自身的可溯源和不可篡改特性保证了所有历史交易信息可查询). Monero 在 17 年 02 月 22 号悄悄修正了漏洞, 并在随后通知了所有基于 CryptoNote 的同样受影响的项目, 虽然 Monero 由于运气和及时补救躲过一劫, 但是同样基于 CrypteNote 的 Bytecoin 项目就没有这么幸运, 同样的安全漏洞在 Bytecoin 项目中

[4] StackExchange: Why/how does monero generate public ed25519 keys without using the standard public key generation provided by ed25519 libraries? https://monero.stackexchange.com/questions/2290/why-how-does-monero-generate-public-ed25519-keys-without-using-the-standard-publ?rq=1

[5] Disclosure of a Major Bug in CryptoNote Based Currencies. 20170517. https://web.getmonero.org/2017/05/17/disclosure-of-a-major-bug-in-cryptonote-based-currencies.html

[6] Trevor Perrin (editor). The XEdDSA and VXEdDSA Signature Schemes. 20161020.https://signal.org/docs/specifications/xeddsa/

被利用凭空增发了大约 7 亿个 BCN (Bytecoin 项目的代币)[7]. 接下来我们讨论该安全漏洞的技术细节以及 Bytecoin 中的双花攻击.

由于数字货币领域广泛采纳基于 secp256k1 曲线实例化 ECDSA 签名机制或者 Schnorr 签名机制, 而 secp256k1 曲线上的椭圆曲线点群的余因子为 1 并且余因子为 1 的情形天然能够避免很多安全隐患, 因此 secp256k1 的余因子参数较少被讨论. 然而蒙哥马利曲线 Curve25519 及其双向有理等价的扭曲爱德华曲线 Edwards25519, 在带来速度和易于安全的优良特性之外, 却也带来了其上的椭圆曲线点群的余因子不为 1 的问题, 为诸多上层协议设计带来困扰, 也许终究没有任何东西是免费的.

定义在有限域 $\mathbb{F}_p, p = 2^{255} - 19$ 上的 Curve25519 或者 Edwards25519 的椭圆曲线点群的余因子为 8, 而大的素数子群 $\mathbb{G}_1$ 的阶 (Order) 为

$$\ell = 2^{252} + 27742317777372353535851937790883648493$$

则曲线上点群的阶为 $8 \cdot \ell$. 根据" 深入理解 X25519"[8]中的讨论, 该点群中存在阶为 8 的子群 $\mathbb{G}_2$, 而该子群本身内部又有阶为 4 和阶为 2 的子群, 也即 $\mathbb{G}_2$ 中的 8 个点包括单位元无穷远点, 1 个阶为 2 的点, 2 个阶为 4 的点以及 4 个阶为 8 的点. 记 $\mathbb{G}_1$ 的生成元为 G_1, $\mathbb{G}_2$ 的生成元为 G_2 ($\mathbb{G}_2$ 中的一个 8 阶点), 则曲线点群中任意的点 P 都可以表示为 $P = xG_1 + yG_2$, 其中 $x \in \mathbb{Z}_\ell, y \in \mathbb{Z}_8$. 对于任意的 $P_1 \in \mathbb{G}_1$ 都有 $\ell \cdot P_1 = \mathcal{O}$, 而对于任意的 $P_2 \in \mathbb{G}_2$ 都有 $8 \cdot P_1 = \mathcal{O}$, 也有 $8\ell \cdot P = \mathcal{O}$. Ed25519 签名机制定义在群 $\mathbb{G}_1$ 中, 而基于 Edwards25519 构建的上层密码协议为了保证安全性也应构建在群 $\mathbb{G}_1$ 上.

环签名技术可以将签名者的公钥隐藏在一组公钥集合当中从而保护签名者的身份. CryptoNote 中构建的环签名技术以及隐私交易 (Confidential Transactions) 都基于哈希函数. Monero 中环签名机制的环大小为 1 时, 环签名机制退化成为 Schnorr 签名机制, 这里指的重申一次, Monero 中只是利用了曲线 Edwards25519, 但没使用基于该曲线的 EdDSA 签名 Ed25519. 如前所述, Monero 中利用 key image 来确保在隐藏交易发起方的同时防止双花. 交易验证时对 key image 的验证逻辑是

$$R = r \cdot \mathsf{H}(P) + c \cdot I,$$

其中 $\mathsf{H}(P)$ 表示公钥 P 的哈希值, I 是 key image, 通过将公钥 P 哈希到一个点然后与私钥进行点的倍乘得到的点, c 是来自哈希函数的输出, 而 r 是签名者用来表示签名唯一的值. 如果能够找到另一个点 U 满足 $c \cdot I = c \cdot U$, 则上述等式的验证可以用 U 验证通过, 也

[7] Exploiting Low Order Generators in One-Time Ring Signatures. 20170523. https://jonasnick.github.io/blog/2017/05/23/exploiting-low-order-generators-in-one-time-ring-signatures/

[8] longcpp. 深入理解 X25519. 20190902. https://github.com/longcpp/CryptoInAction/blob/master/intro-ed25519/190902-intro-x25519.pdf

就意味着同一笔资金可以被花费两次. 根据前述的 Edwards25519 上点群的结构, 容易找到点 $U \notin \mathbb{G}_1$ 但仍满足 $c \cdot I = c \cdot U$. 如果 c 是 8 的倍数, 则对于任意的 $y \in \mathbb{G}_2, y \neq 0$, 以及 $U = I + yG_2$, 都有 $U \neq I, c \cdot I = c \cdot U$, 由于 $\mathbb{G}_2$ 的阶为 8, 则如果 c 是 8 的倍数, 则可以构造出 7 个不同的 U, 意味着此时可以把一笔资金花费 8 次 (1 次正常花费, 7 次双花). 双花安全漏洞可以通过对 key image 进行额外的检查进行避免: 检查 $\ell \cdot I$ 是否为无穷远点 $\mathcal{O}$, 若不是, 则拒绝该 key image. 原理在于, 如果 U 是通过前述方法构建出来的, 则有 $U = I + yG_2$, 意味着 $\ell \cdot U = \ell \cdot (I + yG_2) = \ell \cdot I + \ell yG_2 = (\ell y)G_2 \neq \mathcal{O}$. 至于代码层次的修正也比较容易, 在 2017 年 02 月 21 号的 Updates#1744[9] 中通过在函数 `bool core::check_tx_semantic(const transaction& tx, bool keeped_by_block)const` 中增加对 key image 的检查修正了这一漏洞, 参见 Listing **??**. 函数 的定义也展示在 Listing **??** 中, 也即检查 $\ell G_1 == \mathcal{O}$.

Listing 1: 检查 key image 的唯一性

```
// file: cryptonote_core.cpp
bool core::check_tx_semantic(const transaction& tx, bool keeped_by_block) const{
    // ....
    //check if tx use different key images
    if(!check_tx_inputs_keyimages_diff(tx))
    {
      MERROR_VER("tx uses a single key image more than once");
      return false;
    }
    // ...
}

// file: cryptonote_core.h
/**
 * @brief verify that each input key image in a transaction is unique
 *
 * @param tx the transaction to check
 *
 * @return false if any key image is repeated, otherwise true
 */
bool check_tx_inputs_keyimages_diff(const transaction& tx) const;

// file: cryptonote_core.cpp
bool core::check_tx_inputs_keyimages_diff(const transaction& tx) const
```

[9] Monero Project. Updates #1744. https://github.com/monero-project/monero/pull/1744/commits/d282cfcc46d39dc49e97f9ec5cedf7425e74d71f

```
25 {
26   std::unordered_set<crypto::key_image> ki;
27   for(const auto& in: tx.vin)
28   {
29     CHECKED_GET_SPECIFIC_VARIANT(in, const txin_to_key, tokey_in, false);
30     if(!ki.insert(tokey_in.k_image).second)
31       return false;
32   }
33   return true;
34 }
```

值得庆幸的是, Monero 项目在该漏洞被利用之前就修补了漏洞, 但是 Bytecoin 项目就没有这么幸运. 攻击者利用 Edwards25519 曲线余因子不为 1 的特性, 通过三花 62999999.98 个 BCN 和四花 188999999.98 个 BCN, 凭空增发了总计 692999999.9 个 BCN(大约为 7 亿个 BCN)[10]. 值得注意的是, 虽然同样是利用余因子不为 1 的特性, 但是 Bytecoin 上的三花和四花的具体过程与 Monero 项目所提及的漏洞利用过程有所不同. 在验证过程中, 会检查下面的方程是否成立

$$c = \mathsf{Hs}(m||r \cdot G_1 + c \cdot P||r \cdot \mathsf{Hp}(p) + c \cdot I),$$

其中 $I = x \cdot \mathsf{Hp}(P) \in \mathbb{G}_1$ 是 key image, x 是私钥, 而 $P = x \cdot G_1 \in \mathbb{G}_1$ 是公钥, 在合法交易构造过程中, 基于私钥 x 可以计算得到 c 和 r 使得上述等式成立. 然而在 Bytecoin 的双花攻击当中, 恶意的交易直接从 $\mathbb{G}_2$ 中选择了低阶点 $P, I \in \mathbb{G}_2$. 注意由于 $P \in \mathbb{G}_2$ 而基点 $G_1 \in \mathbb{G}_1$, 此时并不存在私钥 x *s.t.* $P = x \cdot G_1$. 接下来构造双花交易只需要搜索 r 的值, 使得

$$c = \mathsf{Hs}(m||r \cdot G_1||r \cdot \mathsf{Hp}(P)$$

的值为 $\mathbb{G}_2$ 的阶的倍数, 也即使得 c 为 8 的倍数, 就可以使得下面等式成立:

$$\mathsf{Hs}(m||r \cdot G_1 + c \cdot P||r \cdot \mathsf{Hp}(p) + c \cdot I) = \mathsf{Hs}(m||r \cdot G_1||r \cdot \mathsf{Hp}(P)) = c.$$

这是因为由于 $P, I \in \mathbb{G}_2$ 有 $c \cdot P = \mathcal{O}$ 并且 $c \cdot I = \mathcal{O}$.

为了进一步理解 Bytecoin 中基于低阶点完成的双花交易, 在 Listing **??** 中给出了 Edwards25519 的 8 个低阶点的具体值, 其中包括 1 个阶为 1 的点 (也即单位元 (0,1), 等价于蒙哥马利形式中的无穷远点 $\mathcal{O}$), 1 个阶为 2 的点, 2 个阶为 4 的点, 4 个阶为 8 的点.

Listing 2: Edwards25519 的低阶点

[10] How does the recent patched key image exploit work in practice? https://monero.stackexchange.com/questions/4241/how-does-the-recent-patched-key-image-exploit-work-in-practice

```
sage: fp25519 = FiniteField(2^255-19)
....: A, B = fp25519(486662), fp25519(1)
....: curve25519 = EllipticCurve(fp25519, [0, A, 0, B, 0])
....: P = curve25519.random_element()
....: ell = int(curve25519.cardinality() / 8)
....: while true:
....:     P = ell * P
....:     if P.order() == 8:
....:         break
....:     P = curve25519.random_point()
....:
....: mont_G2 = []
....: for i in range(1, 9):
....:     p = i * P
....:     mont_G2.append(p)
....:
....: def mont2edwards25519(u, v):
....:     x = sqrt(fp25519(-486664)) * (u / v)
....:     y = (u - 1) / (u + 1)
....:     return x, y
....:
....: for point in mont_G2:
....:     if point.order() == 1:
....:         x, y = fp25519(0), fp25519(1)
....:     else:
....:         u, v = point.xy()
....:         if u == 0 and v == 0:
....:             x, y = fp25519(0), fp25519(-1)
....:         else:
....:             x, y = mont2edwards25519(u, v)
....:     print "order = %i \n %064x \n %064x" % (point.order(), x, y)
....:
order = 8
 1fd5b9a006394a28e933993238de4abb5c193c7013e5e238dea14646c545d14a
 05fc536d880238b13933c6d305acdfd5f098eff289f4c345b027b2c28f95e826
order = 4
 2b8324804fc1df0b2b4d00993dfbd7a72f431806ad2fe478c4ee1b274a0ea0b0
 0000000000000000000000000000000000000000000000000000000000000000
order = 8
 1fd5b9a006394a28e933993238de4abb5c193c7013e5e238dea14646c545d14a
 7a03ac9277fdc74ec6cc392cfa53202a0f67100d760b3cba4fd84d3d706a17c7
order = 2
```

```
 0000000000000000000000000000000000000000000000000000000000000000
 7fffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffec
order = 8
 602a465ff9c6b5d716cc66cdc721b544a3e6c38fec1a1dc7215eb9b93aba2ea3
 7a03ac9277fdc74ec6cc392cfa53202a0f67100d760b3cba4fd84d3d706a17c7
order = 4
 547cdb7fb03e20f4d4b2ff66c2042858d0bce7f952d01b873b11e4d8b5f15f3d
 0000000000000000000000000000000000000000000000000000000000000000
order = 8
 602a465ff9c6b5d716cc66cdc721b544a3e6c38fec1a1dc7215eb9b93aba2ea3
 05fc536d880238b13933c6d305acdfd5f098eff289f4c345b027b2c28f95e826
order = 1
 0000000000000000000000000000000000000000000000000000000000000000
 0000000000000000000000000000000000000000000000000000000000000001
```

重点关注” 四花”188999999.98 个 BCN 的 4 笔交易, 这四笔花费同一个 UTXO 的交易可以在网站 https://minergate.com/blockchain/bcn/blocks 上检索 4 笔交易的交易 ID 查看, 4 个交易交易 ID 分别为:

1. cef289d7fab6e35ac123db8a3f06f7675b48067e0dff185c72b140845b8b3b23,
2. 7e418cc77935cc349f007cd5409d2b6908e4130321fa6f97ee0fee64b000ff85,
3. 5a3db49ef69e1f9dd9b740cabea7328cd3499c29fc4f3295bac3fa5e55384626,
4. 74298d301eb4b4da30c06989e0f7ff24a26c90bf4ffc4f2c18f34b7a22cf1136.

四笔交易所选择的 key image 分别为:

1. 26e8958fc2b227b045c3f489f2ef98f0d5dfac05d3c63339b13802886d53fc05,
2. 26e8958fc2b227b045c3f489f2ef98f0d5dfac05d3c63339b13802886d53fc85,
3. 0100000000000000000000000000000000000000000000000000000000000000,
4. ecffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff7f.

这 4 笔交易都成功花费了同一个 UTXO, 由交易

07a09e3c26d8ffc2e890713a69974e943a23ef6ad65b3bcbfc2b0f0da1add8f4

所生成的与低阶点相关的目标地址

26e8958fc2b227b045c3f489f2ef98f0d5dfac05d3c63339b13802886d53fc05.

也就印证了前述的利用低阶点进行 “双花” 甚至 “多花” 的攻击过程. 值得指出的是, Edwards25519 点的编码占用 32 个字节, 其中低 255 比特用来表示纵坐标 y 的值, 最高位比

特用来标记横坐标 x 的值是否为 $\mathbb{F}_p$ 中的负值 (也即 x 的最低比特) 具体信息参考 “深入理解 Ed25519: 原理与速度”[11]. 注意 Listing **??** 中是用大端法表示的值, 而 Bytecoin 的区块链浏览器中则是按照 Edwards25519 曲线的底层实现所采用的小端法表示的 256 比特的值. 注意 key image 1 和 2 的差别仅在于最高位比特是否为 1, 这是因为同一个 y 坐标可以对应的 2 个可能的 x 坐标, 其中必然有一个偶数一个奇数, 因此按照前述的编码规则, 会有一个点的编码的最高位被设置为 1.

已经看到 Bernstein 等人为了 X25519 密钥交换协议以及 Ed25519 签名机制而构造的曲线 Curve25519 和 Edwards25519 在挪作他用时会由于曲线的余因子不为 1 容易引入安全漏洞. 上层协议可以像 X25519 和 Ed25519 一样在设计协议时将余因子不为 1 的因素考虑在内, 但是这会使得上层协议的设计变得复杂并且安全性也不容易论证, 另外复杂的密码协议中再叠加一层应对余因子不为 1 的措施会增大协议设计的挑战. 复杂性是安全的大敌, 这种 ad-hoc 方式设计出来的密码协议的安全性也难以保证, 参考前文 CryptoNote 协议中出现的问题. Mike Hamberg 提出的 Decaf 方法为解决这一两难困境提出了解决方案. Decaf 方法在特定条件下可从余因子为 4 的点群中” 萃取” 中素数阶点群, 使得底层点群的问题在底层得到解决, 从而使得上层协议设计者无需再为底层的点群的余因子不为 1 的事实所担忧. 然而 Edwards25519 点群的余因子为 8, Decaf 并不能直接应用. Ristretto 技术通过扩展 Decaf 方法可以从余因子为 8 的点群中萃取中素数阶的点群.

后续我们探讨 Ristretto 技术如何通过代数变换与编解码进行大素数点群的萃取. Decaf 的字面意思为脱因咖啡, 而 Ristretto 字面意思则是升级版的 Espreesso. 与 Decaf 技术一样, Ristretto 技术也利用了雅各比四次形式 (Jacobi Quartic), 扭曲的爱德华形式 (Twisted Edwards), 以及蒙哥马利形式 (Montgomery) 三种形式的椭圆曲线以及这些形式之间的同源 (Isogeny).

[11] longcpp. 深入理解 Ed25519: 原理与速度. 20190930. https://github.com/longcpp/CryptoInAction/blob/master/intro-ed25519/190930-ed25519-theory-speed.pdf