КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (32), 2026
ISSN 2663 – 4023
DOI 10.28925/2663-4023.2026.32.1089
УДК 004.62

Момрик Ярослава Богданівна
асистент кафедри захисту інформації
Національний університет "Львівська політехніка", Львів, Україна
ORCID: 0009-0001-4114-0347
yaroslava.b.momryk@lpnu.ua

ПРОЕКТУВАННЯ ГРАФІЧНИХ ХЕШІВ З СІЛЛЮ НА ОСНОВІ МЕТОДУ КОДУВАННЯ UUID

Анотація. У статті запропоновано нову модель графічного хешуванн на базі методу кодування унікальних ідентифікаторів UUID (Universal Unique Identifier) у графік та доповнення цього методу, що перетворює ідентифікатори в унікальні захищені від несанкціонованого відтворення візуальні представлення**.** Розроблена методика побудови графічних хешів на основі UUID,. поєднує криптографічну унікальність із візуальною інтерпретацією, створюючи одноразові та постійні графічні ключі з мінімальними вимогами до серверних ресурсів. Запропонована графічна модель хешування перетворює UUID у графічне зображення з візуальним доповненням солі. Підхід поєднує числові UUID-ідентифікатори з двовимірним просторовим відображенням та механізмом соління для створення візуально інтерпретованого, але криптографічно захищеного зображення-хешу. При цьому сіль частково промальовується на зображенні. Метод забезпечує двошаровий механізм перевірки – математичний і візуальний – підвищуючи рівень захисту цифрових записів і зменшуючи ризики, пов'язані з атаками. А також запропоновано метод створення підмножини хешів на основі одного унікального ідентифікатора з можливістю побудови функції для збереження правил генерації одноразових ключів, та розглядом підходів для підбору параметрів для неї та шифрування, Наведено механізм створення графічних одноразових хешів-кодів на основі створення множини з часовою позначкою, виходячи з одного унікального ідентифікатора. Розглянуто стратегії генерації таких хешів – з базуванням на сервері та на клієнті. Запропоновано метод інтеграції третього виміру у графічний двовимірний хеш через промальовування геометричних параметрів для багатошарового кодування метаданих, зокрема часу створення. Отримана методика ефективна для реляційних баз даних, систем публікацій, архівів цифрових об'єктів і середовищ, де потрібна перевірка достовірності без розкриття первинних даних.

Ключові слова: графічний хеш, захист даних, кодування UUID, візуальна криптографія, безпечна ідентифікація, хеш з віображенням солі.

ВСТУП

У сучасних інформаційних системах графічні хеші набувають дедалі більшої актуальності як інструмент забезпечення цілісності та автентичності візуальних даних. На відміну від традиційних криптографічних хешів, ефективних для текстових і числових об'єктів, графічні та перцептуальні хеші враховують специфіку зображень і формують стійкі відбитки, здатні фіксувати мінімальні зміни контенту при збереженні загальної структури.

Існуючі підходи переважно орієнтовані на простоту використання та стійкість до незначних трансформацій, таких як стиснення, освітлювальні спотворення чи локальні зміни пікселів. Водночас багато схем не застосовують криптографічну сіль для генерації координат або використовують її лише для зміщення координат, обмежено

контролюють одноразовість або TTL, а серверні системи часто потребують збереження значної кількості даних про поточні стани.

Запропонований у цій роботі підхід поєднує UUID (Universal Unique Identifier), криптографічну сіль та координатну трансформацію для створення графічних хешів із керованими властивостями – як одноразових, так і постійних – при мінімальному обсязі даних на сервері.

Аналіз останніх досліджень і публікацій. Забезпечення цілісності та ідентичності даних розглядається у літературі як одне з ключових завдань інформаційної безпеки. Традиційні алгоритми хешування, зокрема SHA-256 та HMAC, ефективні для текстових і числових даних, однак не враховують особливості візуальних об’єктів. Тому зростає інтерес до методів графічного та перцептуального хешування, що поєднують стійкість із семантичною схожістю [1, 2, 3, 4].

Зокрема, у [4] надано систематичний огляд методів перцептуального хешування, що використовуються для аутентифікації зображень і виявлення маніпуляцій. Підкреслюється, що ці підходи важливі для забезпечення достовірності даних у цифрових публікаціях і судово-експертних системах.

У дослідженні [5] запропоновано самонавчальний метод графічного хешування, який використовує глибокі нейронні мережі для формування стійкого відбитку, здатного фіксувати незначні зміни контенту при збереженні загальної структури. Результати показали високу стабільність при JPEG-стисканні та освітлювальних спотвореннях, що робить підхід придатним для систем контролю автентичності цифрових ресурсів.

Автори [6] проаналізували стійкість таких алгоритмів до атак. Вони доводять, що поширені реалізації (PDQ, PhotoDNA, NeuralHash) можуть бути вразливими до незначних змін пікселів або локальних фільтрів, що дозволяє обійти перевірку. Водночас результати підкреслюють потенціал комбінування перцептуальних і криптографічних хешів для побудови захищених клієнтських систем сканування.

У [3] запропоновано модель GHashing, що забезпечує пошук за подібністю у великих графових базах даних. Автори [7] розробили масштабований підхід SGH, який застосовує перетворення ознак для підвищення точності. Автори [8] продемонстрували переваги глибокого спільного навчання (deep collaborative hashing) для пошуку зображень, що є актуальним для захисту мультимедійних сховищ і баз біометричних даних.

Окремий напрям досліджень пов'язаний з інтеграцією хешів у QR-коди або інші графічні ідентифікатори. Автори [9, 10] розробили схему захищеного QR-коду на основі візуальної криптографії. У [11] запропоновано підхід до аутентифікації зображень, у якому QR-код вбудовується у зображення як водяний знак, що містить хеш-підпис об'єкта.

Окрім того, автори [1, 12] зосереджені на практичних аспектах використання UUID у великих базах даних та стандартах цифрової ідентифікації. У літературі розглянуто також методи оптимізації їхньої продуктивності та захисту від колізій [2, 13]. Це створює підґрунтя для інтеграції графічних хешів у сучасні системи керування даними та стандарти цифрової ідентифікації [14].

Проблематика дослідження. Методи графічного та перцептуального хешування активно розвиваються для задач автентифікації зображень, пошуку за подібністю та інтеграції у QR-коди. Водночас візуальні хеші залишаються статичними та детермінованими, що створює ризики несанкціонованого копіювання та повторного використання. Водночас сьогодення дає нові виклики – почастішали випадки зломів та

захоплення облікових записів при використанні для авторизації QR-кодів та візуальної верифікації обличчя. Актуальним є удосконалення механізмів побудови стійких графічних хешів та інтеграції додаткових рівнів інформації для забезпечення багатофакторної графічної верифікації.

Мета роботи полягає в обґрунтуванні рекомендацій щодо вибору графічного зображення для побудови графічних хешів, розробці практичних стратегій побудови стійких графічних хешів на основі UUID, способів шифрування та створенні одноразових хеш-кодів та додаткових графічних шарів для їх верифікації.

Для реалізації поставленої мети необхідно вирішити наступні завдання:

1. Виробити стратегію побудови графічних хешів на основі методу перекодування UUID у графік для створення унікальної графічної моделі.
2. Розробити оптимальний підхід до соління з явним промальовуванням солі на графі.
3. Розробити методику створення одноразових кодів-хешів.
4. Запропонувати підхід до використання третього виміру координат для кодування додаткової інформації.

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

1. Метод графічного кодування UUID як основа для побудови графічного хешу. Хеш запропоновано будувати на основі методу графічного перетворення UUID, який наведений наведний у [15], з врахуванням деяких удосконалень. Метод базується на наступному алгоритмі:

Крок 1. Розбиття на логічні частини. UUID розділяється на п'ять числових компонентів за символом '–'.

Крок 2. Конвертація у десяткові числа. Кожна частина перетворюється у масив десяткових чисел.

Крок 3. Побудова координат графіка. Для кожного елемента масиву A[s] створюється набір координат (x_i, y_i), що визначає положення відповідної точки на графіку. Кожне число розміщується у своїй чверті системи координат – наприклад , як наведено у таблиці 1. Зміна послідовності побудови та положення координатного центру забезпечать додатковий рівень захисту.

Таблиця 1

Розподіл числових складових UUID за координатними чвертями

№ числа	Координатна чверть
1	I
2	II
3	(на осі x=0)
4	IV
5	III

A[5]={A[1,1]…A[1,n_1]}, A[2,1]…A[2,n_2]}. {A[3,1]…A[3,n_3]}, {A[4,1]…A[4,n_4]}, {A[5,1]…A[5,n_5]}} де n_s – довжина відповідного числа-вектора масиву.

Тоді X Y – масиви координат для кожного A[s], де s=1,5 за формулами 1.

$$X[s,i] = \begin{cases} x=i, & s=1 \\ x=-1*i, & s=2 \\ x=0, & s=3 \\ x=i, & s=4 \\ x=-1*I, & s=5 \end{cases} \qquad Y[s,i] = \begin{cases} y=A[s,i], & s=1 \\ y=-1*A[s,i], & s=2 \\ y=A[s,i], & s=3 \\ y=A[s,i], & s=4 \\ y=-1*A[s,i], & s=5 \end{cases} \qquad (1)$$

i=1, n_s з кроком 1 (крок можна задавати інший)

Фактично це табуляція функції f(y) = y·φ(s), де φ(s) – функція визначення знака числа залежно від номера s у масиві A.

Крок 4. Можливе додавання солі (крок необов'язковий). Для підвищення надійності додається контрольне число A[6], що розміщується симетрично відносно осі y для A[3] (наприклад x=0, а y – модуль кожної цифри з від'ємним значенням). Спосіб розташування «солі» – секрет сервера.

Крок 5. Побудова графіка. Отримані координати використовуються для побудови лінійного графіка або точкового зображення у форматі BMP або SVG.

Цей же алгоритм можна використати для LUID, якщо масив цифр розбити на 5 частин і будувати координати за тими ж формулами, та кроками алгоритму, що й для UUID.

2. Можливості хешування та шифрування – отримання стійкого хешу.

2.1. Векторне подання координат та «сіль»

Як зазначено у [15], якщо на графічному зображенні відсутні координатні позначення, за якими можливо визначити масштаб або систему відліку, то здійснити зворотне перетворення графічних значень у числові координати практично неможливо. Якщо потрібна зворотня конвертація – можна позначати лівий кут початку графіка, або початок координат. Але для покращення цього підходу особливо, якщо є потреба позначати початок координат, слід врахувати, що числа – складові , отримані на кроці 2 алгоритму, мають різну довжину – різна кількість точок у чвертях, по цьому можна здогадатися, яка цифрова частина у якій чверті розташована. Тому нижче пропонується доповнення алгоритму для створення ще більш захищеного зображення.

Доповнення алгоритму графічної конвертації UUID. Згідно поданого у [15] алгоритму UUID розбивається на вектори, кожний з яких містить число – його цифри дають координати точок графіка – отримуємо масив векторів A. Для забезпечення захисту графічного зображення пропонується кожному вектору координат A_i додати сіль s_i, такої довжини, щоб забезпечити однакову довжину чисел-координат – однакову кількість координат у кожній чверті графіка, A[m]={{A_1 = A[1,1]...A[1,n_1] + s_1}, {A_2 = A[2,1]...A[2,n_2] + s_2},...{Am= A[k,1]...A[k,n_k] + s_k}}. Сіль можна формувати на основі незмінної або напівнезмінної характеристики клієнта (наприклад, час створення акаунта, контрольний параметр у базі даних).

Отже, якщо сіль, яка зараз позначена на рисунку 1 іншим кольором, на етапі побудови хешу для UUID a3c1f9b2-7d8e-4a6f-b123-9f0e4c5d8a77 позначати так само, як і значимі цифри координат та розташувати по осі x з тим же кроком, що і точки UUID-зорієнтуватися, яке число (частина UUID) розташоване у якій чверті, стає неможливо.

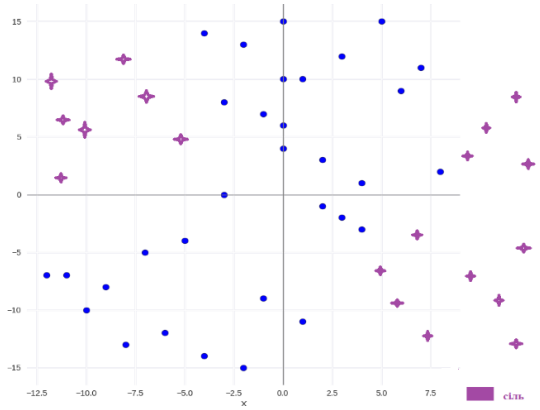

Рис1 Графічне зображення UUID з доповненням солі

Отримуємо ще більш захищений спосіб кодування, бо навіть якщо координати не зашифровані (виходимо з того, що початок координат позначено на графіку) – лише сервер, який знає свій алгоритм перетвоення, зможе вирахувати координати і сам UUID Таке представлення також дозволяє:

- застосувати стандартні хеш-функції до кожного вектора або до конкатенованого масиву;
- здійснювати перевірку цілісності на рівні чисел – складових масиву A.

2.2. Концепція множини модифікованих ідентифікаторів. Побудова одноразових графічних ключів. Пропонується виходити з одного базового UUID та користуватися множиною, створеною на його основі – генерувати кожний елемент множини за певним правилом. Наприклад, в залежності від параметру унікального часу запиту та додаткового параметру – обраної особливості клієнта – модифікувати початковий UUID. Це вже буде аналогія з генерацією цифрових підписів, але відкритим ключем буде UUID, а внутрішнім ключем буде не звичайний ключ, а функція – набір правил, переданий клієнту як ключ, або функція встановлена, як програма криптозахисту. На вхід цієї функції можуть передаватися ще дані, які знатиме лише сервер – наприклад, аргумент, який залежить від унікального часу запиту. Тоді на момент запиту клієнта буде генеруватися нове значення – видозміна початкового UUID на основі переданих правил аргументів та моменту запиту.

Таким чином на основі UUID, який виконує роль «відкритого ключа»/ідентифікатора, генеруємо множину похідних елементів. Кожен її елемент отримується як результат функціональної трансформації базового UUID з використанням:

1. параметра унікального часу запиту (nonce / timestamp);
2. додаткового персоналізованого параметра (наприклад: частина атрибутів користувача, контрольна ознака);
3. набору правил/функцій, які виконуються на клієнті або сервері (ці правила — «внутрішній ключ»).

Відкритим елементом залишається сам UUID, а секретом є функція (набір правил) та/або серверний секрет, що генерує похідні значення динамічно під час кожного запиту.

Алгоритм генерації похідних ідентифікаторів.

1. Вхідні дані: UUID, timestamp, client_param, server_secret (за наявності);
2. Розбиття UUID на вектори A_i згідно з визначеною довжиною блоків;
3. Формування солі s_i = f(client_param, server_secret, i) – може бути постійною або динамічною (можна передбачити глобальну сіль для всього масиву);
4. Застосування перетворення: A'_i = transform(A_i, s_i, timestamp) – наприклад, арифметична модифікація, XOR (виключне АБО) або криптографічне множення в полі;
5. Серіалізація масиву A' = concat(A'_1,...,A'_k) і обчислення хешу H = HASH(A') – наприклад, SHA-256 (Secure Hash Algorithm);
6. За потреби – шифрування E = ENC(A', key) – наприклад, AES-GCM (Advanced Encryption Standard – Galois/Counter Mode) – та/або підпис TAG = HMAC(A', server_secret) (Hash-based Message Authentication Code).

Результатом є тимчасовий модифікований ідентифікатор (або його графічне подання), який передається або зберігається замість оригінального UUID.

Як «маячок» достовірності до солі можна додавати підмножину, що формує контрольне слово (control code). Це може бути частина солі або окремий код, який

сервер перевіряє під час валідації. Наявність контрольного слова збільшує ймовірність виявлення підміни або помилки при відновленні.

Після отримання числового масиву A доцільно застосувати стандартні, перевірені засоби криптографії:

- Salted hashing: H = HASH(salt || A') (SHA-256/384/512 залежно від потреб);
- KDF (Key Derivation Function – функція виведення ключа), наприклад, HKDF (HMAC-based KDF): для виведення ключів з постійних характеристик;
- HMAC (Hash-based Message Authentication Code – код автентифікації повідомлення на основі хеш-функції) для автентичності: TAG = HMAC(key, A');
- Authenticated encryption (AES-GCM або ChaCha20-Poly1305) для шифрування масиву даних із вбудованою перевіркою цілісності.

При реалізації алгоритму потрібно врахувати наступні ризики:

- Допуски при валідації часу одноразових ключів: потрібно встановити допустимі часові вікна (time window) для timestamp-зсувів для синхронізації з годинником клієнта.
- Регуляторні/конфіденційні обмеження: при використанні персональних ознак як солі потрібно переконатись у відповідності вибору характеристик вимогам конфіденційності (GDPR – General Data Protection Regulation, Загальний регламент захисту даних,).

Практична реалізація запропонованого алгоритму може відбуватися за двома принципово різними архітектурними підходами, кожен з яких має свої переваги та особливості застосування.

Серверна генерація. За цього підходу сервер зберігає базовий UUID, функцію трансформації, server_secret та всі необхідні параметри. Клієнт надсилає запит з мінімальним набором даних – timestamp (який може генеруватися та зберігатися також на сервері) та опціонально параметри клієнта (наприклад, IP-адресу), сервер може доєднати дані про клієнта з бази даних. Вся генерація відбувається на сервері:

Алгоритм роботи (псевдкод):

```
// На сервері
input: request від клієнта
timestamp = get_current_timestamp() або extract_from_request()
client_param = extract_client_param(request)
UUID = get_from_database(client_id)
server_secret = get_server_secret()

split UUID -> A = [A₁,...,Aₖ]

for i in 1..k:
    sᵢ = KDF(server_secret, client_param, i)
    A'ᵢ = (Aᵢ + sᵢ + timestamp mod T) mod M

A' = concat(A'₁,...,A'ₖ)
TAG = HMAC(server_secret, A')
token = {A', TAG, timestamp}

return token to client
```

Процес верифікації:

```
// На сервері при перевірці
input: token від клієнта
extract A', TAG, timestamp from token

if |current_time - timestamp| > allowed_window:
    reject // токен застарів
```

```
reconstruct:
    split UUID -> A = [A₁,...,Aₖ]
    for i in 1..k:
        sᵢ = KDF(server_secret, client_param, i)
        A'ᵢ_check = (Aᵢ + sᵢ + timestamp mod T) mod M

    A'_check = concat(A'₁_check,...,A'ₖ_check)
    TAG_check = HMAC(server_secret, A'_check)

if TAG == TAG_check and A' == A'_check:
    accept
else:
    reject
```

Верифікація у такому випадку також повністю серверна. При отриманні токена перевіряється актуальність, після чого відбувається реконструкція з наступною перевіркою TAG та A'. Такий підхід забезпечує повний контроль сервера над процесом генерації та унеможливлює витік функції трансформації, проте вимагає звернення до сервера при кожному запиті та створює додаткове навантаження.

Альтернативний підхід – клієнтська генерація, передбачає, що клієнт отримує функцію трансформації (під час ініціалізації або реєстрації) та може самостійно генерувати похідні ідентифікатори. Сервер при цьому зберігає лише базовий UUID та server_secret для верифікації. Клієнт виконує трансформацію локально:

Алгоритм роботи (псевдокод):

```
// На клієнті (під час ініціалізації)
input: UUID, transform_function від сервера
store UUID, transform_function locally

// На клієнті (під час генерації токена)
timestamp = get_current_timestamp()
client_param = get_local_param() // device_id, session_id, тощо

split UUID -> A = [A₁,...,Aₖ]

for i in 1..k:
    sᵢ = client_KDF(client_param, i) // без server_secret
    A'ᵢ = (Aᵢ + sᵢ + timestamp mod T) mod M

A' = concat(A'₁,...,A'ₖ)
partial_TAG = client_HMAC(client_param, A') // без server_secret

token = {A', partial_TAG, timestamp, client_param}
send token to server
```

Процес верифікації на сервері:

```
// На сервері
input: token від клієнта
extract A', partial_TAG, timestamp, client_param from token

if |current_time - timestamp| > allowed_window:
    reject

reconstruct with server_secret:
    split UUID -> A = [A₁,...,Aₖ]
    for i in 1..k:
        sᵢ = server_KDF(server_secret, client_param, i)
        A'ᵢ_check = (Aᵢ + sᵢ + timestamp mod T) mod M

    A'_check = concat(A'₁_check,...,A'ₖ_check)
```

```
    full_TAG = HMAC(server_secret, A'_check)

if verify_structure(A', A'_check) and verify_signature(full_TAG):
    accept
else:
    reject
```

Примітка: у реальній системі для усіх сценаріїв A'_i має бути сформовано за допомогою криптографічно стійких операцій.

Важливо відзначити, що client_KDF працює без server_secret, тому сервер при верифікації має додатково застосовувати свій секретний ключ для повної перевірки. Це зменшує навантаження на сервер та дозволяє автономну роботу клієнта, але створює ризик витоку функції трансформації.

На практиці доцільним є гібридний варіант, де клієнт генерує частину трансформації (A'_client = client_transform(UUID, timestamp, client_param)), а сервер додає свій захисний шар (A'_final = server_transform(A'_client, server_secret, timestamp)). Це дозволяє поєднати ефективність клієнтської генерації з надійністю серверного контролю.

Вибір конкретного підходу визначається вимогами системи. Серверна генерація критично важлива для високозахищених систем (банківські додатки, медичні системи) або систем з малою кількістю запитів. Клієнтська генерація виправдана при високій частоті генерації токенів, необхідності офлайн-режиму або у розподілених системах з великою кількістю клієнтів. Гібридний підхід являє собою компромісне рішення для більшості прикладних задач.

2.3. Використання третього виміру для кодування часу або солі. Для додаткового рівня захисту та можливості верифікації часу створення або інших задаих характеристик можна використовувати координату z (третій вимір). Йдеться не про повноцінне тривимірне відображення, а про кодування часової мітки чи солі через геометричні властивості у двовимірному просторі.

Принцип роботи:

1. Табуляція графіка відбувається з певним фіксованим кроком по осях X та Y;
2. Точки, що не знаходяться у вузлах табуляції, містять закодовану інформацію про третій вимір;
3. Різні категорії цифр третього виміру розміщуються у різних координатних чвертях.

Способи кодування часу або солі:

1. Відстань між точками:
   - $\Delta d = |P_1 - P_2|$ кодує рік, дату або timestamp.

   Наприклад: відстань у пікселях / 10 = рік (2024) або: $\Delta d \bmod 365$ = день року
2. Зміщення від вузлів табуляції:
   - для точки з очікуваними координатами (x_i, y_i) додається зміщення z;
   - фактичні координати: $(x_i + z \cdot \cos(\alpha), y_i + z \cdot \sin(\alpha))$, де z кодує часову мітку, а α — напрямок зміщення.
3. Функція, що створює додаткові («фантомні») точки:
   - між регулярними точками табуляції з'являються додаткові точки;
   - їхня позиція визначається формулою: z = f(timestamp or salt).

   Наприклад: для року 2024 → точка зміщується на 24 пікселі від базової лінії.

По суті додано ще один рівень безпеки – тепер графік містить не тільки UUID та сіль, але й закодовану часову чи іншу контрольну мітку, що робить підробку практично неможливою без знання всіх параметрів алгоритму.

Детектування при верифікації:

- система знає базовий крок табуляції;
- виявляє точки, що не потрапляють у вузли сітки;
- відповідно до способу кодування та обчислює та декодує z-координату назад у часову мітку;
- порівнює з очікуваним часом створення (±допустима похибка) чи з іншою міткою – контрольною сіллю.

Таким чином створюється тривимірний візуальний хеш, де третій вимір прихований у двовимірному зображенні через геометричні розрахунки.

Слід зазначити, якщо застосовуємо зміщення точок, то може виникнути накладання точки на вузол табуляції. Тоді треба якось її позначати, наприклад кольором(додаткове позначення лише для тих точок, що потрапили у вузол), що може дати незначну інформацію для зловмисника. Більш захищеним зображення буде, якщо вираховувати значення третього виміру через відстані між точками – замість зміщення точок від базових позицій, інформація може кодуватися через відстані між послідовними точками на графіку в наступний спосіб:

- кожна пара сусідніх точок (P_i, P_{i+1}) формує відстань d_i;
- відстань d_i кодує цифру наприклад часової мітки (рік, дату, час).

Таке хешоване зображення буде виглядати більш природно і його важче буде декодувати.

2.4. Створення візуального хеш-токена. Принципи побудови третього шару можна застосувати для створення візуального хеш-токена, у якому можна закодувати будь-які дані. Таким чином отримуємо метод для кодування не лише часових міток чи солі, а й довільної інформації. Це відкриває можливість створення власного формату візуальних токенів, здатних замінити QR-коди у системах авторизації та захисту.

Алгоритм перетворення UUID + сіль у координати відомий лише серверу. Сторонній спостерігач бачить лише зображення, але не може відновити дані без знання алгоритму.

Процес верифікації відбувається так: клієнт передає зображення, сервер розпізнає координати та відновлює закодовану інформацію.

Основна перевага полягає у тому, що такий код неможливо підробити чи згенерувати коректно без доступу до алгоритму. Навіть якщо картинку перехоплено, вона не несе корисної інформації для сторонніх осіб.

ВИСНОВКИ ТА ПЕРСПЕКТИВИ ПОДАЛЬШИХ ДОСЛІДЖЕНЬ

Розроблений підхід до побудови графічного хешу на основі UUID дозволяє поєднати криптографічну унікальність з візуальною інтерпретацією, що підвищує прозорість і контроль цілісності даних. Поєднання методів криптографічного хешування, просторової візуалізації та контекстного соління розвиває новий напрям у захисті даних – графічне хешування як форму візуальної криптографії. Методика дозволяє на базі UUID створити клас одноразових графічних ключів, які:

- не повторюються;
- не залежать від введення користувача;
- економлять ресурси сервера;
- забезпечують високий рівень безпеки та контролю одноразовості.

Запропонований метод забезпечує багатовимірну графічну верифікацію та промальовування солі на графі, що обґрунтовує його новизну та практичну цінність у

порівнянні з класичними графічними хешами. Метод графічних хешів може бути використаний у різних сферах, де важлива автентичність та захист даних:

- Авторизація користувачів. Використання графічних хешів як альтернативи QR-кодам дозволяє уникнути стандартних вразливостей, оскільки алгоритм кодування відомий лише серверу. Це знижує ризик підробки та перехоплення токенів.
- Захист біометричних систем. Поєднання UUID та криптографічної солі з координатними трансформаціями створює додатковий рівень перевірки при верифікації обличчя чи інших біометричних ознак.
- Документування часу та подій. Вбудована часова мітка у графічний хеш дозволяє підтверджувати момент створення ідентифікатора, що корисно для цифрових підписів, журналів подій та систем аудиту.

Подальші дослідження передбачають оптимізацію процесів візуалізації та використання солі, інтеграцію з алгоритмами машинного навчання для автоматичного розпізнавання шаблонів і розширення функціональності для багатокористувацьких середовищ. У перспективі планується розглянути наступні аспекти

- Розширення алгоритму для підтримки різних типів даних (текст, ключі, метадані).
- Інтеграція з мобільними додатками для швидкої авторизації без QR-кодів.
- Використання у блокчейн-системах як візуальних токенів із вбудованою часовою міткою.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Agarwal, V., Singh, R., & Patel, M. (2023). Performance optimization in UUID-based large-scale database systems using multi-criteria decision methods. *International Journal of Computer Applications, 184*(2), 45–52. https://doi.org/10.5120/ijca2023912345
2. Chen, B., & Zhao, Y. (2022). Security vulnerabilities in academic publishing platforms. *Journal of Information Security and Applications, 67*, 103118. https://doi.org/10.1016/j.jisa.2022.103118
3. Qin, Z., Bai, Y., & Sun, Y. (2020). GHashing: Semantic graph hashing for approximate similarity search in graph databases. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 905–913). ACM. https://doi.org/10.1145/3394486.3403085
4. Fridrich, J. (2015). Visual hashing and perceptual image signatures. *IEEE Transactions on Information Forensics and Security, 10*(6), 1199–1212. https://doi.org/10.1109/TIFS.2015.2414424
5. Fonseca-Bustos, J., Ramírez-Gutiérrez, K. A., & Feregrino-Uribe, C. (2024). A robust self-supervised image hashing method for content authentication. *Pattern Recognition Letters, 180*, 50–59. https://doi.org/10.1016/j.patrec.2024.109462
6. Jain, A., & Kumar, R. (2022). Vulnerabilities of perceptual hashing algorithms: A study on PDQ, PhotoDNA, and NeuralHash. *Digital Investigation, 41*, 301–312. https://doi.org/10.1016/j.diin.2022.301312
7. Jiang, Q.-Y., & Li, W.-J. (2015). Scalable graph hashing with feature transformation. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1360–1366). AAAI Press. https://doi.org/10.5555/2832249.2832420
8. Zhang, Z., Li, H., & Gao, L. (2023). Deep collaborative graph hashing for discriminative image retrieval. *Pattern Recognition, 132*, 109462. https://doi.org/10.1016/j.patcog.2023.109462
9. Naor, M., & Shamir, A. (1995). Visual cryptography. In A. De Santis (Ed.), *Advances in cryptology—EUROCRYPT'94* (pp. 1–12). Springer. https://doi.org/10.1007/3-540-49264-X_24
10. Cao, X., Feng, L., Cao, P., & Hu, J. (2016). Secure QR code scheme based on visual cryptography. In *Proceedings of the International Conference on Artificial Intelligence and Information Engineering* (pp. 210–219). Atlantis Press. https://doi.org/10.2991/aiie-16.2016.99
11. Liu, X., Zhang, B., Wen, Y., Tang, X., & Su, H. (2022). An improved image authentication method using QR code watermarking. In *International Conference on Artificial Intelligence and Security (ICAIS)*

(Lecture Notes in Computer Science, Vol. 13340, pp. 350–362). Springer. https://doi.org/10.1007/978-3-031-06791-4_27

12. Thurman, T. R., et al. (2024). *Research results and recommendations for universally unique identifiers in product data standards*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8472
13. Singh, R., & Dandotiya, A. (2023). Bug bounty-driven approaches for detecting IDOR vulnerabilities in web applications. *Journal of Cybersecurity and Information Systems*, *9*(2), 58–66. https://doi.org/10.1109/JCIS.2023.123456
14. Zhang, H., & Li, T. (2022). Integrating SHA-based integrity verification in secure data workflows. *Journal of Network Security*, *19*(3), 141–150. https://doi.org/10.1109/JNS.2022.123456
15. Momryk, Y. (2025). Pidkhody do bezpechnoho zberizhennia ta opratsiuvannia danykh na osnovi UUID identyfikatoriv: Metod hrafichnoho predstavlennia. *Cybersecurity: Education, Science, Technique*, *3*(31), 140–154. https://doi.org/10.28925/2663-4023.2025.31.1010

КІБЕРБЕЗПЕКА: освіта, наука, техніка
CYBERSECURITY: EDUCATION, SCIENCE, TECHNIQUE
№ 4 (32), 2026
ISSN 2663 – 4023

Yaroslava Momryk
Assistant of Department of Information Security
Lviv Polytechnic National University, Lviv, Ukraine
ORCID: 0009-0001-4114-0347
Yaroslava.b.momryk@lpnu.ua

DESIGN OF GRAPHIC HASHES WITH SALT BASED ON THE METHOD OF CODING OF UUID

Abstract. The article proposes a new model of graphic hashing based on the method of encoding unique identifiers UUID (Universal Unique Identifier) into a graph and an addition to this method that converts identifiers into unique visual representations protected from unauthorized reproduction. A method for constructing graphic hashes based on UUID is developed. combines cryptographic uniqueness with visual interpretation, creating one-time and permanent graphic keys with minimal requirements for server resources. The proposed graphic hashing model converts UUID into a graphic image with a visual addition of salt. The approach combines numerical UUID identifiers with a two-dimensional spatial representation and a salting mechanism to create a visually interpreted but cryptographically protected hash image. In this case, the salt is partially drawn on the image. The method provides a two-layer verification mechanism - mathematical and visual - increasing the level of protection of digital records and reducing the risks associated with attacks. A method for creating a subset of hashes based on a single unique identifier is also proposed with the possibility of constructing a function for storing the rules for generating one-time keys, and considering approaches for selecting parameters for it and encryption. A mechanism for creating graphic one-time hash codes based on creating a set with a time stamp, based on a single unique identifier, is presented. Strategies for generating such hashes – server-based and client-based – are considered. A method for integrating the third dimension into a graphic two-dimensional hash through drawing geometric parameters for multilayer metadata encoding is proposed. The resulting technique is effective for relational databases, publication systems, archives of digital objects and environments where authenticity verification is required without revealing the original data.

Keywords: graphical hash, data protection, UUID encoding, visual cryptography, secure identification, hash with salt.

REFERENCES (TRANSLATED AND TRANSLITERATED)

1. Agarwal, V., Singh, R., & Patel, M. (2023). Performance optimization in UUID-based large-scale database systems using multi-criteria decision methods. *International Journal of Computer Applications, 184*(2), 45–52. https://doi.org/10.5120/ijca2023912345
2. Chen, B., & Zhao, Y. (2022). Security vulnerabilities in academic publishing platforms. *Journal of Information Security and Applications, 67*, 103118. https://doi.org/10.1016/j.jisa.2022.103118
3. Qin, Z., Bai, Y., & Sun, Y. (2020). GHashing: Semantic graph hashing for approximate similarity search in graph databases. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (pp. 905–913). ACM. https://doi.org/10.1145/3394486.3403085
4. Fridrich, J. (2015). Visual hashing and perceptual image signatures. *IEEE Transactions on Information Forensics and Security, 10*(6), 1199–1212. https://doi.org/10.1109/TIFS.2015.2414424
5. Fonseca-Bustos, J., Ramírez-Gutiérrez, K. A., & Feregrino-Uribe, C. (2024). A robust self-supervised image hashing method for content authentication. *Pattern Recognition Letters, 180*, 50–59. https://doi.org/10.1016/j.patrec.2024.109462
6. Jain, A., & Kumar, R. (2022). Vulnerabilities of perceptual hashing algorithms: A study on PDQ, PhotoDNA, and NeuralHash. *Digital Investigation, 41*, 301–312. https://doi.org/10.1016/j.diin.2022.301312
7. Jiang, Q.-Y., & Li, W.-J. (2015). Scalable graph hashing with feature transformation. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1360–1366). AAAI Press. https://doi.org/10.5555/2832249.2832420

8. Zhang, Z., Li, H., & Gao, L. (2023). Deep collaborative graph hashing for discriminative image retrieval. *Pattern Recognition, 132*, 109462. https://doi.org/10.1016/j.patcog.2023.109462
9. Naor, M., & Shamir, A. (1995). Visual cryptography. In A. De Santis (Ed.), *Advances in cryptology—EUROCRYPT'94* (pp. 1–12). Springer. https://doi.org/10.1007/3-540-49264-X_24
10. Cao, X., Feng, L., Cao, P., & Hu, J. (2016). Secure QR code scheme based on visual cryptography. In *Proceedings of the International Conference on Artificial Intelligence and Information Engineering* (pp. 210–219). Atlantis Press. https://doi.org/10.2991/aiie-16.2016.99
11. Liu, X., Zhang, B., Wen, Y., Tang, X., & Su, H. (2022). An improved image authentication method using QR code watermarking. In *International Conference on Artificial Intelligence and Security (ICAIS)* (Lecture Notes in Computer Science, Vol. 13340, pp. 350–362). Springer. https://doi.org/10.1007/978-3-031-06791-4_27
12. Thurman, T. R., et al. (2024). *Research results and recommendations for universally unique identifiers in product data standards*. National Institute of Standards and Technology. https://doi.org/10.6028/NIST.IR.8472
13. Singh, R., & Dandotiya, A. (2023). Bug bounty-driven approaches for detecting IDOR vulnerabilities in web applications. *Journal of Cybersecurity and Information Systems, 9*(2), 58–66. https://doi.org/10.1109/JCIS.2023.123456
14. Zhang, H., & Li, T. (2022). Integrating SHA-based integrity verification in secure data workflows. *Journal of Network Security, 19*(3), 141–150. https://doi.org/10.1109/JNS.2022.123456
15. Momryk, Y. (2025). Pidkhody do bezpechnoho zberizhennia ta opratsiuvannia danykh na osnovi UUID identyfikatoriv: Metod hrafichnoho predstavlennia. *Cybersecurity: Education, Science, Technique, 3*(31), 140–154. https://doi.org/10.28925/2663-4023.2025.31.1010

Отримано редакцією журналу / Received: 15.01.26
Прорецензовано / Revised: 30.01.26
Схвалено до друку / Accepted: 26.03.26