КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Кваліфікаційна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформатика»

спеціальності 122 Комп'ютерні науки

на тему:

РЕАЛІЗАЦІЯ ПРОТОКОЛУ "СВІЙ-ЧУЖИЙ"

Виконав студент 4 курсу
Станіслав БЕЗКОРОВАЙНИЙ

(підпис)

Науковий керівник:
Професор, доктор фіз.-мат. наук
Анатолій АНІСІМОВ

(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики
«____»_____2023 р.
протокол №_____

Завідувач кафедри

Василь ТЕРЕЩЕНКО

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи 56 сторінок, 4 ілюстрації, 31 джерело.

ДОВЕДЕННЯ З НУЛЬОВИМ РОЗГОЛОШЕННЯМ, ІНТЕРАКТИВНІ ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ, ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ, ШВИДКА АВЕНТИФІКАЦІЯ.

Об’єктом дослідження є протоколи автентифікації, які також називають протоколами “свій-чужий”. Були розглянуті їх реалізація, властивості та побудова. Об’єктом розроблення є один з таких протоколів а також бібліотека з деякими з них реалізованими на мові програмування Typescript.

Метою роботи було дослідження на аналіз різних протоколів автентифікації та вивчення їх характеристик, та властивостей які ці протоколи мають мати для практичного застосування. Також метою була побудова та реалізація одного з таких алгоритмів.

Методи дослідження: аналіз існуючих протоколів, проектування архітектури бібліотеки з деякими наявними протоколами.

У якості інструмента розроблення було використано мову програмування Typescript з менеджерем залежностей Yarn та у якості середовища програмного забезпечення використовувався редактор тексту Visual Studio Code.

Результат роботи є проведений детальний аналіз різноманітних протоколів автентифікації, були досліджені основні властивості, їх характеристики та можливі вектори атак на ці протоколи, а також сфери застосування цих протоколів. Був запропонований власний варіант протоколу швидкої автентифікації. Була реалізована бібліотека з деякими з розглянутих протоколів автентифікації.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. Постійний прогрес в сфері телекомунікаційних, медичних, онлайн-банкінг та інших послугах вимагає аналогічного розвитку в схемах автентифікації, які також називають протоколами “свій-чужий”, бо часто вони дають змогу виявити приналежність людини до певної групи. У цьому зацікавлені всі організації, яким потрібно обмежувати доступ до конфіденційної інформації або будь-яких інших обмеженнях на доступ, наприклад, доступ до коштів користувачів. Активне використання таких протоколів знайшла і військова сфера через необхідність розрізняти пристрої свої від ворожих.

Актуальність роботи та підстави для її виконання. Постійно збільшується кількість веб-сервісів, мобільних додатків та інших онлайн-ресурсів, які потребують ідентифікації користувачів. Забезпечення безпеки та захисту особистої інформації стає все важливішим завданням, адже зловмисники активно використовують різноманітні методи для отримання несанкціонованого доступу до даних користувачів. Тому протоколи розвиток протоколів автентифікації є дуже актуальним.

Мета й завдання роботи. Метою даної роботи є дослідження та аналіз сучасних криптографічних протоколів автентифікації, а також сигма протоколів, зокрема протоколу Шнора, які є не тільки алгоритмом автентифікації, а й одним з базових прикладів доведень з нульовим розголошенням:

- провести аналіз структури та рівня безпеки відомих відомих протоколів ідентифікації;
- провести аналіз відомих протоколів автентифікації та знайти сфери їх застосування;

- розробити програмну реалізацію деяких з розглянутих протоколів у виді бібліотеки на мові програмування TypeScript.

Об'єкт, методи й засоби розроблення. Об’єктом аналізу дипломної роботи є протоколи автентифікації, криптографічні схеми автентифікації, їх безпека та огляд варіантів їх застосування. Об’єктом розробки є реалізація бібліотеки з протоколами автентифікації з використанням еліптичних кривих. Протокол що був реалізований був обраний з урахуванням результатів попередніх розділів.

Для програмної реалізації було обрано мову TypeScript. TypeScript є типізованою надбудовою над мовою JavaScript та транспілюється в нього. JavaScript є інтерпретованою мовою програмування, що підтримує процедурне та об’єктно-орієнтоване програмування. Завдяки використанню мови Typescipt можна зручність та типізацію з багатою екосистемою існуючих бібліотек на Javascript.

Під час розробки даного програмного засобу використовувався текстовий редактор VS Code. Цей текстовий редактор є дуже легкий в застосуванню та в вимогах до апаратного забезпечення комп’ютера. Він дозволяє встановлювати різноманітні додатки, що дозволяють легко інтегруватися з будь якою мовою.

Можливі сфери застосування. Використання мови TypeScript надає можливість легкого інтегрування бібліотеки як в браузерні додатки, так і в серверну частину. Вона підходить для будь-яких проектів, що потребують використання криптографічної автентифікації та використовують JavaScript або TypeScript.

1 АВТЕНТИФІКАЦІЯ

1.1 Загальний огляд та історія автентифікації

В загальному розумінні протолокол “свій-чужий”, або протокол автентифікації — це процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора[1]. Цим ідентифікатором може бути будь-що: рахунок в банку, інтернет-акаунт, тощо. З різноманітними протоколами автентифікації ми зустрічаємось кожного дня, коли, наприклад входимо в наш гугл акаунт або навіть коли спілкуємось з кимось по телефону, голос можно вважати примітивною формою автентифікації завдяки якій інша людина розуміє, що це саме ви, хто спілкується на іншій стороні лінії.

В давньому світі часто для автентифікації використовували предмети (наприклад родинне кільце або печатка). Ті, хто не мав можливості використання такого роду речей часто використовували паролі – кодові фрази, завдяки яким можна було перевірити належність людини організації. Приклади вище показують, що автентифікація може окрім ідентифікування людини може доводити її приналежність до певних груп. З часом ідея паролів стала фундаментом сучасної взаємодії з інформаційними системами. Кожного разу коли заходимо до інтернет-ресурсів доводиться вводити ім’я свого профілю (логін) та пароль до нього (або будь-яку іншу секретну фразу).

З часом набирають популярність й інші методи автентифікації – біометричні тощо. наприклад, за сітківкою ока або за відбитками пальців можна ідентифікувати людину. Проблема секретного зберігання паролів також породила такі засоби як Google Sign In[2], що дозволяють користувачам використовувати їх аккаунт Google для автентифікації на сайті. Аналогічні сервіси також надаються й іншими компаніями такими як Github, Facebook, тощо.

Загалом, існує безліч протоколів ідентифікації, що засновані не лише на криптографічних примітивах, наприклад це може бути біометрія або буквально

підтвердження через СМС, це все може бути прикладами процедури автентифікації.

1.2 Криптографічні алгоритми автентифікації

Протоколи автентифікації, що були зазначені в минуломі розділі є зручними для звичайних користувачів, але у них є чимало проблем:

- Вони потребують активної взаємодії з користувачем, що не підходить для ситуацій коли автентифікацію варто проводити автоматично.
- Майже завжди некриптографічні протоколи потребують довірених третіх сторін, що будуть перевіряти та зберігати паролі, біометричні дані і т.д.
- Паролі, що легко запам’ятовуються найчастіше можуть бути відносно легко перебрані на сучасних комп’ютерах. З розвитком технологій штучного інтелекту такі природні засоби автентифікації як голос стають повністю скомпромітованими.

Загалом, видно що схеми автентифікації, що є природніми і простими для людини стають все більш взразливими для сучасних зловмисників. Отже, виникає потреба у створенні математично стійких алгоритмів автентифікації, які є:

- Безпечними проти будь яких сучасних комп’ютерів.
- Можуть автоматично виконувати процедуру автентифікації.

Для цього підходять криптографічні протоколи автентифікації, безпека яких засноваана на криптографічних примітивах і вони є відміним засобом для автентифікації в ситуаціях коли більш зрозумілі нетехнічнічним користувачам методи автентифікації є або недоступними або не відповідають вимогам безпеки.

Формальне визначення криптографічного протоколу автентифікації наступне: Схема автентифікації є інтерактивним протоколом між двома сторонами - пруверем (від англійської prover – той, що доводить) P та веріфаєром (від англійської verifier – той, що перевіряє) V. Якщо протокол успішний, то після завершення протоколу перевірювач переконаний, що він взаємодіє з доказувачем

або, точніше, з кимось, хто знає секретний ключ, що відповідає публічному ключу прувера[3].

В залежності від застосування протоколи автентифікації мають різні рівні безпеки. Майже завжди вимагаються наступні властивості:

- Повнота – якщо прувер має секрет та обидва учасники дотримуються правил протоколу, то він зможе переконати веріфаєра.
- Коректність – якщо прувер обмежений поліноміальним часом і не знає секрета, то вірогідність того що він зможе переконати перевіряючого настільки незначна, що нею можна знехтувати.

В залежності від сфер застосування можуть також вимагатися наступні властивості:

- В системах, який треба бути резистентним до зловмисного перевіряючого додається вимога, щодо того, що перевіряючий за поліноміальний час не може відновити секрет.
- Іноді додаються вимоги щодо стійкості щодо атак прослуховування (“eavesdropping”)[4], де зловмисник може читати всі повідомлення в розмові та “людини посередині” (“man-in-the-middle attack”)[5], де, наприклад, зловмисний перевіряч може намагатися паралельно з взаємодією з користувачем представляти себе від імені цього користувача до іншого перевіряючого. Ці вимоги потрібні адже часто схеми автентифікації використовується по незахищеним каналам зв’язку.

Схеми автентифікації найчастіше використовують асиметричну криптографію або доведення нульового розголошення і будуються навколо того, що прувер має довести що він володіє приватним ключем до свого публічного ключа, де публічний ключ обчислюється з приватного за допомогою односторонніх функцій. Функція f називається односторонньою, якщо її можна обчислити за поліноміальний час, але неможливо за поліноміальний час обчислити f^{-1}. Варто зазначити, що не існує функцій для яких це строго доведено, адже існування таких функцій означало би, що $NP \neq P$, що не є наразі

сторого доведеним. Однак існує чимало функцій, які перевіркою часу показали, що скоріш за все їх обчислити за поліноміальний час неможливо: дискретний логарифм на деяких групах, пошук квадратичних залишків тощо.

Криптографічні схеми автентифікації використовуються найчастіше в сферах де використання інших засобів автентифікації не є можливим:

- Військова сфера, адже комунікація часто ведеться по незахищеним каналам, де безпека має бути на найвищому рівні.
- Банківські та інші види пропускних карток, адже вони потребують автоматичної автентифікації.
- Блокчейн проекти, коли треба довести, що користувач володіє певним публічним ключем від аккаунту.

1.2.1 Формальне визначення криптографічного протокола ідентифікації

Криптографічний протокол автентифікації – це трійка (G, P, V), де G – це алгоритм що працює за ймовірнісний поліноміальний час, P та V – це два інтерактивні алгоритми, що працюють за ймовірнісний поліноміальний час, з наступними властивостями[4]:

- Алгоритм G називають алгоритмом генерації ключей. Це приймає на вхід рядок у форматі: 1^k (посуті число k записане в унарній системі числення) і виводить пару рядків (S, I). k називають параметром безпеки, S - секретний (приватний) ключ, а I - відкритий (публічний) ключ.
- P отримує на вхід пару (S, I), а V отримує на вхід I. Після інтерактивного виконання P та V, V виводить 1 (показуючи "прийняти") або 0 (показуючи "відхилити"). Для заданого S та I, вивід V в кінці цієї взаємодії, звичайно, є простором ймовірностей і позначається як $(P(S,\ I),\ V(I))$.
- Коректний P завжди повинен мати змогу домогтись успіху в тому, щоб V погодився. Формально, для всіх k та для всіх

$(S,\ I) \in [G(1^k)],\ (P(S,\ I),\ V(I))\ =\ 1$ з ймовірністю 1, де $[G(1^k)]$ – це множина всіх можливих результатів при виклику G з параметром 1^k.

1.2.2 Безпека криптографічного протокола автентифікації при активних атаках

Безпека є поняттям завжди відносним, але для того, щоб науково підходити до аналізу алгоритмів, треба формально визначити поняття безпеки та зловмисника від якого алгоритм має бути здатним захищатись

Зловмисник (P', V') – це пара ймовірнісних інтерактивних алгоритмів, що виконуються за поліноміальний час. Для заданої пари приватного/публічного ключів $(S,\ I)$, ми позначаємо h рядок, який виводиться V_0 (при вхідних даних I) після взаємодії з P (при вхідних даних $(S,\ I)$) деяку кількість разів. Зверніть увагу, що ці взаємодії виконуються послідовно. Знову ж таки, для заданого S та I, $(P(S,\ I),\ V'(I))$ є простором ймовірностей. Рядок h («рядок допомоги») використовується як вхід для P', який намагається змусити V (на вхід I) погодитися. Ми позначаємо $(P'(h),\ V(I))$ вивід V після взаємодії з $P'(h)$.

Кажучи іншими словами, ми дозволяємо зловмиснику кілька разів (що обмежена поліномом), бути веріфаєром при взаємодії з чесним прувером. Рядок h – це по суті все те, що запам’ятав зловмисник поки взаємодіяв з чесним довідником. В виродженому випадку це може бути списком всіх повідомлень що відбулися під час взаємодії.

Ідентифікаційна схема $(G,\ P,\ V)$ вважається безпечною проти активних атак якщо для всіх зловмисників $(P',\ V')$ для всіх констант $c\ >\ 0$ і для всіх достатньо великих k

$Pr[s\ =\ 1 \mid (S,\ I) \leftarrow G(1^k);\ h \leftarrow (P(S,\ I),\ V'\ (I));\ s \leftarrow (P'\ (h),\ V(I))]\ <\ k^{-c}$[4].

Інтуїтивно кажучи, безпека при активних атаках зазначає, що зловмисник не може переконати чесного перевіряльника в тому що він має секрет навіть якщо зловмисник попередньо був перевіряльником при взаємодії в сутністю що такий секрет має. Це властивість є надзвичайно важливою при побудові схем

автентифікації і однією з тих, що відрізняє від звичайного підпису константного виразу, адже той підпис може бути перевикористаним. Цікаво, що чимало некриптографічних схем автентифікації, що використовуються таку властивість не мають, наприклад пароль, що завжди передається у явному вигляді перевикористаний бути може.

2 СУЧАСНІ ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ ТА ЇХ ПОРІВНЯННЯ

Як вже зазначалось, протоколи автентифікації (хоча й відносно примітивні) використовувались з давніх давен. Більшу частину історії автентифікація перевірялась за допомогою фізичних чинників – печаток, паспортів та інших. Наразі з цифровізацією суспільства зростає потреба в нових, дистанційних формах автентифікації. Дистанційність, часткова анонімність, а також швидкість з якою операції відбуваються онлайн взаємодія роблять будь які атаки де можливо себе ідентифікувати як іншу особу особливо небезпечними. Будь який провал систем автентифікації може вилитись у величезні збитки та навіть втрати життів, якщо справа йде про особливо секретну інформацію.

В залежності від призначення можуть застосовуватися різні протоколи автентифікації. Ті протоколи, що потребують взаємодію з користувачем кожен день роблять акцент в першу чергу на зручному інтерфейсі користувача, роблять так, щоб людина сама не скоювала помилок, навіть якщо з суто математичної точки зору вони не є дуже безпечними. З іншого боку, середовища, що не потребують взаємодії з людиною найчастіше використовують криптографічні протоколи автентифікації.

2.1 Автентифікація за допомогою паролю

Найпростішою і найвідомішою формую автентифікації є автентифікація за допомогою паролю. Ця форма автентифікації найчастіше використовується на вебсайтах, телефонах та інших приладах, де автентифікація робиться користувачами особисто. Найчастіше у кожного користувача є певний логін (це може бути ім’я користувача на сайті, його імейл тощо) та пароль. Логін вважається публічним і не потребує захисту. В свою чергу пароль має бути захищений і пам’ятатися чи бути надійно записаним користувачем.

Існує безліч рекомендацій того, як користувачу варто створювати та зберігати паролі. Загалом поради для створення надійного паролю включають в

себе використання довгих паролів, що використовують велику кількість різних символів. Для того, щоб було легше запам’ятати пароль варто використовувати пароль що складається з кількох слів складених у речення. В ідеальному випадку пароль має зберігатися в пам’яті користувача, але на жаль не завжди це є можливим. Для зберігання паролю часто використовують менеджери паролів.

В найпростішому випадку автентифікація через пароль відбувається наступним чином:

- Користувач відправляє свій логін та пароль на сервер.
- Сервер, що зберігає в базі даних всі паролі перевіряє що пароль для даного логіну є правильним і повертає “так” або “ні” в залежності від того що була автентифікація успішною.

2.1.1. Безпечна передача паролу.

Першим очевидним недоліком є те, що зловмисник може перехопити повідомлення і вкрасти пароль. Для того, щоб цього уникну найчастіше використовують захищені канали зв’язку. Наприклад, через HTTPS або WSS.

- HTTPS – надбудова над протоколом комунікації HTTP, що використовує TLS або SSL для шифрування даних[6]. В свою чергу WSS – це надбудова над WS, що використовує TLS або SSL для шифрування даних.
- SSL (Secure Sockets Layer) - це криптографічний протокол, який був розроблений для забезпечення безпеки мережевих комунікацій в Інтернеті. SSL дозволяє захистити передачу даних між веб-сайтом та користувачем, забезпечуючи конфіденційність, цілісність та автентичність даних. SSL був оновлений до TLS (Transport Layer Security), який є більш сучасним і безпечним протоколом. Однак термін SSL все ще використовується в загальному значенні, щоб описати криптографічний протокол, який застосовується для забезпечення безпеки мережевих комунікацій.

Основні ідеї, що лежать за TLS та SSL – використання довірених третіх організацій, що будуть підтверджувати публічний ключ до сервера. Після того як публічний ключ був встановлений, використовуються стандартні техніки

асиметричного шифрування для того, щоб створити тимчасовий ключ для симетричного шифрування комунікацій.

2.1.2 Безпечне зберігання паролів

Іншим недоліком базового підходу є те, що паролі зберігаються в явному виді в базі даних. В такому разі якщо хтось отримає до неї доступ або буде зловмисний співробітник веб сервісу, що захоче викрасти паролі користувачів, то вони можуть отримати доступ до даних користувачів. Наразі існують стандарти, такі як в [7], що вимагають від розробників зберігання паролів у захешованому вигляді, тобто якщо пароль p, то в базі даних має зберігатися тільки $H(p)$, де H – криптографічно безпечна геш-функція.

Геш-функція – це функція, яка перетворює вхідні дані довільної довжини в геш-код фіксованої довжини. Геш-код - це унікальний числовий ідентифікатор, який представляє вхідні дані. Геш-функція є криптографічно безпечною, якщо вона має незначну ймовірність колізії, тобто ситуації, коли два різні вхідні значення дають один і той же геш-код. Іншою важливою властивістю є складність інвертування, тобто по данному $H(p)$ знаходити p.

Використання геш-функцій означає, що навіть якщо база даних була вкрадена, зловмисник не дізнається про паролі користувачів. Це є головною причиною чому, наприклад, якщо ви намагаєтесь відновити пароль всі надійні сайти будуть вам пропонувати його саме змінити, але не надішлють вам старий. Сайт сам не знає значення вашого старого пароля, адже зберігає лише його геш і не може по ньому обчислити пароль. Якщо ж сайт під час відновлення паролю може вам відправити сам пароль, то цей сервіс є ненадійним і паролі всіх користувачів доступні до будь-кого з доступом до бази даних.

2.1.3 Покращення використання автентифікації через пароль

Одним з основних недоліків при взаємодії з автентифікацією через пароль є необхідність зберігати великі паролі для великої кількості сайтів. Для кожного сайту має бути свій пароль. Інакше якщо зловмисник знайшов його хоча б одного

з них (наприклад, через ненадійне зберігання паролів сервісами), то він отримає доступ до всіх інших сервісів. З іншого боку паролі мають бути довгими, для того, щоб ускладнити атаку повного перебору.

Проблема запам’ятовування довгих паролів найчастіше вирішується одним з наступних способів:

- Використання менеджерів паролів. Оскільки по справжньому довгі паролі найчастіше важко запам’ятати можна використовувати менеджери паролів, але недолік в тому, що варто пам’ятати пароль до менеджера паролів і у випадку якщо зловмисник дізнається ваш пароль до менеджера паролів, він отримає до всіх ваших сервісів одночасно.
- Використання двофакторної автентифікації. Використання додаткового джерела автентифікації робить вірогідність отримати доступ до акаунту менше навіть якщо основний пароль був скомпрометований. Наприклад, під час логіну може вимагатися додатково СМС-підтвердження.
- Біометрія. У кожної людини є деякі унікальні частини тіла: відбитки пальців, будова сітківки тощо. Її можна використовувати як довгий пароль. Хоча випадковим такий пароль назвати важко, але довжина і легкість роблять його одним з рекомендованих виборів для користувачів. До того ж, якщо оминути зчитувач біометрії (наприклад обов’язково використовувати саме сканер відбитків пальців), то навіть якщо хтось отримав доступ до ваших відбитків пальців, то зловмиснику буде все ще важко його ввести.
- Використання сервісів як Google Sign In[8], що дозволяють авторизуватися використовуючи вже існуючий акаунт Google.

2.1.4 Головні недоліки паролів

Метод паролів в першу чергу створений для використання користувачами і очікується, що користувачі будуть пам’ятати свій пароль – це робить паролі більш простими для атак повного перебору, де зловмисник перебирає всі можливі комбінації слів та знаків, які легко запам’ятати людина (справді, людина не може запам’ятати по-справжньому випадковий довгий пароль).

Навіть якщо людина може обрати пароль, що буде достатньо надійним (або використовувати біометрію) і зберігати його в надійному місці, метод паролів розраховує на чесність як автора сертифікату, що вказує публічний ключ до сервера (у випадку TLS), так і на чесність власне сервера. Наприклад, якщо сервер буде перезаписаний зловмисним працівником, то він може записати паролі на моменті їх отримання. Хоча при взаємодії з такими сервісами як Google вірогідність такого можна вважати умовно малою (навряд чи будь яка велика компанія допустить це), це робить підхід абсолютно не підходящим для ситуацій коли постійно треба комунікувати з сутностями, коду яких ми не можемо довіряти. Наприклад:

- У військовій сфері навіть якщо якщо підтверджено, що спілкуєшся з рацією колеги, не можна відправляти йому свій секрет адже логіку рації для запису твого пароля міг переписати зрадник або той, хто вкрав рацію.
- У фінансовій сфері не можна, щоб картки відправляли секрети POS-терміналам, адже зловмисний володар цього терміналу потім зможе виконувати транзакції від імені користувача.

2.2 Протоколи автентифікації засновані на підписах

2.2.1 Протоколи з використанням симетричного шифрування

Як вже було розглянуто в попередньому розділі головний недолік автентифікації за паролем є той факт, що доказувачу потрібно довіряти перевіряючому, адже той отримує приватний ключ у текстовому форматі. Зловмисний перевіряючий може запам’ятати пароль у себе і використовувати для автентифікації в інших сервісах.

Для того, щоб звільнитися від цієї обов’язкової довіри до перевіряючого треба звертатися до криптографії. Одним з найкраще зарекомендованих та найефективніших примітивів, що досі використовується в сучасній криптографії є симетричне шифрування.

Алгоритм симетричного шифрування - це криптографічний метод, який використовується для захисту конфіденційності даних, що передаються між двома

сторонами. У цьому методі використовується один і той же ключ для шифрування та дешифрування даних. Для шифрування повідомлення використовується спеціальний алгоритм, який перетворює звичайний текст у шифрований вигляд. Зашифроване повідомлення може бути передане по відкритій мережі, а тільки отримувач з ключем зможе розшифрувати його та отримати звичайний текст.

Одним з найбільш використовуваних алгоритмів симетричного шифрування є алгоритм AES (Advanced Encryption Standard). Існують AES-128, AES-192, AES-256, всі вони мають схожий алгоритм але підтримують ключі різних довжин (128, 192 та 256 біт відповідно). Окрім AES існує безліч інших алгоритмів симетричного шифрування, що не є наразі настільки популярними, але мали популярність в свій час:

- DES (Data Encryption Standard) – це один з найстаріших алгоритмів симетричного шифрування, який створювався ще в 1970-х роках. DES використовує 64-бітові ключі та 64-бітові блоки даних. Хоча він сьогодні вважається небезпечним через занадто короткі ключі, він ще досі використовується в деяких системах.
- Blowfish – алгоритм симетричного шифрування, який використовується для захисту конфіденційної інформації. Він був створений у 1993 році і використовує 64-бітові блоки даних та ключі розміром від 32 до 448 бітів.
- RC4 (Rivest Cipher 4) – алгоритм симетричного шифрування, який використовується для захисту інформації в Інтернеті. Він був створений у 1987 році і іноді використовується у бездротових мережах, протоколах TLS та WEP. Однак, він сьогодні вважається небезпечним через можливість атаки з використанням статистичної аналізи.

Будь який алгоритм симетричного шифрування можна перетворити в наступну схему автентифікації:

- На першому проці веріфаєр відправляє деякий запит з випадковим набором байтів до прувера.

- Потім прувер має відправити назад повідомлення зашифроване спільним приватним ключем.
- Якщо веріфаєр після розшифровки своїм приватним ключем отримає таке саме повідомлення, то він приймає твердження про те, що прувер володіє спільним приватним ключем. Інакше, не приймає.

Дуже важливо, щоб повідомлення, що відправлялося в першому запиті було кожний раз унікальним, адже інакше зловмисник, що спостерігає за комунікацією зможе підгледіти відповідь на цей запит (другий крок) та перевикористати його в наступних комунікаціях. Часто подібні повідомлення, що не мають ніколи повторюватись називають нонсом (nonce). Ця атака називаються атакою прослуховування (eavesdropping attack).

Алгоритми симетричного шифрування є швидкими, перевіреними часом алгоритмами. Найбільшим їх недоліком є те, що перед початком такого протоколу треба якось розподілити цей секретний ключ, що не є можливим при використанні незахищених каналів зв'язку. Це робить їх непрактичними для автентифікації в інтернет технологіях.

2.2.2 Протоколи з використанням асиметричного шифрування

Симетричне шифрування називається симетричним, адже ключ для розшифрування та шифрування є однаковими. На відміну від них у користувачів асиметричних алгоритмів шифрування існує пара ключів (sk, pk)– секретний і публічний ключ. Тому асиметричне шифрування називають також криптографією з відкритим ключем.

Майже завжди $pk = f(sk)$, тобто публічний ключ може бути обчислений за допомогою приватного ключа за невиликий (поліноміальний) час. Часто публічний ключ використовується для шифрування, а приватний ключ – для дешифрування. Такий підхід називають шифруванням публічним ключем. Але це не єдине, що дозволяє робити асиметрична криптографія. Інше завдання, котре вона може виконувати – цифрові підписи, де маючи підпис можна перевірити чи він був зроблений приватним ключем, що відповідає даному публічному ключу.

Цікаво, що будь який алгоритм цифрових підписів можна перетворити на наступний алгоритм автентифікації:

1. На першому кроці, прувер відправляє свій публічний ключ до перевіряльника.
2. На наступному кроці веріфаєр відправляє певне випадкове неповторюване число (нонс) до довідника.
3. Прувер за допомогою свого приватного ключа шифрує (підписує) це повідомлення. Та відправляє його до веріфаєра.
4. Веріфаєр може перевірити що підпис справді був зроблений приватним ключем, що відповідає до публічного ключа з крока (1).

Алгоритм по суті такий самий як і в видадку з симетричним шифруванням, але тут видно головну перевагу публічної криптографії над симетричною – те, що тепер у кожного користувача може бути свій унікальний ключ, публічний варіант якого він може розділяти між кількома сервісами, адже кожний сервіс буде знати лише публічний ключ і не зможе використовувати приватний ключ при взаємодії з іншими сервісами.

На відміну від алгоритмів симетричного шифрування алгоритми асиметричного шифрування найчастіше мають формальні доведення їх безпеки, що спираються на певні припущення. Найбільш відомий алгоритм шифрування публічним ключем RSA спирається на той факт, шо факторизація цілих чисел не може бути розв’язана за поліноміальний час. Велика кількість інших протоколів, як-от ECDSA (Elliptic Curve Digital Signature Algorithm) спираються на припущення про те, що задача пошуку дискретного логарифму в певній групі не може бути розв’язана за поліноміальний час. Відомо, що обидва припущення зазначені вище не є стійкими при існуванні квантових комп’ютерів. Наразі ведеться багато досліджень у сфері розробки алгоритмів асиметричної криптографії, що дозволять їй бути стійкою до атак квантових ком’ютерів.

Серед головних недоліків асиметричного шифрування – відносно довгий час роботи порівняно з протоколами симетричного шифрування, адже арифметичні операції займають більше часу ніж операції над бітами, котрими

зазвичай користуються алгоритми симетричного шифрування. Через це, найчастіше вони працюють у тандемі: за допогою протоколу публічного шифрування (шифрування за допомогою публічного ключа) обмінюються тимчасовим спільним ключем, що буде використовуватись вже для симетричного шифрування даних.

2.2.3 RSA та ECDSA

Найбільш популярними сучасними алгоритмами цифрових підписів є RSA та ECDSA. Всіх їх об'єднує властивість того, що можна перевірити чи належить підпис користувачу з відповідним публічним ключем.

Алгоритм RSA (Rivest-Shamir–Adleman, названий в честь авторів Рівеста, Шаміра та Адлмана) – це алгоритм, що використовує той факт, що легко знайти такі великі цілі числа e, d, n, такі що, для будь-якого числа $0 < m < n$, $(m\hat{}e)\hat{}d = m\,(mod\,n)$. В цій роботі детально не буде описуватися процедура генерації ключів та підпису за допомогою RSA. Відмітимо тільки, що зі схеми вище e буде приватний ключом, а d – публічним ключем. При цьому $n = pq$, де p і q – великі прості числа. Число n – називають модулем RSA і він є частиною публічного ключа. Безпека RSA гарантується за рахунок того, що число n неможливо факторизувати, всі найкращі атаки на RSA робляться саме через факторизацію числа n.

Загалом, хоча і робочим, RSA вважається алгоритмом неефективним і дещо застарілим. Наприклад, найкраща атака на ідеально імплементований RSA є використання методу решета числового поля для факторизації числа n[9]. Час роботи цього алгоритму обчислюється за наступною формулою:

$$T = e^{(c + O(1))n^{1/3}log^{2/3}n}$$

Тобто, для забезпеченння безпеки в 256 біт (тобто змусити метод решета числового поля працювати за 2^256 операцій) модуль RSA має мати розмір в 15360 бітів. Це є достатньо великим розміром даних, особливо в порівнянні з іншими алгоритмами асиметричної і симетричної криптографії, яким зазвичай

для такого рівня безпеки потрібно від 256 до 512 біт даних. Ідеальний, тобто повністю коректний з точки зору протоколу RSA дуже важко імплементувати. Існує велика кількість атак, які можуть з'явитися через незначні помилки в програмній реалізації, тому багато практичних спеціалістів просять не використовувати RSA за можливості[10], а якщо обійтися неможливо, то використовувати виключно перевірені роками імплементації.

В свою чергу, ECDSA (Elliptic Curve Digital Signature Algorithm) є більш новим та безпечним алгоритмом цифрового підпису. ECDSA є надбудовою над простішим алгоритмом DSA, який працює на групою цілий чисел за модулем. ECDSA ж використовує групу еліптичних кривих. Наразі DSA використовується досить рідко і надається перевега саме ECDSA, через те, що еліптичні криві є більш безпечними. Еліптичною кривою над полем F — є множиною точок в полі F, що задовольняють наступному рівнянню:

$$y\text{^}2 + axy + by = x\text{^}3 + cx\text{^}2 + dx + e$$

Значення параметрів a, b, c, d, e залежать від кожної конкретної кривої. Є різні стандарти кривих таких як secp256k1[11] та інші. Кожна еліптична крива утворює групу, на якій визначена операція додавання.

Для алгоритму потрібно обрати еліптичну криву на якій будуть проходити операції C та та так звану базову точку G, що генерує великі підгрупу простого порядку. Порядок точки G позначимо як n.

Нехай у нас є дві сутності – Аліса та Боб. Аліса хоче підписати повідомлення, а Боб хоче перевірити, що підпис справді належав Алісі. Приватний ключ Аліси – скаляр $p \in F$, де F – це поле, над яким будується еліптична крива. Публічним ключем Аліси буде – $P = p * G$.

Якщо Алісі потрібно підписати повідомлення m, то їй потрібно виконати наступні кроки:

1. $h = H(m)$, де H – криптографічна геш-фукнція.
2. Нехай z – це h обрізаний до кількості бітів в n.
3. Треба вибрати випадкове число k (яке називають нонсом), в діапазоні від 1 до $n - 1$.

4. Обрахувати точку на еліптичній кривій $(x_1, y_1) = k * G$.
5. Порахувати $r = x_1 \, mod \, n$. Якщо $r = 0$, то треба спробувати інше k і повторити кроки 3-4.
6. Обрахувати $s = k^{-1} * (z + r * p)$.
7. Підписом буде пара – (r, s) або $(r, (-s) \, mod \, n)$.

Той факт, що у одного й того ж повідомлення може бути два підписи (при тому, що можна вивести одне легко з іншого) називається властивістю пластичності підпису (в анлійській літературі malleability). Вона часто є небажаною, тому у багатьох протоколах, як наприклад Ethereum[12], дозволяється лише підпис, де s є у певному довзволеному діапазоні, для того, щоб не можна було перевикористати підпис $(r, (-s) \, mod \, n)$. За бажанням можна додати цю вимогу до протоколу, але тоді вона має додатково перевірятися веріфаєром при отримання підпису.

Процедура верифікації є наступною. В першу чергу Боб має перевірити, що публічний ключ Аліси є валідним публічним ключем: не є нульовою точкою або нескінченністю, належить кривій, належить підгрупі генератора G.

Після того, що Боб впевнився в правильності публічного ключу, робиться наступне:

1. Перевіряється, що r і s належать інтервалу від 0 до $n - 1$.
2. Обчислюється h та z так само як в кроках підпису.
3. Обчислюється $a = z * s^{-1}$ та $b = r * s^{-1}$.
4. Знаходиться $(x_2, y_2) = a * G + b * P$.
5. Якщо (x_2, y_2) не є ані нулем ані нескінченністю і $r = x_2$, то підпис вважається правильним.

Коректність алгоритму доводиться відносно тривіально, але власне безпека алгоритму залежить від умови, що задачу дискретного логарифму неможливо розв’язати швидко, тобто за поліноміальний час. Цей алгоритм є набагато більш

практичний ніж RSA, для того, щоб досягнути захисту в 256 біт потрібно лише 512 біт даних.

Критичним для безпеки є неповторність нонсу k. Якщо нонс повториться два рази, це дає змогу обчислити приватний ключ користувача. Відомі випадки взлому ігрових консолей PlayStation 3 саме через такий недолік в імплементації[13].

2.2.3 Безпека протоколів автентифікації, що використовують асиметричну криптографію

Загалом, якщо обирати достатньо великі параметри безпеки (тобто достатньо великий модуль для RSA або достатньо велике поле для кривої в ECDSA), то використовувати протоколи автентифікації, що використовують асиметричну криптографію досить безпечно. Це проколи, які найчастіше використовуються в інтернет технологіях через перевіреність і практичність.

Проте, варто відмітити, що протоколи автентифікації, що засновані на асиметричній криптографії є вразливими до атак з вибраним текстом (“chosen plaintext attack”). На практиці зловмисник може отримати результати шифрування до будь-якого тексту якого він хоче. Приватні ключі користувачів можуть бути дістані завдяки статистичним методам або можливо якимось властивостям алгоритмів. Проте, звичайно, найчастіше не є можливим відновити повний приватний ключ.

Для деяких алгоритмів, наприклад RSA, існування атаки вибраного шифрування (“chosen ciphertext attack”, тобто атаки де зловмисник може знаходити розшифрування до будь яких закодованих текстів) є питанням відкритим[9].

Також автентифікація через підписи якщо неправильно побудована може приносити користь для зловмисного перевіряючого, наприклад у випадку блокчейну, зловмисний перевіряючий може попросити в якості запиту для підпису використати певну транзакцію для цього блокчейна і відправити її, потенційно вкравши гроші у користувачів. У загальному випадку це називається атакою

людини посередині (man-in-the-middle attack), коли певний користувач може стати між комунікаціями інших користувачів і отримати доступ до потрібних даних. У випадку алгоритмів автентифікації через підпис атака може виглядати так: нехай Аліса хоча довести Бобу, що вона володіє приватним ключем до свого публічного ключа. В свою чергу Боб хоче автентифікувати транзакцію від імені Аліси до іншої людини Джона (в реальному житті Боб може хотіти оплатити якісь покупки в інтернеті через картку Аліси).

- Аліса відправляє Бобу свій публічний ключ. Боб відправляє Джону публічний ключ Аліси.
- Джон відправляє послідовність байт, що має бути підписана ключем Аліси. Боб її перенаправляє на Алісу.
- Аліса підписує повідомлення і відправляє до Боба. Боб авторизує транзакцію від імені Аліси до Джона.

Можна вберегтися від подібного роду атак якщо повідомлення, що підписуються мають різний формат для кожного перевіряльника, тобто якби наприклад випадкові повідомлення, що надходять від Боба починалися умовно зі слова “Боб”, а повідомлення, що йдуть від Джона починаються від слова “Джон”, то підпис не можна було б перевикористати. Подібні рішення є популярними у сфері блокчейн, то вводять правила, що для безпеки користувачів повідомлення що авторизують не транзакції мають починатися з певної послідовності байт. Наприклад, у блокчейні Ethereum використовують префікс "\x19Ethereum Signed Message:\n"[19] для всіх повідомлень.

3 ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ ЧЕРЕЗ ДОВЕДЕННЯ З НУЛЬОВИМ РОЗГОЛОШЕННЯМ

Хоча протоколи автентифікації з використанням асиметричної криптографії при правильній організації часто є достатньо добрими для використання на практиці, вони можуть показувати додаткові данні, окрім того, що приватний ключ належить користувачу. В першу чергу ці додаткові дані – це власне підписи до відповідей перевіряючого. Як вже було сказано в попередньому розділі, ці дані створюють окремі вектор для атаки, як атаку з вибраним текстом, так і атаку людини посередині. Варто також відмітити, що протоколи, що використовують підписи дозволяють історично відслідковувати, щоперевіряльник взаємодіяв з доказувачем (наприклад якщо текст для підпису був підібраний як функція від часу). Часом така властивість є не бажаною.

Виникає потреба в алгоритмах автентифікації, що не дозволяють перевіряючому дізнатися нічого більше, ніж той факт, що довідувач володіє секретом. Такі докази називаються доказами з нульовим розголошенням (в англійській літературі zero knowledge proofs). Такі алгоритми автентифікації не мають залишати часового сліду та вони, при правильній побудові, можуть бути абсолютно стійкими до атак людини посередині.

3.1 Доведення з нульовим розголошеннм

Дамо визначення доведенню з нульовим розголошенням. Інтерактивний протокол є доказом з нульовим розголошенням, якщо він має наступні три властивості:

- Повнота (англ. Completeness). Тобто якщо твердження, що доводиться є вірним, то перевіряльник зажди прийме це твердження.

- Надійність (англ. Soundness). Тобто якщо твердження, що доводиться не є правдивим, перевіряючий його не прийме з дуже високою ймовірністю (ймовірність, що прийме настільки мала, що можна знехувати).
- Нульове розголошення (англ. Zero knowledge). Якщо твердження є правдивим, то перевіряч не дізнається нічого нового окрім того що твердження є правдивим.

Найцікавіше є те, як можна визначити “знання” та фразу “не дізнатися нічого нового”. Неформально кажучи, можна це визначити як те, що перевіряч сам міг би провести такий самий діалог, але не маючи доступу до прувера. Тобто навіть якщо якісь дані формально не зберігаються в пам’яті, то принаймні вони можуть бути обчислені за поліноміальний час.

Формальне визначення ж визначення нульового розголошення є досить складним і вимагає створення третьої особи S, яка називається симулятором. Вважається що протокол є протоколом з нульовим розголошенням, якщо для кожного перевіряючого V існує такий симулятор S, який знаючи наперед усі випадкові події (random coins), які може робити V може повністю відтворити діалог між V та P. Ці випадкові події виражаються через допоміжний рядок z[15].

3.1.1 Простий приклад доведень з нульовим розголошенням

Відомим прикладом доведень з нульовим розголошенням є те як можна довести людині, що не розрізняє кольори те, що дві кульки мають різний колір. Це можна виконати за допомогою наступного протоколу[16]: перевіряльником буде людина, що не розрізняє кольори, а доказувачем буде людина, що їх розрізняє. Перевіряльник бере до себе на руки два шари, ті що йому здаються одного кольору, а доказувачу різного.

Далі k разів перевіряльник випадково мішає шари у себе в руках (так щоб доказувач не бачив в якій руці який шар) дістає один шар з руки і питає який він має колір. Якщо доказувач хоча б раз помилився, то твердження вважається невірним.

Якщо всі k разів доказувач правильно відповів, то перевіряльник приймає твердження к вірне. Якщо ці шари не мають кольору, то вірогідність того, що перевіряльник був обманутий 2^{-k}, що відповідає вірогідності навмання обирати правильні шари кожен раз.

Даний протокол виконує всі 3 пункти визначення доведення з нульовим розголошенням – у разі правильного твердження воно завжди приймається, у разі неправильного воно з похибкою, що можна знехтувати при, наприклад, k=256 воно буде відкинуте. Також, перевіряючий міг би цей сам протокол повторити сам, адже знає правильну відповідь на кожний з запитів.

3.1.2 Доведення знань з нульовим розголошенням

Варто відмітити, що означення доведень з нульовим розголошенням нічого не говорить про те, що довідник мусить взагалі знати будь-який секрет. Формально кажучи, доведення з нульовим розголошенням доводить, що деяке число x належить якійсь NP мові L. Іноді докази можуть бути тривіальними. Наприклад у нас є мова $L = \{x \mid \exists w\colon g^w = x\,(mod\,p)\}$ при відомих p, , та g, де g– генератор групи Z_{p^*}. Тоді можна тривіально довести, що x справді належить цій мові, адже оскільки g є генератором тієї групи, то справді існує деяке w, таке що $g^w = x\,(mod\,p)$. Пруверу нічого й доводити не треба. Але формально це теж буде доведенням з нульовим розголошенням адже всі умови за визначенням виконуються.

Однак в алгоритмах автентифікції критично, довести, що користувач знає свій секрет. Для прикладу вище недостатньо лише довести, що x належить цій мові, треба ще й довести, що доказувач знає такий w, що $g^w = x\,(mod\,p)$. Як визначити знання? Вважається, що доказувач “знає” секрет, якщо він може його обчислити за поліноміальний час.

Для більш формального визначення треба ввести поняття екстрактору знань K. Екстрактор знань K – це алгоритм, що працює за ймовірнісний поліноміальний час і, якщо йому дати наперед усі випадкові події (random coins), які може робити

P, а також можливість взаємодії з P як оракулом, то може обчислити секрет (в англомовній літературі witness) за поліноміальний час. Варто зазначити, що екстрактор знань при роботі з P (так само як і симулятор при роботі з перевіряльником V) може “повертатися в часі”, тобто може після виконання P певних операцій відкатити їх та виконати якісь інші замість попередніх.

Отже, протокол є доведенням знань з нульовим розголошенням (zero knowledge proof of knowledge), якщо він є доведенням з нульовим розголошенням і при цьому існує екстрактор знань K, що для будь якої успішної взаємодії (V, P, x), де P зміг довести V, що знає секрет до публічних даних x (приватний ключ, тощо) може, за даними випадковими виборами P та P як оракулом може обчислити цей секрет з такою самою ймовірністю як P може переконати V у знанні секрету[15].

Виникає питання – чи існують нетривіальні приклади доведень з нульовим розголошенням, що не є доведеннями знань з нульовим розголошенням? Таких прикладів є дуже мало, але є приклади доведень з нульовим розголошенням, для яких не доведено, що вони є також доведеннями знань з нульовим розголошенням[17].

3.2 Протоколи Фіата-Шаміра та Фейге-Фіата-Шаміра

3.2.2 Протокол Фіата-Шаміра

Одним з перших практичних схем автентифікації була схема Фіата-Шаміра[18]. Вона було створена у 1987 році Амосом Фіатом і Аді Шаміром. Схема використовувала той факт, що знайти корінь квадратний числа x за модулем $n = pq$, де p і q великі прості числа неможливо зробити за поліноміальний час. На відміну від RSA, цей модуль n є загально використаним і є розрахунок на “довірений центр” – третю особу, який створює для кожного учасника його приватні та публічні ключі. Довіряємо центру в тому, що він не буде зберігати ані p, ані q, а також не буде зберігати приватні ключі жодних з учасників. На практиці ж, якщо цим довіреним центром є банк, що видає картки, то він може зберігати у

себе p і q (щоб була можливість випускати картки). Число n має бути заздалегідь відоме всім учасникам.

Нехай $S_1^2(mod\ n),\dots S_k^2\ (mod\ n)$ – публічний ключ користувача. Приватним ключем його будуть числа $S_1, \dots, S_k$. Доказувач A хоче довести перевіряльнику B, що володіє відповідними приватними ключами до свого публічного ключа. Алгоритм складається з t наступних кроків:

1. А вибирає випадкове число r і відправляє $x = r^2\ (mod\ n)$ до В. Важливо, щоб r ніколи не повторювалось. При великих n (більше 256 бітів) просто вибрати рівномірно випадкове число є достатнім, адже шансом повторитися тоді можна знехтувати.
2. Вибирає випадковий вектор $e_1,\dots, e_k$. Та відправляє його до А.
3. А відправляє до В $y = r * \prod_{e_j=1} S_j^{-1}$.
4. В перевіряє, що $x = y^2 * \prod_{e_j=1} S_j^2$.

Якщо всі t разів протокол був пройдений успішно, то В приймає те, що А володіє відповідними приватними ключами.

В оригінальній роботі Фіат та Шамір описують алгоритм на прикладі кредитних карток, де картка має зберігати приватні ключі і доводити, що ними володіє[18]. Вони пропонували наступний алгоритм створення публічних ключів в практичних умовах. Нехай I – це ідентифікатор користувача (або власника картки), належність якого користувач хоче довести до перевіряльника. В цей рядок I можуть входити такі дані як номер його акаунту, ім’я, прізвище, адреса тощо.

1. На першому кроці обчислюється деяка кількість $v\_j = f(I, j)$, де f – це деяка одностороння фукнція, тобто така, що неможливо обчислити

f^{-1} за поліноміальний час. Найчастіше у якості f виступає якась криптографічно безпечна геш-фунція.

2. Треба обрати деякі k значень j, такі що v_j – є квадратичним залишком, тобто рівняння $x^2 = v_j \, (mod\, n)$ має розв'язок. Нагадаємо, що оскільки цей процес відбувається у довіреному центрі, що знає p,q, такі що $n = pq$, то він може легко порахувати це число x. Натомість інші учасники такого зробити не зможуть. Позначимо $S_j^2 = v\_j$.
3. Тоді на картці будуть записуватися числа I, k індексів відповідних v_j, а також S_j^{-1} (тобто $\sqrt{v_j}^{-1} \, (mod\, n)$ для них).

Тоді публічним ключем картки буде I та відповідні k індексів. Перевіряючий, отримавши I та ці індекси зможе порахувати $f(I, k)$. S_j^{-1} будуть використовуватися карткою для протоколу доведення. Існують деякі рекомендації, що можуть зробити алгоритм більш безпечним на практиці, наприклад конкатенація кількох випадкових байтів до I.

3.2.3 Безпека протоколу Фіата-Шаміра

Доведемо, що це є протокол доведення знань з нульовим розголошенням. По перше, доведемо, що це є протокол з нульовим розголошенням:

Умова повноти справджується. Справді,

$$y^2 * \prod_{e_j=1} S_j^2 = r^2 * \prod_{e_j=1} S_j^{-2} * \prod_{e_j=1} S_j^2 = r^2 = x$$

Отже, протокол завжди завершиться успішно, якщо доказувач знає секрет.

Доведення безпеки та нульового розголошення є більш складним, тому в цій роботі буде наведено лише їх нарис:

Умова безпеки на загальному рівні доводиться наступним чином. Нехай А не знає секрету і не може за поліноміальний час знайти обчислити квадратний

корінь по модулю n для будь якого добутку $\prod_{j=1}^{k} S_j^{2c_j}$, де c_j може бути 1, 0, або мінус 1.

Тоді А може тільки вгадати вектор e_i, і відправити правильний $x = y^2 * \prod_{e_j=1} S_j^2$. Але вірогідність того, що він це може зробити для одного раунду протоколу 2^{-k}, а для всього протоколу 2^{-kt}. Можна цю вірогідність збільшити, якщо він буде обирати такий x, що для великої кількості чисел може обчислити наступний вираз:

$$r = \sqrt{\frac{x}{\prod_{e_j=1} S_j^2}}$$

для принаймні двох чисел r_1, r_2, для двох різних векторів e_1, e_2. Тоді відношення $\frac{r_1}{r_2}$ буде у формі $\prod_{j=1}^{k} S_j^{2c_j}$, де c_j може бути 1, 0, або мінус 1, отже отримали протиріччя.

Умову нульового розголошення інтуїтивно можна побачити, що ніякої інформації про секрет S_j не видається. Більш формальне доведення з побудовою симулятора перевіряючого наведено у відповідній роботі Фіата та Шаміра[18]. Варто відмітити, що побудований у їх доказі симулятор працює за $t * 2^k$. Тобто для того, щоб умова нульового розголошення справджувалася потрібно, щоб $k = O(loglogn)$, щоб швидкість алгоритму симулятора була поліноміальною відносно кількості бітів числа n.

Остання умова, яку варто довести – те, що це протокол, що доводить знання, тобто, що існує екстрактор знань, який може використовувати А як оракул і має доступ до його випадкових виборів і він може для з такою самою ймовірністю обчислити секрет, як з вірогідністю, що А зможе довести В, що у нього є секрет.

Такий екстрактор можна легко побудувати. Нехай кроці 2 обираємо кортеж $E_1 = (e_{11},..., e_{1k}) = (0,..., 0)$. Тоді на запит на кроці 3 отримаємо наступне:

$$y_1 = r * \prod_{e_{1j}=1} S_j^{-1} = r.$$

Далі відкатуємо протокол, і ще раз зробимо запит для нього з кортежем $E_2 = (e_{11},..., e_{1k}) = (0,... 0, 1)$ (тобто з таким самим кортежем, але змінений останній біт). Тоді отримаємо $y_2 = r * \prod_{e_{1j}=1} S_j^{-1} = r * S_k^{-1}$. $\frac{y_2}{y_1} = S_n^{-1}$. За допомогою розширеного алгоритму Евкліда можна відновити S_k.

Для порівняння, перевіряч В, не зможе виконати ті самі дії, адже А буде завжди обирати випадкове нове число r.

3.2.4 Протокол Фейга-Фіата-Шаміра

Не дивлячись на те, що алгоритм Фіата-Шаміра не видає ніяких даних про секрет, він все таки видає деякі дані. А саме те, що числа, з яких складається публічний ключ є квадратичними залишками. Хоч з практичної точки зору це може здаватися дрібницею, проте формально перевіряч не зміг би це перевірити, якби не взаємодія з доказувачем, тож виникає питання проте наскільки таке розголошення є нульовим.

В своїй роботі 1988 році Фейг, Фіат та Шамір виклали ідею модифікованого алгоритму, що вирішує цю проблему[19]. Алгоритм виглядає наступним чином.

Довірений центр обирає два прості числа p і q, що мають вигляд $4r + 3$, і вибирає модуль $n = pq$. Це додаткове обмеження на формат p і q дає змогу стверджувати, що $n - 1\ (mod\ n)$ не є квадратичним залишком, і при цьому має символ Якобі 1. Нагадаємо, що символом Якобі числа a по модулю $n = pq$ називається наступний вираз:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p}\right)\left(\frac{a}{q}\right)$$

Де $\left(\frac{a}{x}\right)$ – символ Лежандра числа a по модулю x, тобто $\left(\frac{a}{p}\right)$ дорівнє 1, якщо а є квадратичним залишком по модулю p, мінус 1, якщо не є, і 0 якщо а дорівнює 0.

З визначення слідує, що якщо число є квадратичним залишком по модулю n, то він має символ Якобі 1. Але обернене твердження не є правдивим: число може не бути квадратичним залишком, хоча мати символ Якобі 1 (наприклад, якщо число Лежандра для x модулю p і q дорівнює мінус 1)..

Нехай S_1, ..., S_k – це секрет користувача А. Тоді його публічним ключем буде I_1, ..., I_k, де $I_j = \pm \frac{1}{S_j^2}$. Тут $\pm$ означає випадковий вибір між $\frac{1}{S_j^2}$ або $-\frac{1}{S_j^2}$). Завдяки такому вибору ми отримаємо, що I_j може бути будь-яким числом з символом Якобі 1.

Далі протокол відбувається схожим чином на оригінальний протокол Фіата-Шаміра:

1. А обирає випадковий r, і відправдяє $x = r^2$ або $x = -r^2$.
2. В відправляє випадковий вектор бітів $E = (e_1,..., e_k)$.
3. А відправляє $y = r * \prod_{e_j=1} S_j^{-1}$.
4. В обчислює $D = y^2 * \prod_{e_j=1} I_j$. В приймає твердження, якщо x дорівнює D або мінус D.

3.2.5 Безпека протоколу Фейга-Фіата-Шаміра

Всі ознаки безпеки протоколу Фейга-Фіата-Шаміра доводяться аналогічно до протоколу Фіата-Шаміра. Головна різниця – це те, що перевіряючий не дізнається чи були I_j квадратичними залишками чи ні. Можна стверджувати, що він дізнається, що $I_j = \pm q_j$, де q_j – це деякий квадратичний залишок. Але для

того модуля n, що ми вибрали справджується властивість, що множина QR та J_1 є однаковими, де множини QR та J_1 визначені наступним чином:

$$QR = \{k: k = q \text{ або } k = -q, \text{ де } q = \text{квадратичний залишок}\}$$

$$J_1 = \{k: \left(\frac{k}{n}\right) = 1\}$$

Але те, що $\left(\frac{I_j}{n}\right) = 1$ він може перевірити без взаємодії з доказувачем А за поліноміальний час[20].

3.2.6 Практичність алгоритмів Фіата-Шаміра та Фейга-Фіата-Шаміра

Оскільки $k = O(loglogn)$, а рівень безпеки після t раундів протоколу буде 2^{kt}, для того, щоб був великий рівень безпеки (2^256) треба мати модуль принаймні 15360 бітів (тобто $k = log_2(15360) \approx 14$) і потрібно виконати принаймні $\frac{256}{14} \approx 18$ раундів протоколу. Це є досить великою кількістю в порівняння з багатьма іншими протоколами.

У оригінальній роботі стверджувалося, що рівня безпеки більше ніж 2^{30} ніколи не буде потрібно, адже, наприклад, навряд чи хтось буде в аеропорту показувати підроблені документи, якщо шанс правильної відповіді один на мільярд (тим паче, що другої спроби вже не буде)[18]. Алгоритм був опублікований у 80-х роках, тому не міг передбачити стрімкий ріст інтернет-технологій, де найчастіше є можливість спробувати авторизуватись декілька разів на набагато швидших комп’ютерах.

З іншого боку цей алгоритм має гарну властивість того, що публічний ключ можливо явно виводити з даних про користувача (тобто завжди можна довести, що у тебе є саме такі дані, а не абстрактний секрет до математичної задачі). Загалом протокол є найчастіше уживаний в ситуаціях, коли кількість можливих хибних спроб не є великою, наприклад в біометричних картках.

3.3 Протокол Шнора

Протокол Шнора[21] а є більш вживаним та ефективним протоколом автентифікації, ніж протокол Фіата-Шаміpач и Фейге-Фіата-Шаміра. Цей протокол використовує твердження про складність пошуку дискретного логарифму в вибраній групі.

Нехай дано велике просте число p, g – генератор Z_{p^*}. p і g є публічними і всім відомими. Нехай s – це секрет користувача, а $h = g^s$ – це його публічний ключ. Задача користувача довести, що він володіє s. Позначимо через A користувача, а через B перевіряльника. Протокол складається з наступних кроків:

1. A вибирає випадкове число $r \in Z_{P^*}$. Та відправляє g^r до B.
2. B відправляє випадкове число e до A.
3. A повертає $y = r + es$. Якщо $g^y = g^r * h^e$, то B приймає це твердження.

Алгоритм можна модифікувати для того, щохб використовувати будь-яку групу, де пошук дискретного логарифму є важкою задачею. На практиці., найчастіше алгоритм імплементований з використанням еліптичних кривих.

3.3.1 Безпека протоколу Шнора

Протокол Шнора є доведенням знань з нульовим розголошенням при умові чесного верифікатора.

- Повнота доводиться очевидно.
- Безпеку можна довести наступним чином. Нехай дискретний логарифм неможливо ефективно розв'язати, при цьому існує якийсь алгоритм, що може ефективно з високою ймовірністю може правильно пройти протокол. Давайте зафіксуємо деякий x. Тоді нехай існують два e_1, e_2 для який протокол був прийнятий, тоді: $g^{y_1 - y_2} = g^{(e_1 s - e_2 s)}$. Тоді $y1 - y2$ – це дискретний логарифм $g^{(e_1 s - e_2 s)}$. Тоді $(y1 - y2)/(e1 - e2)$ – дискретний логарифм g^s. Виходить, що якщо хтось може ефективно "обманювати" цей

протокол, то він також може ефективно обчислювати дискретний логарифм, отже маємо протиріччя.

- Нульове розголошення. Даний протокол не є протоколом з нульовим розголошенням у повному сенсі цього слова. Насправді перевіряючий отримає кожен раз розв'язок задачі про дискретний логарифм для $g^r * h^e$ (хоча користь цих знань для перевіряючого оцінити важко). Наразі не існує ніяких алгоритмів, що дозволяють якось отримати секрет завдяки поліноміальній кількості таких алгоритмів, проте і не існує доведення, що таких немає[5].
- Доведення знань. Нехай вірогідність пройти протокол успішно у деякого доказувача t. Тоді, з вірогідністю t^2 він пройде успішно два таких протоколи. З іншого боку, як вже було показано в частині про безпеку, два рази пройти протокол з різними e достатньо, щоб віднайти s. Таким чином екстрактор знань може обчислити s з ймовірністю поліноміально залежною від ймовірності А пройти протокол.

3.3.1 Використання протоколу Шнора

Протокол Шнора є набагато більш ефективним за протоколи Фіата-Шаміра та Фейге-Фіата-Шаміра. Прикладом можливого використання протоколу Шнора – розроблений Стефаном Брандом digital cash[22]. Суть протоколу в тому, що він дозволяє виводити гроші з банку для офлайн закупів, при цьому не є відомим хто саме вивів ці кошти. Потім користувач може використовувати ці гроші в магазинах, але якщо користувач захоче використати одні й тіж гроші два рази (тобто виконати подвійне використання грошей, в англійській літературі double spend attack), то банк зможе обчислити особистіть цього користувача, тобто банк зможе обчислити особистість тільки тих користувачів, які порушують правила. Деталі протоколу можна прочитати в оригінальній роботі[22]. Заналом, він використовує дещо модифікований протокол Шнора, де числа r з кроку (1) та e з кроку (2) є детермінованими відносно даних про монетку та транзакцію). Якщо

користувач виконає таку версію протокола Шнора два рази для однієї і тієї монетки, то стане можливим обчислити приватний ключ користувача, що намагався одну й ту саму монету використати кілька разів. Ідея була дуже новаторською в свій час, але не практиці не прижилася через те, що є досить ризикова по відношенню до помилок користувача, наприклад навіть якщо подвійне використання було помилковим, то користувачу треба довіряти банку, що він не буде ніяк зберігати її приватний ключ, по суті деградуючи протокол до безпеки подібної до автентифікації з паролем.

3.4 Атака людини посередині та способи її уникнути

Навіть якщо протокол є протоколом з нульовим розголошенням, він не завжди може уникнути атаки людини посередині. Суть цієї атаки була пояснена в розділі про безпеку схем автентифікації, що використовують асиметричну криптографію.

Варто зазначити, що в тій формі, що описана в попередніх параграфах як алгоритм Шнора, так і алгоритм Фейге-Фіата-Шаміра є вразливими до атак людини посередині, коли перевіряч може використовувати взаємодію з користувачем, щоб представити себе від його імені до якогось іншого верифікатора. Для автентифікації, що заснована на підписах найчастішим розв’язком цієї проблеми є включення якогось ідентифікатора перевіряючого у підпис. Для протоколів автентифікації зазначених вище, рішенням проти атак виду людини посередині, що найчастіше пропонується – ця виставляння максимального часу відповіді[23], тобто зробити так, щоб зловмисний веріфаєр не встигав взаємодіяти з іншими перевіряючими. Це є прийнятним у випадку біометричних карток, адже відстань до сканера завжди є малою і можна справді задати деякий ліміт часу, але для інтернет взаємодії це не підходить, адже комунікація через мережу інтернет сильно залежить від швидкості доступу до неї.

3.4.1 Сигма-протоколи та OR-доведення

Одним із способів вирішення проблеми атаки людини посередині

використання властивостей сигма-протоколів[5].

Нехай пруверу P треба довести перевіряючому V, що він знає секрет s до якогось NP-відношення (наприклад при публічному $x = g\hat{}s$, s – це дискретний логарифм числа x). Сигма-протокол, це протокол, який має наступну трикрокову форму:

1. Прувер P відправляє до веріфаєра V повідомлення a.
2. V відповідє t-бітним повідомленням e.
3. Доказувач P повертає z. Після цього V вирішує, прийняти чи не прийняти твердження в залежності від значень a, e, z, x.

   Також від сигма-протоколу вимагаються наступні властивості:

   - Якщо провести дві успішні взаємодії (a, e, z) та $(a,\ e',\ z')$, при $e \neq e'$, можливо обчислити s.
   - Існує поліноміальний симулятор M, що може з даним x та e відтворити розмови (a, e, z), що підтверджуються з такою самою ймовірністю як і те, що перевіряч прийме цю взаємодію. Це називаються спеціальне нульове розголошення при чесному перевіряючому.

Зокрема, протокол Шнора розглянутий в попередніх розділах є сігма-протоколом. Визначимо OR-доведення, як доведення, що доводить, що доказувач знає секрет або для x АБО для x'. За допогомою такого примітиву як сігма-протокол можливо побудувати OR-доведення[5]. На прикладі з протоколом Шнора, якщо $x_1 = g^{s_1}$, а $x_2 = g^{s_2}$, то OR-доведення довело б, що перевіряч знає s_1 або він знає s_2, при цьому неможливо за поліноміальний час розрізнити який саме секрет знає доказувач.

За допомогою OR-доведення можна модифікувати протокол Шнора, де тепер прувер доводить не те, що він знає секрет до свого публічного ключа, а знає секрет до свого публічного ключа або секрет до публічного ключа перевіряючого. Оскільки прувер не може знати приватний ключ веріфаєре, то веріфаєр приймає таке доведення.

Цей протокол є стійким до атаки людини посередині. Позначимо як А чесного доказувача, через Е нечесного перевіряльника, який хоче через взаємодію з А представити себе під виглядом А до чесного перевіряльника В. Позначимо через P_A, P_E, P_B їхні публічні ключі відповідно.

Тепер від час взаємодії між А та Е, А замість того, щоб доводити, що він має секрет до P_A, він буде доводити, що він знає секрет для P_A або для P_E. В свою чергу, тепер під час взаємодії Е та В, якщо Е хоче представити себе як А, то йому варто показати доведення, що він знає приватний ключ або до P_A або до P_B, але таке доведення може створити тільки або В або А. Тож Е не зможе представити себе як А при взаємодії з В.

Більше того, на відміну від протокола Шнора цей протокол є протоколом з нульовим розголошенням навіть при зловмисному перевіряючому. Інтиїтивно кажучи, Е нічого не дізнається нічого з цього протоколу, адже створити доведення, що він знає P_A, або P_E, Е може й сам, адже він знає секрет для P_E[5].

3.5 Алгоритм швидкої автентифікації 3 (ША3)

Анісімовим А. В. було розроблено протокол швидкої автентифікації за два раунди взаємодії під назвою ША3 (алгоритм швидкої автентифікації 3)[24]. Це алгоритм, що використовує протокол Шнора за основу, але використовую ідею довіреного центру як в алгоритмі Фіата-Шаміра.

Схема алгоритму наступна:

Позначимо через Р доказувача, а через V перевіряючого. Секретом доказуючого буде пара (s_1, s_2), а його публічним ключем буде пара $(h_1, h_2) = (g^{s_1}, g^{s_2})$. Перед взаємодією довірений центр записує до доказувача публічний ключ $h = g^s$, $A[0]$,..., $A[k]$, де $A[0] = s_1$, а $A[i] = H(s + A[i-1])$. Натомість в усі засоби, що будуть мати змогу виконувати роль перевіряючого записують $B[0]$,..., $B[k]$, де $B[i] = g^{A[i]}$.

Далі автентифікація виконується за два раунди:

- V обирає три випадкові числа i, j, та e, та відправляє їх P.
- P обчислює $y = A[i] + e * s1 + j * s2$. І відправляє y до V.

Якщо $g^y = B[i] * h_1^e * h_2^j$, то твердження приймається.

У цьому алгоритмі V є довіреною особою, і має утверджуватись центром. Справді, якщо V не є довіреним, то він може для одного й того ж учасника при двох взаємодіях використати (i_1, j_1, e_1)та $(i_1, j_1 + 1, e_1)$ і таким чином обчислити $s2$.

З іншого боку, якщо довіряти V і використатовувати асиметричне шифрування публічним ключем V (по суті зробивши значення $A[i] + e * s1 + j * s2$ необчислюваним для сторони, що підглядає за комунікацією), то цей алгоритм є стійкий до атаки людини посередині та дозволяє за меншу кількість повідомлень автентифікуватися. Це робить алгоритм швидшим за попередні при взаємодії через мережу інтернет, але за умови, що публічний ключ V є заздалегідь відомим.

3.5.1 Можливі покращення за допомогою евристики Фіата-Шаміра

Для того, щоб зробити алгоритм стійким по відношенню до недовіренного перевіряючого V, варто змусити V обирати i, j, e випадковим чином. Для цього, принаймні якась частина випадковості має йти від P. З іншого боку, навіть якщо i, j, e завжди випадкові, то достатньо трьох повторів i, для того, щоб утворилася наступна система рівнянь:

$$y_1 = A[i] + e_1 * x_1 + j_1 * x_2$$

$$y_2 = A[i] + e_2 * x_1 + j_2 * x_2$$

$$y_3 = A[i] + e_3 * x_1 + j_3 * x_2$$

Тут наявні три лінійні рівняння з трьома змінними, а отже їх можна розв’язати методом Гауса. Хоча при $n = 10^6$ шанс отримати однакову трійку і є досить малим[25], але він не є таким, що ним формально можна знехтувати. Для

того, щоб прибрати обмеження на розмір таблиці можливо використати евристику Фіата-Шаміра[18].

Евристика Фіата-Шаміра – це евристика, що допомагає перетворити будь-який інтерактивний протокол на неінтерактивний за рахунок того, що якщо на якомусь кроці перевіряючий має надати якесь випадкове значення, замість нього використовують випадковий оракул, що на вхід приймає усі значення, що відомі перевіряючому до цього моменту в протоколі. Найчастіше в ролі цього оракула виступає криптографічна геш-функція. Приклад того, як евристику Фіата-Шаміра можна застосувати до алгоритму ША3:

- V обирає випадково число r і відправляє його P.
- P вибирає випадкове v та обчислює t та e, де $t = g^v$, $e = H(g, g^s, t, r)$. Далі, відправляє до V числа t, e, та $y = v + es$.

P може перевірити, що e було пораховано правильно та перевірити, що $g\hat{}y = g\hat{}v * h\hat{}e$.

Дана пропозиція наводиться мною без формального доведення. Неформально безпеку можна пояснити так, що V має завжди використовувати якесь псевдовипадкове число, для того, щоб в випадку якщо хтось зміг підгледіти розмову, то він не зміг перевикористати дані з неї в наступних. З іншого боку, оскільки e (яке було б надане випадково V в оригінальному протоколі Шнора) рахується геш-функцією в залежності від g^v, то зловмисному доказувачу важко підібрати пару (v, e) так, щоб вона дозволила йому якось оманути перевіряючого з більшою ймовірністю ніж в оригінальному протоколі. Варто зазначити, що загалом евристика Фіата-Шаміра досі не має формального доведення безпеки, проте має дуже широке застосування в криптографії, тож її можна віднести до алгоритмів, що виправдали себе часом.

3.6 Використання більш складних схем доведень з нульовим розголошенням

Попередньо були розглянуті схеми автентифікації, що дозволяють доводити наявність секретів до специфічних задач, де секрет має бути квадратичним залишком, або секрет має бути дискретним логарифмом.

Проте іноді з'являється потреба у доведенні секретів у більш загальному вигляді. Наприклад, якщо є потреба зробити приватним ключем не $g^x \ (mod\ p)$, а наприклад $H(x)$, де H – криптографічна геш-функція, особливо, якщо ця геш-функція не є арифметичною, тобто при її виконанні виконуються в першу чергу бітові операції (AND, OR, XOR). Для задач подібного роду використовують системи доказів з нульовим розголошенням (zero knowledge proof systems), що дозволяють ефективно доводити практично будь-які твердження з класу NP. Найчастіше подібні системи є дуже складними (детальний опис будь-якого з них міг би зайняти окрему кваліфікаційну роботу), тож ця робота не включає деталі про те, як дані протоколи працюють. Проте, варто ознайомимося з основними напрямами застосування.

В блокчейні ZCash[26] використовується система доведень zkSNARK, яка дозволяє користувачам доводити, що у них є приватний ключ до акаунта у якого є гроші. Велика різниця між цим та тим, що надавали інші попередні протоколи – те, що користувач доводить не просто, що у нього є приватний ключ s до публічного ключ $P = f(s)$, а доводить це навіть не показуючи P. Тобто доводиться, що "серед списку існуючих акаунтів у мене приватний ключ до одного з них і на цьому акаунті вистачає балансу для даної транзакції". Таким чином досягається повна анонімізація транзакцій та балансів користувачів. Схожим працює сервіс міксер грошей Tornado Cash[27].

zkSNARKs, як і попередньо розглянуті протокол Шнора спираються на нерозв'язність задачі дискретного логарифму. Через це вони не є стійкими до атак квантових комп'ютерів. На цьому фоні розвивається технологія zkSTARKs, які є резистентними до атак квантових комп'ютерів, адже ця система доведень спирається на криптографічні геш-функції. Наразі zkSTARKs через меншу

ефективність за zkSNARKs не є часто вживаними, проте іноді використовуються для доведень коректності деяких блокчейнів[28].

Найчастіше ці протоколи використовуються в комбінації з евристикою Фіата-Шаміра для того, щоб доведення відбувалося за один крок автентифікації. Це означає, що доведення можна перевикористовувати. Це є неприпустимим для протоколів ідентифікації, які, наприклад, використовують картки, адже по суті дозволяє зажди ідентифікуватись як певний користувач. У випадку блокчейн-проєктів це є прийнятно, адже у всіх випадках вище блокчейн зберігає дані про те, що цей самий доказ не може бути використаний двічі. На більш формальному рівні, там не є можливим перевикористання монет, адже монети не міняють своїх власників, а завжди створюються нові з новими власниками (с точки зору користувача різниці немає, але з точки зору доведень змінюється ідентифікатор монети). Вимога зберігання додаткових даних не є проблемою, адже найчастіше подібні автентифікації закінчуються зміною стану блокчейну в будь-якому разі (зміна балансів користувачів, тощо).

4 ПРОГРАМНА РЕАЛІЗАЦІЯ БІБЛІОТЕКИ ПРОТОКОЛІВ АВТЕНТИФІКАЦІЇ

4.1 Інструменти реалізації

Для реалізації бібліотеки алгоритмів автентифікації було обрано мову програмування Typescript, в першу завдяки тому, що завдяки тому, що він є надбудовою над мовою програмування Javascript, а отже може отримувати доступ до багатої екосистеми пакетів та засобів розробки для мови програмування Javascript, наприклад зручному репозиторію пакетів npm. З іншого боку, Typescript – мова типізована, а отже набагато більше підходить для розробки великих проєктів, ніж Javascript.

Було використано менеджер залежностей Yarn (v1.22.19), для виконання програми був використаний двигун Javascript Node.js (v14.17.0). В якості інтегрованого середовища розробки було використано текстовий редактор Visual Studio Code. Цей редактор має велике різномаїття плагінів для розробки для різних мов програмування, проте підтримка мови програмування Typescript є вбудованною в цьому редакторі, тож додаткових плагінів не використовував. Для тестування використовував фрейморки Mocha & Chai.

Для роботи з еліптичними кривими була використана бібліотека elliptic. Для роботи з великими цілами числами була обрана бібліотека ethers, що містить багато вбудованих функцій для роботи з блокчейном Ethereum, проте для потреб нашої бібліотеки буде використовуватися тільки зручна робота з великими числами, що є надбудовою над бібліотекою BN.js.

Розробка та тестування бібліотеки відбувалося на Macbook Pro 2021, macOS Monterey v12.6 з об’ємом оперативної пам’яті 64GB та процесором M1.

4.2 Опис програмної реалізації

В даній бібліотеці реалізовано два протоколи:

- Шнора

- ША3 з використанням евристики Фіата-Шаміра, що була запропонована мною в розділі 3.5.1. Надалі будемо застосовувати скорочення (ША3ФШ), анлійською FA3FS.

Відповідні класи SchnorrProtocol, FA3FSProtocol підтримують методи ***newVerifier*/*newProver*.** Оскільки потреби цих протоколів (тобто дані, що передаються для створення відповідного об'єкту доказувача, перевіряючого чи довіреного центру) є різними, об'єднувати їх у спільний інтерфейс не є доцільним. Проте, для простоти інтеграції майбутньої інтеграції схожих протоколів було створено шаблонні інтерфейси ***I3StepProtocolDescriptor*** та ***I2StepProtocolDescriptor*** для протоколів, що складаються з двох та трьох кроків відповідно. У відповідні класи інтерфейси були об'єднанні прувери та веріфаєри відповідних протоколів.

В випадку обох протоколів група над яким виконуються відповідні протоколи – група еліпитичної кривої secp256k1, що є дуже популярною кривою, що зарекомендувала себе на практиці та, наприклад, є еліптичною кривою, що використовується для підписів блокчейну Bitcoin[11].

4.2.1 Опис реалізації протоколу Шнора

Метод ***newProver*** класу Шнора приймає публічний ключ та секрет (взагалі можливим є обчислення публічного ключа зі значення секрету, але це полегшує імплементацію). Він повертає об'єкт класу ***SchnorrProver***, що має два публічні методи:

- ***getFirstProverMessage*** – повертає випадкове число r для першого кроку протоколу Шнора. Оскільки це число r буде ще потрібне для третього кроку протоколу, це число запам'ятовується та не може бути більше перезаписане. Тобто другий раз викликати ***getFirstProverMessage*** не можна, треба створювати новий об'єкт класу для кожної нової взаємодії.
- ***getProverSecondMessage*** – приймає як параметр випадкове число e з другого кроку протокола Шнора та повертає y для третього кроку протоколу Шнора. Цей метод теж не можна викликати двічі, адже під час нього

запам’ятовується значення e. Якщо відбудеться дві взаємодії протоколу Шнора з одним значенням r, але різними значенням e, то це дасть можливість підглядувачам за протоколом обчислити секретний ключ.

Метод ***newVerifier*** класу Шнора примає публічний ключ, секрет до якого до буде доводитись в процесі протоколу Шнора (вважається, що він був отриманий на якомусь із попередніх кроків взаємодії). Після цього отримується об’єкт класу ***SchnorrVerifier***, що має два методи

- ***getVerifierRequest*** (той, що у відповідь на число r доказувача отримує число e).
- ***verify***, що отримує параметр y та повертає ***true*** або ***false*** в залежності від того, чи є відповідь правильна.

4.2.2 Опис реалізації протоколу ШАЗФШ

Протокол реалізований аналогічно до протоколу Шнора. Головною різницею є той факт, що тепер перше повідомлення відправляється перевіряючим. У свою чергу, доказувач тепер імплементує лише один метод ***getProverMessage***, що приймає на вхід випадкове число r з першого кроку.

4.3 Програмний інтерфейс бібліотеки

Бібліотека може бути використана в середині будь-якого JavaScript-сумісного середовища. Прикладом програмного коду, що виконує всі кроки протоколу Шнора є наступний:

```
const authLibrary = require('fast-auth-lib');

const protocol = new authLibrary.SchnorrProtocol();

const proverParams = protocol.getRandomProverParams();
console.log('Параметри прувера: ', proverParams);

// Крок 0. Ініціалізація протоколу та його учасників
const prover = protocol.newProver(proverParams);
const verifier = protocol.newVerifier({ testedPublicKey: proverParams.publicKey });

// Крок 1. Прувер відправляє g^r до верифікатора
const proverMessage1 = prover.getFirstProverMessage();
console.log('Крок 1. Повідомлення прувера: ', proverMessage1);

// Крок 2. Верифікатор відправляє випадкове значення e
const verifierMessage = verifier.getVerifierRequest(proverMessage1);
console.log('Крок 2. Повідомлення верифікатора: ', verifierMessage);

// Крок 3. Прувер відправляє відповідь на виклик верифікатора
const proverMessage2 = prover.getProverSecondMessage(verifierMessage);
console.log('Крок 3. Повідомлення прувера: ', proverMessage2);

console.log('Вердикт перевіряючого: ', verifier.verify(proverMessage2));
```

Рис 1. Приклад імплементації Протоколу Шнора

Програма виводить наступний результат:

```
Параметри прувера:  {
  secret: '722aa350234346eea21263e5283c753fd087e1ed9bc7bec2f034c55830c09d32',
  publicKey: '046349857abef5fac4de62fe056c64244483aa2a4799c02ff76d5f0d3963e004c1cf5f2d761d75d77025
6f1e0218f641ee4d4794d890a39939a8b07a08d11202f3'
}
Крок 1. Повідомлення прувера:  {
  gToR: '04364b340317387c344f7bb585e6874b0d7f3cd7d18c4367aa2c0d3616ceb40adf50f88755fe23235fde4c3df
27367712fc243476d1d41bb37822d91cfc9e4797d'
}
Крок 2. Повідомлення верифікатора:  {
  e: 'a75762bc9c02a6f3a848674c582e918ec985db597fc56996601b7ce147dcf14b'
}
Крок 3. Повідомлення прувера:  {
  y: '500d3987f4277aee591419c7666cce84b469661f3bd76bb20847c4c5ef275198'
}
Вердикт перевіряючого:  true
```

Рис 2. Результат виконання протоколу Шнора

Для простоти демонстрації proverMessage1 був одразу доступний до перевіряючого, проте в реальних умовах так звичайно не буде. Для передачі

повідомлень між перевіряючим та доказувачем можна використати стандартні засоби мови програмування JavaScript, такі як AJAX або fetch API.

Аналогічною є й реалізація протоколу ША3ФШ:

```
const authLibrary = require('fast-auth-lib');

const protocol = new authLibrary.FA3FSProtocol();

const proverParams = protocol.getRandomProverParams();
console.log('Параметри прувера: ', proverParams);

// Крок 0. Ініціалізація протоколу та його учасників
const prover = protocol.newProver(proverParams);
const verifier = protocol.newVerifier({ testedPublicKey: proverParams.publicKey });

// Крок 1. Веріфікатор відправляє випадкове значення r
const verifierMessage = verifier.getVerifierRequest();
console.log('Крок 1. Повідомлення веріфікатора: ', verifierMessage);

// Крок 2. Прувер відправляє відповідь на виклик веріфікатора
const proverMessage2 = prover.getProverMessage(verifierMessage);
console.log('Крок 2. Повідомлення прувера: ', proverMessage2);

console.log('Вердикт перевіряючого: ', verifier.verify(proverMessage2));
```

Рис 3. Приклад імплементації протоколу ША3ФШ

Програма виводить наступний результат:

```
Параметри прувера:  {
  secret: '19a4b400b4b7859033e953de2a1b5b1c68a8f532fdba25e1f286d74f65f56cf8',
  publicKey: '049dd4e9465c2ada6e98e3d239a2fd3f757efa0846bcf80a1ddbf0a01e17bf4bb7afa6acfbdc5d2aaf08
1e8bb18356a0e231afbe0a20cb31bebf271abc21993373'
}
Крок 1. Повідомлення веріфікатора:  {
  r: '3bf7eb491db02671f2778d52fde0042c79b84f6d8ffbcb1609532fa8589e434d'
}
Крок 2. Повідомлення прувера:  {
  y: '6066e836b4b3d605ecec54b35564f46cc2609af52c2821642ef2adb7be22036e36329ee7ebd5285002a9e2fbccd1
031722265e46563ffe00034c5f6fb8bdd3',
  t: '0481479f52470b1f3c762c4c568d48c1267931b017ecc0498c5b638f0397317c6fa5c0df161e5a1c2e8e08df916f
a9f70ec0ccc7889ed789536ff9642a38764910',
  e: '3c263aabe5f6512552ce5ffde3cb31c1783d790d31ff85a38d2b358462c9156'
}
Вердикт перевіряючого:  true
```

Рис 4. Результат виконання протоколу ША3ФШ

4.4 Висновки щодо реалізації бібліотеки

В ході написання кваліфікаційної роботи було імплементовано бібліотеку, що надає можливість виконувати роль будь кого, як і перевіряючого, так і доказувача. Бібліотека використовує сучасні криптографічні стандарти та примітиви і є готовою до використання в будь-якому JavaScript-сумісному середовищі. Бібліотека є доступною за посиланням на GitHub[30].

ВИСНОВКИ

Протоколи автентифікації дозволяють ідентифікувати людину, тобто зрозуміти ким вона є та які відповідно має права доступу до інформації, території, тощо. Протоколи автентифікації існували так довго, як існує людство, для цього використовували або рідкісні предмети або базову біометрію – обличчя людини, її голос, тощо. З розвитком інформаційних технологій з'являється все більша потреба у розвинутих засобах авторизації, які:

- Безпечні (їх неможливо підробити). Потрібний рівень безпеки залежить від ситуації. Так, деяким протоколам достатньо шансу на підробку один на мільярд (якщо наприклад потенційний зловмисник має особисто бути присутнім, тоді невдала спроба може бути тільки одна)[18], іншим потрібний рівень безпеки, що дозволяє захиститися від атак навіть наддержав (наприклад, блокчейн технології).
- Вимагають мало ресурсів, в першу чергу від прувера, яким може бути навіть кредитна картка, або іншого роду малий чіп. Такі засоби немають великих ресурсів як обчислювальних, так і для зберігання пам'яті. Також, часто є потреба в мінімізації комунікаційних ресурсів, особливо якщо взаємодія виконується через слабкі комунікаційні лінії, наприклад рація або мобільний інтернет.

В ході роботи було розглянуто кілька основних напрямів протоколів автентифікації: протоколи, з використанням паролю, протоколи з використанням підписів та протоколи з використанням доведень з нульовим розголошенням. Цей порядок йде як в порядку часу появи таких протоколів автентифікації, так і в порядку покращення безпекових властивостей даних протоколів.

Протоколи, що засновані на паролях існували давно. Навіть в фільмах часто можна побачити, те, що для доступу в деякі клуби потрібно знати їх пароль. Більше того, президенти США для авторизації запуску ядерної зброї використовують пароль під назвою Gold Codes[29]. Автентифікація через пароль найзручнішою для користувачів, але має фундаментальну проблему – безумовна

довіра до сервера, адже сервер отримує цей пароль в чистому вигляді і користувач має довіряти серверу, що він не буде ніяк його зберігати. Через це користувачам додатково доводиться тримати різні паролі на різних сервісах. Не дивлячись на всі недоліки, скоріш за все паролі залишаться головними засобами автентифікації користувачів через їх зручність.

Недоліки методів автентифікації через пароль розв'язує симетричне та асиметричне шифрування. Недивлячись на простоту та перевірку часом найчастіше для таких цілей симетричне шифрування не використовується, адже в симетричному шифруванні ключ має бути попередньо узгоджений, що часто не є практичним в застосуванні. Найчастіше використовується асиметричне шифрування, де перевіряльник відправляє випадкове число x, а доказувач має його підписати своїм приватним ключем. При правильній реалізації, що запобігає перевикористання підпису між різними перевірячами він є досить безпечним способом авторизації, який знайшов багато використання в блокчейн проєктах, наприклад OpenSea. У цій роботі було частково розглянуто популярний протокол асиметричної криптографії RSA та детальніше розглянуто набагато більш безпечний протокол ECDSA.

Головним недоліком методу автентифікації через підписи є той факт, що підписи найчастіше розкривають інформацію, якої раніше у перевіряльника не було, наприклад це дає перевіряльнику, якщо він часто використовується, використовувати доказуючого як оракул для отримання підписів на будь-які повідомлення. При правильній імплементації перевіряльник не зможе використати цей підпис в інших місцях, але часто правильна імплементація спирається на певну чесність перевіряльника та обачність власне самого доказувача, що викликало чимало атак, де користувачі втрачали гроші через те, що підписували повідомлення недовіреним перевіряльникам[31].

Покращенням підходу з підписами є підход з використанням доведень з нульовим розголошенням, де перевіряльник не дізнається нічого нового, окрім того, що доказувач володіє секретом. Це є найкращими видами протоколів автентифікації. Серед розглянутих протоколів були протокол Фіата-Шаміра,

Фейге-Фіата-Шаміра, Шнора, а також алгоритм ША3 запропонований професором КНУ Анісімовим А.В.. Була запропонована модифікація протоколу ША3 з використанням евристика Фіата-Шаміра (ША3ФШ).

В рамках цієї роботи також було створено програмну реалізацію бібліотеки, що містить реалізацію протоколу Шнора та модифікацію ША3 з використанням евристики Фіата-Шаміра. Ця програмна реалізація використовує популярні криптографічні примітиви, що роками користання та криптоаналізу показали свою безпеку. Ця бібліотека є готовою для використання в будь-якому JavaScript-сумісному середовищі, тобто браузери, сервери (Node.js, Deno), та інші.

Висновком даної роботи є те, що алгоритми Шнора та ША3ФШ є підходящими алгоримами для автентифікації. Якщо швидкість комунікації не є проблемою, то більш доречним є використання протоколу Шнора через його перевіреність часом та загальна менша кількість обчислень. Якщо швидкість комунікації є великою проблемою, то ША3ФШ стає більш зручним, через зменшення кількості раундів комунікації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Про затвердження Правил забезпечення захисту інформації в інформаційних, телекомунікаційних та інформаційно-телекомунікаційних системах [Електронний ресурс] : Постанова Каб. Міністрів України від 29.03.2006 р. № 373 : станом на 21 жовт. 2022 р. – Режим доступу: https://zakon.rada.gov.ua/laws/show/373-2006-п#Text.
2. Integrating google sign-in into your web app | authentication | google for developers [Електронний ресурс] // Google for Developers. – Режим доступу: https://developers.google.com/identity/sign-in/web/sign-in.
3. Freeman D. M. Schnorr identification and signatures / David Mandell Freeman. – 2011.
4. Shoup V. On the security of a practical identification scheme / Victor Shoup // Journal of cryptology. – 1999. – № 12. – С. 247–260.
5. Damgard I. On Σ-protocols / Ivan Damgard. – 2010.
6. Enabling HTTPS on your servers [Електронний ресурс] // web.dev. – Режим доступу: https://web.dev/enable-https.
7. Official PCI security standards council site - verify PCI compliance, download data security and credit card security standards [Електронний ресурс] // PCI Security Standards Council. – Режим доступу: https://listings.pcisecuritystandards.org/pci_security/.
8. Authenticate using google with javascript | firebase [Електронний ресурс] // Firebase. – Режим доступу: https://firebase.google.com/docs/auth/web/google-signin.
9. Boneh D. Twenty years of attacks on the RSA cryptosystem / Dan Boneh. – 1999.
10. Isom K. Practical cryptography with go / Kyle Isom. – [Б. м.] : Leanpub, 2014.
11. Secp256k1 - bitcoin wiki [Електронний ресурс] // Bitcoin Wiki. – Режим доступу: https://en.bitcoin.it/wiki/Secp256k1.

12. Wood G. Ethereum: A secure decentralised generalised transaction ledger. Berlin version beacfbd / Gavin Wood. – 2022.
13. Console Hacking 2010 – PS3 Epic Fail. [Електронний ресурс]. – Режим доступу: https://fahrplan.events.ccc.de/congress/2010/Fahrplan/attachments/1780_27c3_console_hacking_2010.pdf.
14. Holst Swende M. ERC-191: signed data standard [Електронний ресурс] / Martin Holst Swende, Nick Johnson // Ethereum improvement proposals. – 2016. – № 191. – Режим доступу: https://eips.ethereum.org/EIPS/eip-191.
15. Bellare M. On defining proofs of knowledge / Mihir Bellare, Oded Goldreich // Advances in Cryptology – CRYPTO' 92, 1 січ. 2001 р. – [Б. м.]. – С. 390–420.
16. Demonstrate how Zero-Knowledge Proofs work without using math [Електронний ресурс]. – Режим доступу: https://www.linkedin.com/pulse/demonstrate-how-zero-knowledge-proofs-work-without-using-chalkias/
17. Goldreich O. How to construct constant-round zero-knowledge proof systems for NP / Oded Goldreich, Ariel Kahan // Journal of cryptology. – 1996. – № 9. – С. 167–189.
18. Fiat A. How to prove yourself: practical solutions to identification and signature problems / Amos Fiat, Adi Shamir // Proceedings on Advances in cryptology -- CRYPTO '86, 1 берез. 1999 р. – [Б. м.].
19. Feige U. Zero-Knowledge proofs of identity / Uriel Feige, Amos Fiat, Adi Shamir // Journal of cryptology. – 1988. – № 1. – С. 77–94.
20. Eikenberry S. M. Efficient algorithms for computing the jacobi symbol / Shawna Meyer Eikenberry, Jonathan P. Sorenson // Journal of symbolic computation. – 1998. – Т. 26, № 4. – С. 509–523
21. Schnorr C.-P. Efficient identification and signatures for smart cards / Claus-Peter Schnorr // Advances in cryptology – CRYPTO' 89 proceedings, 1 січ. 2001 р. – [Б. м.].

22. Untraceable off-line cash in wallet with observers // Advances in Cryptology – CRYPTO' 93. – [Б. м.], 2001. – С. 302–318.
23. Aronsson H. A. Zero knowledge protocols and small systems [Електронний ресурс] / Hannu A. Aronsson. – Режим доступу: https://www.cs.princeton.edu/~chazelle/courses/BIB/ZKprotocols.pdf.
24. Анатолій А. В. Швидка автентифікація типу «свій-чужий» / Анісімов Васильович Анатолій. – [Б. м. : б. в.].
25. DasGupta A. The matching, birthday and the strong birthday problem: a contemporary review / Anirban DasGupta // Journal of statistical planning and inference. – 2005. – Т. 130, № 1-2. – С. 377–389.
26. How it works | zcash [Електронний ресурс] // Zcash. – Режим доступу: https://z.cash/technology/
27. Pertsev A. Tornado cash privacy solution version 1.4 [Електронний ресурс] / Alexey Pertsev, Roman Semenov, Roman Storm. – Режим доступу: https://berkeley-defi.github.io/assets/material/Tornado%20Cash%20Whitepaper.pdf.
28. Introduction :: StarkEx Documentation [Електронний ресурс] // Introduction :: StarkEx Documentation. – Режим доступу: https://docs.starkware.co/starkex/index.html
29. Gold codes - wikipedia [Електронний ресурс] // Wikipedia, the free encyclopedia. – Режим доступу: https://en.wikipedia.org/wiki/Gold_Codes
30. GitHub - StanislavBreadless/authentification-schemes-lib: Бібліотека з імплементацією протоколу Шнора та ША3ФШ [Електронний ресурс] // GitHub. – Режим доступу: https://github.com/StanislavBreadless/authentification-schemes-lib
31. SlowMist. Slow mist: "blank check" eth_sign phishing analysis [Електронний ресурс] / SlowMist // Medium. – Режим доступу: https://slowmist.medium.com/slow-mist-blank-check-eth-sign-phishing-analysis-741115