Version 1.0.0

Wanchain

Yellow Paper

CONTENT

1 Cryptography Overview

1.1 Elliptic Curve Cryptography

1.1.1 Introduction

Elliptic curve cryptography (ECC) is a public-key cryptography method that uses elliptic curves algebraic structures over finite fields. ECC provides security using smaller keys than other cryptographic methods. ECC can be used for key agreement, digital signatures, pseudo-random generators, etc. ECC can be used for indirect encryption by using a symmetric encryption scheme with the key agreement.

Let E be an elliptic curve defined over G, which is a finite field. Actually, it is a set of points described by the Weierstrass equation:

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

The addition operation is defined on the elliptic curve. P and Q represent 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 Abelian 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 Digital Signature Algorithm

The Elliptic Curve Digital Signature Algorithm (ECDSA) is a variant of the Digital Signature Algorithm (DSA) using elliptic curve cryptography.

There is no inverse in ECDSA when calculating order, so it is easier to use than the Schnorr Digital Signature Algorithm, which is based on the intractability of discrete logarithms. ECDSA is also more efficient than the Schnorr signature because it is easier to calculate $H(M)$ than $H(M, R)$, and is therefore better used for safety strength, key length, efficiency, computational cost and bandwidth attack resistance.

Let us assume that $E_p(a, b)$ is an elliptic curve whose order is N with basepoint P. Q is an arbitrary point on $E_p(a, b)$ satisfying $Q = kP$. Then (k, Q) becomes the key pair with k as the private key and Q as the public key.

ECDSA between sender A and receiver B is shown below:

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

Step 2: Take a random integer k as the private key in $[1, N - 1]$.

Step 3: Calculate the public key $Q = kP$.

Step 4: Calculate $r = Q_x \; mod \; N$. Q_x is the abscissa of Q. If $r = 0$, return to Step 2.

Step 5: Calculate $s = k^{-1}(t + rk)\ mod\ N$. k is the private key of A. If $s = 0$, return to Step 2.

Step 6: A delivers (r, s) to B as a signature.

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

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

Step 2: Calculate the hash of plain text M: $t = H(M)$.

Step 3: Calculate $e = s^{-1}\ mod\ N$.

Step 4: Calculate $u = te\ mod\ N$.

Step 5: Calculate $v = re\ mod\ N$.

Step 6: Calculate $R = uP\ +\ vQ$.

Step 7: If $R = 0$, B rejects the signature. Otherwise calculate $r' = R_x\ mod\ N$. R_x is the abscissa of R.

Step 8: If $r' = r$, the signature is valid. Otherwise it is invalid.

1.2 Threshold Key Sharing

1.2.1 Shamir' s Secret Sharing Scheme

Shamir' s Secret Sharing scheme solves the problem of secure key management. The design of modern cryptography mechanisms commits the security of cryptosystems to the safety of secret keys. Divulging the secret key compromises security, and thus key-management takes an important position in security research and crypto-system design. Especially when one account is managed by multiple parties with different interests, distributing the key to multiple participants becomes troublesome where credibility and safety are concerned. To combat this problem, Shamir, an Israeli cryptographer, proposed a new (k, n) Threshold Secret Sharing scheme. In this scheme, the key is divided into n parts and distributed to n participators. Each participator holds one key share and the key is reconstructed only when k key shares are gathered.

1.2.2 Linear Key Sharing Scheme

Linear Key Sharing is an extension of Shamir' s Secret Sharing scheme, where the requests for the main key space, sub-key space and randomly inputted assembly are all linear, and the key construction function is also linear. Its formal definition is as follows:

Suppose that K is a finite field, Π 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}. 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 in any grant set $G \in AS$, $\exists$ is constant as $\{a_{i,j}\colon P_i \in G, 1 \leq j \leq d_i\}$ and makes any main key s, and random number r, with the formula

below:

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

1.2.3 Shamir' s Scheme and Polynomial Interpolation

Shamir designed the threshold secret sharing scheme based on Lagrange's polynomial interpolation by combining properties of a polynomial over a finite field and Lagrange' s reconstruction polynomial theory. Concrete schemes are 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 the key sharing process we know $s_i = (x_i, f(x_i))$ is a point on the curve $f(x)$. According to Lagrange's polynomial interpolation method, any k points of $f(x)$ will reconstruct $f(x)$:

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

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

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

simplified :

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

Where

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

1.3 Secure Multi-Party Computation

1.3.1 Background

With the rapid development of the Internet, more and more application scenarios need computed results to be coordinated between users. Users participating in coordinating computing want to share the computed data with other users, however, due to privacy protection and data safety concerns, this sharing proves difficult. The inability to share the results leads to inefficient resource utilization and prevents realization of some these application scenarios.

Secure multi-party computation solved the problem and also provides the theoretical basis for solving 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 so that users can accomplish a task by coordinative computing without leaking any private information in a mutually distrustful multi-user network. Simply speaking, secure multi-party computation is about a group of participants, referred to as $P_1 \ldots\ldots P_n$, who work together to securely compute the function $f(x_1, \ldots\ldots, x_n) = (y_1, \ldots\ldots, y_n)$. The n inputs of function f are held by n participants, one each. Assuming P_i holds the secret input x_i, after the computation P_i gets output y_i. In this situation, security requires that even if some participants cheat during the computing process, the correctness of the computing result is still ensured. That means every participant gets the correct output y_i and all participants' input remain privileged. P_i gets no extra information besides (x_i, y_i) through the computation.

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

1.3.2 Classification

At present the secure multi-party computation protocol can be divided into four categories based on different implementation methods:

- VSS Sub-Protocol

Most existing secure multi-party computation protocols adopt the 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 field addition and multiplication calculations. 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 calculation.

- Mix and Match

The secure multi-party computation protocol based on VSS can calculate any function, but it cannot efficiently calculate Boolean functions. Therefore, another secure multi-party protocol is proposed - Mix and 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 protocol process, the participant randomly encrypts the public key used by himself, then opens his own encrypted result, and finally all participants get the common output through Mix and Match.

- Oblivious Transfer

The secure multi-party computation protocol based on oblivious transfer (OT) is used to calculate any bit-wise operation function. It uses the OT sub-protocol to perform the three basic operations, “and”, “or”, and “not”, deconstructs any bit-wise operation function into a combination of the three basic operations, and finally iteratively calculates any bit-wise operation functions.

- Homomorphic Encryption

The secure multi-party computation protocol based on Homomorphic Encryption can resist active attacks. The idea is to select an algorithm such that any function calculation is deconstructed into a sequence of atomic computations. The input and output of the atomic computations are both encrypted using the homomorphic encryption algorithm, the final results get encrypted, and only specific participants can get the plaintext results.

1.4 Ring Signatures

A Ring Signature is a digital signature used by a group of users, each of which have keys to perform the signature. The Ring Signature was first invented in 2001 by Ron Rivest as a special kind of group signature. Group signatures need a trusted and secure setup which means that the signer can be traced by the trust center. Ring Signature solves this serious problem by removing the trust center and secure setup, the signer is absolutely anonymous.

Since the Ring Signature was invented, many practical schemes have been designed based on Elliptic Curve Cryptography (ECC), such as the Trapdoor Technique. Generally speaking, there are four kinds of Ring Signatures:

- Threshold Ring Signature
- Linkable Ring Signature
- Ring Signature with Revocable Anonymity
- Deniable Ring Signature

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

2 Locked Account System

2.1 Secure Multi-Party Computation Operation

Addition, multiplication, and unary inverse are the three basic operations over a finite field. Any calculation can be converted into a sequence of operations of addition, multiplication, and unary inverse operations over this finite field. So 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 basic multi-party computation. In the following sections, the basic operations of the secure multi-party computation algorithm are introduced under a secret sharing scheme based on Lagrange's polynomial interpolation.

2.1.1 Addition

It is necessary to determine a polynomial under the secret sharing scheme based on Lagrange's polynomial interpolation. The shared secret is the constant term of the polynomial, and a secret share is the value of this polynomial at some point.

Without loss of generality, assume that α, β are two shared secrets. The corresponding polynomials are $f_{\alpha}(x)$、$f_{\beta}(x)$ and 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 $\alpha + \beta$, so he needs to construct a polynomial $f(x)$ such that the constant term is $\alpha + \beta$, and P_i is calculated to get $f(i)$. The construction process is as follows :

$\because$ α_i、β_i are the secret share of α、β , and the corresponding polynomials are $f_{\alpha}(x)$、$f_{\beta}(x)$

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

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 $\alpha + \beta$. The value of the polynomial is $f(i)$ at point i.

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

The above construction process gets a secure multi-party computation protocol for the addition operation.

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

$x\gamma_i = \alpha_i + \beta_i$

2.1.2 Multiplication

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

If $(f_\alpha(i), f_\beta(i))$ is 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. Taking $(\lambda_1,\cdots,\lambda_{2k-1})$ 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$h_1(x),\cdots,h_{2k-1}(x)$, and is required to meet $h_i(0) = f_{\alpha\beta}(i)$.

If you define$H(x) = \sum_{i=1}^{2k-1} \lambda_i h_i(x)$, then 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 is α_i, β's secret share is β_i

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

Defining α as the shared secret, the corresponding polynomial is $f_\alpha(x)$ and participant P_i has the secret share $\alpha_i = f_\alpha(i)$. Unary inverse computation refers to participant P_i getting the secret share $f_{\alpha^{-1}}(i)$ of α^{-1} through the computing of secret share α_i. In the process of computing (α, α^{-1}) their secret share cannot be leaked. The algorithm is as follows:

Participant P_i selects random number r_i and polynomial $g_i(x)$ to compute secret share $r_{ij} = g_i(j)$ and sends the result to participant P_j. After receiving all the 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 the Multi-Party Computation (MPC) protocol for the multiplication operation, the secret share for $f_{\alpha r}(i)$ of αr is obtained by computing the secret shares of α and r, and sends the result to the other participants. It can also use Lagrange' s polynomial interpolation to reconstruct αr. Defining $m = \alpha r$, obviously lets $f_{\alpha^{-1}}(i) = m^{-1} r_i'$, which is the secret share of α^{-1}.

MPC Protocol For Unary Inverse Operation

Input : α's secret share is α_i

Output : (α^{-1})'s secret share is 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 the secret shares and computes: $r_j' = \sum_{l=1}^{n} r_{l,j}$

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

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

2.2 Locked Account Generation Scheme

The locked account generation scheme, is an improvement on the Lagrange interpolation polynomial-based threshold key sharing scheme. The basic idea is to generate a decentralized locked account with all verification nodes through threshold key sharing, where each verification node is mastering a key share of the locked account private key. This ensures that the private key of the locked account is distributed to the whole network, so it can be managed in a decentralized way. The specific algorithm is as follows:

Locked Account Generating Algorithm

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

2) P_i selects a random $k-1$ degree polynomial $f_i(x) = d_i + a_{i,1}x + \cdots + a_{i,k-1}x^{k-1}$, sends $f_i(j)$ to P_j through a secure channel, and broadcasts $a_{i,1}G$……$a_{i,k-1}G$ 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 it, else P_j rejects it and asks P_i to resend the message.

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

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

2.3 Locked Account Signature Scheme

In the current blockchain project, the Locked Account Signature scheme uses the Elliptic Curve Digital Signature Algorithm (ECDSA) for the mainstream signature algorithm, which improves system compatibility. In the locked account signature generation process, unlike in the original ECDSA signature algorithm, the account private key and the random number participate in the ECDSA signature process in the form of a multi-party computation (MPC). The verification process of the locked account signature is the same as the original ECDSA signature verification algorithm, so only the locked account signature generation process is introduced here:

Locked Account Signature Verifying Algorithm

1) All Participants share the same random number c through MPC. The 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 $(x, y) = \sum_{j=1}^{k} b_j R_j$, $r = x \bmod p$ and $b_j = \sum \frac{j}{j-i}$.

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

5) Using the MPC protocol for multiplication operation, P_i computes the $(c^{-1}d)'$ share v_i via ω_i and t_i where, d is the private key of the Locked Account and t_i is the 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 $R_j = u_{j1}G + u_{j2}Q_j$, $u_{j1} = ms_j^{-1}$, $u_{j1} = rs_j^{-1}$and $Q_j = t_jG$. 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, it reconstructs the complete signature (r, s) using Lagrange interpolating polynomials.

2.4 Locked Account Key Share Updating

The threshold key sharing scheme, based on Lagrange interpolating polynomials, adopted by this protocol belongs to the linear key sharing scheme, so key sharing satisfies the homogeneity: If the (k, n) for threshold key share key_1 is $(a_1, \cdots, a_n)$and the (k, n) for threshold key share key_2 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 we 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 a 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 $(f_1(i), \cdots, f_n(i))$. The new key share for then P_i 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 the user' s public key P, constitutes the public key set $S = \{P_i | i = 1,2 \ldots, \}$. For the user' s public and private keys, the order of point P is (P, x), $x \in [1, l-1]$ and l. This generates public key image I.

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

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

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

The specific process is described as follows:

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

SIG: The signer selects a random number $\{i=1,2,\ldots,n\}$ and $\{\omega_i|i=1,2,\ldots,n,i\neq s\}$ and computes as follows:

$$L_i=\begin{cases} q_iG, & if\ \ i=s \\ q_iG+\omega_iP_i, & if\ \ i\neq s \end{cases}$$

$$R_i=\begin{cases} q_iH_p(P_i), & if\ \ i=s \\ q_iH_p(P_i)+\omega_iI, & if\ \ i\neq s \end{cases}$$

Computing :

$c=H_s(m,L_1,\ldots,L_n,R_1,\ldots,R_n)$, H_s is hash function.

Computing :

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

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

Finally generating a signature:

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

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

$$\begin{cases} L_i{}'=r_iG+c_iP_i, \\ R_i{}'=r_iH_p(P_i)+c_iI \end{cases}$$

Verify whether the equation $\sum_{i=1}^{n} c_i = H_s(m, L'_1, \dots, L'_n, R'_1, \dots, R'_n) \bmod l$ holds. If the equation holds, then the signature is valid and implement LNK.

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

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

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

$i \neq s$, 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 I consists of all I that appears 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 invalid. If I does not appear in the set, then the signature of transaction is valid, and I is added into the set I.

3.2 One-Time Account System

3.2.1 Composition

The one-time account system is the basis of the entire privacy transaction. Every user in the system has a single main account and multiple sub-accounts. A sub-account can also be considered as the account in the smart contract. Usually a sub-account is generated by a transaction sender for the receiver, as is shown in the following chart:

Alice has a unique main account (A, B), and several sub-accounts (A_1, S_1), (A_2, S_2), (A_3, S_3), etc. When Bob makes a transaction to Alice, (A_1, S_1) and (A_2, S_2)

are created through the main account (A, B). When Carol makes a transaction to Alice, (A_3, S_3) is created. Every transaction to Alice creates a sub-account for Alice, which is one-time account. Only Alice has the ability to manage and use these sub-accounts.

3.2.2 Account Generation Algorithm

$E(F_p)[r]$ is the subgroup of order r of the elliptic curve E/F_p. The base point of $E(F_p)[r]$ is G. Choose the elliptic curve of either Bitcoin or Ethereum $y^2 = x^3 + 7$.

1. Main Account Generation

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

2. Sub-Account Generation

When Bob initiates transaction to Alice, Alice's main account (A, B) is used to generate sub-accounts (A_1, S_1).

Bob generates a random number $s \in [1, r-1]$ and computes

$S_1 = [s]G$

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

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

$$(A_1, S_1)$$

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

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

3. Sub-Account Verification

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

$$A_1' = A + [Hash([b]S_1)]G$$

If $A_1' = A_1$, then (A_1, S_1) is Alice's sub-account.

If $A_1' \neq A_1$, then (A_1, S_1) is not Alice's sub-account.

This is because $B = bG$, thus

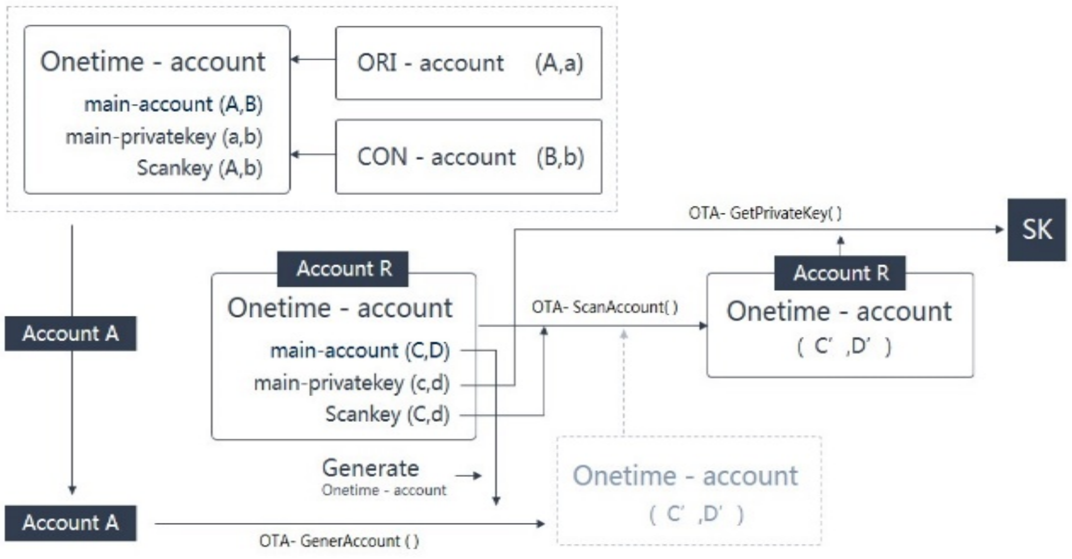
$$[s]B = [sb]G = [b][s]G = [b]S_1$$

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

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

$$x = a + Hash([b]S_1)$$

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



3.3 Stamp System

The above algorithms show that the sender of a transaction is untraceable when privacy is protected by the ring signature. Thus, it results in the problem of figuring out which account pays the transaction fee. Wanchain solves this problem by implementing the Stamp System, which is based on a one-time account system. Each stamp is an one-time account in the stamp system, referred as $Onetime - stamp$. Technically $Onetime - stamp$ is the same as $Onetime - account$. Users purchase stamps beforehand when conducting private transactions. Stamps are pasted on the transactions to complete the private transaction and each stamp can only be used once.

The 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 purchase and return functions:

Stamp Purchase Function

The stamp purchasing function implements the stamp purchasing service using Wancoin. If the user $Account$ needs to purchase a stamp, he sends a transaction to transfer Wancoin with the value of the pre-purchased stamp to the stamp system smart contract and invokes the contract purchasing function. The parameter of the purchasing function is the one-time account generated by the user, which will be stored in the stamp list with a corresponding value to prove that the user has purchased the stamp.

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

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

Stamp Refund Function

The stamp refund function allows the user to return unused stamp. If the user $Account$ purchased a one-time stamp and the stamp is not used, then the user invokes the smart contract stamp refund function to get the corresponding Wancoin value back into his account. The parameters of the return function are $Onetime - stamp$, the stamp face value and the ring signature. When the refund transaction is confirmed, the flag I is shown in the ring signature and stored to prove that the stamp has been refunded:

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

$$payload = ("refund", StampSet, value, ringsig)$$

$$ringsig = (I, c_1, c_2, \dots, c_n, r_1, r_2, \dots, r_n)$$

Using the ring signature in the stamp refund function requires the user to provide the corresponding I value of the stamp to ensure that the stamp is not used and a situation where the stamp is refunded after being used does not occur.

After the Stamp System is implemented, when the user makes a private transaction, he first purchases a stamp in the stamp system. The face value of the stamp is decided by the user and the calculation quantity of the smart contract. As the account information of the user does not occur in Tx of the token transaction, the set of user stamps and a randomly selected stamp with same face value serves as the transaction sender. The ring signature of the user' s stamp ensures the transaction is valid.

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

The transaction fee deducted from the account is the face value of the used stamp. Each stamp can only be used once. When a 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, increasing the $Coinbase$ stamp value. 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 the hash result of string $WanchainStampSystem$, so that nobody has the private key of the contract and the received Wancoin cannot be transferred. This means that the money in the stamp system contract is locked and the amount of Wancoin is constant against the increase of value in $Coinbase$.

3.4 Native Coin Transaction Privacy Protection

The Native Coin transaction privacy protection scheme is similar to the Stamp System method and is implemented by deploying a corresponding Wancoin smart contract on Wanchain.

3.4.1 Wancoin Smart Contract

A Wancoin smart contract is deployed on Wanchain like any other smart contract. It is similar to the Stamp System where the dispersed value of the token money corresponds in a 1:1 ratio with Wancoin. Wancoin smart contracts provide two functions: token purchase and token return.

The token purchase function allows the user to transfer Wancoin to the contract and receive an equal value of tokens into their one-time account. The one-time account provided by the user is added to the storage list of the corresponding face value in the contract.

The token return function is invoked by the user 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. The privacy protection process of Wancoin transactions is explained in detail below.

3.4.2 Transaction Scenario

The transaction scenario is as follows:

The private key of User1' s main account is set (a, b) and the corresponding public key is set (A, B).

The private key of User2' s main account is set (c, d) and the corresponding public key is set (C, D).

User1 transfers Wancoin of amount $value$ to User2.

3.4.3 Transaction Flow

Initiate Transaction

User1 builds a one-time account $Onetime - account2$ by using the main account public key (C, D) of User2. Invoke the token purchase function of the contract $WancoinSC$ using the one-time account as its parameter:

$$Tx = (Account1, WancoinSC, value, payload, sig)$$

$$payload = ("purchase", Onetime - account2)$$

Confirm Transaction Initiation

The Validator receives the transaction and verifies 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.

Receive Transaction

User2 scans the storage list of $WancoinSC$ using his scan key. He finds that $Onetime - account2$ belongs to himself and selects the $n-1$ one-time account in the corresponding face value list to generate a ring signature of $Onetime - account2$. This ring signature is used as the parameter for the token return function:

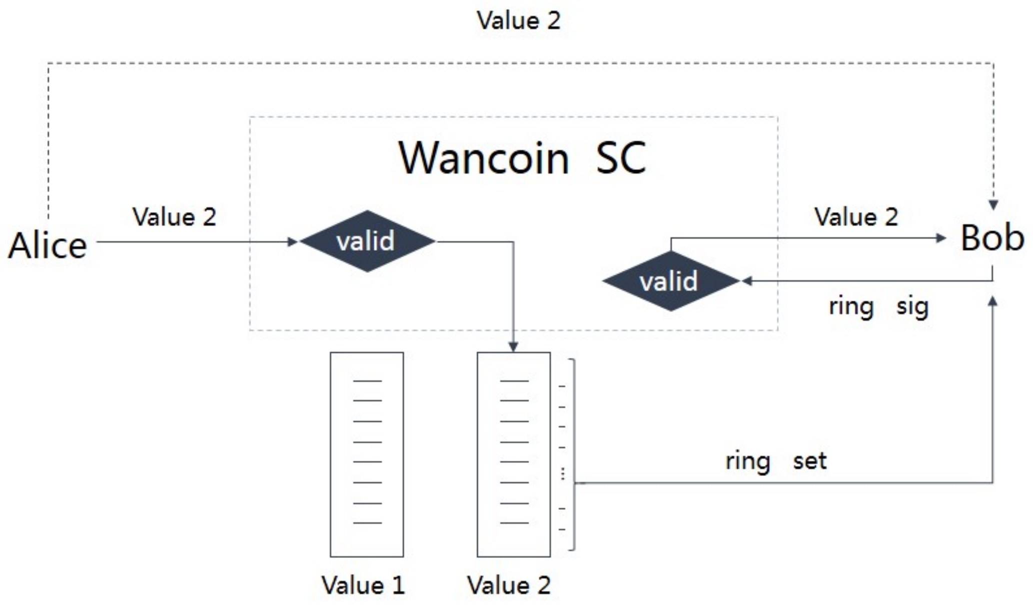
$$Tx = (Account2, WancoinSC, payload, sig)$$

$$payload = ("refund", OTASet, ringsig)$$

Confirm Transaction Received

The Validator receives the transaction and verifies the relationship between sig and $Account2$. If sig is valid, the Wancoin smart contract is invoked. The contract verifies the validity of the ring signature and makes sure that value I

in $ringsig$ does not occur. After the confirmation, Wancoin with corresponding value is transferred to $Account2$.



3.4.4 Privacy Effect Analysis

The Wancoin transaction privacy protection scheme based on one-time accounts and ring signatures shows the following privacy effects:

1) By using one-time accounts, no one knows who receives the token during the transaction process.

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

In this scheme, a relationship is set up between the transaction initiator and the one-time account purchasing the token. But a relationship cannot be set up between the transaction receiver and the one-time account. Thus, the initiator cannot correspond with the receiver and privacy protection is realized. To make a convenient use of the ring signature, storage with several fixed dispersed face values is set in the Wancoin smart contract. Private transactions with arbitrary face values cannot be performed, which will be researched in a follow-up study.

3.5 Smart Contract Token Privacy Protection Scheme

3.5.1 Transaction Scenario

The smart contract token privacy protection scheme scenario is as follows:

The private key of User1's main account of is set (a, b). The public key of the main account is set (A, B).

The private key of User2' s main account is set (c, d). The public key of the main account is set (C, D).

The account of User1 in the smart contract is $Onetime - account1$.

User1 wants to transfer tokens worth amount $value$ to User2.

3.5.2 Transaction Flow

Initiate Transaction

User1 purchases a stamp that records $Onetime - stamp$ 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.

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: first, $Onetime-account2$ is constructed for User2. SC_TransFrom is $Onetime-account1$ of User1. The transaction target addresses SC_TransTo is $Onetime-account2$ of User2. SC_Value is $value$. User1 calculates sig of the transaction that corresponds to $Onetime-account1$.

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

5) The final transaction structure:

$$OTA\_Tx = (StampSet, TokenSC, Data, ringsig)$$
$$Data = (Onetime-account1, Onetime-account2, value, sig)$$

Verify Transaction

After the Validator obtains the transaction, carry out the following verification and calculations:

Transaction Verification Process

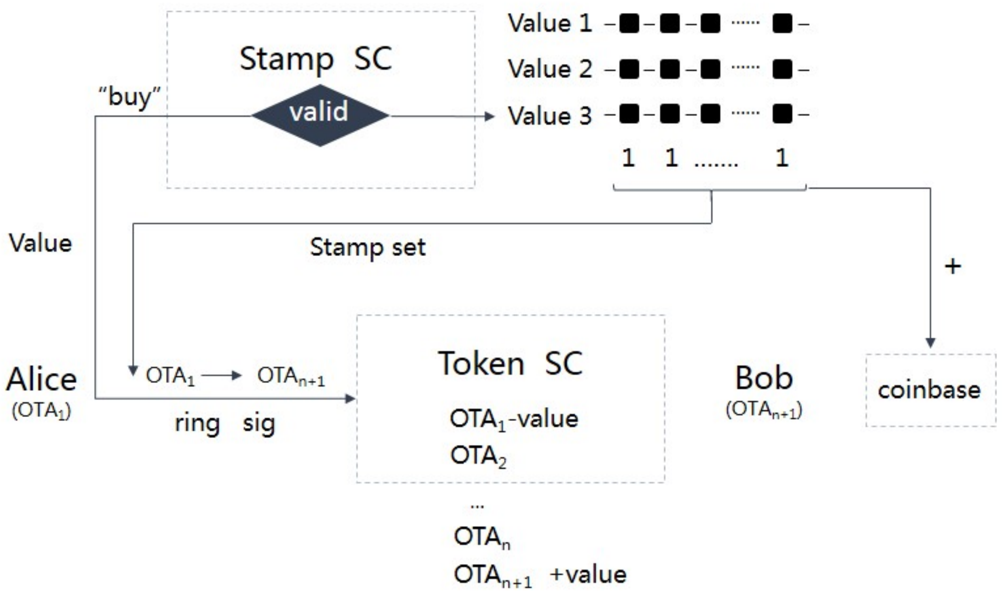
1) Verify whether Ringsig in the transaction is matched with TransForm. If true , the transaction is valid. Otherwise it is rejected.
2) The smart contract of TransTo is invoked with the parameter of Data. It will verify whether SC_Sig matches with SC_TransFrom or not. If it is matched, the token of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

Confirm Transaction

User2 confirms whether the token from User1 is successfully transferred:

Transaction Confirmation Process

1) One-time account in smart contract is scanned with scan key.
2) User2 recognizes that $Onetime - account2$ belongs to himself. The corresponding private key is calculated with the main private key and $Onetime - account2$.
3) User2 then confirms the receipt of the transferred token and takes the permission of $Onetime - account2$.



3.5.3 Privacy Effect Analysis

The smart contract token privacy protection scheme based on one-time accounts and ring signatures show the following privacy effects:

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

2) It ensures that the smart contract token account is isolated from the main account by a one-time address.

In order to hide the transaction sender we implement the stamp system and a ring signature. The relationship between the stamp and user is traceable while the ring signature makes that very stamp untraceable. So the transaction is isolated from the real sender and privacy protection for the sender is realized. For the transaction receiver, we use the one-time account system in the smart contract. A one-time account is created for the receiver in each transaction. No one can know the owner of a one-time account without the corresponding scan key. Thus privacy protection for the receiver is realized.

-- Contributors --

Jack Lu、Demmon、Shi、Zane Liang 、Ying Zhang、Boris Yang、Eric Swartz、Lizzie Lu

Reviewers : Dustin Byington、David A. Johnston、Michael Y.、Jian Shen 、Feng Han、 Albert Ching