ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА ЕЛЕКТРОННА ІНЖЕНЕРІЯ
INFORMATION AND COMMUNICATION TECHNOLOGIES, ELECTRONIC ENGINEERING

№ 4 (1), 2024

PYTHON-МОДЕЛЬ ПРОТОКОЛУ УЗГОДЖЕННЯ СЕКРЕТНОГО КЛЮЧА У ГРУПІ З ДОВІЛЬНОЇ КІЛЬКОСТІ УЧАСНИКІВ

С. Маньковський[ORCID: 0009-0008-5217-6290], **Ю. Матієшин**[ORCID: 0000-0001-8498-3398]

Національний університет «Львівська політехніка», вул. С. Бандери, 12, 79013, Львів, Україна

Відповідальний за рукопис: Юрій Матієшин (e-mail: yurii.m.matiieshyn@lpnu.ua).

(Подано 10 Березня 2024)

Стаття присвячена проблемі узгодження спільного секретного ключа у групі з довільної кількості учасників. Обмін даними між учасниками здійснюється через відкриті канали передачі даних. Проблема обміну секретним ключем через відкриті канали даних виникла через потребу в безпечному обміні інформацією між двома або більше сторонами, які можуть бути віддаленими одна від одної та не мають спільного конфіденційного каналу зв'язку. Надійні методи обміну секретним ключем, такі як передача ключа особисто або використання захищеного каналу, не є практичними у віддалених або масштабованих сценаріях. В процесі розроблення та моделювання криптографічних систем, в яких є необхідність здійснювати узгодження криптографічних ключів в групі з двох та більше учасників, дуже зручно мати модель, яка реалізує ці алгоритми. В основі протоколу узгодження лежить протокол Діффі-Геллмана на еліптичних кривих (ECDH). Робота містить теоретичні обґрунтування, блок-схему алгоритму та програмну реалізацію алгоритму (на Python), який здійснює узгодження секретного ключа у групі з довільної кількості учасників. Для реалізації криптографічних операції на еліптичних кривих застосовано Python-бібліотеку Cryptography, зокрема алгоритми X25519, що використовують еліптичну криву Curve25519. Показано результати роботи на прикладі групи з чотирьох учасників, які демонструють коректну роботу моделі та однаковий секретний ключ, отриманий в результаті процесу узгодження. Робота також містить посилання на репозиторій GitHub з повним текстом програми. Файл multi_participant_ecdh.py містить програму узгодження секретного ключа для N учасників, написану мовою Python. Файл two_participant_ecdh.py демонструє типовий протокол Діффі-Геллмана для двох учасників. Обидві програми використовують алгоритм X25519, реалізований в Python-бібліотеці Cryptography. Таким чином, дана робота дає змогу краще зрозуміти принципи роботи алгоритмів обміну секретними ключами між двома та довільною кількістю учасників, здійснити порівняння результатів з іншими реалізаціями, застосувати розроблену модель в навчальних та в демонстраційних цілях і може бути корисною для ряду інших наукових та інженерних задач.

Ключові слова: *узгодження ключа, протокол Діффі-Геллмана, еліптична криптографія*
УДК: 681.3

1. Вступ

Проблема обміну секретним ключем через відкриті канали виникла через потребу в безпечному обміні інформацією між двома або більше сторонами, які можуть бути віддаленими одна від одної та не мають спільного конфіденційного каналу зв'язку. Надійні методи обміну

секретним ключем, такі як передача ключа особисто або використання захищеного каналу, не є практичними у віддалених або масштабованих сценаріях.

В процесі розроблення та моделювання криптографічних систем, в яких є необхідність здійснювати узгодження криптографічних ключів в групі з двох та більше учасників, дуже зручно мати модель, яка реалізує ці алгоритми. Сучасні програмні пакети, зокрема бібліотеки Python, дозволяють легко реалізувати базові криптографічні алгоритми, що дає змогу реалізації моделей та прототипів більш складних криптографічних систем.

Таким чином, в даній роботі реалізовано на Python та перевірено на ряді прикладів алгоритм обміну секретними ключами між двома та довільною кількістю учасників. Дана робота дає змогу краще зрозуміти принципи роботи цих алгоритмів обміну, здійснити порівняння результатів з іншими реалізаціями, застосувати розроблену модель в навчальних та в демонстраційних цілях і може бути корисною для ряду інших наукових та інженерних задач.

2. Застосування алгоритмів узгодження криптографічних ключів

Історія алгоритмів обміну ключами, починається з розвитку криптографії. Одним з перших алгоритмів був протокол Діффі-Геллмана, розроблений у 1976 році [1], який залишається найбільш популярним і сьогодні. Він дозволив двом сторонам безпечно обмінюватись ключами, навіть через незахищені канали зв'язку. В сучасних криптографічних системах протокол Діффі-Геллмана базується на таких алгоритмах як RSA (розроблений Райвестом, Шаміром та Адлеманом у 1977 році) чи ECC (криптографія на еліптичних кривих).

Протокол Діффі-Геллмана дуже широко використовується у багатьох сучасних криптографічних системах. Нижче перелічено декілька прикладів його застосування [2]:

1) SSL/TLS: для встановлення спільного секретного ключа між клієнтом і сервером під час першої фази обміну ключами;

2) IPsec: набір протоколів, зокрема для створення спільного ключа між двома системами для шифрування IP трафіку;

3) PGP (Pretty Good Privacy), OpenPGP та GPG (GNU Privacy Guard): для встановлення спільного секретного ключа між відправником і отримувачем для шифрування та підпису даних;

4) WireGuard: для шифрування та автентифікації трафіку VPN.

Тема узгодження криптографічних ключів була і є дуже актуальною протягом останніх десятиліть, що підтверджено багатьма науковими працями на цю тему [2-18].

3. Протокол Діффі-Геллмана для узгодження криптографічних ключів

Розглянемо суть протоколу Діффі-Геллмана для узгодження ключів між двома учасниками обміну даними. В залежності від застосування криптографічного алгоритму RSA чи ECC, для шифрування та дешифрування, математичний запис може дещо відрізнятись, однак суть узгодження ключа при цьому не змінюється. Тому, спершу розглянемо випадок застосування RSA, а згодом ECC.

Нехай маємо двох користувачів, яких назвемо Сторона А та Сторона Б, які хочуть узгодити спільний секретний ключ через відкритий канал зв’язку, використавши криптографічний алгоритм RSA, в якому параметри алгоритму g та p попередньо узгоджені, зокрема через відкритий канал. В цьому випадку узгодження ключа відбувається наступним чином:

1) Сторона А придумує приватний ключ a, який тримає в таємниці, обчислює відкритий ключ Y_a за формулою нижче та передає його Стороні Б через відкритий канал:

$$Y_a = g^a \bmod p; \tag{1}$$

2) Сторона Б придумує приватний ключ b, який тримає в таємниці, обчислює відкритий ключ Y_b за формулою нижче та передає його Стороні А через відкритий канал:

$$Y_b = g^b \bmod p; \tag{2}$$

3) Сторона А обчислює спільний секретний ключ S за формулою:

$$S = Y_b^a \bmod p = \left(g^b \bmod p\right)^a \bmod p = g^{ba} \bmod p; \tag{3}$$

4) Сторона Б обчислює спільний секретний ключ S за формулою:

$$S = Y_a^b \bmod p = \left(g^a \bmod p\right)^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p. \tag{4}$$

У випадку застосування еліптичної криптографії (ECC) Сторона А та Сторона Б повинні узгодити параметри еліптичної кривої, включно з так званою базовою точкою (генератором) g. Далі виконати схожі кроки як і у випадку застосування RSA:

1) Сторона А придумує приватний ключ a, який тримає в таємниці, обчислює відкритий ключ Y_a за формулою нижче та передає його Стороні Б через відкритий канал:

$$Y_a = a \times g; \tag{5}$$

2) Сторона Б придумує приватний ключ b, який тримає в таємниці, обчислює відкритий ключ Y_b за формулою нижче та передає його Стороні А через відкритий канал:

$$Y_b = b \times g; \tag{6}$$

3) Сторона А обчислює спільний секретний ключ S за формулою:

$$S = a \times Y_b = a \times b \times g; \tag{7}$$

4) Сторона Б обчислює спільний секретний ключ S за формулою:

$$S = b \times Y_a = b \times a \times g = a \times b \times g. \tag{8}$$

Як бачимо, різниця обчислень в процесі узгодження ключа відрізняється лише застосованими арифметичними операціями. У випадку RSA в основі лежить модулярне експоненціювання, а в основі еліптичної криптографії (ECC) – множення скалярної величини на точку на еліптичній кривій.

4. Узагальнення протоколу узгодження ключа для групи з довільної кількості учасників

У випадку, якщо кількість учасників в групі є довільною і рівною N, можна застосувати аналогічну схему узгодження секретного ключа. Нехай для узгодження ключа застосовано еліптичну криптографію та попередньо узгоджено параметри еліптичної кривої через відкриті канали. В цьому випадку узагальнено можна записати наступні кроки:

1) Кожен учасник групи P_n придумує (генерує) секретний ключ k_n, де $n \in 1...N$;

2) Кожен учасник групи P_n обчислює відкритий ключ Y_n за формулою:

$$Y_n = k_n \times g; \tag{9}$$

3) Кожен учасник групи P_n віддає свій відкритий ключ Y_n всім іншим учасникам групи;

4) Кожен учасник групи P_n після отримання $N-1$ відкритих ключів від всіх інших учасників може обчислити спільний секретний ключ S за формулою:

$$S = k_n \times \left[\prod_{i=1}^{n-1} k_i\right] \times \left[\prod_{i=n+1}^{N} k_i\right] \times g = \left[\prod_{i=1}^{N} k_i\right] \times g\,. \tag{10}$$

Як бачимо, спільний секретний ключ S буде однаковим у кожного учасника групи і рівний добутку всіх секретних ключів учасників групи на базову точку на еліптичній кривій.

5. Реалізація моделі узгодження ключа на Python

На рисунку 1 показано діаграму класу Participant. Даний клас пропонується застосувати для опису поточного стану кожного учасника групи, в якій здійснюватиметься узгодження секретного ключа. Таким чином, клас містить 4 властивості: приватний ключ даного учасника «prkey», відкритий ключ даного учасника «pubkey» (далі після раундів обміну даними він буде перезаписуватись), відкритий ключ отриманий від сусіднього учасника «rx_pubkey» та серіалізований відкритий ключ у вигляді масиву байтів «public_bytes».

Клас також містить 4 публічні методи:

- **reset** – ініціалізує початкові значення внутрішніх змінних;
- **generate_key_pair** – здійснює генерацію пари ключів;
- **receive_public_key** – отримує серіалізований відкритий ключ, десеріалізує його та зберігає у об’єкт «rx_pubkey»;
- **finish_round** – здійснює генерацію нового відкритого ключа застосовуючи «rx_pubkey» та результатом перезаписує «pubkey» та «public_bytes» (серіалізований «pubkey»).

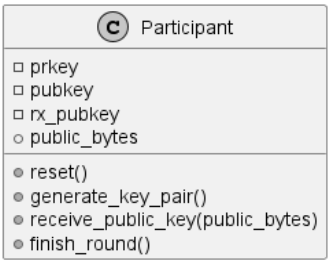

Рис. 1. Клас Participant

Нижче показано програмну реалізацію класу Participant на Python (Рис. 2). Унікальність програми у тому, що вона містить реалізацію розробленого в роботі класу Participant, який, в свою чергу, містить всі необхідні властивості та методи для реалізації запропонованого алгоритму. Приватна властивість «__pubkey» містить об’єкт відкритого ключа, а властивість «public_bytes» масив байтів, які використовують для обміну відкритим ключем між учасниками групи. Таким чином, користувач не має потреби робити серіалізацію та десеріалізацію об’єктів (вона робиться в класі), а достатньо лише передавати цей масив байтів.

```
class Participant():
    def __init__(self, id):
        self.id = id
        self.reset()
```
Ініціалізація атрибутів класу Participant

```
    # Reset all internal variables
    def reset(self):
        self.__prkey = None
        self.__pubkey = None
        self.__rx_pubkey = None
        self.public_bytes = None
```
Ініціалізація початкових значень внутрішніх змінних

```
    # Extracts bytes array from public key.
    def __make_public_bytes(self):
        self.public_bytes = \
            self.__pubkey.public_bytes(Encoding.Raw, PublicFormat.Raw)
```
Отримання значень байтів з масиву

```
    # Generate key pair for ECDH
    def generate_key_pair(self):
        self.__prkey = X25519PrivateKey.generate()
        self.__pubkey = self.__prkey.public_key()
        self.__make_public_bytes()
```
Генерація пари ключів

```
    # Receive public key as byte array and store internal public key
    def receive_public_key(self, pubkey_bytes):
        self.__received_public_key = \
            X25519PublicKey.from_public_bytes(pubkey_bytes)
```
Отримання відкритого ключа у вигляді масиву байтів і збереження внутрішнього відкритого ключа

```
    # Calculate next public key and shared secret
    def finish_round(self):
        public_bytes = self.__prkey.exchange(self.__received_public_key)
        self.__pubkey = X25519PublicKey.from_public_bytes(public_bytes)
        self.__make_public_bytes()
```
Обчислення наступного відкритого ключа

Рис. 2. Клас Participant на Python

На рисунку 3 показано схему процесу взаємодії між учасниками групи з чотирьох учасників під час узгодження секретного ключа. У кожному раунді кожен учасник групи повинен здійснити наступні операції:

- Отримати відкритий ключ від попереднього сусіда (тобто учасника групи з порядковим номером на одиницю меншим, а для нульового учасника, це буде останній учасник групи) та зберегти у поле «rx_pubkey»;
- Поточне значення відкритого ключа у полі «pubkey» передати наступному сусіду у групі. При цьому передається масив байт, за допомогою якого наступний учасник може відтворити відкритий ключ;
- Здійснити генерацію наступного відкритого ключа на основі відомого твірного елемента циклічної групи g та отриманого «rx_pubkey». Отриманим результатом перезаписати поточне значення «pubkey».

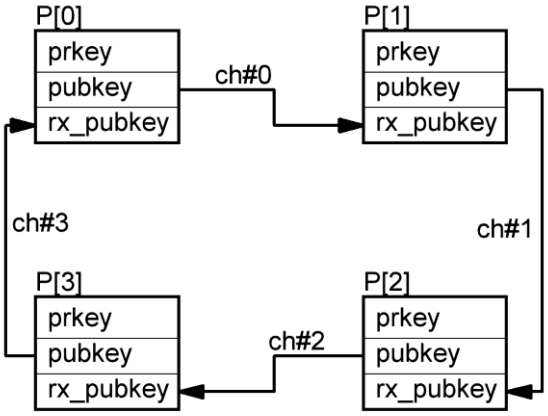

Рис. 3. Схема взаємодії між учасниками групи під час узгодження секретного ключа

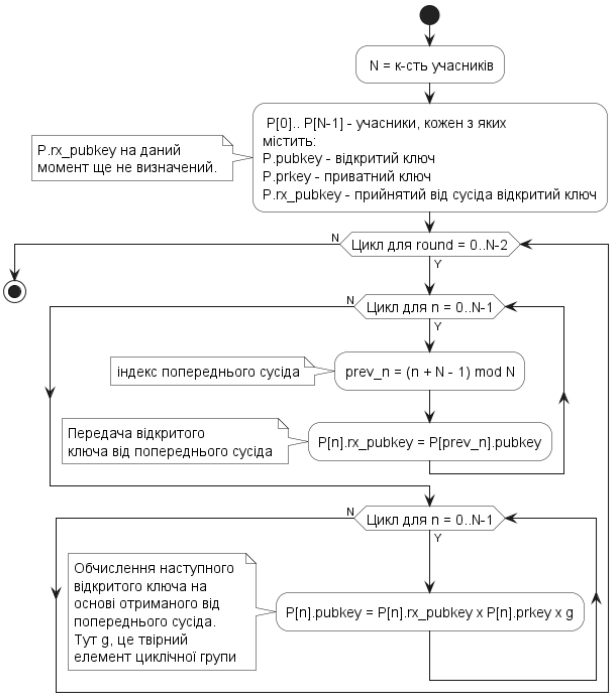

Рис. 4. Блок-схема алгоритму узгодження секретного ключа у групі з N учасників

Блок-схема алгоритму узгодження секретного ключа показана на рисунку 4. На момент початку блок-схеми, вважається, що кількість учасників в групі відома і рівна N. Крім того, кожен

учасник групи має згенеровану пару ключів, а саме відкритий ключ «pubkey» та відповідний йому секретний ключ «prkey». Ключ «rx_pubkey» на початку може бути не ініціалізований. Малою літерою *g* позначено твірний елемент циклічної групи.

Нижче показано програмну реалізацію узгодження секретного ключа в групі, яка складається з «participants_count» учасників (Рис. 5). Алгоритм роботи програми базується на блок-схемі, зображеній на рисунку 4. Унікальність наведеної програми полягає у тому, що вона показує реалізацію запропонованого алгоритму із застосуванням об'єктів розробленого класу Particitant.

Спочатку створюються об'єкти класу Particitant кількістю «participants_count» (див. позначення на Рис. 5). Далі кожен учасник здійснює генерацію пари ключів викликом методу «generate_key_pair». Після цього в циклі «rounds_count» разів відбуваються раунди обміну даними між сусідніми учасниками, в результаті чого кожен учасник отримає спільний секретний ключ.

```
# Create list of participants
participants = [Participant(n) for n in range(participants_count) ]

# Number of rounds to obtain shared secret
rounds_count = participants_count - 1
```

Об'єкти класу Participant

```
# Each participant Generate Key pairs
for cur_participant in participants:
    print(f"Participant {cur_participant.id} generate key pair.")
    cur_participant.generate_key_pair()
```

Генерація пари ключів

```
# Perform rounds for shared secret distributions
for cur_round_idx in range(rounds_count):
    print(f"======== Start round {cur_round_idx} ========")
    # First transfer public keys (public bytes) to the next neighbours
    for cur_participant_idx in range(participants_count):
        prev_participant_idx = \
            (cur_participant_idx + participants_count - 1) % participants_count
        participants[cur_participant_idx].receive_public_key(
            participants[prev_participant_idx].public_bytes)
        print(f"Transfer public key from participant"
              f" {prev_participant_idx} to {cur_participant_idx}")
    # Seoond calculate next public key and shared secret
    for cur_participant in participants:
        cur_participant.finish_round()
        cur_public_bytes =  cur_participant.public_bytes
        print(f"Participant`s {cur_participant.id} shared secret: "
              f"{cur_public_bytes.hex()}")
```

Циклічний обмін даними між учасниками

Рис. 5. Програмна реалізація на Python узгодження секретного ключа в групі

Результати роботи програми показані нижче (Рис. 6 та Рис. 7). Як видно, відбувається 3 раунди обмінів даними між чотирма учасниками групи та вивід спільного секретного ключа після кожного раунду. В результаті, після третього раунду (раунд 2 при нумерації з нуля), видно, що всі учасники групи отримують однакове значення секретного ключа (див. позначення на Рис. 7).

```
Participant 0 generate key pair.
Participant 1 generate key pair.
Participant 2 generate key pair.
Participant 3 generate key pair.
======== Start round 0 ========
Transfer public key from participant 3 to 0
Transfer public key from participant 0 to 1
Transfer public key from participant 1 to 2
Transfer public key from participant 2 to 3
Participant`s 0 shared secret: 1e1a8e518a8101aa60c7b2687e899911675a9d8da8cc842c84c5858aa3f2623a
Participant`s 1 shared secret: 22e7dab6ac99c81ed4801142d0c7b82dbaf2cf304829c60aca97e492de970e3d
Participant`s 2 shared secret: 9e6b4d92872342e79ca2b2291c98ce03a65f02af055f7c43b1470cbc0825480c
Participant`s 3 shared secret: 7bfd08c74ba05f298b2ca2e4691a43d261f53495c43e9ae90b528cff4453d62f
```

Рис. 6. Результати роботи програми узгодження секретного ключа в групі (продовження на Рис. 7)

```
======== Start round 1 ========
Transfer public key from participant 3 to 0
Transfer public key from participant 0 to 1
Transfer public key from participant 1 to 2
Transfer public key from participant 2 to 3
Participant`s 0 shared secret: 327bdd53ef3e975627fe031012b011709096eef5e8f3504730a5783c8dc3481d
Participant`s 1 shared secret: cc19473c8c772f7e563053cdf673e8f2ed202cdd9b385c139f81eeef1b3f7f55
Participant`s 2 shared secret: ce1e962f5699baf2f4daeb6c49f6c85c2366afdc094e5d1f27a90012b16f7b48
Participant`s 3 shared secret: f7f90cb15d7131ca17d27944ce2de9b15e5c03bed4961440840e06719c1fea47
======== Start round 2 ========
Transfer public key from participant 3 to 0
Transfer public key from participant 0 to 1
Transfer public key from participant 1 to 2
Transfer public key from participant 2 to 3
Participant`s 0 shared secret: a7e9f995ab17a847faa0a92fd0339b4dc58280cc571a67ebbbf513c9f11d2b0f
Participant`s 1 shared secret: a7e9f995ab17a847faa0a92fd0339b4dc58280cc571a67ebbbf513c9f11d2b0f
Participant`s 2 shared secret: a7e9f995ab17a847faa0a92fd0339b4dc58280cc571a67ebbbf513c9f11d2b0f
Participant`s 3 shared secret: a7e9f995ab17a847faa0a92fd0339b4dc58280cc571a67ebbbf513c9f11d2b0f
```

Рис. 7. Результати роботи програми узгодження секретного ключа в групі (продовження)

Повний текст програми доступний на GitHub за наступним посиланням: https://github.com/mspartak/science/tree/master/multi_party_ecdh.

Файл multi_participant_ecdh.py містить програму узгодження секретного ключа для *N* учасників, написану мовою Python. Файл two_participant_ecdh.py демонструє типовий протокол Діффі-Геллмана для двох учасників. Обидві програми використовують алгоритм X25519 реалізований в Python-бібліотеці Cryptography.

Висновки

В роботі реалізовано на Python та перевірено на ряді прикладів модель узгодження секретного ключа у групі з довільною кількістю учасників. Показано коректність роботи моделі на основі отримання однакового ключа всіма учасниками на останньому раунді обмінів. Модель може бути корисною при реалізації схожих систем узгодження ключів, а також для подальших наукових та інженерних досліджень і експериментів.

Список використаних літературних джерел

[1]. W. Diffie and M. Hellman, "New directions in cryptography," in IEEE Transactions on Information Theory, November 1976, vol. 22, no. 6, pp. 644-654, available at: https://www-ee.stanford.edu/~hellman/publications/24.pdf (Accessed 24 February 2024), doi: 10.1109/TIT.1976.1055638.

[2]. Burmester, M. (2011), "Group Key Agreement," in Encyclopedia of Cryptography and Security, Springer, Boston, MA, pp. 520-526, available at: https://doi.org/10.1007/978-1-4419-5906-5_320 (Accessed 24 February 2024), doi: 10.1007/978-1-4419-5906-5_320.

[3]. Wu, Q., Mu, Y., Susilo, W., Qin, B. and Domingo-Ferrer, J. (2009), "Asymmetric Group Key Agreement," in Advances in Cryptology – EUROCRYPT 2009, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, vol. 5479, pp. 153-170, available at: https://doi.org/10.1007/978-3-642-01001-9_9 (Accessed 24 February 2024), doi: 10.1007/978-3-642-01001-9_9.

[4]. K. Shen, L. Zhang, R. Zhang and Q. Fang, "Asymmetric Group Key Agreement Protocol from Short Signatures," in 2022 IEEE 8th International Conference on Computer and Communications (ICCC), Chengdu, China, 2022, pp. 1229-1233, available at: https://ieeexplore.ieee.org/document/10065683 (Accessed 24 February 2024), doi: 10.1109/ICCC56324.2022.10065683.

[5]. Alwen, J., Coretti, S., Jost, D. and Mularczyk, M. (2020), "Continuous Group Key Agreement with Active Security," in Theory of Cryptography, TCC 2020, Lecture Notes in Computer Science, Springer, Cham, vol. 12551, pp. 261-290, available at: https://doi.org/10.1007/978-3-030-64378-2_10 (Accessed 24 February 2024), doi: 10.1007/978-3-030-64378-2_10.

[6]. L. Harn and C. Lin, "Efficient group Diffie–Hellman key agreement protocols", Computers & Electrical Engineering, 2014, vol. 40, issue 6, pp. 1972-1980, ISSN 0045-7906, available at: https://doi.org/10.1016/j.compeleceng.2013.12.018 (Accessed 24 February 2024), doi: 10.1016/j.compeleceng.2013.12.018.

[7]. Byun, J.W. and Lee, D.H. (2005), "N-Party Encrypted Diffie-Hellman Key Exchange Using Different Passwords," in Applied Cryptography and Network Security, ACNS 2005, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, vol. 3531, pp. 75-90, available at: https://doi.org/10.1007/11496137_6 (Accessed 24 February 2024), doi: 10.1007/11496137_6.

[8]. K. V. Pradeep, V. Vijayakumar and V. Subramaniyaswamy, "An Efficient Framework for Sharing a File in a Secure Manner Using Asymmetric Key Distribution Management in Cloud Environment", Journal of Computer Networks and Communications, 2019, vol. 2019, 8 pages, article ID 9852472, available at: https://doi.org/10.1155/2019/9852472 (Accessed 24 February 2024), doi: 10.1155/2019/9852472.

[9]. X. Li, Y. Wang, P. Vijayakumar, D. He, N. Kumar and J. Ma, "Blockchain-Based Mutual-Healing Group Key Distribution Scheme in Unmanned Aerial Vehicles Ad-Hoc Network," in IEEE Transactions on Vehicular Technology, Nov. 2019, vol. 68, no. 11, pp. 11309-11322, available at: https://ieeexplore.ieee.org/abstract/document/8846098 (Accessed 24 February 2024), doi: 10.1109/TVT.2019.2943118.

[10].L. Zhang, Q. Wu, B. Qin, J. Domingo-Ferrer and Ú. González-Nicolás, "Asymmetric group key agreement protocol for open networks and its application to broadcast encryption", Computer Networks, 2011, vol. 55, issue 15, pp. 3246-3255, ISSN 1389-1286, available at: https://doi.org/10.1016/j.comnet.2011.06.016 (Accessed 24 February 2024), doi: 10.1016/j.comnet.2011.06.016.

[11].L. Zhang, F. Zhang, Q. Wu and J. Domingo-Ferrer, "Simulatable certificateless two-party authenticated key agreement protocol", Information Sciences, 2010, vol. 180, issue 6, pp. 1020-1030, ISSN 0020-0255, available at: https://doi.org/10.1016/j.ins.2009.11.036 (Accessed 24 February 2024), doi: 10.1016/j.ins.2009.11.036.

[12].Y. Sun, Q. Wen, H. Sun, W. Li, Z. Jin and H. Zhang, "An Authenticated Group Key Transfer Protocol Based on Secret Sharing", Procedia Engineering, 2012, vol. 29, pp. 403-408, ISSN 1877-7058, available at: https://doi.org/10.1016/j.proeng.2011.12.731 (Accessed 24 February 2024), doi: 10.1016/j.proeng.2011.12.731.

[13].I. Ingemarsson, D. Tang and C. Wong, "A conference key distribution system," in IEEE Transactions on Information Theory, September 1982, vol. 28, no. 5, pp. 714-720, available at: https://ieeexplore.ieee.org/document/1056542 (Accessed 24 February 2024), doi: 10.1109/TIT.1982.1056542.

[14].Padmashree, M.G., Mallikarjun, Arunalatha, J.S., Venugopal, K.R. (2022), "GKEAE: Group Key Exchange and Authentication with ECC in Internet of Things," in Intelligent Systems, Lecture Notes in Networks and Systems, Springer, Singapore, vol. 431, pp. 1-10, available at:
https://doi.org/10.1007/978-981-19-0901-6_1 (Accessed 24 February 2024), doi: 10.1007/978-981-19-0901-6_1.

[15].S. Gupta, A. Kumar and N. Kumar, "Design of ECC based authenticated group key agreement protocol using self-certified public keys," in 2018 4th International Conference on Recent Advances in Information Technology (RAIT), Dhanbad, India, 2018, pp. 1-5, available at: https://ieeexplore.ieee.org/document/8388999 (Accessed 24 February 2024), doi: 10.1109/RAIT.2018.8388999.

[16].V. C. Giruka, S. Chakrabarti and M. Singhal, "A distributed multi-party key agreement protocol for dynamic collaborative groups using ECC", Journal of Parallel and Distributed Computing, 2006, vol. 66, issue 7, pp. 959-970, ISSN 0743-7315, available at: https://doi.org/10.1016/j.jpdc.2006.03.006 (Accessed 24 February 2024), doi: 10.1016/j.jpdc.2006.03.006.

[17].Letenko, Yu.O., Riabukho, O.M. and Turka, T.V. (2015), "Protokoly rozpodilu ta uzghodzhennia kliucha [Key Distribution and Reconciliation Protocols]", Zbirnyk naukovykh prats fizyko-matematychnoho fakultetu DDPU, issue 5., pp. 30-37, available at: http://dspace.ddpu.edu.ua/ddpu/bitstream/123456789/409/1/znp-2015_030.pdf (Accessed 24 February 2024).

[18].Krasylenko, V. H. and Nikitovych, D.V. (2017), "Modeliuvannia protokoliv uzghodzhennia sekretnoho matrychnoho kliucha dlia kryptohrafichnykh peretvoren ta system matrychnoho typu [Modelling secret matrix key agreement protocols for cryptographic transformations and matrix-type systems]", Systemy obrobky informatsii, issue 3, pp. 151-157, available at: http://nbuv.gov.ua/UJRN/soi_2017_3_32 (Accessed 24 February 2024).

PYTHON MODEL OF SECRET KEY AGREEMENT IN THE GROUP OF ARBITRARY NUMBER OF PARTICIPANTS

Spartak Mankovskyy, Yurij Matiieshyn

Lviv Polytechnic National University, S. Bandery Str., 12, 79013, Lviv, Ukraine

The article is devoted to the problem of common secret key agreement in a group of an arbitrary number of participants. Data is exchanged between participants through open data channels. The problem of sharing a secret key over open data channels arose due to the need for a secure exchange of information between two or more parties that may be remote from each other and do not have a common confidential communication channel. Reliable methods of secret key exchange, such as transferring the key in person or using a secure channel, are not practical in remote or scalable scenarios. In the process of developing and modelling cryptographic systems, in which there is a need of cryptographic keys agreement in a group of two or more participants, it is very convenient to have a model that implements these algorithms. The agreement protocol is based on the Diffie-Hellman protocol on elliptic curves (ECDH). The paper contains theoretical justifications, a flow chart of the algorithm, and a Python implementation of the algorithm that performs the secret key agreement in a group of an arbitrary number of participants. To implement cryptographic operations on elliptic curves, the Python library Cryptography is used, in particular, the X25519 algorithms that use the elliptic curve Curve25519. The results of the work are shown on an example for a group of four participants, which demonstrate the correct operation of the model and the same secret key obtained as a

result of agreement process. The paper also contains the link to a GitHub repository with the full program. The multi_participant_ecdh.py file contains a secret key agreement program for N participants written in Python. The file two_participant_ecdh.py demonstrates a typical two-participant Diffie-Hellman protocol. Both programs use the X25519 algorithm implemented in the Cryptography Python library. Thus, this work makes it possible to better understand the principles of secret key exchange algorithms between two and an arbitrary number of participants, to compare the results with other implementations, to apply the developed model for educational and demonstration purposes, and may be useful for a number of other scientific and engineering tasks.

Keywords: *Key Agreement, Diffie-Hellman Protocol, Elliptic Cryptography*