Wanchain Yellow Paper

Version 1.0.0

CONTENT

1 Basic knowledge of cryptography

1.1 Elliptic Curve Cryptography

1.1.1 Introduction to elliptic curve cryptography

E is an elliptic curve defined over G which is a finite field. Actually it is a set of points:

$$E/G = \{(x, y) \mid y^2 + a_1xy + a_3y = x^3 + a_2x_2 + a_4x + a_6,$$
$$a_1, a_3, a_2, a_4, a_6, x, y \in G\} \cup \{O\},\ O \text{ is the point at infinity}$$

Addition operation is defined on elliptic curves as group law. P and Q represents two points on the elliptic curve E. $P + Q = R$ means that R is the symmetric point of the intersection of E and the line across P and Q with respect to the x-axis. If $P = Q$, R is the symmetric point of the intersection of E and the tangent line across P. Therefore $(E, +)$ is an Abel group defined over G with the identity O.

Given$P = (x_1, y_1) \in E, Q = (x_2, y_2) \in E$, if $x_1 = x_2$ and $y_1 = y_2$, then $P + Q = O$. Otherwise $P + Q = (x_3, y_3)$ with $x_3 = \lambda^2 - x_1 - x_2, y_3 = \lambda(x_1 - x_3) - y_1$:

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, P \neq Q \\ \dfrac{3{x_1}^2 + a}{2y_1}, P = Q \end{cases}$$

1.1.2 Elliptic Curve Digital Signature Algorithm

$Ep(a, b)$ is an elliptic curve whose order is N with base point P. Q is an arbitrary point on $Ep(a, b)$ satisfying $Q = kP$. Then $(k,\ Q)$ becomes the key pair with k as a private key and Q as a public key.

ECDSA between sender A and receiver B is shown below:

Step1: Calculate the hash of plain text M using hash function such as MD5 or SHA-1: $t = H(M)$.

Step2: Take a random integer k as the private key in $[1, N-1]$.

Step3: Calculate the public key $Q = kP$.

Step4: Calculate $r = Q_x \ mod \ N$. Q_x is the abscissa of Q. If $r = 0$, return to Step2.

Step5: Calculate $s = k^{-1}(t + rk) \ mod \ N$. k is the private key of A. If $s = 0$, return to Step2.

Step6: A delivers (r, s) to B as a signature.

The verification of (r, s) by B is shown below:

Step1: Verify that whether r and s are positive integers in $[1, N-1]$. If not, the signature is invalid.

Step2: Calculate the hash of plain text M: $t = H(M)$.

Step3: Calculate $e = s^{-1} \ mod \ N$.

Step4: Calculate $u = te \ mod \ N$.

Step5: Calculate $v = re \ mod \ N$.

Step6: Calculate $R = uP + vQ$.

Step7: If $R = 0$, B rejects the signature. Otherwise calculate $r' = R_x \ mod \ N$. R_x is

the abscissa of R.

Step8: If $r' = r$, the signature is valid. Otherwise it is invalid.

There is no inverse in ECDSA when calculate the order. So it is easier than the signature algorithm based on discrete logarithm problem. Meanwhile ECDSA is more efficient than Schnorr digital signature algorithm because it is easier to calculate $H(M)$ than $H(M, R)$. Therefore ECDSA is better on safety strength, keylength,

efficency, computitional cost and band width of attack resistance.

1.2 Threshold Key Sharing Scheme

1.2.1 Shamir's threshold key sharing scheme

Threshold key sharing scheme solves the problem of secure key management. The design of modern cryptography mechanism correlates the security of crypto-system to the safety of secret key decisively. Divulging of the secret key means the loss of safety, and thus the key management takes important position in security research and crypto-system design. Especially when one account is managed by multi-parties of different interests, how to distribute the key to multiple participators in credibility and safety will become very troublesome. Concerning this problem, Shamir, an Israeli cryptographer, proposed Shamir's (k, n) threshold key sharing concept: the key is divided as *n* parts to be distributed to *n* participators, and each participator holds one key share. Only when *k* key shares are gathered, will the key be reconstructed.

1.2.2 Linear key sharing scheme

Linear key sharing is the extension of Shamir's threshold key sharing scheme, whose essence is the request that main key space, sub-key space and randomly-input assemble are all linear space, and key construction function is also linear. Its formal definition is as follows:

Suppose that K is a finite field, Π is is a key sharing system that realizes Access Structure (AS), and $S \subset K$ is the main key space, then we say that Π will be a linear key sharing system over K, if the following requests are met:

- Sub-key space is the linear space over K, i.e. that $\forall i$ and $\exists$ constant d_i make sub-key space $S_i \subset K^{d_i}$. Record $\Pi_{i,j}(s, r)$ as the No. j component of the vector received by P_i from space K^{d_i}, and this component is dependent on main key s, and random number r.

- Each grant set can gain the main key through the linear combination of sub-keys, i.e. That any grant set $G \in AS$, $\exists$ constant as $\{a_{i,j}: P_i \in G, 1 \leq j \leq d_i\}$ makes any main key s, and random r, with the formula below:

$$s = \sum_{P_i \in G} \sum_{1 \leq j \leq d_i} a_{i,j} \cdot \Pi_{i,j}(s, r).$$

1.2.3 Shamir's threshold key sharing scheme based on Lagrange's polynomial interpolation

Shamir designs threshold key sharing scheme based on Lagrange's polynomial interpolation by combining properties of polynomial over finite field and Lagrange's reconstruction polynomial theory. Concrete scheme as follows:

- Key sharing process

1) When sharing key s, the key owner generates a random $k-1$ degree polynomial over finite field: $f(x) = s + a_1 x + \cdots + a_{k-1} x^{k-1}$, obviously $f(0) = s$.

2) The key owner selects random number $x_1, x_2 \ldots, x_n$ and computes $f(x_1), f(x_2) \ldots, f(x_n)$.

3) Each participant P_i gets its key share $s_i = (x_i, f(x_i))$

- Key reconstructing process

From key sharing process we know $s_i = (x_i, f(x_i))$ is a point of curve $f(x)$. According to Lagrange's polynomial interpolation method, any k points of $f(x)$ will reconstruct $f(x)$:

$$f(x) = \sum_{i=1}^{k} f(x_i) \prod_{j=1, j \neq i}^{k} \frac{x - x_j}{x_i - x_j}$$

set $x = 0$, then s is reconstructed:

$$s = \sum_{i=1}^{k} f(x_i) \prod_{j=1, j \neq i}^{k} \frac{x_j}{x_i - x_j}$$

simply:

$$s = \sum_{i=1}^{k} b_i f(x_i)$$

where

$$b_i = \prod_{j=1, j\neq i}^{k} \frac{x_j}{x_i - x_j}$$

1.3 Secure Multi-Party Computation

1.3.1 Proposal background of secure multi-party computation

With the rapid development of Internet, more and more application scenarios need the coordinative computing between Internet users. However, in consideration of privacy protection and data safety, the users participating in coordinative computing do want to share the computing data with other users, which causes the coordinative computing operation difficulty, and thereby leading to the fact that the internet resources cannot be shared and utilized in high efficiency and some application scenarios cannot be realized. Secure multi-party computation solved the problem and also provided the theoretical basis to solve the conflict between data-privacy protection and coordinative computing.

Secure multi-party computation is both the theoretical basis of distributed cryptography and a fundamental problem of distributed computing. It solves the problem that users accomplish a task by coordinative computing without leaking any private information in a mutual distrusting multi-user network. Simply speaking, secure multi-party computation is about a group of participants, referred as $P_1 \cdots\cdots P_n$, who work together to compute function $f(x_1, \ldots\ldots, x_n) = (y_1, \ldots\ldots, y_n)$ securely. The n inputs of function f are held by n participants and one for each. Assuming P_i holds the secret input x_i and after the computation P_i gets output y_i. In this situation, security requires that even some participants cheat in the computing process, the correctness of the computing result is still ensured. That means every participant gets the right output y_i when the computing is finished and all participants' input

remain privileged. P_i gets no extra information except (x_i, y_i) through the computation.

So far, the theory of secure multi-party computation has been obtained rich theory and application results. It will be an indispensable part of computer security.

1.3.2 The classification of secure multi-party computation protocol

At present secure multi-party computation protocol can be divided into four categories according to different implementation methods:

- Secure multi-party computation protocol based on VSS sub-protocol

Most existing secure multi-party computation protocol adopts Verifiable Secret Sharing (VSS) sub-protocol as the basis of protocol construction, which is suitable for computing functions over any finite field. Any function over a finite field can be represented by a directed graph of addition and multiplication of the filed, thus as long as the addition and multiplication can be securely calculated, the calculation of any function over the finite field can be completed by computing each addition and multiplication.

- Secure multi-party computation protocol based on Mix-Match

Secure multi-party computation protocol based on VSS can calculate any function, but it cannot efficiently calculate Boolean function, therefore another secure multi-party protocol is proposed---Mix-Match. The basic idea of this protocol is that the participant uses the secret sharing scheme to share the private key, and the public key of the system is open. During the process of protocol, the participant randomly encrypts the public key used by himself and then open his own encrypted result, and finally all participants get the common output through Mix-Match.

- Secure multi-party computation protocol based on OT

Secure multi-party computation protocol based on OT is used to calculate any bit-wise operation function. It uses the OT sub-protocol to realize the three basic operations of “and”, “or”, and “not”, and then decomposes any bit-wise operation

function into a combination of three basic operations, and finally calculates any bit-wise operation function in an iterative way.

- Secure multiparty computation protocol based on Homomorphic encryption

Secure multi-party computation protocol based on Homomorphic encryption can resist against active attacks, whose idea is to select atomic computation so that any calculation of function can be decomposed into the sequences of atomic computation, while the input and output of atomic computation are both encrypted using the homomorphic encryption algorithm, the final results can get in the encrypted state, and only specific participants can get the results for clear text.

1.4 Introduction To The Ring Signature

Ring Signature was provided first in 2001 by Rivest. It is a special kind of Group Signature. While Group signature needs a trustable and secure setup. This causes that the signer can be traced by the trust center. Ring Signature solves this serious problem by removing the trust center and secure setup. It is significant for the signer that he is absolutely anonymous to anyone.

Since Ring Signature provided, many practical scheme has been designed based on ECC(Elliptic Curve Cryptography), trapdoor technique and so on. Generally speaking, there are 4 kinds of Ring Signature:

- Trapdoor Ring Signature
- Linkable Ring Signature
- Anonymity revocable Ring Signature
- Deniable Ring Signature

In order to provide the anonymity in Smart Contract Token Transaction, a kind of Ring Signature based on ECC is implemented on Wanchain.

2 Locked Account System

2.1 Basic operation of secure multi-party computation

Addition, multiplication, and unary inverse are the three basic operations over a finite field. Any calculation can be decomposed into a sequence of operations of addition, multiplication, and unary inverse operations over this finite field. So as long as the multi-party computation over the finite field can complete the three basic operations, any calculation process can be obtained in an iterative way through the basic multi-party computation. In the following, basic operation of secure multi-party computation algorithm will be introduced under a secret sharing scheme based on Lagrange's polynomial interpolation.

2.1.1 Addition

Under a secret sharing scheme based on Lagrange's polynomial interpolation, it is necessary to determine a polynomial. The shared secret is the constant term of the polynomial, and a secret share is the value of this polynomial in some point. Without loss of generality, assume that α, β are two shared secret, the corresponding polynomials are $f_\alpha(x)$、$f_\beta(x)$, the participant P_i has the secret share $\alpha_i = f_\alpha(i)$, $\beta_i = f_\beta(i)$. The participant P_i wants to get the secret share of $\alpha + \beta$，he needs to construct a polynomial $f(x)$, assuring the constant term of this polynomial is $\alpha + \beta$，and P_i can calculate to get $f(i)$. The construction process is as follows：

∵ α_i、β_i are the secret share of α、β，and the corresponding polynomial are $f_\alpha(x)$、$f_\beta(x)$

∴ $f_\alpha(x) = \alpha + a_{\alpha,1}x + \cdots + a_{\alpha,k-1}x^{k-1}$ $\quad$ $f_\beta(x) = \beta + a_{\beta,1}x + \cdots + a_{\beta,k-1}x^{k-1}$

Define $f(x) = f_\alpha(x) + f_\beta(x)$, then $f(i) = f_\alpha(i) + f_\beta(i) = \alpha_i + \beta_i$

Obviously $f(x)$ is a polynomial of $k-1$, and the constant term is $\alpha + \beta$, $f(i)$ is the value of this polynomial in i point.

$\therefore\ \gamma_i = f(i)$ is the secret share of $\alpha + \beta$

A secure multi-party computation protocol for addition operation can get from the construction process above.

MPC Protocol For Addition Operation

Input：α's secret share α_i, β's secret share β_i

Output： $(\alpha + \beta)'$s secret share γ_i

1) $\gamma_i = \alpha_i + \beta_i$

2.1.2 Multiplication

Define α, β as two shared secrets, the corresponding polynomial are $f_\alpha(x)$, $f_\beta(x)$. the participant P_i has the secret share, respectively $\alpha_i = f_\alpha(i)$, $\beta_i = f_\beta(i)$. If the participant directly computes the products of secret shares α_i, β_i locally, although after computing, the constant term of polynomial of shared secret $\alpha\beta$ is actually $\alpha\beta$, the degree of polynomial is $2(k-1)$, so we need to lower the degree of polynomial.

$f_\alpha(i)$, $f_\beta(i)$ are the secret share that possessed by the participant P_i, the product of $f_\alpha(x)$ and $f_\beta(x)$ is：

$$f_{\alpha\beta}(x) = f_\alpha(x) f_\beta(x) = \alpha\beta + a_1 x + \cdots + a_{2(k-1)} x^{2(k-1)}$$

$$f_{\alpha\beta}(i) = f_\alpha(i) f_\beta(i),\ \ 1 \le i \le 2(k-1) + 1$$

Matrix representation：

$$\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & (2k-1)^{2(k-1)} \end{bmatrix} \begin{bmatrix} \alpha\beta \\ \vdots \\ a_{2(k-1)} \end{bmatrix} = \begin{bmatrix} f_{\alpha\beta}(1) \\ \vdots \\ f_{\alpha\beta}(2k-1) \end{bmatrix}$$

Take the upper coefficient matrix as A, obviously A is a nonsingular matrix. Then take the inverse matrix of A as A^{-1}, which is a constant matrix. Take the $(\lambda_1, \cdots, \lambda_{2k-1})$ as

the first line of matrix A^{-1}, then：

$$\alpha\beta = \lambda_1 f_{\alpha\beta}(1) + \cdots + \lambda_{2k-1} f_{\alpha\beta}(2k-1)$$

Each participant randomly selects $k-1$ of the $2k-1$ polymerization $h_1(x), \cdots, h_{2k-1}(x)$, requires to meet $h_i(0) = f_{\alpha\beta}(i)$. Define$H(x) = \sum_{i=1}^{2k-1} \lambda_i h_i(x)$, obviously：

$$H(0) = \sum_{i=1}^{2k-1} \lambda_i h_i(0) = \lambda_1 f_{\alpha\beta}(1) + \cdots + \lambda_{2k-1} f_{\alpha\beta}(2k-1) = \alpha\beta$$

$$H(j) = \sum_{i=1}^{2k-1} \lambda_i h_i(j)$$

So $H(x)$ is the polymerization of sharing secret of $\alpha\beta$, and $H(i)$ is the secret share.

MPC Protocol For Multiplication Operation

Input：α's secret share α_i, β's secret share β_i

Output： $(\alpha\beta)'$s secret share γ_i

1) P_i selects a random $k-1$ degree polynomial $h_i(x)$, which satisfies $h_i(0) = \alpha_i\beta_i$.

2) P_i computes $h_i(j)$ and sends it to P_j, $1 \le j \le 2k-1$

3) Each participant P_i collects secret shares from other participants, and computes $\gamma_i = H(i) = \sum_{j=1}^{2k-1} \lambda_j h_j(i)$, then P_i gets its secret share for $\alpha\beta$: γ_i.

2.1.3 Unary Inverse

Define α as the shared secret, the corresponding polynomial is $f_\alpha(x)$, the participant P_i has the secret share $\alpha_i = f_\alpha(i)$. Unary inverse computation refers to the participant P_i gets the secret share $f_{\alpha^{-1}}(i)$ of α^{-1} through the computing of secret share α_i, and in the process of computing α, α^{-1} and their secret share cannot be

leaked. The algorithm is as follows:

The participant P_i selects random number r_i, and polynomial $g_i(x)$, computes its secret share $r_{ij} = g_i(j)$ and sends to the participant P_j. After receiving all secret shares, P_j computes $r_j' = \sum_{l=1}^{n} r_{l,j}$. Thus all participants share the same random number $r = r_1 + \cdots + r_n$. Using MPC protocol for multiplication operation, the secret share $f_{\alpha r}(i)$ of αr can be obtained through computing the secret shares of α and r, and at the same time send the result to other participants. Thus it can also use the Lagrange's polynomial interpolation to reconstruct αr. Define $m = \alpha r$, Obviously let $f_{\alpha^{-1}}(i) = m^{-1} r_i'$, which is the secret share of α^{-1}.

MPC Protocol For Unary Inverse Operation

Input：α's secret share α_i

Output： $(\alpha^{-1})'$s secret share γ_i

1) P_i selects random number r_i and random polynomial $g_i(x)$, it computes $r_{ij} = g_i(j)$ and sends it to P_j, $1 \le j \le n$.

2) P_j collects all secret shares and computes: $r_j' = \sum_{l=1}^{n} r_{l,j}$

3) Using MPC protocol for multiplication operation, compute $\alpha r'$s secret share $f_{\alpha r}(i)$ and reconstruct αr.

4) Set $m = \alpha r$, $\gamma_i = f_{\alpha^{-1}}(i) = m^{-1} r_i'$, then γ_i is $(\alpha^{-1})'$s secret share for P_i.

2.2 Locked Account Generation Scheme

Locked account generation scheme marks an improvement of threshold key sharing scheme based on Lagrange's polynomial interpolation. Its basic idea is to generate locked account with all verification nodes by decentralization through threshold key

sharing, and each verification node is mastering a key share of locked account private key. This ensures that the locked account private key exist in the whole network in the form of distribution, so it can be managed in the manner of decentralization. The specific algorithm is as follows:

Locked Account Generating Algorithm

1) P_i selects random number d_i and broadcasts d_iG through the network, G is the base point of elliptic curve.

2) P_i selects random $k-1$ degree polynomial：$f_i(x) = d_i + a_{i,1}x + \cdots + a_{i,k-1}x^{k-1}$, and sends $f_i(j)$ to P_j through secure channel, and broadcasts $a_{i,1}G \cdots\cdots a_{i,k-1}G$ through the network.

3) P_j verifies $\sum_{t=0}^{k-1} j^t\, a_{i,t}G = f_i(j)G$, if the equation is valid, then P_j accepts; else P_j rejects and asks P_i to resend the message.

4) When all messages are sent and accepted, each participant gets its key share: $t_s = \sum_{j=1}^{n} f_j(s),\ s = 1, \ldots\ldots, n$

5) (k, n) threshold Locked Account address is: $address = Hash(Q)$, $Q = \sum_{i=0}^{d} d_i\, G$, its $privatekey = \sum_{i=0}^{d} d_i$, reconstructing this private key requires at least k key shares.

2.3 Locked Account Signature Scheme

Locked account signature scheme uses the ECDSA signature algorithm, for it's the mainstream signature algorithm for the current blockchain project, which improves the system compatibility. In the process of locked account signature generation, unlike the original ECDSA signature algorithm, account private key and random number participate in the process of ECDSA signature in a MPC form. The verification

process of locked account signature is the same as the original ECDSA signature verification algorithm. So only the process of locked account signature generation is introduced in the following:

Locked Account Signature Verifying Algorithm

1) All Participants share a same random number c through MPC, P_i' random share is c_i.

2) P_i computes $R_i = c_i G$ and broadcasts it.

3) When all messages are broadcasted, P_i computes $(x, y) = \sum_{j=1}^{k} b_j R_j$, $r = x \ mod \ p$, $b_j = \sum \frac{j}{j-i}$.

4) P_i computes $(c^{-1})'$ share ω_i using MPC protocol for unary inverse operation.

5) Using MPC protocol for multiplication operation, P_i computes $(c^{-1}d)'$ share v_i via ω_i and t_i, d is the private key of Locked Account, t_i is d' key share for P_i.

6) Afterwards, P_i computes $s_i = \omega_i m + v_i \ r$, P_i gets its signature share s_i and broadcasts it.

7) P_i computes and verifies $R_j = u_{j1} G + u_{j2} Q_j$, $u_{j1} = m s_j^{-1}$, $u_{j1} = r s_j^{-1}$, $Q_j = t_j G$. If the equation is valid, then P_i accepts signature share s_j; else P_i rejects s_j.

8) When P_i collects more than k signature shares, then it reconstructs complete signature (r, s) using Lagrange's polynomial interpolation.

2.4 Locked Account Key Share Update Scheme

The threshold key sharing scheme based on Lagrange's polynomial interpolation adopted by this protocol belongs to the linear key sharing scheme, so key sharing shall satisfy the homogeneity: key_1's (k,n) threshold key share is $(a_1,\cdots,a_n)$, key_2's (k,n) threshold key share is $(b_1,\cdots,b_n)$, then $(a_1+b_1,\cdots,a_n+b_n)$ is the threshold key share of key_1+key_2. If let $key_2=0$, then we can obtain the new (k,n) threshold key share of key_1. The specific algorithm is as follows:

Locked Account Key Share Updating Algorithm

1) Each Participant P_i selects random polynomial to share value 0 and computes 0's share: $(f_i(1),\cdots,f_i(n))$
2) P_i sends $f_i(j)$ to P_j through secure channel, $j=1,\ldots\ldots,n$
3) When all messages are sent and verified, P_i gets $(f_1(i),\cdots,f_n(i))$, then P_i's new key share is: $t_i^{new}=t_i+\sum_{j=1}^{n}f_j(i)$

3 Smart Contract Token Transaction Anonymity

3.1 Ring Signature

The ring signature can be divided into four parts: **GEN**, **SIG**, **VER**, **LNK**.

GEN: Using the public parameters, randomly select $n-1$ public keys, along with user's public key P constitutes the public key set $S=\{P_i|i=1,2\ldots,\}$. For user's public and private key (P,x), $x\in[1,l-1]$, l is the order of point P. Generates public key image I.

SIG: Aiming at the need to sign message m, use public key set $S=\{P_i|i=1,2\ldots,n\}$, where P_s is use's true public key. Compute signature $ringsig$.

VER: Based on information m, public key set S and signature $ringsig$, verify the validity of signature, output "True" or "False".

LNK: Using the set $\mathcal{I} = \{I_i\}$, decide whether the signature $ringsig$ has been used.

The specific process is described as follows:

GEN: Signer uses the key pair (P, x), $P = xG$, computes $I = xH_p(P)$, where H_p is hash function, which outputs a random point on ECC. Then signer randomly selects $n-1$ public keys along with P to constitute public key address set $S = \{P_i | i = 1,2 \dots, n\}$, where $P_s = P$.

SIG: Signer selects random number $\{q_i | i = 1,2, \dots, n\}$ and $\{\omega_i | i = 1,2, \dots, n, i \neq s\}$, computing as follows:

$$L_i = \begin{cases} q_i G, & if \quad i = s \\ q_i G + \omega_i P_i, & if \quad i \neq s \end{cases}$$

$$R_i = \begin{cases} q_i H_p(P_i), & if \quad i = s \\ q_i H_p(P_i) + \omega_i I, & if \quad i \neq s \end{cases}$$

Computing:

$$c = H_s(m, L_1, \dots, L_n, R_1, \dots, R_n), \quad H_s \text{ is hash function.}$$

Computing:

$$c_i = \begin{cases} \omega_i, & if \quad i \neq s \\ c - \sum_{i=1, i \neq s}^{n} c_i \mod l, & if \quad i = s \end{cases}$$

$$r_i = \begin{cases} q_i, & if \quad i \neq s \\ q_s - c_s x \mod l, & if \quad i = s \end{cases}$$

Finally generate signature:

$$ringsig = (I, c_1, \dots, c_n, r_1, \dots, r_n)$$

VER: When the verifier verifies the signature, using message m, public parameters and $S = \{P_i | i = 1,2 \dots, n\}$, $ringsig = (I, c_1, \dots, c_n, r_1, \dots, r_n)$, computing:

$$\begin{cases} L_i' = r_i G + c_i P_i, \\ R_i' = r_i H_p(P_i) + c_i I \end{cases}$$

Then verify whether the equation $\sum_{i=1}^{n} c_i = H_s(m, L_1', \dots, L_n', R_1', \dots, R_n') \ mod \ l$ holds, if the equation holds, then the signature is valid, implement LNK.

Explanation: in the process of verification, if the equation holds, then $L_i' = L_i$, $R_i' = R_i$, take L_i' as an example to explain:

$$L_i = \begin{cases} q_i G, & if \quad i = s \\ q_i G + \omega_i P_i, & if \quad i \neq s \end{cases}$$

$$L_i' = r_i G + c_i P_i,$$

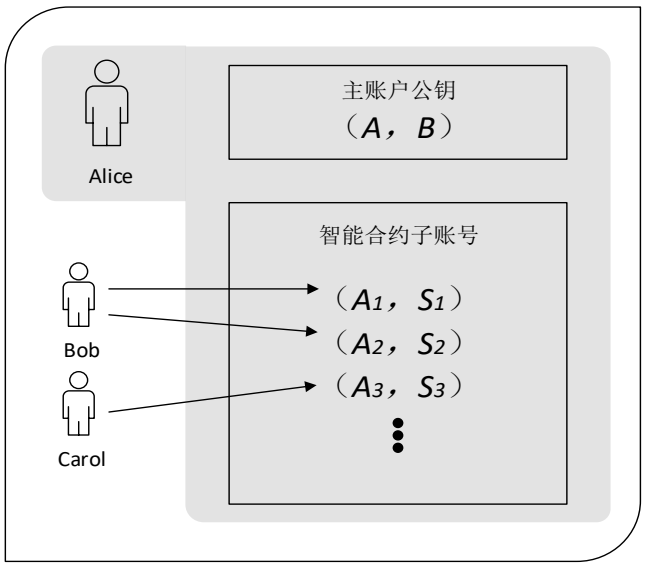
$i \neq s$, has $q_i = r_i$, $\omega_i = c_i$, obviously holds, however when $i = s$: $L_s = q_s G = (r_s + c_s x)G = r_s G + c_s P_s = L_s'$, so $L_i' = L_i$, similarly $R_i' = R_i$ holds, then the process of VER is valid.

LNK: Set $\mathcal{I}$ constituted by all I that appeares on the blockchain, if the current I appears in the set, it shows that the public key has been used, so the signature of transaction is illegal; if I does not appear in the set, then the signature of transaction is legal, and add the I into the set $\mathcal{I}$.

3.2 Onetime-account System

3.2.1 Constitution of One-time account system

One-time account system is the basis of the entire privacy transaction, every user in this system has a single main account and multiple sub-accounts. Sub-account can also be considered as the account in smart contract. Usually a sub-account is generated by a transaction sender for the receiver. As is shown in the following chart.



Alice had a unique main account (A, B), and several sub-accounts (A_1, S_1), (A_2, S_2), (A_3, S_3), ... When Bob makes a transaction to Alice, (A_1, S_1) and (A_2, S_2) is created through main account (A, B), when Carol makes a transaction to Alice, (A_3, S_3) is created. It can be seen that every transaction to Alice will create a sub-account for Alice, which is One-time account. However only Alice has the ability to manage and use these sub-accounts.

3.2.2 Account Generation Algorithm

$E(\mathbb{F}_p)[r]$ is the subgroup of order r of elliptic curve $E/\mathbb{F}_p$. G is the base point of $E(\mathbb{F}_p)[r]$. Choose the elliptic curve of Bitcoin or Ethereum $y^2 = x^3 + 7$.

- Generation of Main Account

The original account of Alice on Wanchain is (A, a), where $A = [a]G$. To generate main account of one-time account system, select random number $b \in [1, r-1]$, $B = [b]G$. (a, b) is used as the private key of Alice's main account, and (A, B) is used as the public key of Alice's main account. Finally, Alice owns the private key (a, b) and scan key (A, b). (A, B) is disclosed as the address of main account.

- Generation of sub-account

When Bob makes a transaction to Alice, Alice's main account (A, B) will be used to generate sub-accounts (A_1, S_1).

Bob generates random number $s \in [1, r-1]$, compute

$$S_1 = [s]G$$

$$A_1 = A + [\text{Hash\_p}([s]B)]G$$

Hash_p above is the function for mapping the points of elliptic curve to $[1, r-1]$. The sub-account generated by Alice is

$$(A_1, S_1)$$

(A_1, S_1) is an one-time account. The random number s remains undisclosed while S_1 which includes s information is disclosed. s cannot be calculated through S_1 and it is ensured by the problem of elliptic curve discrete logarithm. A_1 is a component of public key of sub-account and S_1 is the random factor component.

Each sub-account of Alice is the random account based on main account (A, B).

- Verification of sub-account

Alice scans all one-time accounts in the blockchain. Using S_1 and scan key (A, b) compute:

$$A_1' = A + [\text{Hash}([b]S_1)]G$$

If $A_1' = A_1$, (A_1, S_1) can be confirmed as Alice's sub-account.

If $A_1' \neq A_1$, it can confirm that (A_1, S_1) is not Alice's sub-account.

This is because $B = b\,G$, thus

$$[s]B = [sb]G = [b][s]G = [b]S_1$$

Alice is the unique person who has the scan key (A, b), so only Alice can verify the sub-account (A_1, S_1) belongs to her or not.

When Alice spends the assets of the sub-account (A_1, S_1), she needs to calculate the corresponding private key x of (A_1, S_1):

$$x = a + \mathrm{Hash}([b]S_1)$$

So with the public key component $A_1 = [x]G$ of a sub-account, Alice can calculate the corresponding private key.

3.3 Stamp System Based on One-time Account

The above text reveals that the sender of a transaction is untraceable when the privacy is protected by ring signature. Thus it results in the problem of which account shall pay the transaction fee. In order to solve this, the stamp system based on one-time account is designed and realized on Wanchain. Each stamp is an one-time account in the stamp system, referred as $\mathrm{Onetime-stamp}$. Technically $\mathrm{Onetime-stamp}$ is the same as $\mathrm{Onetime-account}$. Users need to purchase stamps beforehand when conducting private transaction. Stamps are pasted on the transactions to complete the private transaction and each stamp can only be used for once.

Stamp system is a smart contract with multiple face values. The stamp information of the corresponding face value purchased by the user is stored in each face value storage. The contact provides the stamp purchasing and returning functions:

Stamp purchasing function: the stamp purchasing function can provide the service for purchasing stamp by Wancoin. If user $Account$ needs to purchase a stamp, he needs to send a transaction to transfer Wancoin of the value with pre-purchased stamp to the smart contract of the stamp system and invoke the purchasing function in the contract. Purchasing function's parameter is the one-time account generated by the user, which will be stored in the list of stamps of the corresponding value to prove that the user has purchased the stamp.

$$Tx = (Account, StampSC, value, payload, sig)$$

$$payload = ("buy", Onetime - stamp)$$

Stamp returning function: the stamp returning function can offer the user the service of returning the unused stamp. If user $Account$ has purchased one-time stamp and the stamp is not used, then the user can get the Wancoin with the corresponding value back to his account by invoking the stamp returning function in the stamp system smart contract. The returning function's parameter are $Onetime-stamp$, stamp face value and ring signature. When the returning transaction is confirmed, the stamp will be deleted from the corresponding value list to prove that the stamp has been returned:

$$Tx = (Account, StampSC, payload, sig)$$

$$payload = ("refund", Onetime-stamp, value, ringsig)$$

$$ringsig = (I, c_1, c_2, r_1, r_2)$$

The reason of using ring signature in the stamp returning function is to require the user to provide the corresponding I value of the stamp to ensure that the stamp is not used and the situation where the stamp is returned after being used does not occur.

After stamp system is realized, when the user makes a private transaction, he needs to purchase a stamp first in the stamp system. Face value of the stamp is decided by the calculation quantity of the smart contract and user's will. As the account information of the user does not occur in Tx of token transaction, the set of stamp used by the user and randomly selected stamp with same face value serves as the transaction sender. Ring signature of user's stamp is used to ensure transaction is valid.

$$OTA\_Tx = (StampSet, TokenSC, payload, ringsig_{StampSet})$$

The transaction fee deducted is the face value of the used stamp. Each stamp can only be used for once. When the miner excavates new blocks, the mining award is made up by the former part of $Coinbase$ and the stamp value used in private transactions of the block. So the stamp value of $Coinbase$ is increased when compared with former condition. In order to ensure the fixed amount of Wancoin of the system, we set the address of the stamp smart contract as a fixed value, such as hash result of string $WANChainStampSystem$, so that nobody has private key of the contract and the

received Wancoin cannot be transferred, which means that the money in the stamp system contract is locked and the amount of Wancoin is constant against the increase of $Coinbase$.

3.4 Privacy Protection Scheme of Native Coin Transaction

The transaction privacy protection scheme of native coin is similar to the realization method of the stamp system and it is realized by deploying a corresponding Wancoin smart contract on Wanchain.

3.4.1 Wancoin smart contract

Wancoin smart contract is a smart contract deployed on Wanchain and similar to the stamp system with dispersed value token money is set which corresponds to the Wancoin in a ratio of 1:1. Wancoin smart contract provides two functions: token purchasing function and token returning function. The token purchasing function allows the user to transfer Wancoin to the contract to get equal value tokens into the one-time account provided by them. The one-time account provided by the user will be added to the storage list of the corresponding face value in the contract. The token returning function allows the user to invoke it by providing the ring signature of his one-time account. When the process is successful, Wancoin with the corresponding face value will be returned to the account of the user. Privacy protection process of Wancoin transaction is explained in details below.

3.4.2 Transaction scenario

The private key of main account of user 1 is (a, b), the corresponding public key is (A, B). The private key of main account of user 2 is (c, d), the corresponding public key is (C, D). User 1 will transfer Wancoin of $value$ to user 2. This is the transaction scenario.

3.4.3 Transaction flow

- Transaction initiating

User 1 builds one-time account $Onetime - account2$ by using the main account public key (C, D) of user 2. Then the token purchasing function of the contract $WANCoinSC$ is invoked using one-time account as its parameter:

$$Tx = (Account1, WANCoinSC, value, payload, sig)$$
$$payload = ("buy", Onetime - account2)$$

- Confirmation of transaction initiating

Validator receives the transaction to verify the relationship between sig and $Account1$. If sig is valid, the Wancoin smart contract is invoked and $Onetime - account2$ is stored in the account list of the corresponding value in the contract.

- Transaction receiving

User 2 scans the storage list of $WANCoinSC$ using his scan key. He finds that $Onetime - account2$ belongs to himself and then selects $n - 1$ one-time account in the corresponding face value list to generate ring signature of $Onetime - account2$. This ring signature is used as the token returning function's parameter:

$$Tx = (Account2, WANCoinSC, payload, sig)$$
$$payload = ("refund", OTASet, ringsig)$$

- Confirmation of transaction receiving

Validator receives the transaction and verifies the relationship between sig and $Account2$. If sig is valid, the Wancoin smart contract is invoked. The contract will verify the validity of the ring signature and make sure that I value in $ringsig$ does not occur. After all the confirmation, the Wancoin with corresponding value is transferred to $Account2$.

3.4.4 Analysis on privacy effect

Wancoin privacy transaction scheme based on one-time account and ring signature can realize the following private effects:

1) By using one-time account, no one knows to which user the token is transferred to during the transaction process.

2) By using ring signature, no one knows which account's token the receiver is drawing out during the transaction receiving process. At the same time, it must ensure that token will never be repeatedly withdrawn from the same one-time account.

In this scheme, relationship between the transaction initiator and the onetime account purchasing the token can be set up. But relationship between the transaction receiver and the onetime account cannot be set up. Thus the initiator cannot be corresponding to the receiver and the privacy protection is realized. To make a convenient use of ring signature, the storage with several fixed dispersed face values will be set in Wancoin smart contract. The private transaction with arbitrary face value cannot be realized, which will be researched in follow-up study.

3.5 Privacy Protection Scheme of Smart Contract Token

3.5.1 Transaction scenario

The private key of main account of user 1 is (a, b). The public key of main account is (A, B). The private key of main account of user 2 is (c, d). The public key of main account is (C, D). The account of user 1 in the smart contract is $Onetime - account1$. User 1 wants to transfer token worth $value$ to user 2. This is the transaction scenario.

3.5.2 Transaction initiating

User 1 needs to purchase the stamp which is recorded as $Onetime - stamp$ on the

transaction to make it valid. The transaction comprises four fields: TransFrom, TransTo, Data and RingSig. Then it is transmitted by P2P network. The construction process is as follows:

Transaction Initiating Process

1) $n-1$ stamps with the same value of $Onetime-stamp$ are randomly selected in the stamp system to constitute $StampSet$ together with $Onetime-stamp$. TransFrom is $StampSet$. This is the preparation for ring signature and the anonymity of the transaction sender. The node P_i sends $f_i(j)$ to P_j, $j=1,\dots\dots,n$ by a secure channel.

2) TransTo is the address of smart contract TokenSC.

3) Data comprises four fields: SC_TransFrom, SC_TransTo, SC_Value and SC_Sig. The construction process is as follows: firstly, $\mathrm{Onetime}-\mathrm{account2}$ is constructed for user 2. SC_TransFrom is $\mathrm{Onetime}-\mathrm{account1}$ of use 1. The transaction target addresses SC_TransTo is $\mathrm{Onetime}-\mathrm{account2}$ of user 2. SC_Value is value. User 1 calculates sig of the transaction which corresponds to $\mathrm{Onetime}-\mathrm{account1}$.

4) The construction of transaction field RingSig: Trans Form is used as the public key set to construct the ring signature ringsig.

5) The final transaction structure:

$$OTA\_Tx=(StampSet,TokenSC,Data,ringsig)$$

$$Data=(Onetime-account1,Onetime-account2,value,sig)$$

3.5.3 Transaction verification

After Validator obtains the transaction, the following verification and calculation are carried out:

Transaction Verification Process

1) Verify whether Ringsig in the transaction is matched with Trans Form. If true , the transaction is valid. Otherwise it is rejected.
2) The smart contract of TransTo is invoked with the parameter of Data. It will verify whether SC_Sig matches with SC_TransFrom or not. If it is matched, the token of SC_Value is reduced from SC_TransFrom. Meanwhile, SC_TransTo account is created with value of SC_Value.

3.5.4 Transaction confirmation

User 2 needs to confirm whether the token from user 1 is successful transferred:

Transaction Confirmation Process

1) One-time account in smart contract is scanned with scan key.
2) User 2 recognizes that $\mathrm{Onetime-account2}$ belongs to himself. The corresponding private key is calculated with the main private key and $\mathrm{Onetime-account2}$.
3) Therefore, user 2 confirms to receive the token transferred and takes the permission of $\mathrm{Onetime-account2}$.

3.5.5 Analysis on privacy effect

The privacy protection scheme of smart contract token based on One-time account and ring signature can obtain the privacy effect:

1) It can ensure that the transaction sender is anonymous in the whole network by the stamp system and ring signature scheme.

2) It can ensure that the smart contract token account is isolated from the main account by One-time address.

In order to hide the transaction sender, the stamp system and ring signature are implemented. The relationship between the stamp and user is traceable while ring signature makes the very stamp untraceable. So the transaction is isolated from the real sender and the privacy protection of the sender is realized. For the transaction receiver, one-time account system is used in the smart contract. One-time account is created for the receiver in each transaction. No one could know the owner of an one-time accout without the corresponding scan key. Thus the privacy protection of the receiver is realized.