Log-size Linkable Ring Signature and Hidden Amounts integrated listing

Anton A. Sokolov

Zano

anton@zano.org, acmxddk@gmail.com

Abstract *This is an unified listing for the Lin2-Xor Signature and Hidden Amounts schemes. The listing is provided in pseudo-code using the same notation as for the 'Lin2-Xor Lemma and Lig-Size Linkable Ring Signature' paper. The hidden amounts scheme follows the 'Hidden amounts scheme' and 'Elementary proofs for the Hidden amounts scheme' drafts. A number of modifications and improvements compared to the signature paper and drafts are applied. For instance, the signature linkability is moved to the hidden amounts part of the scheme now. Also, the even elements of the decoy set, namely, all Q_i's, are calculated in a slightly different manner hereinafter, still carrying all the same properties as in the signature paper.*

Keywords: Ring signature, linkable ring signature, log-size shceme, hidden amounts.

1 TRESHOLD LOG-SIZE RING SIGNATURE (TRS)

An original linkable version for this signature is described in https://eprint.iacr.org/2020/688.pdf. A modification to the original version is that key images are moved to the hidden amounts part of the unified scheme and, thus, the TRS represents a non-linkable variant of the original version.

1.1 HELPERS

Listing 1: **TRS.Helpers.CalculateFirstH**

```
Input:   [X_j]_{j=0}^{N-1}       --decoy set
         Z                       --commitment
         (w, s)                  --opening
Output:  H                       --first H
         (q, a, z, h)            --context
Procedure:
    if Z ≠ wX_{2s} then Failure
    (z, h) = (2s, 2s + 1)
    a=1
    q ←random, non-zero
    H = (w/q)X_h
    Return (H, (q, a, z, h))
```

Listing 2: **TRS.Helpers.FoldOneRsumLevel**

```
Input:   (c_1, c_3)                      --challenge pair
         [Y_j]_{j=0}^{2M-1}              --set
Output:  [F_j]_{j=0}^{M-1}               --folded set
Procedure:
    [F_j]_{j=0}^{M-1} = [Y_{2j} + c_{((2j+1)%4)} Y_{2j+1}]_{j=0}^{M-1}
    Return [F_j]_{j=0}^{M-1}
```

Listing 3: **TRS.Helpers.CalculateRiAndHiplusone**

```
Input:   (c_1, c_3)                      --challenge pair
         (q, a, z, h)                    --context
         w                               --witness part of opening
         [Y_j]_{j=0}^{M-1}               --set
Output:  (r, H_{+1})                     --i'th r and (i+1)'th H
         (q, a, z, h)                    --context
Procedure:
    (c_0, c_2) = (1, 1)
    (f, g) = (c_{(z%4)}, c_{(h%4)})
    r = qg/f
    a = fa
    z = (z//2)
    h = InvertLastBit(z)
    q ←random, non-zero
    H_{+1} = (w/(qa))Y_h
    Return ((r, H_{+1}), (q, a, z, h))
```

Listing 4: **TRS.Helpers.CalculateRn**

```
Input:   c                               --last challenge
         (q, a, z, h)                    --context
Output:  r                               --last r
         a                               --accumulated multiplier
Procedure:
    (c_0, c_1) = (1, c)
    (f, g) = (c_z, c_h)
    r = qg/f
    a = fa
    Return (r, a)
```

Listing 5: **TRS.Helpers.CalculateT**

```
Input:   R                        --Rsum
         x                        --secret scalar
Output:  T                        --right part of Schnorr id equality
         q                        --randomness used for T
Procedure:
    W = R/x
    q ←random, non-zero
    T = qW
    Return (T, q)
```

Listing 6: **TRS.Helpers.RestoreChallenges**

Input: e --same seed as for TRS.Sign

$[([(r_i^p, H_i^p)]_{i=1}^{n}, T^p)]_{p=0}^{L-1}$ --signature without the last t replies

Output: $([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c)$ --challenges

Procedure:

$c_{03} = e$

$[r_0^p]_{p=0}^{L-1} = [1]_{p=0}^{L-1}$

Forall $i = 1...(n-1)$:

$c_{i1} = \mathbf{H}_{\mathtt{scalar}}(c_{(i-1),3}, [r_{i-1}^p]_{p=0}^{L-1}, [H_i^p]_{p=0}^{L-1})$

$c_{i3} = \mathbf{H}_{\mathtt{scalar}}(c_{i1})$

$c_n = \mathbf{H}_{\mathtt{scalar}}(c_{(n-1),3}, [r_{n-1}^p]_{p=0}^{L-1}, [H_n^p]_{p=0}^{L-1})$

$c = \mathbf{H}_{\mathtt{scalar}}(c_n, [r_n^p]_{p=0}^{L-1}, [T^p]_{p=0}^{L-1})$

Return $([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c)$

Listing 7: **TRS.Helpers.BuildDecoySet**

Input: e --same seed as for TRS.Sign

$[S_j]_{j=0}^{N/2-1}$ --ring

Output: $[X_j]_{j=0}^{N-1}$ --decoy set

Procedure:

$Q' = eG$

$[Q_j]_{j=0}^{N/2-1} = \mathbf{H}_{\mathtt{point}}(Q' + S_j)$

$[X_j]_{j=0}^{N-1} = Flatten([(S_j, Q_j)]_{j=0}^{N/2-1})$

Return $[X_j]_{j=0}^{N-1}$

1.2 SIGN AND VERIFY CALLS

Listing 8: **TRS.Sign**

Input: e --scalar seed containing a hash of the --message, ring, and input commitments

$[S_j]_{j=0}^{N/2-1}$ --ring

$[Z^p]_{p=0}^{L-1}$ --L commitments

$[(w^p, s^p)]_{p=0}^{L-1}$ --L openings

Output: $[([(r_i^p, H_i^p)]_{i=1}^n, T^p, t^p)]_{p=0}^{L-1}$ --signature

Procedure:

$M = N$

$[Y_j]_{j=0}^{M-1}$ =**TRS.Helpers.BuildDecoySet**$(e, [S_j]_{j=0}^{N/2-1})$

Forall $p = 0...(L-1)$:

$(H_1^p, (q^p, a^p, z^p, h^p))$ =**TRS.Helpers.CalculateFirstH**$([Y_j]_{j=0}^{M-1}, Z^p, (w^p, s^p))$

$c_{03} = e$

$[r_0^p]_{p=0}^{L-1} = [1]_{p=0}^{L-1}$

Forall $i = 1...(n-1)$:

$c_{i1} = \mathbf{H}_{\mathbf{scalar}}(c_{(i-1),3}, [r_{i-1}^p]_{p=0}^{L-1}, [H_i^p]_{p=0}^{L-1})$

$c_{i3} = \mathbf{H}_{\mathbf{scalar}}(c_{i1})$

$M = (M/2)$

$[Y_j]_{j=0}^{M-1}$ =**TRS.Helpers.FoldOneRsumLevel**$((c_{i1}, c_{i3}), [Y_j]_{j=0}^{2M-1})$

Forall $p = 0...(L-1)$:

$((r_i^p, H_{i+1}^p), (q^p, a^p, z^p, h^p)) =$

TRS.Helpers.CalculateRiAndHiplusone$((c_{i1}, c_{i3}), (q^p, a^p, z^p, h^p), w^p, [Y_j]_{j=0}^{M-1})$

$c_n = \mathbf{H}_{\mathbf{scalar}}(c_{(n-1),3}, [r_{n-1}^p]_{p=0}^{L-1}, [H_n^p]_{p=0}^{L-1})$

$R = Y_0 + c_n Y_1$

Forall $p = 0...(L-1)$:

(r_n^p, a^p) =**TRS.Helpers.CalculateRn**$(c_n, (q^p, a^p, z^p, h^p))$

$x^p = a^p / w^p$

(T^p, q^p) =**TRS.Helpers.CalculateT**(R, x^p)

$c = \mathbf{H}_{\mathbf{scalar}}(c_n, [r_n^p]_{p=0}^{L-1}, [T^p]_{p=0}^{L-1})$

Forall $p = 0...(L-1)$:

$t^p = q^p - cx^p$

Return $[([(r_i^p, H_i^p)]_{i=1}^n, T^p, t^p)]_{p=0}^{L-1}$

Listing 9: **TRS.Verify**

```
Input:  e                                     --same seed as for TRS.Sign
        [S_j]_{j=0}^{N/2-1}                   --ring
        [Z^p]_{p=0}^{L-1}                     --L commitments
        [(([(r_i^p, H_i^p)]_{i=1}^n, T^p, t^p)]_{p=0}^{L-1}   --signature
Output: 1 or 0                                --1 on success, 0 on failure
Procedure:
    [X_j]_{j=0}^{N-1} =TRS.Helpers.BuildDecoySet(e, [S_j]_{j=0}^{N/2-1})
    ([(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n, c) =TRS.Helpers.RestoreChallenges(e, [([(r_i^p, H_i^p)]_{i=1}^n, T^p)]_{p=0}^{L-1})
    R =Rsum(n, N, [X_j]_{j=0}^{N-1}, [(c_{i1}, c_{i3})]_{i=1}^{n-1}, c_n)
    Forall p = 0...(L - 1):
        if Z^p == 0 then Return 0
        S = Z^p
        Forall i = 1...n:
            if (r_i^p == 0 or H_i^p == 0) then Return 0
            S = S + r_i^p H_i^p
            if S == 0 then Return 0
        W = S
        if (tW + cR) ≠ T then Return 0
    Return 1
```

2 HIDDEN AMOUNTS BASED ON THE TRS

2.1 ELEMENTARY PROOFS

Listing 10: **EP.SchnorrSig.Sign**

```
Input:  e                   --seed
        G                   --generator
        X                   --point
        x                   --scalar such that X = xG
Output: (s, c)              --signature
Procedure:
    q ←random, non-zero
    R = qG
    c = H_scalar(e, G, X, R)
    s = q − cx
Return (s, c)
```

Listing 11: **EP.SchnorrSig.Verify**

```
Input:  e                   --same seed as for EP.SchnorrSig.Sign
        G                   --generator
        X                   --point
        (s, c)              --signature
Output: 1 or 0              --success or failure
Procedure:
    R = sG + cX
    c' = H_scalar(e, G, X, R)
Return (c' == c)
```

Listing 12: **EP.GeneralizedSchnorrSig.Sign**

```
Input:  e                   --seed
        G_0                 --first generator
        G_1                 --second generator such that G_0 !~ G_1
        X                   --point
        (x_0, x_1)          --opening such that X = x_0 G_0 + x_1 G_1
Output: (s_0, s_1, c)       --signature
Procedure:
    q_0 ←random, non-zero
    q_1 ←random, non-zero
    R = q_0 G_0 + q_1 G_1
    c = H_scalar(e, G_0, G_1, X, R)
    s_0 = q_0 − c x_0
    s_1 = q_1 − c x_1
    Return (s_0, s_1, c)
```

Listing 13: **EP.GeneralizedSchnorrSig.Verify**

```
Input:  e                --same seed as for EP.GeneralizedSchnorrSig.Sign
        G_0              --first generator
        G_1              --second generator such that G_0 !~ G_1
        X                --point
        (s_0, s_1, c)    --signature
Output: 1 or 0           --success or failure
Procedure:
    R = s_0 G_0 + s_1 G_1 + cX
    c' = H_scalar(e, G_0, G_1, X, R)
    Return (c' == c)
```

Listing 14: **EP.VectorSchnorrSig.Sign**

```
Input:  e                        --seed
        [G_i]_{i=0}^{K-1}        --first point vector
        [X_i]_{i=0}^{K-1}        --second point vector
        x                        --scalar such that [X_i]_{i=0}^{K-1} = [xG_i]_{i=0}^{K-1}
Output: ([s_i]_{i=0}^{K-1}, c)   --signature
Procedure:
    [q_i]_{i=0}^{K-1} ←random, non-zero
    [R_i]_{i=0}^{K-1} = [q_i G_i]_{i=0}^{K-1}
    c = H_scalar(e, [G_i]_{i=0}^{K-1}, [X_i]_{i=0}^{K-1}, [R_i]_{i=0}^{K-1})
    [s_i]_{i=0}^{K-1} = [q_i - cx]_{i=0}^{K-1}
    Return ([s_i]_{i=0}^{K-1}, c)
```

Listing 15: **EP.VectorSchnorrSig.Verify**

```
Input:  e                        --same seed as for EP.VectorSchnorrSig.Sign
        [G_i]_{i=0}^{K-1}        --first point vector
        [X_i]_{i=0}^{K-1}        --second point vector
        ([s_i]_{i=0}^{K-1}, c)   --signature
Output: 1 or 0                   --success or failure
Procedure:
    [R_i]_{i=0}^{K-1} = [s_i G_i + cX_i]_{i=0}^{K-1}
    c' = H_scalar(e, [G_i]_{i=0}^{K-1}, [X_i]_{i=0}^{K-1}, [R_i]_{i=0}^{K-1})
    Return (c' == c)
```

Listing 16: **EP.BatchSchnorrSig.Sign**

```
Input:  e                       --seed
        G                       --generator
        [X_i]_{i=0}^{K-1}       --point vector
        [x_i]_{i=0}^{K-1}       --openings
Output: (s, c)                  --signature
Procedure:
    q ←random, non-zero
    R = qG
    c = H_scalar(e, G, [X_i]_{i=0}^{K-1}, R)
    c_0 = c
    [c_i]_{i=1}^{K-1} = [H_scalar(c_{i-1})]_{i=1}^{K-1}
    s = q − ∑_{i=0}^{K-1} c_i x_i
    Return (s, c)
```

Listing 17: **EP.BatchSchnorrSig.Verify**

```
Input:  e                       --same seed as for EP.BatchSchnorrSig.Sign
        G                       --generator
        [X_i]_{i=0}^{K-1}       --point vector
        (s, c)                  --signature
Output: 1 or 0                  --success or failure
Procedure:
    c_0 = c
    [c_i]_{i=1}^{K-1} = [H_scalar(c_{i-1})]_{i=1}^{K-1}
    R = sG + ∑_{i=0}^{K-1} c_i X_i
    c' = H_scalar(e, G, [X_i]_{i=0}^{K-1}, R)
    Return (c' == c)
```

Listing 18: **EP.GeneralizedBatchSchnorrSig.Sign**

```
Input:  e                           --seed
        G_0                         --first generator
        G_1                         --second generator such that G_0 !~ G_1
        [X_i]_{i=0}^{K-1}           --point vector
        [(x_{0i}, x_{1i})]_{i=0}^{K-1}  --openings
Output: (s_0, s_1, c)               --signature
Procedure:
    q_0 ←random, non-zero
    q_1 ←random, non-zero
    R = q_0 G_0 + q_1 G_1
    c = H_scalar(e, G_0, G_1, [X_i]_{i=0}^{K-1}, R)
    c_0 = c
    [c_i]_{i=1}^{K-1} = [H_scalar(c_{i-1})]_{i=1}^{K-1}
    s_0 = q_0 − ∑_{i=0}^{K-1} c_i x_{0i}
    s_1 = q_1 − ∑_{i=0}^{K-1} c_i x_{1i}
    Return (s_0, s_1, c)
```

Listing 19: **EP.GeneralizedBatchSchnorrSig.Verify**

```
Input:  e            --same seed as for EP.GeneralizedBatchSchnorrSig.Sign
        G_0          --first generator
        G_1          --second generator such that G_0 !~ G_1
        [X_i]_{i=0}^{K-1}  --point vector
        (s_0, s_1, c)      --signature
Output: 1 or 0       --success or failure
Procedure:
```

$c_0 = c$

$[c_i]_{i=1}^{K-1} = [\mathbf{H}_{\mathtt{scalar}}(c_{i-1})]_{i=1}^{K-1}$

$R = s_0 G_0 + s_1 G_1 + \sum_{i=0}^{K-1} c_i X_i$

$c' = \mathbf{H}_{\mathtt{scalar}}(e, G_0, G_1, [X_i]_{i=0}^{K-1}, R)$

Return $(c' == c)$

2.2 SIGN AND VERIFY FOR THE INTEGRATED SIGNATURE AND HIDDEN AMONTS PROOF

It's assumed four linearly independent generators G, H_0, H_1, H_2 are predefined and known to the provers and verifiers. The generators H_0, H_1, H_2 are defined in such a way so that they remains linearly independent together with $\mathbf{H_{point}}(P)$ for any point P such that a linear relation of P to G could be known.

The generators H_0, H_1, H_2 are denoted using the subscripts $0, 1, 2$ only, and this distinguishes them from the points H_i^j returned as a part of the TRS signature, which always have both superscript and subscript.

Listing 20: **HA.Sign**

Input: m --message

$[(P_i, A_i)]_{i=0}^{N/2-1}$ --ring of (CN_address, Hidden_amount) pairs

$[(x^p, s^p, (f^p, v^p))]_{p=0}^{L-1}$ --L private keys and hidden amount openings

$[(R_j, E_j)]_{j=0}^{M-1}$ --output (CN_address, Hidden_amount) pairs

$[(f^j, v^j)]_{j=L}^{L+M-1}$ --M output h/a openings indexed from L

Output: $([(I^p, (T_p, B_p, U_p, Y_p), (s^1_{0p}, s^1_{1p}, c^1_p), K_p, W_p, (s^3_{0p}, s^3_{1p}, c^3_p), ([(r_i^p, H_i^p))]_{i=1}^n, T^p, t^p)]_{p=0}^{L-1},$
$(s^2, c^2), (s^4_0, s^4_1, c^4), (s^5, c^5))$ --signature

Procedure:

- Check all N/2 P_i's are different
- Check all L s^p's are different

$[(P^p, A^p)]_{p=0}^{L-1} = [(P_{s^p}, A_{s^p})]_{p=0}^{L-1}$

$[I^p]_{p=0}^{L-1} = [\mathtt{H_{point}}(P^p)/x^p]_{p=0}^{L-1}$

$[\xi_p]_{p=0}^{L-1} \leftarrow$random, non-zero

$[(T_p, B_p, U_p, Y_p)]_{p=0}^{L-1} = [(\xi_p H_0, \xi_p A^p, \xi_p P^p, \xi_p \mathtt{H_{point}}(P^p))]_{p=0}^{L-1}$

$z_0 = \mathtt{H_{scalar}}(G, H_0, H_1, H_2, m, [(P_i, A_i)]_{i=0}^{N/2-1}, [I^p]_{p=0}^{L-1}, [(T_p, B_p, U_p, Y_p)]_{p=0}^{L-1})$

$z_1 = \mathtt{H_{scalar}}(z_0)$

$e = \mathtt{H_{scalar}}(z_1)$

$[X_i]_{i=0}^{N/2-1} = [H_0 + A_i + z_0 P_i + z_1 \mathtt{H_{point}}(P_i)]_{i=0}^{N/2-1}$

$[Z^p]_{p=0}^{L-1} = [T_p + B_p + z_0 U_p + z_1 \mathtt{H_{point}}(Y_p)]_{p=0}^{L-1}$

$[([(r_i^p, H_i^p)]_{i=1}^n, T^p, t^p)]_{p=0}^{L-1} = \mathtt{TRS.Sign}(e, [X_i]_{i=0}^{N/2-1}, [Z^p]_{p=0}^{L-1}, [(\xi_p, s^p)]_{p=0}^{L-1})$

$[(s^1_{0p}, s^1_{1p}, c^1_p)]_{p=0}^{L-1} = [\mathtt{EP.VectorSchnorrSig.Sign}(e, (G, I^p), (U_p, Y_p), \xi_p x^p)]_{p=0}^{L-1}$

$[k_p]_{p=0}^{L-1} \leftarrow$random, non-zero

$[K_p]_{p=0}^{L-1} = [k_p H_1]_{p=0}^{L-1}$

$(s^2, c^2) = \mathtt{EP.BatchSchnorrSig.Sign}(e, H_1, [K_p]_{p=0}^{L-1}, [k_p]_{p=0}^{L-1})$

$[W_p]_{p=0}^{L-1} = [(B_p + K_p)/\xi_p]_{p=0}^{L-1}$

$[(s^3_{0p}, s^3_{1p}, c^3_p)]_{p=0}^{L-1} = [\mathtt{EP.VectorSchnorrSig.Sign}(e, (H_0, W_p), (T_p, B_p + K_p), \xi_p)]_{p=0}^{L-1}$

$[A^p]_{p=L}^{L+M-1} = [E_j]_{j=0}^{M-1}$

$(s^4_0, s^4_1, c^4) = \mathtt{EP.GeneralizedBatchSchnorrSig.Sign}(e, H_0, H_1, [A^p]_{p=0}^{L+M-1}, [(f^p, v^p)]_{p=0}^{L+M-1})$

$D = \sum_{j=0}^{L-1} W_j - \sum_{j=0}^{M-1} E_j$

$d = \sum_{j=0}^{L-1}(f^j + k_j/\xi_j) - \sum_{j=L}^{L+M-1} f^j$

$(s^5, c^5) = \mathtt{EP.SchnorrSig.Sign}(e, H_1, D, d)$

Return $([(I^p, (T_p, B_p, U_p, Y_p), (s^1_{0p}, s^1_{1p}, c^1_p), K_p, W_p, (s^3_{0p}, s^3_{1p}, c^3_p), ([(r_i^p, H_i^p))]_{i=1}^n, T^p, t^p)]_{p=0}^{L-1},$
$(s^2, c^2), (s^4_0, s^4_1, c^4), (s^5, c^5))$

Listing 21: **HA.Verify**

```
Input:  m                              --message
        [(P_i, A_i)]_{i=0}^{N/2-1}     --ring of (CN_address, Hidden_amount) pairs
        [(R_j, E_j)]_{j=0}^{M-1}       --output (CN_address, Hidden_amount) pairs
        ([(I^p, (T_p, B_p, U_p, Y_p), (s^1_{0p}, s^1_{1p}, c^1_p), K_p, W_p, (s^3_{0p}, s^3_{1p}, c^3_p), ([(r^p_i, H^p_i))]_{i=1}^n, T^p, t^p)]_{p=0}^{L-1},
         (s^2, c^2), (s^4_0, s^4_1, c^4), (s^5, c^5))   --signature
Output: 1 or 0                         --success or failure
Procedure:
```

- Check all N/2 P_i's are different
- Check all L I^p's are different
- For all Verify(...) calls below: if a Verify(...) call returns 0 then Return 0

$z_0 = \mathtt{H_{scalar}}(G, H_0, H_1, H_2, m, [(P_i, A_i)]_{i=0}^{N/2-1}, [I^p]_{p=0}^{L-1}, [(T_p, B_p, U_p, Y_p)]_{p=0}^{L-1})$

$z_1 = \mathtt{H_{scalar}}(z_0)$

$e = \mathtt{H_{scalar}}(z_1)$

$[X_i]_{i=0}^{N/2-1} = [H_0 + A_i + z_0 P_i + z_1 \mathtt{H_{point}}(P_i)]_{i=0}^{N/2-1}$

$[Z^p]_{p=0}^{L-1} = [T_p + B_p + z_0 U_p + z_1 \mathtt{H_{point}}(Y_p)]_{p=0}^{L-1}$

$\mathtt{TRS.Verify}(e, [X_i]_{i=0}^{N/2-1}, [Z^p]_{p=0}^{L-1}, [([(r^p_i, H^p_i)]_{i=1}^{n}, T^p, t^p)]_{p=0}^{L-1})$

$[\mathtt{EP.VectorSchnorrSig.Veryfy}(e, (G, I^p), (U_p, Y_p), (s^1_{0p}, s^1_{1p}, c^1_p))]_{p=0}^{L-1}$

$\mathtt{EP.BatchSchnorrSig.Verify}(e, H_1, [K_p]_{p=0}^{L-1}, (s^2, c^2))$

$[\mathtt{EP.VectorSchnorrSig.Verify}(e, (H_0, W_p), (T_p, B_p + K_p), (s^3_{0p}, s^3_{1p}, c^3_p))]_{p=0}^{L-1}$

$[A^p]_{p=L}^{L+M-1} = [E_j]_{j=0}^{M-1}$

$\mathtt{EP.GeneralizedBatchSchnorrSig.Verify}(e, H_0, H_1, [A^p]_{p=0}^{L+M-1}, (s^4_0, s^4_1, c^4))$

$D = \sum_{j=0}^{L-1} W_j - \sum_{j=0}^{M-1} E_j$

$\mathtt{EP.SchnorrSig.Verify}(e, H_1, D, (s^5, c^5))$

Return 1

3 ADDITIONAL CHECKS AND TRANSMISSION

3.1 EXCLUDING LOW ORDER POINTS

The low-order ed25519 curve points are excluded using the following check (this solution idea is described in `https://suyash67.github.io/homepage/assets/pdfs/bulletproofs_plus_audit_report_v1.1.pdf`, `https://loup-vaillant.fr/tutorials/cofactor`):

Listing 22: **ELO.PackPoint**

```
Input:  X                        --point
Output: P                        --packed point
Procedure:
    • Precalculate a scalar 1/8 = ((1/8) mod p),
      where p is the order of the prime-order subgroup, once
      for all the procedure calls
    P = 1/8 X
    Return P
```

Listing 23: **ELO.UnpackPoint**

```
Input:  P                    --packed point
Output: X                    --original point if it was of the prime-order
Procedure:
    X = 8P
    Return X
```

Having any ed25519 curve point X packed and subsequently unpacked with the above procedures as

$$R = \text{ELO.UnpackPoint}(\text{ELO.PackPoint}(X))$$

we obtain two following guarantees:

- the resulting point R is always of prime-order
- $(R == X)$ iff X is of prime-order

3.2 HIDDEN AMOUNT RANGE PROOF

It's required to ensure all the hidden amounts participating in the scheme are non-negative, namely, belong to a particular non-negative range. For the sake of this, it's assumed a couple of external range proof procedures are provided.

Listing 24: **RP.ProveRange**

```
Input:  A                  --point
        (f, v)             --opening
Output: range_proof        --range proof
Procedure:
    • Check A = fH_1 + vH_2
    • Obtain range_proof for v is within the range
Return range_proof
```

Listing 25: **RP.VerifyRange**

```
Input:  A                  --point
        range_proof        --range proof
Output: 1 or 0             --success or failure
Procedure:
    • Check if range_proof is a range proof for A. If this is the case,
      then result = 1, otherwise result = 0
Return result
```

3.3 SIGNING, TRANSMISSION, VERIFICATION, RECEIVING

At any time, all points in blockchain are stored in the packed state. Namely, it's assumed that the ELO.PackPoint was previously applied to them. (The old addresses, if they are used in the new signature rings, are to be explicitly packed, i.e. divided by 8.)

Each time a ring is constructed by either participant, all the ring addresses and hidden amounts are unpacked with ELO.UnpackPoint. Thus, honest participants deal with the prime-order points only. The key images are constructed from the unpacked points too.

If a low-order point was added to the blockchain by a dishonest paricipant, then the honest verifiers, after unpacking it, deal with it as though it was a prime-order point.

Overall the signing, transmission, verification and receiving process looks as follows:

- Signer steps
  - Signer collects a ring from the blockchain. It unpacks all the ring points using ELO.UnpackPoint
  - Signer obtains a signature using the HA.Sign
  - For all the output hidden amounts $[E_j]_{j=0}^{M-1}$ participated in the HA.Sign it obtains range proofs using RP.ProveRange
  - It packs all the outputs, obtained signature, and range proofs points using the ELO.PackPoint
  - It encrypts all the openings of $[E_j]_{j=0}^{M-1}$ with corresponding output public keys $[R_j]_{j=0}^{M-1}$
- Transmission
  - Signer publishes in the blockchain a hint to restore the ring
  - It publishes the outputs $[(R_j, E_j)]_{j=0}^{M-1}$, the signature, and the corresponding range proofs, with all the points packed
  - Also, it publishes the encrypted openings
- Verifier steps
  - Verifier restores and unpacks the ring
  - Verifier unpacks all the outputs $[(R_j, E_j)]_{j=0}^{M-1}$, the signature, and the range proofs points
  - It checks the signature using HA.Verify
  - It checks the range proofs using RP.VerifyRange
- Receiver steps
  - Receiver performs all the Verifier's steps
  - It checks if it can know a private key for any of the output addresses $[R_j]_{j=0}^{M-1}$. If so, it decrypts a corresponding hidden amount