深入理解 X25519

longcpp

longcpp9@gmail.com

September 2, 2019

1 X25519 设计理念概述

Curve25519 是 Bernstein 在 2006 年构建的蒙哥马利曲线, 其中 25519 表示曲线基于的素数域的特征为 $2^{255}-19$. 并在该曲线上构建了仅基于 x 坐标的 ECDH 密钥协议 X25519. 仅利用椭圆曲线点的 x 坐标构建 ECDH 的想法最初来自于 Victor Miller 在 1985 年发表的奠基性文章"Use of elliptic curves in cryptography"[1]. 在仅依赖 x 坐标的 ECDH 协议中, Alice 计算 $(a,\mathbf{x}(P))\rightarrow\mathbf{x}(aP)$ 之后, 将 $\mathbf{x}(aP)$ 传递给 Bob, Bob 计算 $(b,\mathbf{x}(P))\rightarrow\mathbf{x}(bP)$, $\mathbf{x}(bP)$ 传递给 Alice, 其中 a 和 b 是 Alice 和 Bob 各自生成的临时私钥, P 为曲线上的一个点. Alice 收到的 Bob 的消息之后计算 $(a,\mathbf{x}(bP))\rightarrow\mathbf{x}(abP)$, 同样 Bob 在收到 Alice 的消息之后, 计算 $(b,\mathbf{x}(aP))\rightarrow\mathbf{x}(baP)$. 这也是 Bernstein 的基于 Curve25519 曲线的 X25519 的密钥交换协议的基础. 初次接触仅基于 x 坐标系的协议总会感觉有些诡异, 然而考虑到根据 x 坐标和曲线方程, 基本上可以完全确定出一个点, 例外在于点 $P=(x,y)$ 和 $-P=(x,-y)$ 会有相同的 x 坐标. 通过构建椭圆曲线点群关于逆映射的商群 (Quotient Group)$E/\langle-\rangle$, 则在商群的视角下, 点的 x 坐标可以唯一表示商群中的元素. 到商群 $E/\langle-\rangle$ 的映射 $\mathbf{x}$ 为

$$\mathbf{x}:P\rightarrow\begin{cases}(x_P:1) & \text{if } P=(x_P:y_P:1)\\ (1:0) & \text{if } P=\mathcal{O}=(0:1:0)\end{cases},\tag{1}$$

需要指出的是, 写法 $\mathbf{x}((X:Y:Z))=(X:Z)$ 只有当 $Z\neq 0$ 时成立, 对于无穷远点 $\mathcal{O}$ 不成立. 对于上述映射 $\mathbf{x}$ 对于点直观上理解, 点 P 和点 $-P$ 在商群中塌缩成同一个元素: $\mathbf{x}(P)=\mathbf{x}(-P)$. 另外点 $P,-P$ 的 k 倍乘运算的结果 $kP,k(-P)$ 在商群 $E/\langle-\rangle$ 中也坍缩

[1] Miller, Victor S. "Use of elliptic curves in cryptography." In Conference on the theory and application of cryptographic techniques, pp. 417-426. Springer, Berlin, Heidelberg, 1985. https://link.springer.com/content/pdf/10.1007/3-540-39799-X_31.pdf

成同一个元素:

$$k(-P) = -kP \implies \mathbf{x}(k(-P)) = \mathbf{x}(-kP) = \mathbf{x}(kP).$$

至此可以理解仅基于 x 坐标可以构建 ECDH 协议, 因为其中仅涉及到群中的倍乘运算.

然而在给定 $\mathbf{x}(P)$ 和 k 的条件下, 如何高效计算 $\mathbf{x}(kP)$ 并不显然. 在各种形式表示的椭圆曲线的点群上都可以构建仅基于 x 坐标的运算, 但是利用蒙哥马利曲线上的点群的结构 (具体来说是点群中 4 阶点的存在) 配合蒙哥马利阶梯算法更容易完成高效的常量时间实现. 摘录 Bernstein 等人在文章 "Montgomery curves and the Montgomery ladder"[2]的表述: *The Montgomery ladder is much simpler and almost three times faster. The structure of Montgomery curves is important for this simplicity and speed: from the modern Edwards perspective, Montgomery takes advantage of having a point of order 4 on the curve or its twist.* 这也是 Bernstein 选择蒙哥马利形式曲线的原因. 在这些原因之外, X25519 的设计也考虑到了 ECDH 的实际部署中的可能遇到的问题并基于对蒙哥马利曲线的观察进行了巧妙的规避, 简化了 X25519 的实现和应用.

ECDH 的实际部署中需要着重考虑的是对接收到的消息的检查. 在典型的 ECDH 协议中, 参与协议的双方都会收到对方发来的临时公钥点, 为了保证安全 (例如确保自己的私钥信息不会泄露), 需要首先检查收到的点的合法性, 因为如果收到的点是对方刻意构造的点, 则对方有可能通过 ECDH 协议的交互过程窃取私钥信息. 对点的检查通常要验证收到的点不是无穷远点, 点的坐标是底层素数域的中的元素并且确实是椭圆曲线方程上的点. 如果没有确认收到的点不是无穷远点, 会引发 small subgroup 攻击; 如果不确认收到的点确实在椭圆曲线上, 则会招致 invalid-curve 攻击, 参见 "Guide to elliptic curve cryptography"[3]中 4.3 小节. 类似的问题也存在于仅基于 x 坐标的 ECDH 密钥交换协议当中. 另外仅利用 x 坐标的另一问题在于验证 "点在椭圆曲线上" 会更困难一些. 检查完整的点只需要把横纵坐标带入方程然后验证返程是否成立即可. 在仅基于 x 坐标的 ECDH 协议中, 要判定给定的 x 值是否合法等价于判定 $x^3 + Ax^2 + x$ 是否是二次剩余. 由于域上的平方运算是 2-to-1 的映射, x 的所有可能取值大概只有一半的 x 是合法的. 这也是 X25519 的另一个创新之处: 将二次扩域上的椭圆曲线 $E(\mathbb{F}_{p^2}), \mathbb{F}_{p^2} = \mathbb{Z}_{2^{255}-19}[\sqrt{2}]$ 的点群纳入考量. 同时考虑两个点群 $E(\mathbb{F}_p)$ 和 $E(\mathbb{F}_{p^2})$, 则任意的 $x \in \mathbb{F}_p$ 都可以对应到某个点群上的点, 则前述的需要判定二次剩余来判定 x 是否是合法值的步骤可以省略, 另外由于 X25519 仅依赖 x 坐标, 所以虽然引入了 $E(\mathbb{F}_{p^2})$, 但 X25519 所有的运算仍是在域 $\mathbb{F}_p$

[2]Bernstein, Daniel J., and Tanja Lange. "Montgomery curves and the Montgomery ladder." IACR Cryptology ePrint Archive 2017 (2017): 293. https://eprint.iacr.org/2017/293.pdf

[3]Hankerson, Darrel, Alfred J. Menezes, and Scott Vanstone. "Guide to elliptic curve cryptography." Computing Reviews 46, no. 1 (2005): 13.

上完成的. 这也是除了仅依赖 x 坐标之外, X25519 相比其他的 ECDH 过程, 个人认为最迥异的地方: 为了使所有的 $x \in \mathbb{F}_p$ 都是合法的公钥从而省去繁重的公钥合法性验证, 引入了扩域上的椭圆曲线点群. 只要在两个点群上的离散对数问题都是困难的, X25519 密钥交换协议就是安全的.

2 深入理解 Curve25519 与 X25519

蒙哥马利形式的椭圆曲线 Curve25519 和 Curve448 被收录在 RFC 7748 中, 其中 Curve448 是 Mike Hamburg 在 2015 年设计的新曲线, 旨在提供 224 比特的安全性[4]. RFC 7748 也同时给出了基于两条椭圆曲线的仅依赖 x 坐标的 ECDH 密钥交换协议 X25519 和 X448. 本文重点关注 Curve25519 及相应的密钥交换协议 X25519. Curve25519 是定义在有限域 $\mathbb{F}_p, p = 2^{255} - 19$ 的蒙哥马利形式椭圆曲线 $y^2 = x^3 + 486662x^2 + x$, 其中曲线上点的个数 $\#E(\mathbb{F}_p) = 8 \cdot n_1$, 也即余因子为 8, 并且有

$$n_1 = \texttt{0x1000000000000000000000000000000014def9dea2f79cd65812631a5cf5d3ed},$$

Listing 1: Curve25519 的 Sage 示例

```
sage: fp25519 = FiniteField(2^255-19)
sage: A = fp25519(486662)
sage: B = fp25519(1)
sage: curve25519 = EllipticCurve(fp25519, [0, A, 0, B, 0])
sage: card = curve25519.cardinality()
sage: hex(int(card))
'0x8000000000000000000000000000000a6f7cef517bce6b2c09318d2e7ae9f68L'
sage: n1 = int(card / 8)
sage: hex(n1)
'0x1000000000000000000000000000000014def9dea2f79cd65812631a5cf5d3edL'
sage: curve25519.abelian_group()
Additive abelian group isomorphic to
Z/57896044618658097711785492504343953926856930875039260848015607506283634007912
embedded in Abelian group of points on Elliptic Curve defined by
y^2 = x^3 + 486662*x^2 + x over Finite Field of size
57896044618658097711785492504343953926634992332820282019728792003956564819949
sage: def print_point(p):
....:     if p.order() == 1:
....:         print "point at infinity"
....:     else:
```

[4]Hamburg, Mike. Ed448-Goldilocks, a new elliptic curve. IACR Cryptology ePrint Archive 2015 (2015): 625. https://eprint.iacr.org/2015/625.pdf

```
....:         x, y = p.xy()
....:         print "(%064x,\n %064x)" % (x, y)
....:
sage: G = curve25519.lift_x(9)
sage: print_point(G)
(0000000000000000000000000000000000000000000000000000000000000009,
 5f51e65e475f794b1fe122d388b72eb36dc2b28192839e4dd6163a5d81312c14)
sage: print_point(-G)
(0000000000000000000000000000000000000000000000000000000000000009,
 20ae19a1b8a086b4e01edd2c7748d14c923d4d7e6d7c61b229e9c5a27eced3d9)
sage: P = curve25519.random_point()
sage: while true:
....:     P = n1 * P
....:     if P.order() == 8:
....:         break
....:     P = curve25519.random_point()
....:
sage: print_point(P)
(57119fd0dd4e22d8868e1c58c45c44045bef839c55b1d0b1248c50a3bc959c5f,
 173a6c76c2ba719bce3935ffba04afeadf5bbcb971559722f0efc7bdfb7f9a36)
sage: for i in range(1,9):
....:     p = i * P
....:     print "order = %d" % p.order()
....:     print_point(p)
....:
order = 8
(57119fd0dd4e22d8868e1c58c45c44045bef839c55b1d0b1248c50a3bc959c5f,
 173a6c76c2ba719bce3935ffba04afeadf5bbcb971559722f0efc7bdfb7f9a36)
order = 4
(0000000000000000000000000000000000000000000000000000000000000001,
 141b0b6806563d503de05885280b59109ca5ee38d7b56c9c165db7106377bbd8)
order = 8
(00b8495f16056286fdb1329ceb8d09da6ac49ff1fae35616aeb8413b7c7aebe0,
 46ce3ed6a9617c5ad6b7d3eb19d74ba86cc403d6127fe4b29778eb7c6daf84d3)
order = 2
(0000000000000000000000000000000000000000000000000000000000000000,
 0000000000000000000000000000000000000000000000000000000000000000)
order = 8
(00b8495f16056286fdb1329ceb8d09da6ac49ff1fae35616aeb8413b7c7aebe0,
 3931c129569e83a529482c14e628b457933bfc29ed801b4d6887148392507b1a)
order = 4
(0000000000000000000000000000000000000000000000000000000000000001,
```

```
63  6be4f497f9a9c2afc21fa77ad7f4a6ef635a11c7284a9363e9a248ef9c884415)
64 order = 8
65 (57119fd0dd4e22d8868e1c58c45c44045bef839c55b1d0b1248c50a3bc959c5f,
66  68c593893d458e6431c6ca0045fb501520a443468eaa68dd0f103842048065b7)
67 order = 1
68 point at infinity
```

根据上述 Sage 示例代码, 可以对定义在 $\mathbb{F}_p$ 上的 Curve25519 椭圆曲线点集有更好的理解. `curve25519.abelian_group()` 的执行结果显示, 该点集同构于 $\mathbb{Z}_{8n_1}$, 意味着 $E(\mathbb{F}_p)$ 中存在阶为 2, 4, 8 点. 阶为 2 的点根据曲线方程显然为 $(0,0)$. 随后的代码通过随机选取椭圆曲线上的点 P, 并判定 n_1P 的阶是否为 8 来寻找阶为 8 的点, 由于点集 $E(\mathbb{F}_p)$ 同构于 $\mathbb{Z}_{8n_1}$, 则会有一个 8 阶的子群, 一个 n_1 阶的子群, 8 阶子群中有 1 个 1 阶点 (也即无穷远点), 1 个 2 阶点 (也即点 $(0,0)$), 2 个 4 阶点,4 个 8 阶点. 每次从点集 $E(\mathbb{F}_p)$ 随机取点 P, 则点 n_1P 的阶为 8 的概率为 1/2, 也即 Listing 1 中第 32 至第 37 行的代码在几次尝试之后即可完成. 有了 8 阶点之后, Listing 1 中第 41 至 45 行输出 8 阶子群中所有点和点的阶. 关于基点 G, RFC 7748 中一开始给出的 X25519 采用的基点 G 为 (`0x9, 0x20ae19a1b8a086b4e01edd2c7748d14c923d4d7e6d7c61b229e9c5a27eced3d9`). 然而在随后的 RFC 7748 的勘误[5] 中将基点 G 的具体值修正为 (`0x9, 0x5f51e65e475f794b1fe122d388b72eb36dc2b28192839e4dd6163a5d81312c14`). 这是因为, X25519 所依赖的椭圆曲线点群运算只涉及点的横坐标, 所以 X25519 涉及的运算只关心横坐标. 然而由于 Curve25519 与 Edwards25519 双向有理等价, 而 Ed25519 所依赖的点群运算同时需要横纵坐标, 并且已经有广泛使用的基点的值. RFC 7748 中给出的基点的值, 会映射到 Edwards25519 时会映射成 Edwards2519 所依赖的基点的负值, 因此有了上述修正, 以便在双向有理映射的条件下保持 Edwards25519 和 Curve25519 的基点保持一致, 参见 Listing 1 中第 24 至第 30 行.

X25519 的设计中为了使得所有的 $x \in \mathbb{F}_p$ 都是合法公钥值引入了二次扩域上的椭圆曲线点群 $E(\mathbb{F}_{p^2})$:

$$E(\mathbb{F}_{p^2}) = \{\mathcal{O}\} \cup \{(x,y) : y^2 = x^3 + Ax^2 + x, x, y \in \mathbb{F}_{p^2}, \mathbb{F}_{p^2} = \mathbb{Z}_{2^{255}-19}[\sqrt{2}]\}.$$

定义 $E(\mathbb{F}_{p^2})$ 上的取反操作 $-$ 为: $-\mathcal{O} = \mathcal{O}, -(x,y) = (x,-y)$. $E(\mathbb{F}_{p^2})$ 上加法操作 $+$ 定义如下:

- $\mathcal{O} + \mathcal{O} = \mathcal{O}, \mathcal{O} + (x,y) = (x,y), (x,y) + \mathcal{O} = (x,y), (x,y) + (x,-y) = \mathcal{O}$

[5] RFC 7748 Errata. https://www.rfc-editor.org/errata/rfc7748

- $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$,

$$\begin{cases} x_3 &= \lambda^2 - (x_1 + x_2) - A \\ y_3 &= (2x_1 + x_2 + A)\lambda - \lambda^3 - y_1 = \lambda(x_1 - x_3) - y_1 \end{cases},$$

其中,

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1), \text{ if } x_1 \neq x_2, \\ (3x_1^2 + 2Ax_1 + 1)/(2y_1), \text{ if } (x_1, y_1) = (x_2, y_2), \end{cases}$$

$E(\mathbb{F}_p)$ 显然是 $E(\mathbb{F}_{p^2})$ 中的子群 (Subgroup):

$$E(\mathbb{F}_p) = \{\mathcal{O}\} \cup \{E(\mathbb{F}_{p^2}) \cap \mathbb{F}_p \times \mathbb{F}_p\},$$

根据前述的加法运算规则可知, 当 $(x_1, y_1), (x_2, y_2) \in \mathbb{F}_p \times \sqrt{2}\mathbb{F}_p$ 时, $\lambda \in \sqrt{2}\mathbb{F}_p$, 根据运算规则有 $(x_3, y_3) \in \mathbb{F}_p \times \sqrt{2}\mathbb{F}_p$. 则有 $(x_3, y_3) \in \mathbb{F}_p \times \sqrt{2}\mathbb{F}_p$, 所以 $E(\mathbb{F}_{p^2})$ 中的另一个子群为

$$E'(\mathbb{F}_p) = \{\mathcal{O}\} \cup \{E(\mathbb{F}_{p^2}) \cap \mathbb{F}_p \times \sqrt{2}\mathbb{F}_p\}.$$

同时考虑 $E(\mathbb{F}_p)$ 和 $E'(\mathbb{F}_p)$, 则任意的 $x \in \mathbb{F}_p, x \neq 0$ 都对应两个点. 值得注意的是, 这两个子群的交集 $\{\mathcal{O}, (0,0)\}$ 也是 $E(\mathbb{F}_{p^2})$ 的子群. 如前所述, 对于所有的 $x \in \mathbb{F}_p$, $E(\mathbb{F}_p)$ 和 $E'(\mathbb{F}_p)$ 中都有 2 个点的横坐标等于 x, 例外在于当 $x = 0$ 的情况, 只有点 $(0,0)$ 的横坐标为零. 由于 $E(\mathbb{F}_p)$ 和 $E'(\mathbb{F}_p)$ 中都包含无穷远点 $\mathcal{O}$, 则 $\#E(\mathbb{F}_p) + \#E'(\mathbb{F}_p) = 2(p-1) + 2 + 2 = 2p + 2$, 第一次 '+2' 是因为两个集合中均有点 $(0,0)$, 第二次 '+2' 是因为两个集合中均有的无穷远点 $\mathcal{O}$. 则 $\#E'(\mathbb{F}_p) = 2p + 2 - 8n_1 = 4n_2$, 其中 $n_2 =$ `0x1fffffffffffffffffffffffffffffffd6420c42ba10c6534fdb39cb4614581d`.

事实上 $E'(\mathbb{F}_p)$ 可以看做是与 $E(\mathbb{F}_p)$ 在 $\mathbb{F}_{p^2}$ 上同构的二次扭曲线 (Quadratic Twist) 上的点群. 蒙哥马利曲线 $E^M_{A,B}$ 中的两个参数 A, B 中, 更重要的是参数 A, 因为 $E^M_{A,B}$ 的 j-不变量 (j-invariant) 为:

$$j(E^M_{A,B}) = \frac{256(A^2 - 3)^3}{A^2 - 4}$$

也即蒙哥马利曲线的 j-不变量完全由参数 A 定义. 而参数 B 可以被看做是扭曲因子 (Twisting Factor): 对于 $B' \in \mathbb{F}_p, B' \neq 0$, 则通过符号代换 $(x, y) \to (x, \sqrt{B/B'}y)$ 有 $E^M_{A,B} \cong E^M_{A,B'}$. 当 B/B' 是 $\mathbb{F}_p$ 上的二次剩余时, 该同构 (Isomorphism) 是定义在 $\mathbb{F}_p$ 上的, 否则 $E^M_{A,B'}$ 是 $E^M_{A,B}$ 的二次扭曲线, 也即两个曲线在 $\mathbb{F}_{p^2}$ 中是同构的. 对于 Curve25519 曲线有 $B = 1$, 根据前述定义 $\mathbb{F}_{p^2} = \mathbb{Z}_p[\sqrt{2}]$, 有 $B' = 2$, 注意到 $1/2$ 不是 $\mathbb{F}_p$ 上的二次剩余, 所以 $E^M_{A,B'}$ 是与 $E^M_{A,B}$ 在 $\mathbb{F}_{p^2}$ 上同构的二次扭曲线, 而前述的 $E'(\mathbb{F}_p)$ 也可以看做是 $E^M_{A,B'}$ 在 $\mathbb{F}_{p^2}$ 上的子群.

同时考虑 $E(\mathbb{F}_p)$ 和 $E'(\mathbb{F}_p)$, 则对于任意的元素 $q \in \mathbb{F}_p$, 都存在点 $Q \in E(\mathbb{F}_{p^2})$ ($Q \in E(\mathbb{F}_p) \cup E'(\mathbb{F}_p)$) 满足其横坐标值为 q. 基于此, Bernstein 证明了映射 $\mathbf{x_0} : E(\mathbb{F}_{p^2}) \to \mathbb{F}_{p^2}$:

$$\begin{cases} \mathbf{x_0}(\mathcal{O}) &= 0 \\ \mathbf{x_0}((x,y)) &= x \end{cases} \tag{2}$$

对于任意的整数 n, 和所有的点 $Q \in E(\mathbb{F}_{p^2}), \mathbf{x_0}(Q) = q, q \in \mathbb{F}_p$, 都存在唯一的 $s \in \mathbb{F}_p$ 使得 $\mathbf{x_0}(nQ) = s$. 这个结论也为构建仅依赖 x 坐标的 X25519 密钥交换协议铺平了道路. 简单陈述下证明过程. 对于 $q \in \mathbb{F}_p$, 计算 $\alpha = q^3 + Aq^2 + q$, 则需要对 α 的值是否为零, 是否是二次剩余以及是否是二次非剩余的情况进行讨论.

- $\alpha = 0$ 时, 意味着 $q = 0$, 由于 $0^2 = 0$, 因此集合 $\{Q \in E(\mathbb{F}_{p^2}) : \mathbf{x_0}(Q) = 0\} = \{\mathcal{O}, (0,0)\}$, 因此对于任意的 $Q \in \{\mathcal{O}, (0,0)\}$ 都有 $nQ \in \{\mathcal{O}, (0,0)\}$, 也即 $\mathbf{x_0}(nQ) = 0$.

- α 不为零且为 $\mathbb{F}_p$ 上的二次剩余时, 则 $q(q^2 + Aq + 1) \neq 0 \implies q \neq 0$. 用 r 表示 α 的平方根, 则 $\mathbb{F}_{p^2}$ 上的 α 的平方根只有 $\pm r$, 则集合 $\{Q \in E(\mathbb{F}_{p^2}) : \mathbf{x_0}(Q) = q\} = \{(q,r), (q,-r)\}$. 记 $s = \mathbf{x_0}(n(q,r))$, 由于 $(q,r) \in E(\mathbb{F}_p)$, 则有 $n(q,r) \in E(\mathbb{F}_p)$, 则 $s \in \{1, 2, \ldots, p-1\}$. 另外 $n(q,-r) = n(-(q,r)) = -(n(q,r)) \implies \mathbf{x_0}(n(q,-r)) = \mathbf{x_0}(n(q,r)) = s$. 因此对于所有的满足 $\mathbf{x_0}(Q) = q$ 的点 $Q \in E(\mathbb{F}_{p^2})$, 都有 $\mathbf{x_0}(nQ) = s$.

- α 不为零且为 $\mathbb{F}_p$ 上的二次非剩余时, 同样有 $q \neq 0$. 由于 2 也是 $\mathbb{F}_p$ 上的二次非剩余, 则 $\alpha/2$ 是 $\mathbb{F}_p$ 上的二次剩余, 记相应的二次根为 r, 则 $\mathbb{F}_{p^2}$ 上的二次根只能为 $\pm r$, 因此 $\{Q \in E(\mathbb{F}_{p^2}) : \mathbf{x_0}(Q) = q\} = \{(q, r\sqrt{2}), (q, -r\sqrt{2})\}$. 记 $s = \mathbf{x_0}(n(q, r\sqrt{2}))$, 由于 $(q, r\sqrt{2}) \in E'(\mathbb{F}_p)$, 则有 $n(q, r\sqrt{2}) \in E'(\mathbb{F}_p)$, 则 $s \in \{1, 2, \ldots, p-1\}$. 另外 $n(q, -r\sqrt{2}) = n(-(q, r\sqrt{2})) = -(n(q, r\sqrt{2})) \implies \mathbf{x_0}(n(q, -r\sqrt{2})) = \mathbf{x_0}(n(q, r\sqrt{2})) = s$. 因此对于所有的满足 $\mathbf{x_0}(Q) = q$ 的点 $Q \in E(\mathbb{F}_{p^2})$, 都有 $\mathbf{x_0}(nQ) = s$.

上述结论证实了在二次扩域视角下, 仅依赖 x 坐标也可以进行倍乘运算, 也因此仅依赖二次扩域上的椭圆曲线上点的 x 坐标以及运算可以构建 ECDH 密钥交换协议, 参见 Figure 1 中展示的 Alice 和 Bob 之间的 X25519 密钥协商过程. 值得注意的是, 虽然概念层面上引入了二次扩域但是由于仅依赖 x 坐标, 所以倍乘运算还是在 $\mathbb{F}_p$ 上完成的. Figure 1 中 'Public string 9' 表示 X25519 的公共参数基点 G 横坐标 9 的字节数组. 值得提及的是, 在 X25519 和 Ed25519 的设计文档中, 也同时给出了对元素进行编解码的规定. 定义在 $\mathbb{F}_p$ 上的坐标, 以小端法编码为字节数组, 也即坐标值 $x \in \mathbb{F}_p$ 与 $x[0] + 256 * x[1] + \ldots + 256^{n-1} * x[n-1]$ 模 p 同余. 从 32 个字节的数组解码坐标值

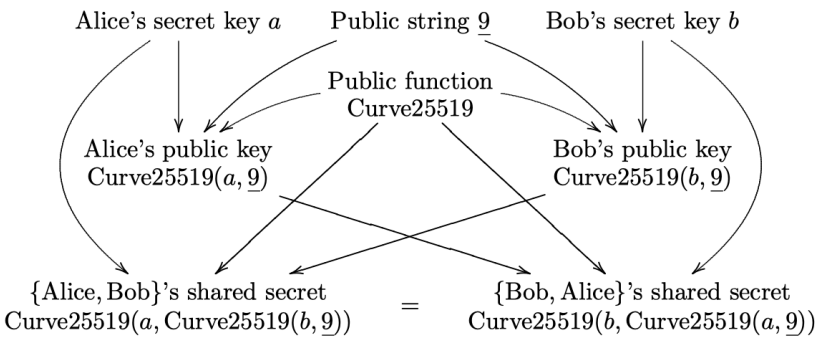

Figure 1: X25519 密钥交换协议

时, 对于 X25519 协议需要首先将最后一个字节的最高比特位清零, 参见 Listing 2 中第 7 行, 该操作是为了忽略未使用的最高位的比特. 另外, 输入参数 `bits` 的值为 255(对应 Curve25519) 或者 448(对应 Curve448). 另外 RFC 7748 中也明确指出, X25519 的协议的实现必须接受非规范 (Non-Canonical) 的值, 对于 X25519 而言, 非规范的值包括 $2^{255}-19$ 到 $2^{255}-1$ 的所有值. 注意到函数 `decodeLittleEndian` 没有对输入参数 `u` 做任何限制, 也即 32 字节的任意值都可以作为 Curve25519 的公钥 $\{\underline{u}, u \in \{0, 1, \ldots, 2^{256}-1\}\}$.

Listing 2: X25519 和 X448 的编解码

```
1  def decodeLittleEndian(b, bits):
2      return sum([ b[i] << 8*i for i in range((bits+7)//8) ])
3
4  def decodeUCoordinate(u, bits):
5      u_list = [b for b in u]
6      # Ignore any unused bits.
7      if bits % 8:
8          u_list[-1] &= (1 << (bits % 8)) - 1
9      return decodeLittleEndian(u_list, bits)
10
11 def encodeUCoordinate(u, bits):
12     return bytearray([ (u >> 8*i) & 0xff for i in range((bits+7)//8) ])
13
14 def decodeScalar25519(k):
15     k_list = [b for b in k]
16     k_list[0] &= 248  # 1111 1000
17     k_list[31] &= 127 # 0111 1111
18     k_list[31] |= 64  # 0100 0000
19     return decodeLittleEndian(k_list, 255)
20
```

```
21 def decodeScalar448(k):
22     k_list = [b for b in k]
23     k_list[0] &= 252
24     k_list[55] |= 128
25     return decodeLittleEndian(k_list, 448)
```

考虑 $\mathbf{x}(kP)$ 中的标量 k 的解析, 参见 Listing 2 中的函数 `decodeScalar25519`. 与解码坐标值时类似, 解码 k 时将 32 个字节的数组看成是小端法表示的 k, 但是按照小端法将字节数组转换成标量 k 之前, 需要将最低 3 比特清零 (第 16 行), 将最高位清零 (第 17 行), 并将紧邻最高位的比特设置为 1 (第 19 行). 也即 X25519 的私钥取值空间为 $\{\underline{k} : k \in 2^{254} + 8 \cdot \{0, 1, \ldots, 2^{251} - 1\}\}$. 将最低 3 比特清零可以保证私钥值是 8 的倍数, 考虑到 Curve25519 曲线的余因子为 8, 这样可以避免 small-subgroup 一类的攻击. 将最高位清零, 是因为底层素数域的最高位为零, 也即公钥的可能取值将紧邻最高位的比特设置为 1, 有利于常量时间的蒙哥马利阶梯算法实现[6]. 值得指出的是, 与 secp256k1 或者 secp256r1 等曲线的私钥可以在某个区间内连续取值 (整数值) 不同, 曲线 Curve25519 上的私钥并不是某个区间内的连续取值, 这是为了规避余因子不为 1 可能引发的安全隐患, 但同时也部分部分影响了曲线的应用方式, 尤其是当希望在 Ed25519 签名算法上实现 BIP-32 的功能时.

至此, 介绍了 X25519 密钥协商协议的设计理念, 支撑协议的底层点群结构以及这些点群结构可以实现仅依赖 x 坐标的倍乘运算的原理, 以及公钥和私钥的取值和编解码转换. 但还未探讨在给定标量 k 和点 P 的横坐标 $\mathbf{x}(P)$ 的条件下如何完成倍乘计算 $\mathbf{x}(kP)$. 点的倍乘运算依赖点的加法运算, 然而无法直接从 $\mathbf{x}(P)$ 和 $\mathbf{x}(Q)$ 计算 $\mathbf{x}(P+Q)$, 因为如前所述 $\mathbf{x}(P)$ 可能对应点 P 也可能对应点 $-P$. $\pm P$ 与 $\pm Q$ 的点加可能结果为 $P+Q, -P+Q, P+(-Q), (-P)+(-Q)$, 可以注意到 $\mathbf{x}(P+Q) = \mathbf{x}(-P-Q)$ 并且 $\mathbf{x}(-P+Q) = \mathbf{x}(P-Q)$, 也即 $\mathbf{x}(P)$ 和 $\mathbf{x}(Q)$ 能够完全确定集合 $\{\mathbf{x}(P+Q), \mathbf{x}(P-Q)\}$ 的值. 更进一步, 集合 $\{\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P+Q), \mathbf{x}(P-Q)\}$ 中的任意 3 个元素可以确定出另外一个元素的值. 由此可以定义仅基于 x 坐标的加法运算:

$$\begin{aligned} \mathsf{xADD}: &\quad (\mathbf{x}(P), \mathbf{x}(Q), \mathbf{x}(P-Q)) \to \mathbf{x}(P+Q) \\ \mathsf{xDBL}: &\quad \mathbf{x}(P) \to \mathbf{x}(2P) \end{aligned}$$

考虑点 $P = (x_P, y_P), Q = (x_Q, y_Q), P, Q \notin \{\mathcal{O}, (0,0)\}$, 并记 $P+Q = (x_{P+Q}, y_{P+Q})$, $P-Q = (x_{P-Q}, y_{P-Q})$, Montgomery 发现当 $P \neq Q$ 时, $x_P, x_Q, x_{P+Q}, x_{P-Q}$ 之间的关

[6] StackExchange: When using Curve25519, why does the private key always have a fixed bit at 2^254? https://crypto.stackexchange.com/questions/11810/when-using-curve25519-why-does-the-private-key-always-have-a-fixed-bit-at-2254/11818#11818

系满足:

$$x_{P+Q}x_{P-Q}(x_P - x_Q)^2 = (x_P x_Q - 1)^2 \tag{3}$$

而当 $P = Q, P + P = (x_{2P}, y_{2P})$ 时, x_P 与 x_{2P} 之间的关系满足:

$$4x_{2P}x_P(x_P^2 + Ax_P + 1) = (x_P^2 - 1)^2 \tag{4}$$

在此处的目标是根据 x_P, x_Q, x_{P-Q} 计算 x_{P+Q} 以及根据 x_P 计算 x_{2P}.

转换到射影坐标系 (Projective Coordinates), 并记 $\mathbf{x}(P) = (X_P : Z_P)$, 其中 $x_P = X_P/Z_P$, 同样有 $\mathbf{x}(Q) = (X_Q : Z_Q), \mathbf{x}(P+Q) = (X_{P+Q} : Z_{P+Q}), \mathbf{x}(P-Q) = (X_{P-Q} : Z_{P-Q})$. $P, Q \notin \{\mathcal{O}, (0,0)\}$, 则当 $P \neq Q$ 时, 将射影坐标带入方程 (3) 中有:

$$\begin{aligned}
\frac{X_{P+Q}}{Z_{P+Q}}\frac{X_{P-Q}}{Z_{P-Q}}\left(\frac{X_P}{Z_P} - \frac{X_Q}{Z_Q}\right)^2 &= \left(\frac{X_P}{Z_P}\frac{X_Q}{Z_Q} - 1\right)^2 \\
\Longrightarrow \frac{X_{P+Q}}{Z_{P+Q}}\frac{X_{P-Q}}{Z_{P-Q}}\left(\frac{X_P Z_Q - X_Q Z_P}{Z_P Z_Q}\right)^2 &= \left(\frac{X_P X_Q - Z_P Z_Q}{Z_P Z_Q}\right)^2 \\
\Longrightarrow \frac{X_{P+Q}}{Z_{P+Q}}\frac{X_{P-Q}}{Z_{P-Q}}(X_P Z_Q - X_Q Z_P)^2 &= (X_P X_Q - Z_P Z_Q)^2 \\
\Longrightarrow X_{P+Q}X_{P-Q}(X_P Z_Q - X_Q Z_P)^2 &= Z_{P+Q}Z_{P-Q}(X_P X_Q - Z_P Z_Q)^2
\end{aligned}$$

为保证上式成立, 如下指定 X_{P+Q} 和 Z_{P+Q} 的值即可:

$$\begin{cases} X_{P+Q} &= Z_{P-Q}(X_P X_Q - Z_P Z_Q)^2 \\ Z_{P+Q} &= X_{P-Q}(X_P Z_Q - X_Q Z_P)^2 \end{cases},$$

也等价于

$$\begin{cases} X_{P+Q} &= Z_{P-Q}\left[(X_P - Z_P)(X_Q + Z_Q) + (X_P + Z_P)(X_Q - Z_Q)\right]^2 \\ Z_{P+Q} &= X_{P-Q}\left[(X_P - Z_P)(X_Q + Z_Q) - (X_P + Z_P)(X_Q - Z_Q)\right]^2 \end{cases}, \tag{5}$$

当 $P = Q$ 时, 将射影坐标带入方程 (4) 中有:

$$\begin{aligned}
4\frac{X_{2P}}{Z_{2P}}\frac{X_P}{Z_P}\left(\left(\frac{X_P}{Z_P}\right)^2 + A\frac{X_P}{Z_P} + 1\right) &= \left(\left(\frac{X_P}{Z_P}\right)^2 - 1\right)^2 \\
\Longrightarrow 4\frac{X_{2P}}{Z_{2P}}\frac{X_P}{Z_P}\left(\frac{X_P^2 + AX_P Z_P + Z_P^2}{Z_P^2}\right) &= \frac{\left(X_P^2 - Z_P^2\right)^2}{Z_P^4} \\
\Longrightarrow 4\frac{X_{2P}}{Z_{2P}}(X_P Z_P)\left(X_P^2 + AX_P Z_P + Z_P^2\right) &= \left(X_P^2 - Z_P^2\right)^2 \\
\Longrightarrow 4X_{2P}(X_P Z_P)\left(X_P^2 + AX_P Z_P + Z_P^2\right) &= Z_{2P}\left(X_P^2 - Z_P^2\right)^2
\end{aligned}$$

为保证上式成立, 如下指定 X_{2P} 和 Z_{2P} 的值即可:

$$\begin{cases} X_{2P} &= \left(X_P^2 - Z_P^2\right)^2 \\ Z_{2P} &= 4(X_P Z_P)\left(X_P^2 + AX_P Z_P + Z_P^2\right) \end{cases},$$

也等价于

$$\begin{cases} X_{2P} &= (X_P + Z_P)^2(X_P - Z_P)^2 \\ Z_{2P} &= (4X_P Z_P)((X_P - Z_P)^2 + ((A+2)/4)(4X_P Z_P)) \end{cases}, \tag{6}$$

根据方程 (5) 给出的 xADD 运算过程和方程 (6) 给出的 xDBL 运算过程即可完成根据标量 k 和 $\mathbf{x}(P)$ 完成倍乘计算 $\mathbf{x}(kP)$. 在继续描述 $\mathbf{x}(kP)$ 的计算过程之前, 值得指出的是, 在方程 (5) 中没有出现涉及到蒙哥马利曲线的参数 A 和 B, 而在方程 (6) 中仅涉及到了蒙哥马利曲线的参数 A, 也即基于 x 坐标的点的运算与参数 B 无关. 前面提到 X25519 协议为了使所有的 $x \in \mathbb{F}_p$ 都是合法的公钥点, 同时考虑了基域上的椭圆曲线的点群 $E(\mathbb{F}_p)$ 以及二次扩域上的椭圆曲线点群 $E'(\mathbb{F}_p)$, 而 $E'(\mathbb{F}_p)$ 也可以看做是曲线 $E^M_{A,B}$ 的二次扭曲线 $E^M_{A,B'}$ 上的椭圆曲线点群. 由于两个曲线参数中只有 B 不同, 因此前述的 xADD 运算过程和 xDBL 运算过程同时适用于两个椭圆曲线点群, 而无需区别对待. 另外注意到方程 (6) 中需要完成与 $(A+2)/4$ 的计算, 因此在选择曲线参数的时候, 通过选择 A 的值使得 $(A+2)/4$ 的值很小对于加速计算有帮助. 以 Curve25519 为例, 其中 $A = 486662$, 则 $(A+2)/4$ 的值为 121665 (十六进制表示形式 0x1db41).

接下来考虑如何利用 xADD 和 xDBL 完成点的倍乘运算 $\mathbf{x}(kP)$. 首先从 Double-and-Add 方法说起, 参见 Algorithm 1. Algorithm 1 可以看做是借助 R_1 完成的 Double-and-Add 算法的变形. 其中的 for 循环维持了两个不变量. 第一个不变量是根据第 1, 4, 6 行可以观察到的 R_0 和 R_1 之间差值的不变量 $R_1 - R_0 = P$. 观察 Algorithm 1 中对 R_0 的操作, R_0 在循环开始之前 (第 1 行) 被设为 P, 当 $k_i = 0$ 时 $R_0 \leftarrow 2R_0$ (也即 double 操作), 而当 $k_i = 1$ 时有 $R_0 \leftarrow R_0 + R_1 = 2R_0 + P$. Algorithm 1 实质上借助 R_1 完成了对 R_0 的 Double-and-Add 操作. 这也引出了 Algorithm 1 中的第二个不变量, 在第 i 次迭代中 (i 从 $\ell - 2$ 逐次递减到 0) 有 $R_0 = \lfloor k/2^i \rfloor P, R_1 = R_0 + P$, 则当 $i = 0$ 时, 就有 $R_0 = kP$, 因此 Algorithm 1 正确计算了点的倍乘 kP.

Algorithm 1: Montgomery's Binary Algorithm

Input: $k = \sum_{i=0}^{\ell-1} k_i 2^i, k_{\ell-1} = 1, P \in E_{A,B}(\mathbb{F}_p)$

Output: kP

1 $(R_0, R_1) \leftarrow (P, 2P)$;
2 **for** $i = \ell - 2$ *down to* 0 **do**
3 **if** $k_i = 0$ **then**
4 $(R_0, R_1) \leftarrow (2R_0, R_0 + R_1)$
5 **else**
6 $(R_0, R_1) \leftarrow (R_0 + R_1, 2R_1)$
7 **return** R_0

基于 Algorithm 1 可以发展出计算 $\mathbf{x}(kP)$ 的蒙哥马利阶梯算法 (Montgomery Lad-

der). 用 xADD 和 xDBL 替换 Algorithm 1 中的点加和点倍乘运算, 同时考虑射影坐标系的表示方法 $(X_P : Z_P) = \mathbf{x}(P)$, 可以得到 Algorithm 2. 需要指出的是, 如前所述根据方程 (5) 计算两个点的 xADD 时, 除了这两个点之外, 还需要这两点的差值. 根据循环不变量, 一直有 $\mathsf{x}_1 - \mathsf{x}_0 = P = (X_P, Z_P)$, 因此在第 4 行和第 6 行, xADD 的输入参数为 $(\mathsf{x}_0, \mathsf{x}_1, (X_P, Z_P))$.

Algorithm 2: Montgomery's Ladder Algorithm

Input: $k = \sum_{i=0}^{\ell-1} k_i 2^i, k_{\ell-1} = 1, (X_P : Z_P) = \mathbf{x}(P)$

Output: $(X_k, Z_k) = \mathbf{x}(kP)$, if $P \notin \{\mathcal{O}, (0,0)\}$ otherwise $Z_k = 0$

1 $(\mathsf{x}_0, \mathsf{x}_1) \leftarrow ((X_P, Z_P), \mathsf{xDBL}(X_P, Z_P))$;

2 **for** $i = \ell - 2$ *down to* 0 **do**

3 **if** $k_i = 0$ **then**

4 $(\mathsf{x}_0, \mathsf{x}_1) \leftarrow (\mathsf{xDBL}(\mathsf{x}_0), \mathsf{xADD}(\mathsf{x}_0, \mathsf{x}_1, (X_P, Z_P)))$

5 **else**

6 $(\mathsf{x}_0, \mathsf{x}_1) \leftarrow (\mathsf{xADD}(\mathsf{x}_0, \mathsf{x}_1, (X_P, Z_P)), \mathsf{xDBL}(\mathsf{x}_1))$

7 **return** x_0

在 Algorithm 1 和 Algorithm 2 中, k_i 的值会影响程序的执行流程. 在 ECDH 协议中, k 的值通常为协议参与方的私钥, 涉及密钥信息的操作通常希望是常量时间的实现, 也即密钥的取值不应该影响具体的执行流程, 否则会泄露关于密钥的信息, 从而招致侧信道攻击. 实践中惯用的手法是用加法乘法等运算实现条件交换 (Conditional Swap) 替换条件判断, 参见 Listing 3 中函数 `cswap`. Algorithm 2 用条件交换 SWAP 改进之后, 得到 Algorithm 3. 另一处在 Algorithm 2 和 Algorithm 3 中展示的尚未解释的常量时间实现的技术细节也与参数 k 相关. 在两个算法输入参数中, 都指定了 $k_{\ell-1} = 1$, 这是为了使得两个算法中的循环次数与 k 的具体取值无关. 由于 Curve25519 定义在 255 比特的素数域上, 因此最高位一直为零, 为了便于常量时间实现, 遵循了同样的常量时间技巧: 将次高位比特设置为 1, 使迭代次数为常数, 这也是 Listing 2 中执行第 18 行操作的原因.

值得注意的是, Algorithm 2 和 Algorithm 3 在 $P \in \{\mathcal{O}, (0,0)\}$ 时, 无法正确计算 $\mathbf{x}(kP)$. $P \in \{\mathcal{O}, (0,0)\}$ 时, 返回值为 $(X_k, Z_k) = (e, 0)$, 其中 e 可能是零也可能是非零值. 对于仅依赖 x 坐标的 ECDH 过程, 这意味着协议参与方需要执行检查, 以确认输入不是 $\mathbf{x}(\mathcal{O})$ 也不是 $\mathbf{x}((0,0))$. Bernstein 在构建 X25519 协议时的另一个贡献是, 通过调整映射 $\mathbf{x}$(参见公式 (1)) 为映射 $\mathbf{x_0}$(参见公式 (2)), 这种对输入参数的检查可以省略. 这意味着蒙哥马利阶梯算法的输入为 $(X_P, Z_P) = (x, 1)$, 其中 x 可以是 $\mathbb{F}_p$ 上的任意的元素, 也即任

Algorithm 3: Constant-Time Montgomery's Ladder Algorithm

Input: $k = \sum_{i=0}^{\ell-1} k_i 2^i, k_{\ell-1} = 1, (X_P : Z_P) = \mathbf{x}(P)$

Output: $(X_k, Z_k) = \mathbf{x}(kP)$, if $P \notin \{\mathcal{O}, (0,0)\}$ otherwise $Z_k = 0$

$(\mathsf{x}_0, \mathsf{x}_1) \leftarrow (\mathsf{xDBL}(X_P, Z_P), (X_P, Z_P))$;

for $i = \ell - 2$ *down to* 0 **do**

  $(\mathsf{x}_0, \mathsf{x}_1) \leftarrow \mathsf{SWAP}(k_{i+1} \oplus k_i, (\mathsf{x}_0, \mathsf{x}_1))$;

  $(\mathsf{x}_0, \mathsf{x}_1) \leftarrow (\mathsf{xDBL}(\mathsf{x}_0), \mathsf{xADD}(\mathsf{x}_0, \mathsf{x}_1, (X_P, Z_P)))$;

$(\mathsf{x}_0, \mathsf{x}_1) \leftarrow \mathsf{SWAP}(k_0, (\mathsf{x}_0, \mathsf{x}_1))$;

return x_0

意的 $x \in \mathbb{F}_p$ 都对应蒙哥马利曲线或者二次扭曲线上的一个点: $x = \mathbf{x_0}(P)$. 另外也需要同步修改返回值为 $x_k = X_k Z_k^{p-2} \in \mathbb{F}_p$. 这是因为当 $Z_k \neq 0$ 时, 有 $x_k = X_k/Z_k$, 而当 $Z_k = 0$ 时, $x_k = 0$, 两种情形都下都有 $x_k = \mathbf{x_0}(kP)$, 另外根据费马小定理有 $Z_k^{-1} = Z_k^{p-2}$, 则蒙哥马利阶梯算法的返回值也统一为 $x_k = X_k Z_k^{p-2} \in \mathbb{F}_p$.

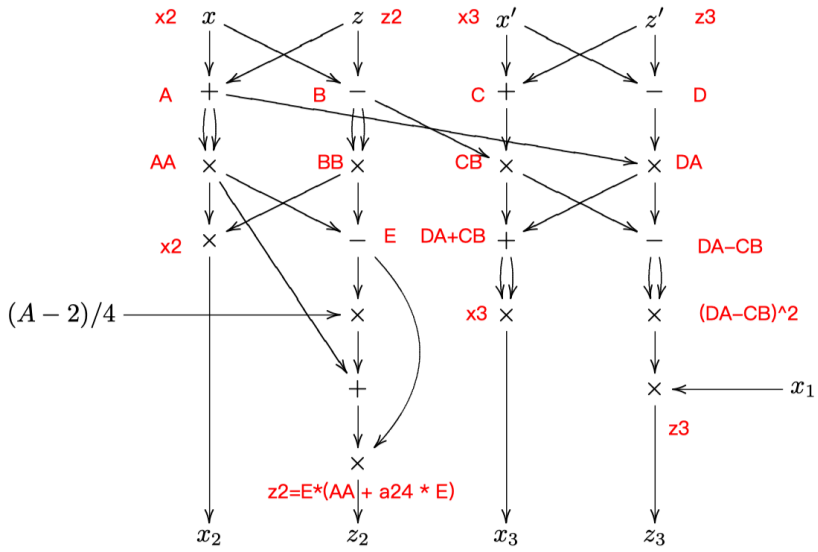

Figure 2: X25519 密钥交换依赖的点运算

根据方程 (5) 和方程 (6) 中展示的 xADD 和 xDBL 中展示的计算过程以及 Algorithm 3 展示的蒙哥马利阶梯算法, 可以实现 $\mathbf{x}(kP)$, 参见 Listing 3 中的函数 `mul`. 函数 `mul` 并不是 Algorithm 3 的直接翻译, 而是根据方程 (5) 和方程 (6) 中展示的计算过程, 将 xADD 和 xDBL 中的共同操作抽取出来之后的计算过程, 参见 Figure 2.

Listing 3: Curve25519 和 Curve448 上的乘法运算

```
# Finite field with p
def FiniteField(p):
    class Fp:
        def __init__(self, val: int):
            assert isinstance(val, int)
            self.val = val
        def __add__(self, other):
            return Fp((self.val + other.val) % Fp.p)
        def __sub__(self, other):
            return Fp((self.val - other.val) % Fp.p)
        def __mul__(self, other):
            return Fp((self.val * other.val) % Fp.p)
        def __rmul__(self, n):
            return Fp((self.val * n) % Fp.p)
        def __pow__(self, e):
            return Fp(pow(self.val, e, Fp.p))
        def __repr__(self):
            return hex(self.val)
        def __int__(self):
            return int(self.val)
    Fp.p = p
    return Fp

def cswap(swap, x_2, x_3):
    "Conditional swap in constant time."
    dummy = swap * (x_2 - x_3)
    x_2 = x_2 - dummy
    x_3 = x_3 + dummy
    return x_2, x_3

def mul(k: int, u: int, bits: int, p: int, a24: int):
    Fp = FiniteField(p)
    x_1 = Fp(u)
    x_2 = Fp(1)
    z_2 = Fp(0)
    x_3 = Fp(u)
    z_3 = Fp(1)
    swap = 0

    for t in range(bits-1, -1, -1):
        k_t = (k >> t) & 1
```

```
        swap ^= k_t
        (x_2, x_3) = cswap(swap, x_2, x_3)
        (z_2, z_3) = cswap(swap, z_2, z_3)
        swap = k_t

        A = x_2 + z_2
        AA = A**2
        B = x_2 - z_2
        BB = B**2
        E = AA - BB
        C = x_3 + z_3
        D = x_3 - z_3
        DA = D * A
        CB = C * B
        x_3 = (DA + CB)**2
        z_3 = x_1 * (DA - CB)**2
        x_2 = AA * BB
        z_2 = E * (AA + a24 * E)

    x_2, x_3 = cswap(swap, x_2, x_3)
    z_2, z_3 = cswap(swap, z_2, z_3)
    res = x_2 * (z_2**(p - 2))
    return res
```

有了 $\mathbf{x}(kP)$ 的计算方法, 则可以实现 X25519 密钥交换协议, 参见 Listing 4 中的函数 `x25519`. 函数 `x25519` 根据随机数 `k` (在 ECDH 协议中是某个参与方的私钥) 和接收到的公钥点 `u` 执行点的倍乘运算 $\mathtt{k} \cdot \mathtt{u}$. 基于 `x25519` 容易实现 X25519 密钥交换协议.

Listing 4: X25519 与 X448 的 Python 示例

```
def x25519(k: bytes, u: bytes):
    # Curve25519 for the ~128-bit security level.
    # Computes u := k * u where k is the scalar and u is the u-coordinate.
    bits = 255
    k = decodeScalar25519(k)
    u = decodeUCoordinate(u, bits)
    p = 2**255 - 19
    a24 = 121665
    res = mul(k, u, bits, p, a24)
    return encodeUCoordinate(int(res), bits)

def x448(k: bytes, u: bytes):
    # Curve448 for the ~224-bit security level.
```

```
    bits = 448
    k = decodeScalar448(k)
    u = decodeUCoordinate(u, bits)
    p = 2**448 - 2**224 - 1
    a24 = 39081
    res = mul(k, u, bits, p, a24)
    return encodeUCoordinate(int(res), bits)
```

至此已经讲述了关于 Curve25519 和 X25519 的设计理念, 算法规范以及背后的数学原理, 现在可以尝试理解 Bernstein 在 X25519 的设计文档中总结的关于 X25519 协议的各项特点.

- **Short secret keys**: X25519 协议对应的私钥是 32 字节, 这一点并没有特殊之处, 基于椭圆曲线的旨在提供 128 比特安全性的 ECDH 协议通常都是 32 字节的私钥.
- **Short public keys**: X25519 协议对应的公钥是 32 字节, 相比其他的 ECDH 协议, 这点可算作优势. 因为基于 256 比特素数域的 ECDH 的压缩公钥表示通常需要 33 字节, 而压缩和解压缩需要耗费的时间通常在 ECDH 的速度测试中没有被纳入考量, 尤其是解压缩操作. X25519 由于只需要 x 坐标, 所以无需压缩就是 32 个字节.
- **No time variability**: 这是 X25519 协议的一个显著优势. 椭圆曲线点群运算的常量时间是非常困难的, 而且对特定实现添加侧信道防护通常会显著降低代码的执行速度. 利用蒙哥马利阶梯算法以及仅依赖 x 坐标的倍乘运算, 以及在 X25519 的实现中可以避免由于私钥信息不同而导致的分支操作, 对常量时间实现非常友好.
- **Free key validation**: 这是 X25519 协议的一个显著优势. 通过引入二次扩域上的椭圆曲线点群, 使得所有的 $x \in \mathbb{F}_p$ 都是合法的公钥, 更进一步在模运算的视角下, 所有的 32 字节的数组都可以作为合法的公钥值, 这就避免了在 ECDH 协议执行过程中检查公钥点这一比较耗费资源的操作.
- **Extremely high speed**: 通过精心选择的对 64 位寄存器友好的大素数以及蒙哥马利阶梯算法的利用, 能够显著提升仅依赖 x 坐标的倍乘运算, 而公钥合法性验证的可省略也进一步提升了 X25519 的效率.
- **Short Code**: X25519 协议用很少的代码量即可实现, 提出 X25519 协议的论文中, Bernstein 提及他的实现仅有 16KB 的大小. 一个有意思的事情是, Wesley Janssen 在其硕士论文中用 18 条 twitter 实现了 Curve25519 [7]

[7] Janssen, Wesley. "Curve25519 in 18 tweets." PhD diss., Bachelor's thesis, Radboud Univer-

3 Tendermint 中的 X25519

点对点 (Peer-to-Peer) 通信协议是区块链技术的典型特征, 以 Bitcoin, Ethereum 为代表的主流公链中的点对点通信都是明文通信, 交易的全网广播过程会因为明文通信的原因被观测到, 由此可以定位交易的发起方, 导致交易隐私的泄露. 也因此改进点对点通信的安全性也是区块链技术演进中的一个技术方向. 典型的改进策略包括调整交易广播过程的 Dandelion++ 协议[8](已经在 Zcash, Grin 等区块链项目中部署) 也包括将 P2P 的明文通信升级为加密通信, 例如 Tendermint Core 项目中采用的基于 Station-to-Station 协议的密文通信协议[9], Libra 项目中采用的 Noise 协议[10].

一个区块链项目从架构层面可以分为 3 个功能层, 最底层的是 P2P 网络通信, 中间的共识协议层以及上层的与具体业务逻辑相关的应用层. 开发区块链项目是困难的, 尤其是涉及到 P2P 网络通信与共识协议实现. 为了简化区块链项目的开发, 出现了 Tendermint Core/Cosmos SDK 以及 Polkadot/Substrate 等致力于为区块链开发提供开发框架的项目. 其中 Tendermint Core 项目致力于成为构建区块链项目的基础平台, 其中实现了 Tendermint 共识协议以及完成共识协议所需要的 P2P 网络通信功能, 而 Cosmos SDK 项目则为上层应用的开发提供了一些通用的模块, 例如账户模型所需的账户模块, PoS 机制所需要的质押等模块.

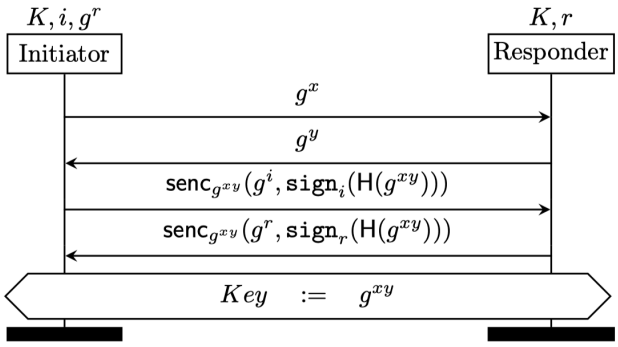

Figure 3: Tendermint Core 中的密钥协商

sity Nijmegen, 2014. https://www.cs.ru.nl/bachelors-theses/2014/Wesley_Janssen___4037332___Curve25519_in_18_tweets.pdf

[8]Fanti, Giulia, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. "Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees." Proceedings of the ACM on Measurement and Analysis of Computing Systems 2, no. 2 (2018): 29. https://arxiv.org/pdf/1805.11060.pdf

[9]Secure P2P. https://tendermint.com/docs/tendermint-core/secure-p2p.html#caveat

[10]Trevor Perrin. The Noise Protocol Framework. Revision 34. https://noiseprotocol.org/noise.pdf

本文关注 Tendermint Core 项目中的 P2P 网络通信协议中加密通道建立过程. Tendermint Core 项目中实现了基于 Station-to-Station 的可认证加密机制 (Authenticated Encryption Scheme) 的 P2P 协议. 具体的密钥协商过程参见 Figure 3, 图片截取自 Cas Cremers 和 Dennis Jackson 的论文[11]. 每个节点都会生成一个 Ed25519 密钥来作为自己身份 ID, 例如 Figure 3 中 Initiator 的私钥 i 和 Responder 的私钥 r. 当两个节点之间想要建立 TCP 连接时, 首先利用 X25519 密钥交换协议来协商密钥, 通过 HKDF_SHA256 派生出真正对流量加密的对称密钥, 然后用 AEAD 算法 chacha20poly1305 加密流量. 为了防止消息重放攻击, 每个节点都维持一个接受消息计数器和发送消息计数器, 两个计数器初始值都为 0. 相应的计数器也用来作为 chacha20poly1305 算法的 nonce 值. 密钥协商之后通信双方需要进行身份认证并可以同时确认双方通过密钥协商派生的密钥值是一致的, 也即 Figure 3 中双方交换的消息 $\mathsf{senc}_{g^{xy}}(g^i, \mathsf{sign}_i(\mathsf{H}(g^{xy})))$ 和 $\mathsf{senc}_{g^{xy}}(g^r, \mathsf{sign}_r(\mathsf{H}(g^{xy})))$, 其中 $\mathsf{senc}_{g^{xy}}$ 表示利用 g^{xy} 派生而来的密钥对通信载荷进行加密. 对 Initiator 而言, 通信载荷为代表自己身份的公钥 g^i, 以及用自己的私钥 i 对 $\mathsf{H}(g^{xy})$ 进行签名的签名值 $\mathsf{sign}_i(\mathsf{H}(g^{xy}))$. 对 Responder 而言, 通信载荷为代表自己分的公钥 g^r, 以及用自己的私钥 r 对 $\mathsf{H}(g^{xy})$ 进行签名的签名值 $\mathsf{sign}_r(\mathsf{H}(g^{xy}))$. 通过验证签名值, 可以验证对方节点并确保两者通过密钥协商得到了相同的流量加密密钥.

虽然 X25519 和 Curve25519 在设计时已经尽可能多的考虑应用时的安全陷阱, 然而 Cremers 和 Jackson 指出了 Tendermint Core 中密钥协商过程中存在的安全隐患. 值得注意的是另外一组研究人员也独立发现了 Tendermint Core 项目中握手协议存在的安全问题[12]. 这个攻击与 Curve25519 的余因子不为 1 并且在接受私钥时会将私钥的最低 3 位清零有关, 参见 Listing 2 中第 16 行的函数 `decodeScalar25519` 中的操作. 前面有提及过, 将最低 3 位清零是为了保证点的倍乘运算是在大素数子群上. 也正因为这种提前预防的措施, 会导致如果 Tendermint Core 的密钥协商中有一方发送的公钥隶属于 8 阶或者 4 阶小子群时, 则 X25519 密钥协商得到的值 g^{xy} 会是全零的值. 参考前述的 Tendermint Core 项目中安全信道的建立方式, 不失一般性, 假设 Initiator 发送的临时公钥隶属于小子群, 则在交互的最后 Initiator 会获得 Responder 的签名值 $\mathsf{senc}_{g^{xy}}(g^r, \mathsf{sign}_r(\mathsf{H}(g^{xy})))$. 这允许 Initiator 在后续与别的节点建立安全信道时利用 $g^r, \mathsf{sign}_r(\mathsf{H}(g^{xy}))$ 冒充 Responder 的身份 (只要 Initiator 向对方发送低阶临时公钥, 就可以保证 g^{xy} 的值仍为零). 这也允许中间人攻击 (Man-In-The-Middle), 在安全信道建立过程中, 中间人注入低阶临时公钥, 并通

[11] Cremers, Cas, and Dennis Jackson. "Prime, Order Please! Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman." IEEE CSF 19 (2019). https://eprint.iacr.org/2019/526.pdf

[12] p2p/conn/secret_connection.go allows MITM #3010 https://github.com/tendermint/tendermint/issues/3010

过前述方式进行身份冒充即可. 这两种情形表明如果不对接收到的临时公钥进行检查, 则 Tendermint Core 项目期望通过这种方式建立的保密带认证的安全信道信道的目标无法达成.

How do I validate Curve25519 public keys?

Don't. The Curve25519 function was carefully designed to allow all 32-byte strings as Diffie-Hellman public keys. Relevant lower-level facts: the number of points of this elliptic curve over the base field is 8 times the prime 2^252 + 27742317777372353535851937790883648493; the number of points of the twist is 4 times the prime 2^253 - 55484635554744707071703875581767296995. This is discussed in more detail in the curve25519 paper.

There are some unusual non-Diffie-Hellman elliptic-curve protocols that need to ensure ``contributory'' behavior. In those protocols, you should reject the 32-byte strings that, in little-endian form, represent 0, 1, 325606250916557431795983626356110631294008115727848805560023387167927233504 (which has order 8), 39382357235489614581723060781553021112529911719440698176882885853963445705823 (which also has order 8), 2^255 - 19 - 1, 2^255 - 19, 2^255 - 19 + 1, 2^255 - 19 + 325606250916557431795983626356110631294008115727848805560023387167927233504, 2^255 - 19 + 39382357235489614581723060781553021112529911719440698176882885853963445705823, 2(2^255 - 19) - 1, 2(2^255 - 19), and 2(2^255 - 19) + 1. But these exclusions are unnecessary for Diffie-Hellman.

Figure 4: Bernstein 关于是否需要验证 X25519 公钥的论述

最简单的应对措施是对收到的临时公钥点进行检查并拒绝低阶临时公钥. 然而 Bernstein 在网页 “A state-of-the-art Diffie-Hellman function”[13]明确指出, 在 Diffie-Hellman 协议中应用 Curve25519 时, 不需要检查临时公钥点, 参见 Figure 4. 首先分析下, Figure 4 中列出的具体的值. 记 $x_1 = 32560\cdots233504$, $x_2 = 39382\cdots705823$. 则 Figure 4 中 Bernstein 列出的需要拒绝的值中 0, 1, x_1 和 x_2 对应 Listing 1 中列出的 8 阶子群中的点的横坐标. 考虑到底层素数域的特征为 $2^{255}-19$, 而公钥的是用 32 字节表示, 则同时需要拒绝的值有 (与 0, 1, x_1, x_2 模 $2^{255}-19$ 同余并且不超过 32 字节整数的表示范围): $2^{255}-19$, $2^{255}-19+1$, $2^{255}-19+x_1$, $2^{255}-19+x_2$, $2(2^{255}-19)$, $2(2^{255}-19)+1$. Figure 4 中另外两个模 $2^{255}-19$ 同余的值是 $2^{255}-19-1$, $2(2^{255}-19)-1$. $2^{255}-19-1$ 是 $\mathbb{F}_p$ 上的二次非剩余, 也即这是二次扩域上的椭圆曲线中的点: $E'(\mathbb{F}_p) = \{\mathcal{O}\} \cup \{E(\mathbb{F}_{p^2}) \cap \mathbb{F}_p \times \sqrt{2}\mathbb{F}_p\}$. 根据前述的信息 $E'(\mathbb{F}_p)$ 的余因子为 4, 也即存在阶为 4 的子群, 该子群中除了 1 阶点 $\mathcal{O}$ 和 2 阶点 $(0,0)$, 还存在 2 个 4 阶点, 这 2 个 4 阶点具有相同的横坐标 $2^{255}-19-1$. 也即通过排除 Figure 4 中展示的临时公钥点, 可以排除所有的低阶公钥.

通过前面的描述可以发现, 虽然排除低阶点是一个简单直接的方法, 但是 Bernstein

[13] A state-of-the-art Diffie-Hellman function. https://cr.yp.to/ecdh.html

明确指出在 Diffie-Hellman 协议中无需做临时公钥的合法验证. 而遵循了这一原则的 Tendermint Core 项目中安全信道的建立却因为没有验证临时公钥而出现了安全隐患. 所以到底是哪里出了问题? 值得注意的是 TLS 1.3 和 Noise 协议中也同样支持 X25519 密钥交换协议, 同样不对临时公钥进行检查却没有发生类似 Tendermint Core 项目中的问题. Noise 协议的规范中特意说明了这一点[14]. 虽然非法的公钥值会产生全零的输出, 但是仍不推荐对公钥进行合法性验证, 因为这会增加协议复杂度与实现的变种, 并且并没有增强安全性. 允许做公钥合法性验证的原因只是因为这样能够跟部分软件行为相匹配. 可以看到, 不做公钥合法性验证并不一定会导致前述的 Tendermint Core 项目中存在的安全问题. Tony Arcieri 指出[15], Tendermint Core 项目中存在上述安全的问题的根源并不在于没有检查公钥的合法性, 问题的根源在于建立安全信道过程中握手过程的可锻造性 (Handshake Malleability). 而 TLS 1.3 和 Noise 协议中, 用脚本哈希 (Script Hashing) 解决了这一问题: 参与通信的双方对握手过程中发送和接收到的全部信息计算摘要, 如果双方得到的摘要值不同, 则当前握手协议必须终中断 (原理类似于用 MAC 来保证完整性). Tony Arcieri 同时建议使用类似于 Noise 协议做法来改良 Tendermint Core 项目中握手协议的安全性. 值得提及的是, 是否要排除 Bernstein 列出的所有的低阶点一直是个有争议的话题. 虽然在 Diffie-Hellman 协议中, 不检查公钥的合法性的做法对于正确设计的协议而言不会引入安全隐患, 但是 Daniel Genkin, Luke Valenta 和 Yuval Yarom 指出[16], 拒绝这些低阶的公钥对于增强相应实现的抗侧信道攻击能力有益处. 另外的可参考文章是 "Why not validate Curve25519 public keys could be harmful"[17]. 虽然 X25519 协议的设计与 Curve25519 曲线参数的选取已经充分考虑了常量时间实现的因素, 但是针对 Curve25519 或者 X25519 实现的侧信道攻击仍是可能的, 参见[18].

[14] *Invalid public key values will produce an output of all zeros. Alternatively, implementations are allowed to detect inputs that produce an all-zeros output and signal an error instead. This behavior is discouraged because it adds complexity and implementation variance, and does not improve security. This behavior is allowed because it might match the behavior of some software.*

[15] p2p/conn/secret_connection.go allows MITM #3010 https://github.com/tendermint/tendermint/issues/3010

[16] Genkin, Daniel, Luke Valenta, and Yuval Yarom. "May the fourth be with you: A microarchitectural side channel attack on several real-world applications of curve25519." In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 845-858. ACM, 2017. https://eprint.iacr.org/2017/806.pdf

[17] Why not validate Curve25519 public keys could be harmful https://vnhacker.blogspot.com/2015/09/why-not-validating-curve25519-public.html

[18] Kaufmann, Thierry, Hervé Pelletier, Serge Vaudenay, and Karine Villegas. "When constant-time source yields variable-time binary: Exploiting curve25519-donna built with MSVC 2015." In International Conference on Cryptology and Network Security, pp. 573-582. Springer, Cham, 2016. https://infoscience.epfl.ch/record/223794/files/32_1.pdf