НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт з виконання комп'ютерного практикуму №1 "Побудова атак на геш-функції"

Виконала:
студентка групи ФІ-13
Балацька Вікторія

Викладач:
Яковлєв С. В

Київ — 2024

Мета: Дослідити криптографічні властивості геш-функцій, засвоїти еталонні оцінки стійкості геш-функцій, перевірити на практиці теоретичні положення.

Постановка завдання: Основне завдання практикуму: провести атаку випадкового пошуку прообразу та атаку днів народження на задану геш-функція(у моєму випадку SHA-512), експериментально оцінити складності даних атак.

Порядок виконання роботи

0. Ознайомилась із порядком виконання комп'ютерного практикуму та відповідними вимогами до його виконання.

Для виконання даного комп'ютерного практикуму було написано програму на мові програмування Python. Варіант мого завдання потребув побудови атак на геш-функцію SHA-512. Дана функція повертає фіктований 512-бітний (64-байтовий) геш.

1. Проведення атак пошуку прообразу на геш-функцію.

1.1 Перед початком побудови атак, мною було сформовано вхідне повідомлення, яке містить мої ПІБ: `BalatskaViktoriaVitaliivna`.

1.2 Для першого варіанту атаки пошуку прообразу до початкового повідомлення послідовно додавалися натуральні числа від 1 до N, поки значення 16-ти бітів гешу вхідного повідомлення не зійшлось із значенням 16-ти бітів гешу повідомлення$\{N\}$.

1.3 Для другого варіанту атаки на кожній ітерації у вхідне повідомлення вносились випакові модифікації, а вхідне повідомлення залишалось не змінним. Цей процес тривав доти, доки не було знайдено випадкову модифікацію вхідного повідомлення, у якій останні 16 бітів гешу не зійшлись із 16-ма бітами гешу початкового повідомлення.

1.4 Функції реалізації атак пошуку прообразу

Для реалізації першого та другого варіантів атаки пошуку прообразу було сторено функції `preimage_attack` та `write_preimage`. Функція `preimage_attack` відповідає за реалізацію атаки пошуку прообразу на геш-функцію SHA-512. Дана функція приймає наступні параметри:

- `alphabet`: алфавіт символів, що використовується для модифікації повідомлення при другому типі атаки;
- `start_message`: вхідне повідомлення, створене у пункті 1.1;
- `output_file`: файл, у який записуються результати атаки;
- `attack`: тип атаки, що застосовується (за замовчування 1);
- `logs`: прапорець для запису результатів у файл (за замовчуванням True);
- `output`: прапорець для виведення результатів на екран (за замовчуванням True).

Опис процесу виконання функції:

1. Обчислюється геш SHA-512 вхідного повідомлення, після чого отримується його повне hex-значення та значення останніх двох байтів.
2. Запускається цикл, на початку якого щоразу генерується нове повідомлення. Залежно від типу атаки: до повідомлення додаються натуральні числа або вносяться випадкові модифікації. Цикл завершується тоді, коли останні 16 біт гешу вхідного повідомлення збігаються із 16-ма бітами гешу згенерованого циклом повідомлення.
3. Якщо прапорці `logs` та `output` мають значенн True, то результати записуютсья у файл та виводяться на екран відповідно.
4. Функція повертає кількість ітерацій, які необхідні були для успішної атаки пошуку прообразу.

Код реалізації функції:

```
def preimage_attack(alphabet, start_message, output_file, attack=1,
    logs=True, output=True):
    pra_char = 124  # 128 - 4
    pra_bytes = 62  # 64 - 2
    msg_len = len(start_message)
```

```
    with open(output_file, 'w') as log_file:
        preimage_hash = hl.sha512(start_message.encode())
        preimage_hash_bytes = preimage_hash.digest()[pra_bytes:]
        preimage_hash_hex = preimage_hash.hexdigest()

        if logs:
            log_file.write(f"{start_message:{msg_len}s}: "
                           f"{preimage_hash_hex[:pra_char]}\t{
                              preimage_hash_hex[pra_char:]}\n")
        if output:
            print(f"Preimage attack {2 if attack == 2 else 1}\nBase
               message is: {start_message}\n"
                  f"{start_message:{msg_len}s}: {preimage_hash_hex[:
                     pra_char]}\t{preimage_hash_hex[pra_char:]}")

        iteration = 0
        for i in range(1, sys.maxsize):
            msg = modify_message(start_message, alphabet) if attack ==
               2 else f"{start_message}{i}"
            msg_hash = hl.sha512(msg.encode())
            msg_hash_hex = msg_hash.hexdigest()

            if logs:
                log_file.write(f"{msg:{msg_len}s}: {msg_hash_hex[:
                   pra_char]}\t{msg_hash_hex[pra_char:]}")
            if msg_hash.digest()[pra_bytes:] == preimage_hash_bytes:
                if attack == 2 and msg == start_message:
                    if logs:
                        log_file.write('\n')
                    continue
                if output:
                    write_preimage(log_file, i, msg, msg_hash_hex,
                       msg_len, pra_char)
                iteration = i
                break
            else:
                if logs:
                    log_file.write('\n')

    return iteration
```

Функція `write_preimage` записує повідомлення у файл та виводить відформатовані результати на екран.
Код реалізації функції:

```
def write_preimage(log_file, i, msg, msg_hash_hex, msg_len, pra_char):
    log_file.write(f"\nSecond preimage found on attempt {i}!")
    print(f"{msg:{msg_len}s}: {msg_hash_hex[:pra_char]}\t{msg_hash_hex[
       pra_char:]}\n"
          f"Second preimage found on attempt {i}!\n")
```

2. Проведення атак днів народжень на геш-функцію.

2.1 Перед початком побудови атак, мною було сформовано вхідне повідомлення, яке містить мої ПІБ та модифіковано за допомогою функції `modify_message` про яку я розкажу пізніше: `modify_message(BalatskaViktoriaVitaliivna, alphabet)`.

2.2 Для першого варіанту атаки днів народжень до початкового модифікованого повідомлення послідовно додавалися натуральні числа від 1 до N, поки серед підрахованих геш-значень не виникне колізія.

2.3 Для другого варіанту атаки на кожній ітерації у вхідне модифіковане повідомлення вносились випакові модифікації. Цей процес тривав доти, доки не було знайдено колізію серед підрахованих геш-значень.

Геш-значення для пошуку колізій усікалися до 32-х бітів.

1.4 Функції реалізації атак днів народжень

Для реалізації першого та другого варіантів атаки днів народжень було сторено функції `birthday_attack` та `write_collision`. Функція `birthday_attack` відповідає за реалізацію атаки днів народжень на геш-функцію SHA-512. Дана функція приймає наступні параметри:

- `alphabet`: алфавіт символів, що використовується для модифікації повідомлення при другому типі атаки;
- `start_message`: вхідне повідомлення, створене у пункті 2.1;
- `output_file`: файл, у який записуються результати атаки;
- `attack`: тип атаки, що застосовується (за замовчування 1);
- `logs`: прапорець для запису результатів у файл (за замовчуванням True);
- `output`: прапорець для виведення результатів на екран (за замовчуванням True).

Опис процесу виконання функції:

1. Обчислюється геш SHA-512 вхідного модифікованого повідомлення, після чого отримується його повне hex-значення та значення останніх чотирьох байтів.
2. Створюється словник, який зберігатиме останні чотири байти гешу для кожного повідомлення, яке генерується. Першим значенням цього словника є геш-значення вхідного повідомлення (останні чотири байти).
3. Запускається цикл, на початку якого щоразу генерується нове повідомлення. Залежно від типу атаки: до повідомлення додаються натуральні числа або вносяться випадкові модифікації. Обчислюється геш нового повідомлення, після чого отримується його повне hex-значення та значення останніх чотирьох байтів. Після цього перевіряється чи отримане байтове геш-значення є в створеному раніше словнику. Якщо воно вже є, то це означає що колізію знайдено. Якщо ж цього значення в словнику немає, то воно знаписується у словник. Цикл завершується тоді, коли колізія вважається знайденою.
4. Якщо прапорці `logs` та `output` мають значенн True, то результати записуютсья у файл та виводяться на екран відповідно.
5. Функція повертає кількість ітерацій, які необхідні були для того, щоб знайти колізію.

Код реалізації функції:

```
def birthday_attack(alphabet, start_message, output_file, attack=1,
    logs=True, output=True):
    bra_char = 120  # 128 - 8
    bra_bytes = 60  # 64 - 4
    msg_len = len(start_message)

    with open(output_file, 'w') as log_file:
        birthday_hash = hl.sha512(start_message.encode())
        birthday_hash_hex = birthday_hash.hexdigest()

        if logs:
            log_file.write(f"{start_message:{msg_len}s}: "
                           f"{birthday_hash_hex[:bra_char]}\t{
                               birthday_hash_hex[bra_char:]}\n")
        if output:
            print(f"Birthday attack {2 if attack == 2 else 1}\nBase
                message is: {start_message}\n"
```

```
                f"{start_message:{msg_len}s}: {birthday_hash_hex[:
                    bra_char]}\t{birthday_hash_hex[bra_char:]}")

        prev_dict = {birthday_hash.digest()[bra_bytes:]: start_message}
        iteration = 0
        for i in range(1, sys.maxsize):
            msg = modify_message(start_message, alphabet) if attack ==
                2 else f"{start_message}{i}"

            msg_hash = hl.sha512(msg.encode())
            msg_hash_hex = msg_hash.hexdigest()
            msg_hash_bytes = msg_hash.digest()[bra_bytes:]

            if msg_hash_bytes in prev_dict:
                collision = prev_dict[msg_hash_bytes]
                collision_hash = hl.sha512(collision.encode()).
                    hexdigest()
                if output:
                    write_collision(log_file, i, msg, collision,
                        msg_hash_hex, collision_hash, msg_len, bra_char)
                iteration = i
                break
            else:
                prev_dict[msg_hash_bytes] = msg
                if logs:
                    log_file.write(f"{msg:{msg_len}s}: {msg_hash_hex[:
                        bra_char]}\t{msg_hash_hex[bra_char:]}\n")

    return iteration
```

Функція `write_collision` записує повідомлення та його геш-значення, а також колізію у файл та виводить відформатовані результати на екран.
Код реалізації функції:

```
def write_collision(log_file, i, msg, col, msg_hash_hex, col_hash,
    msg_len, bra_char):
    log_file.write(f"\nMessage: {msg:{msg_len}s}: {msg_hash_hex[:
        bra_char]}\t{msg_hash_hex[bra_char:]}\n"
                   f"Collision: {col:{msg_len}s}: {col_hash[:bra_char
                       ]}\t{col_hash[bra_char:]}\n"
                   f"Collision found in {i} iterations!\n")
    print(f"Message: {msg:{msg_len}s}: {msg_hash_hex[:bra_char]}\t{
        msg_hash_hex[bra_char:]}\n"
          f"Collision: {col:{msg_len}s}: {col_hash[:bra_char]}\t{
              col_hash[bra_char:]}\n"
          f"Collision found in {i} iterations!\n")
```

3. Опис додаткових функцій, які знадобились при побудові атак.
Додатковими функціями можна вважати наступні функції:

- `modify_message`: вносить випадкові модифікації у повідомлення;
  Код реалізації функції:

```
def modify_message(message, alphabet, replacement_chance=0.2,
    addition_chance=0.2):
    modified_message = list(message)
    length = len(modified_message)

    for i in range(length):
        if random.random() < replacement_chance:
            modified_message[i] = random.choice(alphabet)
```

```
    i = 0
    while i < len(modified_message):
        if random.random() < addition_chance:
            modified_message.insert(i, random.choice(alphabet))

        i += 1

    return ''.join(modified_message)
```

- `multy_run_of_attack`: запускає обрану атаку визначену кількість разів;
  Код реалізації функції:

```
def multy_run_of_attack(output_file, attack, num_attack, alphabet,
    base_msg):
    stats: List[Tuple[str, int]] = []

    for _ in range(120):
        msg = modify_message(base_msg, alphabet)
        attack_result = birthday_attack(alphabet, msg, output_file,
            num_attack, False, False) \
            if attack == 2 else preimage_attack(alphabet, msg,
               output_file, num_attack, False, False)
        stats.append((msg, attack_result))

    return stats
```

- `calculation_static_values`: обчислює статистичні дані такі як кількість згенерованих повідомлень, середнє значення, дисперсію та довірчий інтервал;
  Код реалізації функції:

```
def calculation_static_values(stats, gamma=0.95):
    data = [count for _, count in stats]
    sum_itr = np.sum(data)
    mean_val = np.mean(data)
    variance = np.var(data)
    std_dev = np.sqrt(variance)

    q = 1 - ((1 - gamma) / 2)
    scale_t = sp.t.ppf(q, 120 - 1)
    confidence_interval = (mean_val - (scale_t * std_dev / np.sqrt
       (120)),
                           mean_val + (scale_t * std_dev / np.sqrt
                              (120)))

    print(f"Sum of iterations: {sum_itr}")
    print(f"Mean: {mean_val}")
    print(f"Variance: {variance} -> Standart deviance: {std_dev}")
    print(f"Confidence interval: {confidence_interval}")
```

- `excel_create`: створює таблицю із вхідним повідомленням кожної ітерації multiple-атаки та кількістю ітерації для її успішного завершення;
  Код реалізації функції:

```
def excel_create(data, file_name):
    wb = openpyxl.Workbook()
    ws = wb.active

    headers = ["Start message", "Iteration count"]
```

```
    ws.append(headers)
    for cell in ws[1]:
        cell.font = Font(bold=True)

    for start_message, iteration_count in data:
        ws.append([start_message, iteration_count])

    for column in ws.columns:
        max_length = 0
        column = [cell for cell in column]
        for cell in column:
            try:
                if len(str(cell.value)) > max_length:
                    max_length = len(str(cell.value))
            except:
                pass
        adjusted_width = (max_length + 2)
        ws.column_dimensions[column[0].column_letter].width =
            adjusted_width

    reports_dir = 'reports'
    os.makedirs(reports_dir, exist_ok=True)
    report_filepath = os.path.join(reports_dir, file_name)
    wb.save(report_filepath)
```

- `create_hist`: створює гістограми на основі отриманих даних атак.
  Код реалізації функції:

```
def create_hist(data_array, attack):
    plt.figure(figsize=(15, 6))
    counts, bins, _ = plt.hist(data_array, bins=20, edgecolor='
        black')

    plt.title(f"Iterations Distribution for Attack {attack}")
    plt.grid(True)
    plt.yticks(range(int(np.max(counts)) + 1))
    plt.xlim([np.min(data_array) - 1000, np.max(data_array) +
        1000])
    plt.xticks(bins, rotation=45)

    plt.savefig(f'hist{attack}.png')
    plt.show(block=False)
```

4. Опис головної функції.

Ця функція `main` є основною точкою входу для програми, яка виконує атаки на основі гешування (`preimage` та `birthday` атаки) на вхідному повідомленні `BalatskaViktoriaVitaliivna`. Функція дозволяє вибрати тип атаки, варіант її виконання та кількість запусків, зокрема, або одноразово, або багаторазово з подальшим статистичним аналізом. Фунція містьв собі дві основні змінні: базове повідомлення та алфавіт для модифікацій. Корстувачу пропонується обрати тип атаки: `preimage` чи `birthday`, її варіант: 1 чи 2 та запуск: `одноразовий` чи `багаторазовий`. Якщо обрані значення не відповідають 1 або 2, то програма завершує роботу.

Далі, у залежності від введених користувачем параметрів, запускається `preimage` або `birthday` атака з відповідним номером та відповідну кількість разів (1 чи 120). Також, залежно від обраної кількості запусків, користувач отримує текстовий файл з усіма повідомленнями та їх геш-значеннями для одноразової атаки або excel-файл з вхідними повідомленнями та кількістю ітерацій для кожної запущеної циклом атаки при багаторазовій атаці та гістограмою, яка відображає цю статистику.
Код реалізації функції:

```
def main():
    name_msg = "BalatskaViktoriaVitaliivna"
    alphabet = '
       abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
       !"#$%&\'()*+,-./:;<=>?@[]^_`{|}\\~'

    attack = int(input(f'Choose type of attack:\n\t1. Preimage attack\n
       \t2. Birthday attack\n'))
    num_of_attack = int(input(f'Choose number of attack:\n\t1. First
       number of attack\n\t2. Second number of attack\n'))
    multiple = int(input(f'Choose type of attack:\n\t1. Single attack\n
       \t2. Multiple attack\n'))

    if (attack != 1 and attack != 2) or (num_of_attack != 1 and
       num_of_attack != 2) or (
            multiple != 1 and multiple != 2):
        exit('Invalid input for some variant')

    if attack == 1:
        if multiple == 2:
            stats = multy_run_of_attack(
                f'preimage_attack_1.{num_of_attack}_stats.txt', 1,
                   num_of_attack, alphabet, name_msg)
            excel_create(stats, f'preimage_attack_1.{num_of_attack}
               _stats.xlsx')
            calculation_static_values(stats)
            create_hist([iteration[1] for iteration in stats], f'1.{
               num_of_attack}')
        else:
            preimage_attack(alphabet, name_msg, f'preimage_attack_{
               num_of_attack}.txt', num_of_attack)
    else:
        if multiple == 2:
            stats = multy_run_of_attack(
                f'birthday_attack_2.{num_of_attack}_stats.txt', 2,
                   num_of_attack, alphabet, name_msg)
            excel_create(stats, f'birthday_attack_2.{num_of_attack}
               _stats.xlsx')
            calculation_static_values(stats)
            create_hist([iteration[1] for iteration in stats], f'2.{
               num_of_attack}')
        else:
            birthday_attack(alphabet, modify_message(name_msg, alphabet
               ),
                            f'birthday_attack_{num_of_attack}.txt',
                               num_of_attack)
```

5. Результати.

1. Результати виконання одиночної атаки пошуку прообразу:

```
Choose type of attack:
    1. Preimage attack
    2. Birthday attack
1
Choose number of attack:
    1. First number of attack
    2. Second number of attack
1
Choose type of attack:
    1. Single attack
    2. Multiple attack
1
Preimage attack 1
Base message is: BalatskaViktoriaVitaliivna
BalatskaViktoriaVitaliivna: f4a041bdc3167197848fa2dad41c56fbb727f33733a9e1fb77a250b91501734a2fae22a9a4737bccca24ad0049c8560b22e2f5d5a00c81f085ca4d6698a7    98ec
BalatskaViktoriaVitaliivna14254: 61c5570859fcb00d272cbcb7b24d67316a9b4a8e11becb2da16ee73befd111464e84dd670ea7b33e09ae07a1bf3058c35a7e1059f80ea52b134cd80218c4   98ec
Second preimage found on attempt 14254!
```

Рис. 1: Атака 1.1

```
Choose type of attack:
    1. Preimage attack
    2. Birthday attack
1
Choose number of attack:
    1. First number of attack
    2. Second number of attack
2
Choose type of attack:
    1. Single attack
    2. Multiple attack
1
Preimage attack 2
Base message is: BalatskaViktoriaVitaliivna
BalatskaViktoriaVitaliivna: f4a041bdc3167197848fa2dad41c56fbb727f33733a9e1fb77a250b91501734a2fae22a9a4737bccca24ad0049c8560b22e2f5d5a00c81f085ca4d6698a7    98ec
Balat)kaViktoHriIVi9;"l{ivlnj: a55d86ea058b6d03b4f12289dbb6b3fafc2fe3af183a812cb68a2ab6e81428f8fb79615b5da20241d35a51810e55fe75f712803da7951ba14ff060cf2dab 98ec
Second preimage found on attempt 13529!
```

Рис. 2: Атака 1.2

Усі змодифіковані повідомлення та їх геш-значення можна побачити у файлах `preimage_attack_1.txt` та `preimage_attack_2.txt` (див. GitHub).

2. Результати виконання одиночної атаки днів народжень:

```
Choose type of attack:
    1. Preimage attack
    2. Birthday attack
2
Choose number of attack:
    1. First number of attack
    2. Second number of attack
1
Choose type of attack:
    1. Single attack
    2. Multiple attack
1
Birthday attack 1
Base message is: 4Ba~8tykaHVi4btoriaUi&xtaliivn!
4Ba~8tykaHVi4btoriaUi&xtaliivn!: 6ad2bd0757a335d6eb13ca8b4293d3db4ecd66f430a3f53295588b0a3666db8e67d0c6f8d57308d08797f53f42004bf71c66733d62486bc76ee933b8   a680c677
Message: 4Ba~8tykaHVi4btoriaUi&xtaliivn!219940: aaddb9e1f50e842e041b5e8cd60bd25245690a00c82aaaf82541d44ee61670873ea3381ce5036ec1cc2c4b61dc5dee654ae36dd0180acc508b44e228    45e1bd06
Collision: 4Ba~8tykaHVi4btoriaUi&xtaliivn!76932: 2bb4daf4ad8eea80acfc67d1c73dddf64707edf67c8e8519b9e134174ec7eb71146b7fa805148920380e6373ac7ffc78056e2620fba4a5ce1176f196   45e1bd06
Collision found in 219940 iterations!
```

Рис. 3: Атака 2.1

```
Choose type of attack:
    1. Preimage attack
    2. Birthday attack
2
Choose number of attack:
    1. First number of attack
    2. Second number of attack
2
Choose type of attack:
    1. Single attack
    2. Multiple attack
1
Birthday attack 2
Base message is: KBala3tskaViMktoriJaXitaliivn.a
KBala3tskaViMktoriJaXitaliivn.a: 383d2cccfbd1ac6ea2f565bd92d116dc46305986bd78533fc24a620cef60fb72a0fb7efad39de27b9643e21c841a041d374a52f19c30cc5cc4490995   8b7f9d61
Message: KBgS)a30tsWaV4wMktoriJaXiDtaliu,i,v2n~!a: 98e54170d671c30eaa78c2dd1016cef0b7559d022130a9092da3bd05e2252aed293bbc525651db40b4ba95df9d23718fb369cc9ad9e472a2a140bab6 43b13116
Collision: KBm1wa#la3-Lsxk6aV`M7tor8JaCiCV1"alMFivn.a: 244e5abfa464c261526c2bafbd48bdb303271c6c48095347347c77b62bc0be00ce0a3c154300319eb84e36408fd97c59e5399dff23fdb536d12adc36 43b13116
Collision found in 67822 iterations!
```

Рис. 4: Атака 2.2

Усі змодифіковані повідомлення та їх геш-значення можна побачити у файлах `birthday_attack_1.txt` та `birthday_attack_2.txt` (див. GitHub).

3. Результати виконання multiple атаки пошуку прообразу:

Початкові повідомлення та кількість ітерацій яка знадобились для пошуку прообразу можна знайти у excel-файлах `preimage_attack_1.1_stats` та `preimage_attack_1.2_stats`. Також на основі даних було побудовано гістограми та обраховано статистичні дані.

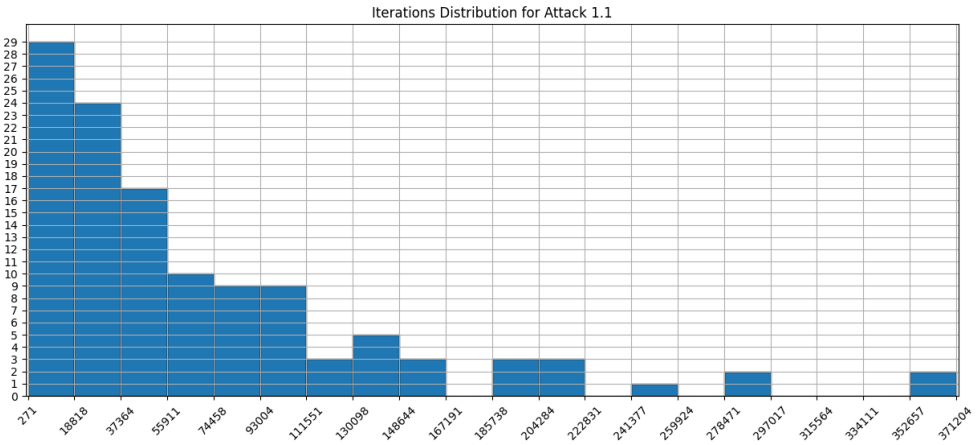

Рис. 5: Гістограма 1.1

```
Sum of iterations: 8352117
Mean: 69600.975
Variance: 5237931403.641042 -> Standart deviance: 72373.55458757737
Confidence interval: (56518.90776251718, 82683.04223748283)
```

Рис. 6: Статистичні дані 1.1

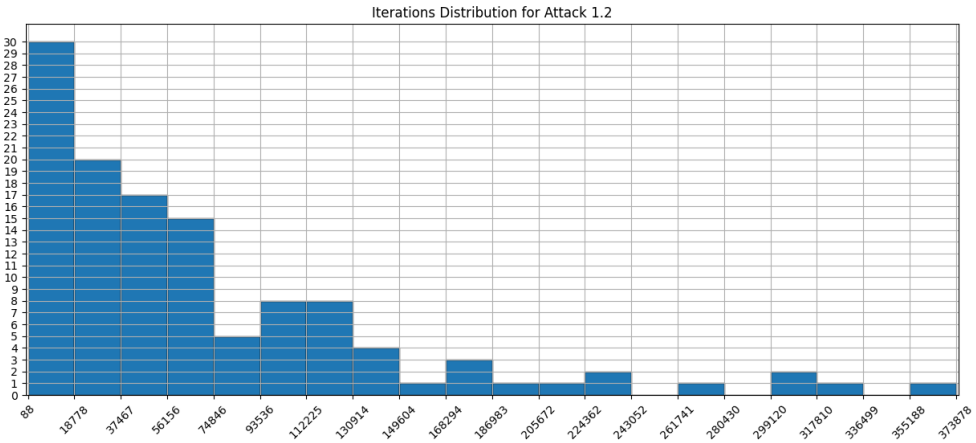

Рис. 7: Гістограма 1.2

```
Sum of iterations: 8541183
Mean: 71176.525
Variance: 5368899957.716041 -> Standart deviance: 73272.77773986763
Confidence interval: (57931.916378350004, 84421.13362164998)
```

Рис. 8: Статистичні дані 1.2

4. Результати виконання multiple атаки днів народжень:

Початкові повідомлення та кількість ітерацій яка знадобились для пошуку колізії можна знайти у excel-файлах `birthday_attack_1.1_stats` та `birthday_attack_1.2_stats`. Також на основі даних було побудовано гістограми та обраховано статистичні дані.

```
Sum of iterations: 10721801
Mean: 89348.34166666666
Variance: 1755522914.1749306 -> Standart deviance: 41898.960776789325
Confidence interval: (81774.7876912385, 96921.89564209482)
```

Рис. 9: Статистичні дані 2.1

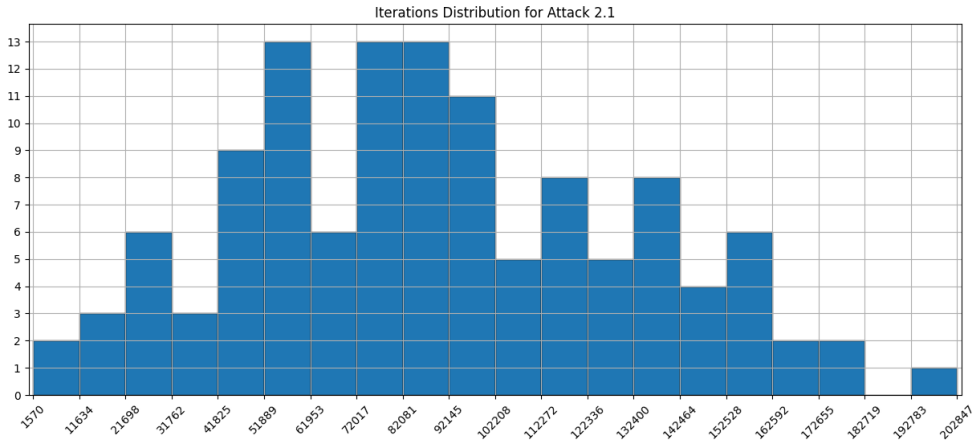

Рис. 10: Гістограма 2.1

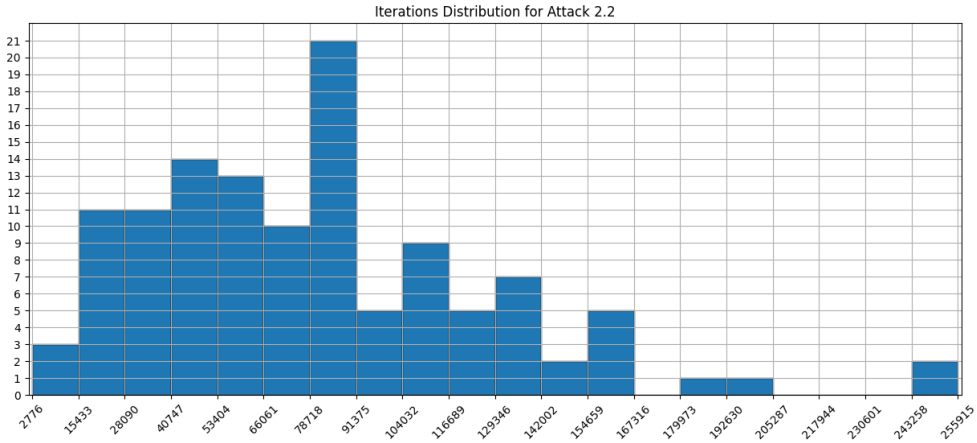

Рис. 11: Гістограма 2.2

```
Sum of iterations: 9613288
Mean: 80110.73333333334
Variance: 2193630061.828889 -> Standart deviance: 46836.20460529321
Confidence interval: (71644.73520340152, 88576.73146326515)
```

Рис. 12: Статистичні дані 2.2

6. Теоретичні оцінки складності атак.
6.1 Оцінка складності атаки пошуку прообразу:

1. Обчислення початкового гешу `preimage_hash = hl.sha512(start_message.encode())`: Це операція фіксованої складності $O(1)$, оскільки виконується один раз незалежно від ітерацій.

2. Основний цикл `for i in range(1, sys.maxsize)`:Цей цикл виконується до тих пір, поки не буде знайдено прообраз гешу (тобто повідомлення, яке відповідає частковому збігу з початковим повідомленням). У кожній ітерації ми генеруємо нове повідомлення `msg` і обчислюємо його геш `msg_hash`.У кожній ітерації функція генерує нове повідомлення шляхом простого додавання індексу і до початкового повідомлення `msg = f"{start_message}{i}"`, що має постійну складність $O(1)$. Для цієї атаки прообраз буде знайдено в середньому через 2^k ітерацій, де k — кількість значущих бітів у геш-просторі, що порівнюється.

3. Таким чином, загальна складність цього циклу для атаки покушу прообразу — $O(2^k)$

6.2 Оцінка складності атаки днів народжень:

1. Обчислення початкового гешу `preimage_hash = hl.sha512(start_message.encode())`: Це операція фіксованої складності $O(1)$, оскільки виконується один раз незалежно від ітерацій.

2. Основний цикл з обчисленням гешів: На кожній ітерації обчислюється геш нового повідомлення `msg`, який зберігається у вигляді його значущих байтів `msg_hash.digest()[bra_bytes:]`. Генерація нового повідомлення `msg` і обчислення його гешу мають сталу складність $O(1)$.

3. Оскільки атака "день народження"працює за ймовірнісним принципом, середня кількість ітерацій для знаходження колізії становить $O(2^{k/2})$, де k — розмір геш-простору (кількість бітів, що порівнюються). Причина цього в теоремі про день народження: зберігаючи всі обчислені геші у `prev_dict,` ймовірність знайти колізію зростає, і в середньому вона буде знайдена за $2^{k/2}$ ітерацій.

4. Кожна перевірка чи зберігання гешу в `prev_dict` має середню складність $O(1)$.

5. Загальна теоретична складність функції для атаки днів народжень - $O(2^{k/2})$, де k - кількість значущих бітів у геш-просторі, що порівнюються.

7. Висновки.

У рамках цієї лабораторної роботи було досліджено принципи побудови атак на геш-функції на прикладі SHA-512. Було реалізовано атаку пошуку першого прообразу (preimage attack), де метою було знайти повідомлення, яке має такий самий геш, як і заданий початковий текст; та атаку днів народжень (birthday attack), метою якох було знайти повідомлення, які утворюють колізію.

Результати атаки першого прообразу показали наступне:

Обчислювальні витрати: Оскільки використовувалася лише частина гешу, знизилася кількість необхідних обчислень для підбору відповідного повідомлення. Це показало, що часткова атака може мати успіх за меншого обсягу ресурсів.

Залишкова стійкість SHA-512: Попри використання лише кількох останніх бітів, атака залишалася обмежено успішною — для досягнення точного співпадіння були потрібні значні перебори, хоча й у меншому обсязі, ніж при повній атаці.

Ефективність: Хоча зменшення кількості бітів зробило атаку практично можливою навіть на звичайних комп'ютерах, повна стійкість SHA-512 при цьому не порушується, оскільки у реальних умовах зазвичай враховують повний геш.

Результати атаки днів народжень:

Ефективність атаки: Оскільки для атаки днів народжень не потрібно обчислювати повний 512-бітний хеш, а достатньо зосередитись лише на частині бітів, вдалося знизити обчислювальні витрати, що дозволило значно збільшити ймовірність знаходження колізій. Тестування показало, що навіть для часткових значень хешу можна знайти колізії за відносно короткий час.

Підтвердження теоретичної оцінки: Практичні результати підтвердили теоретичну ймовірність колізії при зменшенні розміру хеш-простору — знаходження колізій стало ймовірним у межах $\sqrt{2^n}$ для n бітів. Це підтвердило, що атака днів народжень є реальною загрозою для будь-якого скороченого хешу.

Обмеження для повного SHA-512: Попри те, що колізії вдалося знайти при обмеженій кількості бітів, для повного 512-бітного хешу таку атаку практично реалізувати дуже складно без великих обчислювальних потужностей. Це говорить про високу криптографічну стійкість SHA-512 для повної довжини хешу.

SHA-512 виявляється надзвичайно стійким до обох атак при використанні повного 512-бітного гешу, що забезпечує надійний захист у криптографічних системах. При скороченні бітів атака днів народжень стає більш досяжною, але на повному геші SHA-512 залишається надійним захистом як для цілісності, так і для автентифікації даних.

Загалом, атака днів народжень є більш ймовірною, але і вона є малоефективною для повного гешу SHA-512. Обидві атаки продемонстрували високу надійність цього алгоритму, який продовжує залишатися міцним криптографічним стандартом.