Київський Національний університет імені Тараса Шевченка
Механіко-Математичний факультет
Кафедра алгебри і комп'ютерної математики

Курсова робота

На тему
Нечіткі екстрактори

Студента третього курсу
Механіко-математичного факультету
ОР Бакалавр
ОП Компютерна математика
Красовського Олександра Ігоровича

Науковий керівник:
доктор фізико-математичних наук, професор
Олійник Андрій Степанович

1.1 Зміст

1.2 Нечіткі екстрактори

Нечіткі екстрактори — це метод, який дозволяє використовувати біометричні дані як вхідні дані для стандартних криптографічних методів для підвищення їх безпеки. «Нечіткість» у цьому контексті стосується того факту, що фіксовані значення, необхідні для роботи криптографічних алгоритмів, будуть отримані зі значень, близьких до вихідного ключа, але не ідентичних, без шкоди для необхідної безпеки.

Нечіткі екстрактори — алгоритми, які дозволяють автентифікувати користувача за допомогою біометричного шаблону, створеного з біометричних даних користувача як ключа, шляхом вилучення однорідного випадкового рядка R з вхідних даних w, з допуском на шум. Якщо вхідні дані змінюється на w' але є близькими до w відповідно до умов алгоритму, рядок R буде реконструйовано як на w, так і на даних w'.

Для досягнення даного результату, під час початкового обчислення R процес також повертає допоміжний рядок P, який буде збережено для відновлення R і може бути оприлюднений без шкоди для безпеки R.

Безпека процесу також забезпечується, у випадку, коли зловмисник вносить зміни до значення P.

Після обчислення фіксованого рядка R, Дані можуть бути використані, наприклад, для узгодження ключів між користувачем і сервером лише на основі біометричного введення.

Властивості нечітких екстракторів:

- Нечіткий екстрактор є ймовірнісним Монте-Карло алгоритмом, для якого виконються умови:
  - $\forall$ неавторизованого користувача R^*, дані R^* відхиляються з ймовірністю 1
  - Для авторизованого користувача R, алгоритм з ймовірністю $1-\epsilon$ (де ϵ - ймовірність однобічної помилки алгоритму), приймає дані R та авторизує користувача
- Більшість нечітких екстракторів є параметричними моделями. Для параметричних нечіткоих екстрактор, існує набір параметрів $\theta \in \Theta$, для якого ймовірність однобічної помилки є як завгодно малою.
- На практиці, нечіткі екстрактори часто використовуються у комбінації з кодами виправлення помилок (англ. Error Correction codes, ECC). Властивості ECC надають можливість створення публічного значення P (check symbols), яке за умов безпеки криптографічного ескізу може бути оприлюднене без ризиків для безпеки користувача, і використане для відновлення ключа за вихідними даними нечіткого екстрактора.

1.3 Постановка задачі та огляд літератури

Постановка задачі:

Побудова параметричної моделі нечіткого екстрактору на основі однієї з моделей face-recognition з відкритим вихідним кодом. Вхідними даними нечіткого екстрактора є послідовність зображень, та набір параметрів з простору Θ. Алгоритм має два режими роботи: створення перевірочних символів, або ініціалізація алгоритму з існуючими перевірочними символами і побудова ключа. Результатом роботи алгоритму є криптографічний ключ, безпечий для використання у інших криптографічних алгоритмах. За випадково опублікованим ключем, не повино існувати можливостей отримання біометричних даних користувача.

Огляд літератури:

- Доцільність використання кодів Ріда-Соломона у якості ECC
  1. Відповідно до ресурсу [3]:
     The tradeoff between the error tolerance and the entropy loss depends on the choice of errorcorrecting code. For large alphabets ($\mathbb{F}$ is a field of size $\geq n$),
     one can use Reed-Solomon codes to get the optimal entropy loss of $2t \log |\mathbb{F}|$.
     No secure sketch construction can have better tradeoff between error tolerance and entropy loss, as searching for better secure sketches for the Hamming distance is equivalent to searching for better error-correcting codes.
     Better secure sketches, however, can be achieved if one is willing to
     slightly weaken the guarantee of correctness.
     ($\mathbb{F}$ - алфавіт ECC; t - вага Хемінга векторів, для яких вимірюється значення loss $= 2t \log |\mathbb{F}|$)

     Відповідно до [3], даний код конструює безпечну послідовність перевірочних символіволів, за якою не існує оптимального алгоритму отримання вхідних даних, якщо кількість перевірочних символів є достатньо малою. Також, даний код має оптимальну втрату ентропії після додавання перевірочних символів до послідовності: $2t \log |\mathbb{F}|$. Результат було враховано під час вибору коду ECC та реалізації нечіткого екстрактора.

1.3.1 Необхідні допоміжні відомості

Метричний простір

Означення (4): Метричний простір: Метричний простір — це набір $\mathcal{M}$ із функцією відстані $dis : \mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R}+$, для якої виконуються властивості: - нерівність трикутника dis(x, z) $\leq$ $dis(x, y) + dis(y, z)$ - симетрія $dis(x, y) = dis(y, x)$.

Examples: - dis = Hamming distance: \$h(v_1,v_2) = \$ sum of one-bits of $v_1 - v_2$ over $\mathbb{F}_2$ - dis = set difference: $\cup$ - dis = Levenstein distance

Min-Entropy, Average Min-Entropy

Означення (Min-Entropy):

Мінімальною ентропією $H_\infty(A)$ випадкової величини A називається значення:

$$H_\infty(A) = -\log \max_a Pr[A = a]$$

де $\max_a Pr[A = a]$ - передбачуваність A.

Означення (Average Min-Entropy):

Середньою ентропією випадкової величини A за умови B називають значення:

$H_\infty(A|B) =$ todo add descrtiption

Статистична відстань

Означення: Статистична відстань між двома ймовірнісними розподілами A_1 і A_2:

$$\mathrm{SD}(A_1, A_2) = \frac{1}{2} \sum_u |Pr(A_1 = u) - Pr(A_2 = u)|$$

Безпека нечіткого екстрактора зазвичай враховує статистичну
відстань між заданим розподілом (ключів екстрактору) і рівномірним розподілом U.

Захищені ескізи та нечіткі екстрактори

Захищений ескіз є важливою компонентою нечітких екстракторів. Безпечний ескіз є схемою, яка приймає в якості вхідної інформації зашумлену інформацію ω, наприклад біометричну інформацію,
та створює на її основі ескіз s, який є допоміжним рядком.

Означення: Нечітким ескізом називається набір функцій $(\mathrm{SS}, \mathrm{Rec})$, які мають наступні властивості: - Фунція створення ескізу SS на вхідних даних $\omega \in \mathcal{M}$ повертає ескіз
$s \in \{0,1\}^*$ - Функція відновлення зашумлених даних Rec для вхідного елементу $\omega_1 \in \mathcal{M}$ та ескізу $s \in \{0,1\}^*$, повертає рядок $\omega \iff \mathrm{dist}(\omega_1, \omega) < t$, де t - точність нечіткого екстрактора

Захищені схеми ескізу зазвичай використовують коди виправлення помилок, причому, захищений ескіз відновлює
вхідні дані ω_1 до $\omega \iff$ рядок ω_1 є подібним до ω.
Ескіз s є публічно доступним, оскільки він це розкриває достатньої інформації про секретні дані ω.
Одним з можливих методів створення безпечного ескізу є БЧХ-коди:

1.3.2 Криптографічна стійкість безпечних ескізів

Означення: Ескіз s називається $(\mathcal{M}, m, m, t)$ - безпечним,якщо

- $\forall$ W - розподілу над метричним простором $\mathcal{M}$, з мінімальною ентропією m,зловмисник має перевагу
  обмежену 2^{-m}, де $m \leq H_\infty(W|\mathrm{SS}(W)$ для відновлення значення W.

1.3.3 БЧХ-код

и є циклічними кодами, які задаються своїм породжуючим поліномом. Для його знаходження необхідно передусім визначити довжину коду n і мінімальну кодову відстань $d \leqslant n$. Знайти породжуючий поліном можна наступним чином:

Нехай α - примітивний елемент поля $GF(q^m)$ (тобто $\alpha^{q^m-1} = 1,\ \alpha^i = 1,\ i < q^m - 1$), нехай $\eta = \alpha^s$ - елемент поля $GF(q^m)$ порядку n, $s = (q^m - 1)/n$. Тоді нормований поліном $g(x)$ мінімальної степені над полем $GF(q)$, коренями якого є $d-1$ послідовних степеней $\eta^{l_0}, \eta^{l_0+1}, \ldots, \eta^{l_0+d-2}$ елемента η для деякого цілого l_0 (в тому числі 0 і 1), є породжуючим поліномом БЧХ-коду над полем $GF(q)$ з довжиною n і мінімальною відстанню $d_0 \geqslant d$.

Пояснимо, чому у отриманого коду будуть саме такі характеристики (довжина коду n, мінімальна відстань d_0). Справді, як показано у (8), довжина БЧХ-коду дорівнює порядку елемента η, якщо $d > 2$, і дорівнює порядку елемента η^{l_0}, якщо $d = 2$. Оскільки випадок $d = 2$ нас не цікавить (такий код може виявляти, але не може виправляти помилки), довжина коду буде дорівнювати порядку елемента η, тобто дорівнюватиме n. Мінімальна відстань d_0 може бути більшою за d, коли коренями мінімальних функцій (8) від елементів $\eta^{l_0}, \eta^{l_0+1}, \ldots, \eta^{l_0+d-2}$ будуть елементи, які доповнюють послідовність, тобто елементи $\eta^{l_0+d-1}, \eta^{l_0+d}, \ldots, \eta^{l_0+d_0-2}$.

Кількість перевірочних символів r дорівнює степеню $g(x)$, кількість інформаційних символів $k = n - r$, величина d називається конструктивною відстанню БЧХ-коду. Якщо $n = q^m - 1$, то код називається примітивним, інакше, неприміти вним.

Так само, як і для циклічного коду, кодовий поліном $c(x)$ може бути отриманий з інформаційного поліному $m(x)$ степені не більше $k-1$, шляхом перемноження $m(x)$ і $g(x)$.

1.3.4 Коди Ріда-Соломона (7)

Коди Ріда - Соломона є важливим окремим випадком БЧХ-коду, корені породжуючого многочлена якого лежать у тому ж полі, над яким будується код. Нехай α — елемент поля $GF(q)$, що має порядок n. Якщо α — примітивний елемент, його порядок дорівнює $q-1$, тобто $\alpha^{\{q\text{-}1\}=1,\ \alpha}i \neq 1, 0 < i < q-1$. Тоді нормований поліном $g(x)$ мінімального ступеня над полем $GF(q)$, коренем якого є $d-1$ послідовних ступенів $\alpha^{l_0}, \alpha^{l_0+1}, ..., \alpha^{l_0+d-2}$ елемента α, є породжуючим многочленом коду Ріда — Соломона над полем $GF(q)$:

$g(x) = (x - \alpha^{l_0})(x - \alpha^{l_0+1}) \ldots (x - \alpha^{l_0+d-2})$,

де l_0 - деяке ціле число (у тому числі 0 і 1), зазвичай обирається $l_0 = 1$. Ступінь многочлена $g(x)$ дорівнює $d-1$. Довжина отриманого коду n, мінімальна відстань d(є мінімальною з усіх відстаней Хеммінга всіх пар кодових слів, див. Лінійний код). Код містить $r = d-1 = \deg(g(x))$ перевірочних символів, де deg() позначає ступінь полінома; число інформаційних символів: $k = n - r = n - d + 1$. Таким чином, $d = n - k + 1$ і код Ріда — Соломона є кодом, що має максимальну кодову відстань (є оптимальним у сенсі границі Сінглтона). Кодовий поліном $c(x)$

може бути отриманий з інформаційного полінома $m(x)$,$\deg m(x) \leqslant k-1$ шляхом перемноження $m(x)$ і $g(x)$:

$c(x) = m(x)g(x)$.

1.3.5 Нечіткі екстрактори

Означення: Нечітким екстрактором називаються два ймовірнісні алгоритми Gen, Rep з наступними властивостями:

- Генеруюча функція $Gen(x) \to (R, P)$, на вхідних даних $x \in \mathcal{M}$, повертає рядок $R \in \{0,1\}^l$ і допоміжні дані $P \in \{0,1\}^*$
- Функція відтворення вхідних даних Rep, на вхідному елементі $\omega_1 \in \mathcal{M}$, і допоміжних даних $P \in \{0,1\}^*$, повертає рядок R такий що:
  $\text{Rep}(\omega_1, P) \to R$ якщо $dis(x, \omega_1) \leq t$

1.3.6 Криптографічна стійкість нечітких екстракторів

Означення: Нечіткий екстрактор називається $(\mathcal{M}, m, l, t, \epsilon)$-безпечним, якщо

- $\forall$ W - розподілу вхідних даних над метричним простором $\mathcal{M}$, з мінімальною ентропією m, вихідні дані (за виключенням допоміжних), розподілені згідно з розподілом R, і виконуються умови:
  - Статистична відстань між R та U_l - рівномірним розподілом розмірності l, є незначною навіть якщо допоміжні дані P є доступними зловмиснику. Тобто, $\mathbf{SD}((R, P), (U_l, P)) \leq \epsilon$, де ϵ є малим числом.

1.3.7 Модель Face recognition

Source-код моделі: https://github.com/ageitgey/face_recognition

1. Модель Face recognition є бібліотекою з відкритим кодом, ліцензованою згідно з MIT License
2. Модель побудована на основі deep face-recognition model, реалізованій у бібліотеці http://dlib.net/
3. Більш детально розглянуто метод `face_recognition.api.face_distance`, для отримання формату представлення face-вектору v: $v \in \mathbb{F}_2^{128}; d(v_1, v_2) = ||v_2 - v_1||, \mathbb{F}_2 = \{0,1\}$
4. Відповідно до документації бібліотеки `dlib` та методу `face_recognition.api.face_distance`, для отримання статистики, яка визначає подібність обличь за їх face-вектором, автор використовує базову Евклідову метрику у просторі розмірності 128.

1.4 Опис середовища розробки

Environment info: - Platform: `Docker`
- Container image: `python:3.10`
- SSH port: `2222`
- IDE: `PyCharm 2023.1 RC 2`
- Base: `Windows 11 (WSL2)`

dependencies (direct):
- itertools - numpy=1.24.2
- opencv-python=4.7.0.72
- face_recognition
- Pillow=9.4.0
- PIL
- hashlib

dependencies (transitive, manually installed) - ffmpeg - CMake - dlib - libsm6 - libxext6 - libsasl2-dev - python-dev - libldap2-dev - libssl-dev - openssh-server

1.5 Перелік класів та модулів проекту

python Modules:
- extractors.py: Реалізація нечіткого екстрактора
- cache.py: Кешування обєктів, пришвидшення повторного виконання тестів
- logger.py
- main.py
- testing.py: додаткові класи для тестування нечіткого екстрактора - test_face_recognition.py: unit-тести окремих класів та модулів

python Classes:
- Cache - кешування обєктів
- FrameIterator - ітерування зображень у відео-файлах
- FaceVectorExtractor - допоміжні функції для отримання face-вектору за зображенням
- FuzzyExtractorFaceRecognition - клас нечіткого екстрактора.
- LogFormatter - форматування виводу (stdin/log file)
- TestFaceVectorExtractor - основний тестовий клас
- TestCases - допоміжний клас для проведення тестування

1.6 Опис структури нечіткого екстрактора

- Input data preprocessing:
  1. Перетворення вхідних даних до вигляду: `list[PIL.Image.Image]`
  2. Перетворення: `PIL.Image.Image` $\overset{convert:RGB}{\longrightarrow}$ `np.ndarray`
  3. Визначення face-вектору для кожного з зображень. Зображення для яких не було знайдено жодного face-вектору, або було знайдено декілька face-векторів, далі не обробляються.
  4. Повернення послідовності face-векторів у вигляді `np.ndarray[np.ndarray]`

- Face vector preprocessing (Input data: V - `np.ndarray[np.ndarray]`):
  1. Обчислення $S = \{(\#|v_1 - v_2|, v_1, v_2)|v_1, v_2 \in V, v_1 \neq v_2\}$ (1)
  2. Обчислення викидів вибірки S (S^*) за значеннями першої координати:
     – Нехай μ - середнє значення вибірки за першою координатою: $\mu = \#\{v[0], v \in S\}$, σ - стандартне відхилення вибірки за першою координатою: $\sigma = \sqrt{\#\{(v[0] - \mu)^2, v \in S\}}$
     σ_0 - максимальне стандартне відхилення, параметр
     – $v \in S$ будемо вважати викидом, якщо для v виконується: $|\mu - v[0]| > \sigma_0$
  3. Створення вибірки S_1 за елементами вибірки $H = S \setminus S^*$:
     – $S_1 = \{(|\{s|v = s[1] \vee v = s[2], s \in H\}|, v)|v \in \{s[1], s[2]|s \in H\}\}$
  4. Знаходження викидів вибірки S_1: S_1^*
  5. Повернення $\{v[1]|v \in S_1 \setminus S_1^*\}$ (тип даних: `np.ndarray[np.ndarray]`)

  – Передумови:
    1. Для обчислення методу (1), необхідною передумовою є взаємна незалежність координат face-вектору. Оскільки face-vector належить простору $(-1, 1)^{128} \subset \mathbb{Q}^{128}$, з метрикою Евкліда, можна зробити висновок, що кожна координата має однаковий вплив на ідентифікацію обличчя. Також, відповідно до проведених тестів (dlib) не було знайдено кореляції між координатами face-вектору, що дозволяє зробити припущення про їх незалежність.

  – Зауваження
    2. Для більш точної побудови ключа, параметр σ_0 може бути зменшений (значення за замовчуванням: 0.7). Дана змінна є одним з параметрів моделі нечіткого екстрактора.

- Primary hash: (Input data: `np.ndarray[np.ndarray]`)
  1. Нехай простір $M \subset (-1, 1)^{128} \subset \mathbb{Q}^{128}$ розбито на множини:
     (1) $M_{i_1,i_2,...,i_{128}}$: $\bigcup\limits_{(i_1,..,i_{128}) \in I} M_{i_1,i_2,...,i_{128}} = M$ (I - множина індексів) за правилом:
     (2) $M_{i_1,i_2,...,i_{128}} = (i_1 d, (i_1 + 1)d) \times (i_2 d, (i_2 + 1)d) \times ... \times (i_{128} d, (i_{128} - 1)d)$ - (гіперкуб зі стороною d; d - параметр моделі нечіткого екстрактора).

     Очевидно, для покриття простору (2), виконується (1), якщо M не містить граней гіперкубів $M_{i_1,i_2,...,i_{128}}$. У реалізованому екстракторі, дане твердження є передумовою прийняття вхідних даних:

     > Якщо принаймні одна координата face-вектору знаходиться на перетині граней двох або більшої кількості гіперкубів, даний face-вектор не може бути оброблені нечітким екстрактором, тому вхідні дані будуть відхилені. (На практиці, ймовірність такої події дуже мала, під час тестування вхідні дані не були відхилені жодного разу).

Параметр $d \in (0,1)$, є параметром нечіткого екстрактора, який впливає на рівень подібності face-векторів.

2. Нехай std_max, α - параметри нечіткого екстрактора. Нехай $\overline{v}, v$

– вектори розмірності 128, $\overline{v}$ - містить середні значення, v - стандартні відхилення face-векторів по кожній координаті.

* Якщо для вхідних даних виконується: $\#\{v[i] > std\_max | i = \overline{1, \dim v}\} > \alpha \dim(v)$, модель нечіткого екстрактора відхилить вхідні дані, оскільки для обраних параметрів стандартне відхилення деякої множини координат перевищує задане максимально допустиме значення.

3. До даних (V) застосовано наступне перетворення:
   – Нехай $v \in V, v \in M_{I_0}, I_0 \in I, I_0 = (i_1, i_2, ..., i_{128}),\ f : V \to M$
   – Побудуємо функцію f:
   $f(v) : v \longrightarrow (i_1 + \frac{d}{2}, i_2 + \frac{d}{2}, .., i_{128} + \frac{d}{2})$
   Таким чином, кожен face-вектор, який належить гіперкубу M_{I_0}, буде відображено у центр M_{I_0}
4. Алгоритм повертає представлення значення $f(\overline{v})$ у вигляді послідовності байтів `bytes`

– Зауваження:

* Покриття простору підмножинами у вигляді гіперкубів було обрано на перевагу розбиттю простору на сфери, оскільки реалізація даного розбиття є досить простою для просторів розмірності більшої за 3, а обчислювана складність задачі є найменшою серед усіх можливих розбиттів: $O(1)$. Задача ефективного покриття простору розмірності 128 на сфери, на даний момент не була розв'язана. Тому не існує ефективного алгоритму за яким було б можливо побудувати таке розбиття.
* За рахунок розбиття (1), вектори $v_1, v_2 \in V$, однаково віддалені від центру M_{I_0}: c_0, такі що $|v_1 - c_0| \le d, |v_2 - c_0| \le d)$, можуть бути відображені у центри різних елементів розбиття, що вважатиметься помилкою обчислень. Для виправлення даної помилки реалізовано метод, опис якого наведено нижче.
* Primary hash не є стійким до взяття прообразу
* Primary hash не є стійким до колізій
* Primary hash не є стійким до взяття другого прообразу

- Primary hash error correction, Input data: `np.ndarray[np.ndarray]`
  1. Виконання N незалежних тестів `hash_primary`, побудова вибірки H яка містить N хеш значень.
  2. Обчислення ймовірностей p_h появи значень $h \in \text{set}(H)$
  3. Нехай max_unique_hashes - параметр моделі нечіткого екстрактора, який визначає максимально можливу кількість унікальних хеш-значень, допустиму у вибірці H
  4. Відповідно, якщо вхідні дані задовільняють нерівність: $|set(H)| > max\_unique\_hashes$: алгоритм відхиляє вхідну послідовність face-векторів. Дана нерівність справджується, якщо після обробки face-векторів та аналізу вибірки на предмет викидів, оцінка хеш-значення є недостатньо ефективною. (за замовчуванням, $max\_unique\_hashes = 1000$)
  5. Нехай $p\_a\_min \in (0,1)$ - параметр моделі нечіткого екстрактора
  6. Якщо $\exists h \in H : p_h > p\_a\_min$, результатом роботи алгоритму є значення h
  7. Інакше, знайдемо значення $h, l \in H$, які мають найбільшу ймовірність появи p_h, p_l.
  8. Визначимо значення ентропії Шенона (над алфавітом, який містить всі можливі

байти), послідовностей $h, l \in H$: e_h, e_l

9. Результатом роботи алгоритму є вектор з найбільшим значенням ентропії. Якщо $e_h = e_l$, повертається послідовність випадкова послідовність (l або h).

- Secondary hash
  - Перетворення вихідних даних алгоритму primary hash та додаткових перевірочних символів, отримання безпечного криптографічного ключа.

1. Нехай $check\_symbols\_count$ - параметр нечіткого екстрактора, який відповідає за кількість перевірочних символів, які будуть згенеровані для вихідних даних $hash\_primary$ використовуючи код Ріда-Соломона. (за замовчуванням, $check\_symbols\_count = 32$, тобто максимально дозволена похибка обчислення $hash\_primary$: 16 координат face-вектору)
2. Вхідними даними алгоритму $hash\_secondary$ є первинне $hash\_primary$, та послідовність перевірочних символів $check\_symbols$.
3. Обчислюється рядок $hash\_primary + check\_symbols$, помилки рядка виправляються за допомогою стандартного коду Ріда-Соломона. Якщо кількість помилок перевищує $\frac{check\_symbols\_count}{2}$, код Ріда-Соломона не декодує вхідні дані, тому під час виникнення помилки дані будуть відхилені (користувач не пройшов аутентифікацію).
4. Результатом роботи методу hash_secondary, є послідовність байтів, яка може бути безпечно використана для ініціалізації/шифрування/створення цифрового підпису користувача та інших криптографічних алгоритмів. Метод `hash_secondary` додатково містить аргумент `salt` (сіль, яка буде використана під час створення безпечного хеш-значення SHA256), збереження послідовностей `salt` та `check_symbols` передбачено у окремій базі даних без необхідності додаткового шифрування.

  - Властивості:
    - `hash_secondary` є стійким до колізій
    - `hash_secondary` є стійким до взяття прообразу
    - `hash_secondary` є стійким до взяття другого прообразу
  - Зауваження:
    - Алгоритм `hash_secondary` генерує криптографічно безпечний ключ, що дозволяє уникнути потенційних вразливостей, пов'язаних з отриманням зловмисником доступу до ключа: за значенням ключа, згенерованого `hash_secondary`, не існує ефективного алгоритму знаходження face-вектору користувача.
    - Використання солі (`salt`) під час хешування $hash\_primary$ рекомендується, оскільки це дозволяє запобігти вразливостям типу Rainbow table attack

1.6.1 Аналіз криптографічної безпеки побудованого нечіткого екстрактора.

Надалі будемо вважати, що значення face-векторів, згенерованих бібліотекою `dlib`, мають рівномірний розподіл $U(0,1)^{128}$

1. Аналіз статистичної відстані між розподілом ключів, які генеруються нечітким екстрактором, та рівномірним розподілом U:
   - Ключові моменти:
     1. Відображення простору face-векторів у методі `hash_primary`
     2. Виправлення помилок у вихідному рядку за допомогою кодів Ріда-Соломона.
     3. Хешування вихідних даних (SHA256+salt)
   - Аналіз відображення, реалізованого у методі `hash_primary`:
     1. Вхідними даними є вектори розмірності 128 з рівномірного росподілу $U(0,1)^{128}$
     2. Кожен вектор відображається у центр деякого гіперкубу з довжиною ребра d. Відображення переводить однакову кількість векторів до кожного з гіперкубів розбиття, тому у вихідних даних зберігається рівномірний розподіл даних. Статистична відстань $\mathrm{SD}(A_1, A_2) = \frac{1}{2}\sum\limits_{u}|Pr(A_1 = u) - Pr(A_2 = u)|$, рівна $\frac{1}{2}\frac{1}{d^{128}}|0 - d^{128}| = \frac{1}{2}$. Отже, розподіл вихідних даних не дорівнює $U(0,1)^{128}$, але низьке значення статистичної відстані вказує на значну подібність даних розподілів. Тому, відповідно до
     аналізу статистичної відстані, відображення побудовано коректно (дискретизація $U(0,1)^{128}$).
   - Аналіз перетворення коду Ріда Соломона
     1. Вхідними даними є значення `hash_primary` з рівномірного росподілу, отриманого дискретизацією $U(0,1)^{128}$, та послідовність перевірочних символів s. Оскільки значення для яких були згенеровані перевірочні символи є нормально розподіленими, враховуючи властивості БЧХ-кодів, будемо вважати що послідовність s також є рівномірно розподіленою величиною.
     2. Тоді для кожної пари (x, s), код Ріда-Соломона виправляє фіксовану кількість помилок, або відхиляє вхідні дані. Розглядаючи простір вхідних рядків з метрикою L - відстань Левенштейна, приходимо до висновку, що $\forall x \in \mathcal{M}, \forall s \in \{0,1\}^*$ – фіксованого: Код Ріда-Соломона перетворює однакову кількість рядків x у рядок, якому відповідає послідовність перевірочних символів (рядки, у яких значення метрики Левенштейна менше ніж задане фіксоване значення m, причому кількість перевірочних символів $|s| \geq 2m$). Тому при перетворенні вхідних даних за допомогою кода Ріда-Соломона, рівномірна розподіленість зберігається.
   - Хешування за допомогою SHA256 + salt.
     1. Вважатимемо, що послідовність salt є рівномірно розподіленою величиною.
     2. За означенням, SHA256 є криптографічно безпечною хеш функцією, з чого випливає наступна властивість:
     SHA256 перетворює вхідні рядки, які є рівномірно розподіленими, у послідовність хеш-символів, яка також є рівномірно розподіленою величиною.
     Тому, перетворення хешування (SHA256 + salt), зберігає рівномірний розподіл вхідних даних, за у мови що послідовність salt є рівномірно розподіленою величиною.
   - $\Longrightarrow$ Отже, за результатом аналізу статистичної відстані розподілу ключів, можемо зробити висновок про відсутність вразливостей пов'язаних з нерівномірною ймовір-

ністю отримання деяких ключів. Тому нечіткий екстрактор (з точки зору розподілу ключів), є криптографічно безпечним.

2. Аналіз на криптографічну безпеку з точки зору хеш-функцій.
   - Вимоги до будь-якого криптографічно стійкого нечіткого екстрактора, включають деякі властивості криптографічно безпечних хеш-функцій:
     1. Стійкість до взяття першого прообразу
     2. Стійкість до колізій
     3. Стійкість до взяття другого прообразу
   - Стійкість до взяття першого прообразу:
     Дана властивість випливає з криптографічної безпеки функції SHA256, оскільки вхідні дані
     даного алгоритму є рівномірно розподіленими.
   - Стійкість до колізій:
     Для алгоритму нечіткого екстрактора можна стверджувати про стійкість до колізій елементів, які не є
     подібними у метричному просторі $\mathcal{M}$ (подібними є елементи, для яких значення метрики
     простору $\mathcal{M}$ менше заданого порогового значення d, яке є точністю нечіткого екстрактора)
   - Стійкість до взяття другого прообразу:
     Алгоритм є стійким до взяття другого прообразу $\forall$ вхідних рядків $x, x_1 \in \mathcal{M}$, які не є подібними. Тобто, не існує ефективного алгоритму отримання рядку $x_1 \not\approx x$, для якого значення ключа є рівне значенню ключа вхідних даних x

3. Аналіз безпеки ескізу нечіткого екстрактора:
   - Нехай задано Код Ріда-Соломона з параметрами:
     - n - кількість символів рядка, що кодується, причому $n = 2^m - 1$
     - t - кількість перевірочних символів
   - Нехай $\mathscr{U} = GF(2^m)^*$ - скінченне поле порядку m
   - Нехай $SDif(\mathscr{U})$ - метричний простір з метрикою, яка повертає кількість елементів у симетричній різниці множин $s, s_1 \in \mathscr{U}$ на всіх підмножинах множини $\mathscr{U}$
   - Тоді Код Ріда-Соломона, як часний випадок БЧХ-кодів, є
     в середньому $(SDif(\mathscr{U}), m, m - t\log n + 1, t)$ - безпечним ескізом. (Theorem 6.3 для PinSketch з ресурсу [1])
   - Згідно з теоремою, даний безпечний ескіз має більший рівень ентропії, і, відповідно, є більш безпечним для меншої кількості перевірочних символів t. У проекті за замовчуванням використано 32 перевірочні символи, що складає 6.25% від розміру вхідних даних алгоритму.

1.6.2 FuzzyExtractor: Вимоги до вхідних даних

Для корректної побудови ключа нечітким екстрактором, необхідними є також початкові умови до вхідних даних:

- Повна видимість та гарна освітленість обличчя
- Відсутність інших людей на зображеннях
- Бажано монотонний фон та відсутність кольорового освітлення

1.6.3 Аналіз ефективності побудованої моделі

- Часова складність реалізованого алгоритму (worst-case): $O(n + n^2 + nA + B + C)$, де
  - n - кількість вхідних зображень
  - A - часова складність алгоритму отримання face-вектору бібліотеки dlib.
  - B - часова складність алгоритму SHA512
  - C - часова складність роботи коду Ріда-Соломона
- Просторова складність нечіткого екстрактора: O(n)
  - n - кількість вхідних зображень
- Середній час створення ключа на даних, які містять 30 зображень:
  - A: 0.3 − 1.1 (s) в залежності від розміру зображення
  - B: 0.1 (ms)
  - C: 0.1 (ms)
  - Алгоритм нечіткого екстрактора: 10 − 33 (s)
- Середній час перевірки ключа на даних, які містять 5 зображень: 1.6 − 5.6 (s)
- Час виконання було протестовано на docker-контейнері, без використання GPU ресурсів.
  - cpu: `AMD Ryzen 5/5600H`
- Параметри моделі нечіткого екстрактора:
  1. $\sigma_0 \in \mathbb{R}^+ \setminus \{0\} = 0.7$
  2. $d \in (0, 1) = 0.055$
  3. $max\_unique\_hashes \in \mathbb{N} \cup \{-1\}$ = -1
  4. $p\_a\_min \in (0, 1) = 0.6$
  5. $check\_symbols\_count \in \mathbb{N} = 32$
  6. $n\_tests \in \mathbb{N} = 250$
  7. $sample\_size \in (0, 1] = 0.7$
  8. $min\_images \in \mathbb{N} = 5$
  9. $\alpha \in (0, 1) = 0.5$

1.6.4 Приклади використання

- Користувач бажає закодувати файл за допомогою алгоритму AES.
  Ключ для алгоритму генерується за допомогою класу FuzzyExtractor. Оскільки секретний ключ базується на біометричних даних, для декодування файлу потрібно пройти тест розпізнавання обличчя.
- Користувач бажає створити пару ключів для алгоритму ECDSA для підписання документа.
  Зі сторони сетрифікаційного агентства використовується клас FuzzyExtractor.
  Тоді можна створити пару ключів на основі face-вектору користувача (та послідовності salt). Причому, якщо користувач бажає підписати інший документ (без наявності закритого та відкритого ключів), це буде набагато простіше, оскільки користувачеві потрібно буде пройти тест розпізнавання обличчя для отримання ключа сертифікату.
- Автентифікація користувачів на деякому web-ресурсі. Server-side web-ресурсу зберігає послідовності salt та $check\_symbols$, які є доступними за запитом всім користувачам. Клієнт проходить біометричну аутентифікацію, і отримує ключ, який є спільним секретом клієнта і сервера.

1.6.5 Імітаційні експерименти

- Теоретичний розподіл вхідних даних: Теоретичний розподіл зображень є рівномірним за значенням face-вектору. Теоретичний розподіл послідовності salt є рівномірним за значенням кожного регістру.
- Алгоритми, які використовувалися для обробки модельованих даних:
  1. Face bounding box (`dlib` implementation)
  2. Face image to face vector mapping (`face-recignition` package)
- Кількість повторних моделювань для кожного експерименту: порядку 30 моделювань для кожного набору вхідних даних.
- Узагальнені характеристики якості, які підраховувались:
  - Неможливість візуального порівняння вихідних даних ідентичного користувача.
  - Значна статистична відстань розподілу ключів від рівномірного для корректного набору вхідних даних.
  - Можливість створення та відтворення секретного ключа з використанням солі (salt)
  - Можливість відновлення ключа за різними даними ідентичного користувача.
  - Неможливість відновлення ключа одного користувача з використанням даних іншого користувача
  - Значення ентропії для послідовності ключа та перевірочних символів
  - Неідентичність вихідних даних алгоритму для різних вхідних даних ідентичного користувача

Більшість тестових випадків реалізовано у репозиторії за посиланням: https://github.com/al3xkras/fuzzy-extractors

1.6.6 Приклад допоміжних значень, які генеруються в процесі роботи алгоритму

Розподіл хеш-значень, згенерованих алгоритмом *Primary hash* (500 випробувань):

- Для вхідних даних, які задовольняють вимогам (top-5 by frequency):
- Для вхідних даних, які не задовольняють вимогам (top-5 by frequency):

Згідно з емпіричними тестами, у розподілі даних, які задовольняють вимогам деякі елементи спостерігаються із значно більшою ймовірністю. Якщо дані не задовольняють необхідним вимогам, розподіл ключів є ближчим до рівномірного розподілу, для подібних даних не було реалізовано алгоритм створення надійного ключа.

Приклади пар $(\text{key}, check\_symbols)$, які згенеровано з використанням нечіткого екстрактора:
1.

2.

3.

Приклади 1. та 3. були згенерованим для однієї людини, у прикладі 2. було використано біометричні дані іншого користувача. Визначення належності ключа та перевірочних символів конкретному користувачу є візуально неможливим.

Приклади шифрування з використанням солі (SHA512+salt):

1.

2.

1.6.7 Висновки

У роботі було реалізовано криптографічно безпечний, ефективний алгоритм нечіткого екстрактора.
Алгоритм генерує приватний ключ базуючись на послідовності зображень або відеофайлу які відповідають необхідним вимогам. Результатом роботи є послідовності ключа та перевірочних символів, які для коректно обраних параметрів моделі є криптографічно безпечними, і можуть бути використані для ініціалізації інших криптографічних алгоритмів.
Було показано, що послідовність перевірочних символів є принаймні $(SDif(\mathscr{U}), m, m - t\log n + 1, t)$-безпечною.
Значна кількість enterprise-моделей нечітких екстракторів, використовує коди Ріда-Соломона з малою кількістю перевірочних символів для створення безпечного ескізу, що також було враховано у роботі, при визначенні алгоритму та параметрів моделі за замовчуванням.
Було протестовано коректність роботи моделі на біометричних даних ідентичної людини, які отримувалися впродовж 2-х місяців (всього 4 тести, які завершилися успішним відновленням ключа).
Позитивними рисами реалізованої моделі нечіткого екстрактора є значна гнучкість в обранні параметрів, що дозволяє налаштувати нечіткий екстрактор в залежності від потреб, висока ентропія і можливість повторного створення неідентичного ключа, який авторизує користувача, у випадку якщо минулий ключ був опублікований, і можливість використання послідовності солі для значного ускладнення ряду вразливостей таких як rainbow table attack, які потенційно дозволяють отримати біометричні дані користувача.
Негативними рисами алгоритму є значна обчислювальна складність створення ключа зі сторони серверу, що може бути використано зловмисниками для проведення DoS атак. Тому, використання алгоритму для ініціалізації web-протоколів є неефективним і потенційно вразливим.
Натомість, алгоритм нечіткого екстрактора можна використовувати як додатковий рівень захисту користувача для дій, які є виключно конфіденційними (наприклад, створення електронного підпису документу).
Для ініціалізації ключа зі сторони серверу, необхідною є персональна присутність користувача, або надання його даних з довірених ресурсів. Також користувач може створити ключ локально для ряду задач (наприклад, аутентифікація у локальній мережі або шифрування файлів). Слід зауважити, що алгоритм базується на бібліотеці dlib, тому коректна робота моделі частково залежить від алгоритмів, які були реалізовані у даній бібліотеці.

1.7 Список використаних першоджерел:

- (1) http://web.cs.ucla.edu/~rafail/PUBLIC/89.pdf
  Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data (Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, Adam Smith. January 20, 2008)
- (2) https://ro.uow.edu.au/cgi/viewcontent.cgi?article=1698&context=eispapers1
  LI, N., Guo, F., Mu, Y., Susilo, W. & Nepal, S. (2017). Fuzzy Extractors for Biometric Identification. 37th IEEE
  Internaitonal Conference on Distributed Computing Systems (ICDCS 2017) (pp. 667-677). United States: IEEE.
- (3) https://www.cs.bu.edu/~reyzin/papers/fuzzysurvey.pdf
  Fuzzy Extractors. A Brief Survey of Results from 2004 to 2006

(Yevgeniy Dodis, Leonid Reyzin, Adam Smith)

- (4) https://faculty.math.illinois.edu/~duursma/CT/RS-1960.pdf
  POLYNOMIAL CODES OVER CERTAIN FINITE FIELDS* I. S. REED AND G. SOLOMON
- (5) https://www.arijuels.com/wp-content/uploads/2013/09/JS02.pdf
  A Fuzzy Vault Scheme (Ari Juels and Madhu Sudan) RSA Laboratories, Bedford, MA 01730, USA
- (6) https://digital.csic.es/bitstream/10261/15966/1/SAM3262.pdf
  Biometric Fuzzy Extractor Scheme for Iris Templates
  F. Hernandez Alvarez, L. Hernandez Encinas, C. Sanchez Avila
  Departamento Matematica Aplicada a las Tecnologıas de la Informacion, E.T.S.I.T.
  Universidad Politecnica de Madrid. Spain.
- (7) https://nure.ua/wp-content/uploads/2018/Scientific_editions/are_2018_24.pdf
  ПОРІВНЯЛЬНИЙ АНАЛІЗ БІОМЕТРИЧНИХ КРИПТОСИСТЕМ
  М. С. ЛУЦЕНКО, О. О. КУЗНЕЦОВ, Д. І. ПРОКОПОВИЧ-ТКАЧЕНКО, В. П. ЗВЄРЄВ, А. О. УВАРОВА