Platin Crypto Collection

Private Location Verification with Incentivization

Platin Technical Report

Vadym Fedyukovych

September 18, 2018

Abstract

We design an interactive proof system for location attributes.

1 Definitions

Node proves the statement "distance is within a threshold" (less or equal) for node coordinates (x_n, y_n, z_n), given location (x_l, y_l, z_l), and some threshold d (all integers):

$$d^2 - ((x_n - x_l)^2 + (y_n - y_l)^2 + (z_n - z_l)^2) = a_1^2 + a_2^2 + a_3^2 + a_4^2 \quad (1)$$

We rely on 4-squares Lagrange theorem to prove equality statement (Lipmaa). Proofs for integer relations are possible in hidden group order setup.

2 Proof setup

Let g be a generator of a proper group of a hidden order, and h be a group element (Pedersen commitment scheme). We use multiplicative group of invertible residue classes modulo a composite n such that $n = pq$, $p = 2p' + 1$, $q = 2q' + 1$ and p, q, p', q' primes (Idemix).

3 Signals harvesting

Node picks random (r_x, r_y, r_z), creates commitment (s_x, s_y, s_z) to it's coordinates

$$s_x = g^{x_n} h^{r_x}, \qquad s_y = g^{y_n} h^{r_y}, \qquad s_z = g^{z_n} h^{r_z} \quad (2)$$

and keeps coordinates-randoms pairs (x_n, y_n, z_n), (r_x, r_y, r_z) private.

This section is of historical value only; 'representation-based' commitment was implemented as a proof-of-concept.

3.1 Representation-based commitment

Following U-Prove, consider commitment scheme resulting in a single group element:

$$s_U = g_x^{x_n} g_y^{y_n} g_z^{z_n} g^r \tag{3}$$

where g_x, g_y, g_z are group elements, and r is a random. This scheme admits a proof of knowledge with the same responses required form threshold location verification. This scheme can be extended with additional components.

3.2 Two-level commitment

To achieve expected properties of Merkle-tree based scheme while keeping an option to run location proof protocols, representation-based commitments could be leaves of Merkle tree.

4 Proof

Sigma-protocol with 3 messages. Public information is node location commitment (3), given location and threshold (center and radius), proof parameters. Private information is node location and randomness to commitment. Setup is $g, h_1 \dots h_4, g_x, g_y, g_z, g_r$ (9 group elements).

1. Prover (node) calculates (1) $a_1 \dots a_4$ from locations and threshold with Rabin-Shallit algorithm [1], picks random $\alpha_j, \eta, \gamma, \beta_x, \beta_y, \beta_z, \beta_r, \rho_0, \rho_1$, computes and sends initial commitments b_0, b_1, t_a, t_n
   (f_0 and f_1 are explained at Background section)

$$b_0 = g^{f_0} g_r^{\rho_0}, \qquad b_1 = g^{f_1} g_r^{\rho_1} \tag{4}$$

$$s_a = g^{\gamma} \prod_{j=1}^{4} h_j^{a_j}, \qquad t_a = g^{\eta} \prod_{j=1}^{4} h_j^{\alpha_j}, \qquad t_n = g_x^{\beta_x} g_y^{\beta_y} g_z^{\beta_z} g^{\beta_r} \tag{5}$$

2. Verifier chooses and sends his challenge c

[1] Before R-S algorithm is implemented, we actually pick some a_j and calculate threshold (radius) instead.

3. Prover computes and sends responses

$$X_n = cx_n + \beta_x,\ Y_n = cy_n + \beta_y,\ Z_n = cz_n + \beta_z,\ R = cr + \beta_r \tag{6}$$

$$A_j = ca_j + \alpha_j,\ R_a = c\gamma + \eta,\ R_d = c\rho_1 + \rho_0 \tag{7}$$

4. Proof verification

$$g_x^{X_n} g_y^{Y_n} g_z^{Z_n} g^R s_U^{-c} = t_n, \quad g^{R_a} (\prod_{j=1}^{4} h_j^{A_j}) s_a^{-c} = t_a \tag{8}$$

$$g^{c^2d^2 - ((X_n - cx_l)^2 + (Y_n - cy_l)^2 + (Z_n - cz_l)^2) - (A_1^2 + A_2^2 + A_3^2 + A_4^2)} g_r^{R_d} = b_1^c b_0 \tag{9}$$

5 Background

Consider quadratic (degree 2 in v) polynomial

$$\begin{aligned} f_V(v) = f_2 v^2 + f_1 v + f_0 = \\ v^2 d^2 - (((vx_n + \beta_x) - vx_l)^2 + ((vy_n + \beta_y) - vy_l)^2 + ((vz_n + \beta_z) - vz_l)^2) \\ - ((va_1 + \alpha_1)^2 + (va_2 + \alpha_2)^2 + (va_3 + \alpha_3)^2 + (va_4 + \alpha_4)^2) \end{aligned} \tag{10}$$

This polynomial is actually linear ($f_2 = 0$, degree-one in v) if, and only if statement about distance (1) holds for node coordinates that are hidden from verifier. We evaluate this polynomial at a random point chosen as the challenge of verifier. It follows, distance verification equation (9) only needs constant b_0 and degree-one b_1^c components.

$$f_0 = -\beta_x^2 - \beta_y^2 - \beta_z^2 - \alpha_1^2 - \alpha_2^2 - \alpha_3^2 - \alpha_4^2 \tag{11}$$

$$\begin{aligned} f_1 = -2(x_n - x_l)\beta_x - 2(y_n - y_l)\beta_y - 2(z_n - z_l)\beta_z \\ - 2a_1\alpha_1 - 2a_2\alpha_2 - 2a_3\alpha_3 - 2a_4\alpha_4 \end{aligned} \tag{12}$$

Prover calculates b_0, b_1 from f_0, f_1.

6 Not-at-location proof

Proving a negative location statement is a valid usecase, that could be demonstrated with "not at the grocery store" scenario. Rather that proving "distance is smaller than" (1), complementary "is larger"

proof is given. In the following, we only show changes required to the main protocol.

$$((x_n - x_l)^2 + (y_n - y_l)^2 + (z_n - z_l)^2) - d^2 = a_1^2 + a_2^2 + a_3^2 + a_4^2 \quad (13)$$

$$g^{((X_n - cx_l)^2 + (Y_n - cy_l)^2 + (Z_n - cz_l)^2) - c^2 d^2 - (A_1^2 + A_2^2 + A_3^2 + A_4^2)} h^{R_a} = b_1^c b_0 \quad (14)$$

$$f_V(v) = f_2 v^2 + f_1 v + f_0 =$$
$$(((vx_n + \beta_x) - vx_l)^2 + ((vy_n + \beta_y) - vy_l)^2 + ((vz_n + \beta_z) - vz_l)^2) - v^2 d^2$$
$$- ((va_1 + \alpha_1)^2 + (va_2 + \alpha_2)^2 + (va_3 + \alpha_3)^2 + (va_4 + \alpha_4)^2) \quad (15)$$

7 Logical-OR threshold location

Consider a franchise operating multiple stores, and a usecase of proving location is "at Starbucks" without telling which one of K known. We define each such store with it's center (x_k, y_k, z_k) and radius (size) d_k, $k \in [1..K]$. We elaborate basic threshold proof such that prover can produce 4-squares representation for center-size of some store $k = p$, and pick arbitrary 4-tuples for all other stores $k \neq p$.

$$\prod_{k=1} ((d_k^2 - ((x_n - x_k)^2 + (y_n - y_k)^2 + (z_n - z_k)^2)$$
$$- (a_{1,k}^2 + a_{2,k}^2 + a_{3,k}^2 + a_{4,k}^2)) = 0 \quad (16)$$

Verifier is testing that polynomial $f_{KV}(v)$ is of degree at most $2K - 1$, not $2K$.

$$f_{KV}(v) = \sum_{j=0}^{2K} f_j v^j =$$
$$\prod_{k=1}^{K} (v^2 d_k^2 - (((vx_n + \beta_x) - vx_k)^2 + ((vy_n + \beta_y) - vy_k)^2 + ((vz_n + \beta_z) - vz_k)^2)$$
$$- ((va_{1,k} + \alpha_1)^2 + (va_{2,k} + \alpha_2)^2 + (va_{3,k} + \alpha_3)^2 + (va_{4,k} + \alpha_4)^2)) \quad (17)$$

8 Private incentivization

To engage users, verifier (map service) is giving tokens in exchange for verifying location proofs. To keep privacy of users, proof verification and token issuance are separated with intermediate tokens. Intermediate tokens should unlink locations that users were confirming.

Intermediate token is a Schnorr proof instance, non-interactive variant (Fiat-Shamir), in another group of a known prime order q generated by g_i. Issuing (private) key x_i and public key X_i of map service.

A new blinded intermediate token is sent to user on each successful location verification. Users unblind their intermediate tokens and periodically exchange them for Platin tokens.

1. Issuer (map service) chooses random γ and sends $\tilde{w}$ to Recipient
$$\tilde{w} = g_i^{\gamma} \tag{18}$$
2. Recipient (node) chooses random blinding (δ, μ), produces blinded challenge $\tilde{c}_i$ and sends it to Issuer
$$w = \tilde{w} g_i^{-\delta} X_i^{\mu} \tag{19}$$
$$\tilde{c}_i = H(w) + \mu \pmod{q} \tag{20}$$
3. Issuer produces blinded response $\tilde{r}_i$ and sends it to Recipient
$$\tilde{r}_i = \tilde{c}_i x_i + \gamma \pmod{q} \tag{21}$$
4. Recipient verifies validity of response received and unblinds challenge and response
$$\tilde{c}_i = H(g_i^{\tilde{r}_i} X_i^{-\tilde{c}_i}) \tag{22}$$
$$c_i = \tilde{c}_i - \mu \pmod{q} \tag{23}$$
$$r_i = \tilde{r}_i - \delta \pmod{q} \tag{24}$$
Intermediate token is (c_i, r_i).
5. Intermediate token verification (while exchange for Platin tokens)
$$c_i = H(g_i^{r_i} X_i^{-c_i}) \tag{25}$$

9 Intermediate token properties

Both components of intermediate token are statistically (unconditionally) independent from blinded intermediate token.

Consider an Adversary trying to distinguish two pairs of unblinded intermediate tokens (c_1, r_1) and (c_2, r_2), matching them to issuing session identified with $\tilde{w}$. View of Issuer is $(\tilde{c}_i, \tilde{r}_i, c_j, r_j)$ for $j = 1, 2$.

$$g_i^{r_j} X_i^{-c_i} \stackrel{?}{=} \tilde{w} g_i^{-\delta_j} X_i^{\mu_j} = \tilde{w} g_i^{r_j - \tilde{r}_i} X_i^{\tilde{c}_i - c_j} = (\tilde{w} g_i^{-\tilde{r}_i} X_i^{\tilde{c}_i})(g_i^{r_j} X_i^{-c_j}) \tag{26}$$

It could be seen that the only possible relation does not depend on issuing session identifier $\tilde{w}$, resulting in unconditional privacy for intermediate tokens.

10 Platin Encoding Standard 01

With the scenario of user device (node) being close enough to a cell tower from sections 1 and 4, we define encoding of coordinates, commitment, witness and proof of location.

10.1 Non-interactive proof

Fiat-Shamir technique could be applied to produce a challenge c with proper hash function from concatenation of group elements t_x, t_y, t_z, b_1, b_0 from first step of interactive protocol. Non-interactive proof verification would be re-creating most of group elements and re-producing the challenge:

$$c = H(g^{X_n} h^{R_x} s_x^{-c} || g^{Y_n} h^{R_y} s_y^{-c} || g^{Z_n} h^{R_z} s_z^{-c} || b_1 || \\ g^{c^2 d^2 - ((X_n - cx_l)^2 + (Y_n - cy_l)^2 + (Z_n - cz_l)^2) - (A_1^2 + A_2^2 + A_3^2 + A_4^2)} h^{R_a} b_1^{-c}) \quad (27)$$

Non-interactive proof is

$$(c, (X_n, Y_n, Z_n, A_1, A_2, A_3, A_4), b_1) \quad (28)$$

with group element b_1 and 8 integers. With this proof, multiplicative group is used, so group elements are represented as integers.

10.2 Plaintext encoding of non-interactive proof

Popular bignumber libraries have plaintext hex representation for bignumbers, designed to inter-operate with each other. Non-interactive proof is encoded as 9-lines, with a large number on each line, in hex encoding, in the order given by (28).

10.3 Plaintext encoding of location commitment

U-Prove style commitment (3) is a single group element, represented as a single-line number in hex encoding.

10.4 Plaintext encoding of location witness

Witness is a common term for private data held by prover that allows to verify validity of a statement, open a commitment instance (equation (3)), or run a proof protocol (section 4).

Witness is represented as four lines plaintext integer numbers; 3 lines with x_n, y_n, z_n in decimal, and a big random number r in hex encoding.

Additional locations that are input to the protocol like coordinates of cell tower x_l, y_l, z_l are encoded the same as witness: 3 lines, integer numbers, decimal notation.

10.5 Plaintext encoding of group elements

Group is specified with two bignumbers, composite modulo n and generator g, as two lines plaintext, with hex encoding. Additional group elements required to produce commitment (3) are encoded on following lines the same way.

11 Platin Encoding Standard 02

Geo-coordinates are generally represented as latitude-longitude-altitude triple, that could be further converted into some local x-y-z coordinate system. This choice of coordinates apply globally, while PES 01 better fit local (like city-scale) statements. To commit to lat-lon-al triple, we amend two-level commitment approach by converting GPS data into integers and commit with the U-Prove-style scheme (3). We commit to a fixed set of locations with Merkle tree, with U-Prove commitments considered leaves of the tree.

This two-level scheme requires storage of

1. Original raw lat-lon-al data and scales (multipliers) used to convert them to integers;
2. Random integer used to produce location commitment;
3. Commitments (Merkle leaves) and intermediate hashes of the tree;
4. Merkle root as commitment to the set of location.

Conversion from lat-lon-alt can be done with decimal (Decimal degrees, DD) representation of angles (not degrees-minutes-seconds, DMS) multiplied by proper power of ten and rounded. Keeping 5 digits after the decimal point of DD representation would result in approximately one meter (or better) precision.

Further design decisions would be taken to account for project ideas other than location proofs, and for implementation-level needs. Location security policy apply for original data and commitment random input.

11.1 Local system of coordinates

It is generally acceptable to point to places on the map with latitude-longitude coordinates, sometimes accompanied with elevation. In this document, we apply Pythagorean thm to calculate distance, assuming rectangular 'meters only' system.

A C++ class 'Geocoord' was introduced at demoCircle.cpp to handle conversion to such a local system;

`get_coord_x(), get_coord_y(), get_coord_z()`

methods actually produce local coordinates. Origin of this coordinate system is at 'airdrop' location (x_l, y_l, z_l). Direction of Z axis is up, X is from North to South, Y is from East to West.

12 Proving Inside a polygon

Consider an area defined by a polygon serving as it's border, and proof systems for "inside" and "outside" statements. We design "Inside" proof as an AND of the node location $\vec{l} = (x_n, y_n)$ to the right side of the plane (line in 2D), for all lines of the polygon. We define proper side with normale vector $\vec{n} = (n_x, n_y)$ to the border, scalar product, and with proper distance d from the origin:

$$(\vec{l}\vec{n}) \geq d \tag{29}$$

As with simple "circle" case, any non-negative integer could be represented as a sum of 4 squares, and this inequality can be proved for all lines (indexed by j) of the polygon border:

$$x_n n_{x,j} + y_n n_{y,j} - d_j = a_{1,j}^2 + a_{2,j}^2 + a_{3,j}^2 + a_{4,j}^2 \tag{30}$$

This proof can be given both for known polygon, and for hidden polygon available as a commitment while verification. For known polygon case, challenge c and responses $X_n, Y_n, A_{1,j}, A_{2,j}, A_{3,j}, A_{4,j}$ we have a replacement equation:

$$g^{Y_n c n_{x,j} + Y_n c n_{y,j} - c^2 d_j - (A_{1,j}^2 + A_{2,j}^2 + A_{3,j}^2 + A_{4,j}^2)} = b_{1,j}^c b_{0,j} \tag{31}$$

with b_1, b_0 defined at the first step of the protocol.

12.1 Hidden area/polygon

Conside a case of polygon kept private, and only available as commitment. We extend the proof from previous section by replacing known normale vector and distance with responses $N_{x,j}, N_{y,j}, D_j$:

$$g^{Y_n N_{x,j} + Y_n N_{y,j} - c D_j - (A_{1,j}^2 + A_{2,j}^2 + A_{3,j}^2 + A_{4,j}^2)} = b_{1,j}^c b_{0,j} \tag{32}$$

with corresponding (alternative) b_1, b_0.

13 Calculating the four-squares

Most of protocols in this document were designed to prove 'inequality' statements (like 'more than') about integers. All such protocols are transformed into 'equality' by applying Lagrange theorem. It states that any non-negative integer can be presented as a sum of four squares. In other words, some four-tuple always exists for any non-negative integer. This is a well-known result in number theory. However, we need to actually calculate that four numbers to run an 'inequality'-type proof about integers.

A compact concise description of a candidate algorithm is given at Enrique Trevino' presentation [2]. Complexity estimates of Rabin-Shallit algorithm are given by Pollack-Trevino [3], Pollack-Schorn [4]

Rabin-Shallit algorithm for finding 4-tuple is mentioned at Idemix specifications (section 6.2.6 Protocol ProveInequality, step 1.2) and "[it] accounts for a substantial fraction of the computation time of this protocol".

This algorithm requires some background, in particular quaternions (Hurwitz integers). Gaussian integers (a kind of complex numbers) are relevant as a simple object to start from.

Comprehensive documents considered include Keith Conrad's overview [5] and an article [6]. Yijian Liu' Master thesis [7] is considered helpful. Entry-level reading include forum [8] and stackoverflow [9].

Current intermediate target is extended Euclidean algorithm for quaternions, C++ implementation on top of Crypto++. Having it implemented, algorithm from Trevino presentation should be straightforward.

'Final Algorithm' is shown on page 11, Trevino presentation. It includes calculation of greatest common divisors over Gaussian integers and quaternions.

[2] http://campus.lakeforest.edu/trevino/JMM2018.pdf

[3] http://campus.lakeforest.edu/trevino/finding4squares.pdf

[4] http://www.ams.org/journals/mcom/0000-000-00/S0025-5718-2018-03349-X/home.html

[5] http://www.math.uconn.edu/~kconrad/blurbs/ugradnumthy/Zinotes.pdf

[6] https://www.sciencedirect.com/science/article/pii/S0747717101905188

[7] https://mspace.lib.umanitoba.ca/bitstream/handle/1993/32638/Yijian_Liu.pdf

[8] http://mathforum.org/library/drmath/view/67068.html

[9] https://stackoverflow.com/questions/2269810/whats-a-nice-method-to-factor-gaussian-integer

13.1 Lipmaa algorithm and 'emmy' implementation

`https://github.com/xlab-si/emmy/blob/master/crypto/common/decomposition.go`
`https://github.com/xlab-si/emmy/blob/master/crypto/common/decomposition_test.go`

1. lipmaaDecompose() is the entrypoint; getSpecialDecomposition() is for small (0, 1, 2) and negative
2. powers of 2; recursive lipmaaDecompose()
3. 'working' non-recursive is
   findPrimeAndTwoRoots();
   sqrtOfMinus1();
   decomposePrimeToTwoSquares().
4. case of an even power of 2
5. findPrimeAndTwoRoots(n) to find random w_1 and w_2 such that $p = n - w_1^2 - w_2^2 = 4k + 1$ and is a prime. odd/even.
6. decomposePrimeToTwoSquares() two largest reminders of Euclid algorithm that are less than sqrt(p); sqrt(-1) as the input. `https://math.stackexchange.com/questions/5877/efficiently-finding-two-square 5883`
7.
8.

14 Verification as Solidity contract

Two potential non-critical issues were identified on the roadmap to Solidity contract verifying location proofs: avoiding negative integers in the exponent and non-interactive proofs with deterministic challenges (covered at section 10.1).

14.1 Complete proof variant for Solidity

A non-interactive proof variant can be adopted for Solidity (no random numbers while verification), modified to expect only positive exponents.

1. Prover calculates $a_1 \dots a_4$ from locations and threshold, picks

random $\alpha_j, \eta, \gamma, \beta_x, \beta_y, \beta_z, \beta_r, \rho_0, \rho_1$, calculates t_n, s_a, t_a, b_0, b_1

$$t_n = g_x^{\beta_x} g_y^{\beta_y} g_z^{\beta_z} g^{\beta_r} \tag{33}$$

$$s_a = g^{\gamma}(\prod_{j=1}^{4} h_j^{a_j}), \qquad t_a = g^{\eta}(\prod_{j=1}^{4} h_j^{\alpha_j}) \tag{34}$$

$$\tilde{f}_0 = \beta_x^2 + \beta_y^2 + \beta_z^2 + \alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2$$
$$\tilde{f}_1 = (x_n - x_l)\beta_x + (y_n - y_l)\beta_y + (z_n - z_l)\beta_z + a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + a_4\alpha_4$$
$$b_0 = g^{\tilde{f}_0} g_r^{\rho_0}, \qquad b_1 = g^{2\tilde{f}_1} g_r^{\rho_1} \tag{35}$$

2. Prover calclulates a challenge

$$c = H(t_n||s_a||t_a||b_1||b_0||s_U) \tag{36}$$

3. Prover computes responses

$$X_n = -cx_n + \beta_x,\ Y_n = -cy_n + \beta_y,\ Z_n = -cz_n + \beta_z,\ R = -cr + \beta_r$$
$$A_j = -ca_j + \alpha_j,\ R_a = -c\gamma + \eta,\ R_d = -c\rho_1 + \rho_0 \tag{37}$$

Non-interactive proof is $(c, X_n, Y_n, Z_n, R, \{A_j\}, R_a, R_d, s_a, b_1)$.

4. Proof verification

$$F_d = ((X_n + cx_l)^2 + (Y_n + cy_l)^2 + (Z_n + cz_l)^2) + (A_1^2 + A_2^2 + A_3^2 + A_4^2) - c^2 d^2$$
$$H(g_x^{X_n} g_y^{Y_n} g_z^{Z_n} g^R s_U^c||s_a||g^{R_a}(\prod_{j=1}^{4} h_j^{A_j}) s_a^c||b_1||g^{F_d} g_r^{R_d} b_1^c||s_U) = c \tag{38}$$

Verifying completeness property (not part of the protocol):

$$f_V(v) = f_2 v^2 - f_1 v + f_0 =$$
$$(((-vx_n + \beta_x) + vx_l)^2 + ((-vy_n + \beta_y) + vy_l)^2 + ((-vz_n + \beta_z) + vz_l)^2) +$$
$$((-va_1 + \alpha_1)^2 + (-va_2 + \alpha_2)^2 + (-va_3 + \alpha_3)^2 + (-va_4 + \alpha_4)^2) - v^2 d^2 =$$
$$v^2(((x_n - x_l)^2 + (y_n - y_l)^2 + (z_n - z_l)^2) + (a_1^2 + a_2^2 + a_3^2 + a_4^2) - d^2) -$$
$$2v(((x_n - x_l)\beta_x + (y_n - y_l)\beta_y + (z_n - z_l)\beta_z) + (a_1\alpha_1 + a_2\alpha_2 + a_3\alpha_3 + a_4\alpha_4)) +$$
$$((\beta_x^2 + \beta_y^2 + \beta_z^2) + (\alpha_1^2 + \alpha_2^2 + \alpha_3^2 + \alpha_4^2)) \tag{39}$$

14.2 Data flow

Non-Interactive proof shown above will be implemented in C++ as library functions

```
std::string ni_proof_create(
  const double xn, const double yn, const double zn,
  const double xl, const double yl, const double zl,
  const double d);
bool ni_proof_verify(const std::string proof);
```