TRZYDZIESTE FORUM TELEINFORMATYKI®

Implementacja hierarchicznych podpisów pierścieniowych w systemie blockchain z funkcjonalnością anamorficzną

Opiekun pracy: **dr hab. inż. Łukasz Krzywiecki, prof. uczelni**
Autorzy: **Witold Karaś, Adam Niezgoda, Mateusz Chęciński**

Idea - popis pierścieniowy

Podpis pierścieniowy to cyfrowy podpis używany do anonimowego podpisywania wiadomości.

Właściwości podpisów pierścieniowych:

- Sygnatariusz używa swojego klucza prywatnego oraz zbioru kluczy publicznych wybranej grupy osób (nazywanego **pierścieniem**).
- Tożsamość sygnatariusza jest ukryta w pierścieniu kluczy publicznych. - Nie ma możliwości wskazania kto podpisał wiadomość.
- Nikt nie może zapobiec włączeniu się do pierścienia.

Problem - popis pierścieniowy

Podpis musi być weryfikowalny dla wszystkich kluczy użytych w **pierścieniu**.

Oznacza to, że:

- Rozmiar podpisu rośnie proporcjonalnie z ilością użytych kluczy.
- Większa anonimowość oznacza używanie dłuższych kluczy.

Rozwiązanie problemu - podpisy hierarchiczne

Rozwiązaniem problemu zwiększania rozmiaru pierścienia kluczy jest zastosowanie **hierarchicznych podpisów pierścieniowych.**

Ich głównymi cechami są:

- Ponowne wykorzystanie informacji o wcześniej użytych pierścieniach kluczy w celu skrócenia podpisów.
- Tworzenie struktury hierarchicznej – podpisy na danym poziomie wykorzystują zbiory anonimowości z niższych poziomów.

Idea - struktura hierarchiczna

$$\sigma_A = (y^A_{new}, \hat{R}^A, s^A)$$
$$A = \{y^B_{new}, y^C_{new}\}$$

$$\sigma_B = (y^B_{new}, \hat{R}^B, s^B)$$
$$B = \{y^D_{new}, y^E_{new}\}$$

$$\sigma_C = (y^C_{new}, \hat{R}^C, s^C)$$
$$C = \{y^F_{new}, y^G_{new}\}$$

$$\sigma_D = (y^D_{new}, \hat{R}^D, s^D)$$
$$D = \{y^D_1, ..., y^D_k\}$$

$$\sigma_E = (y^E_{new}, \hat{R}^E, s^E)$$
$$E = \{y^E_1, ..., y^E_k\}$$

$$\sigma_F = (y^F_{new}, \hat{R}^F, s^F)$$
$$F = \{y^F_1, ..., y^F_k\}$$

$$\sigma_G = (y^G_{new}, \hat{R}^G, s^G)$$
$$G = \{y^G_1, ..., y^G_k\}$$

Problem 2 - weryfikacja podpisów hierarchicznych

W oryginalnej pracy weryfikacja podpisu wymaga rekurencyjnej weryfikacji całego drzewa. Dlatego użycie struktury rejestru, opartej o **blockchain**, pozwala na zapamiętywanie i rozproszoną weryfikację, co gwarantuje ich niepodrabialność oraz niezmienność.

W naszym projekcie zaimplementowaliśmy:

- Klienta CLI służącego do tworzenia podpisów oraz interakcji z rejestrerem w języku **Python**.
- Inteligentne kontrakty działające na łańcuchu **Ethereum** - służące do weryfikacji oraz zapisu podpisów w strukturze, przy użyciu języka **Solidity**.

Implementacja - weryfikacja podpisu w Solidity

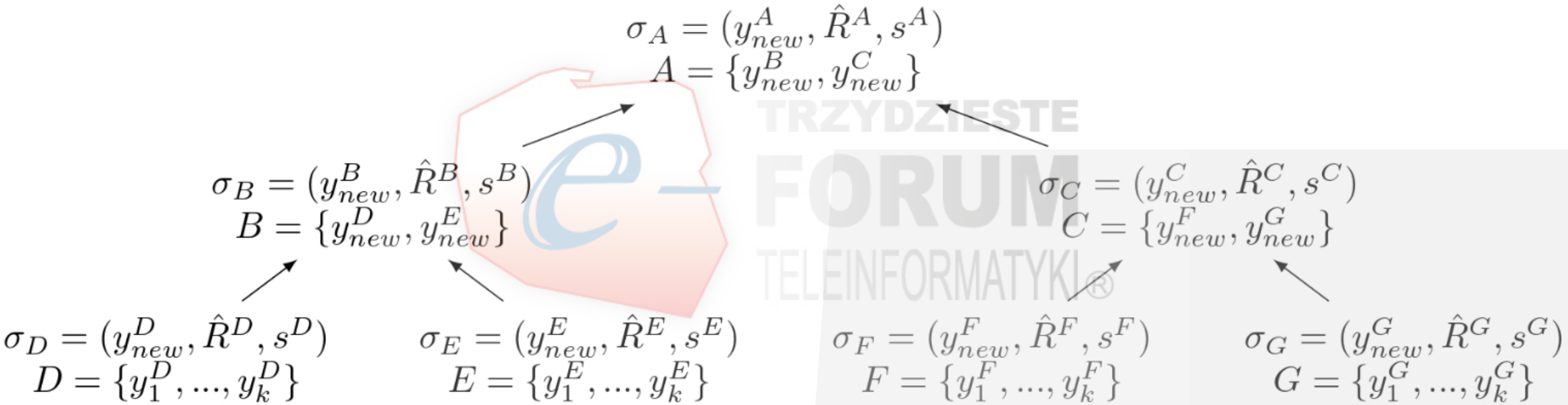
```
function verifyHierarchicalSignature(
    uint256 message,                    // Original message that was signed.
    uint256[2] memory newPublicKey,     // Newly generated public key for this level of the hierarchy
    uint256[2][] memory Rs,             // Parts of Schnorr sub-signatures collected in the ring signature.
    uint256[2][] memory RSigmas,        // R components of the aggregated signatures for verification
    uint256[] memory sSigmas,           // S components of the aggregated signatures.
    uint256 sum,                        // The aggregated sum of scalars, used to verify the combined signature's validity.
    uint256[2][] memory publicKeys      // Public keys of all participants in the ring signature.
) public view returns (bool) {

        // Initialize a point on the curve to aggregate sums.
        G1Point memory aggregatePoint = G1Point(0, 0);

        // Flag to verify each sub-signature within the ring.
        bool subSignatureVerified = true;

    for (uint i = 0; i < publicKeys.length; i++) {

            // Verifies each sub-signature by checking if the given sSigma generates the expected RSigma when combined with the message and public key.
            subSignatureVerified = subSignatureVerified && verifySchnorrSignature(Rs[i][0] + Rs[i][1], newPublicKey, RSigmas[i], sSigmas[i]);

            // This hash represents a unique challenge for each signer's contribution.
            uint256 h = hash(abi.encode(message, Rs[i][0], Rs[i][1], RSigmas[i][0], RSigmas[i][1], sSigmas[i]));

            // Aggregate the sum of all R points and public key points weighted by the hash.
            // This is part of the verification equation for the ring signature.
            aggregatePoint = add(aggregatePoint, add(G1Point(Rs[i][0], Rs[i][1]), multiply(G1Point(publicKeys[i][0], publicKeys[i][1]), h)));
        }

        // Verify the final aggregated signature
        return isEqual(multiply(G1, sum), aggregatePoint) && subSignatureVerified;
}
```

Funkcjonalność anamorficzna

Poprzez rozszerzenie procedury podpisów, względem oryginalnej pracy, możliwe jest używanie haszy ukrytych wiadomości jako losowości przy tworzeniu częściowych podpisów.

Upublicznienie treści takich wiadomości dowodzi, że skorelowany klucz publiczny nie został wykorzystany do stworzenia podpisu hierarchicznego - wyłączając poddrzewo z grona potencjalnych podpisujących.

Funkcjonalność anamorficzna



Bibliografia

1. Łukasz Krzywiecki, Małgorzata Sulkowska, and Filip Zagórski. "Hierarchical Ring Signatures Revisited – Unconditionally and Perfectly Anonymous Schnorr Version." In Rajat Subhra Chakraborty, Peter Schwabe, and Jon Solworth, editors, Security, Privacy, and Applied Cryptography Engineering, pages 329–346, Cham, 2015. Springer International Publishing.

1. Mirosław Kutyłowski, Giuseppe Persiano, Duong Hieu Phan, Moti Yung, and Marcin Zawada. "The Self-Anti-Censorship Nature of Encryption: On the Prevalence of Anamorphic Cryptography." Cryptology ePrint Archive, Paper 2023/434, 2023. https://eprint.iacr.org/2023/434.