Cyber Security

系統與程式安全機制

睿決科技股份有限公司

多方計算架構簽核的安全性錢包

我們錢包系統基於 Go 語言 開發，採用 MPC（多方計算）架構確保私鑰的安全性。

傳統區塊鏈錢包大多將私鑰集中存儲於單一設備或伺服器上，這樣的設計會增加被攻擊者盯上的風險，特別是當涉及大量加密資產時，單一私鑰成為攻擊的首要目標。為了降低風險，系統將私鑰分割為多個碎片，並分佈存儲在不同的節點上。這樣，即使某個節點被攻擊或遭到入侵，也無法完整地還原私鑰或轉移資產。

私鑰分片

系統的私鑰管理基於分佈式架構。通過 MPC 協議，將用戶的私鑰分割成多個碎片，並分佈至不同的節點或設備中。確保私鑰不會被集中存儲在單一位置，從而降低被攻擊的風險。

門限簽名

門限簽名技術能讓用戶在多個節點之間設定簽名門檻（如 3-of-5），確保即便部分節點無法參與，也能正常進行交易，這使得系統具有很高的容錯能力。

雙因素認證

系統具備雙因素認證（2FA）功能，如出現敏感操作時需輸入 OTP 驗證碼或使用生物識別技術來完成身份驗證。因關鍵操作要求多層身份驗證。即使密碼被洩漏，資產仍受保護。

私鑰分割理論概述

上述提到的MPC架構，以私鑰分割如何設定為例。

分割密鑰

split_secret 函數利用 Shamir’ s Secret Sharing 將密鑰（或私鑰）分割成多個分片。
它生成一個隨機多項式，並將該多項式在 x 軸的不同點（即不同參數）進行計算以生成密鑰分片。

```cpp
#include <iostream>
#include <vector>
#include <openssl/bn.h>
#include <random>

class ShamirSecretSharing {
public:
    ShamirSecretSharing(int threshold, int total_shares)
        : threshold(threshold), total_shares(total_shares) {}

    std::vector<std::pair<int, BIGNUM*>> split_secret(BIGNUM* secret) {
        // Generate random polynomial coefficients
        std::vector<BIGNUM*> coefficients;
        BN_CTX* ctx = BN_CTX_new();
        BIGNUM* coefficient = BN_new();

        // First coefficient is the secret
        coefficients.push_back(BN_dup(secret));
        for (int i = 1; i < threshold; ++i) {
            BN_rand(coefficient, 256, BN_RAND_TOP_ANY, BN_RAND_BOTTOM_ANY);
            coefficients.push_back(BN_dup(coefficient));
        }
```

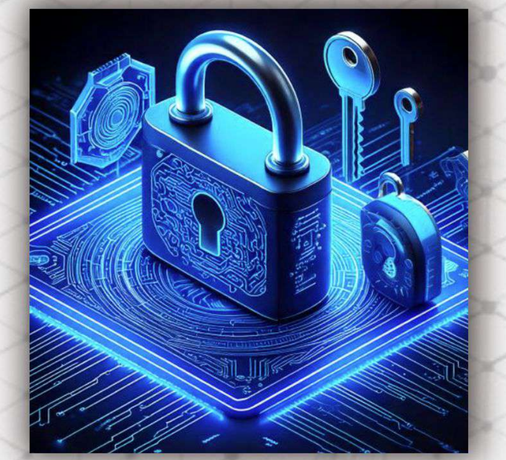
```cpp
        // Generate shares
        std::vector<std::pair<int, BIGNUM*>> shares;
        for (int x = 1; x <= total_shares; ++x) {
            BIGNUM* share = BN_new();
            BN_zero(share);

            // Evaluate polynomial at x
            for (int i = 0; i < threshold; ++i) {
                BIGNUM* term = BN_new();
                BN_copy(term, coefficients[i]);
                BN_mul(term, term, BN_value_one(), ctx); // x^i * coefficient
                BN_mod_add(share, share, term, BN_get0_nist_prime_192(), ctx);
                BN_free(term);
            }

            shares.push_back({x, share});
        }

        // Clean up
        for (auto coef : coefficients) {
            BN_free(coef);
        }
        BN_free(coefficient);
        BN_CTX_free(ctx);

        return shares;
    }
```



重建密鑰理論概述

上述提到的MPC架構，再以重建密鑰如何設定為例。

重建密鑰

reconstruct_secret 函數使用拉格朗日插值法通過任意門限數量的分片重建密鑰，來實際用於簽名操作。

```
private:
    int threshold;
    int total_shares;
};

int main() {
    int threshold = 3;
    int total_shares = 5;

    ShamirSecretSharing sss(threshold, total_shares);

    // Define a simple integer as the "secret" (e.g., private key)
    BIGNUM* secret = BN_new();
    BN_set_word(secret, 12345);

    // Split the secret into shares
    auto shares = sss.split_secret(secret);
    std::cout << "Shares generated:\n";
    for (const auto& share : shares) {
        std::cout << "Share " << share.first << ": " << BN_bn2hex(share.second) << "\n";
    }

    // Select a subset of shares to reconstruct the secret
    std::vector<std::pair<int, BIGNUM*>> subset_shares = {shares[0], shares[1], shares[2]};
    BIGNUM* reconstructed_secret = sss.reconstruct_secret(subset_shares);

    std::cout << "\nReconstructed Secret: " << BN_bn2hex(reconstructed_secret) << "\n";

    // Clean up
    BN_free(secret);
    for (auto& share : shares) {
        BN_free(share.second);
    }
    BN_free(reconstructed_secret);

    return 0;
}
```

```
BIGNUM* reconstruct_secret(const std::vector<std::pair<int, BIGNUM*>>& shares) {
    BN_CTX* ctx = BN_CTX_new();
    BIGNUM* secret = BN_new();
    BN_zero(secret);

    // Lagrange interpolation to reconstruct the secret
    for (size_t i = 0; i < shares.size(); ++i) {
        BIGNUM* numerator = BN_new();
        BIGNUM* denominator = BN_new();
        BN_one(numerator);
        BN_one(denominator);

        for (size_t j = 0; j < shares.size(); ++j) {
            if (i != j) {
                BIGNUM* temp = BN_new();
                BN_set_word(temp, -shares[j].first);
                BN_mul(numerator, numerator, temp, ctx);

                BIGNUM* diff = BN_new();
                BN_sub_word(diff, shares[i].first - shares[j].first);
                BN_mul(denominator, denominator, diff, ctx);

                BN_free(temp);
                BN_free(diff);
            }
        }

        BIGNUM* term = BN_new();
        BN_mod_inverse(denominator, denominator, BN_get0_nist_prime_192(), ctx);
        BN_mul(term, shares[i].second, numerator, ctx);
        BN_mod_mul(term, term, denominator, BN_get0_nist_prime_192(), ctx);
        BN_mod_add(secret, secret, term, BN_get0_nist_prime_192(), ctx);

        BN_free(numerator);
        BN_free(denominator);
        BN_free(term);
    }

    BN_CTX_free(ctx);
    return secret;
}
```

更全面的保護機制

除了核心的MPC架構技術來實現高度安全的私鑰保護外，
也有更全面的安全措施，進而提高資產與數據的保障。

通訊加密

系統內部的所有數據傳輸均通過 TLS/SSL 加密進行，能夠有效防止中間人攻擊和數據篡改，保護私鑰碎片和其他敏感數據在節點間傳輸，確保數據在傳輸過程中的機密性、完整性和不可否認性。

代碼審計

由第三方專業公司來對錢包的核心模組進行深入檢查，特別是私鑰管理、交易簽名、身份驗證等高風險的功能模塊。

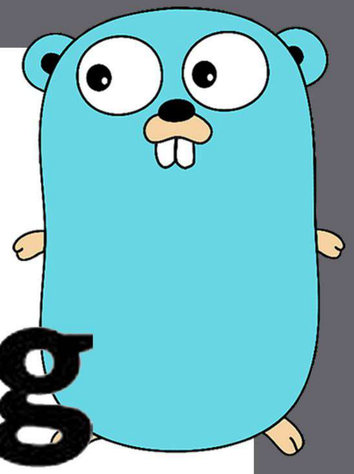
可提前發現並修復潛在的漏洞，確保系統運行安全和穩定。也保障系統符合各類區塊鏈和加密貨幣的監管要求，提升用戶對系統的可信度。

錯誤處理與回滾機制

在 MPC 協議中，當簽名過程出現故障或某個節點無法參與時，系統能夠安全的取消操作，並將系統狀態恢復到簽名操作之前，避免未簽名的交易被意外廣播至區塊鏈網路。

在多方計算環境下有效的處理錯誤與回滾機制，並記錄相關日誌，讓開發者追蹤問題根源與故障排除。

Go語言開發模型



系統的後端開發基於 Go 語言。Go 語言以其強大的併發處理能力和高效的內存管理而著稱，特別適合用於區塊鏈應用開發。

Golang

01 處理大量用戶請求

對於一個需要處理大量用戶請求、交易簽名和節點交互的區塊鏈錢包來說，Go 的優勢在於它能夠在不犧牲性能的情況下，同時處理大量的併發操作。

02 強大的網路和加密功能

Go 語言的標準庫中內置了強大的網路和加密功能，讓開發人員可以輕鬆構建安全且高效的系統架構，在高流量場景下系統能夠保持穩定運行，不會因大量併發請求而崩潰或延遲。

03 與多種區塊鏈的互動

為了實現與多種區塊鏈的互動，Go 語言的開發環境中可以使用如 go-ethereum 這樣的庫來實現與以太坊的交互，包括交易創建、簽名和智能合約調用。

04 靈活的API層

Go 的網絡框架（如 Gin 和 gRPC）提供高效且靈活的 API 層，支持與前端應用程序進行數據交換，並確保用戶能快速響應各種操作請求。這些 API 實現了從基礎的資產查詢、轉賬等功能，到更複雜的智能合約操作。

交易程式異常處理

交易系統在面對市場出現極端波動時，交易機會可能會消失甚至反轉，或是程式出現異常時須要有防呆機制，來避免資金上因極端市場或程式出錯造成損失。

市場風險機制

系統應該具備自動應對 市場風險 的機制，當檢測到市場異常波動時（例如市場價格一定時間內超過一定範圍），系統應立即停止交易，由管理員監測後回報，待緩後再做出下個交易決策以確保交易安全。

監控交易所API

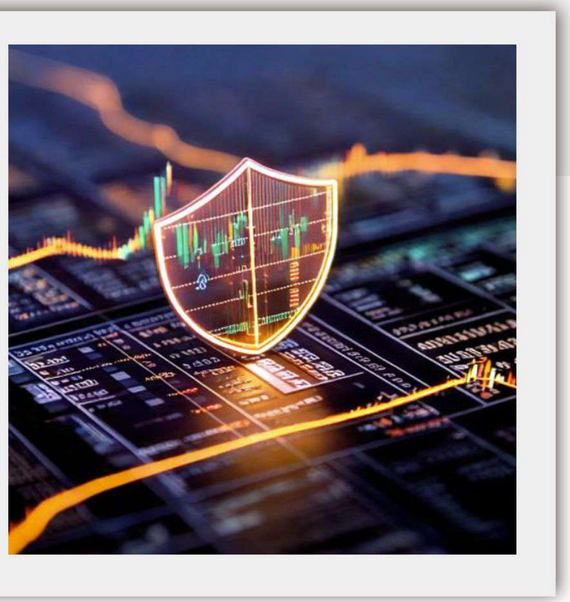
時時監控交易所的狀態，確保每個交易所的 API 正常工作。如果某個交易所的 API 出現故障或延遲，系統應能夠自動切換到備份策略，或暫停相關交易，並通報管理員。

異常處理機制

為了防止資金被鎖定或延遲，系統自動執行異常處理機制，這包括當某筆交易未能成功執行時的回滾策略，以及資金轉移過程中的錯誤修正機制。所有異常情況應及時觸發警報系統，並在管理員端生成詳細的報告，以便快速響應。

交易程式保護機制

交易系統因涉及大量的資金交易，因此必須保證系統本身的安全性。除此之外，系統的核心數據庫必須進行加密存儲，尤其是關於用戶資產和私密數據的部分。



TLS/SSL 加密

系統與交易所 API 之間的所有通信都必須通過來防止中間人攻擊或數據篡改，以此確保交易指令和資金數據在傳輸過程中的機密性和完整性。

雙因素認證（2FA）

應對所有用戶和管理員操作設置多層身份驗證機制，特別是對於高風險操作（如提款、轉帳等），使預雙因素認證（2FA），要求使用動態驗證碼或生物識別技術來進行額外的身份驗證。

角色基礎存取控制（RBAC）

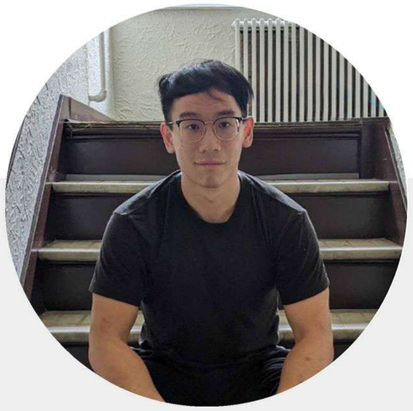
為了防止未授權的訪問，系統應使用 角色基礎存取控制（RBAC）機制，確保只有授權的用戶能夠訪問或操作關鍵數據。此外，應對所有系統操作進行詳細日誌記錄，以便在發生問題時能夠快速追蹤和恢復。

Core Members

Vance Lin / 林廷翰

Founder & CEO

台灣科技大學資訊管理所 碩士
中國 浙江大學 區塊鏈實驗室訪問研究
唯數娛樂 區塊鏈專案負責人
5年區塊鏈產業經驗



Kobe Chao

Co-Founder & CTO

政治大學 資訊科學
新加坡商 Xfers 區塊鏈開發主管
多年區塊鏈開發經驗
參與兩國穩定幣發行專案

Kilin Chen

CMO

十年電商經驗
Groupon、Klook、
foodpanda 資深業務經理