КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра математичної інформатики

«До захисту допущено»
Завідувач кафедри
Терещенко В.М. _____
(підпис)

«___» _____20__ р.

Дипломна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

ШВИДКА АВТЕНТИФІКАЦІЯ

Виконав студент 4 курсу
Денісенко Артем Сергійович _____
(підпис)

Науковий керівник:
Доктор фізико-математичних наук, професор
Анісімов Анатолій Васильович _____
(підпис)

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

Студент _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 43 сторінки 4 ілюстрації 3 таблиці 10 джерел.

КРИПТОГРАФІЯ, АВТЕНТИФІКАЦІЯ, ПРОТОКОЛ, КЛЮЧ, ZERO-KNOWLEDGE PROOF, C++.

Об’єктом дослідження є існуючі протоколи автентифікації т їх порівнняня. Об’єктом розробки є один з розглянутих протоколів - а саме альфа-бета.

Метою дипломної роботи є дослідження технологій автентифікації та розробка і реалізація одного з протоколів zero-knowledge автентифікації.

Інструменти реалізації: для розробки програмної частини використовувалася мова C++, у якості редактору кода використано Visual Studio Code, а для сборки програми - утиліта make.

Результат роботи: досліджено та проаналізовано існуючі протоколи автентифікації, розглянуто їх призначення та сильні та слабкі сторони. Реалізовано протокол альфа-бета, що є одним з zero-knowledge протоколів автентифікації.

ЗМІСТ

ВСТУП

Оцінка сучасного стану об’єкта розробки. Автентифікація - процедура перевірки “справжності” - є релевантною в багатьох сферах, як, наприклад, мистецтві, соціальній сфері та антропології. Автентифікація також грає важливу роль в інформаційних системах та комп'ютерних науках - для надання користувачу доступу до будь-якої конфіденційної інформації або системи зазвичай необхідно підтвердити його ідентичність. На сьогодні інтернет став значною частиною повсякденного життя людини, і багато операцій в ньому потребують підтвердження права користувача їх виконувати, що призводить до росту попиту на надійні та швидкі протоколи автентифікації.

Існує велика кількість протоколів автентифікації, таких як PAP (Password Authentication Protocol), Kerberos, EAP (Extensible Authentication protocol), OpenID та багато інших. Ці протоколи мають свої переваги та недоліки та використовуються різними компаніями або державними органами.

Актуальність роботи та підстави для її виконання. Сьогодні аутентифікаційні системи використовуються майже всюди, де існує необхідність надати користувачу доступ до конфіденційних даних, систем, або операцій. Операційні системи, соціальні мережі, платіжні сервіси та багато іншого не могли б існувати без засобів підтвердження особистості користувача. Усе це є дуже важливою частиною функціонування сьогоденного суспільства.

Зважаючи на це, технології аутентифікації постійно оновлюються та покращуються, з’являються нові протоколи. Різні протоколи дозволяють досягти дещо різних цілей у процесі автентифікації, як, наприклад,

можливість підтвердження володіння інформацією без передачі цієї інформації. Таким чином, проблема автентифікації далека від закриття.

Мета й завдання роботи. Метою дипломної роботи є дослідження існуючих методів та протоколів автентифікації, а також реалізація одного з них - αβ(альфа-бета) протоколу.

Для досягнення поставленої мети повинні бути виконані наступні завдання:

- Проаналізувати існуючі популярні протоколи автентифікації, визначити їх особливості, переваги та недоліки.
- Розробити програмну реалізацію альфа-бета протоколу з можливістю її інтеграції в існуючі системи.

Об'єкт, методи й засоби розроблення. Об’єктом дослідження та аналізу є існуючі протоколи автентифікації. Об’єктом розробки є протокол альфа-бета.

Для розробки програмної реалізації було обрано мову C++. Мова є однією з найбільш використовуваних у сфері криптографії та дозволяє розробляти високошвидкісні програми. Це є високорівнева об'єктно-орієнтована, узагальнена та процедурна мова, що має багатий вибір ресурсів, як то структури даних та алгоритми, у стандартній або сторонніх бібліотеках.

Під час розробки використовувався редактор коду Visual Studio Code, що, незважаючи на свою легковажність, надає широку підтримку для ряда мов програмування, а також підтримку git. Для сборки використовувалася утилита make, для компіляції використано GCC 7.5.0.

Можливі сфери застосування. Розроблений протокол автентифікації може бути використаний у існуючих інформаційних системах.

РОЗДІЛ 1 АВТЕНТИФІКАЦІЯ

1.1 Поняття та історія

Автентифікація є актом доказу якогось твердження, як, наприклад, особи користувача комп’ютерної системи. Частинним випадком автентифікації є ідентифікація - процес підтвердження своєї особистості. В загальному сенсі автентифікація може включати перевірку особистих документів, що підтверджують особу, встановлення справжності веб-сайту за допомогою цифрового сертифіката, визначення віку артефакту шляхом вуглецевого датування або забезпечення того, що товар чи документ не є фальшивими. Цей процес був значною частиною функціонування суспільства задовго до появи комп'ютерних систем і ств лише важливішим з їх популяризацією.

Історично відбитки пальців використовувались як найавторитетніший метод аутентифікації, проте з часом судові справи в різних регіонах світу привели до виникнення фундаментальних сумнівів щодо надійності цього методу. [1]

У контексті комп'ютерних наук існують криптографічні методи, які на сьогодні не піддаються підробці тоді і тільки тоді, коли ключ відправника не був скомпрометований. Те, що відправник (або хтось інший, відмінний від зловмисника) знає (або не знає) про порушення, не має значення. Невідомо, чи ці криптографічно засновані методи автентифікації є доказово безпечними, оскільки непередбачувані математичні розробки можуть зробити їх вразливими до атак в майбутньому. У разі знаходження подобної розробки вона може поставити під сумнів усі операції, що використовували нині вразливий алгоритм для аутентифікації, у минулому. Зокрема, цифрово підписаний контракт може

бути поставлений під сумнів при виявленні нової атаки на алгоритм, що лежить в основі підпису.

Тісно пов’язаним з автентифікацією є процес авторизації - надання доступу до ресурсів, систем або інструментів. Дуже часто автентифікація виконується безпосередньо перед авторизацією для встановлення наявності у користувача права на ресурси, але це не є обов'язковим - анонімні користувачі також можуть отримувати обмежений доступ до ресурсів, не потребуючі автентифікації.

1.2 Цілі автентифікації

Автентифікація використовується для підтвердження “справжності” об'єкта чи особи. Наприклад, у випадку ідентифікації, з точки зору верифікатора, результатом протоколу автентифікації суб’єкта є або прийняття його особи як справжньої (завершення з прийняттям), або припинення без прийняття (відхилення).

Таким чином, цілі автентифікації відрізняються в залежності від сценарію:

- У випадку двох доброчесних сторін A та B верифікатор зможе успішно завершити протокол та підтвердити, що заявник є тим, за кого себе видає.
- Недоброчесна сторона E не може використати ідентифікаційний обмін з A щоб успішно видати себе за A при обміні з B.
- Ймовірність того, що будь-яка сторона E, відмінна від A, що виконує протокол і відіграє роль A, може змусити B успішно завершити виконання протоколу і прийняти особистість A, настільки мала, що не має практичного значення і нею можна знехтувати.

- Попередні пункти повинні залишатися вірними незалежно від обставин E. Наприклад, наявність у E досвіду спілкування з обома A та B, велика кількість спостережень за виконанням протоколу між A та B, або велика кількість одночасно виконуваних інстанцій протоколу не повинні вплинути на правдивість цих пунктів.

Ідея zero-knowledge-based протоколів дозволяє сторонам не розкривати взагалі ніякої важливої інформації, що могла б потенційно зробити задачу E дещо простішою.

1.3 Фактори аутентифікації

Можна виділити три основних категорії технік автентифікації[2]:

- *Щось відоме.* Приклади включають стандартні паролі (які можуть використовуються для отримання симетричного ключа), персональні ідентифікаційні номери (PIN-коди), а також секретні або приватні ключі, знання яких демонструються в виклик-відповідь протоколах.
- *Щось у власності.* Зазвичай це фізичний аксесуар, що за функціональністю нагадує паспорт. Приклади включають картки з магнітною смужкою, розумні карти (пластикові картки розміром із кредитні картки, що містять вбудований мікропроцесор або інтегральну схему; їх також називають IC-картами), а також ручні персоналізовані калькулятори (генератори паролів), які дозволяють генерувати паролі залежно від часу.
- *Щось притаманне (конкретній особі).* Ця категорія включає методи, які використовують фізичні характеристики і мимовільні дії людини (біометрія), такі як рукописний підпис, відбитки пальців, голос, малюнок сітківки ока, геометричні

проби руки і динамічні характеристики клавіатури. Ці методи, як правило, не є криптографічними.

Однофакторна автентифікація - це найслабший рівень автентифікації, що використовує лише одну з трьох категорій вище. Наприклад, будь-яка система, що використовує лише пароль для автентифікації користувача є однофакторною.

Багатофакторна автентифікація використовує декілька факторів. Її часним випадком є двофакторна аутентифікація (2FA), що перевіряє наявність двох і саме двох факторів. Найчастіше зустрічаються системи, що використовують комбінацію з пароля (щось користувач знає) та псевдовипадкового числа, згенерованого за допомогою токену (щось користувач має).

1.4 Властивості автентифікації

Протоколи автентифікації мають багато властивостей, серед яких можна зазначити:

1. *Взаємність ідентифікації.* Одна або обидві сторони можуть підтвердити свої ідентифікаційні дані іншій стороні, забезпечуючи, відповідно, односторонню або взаємну ідентифікацію. Деякі технології, такі як схеми з фіксованими паролями, можуть бути вразливі до того, що організація, яка видає себе за верифікатора, може просто перехопити пароль заявника.
2. *Обчислювальна ефективність.* Кількість операцій, необхідна для підтвердження особистості чи інформації. Для систем, що виконують велику кількість автентифікації щоденно, ця характеристика є дуже важливою.

3. *Ефективність комунікації.* Сюди входить кількість проходів (обмінів повідомленнями) і необхідна пропускна здатність (загальна кількість переданих бітів).
4. *Участь третьої сторони (за наявності) в режимі реального часу.* Прикладами третіх сторін можуть бути довірена онлайн-система для поширення спільних симетричних ключів серед взаємодіючих суб'єктів для подальшої аутентифікації або онлайн (не довірена) службу каталогів для поширення сертифікатів з відкритим ключем, що підтримується автономним офлайн центром сертифікації.
5. *Характер довіри до третьої сторони (за необхідності).* Приклади включають довіру, що третя сторона правильно організує аутентифікацію і прив'язку імені організації до відкритого ключа або довіру третій стороні зі знанням закритого ключа організації.
6. *Характер гарантій безпеки.* Приклади включають provable security (доказуєму безпеку) та zero-knowledge (нульове знання) властивості.
7. *Зберігання секретної інформації.* Це включає локацію та методи, що використовуються для зберігання критичної для системи інформації.

1.5 Атаки на аутентифікаційні протоколи

Автентифікація є привабливою ціллю для зловмисників, бо можливість видавати щось за те, чим воно не є, або себе за іншу людину відкриває дуже багато можливостей.

Найбільш очевидною такою є *уособлення* - ситуація, де одна сутність видає себе за іншу. За допомогою цього зловмисник може отримати доступ до конфіденційних ресурсів або систем.

Зловмисник може намагатися уособлювати сутність рядом способів, запобігання яких було згадано у пункті 1.2. Ці способи включають до себе:

- *Replay attack* - атака, що включає використання інформації з попереднього виконання протоколу на тому самому або іншому верифікаторі. У випадку файлів, що зберігаються на девайсі, аналогом цієї атаки є *атака відновлення (restore attack),* при якій файл замінюється на його попередню версію.
- *Known-key-attack* - зловмисник отримує доступ до ключів, що були використані раніше, і використовує їх для своїх цілей.
- *Man-in-the-middle* - клас атак, при яких зловмисник перехоплює повідомлення і надсилає його до кінцевої точки, можливо вносячи до повідомлення деякі зміни.
- *Interleaving attack* - атака, при якій зловмисник чергує дані з декількох попередніх або поточних виконань протоколу, можливо створюючи нові екземпляри протоколу самостійно.
- *Reflection attack* - атака, націлена на системи, що використовують однакові протоколи в обидва напрямки. Інформація з поточного протоколу надсилається назад відправнику, таким чином змушуючи його успішно завершити автентифікацію.
- *Chosen-text attack* - атака на протокол "виклик-відповідь", при якій зловмисник стратегічно вибирає виклики в спробі отримати інформацію про довгострокові ключі заявника.

РОЗДІЛ 2 ТИПИ ПРОТОКОЛІВ АВТЕНТИФІКАЦІЇ

Комунікаційний протокол - це набір правил передачі інформації між двома пристроями. Протокол задає правила, синтаксис та семантику обміну інформцією, а також методи обробки помилок, умови завершення комунікації та синхронронізацію.

Протокол аутентифікації є типом криптографічного комунікаційного протоколу, що використовується для передачі автентифікациоих даних між двома сутностями. Він дозволяє отримувачу даних ідентифікувати відправника, а також ідентифікувати себе для нього, визначаючи необхідну для цього інформацію та синтаксис.

Існує велика кількість протоколів автентифікації, і у цьому розділі буде розглянуто їх основні типи.

2.1 Автентифікація за паролем

Стандартні системи з постійними паролями надають те, що називається **слабкою автентифікацією**. Основна ідея досить проста і полягає в наступному. До кожного користувача прив’язується *пароль*, що зазвичай являє собою рядок з 6-10 або більше символів, які користувач здатний записати в пам'ять. Цей пароль виступає у якості спільного секрета між користувачем та системою.

Для ідентифікації у такій системі користувач використовує пару (ідентифікатор, пароль), де за допомогою ідентифікатора він вказує свою особу, а пароль використовується для підтвердження цієї особи. Наданий користувачем пароль перевіряється проти пароля, що відповідає заданому ідентифікатору у системі, і автентифікаційний протокол завершується успішно у випадку їх співпадіння, або не завершується у випадку невдалої верифікації.

Системи цього типу є досить розпоюдженними і широко використовуються у соціальних мережах та інших інтернет-платформах. Прикладом протоколу, що спирається на пароль, може слугувати Password Authentication Protocol (PAP) - протокол цього типу, що використовується Point-to-Point Protocol(PPP).[3]

Системи, що використовують автентифікацію, що спирається на фіксовані паролі, розрізняються за методами збереження даних та верифікації паролів.

2.1.1 Підходи до систем з фіксованим паролем

Найбільш очевидний підхід полягає в тому, що система зберігає паролі користувачів у текстовому вигляді у деякому системному файлі або базі даних, які захищені як від читання, так і від запису за допомогою системних засобів. При отриманні пароля система порівнює його із записом у файлі паролів, що відповідає ідентифікатору користувача; при цьому не використовуються ніякі криптографічні методи, лише пряме порівняння. Такий підхід не забезпечує захист від привілейованих зловмисників, що можуть отримати доступ до файлу у будь-який момент часу.

Більш безпечним є зберігання паролів у вигляді результату виконання односторонньої функції (функції, яка може бути легко обчислена в один бік, але не в інший) над безпосередньо текстовими паролями[4]. При автентифікації ця функція обчислюється для пароля користувача, і її результат порівнюється з даними у файлі. У цьому випадку файл не потребує захисту від читання, але досі існує шанс, що привілейований користувач може змінити чи пошкодити дані.

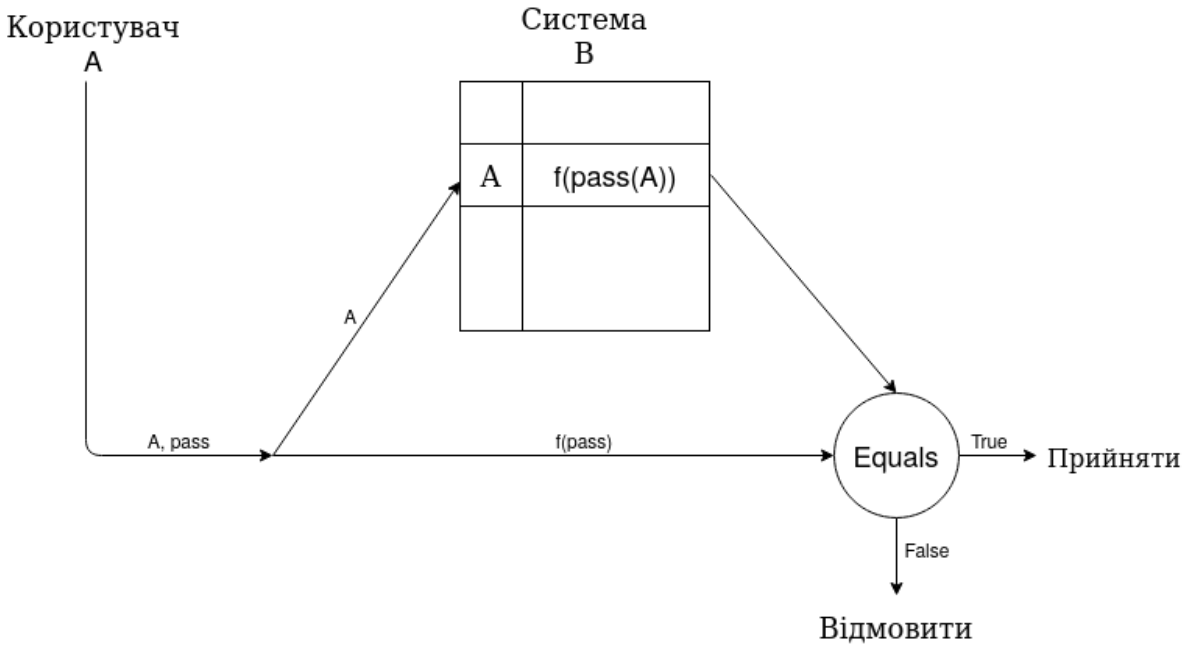

Рис.1 Приклад системи з односторонньою функцією

2.1.2 Атаки на системи з фіксованим паролем

Системи, що використовують фіксовані паролі, дуже вразливі до replay attack. Пароль може бути підглянуто у процесі його набору, вкрадено з місця, де користувач його зберігає, або підслухано у процесі передачі, бо користувач відправляє пароль у текстовому вигляді. Після цього зловмисник може успішно уособити користувача. Таким чином, якщо процес аутентифікації відбувається в мережі, що не викликає повної довіри, системи з фіксованим паролем не є оптимальними.

Інший тип атак базується на пошуку пароля. Найпростіший випадок цієї атаки - це простий наївний перебір можливих паролів, де зловмисник намагається раз за разом автентифікуватися, сподіваючись знайти правильний пароль. Така атака не є ефективною і протидіяти їй досить легко - наприклад, через обмеження кількості спроб аутентифікації користувач може здійснити у фіксований період часу або через вибір більш часозатратної функції, що виконується на паролі, для збільшення часу, що

витрачається при кожній спробі автентифікації. Іншим можливим засобом боротьби з цією атакою є збільшення простору імен, з якого користувачі можуть обирати паролі, для ускладнення задачі зловмисника.

Покращеною версію переборної атаки є перебір за словником (dictionary attack), що спирається на той факт, що більшість користувачів використовують лише малу підмножину з можливого словарного простору паролів. Зловмисник, що має “словник” найбільш популярних паролів, може значно пришвидшити процес пошуку пароля шляхом використання лише найбільш ймовірних його варіантів замість усієї множини.

2.2 Виклик-відповідь автентифікація

Виклик-відповідь автентифікація (challenge-response authentication) - це сімейство протоколів, де один із учасників процесу надсилає іншому “виклик” (задає якесь питання), а інший повинен надати “відповідь” для того, щоб бути автентифікованим[5].

Автентифікація за паролем є найпростішим випадком протоколу виклик-відповідь, де знання пароля є питанням, а сам пароль - відповіддю. Як розглянуто в пункті 2.1.2, цей підхід є дуже вразливим до ряду атак. Для запобігання цього використовується виклик, який змінюється в залежності від часу, і правильна відповідь залежить від обох секрета та виклика. Таким чином, навіть якщо зловмисник прослухає лінію зв’язку, цей обмін не надасть йому знання, достатні для підробки ідентифікації, бо наступний виклик буде відрізнятися.

2.2.1 Способи генерації виклику

У цьому пункті вкажемо декілька основних способів генерації виклику для виклик-відповідь протоколу:

1. **Випадкові числа.** Один суб'єкт включає випадкове число в вихідне повідомлення. Отримане згодом вхідне повідомлення

(наприклад, наступне повідомлення того ж екземпляра протоколу), для створення якого було необхідно знання цього числа і з яким це число нерозривно пов'язано, вважається *свіжим* (тобто згенерованим після старту протоколу) на підставі того, що випадкове число пов'язує ці два повідомлення. Нерозривний зв'язок необхідний для того, щоб запобігти додавання цього числа до старого повідомленням. Випадкові числа, використовувані таким чином, служать для фіксації відносної точки в часі для сторін-учасниць, аналогічно стандартним методам виміру часу. Максимально допустимий час між повідомленнями протоколу зазвичай обмежується деякою наперед заданою величиною, яка забезпечується за допомогою локальних, незалежних засобів виміру часу.

2. **Послідовності чисел.** З деякої послідовності чисел (наприклад, порядкової нумерації) дістається номер, що ідентифікує повідомлення, і зазвичай використовується для виявлення повторного відтворення повідомлення. Порядкові номери специфічні для певної пари сутностей і повинні явно або неявно асоціюватися як з відправником, так і з одержувачем повідомлення; для повідомлень від A до B і від B до A зазвичай потрібні різні послідовності.
   Сторони дотримуються заздалегідь визначеної політики нумерації повідомлень. Повідомлення приймається тільки в тому випадку, якщо зазначений в ньому порядковий номер не використовувався раніше (або ним не користувалися раніше протягом певного періоду часу) і задовольняє цю політику. Найпростіша політика полягає в тому, що номер послідовності починається з нуля, послідовно збільшується, і кожне наступне

повідомлення має номер на одиницю більше, ніж попереднє отримане.

3. **Часові помітки.** Часові помітки можуть гарантувати унікальність повідомлення, запобігаючи атаки через повторення повідомлення, а також допомогти помітити інші типи атак, як то повторення чи затримка повідомлень. Сторона, що відправляє повідомлення, отримує часову помітку з локального часу і криптографічно прив'язує її до повідомлення. Отримавши повідомлення з часовою міткою, інша сторона отримує поточний час зі свого локального годинника і віднімає отриману часову помітку. Отримане повідомлення є дійсним за умови, що різниця часів є у допустимому проміжку (який залежить від швидкості обміну повідомленнями та задається в залежності від системи), а також що це єдине повідомлення з цією міткою від цієї сутності.

2.2.2 Виклик-відповідь з симетричним ключем

Під **алгоритмами з симетричним ключем** розуміються алгоритми, що використовують однакові криптографічні ключі як для шифрування текстових даних, так і для дешифрування. Прикладом протоколу виклик-відповідь, що використовує симетричні ключі, може слугувати Kerberos.

Такий протокол може бути як одностороннім, так і двостороннім, де обидві сутності повинні переконати одна одну, що кожна з них має доступ до спільного ключа, не пересилаючи його, таким чином запобігаючи необхідності передавати ключ по спільному каналу. Далі розглянуто приклади технік для виклик-відповідь протоколів з симетричним ключем:

Одностороння автентифікація:

$$A \rightarrow B: E_k(t_A, B^*),$$

де А та В - відправник та перевіряючий, E_k - алгоритм шифрування з ключем К, t_A - часова помітка, B^* - ідентифікатор, що позначає отримувача.

Тут А відправляє В повідомлення разом із часовою міткою, яке шифрується. В отримує повідомлення, розшифровує його за допомогою ключа К та перевіряє, чи є часова мітка дійсною. Ідентифікатор В може бути використано, щоб запобігти використання зловмисником цього ж повідомлення на А.

Замість часової помітки можуть бути використані інші розглянуті в 2.2.1 способи генерації виклику. Так, для випадку використання випадкових чисел процес буде мати вигляд:

$$A \leftarrow B: r_B$$

$$A \rightarrow B: E_k(r_B, B^*),$$

де r_B - випадково згенероване число.

Двостороння автентифікація:

При використанні часових міток процес аутентифікації буде аналогічним до односторонньої схеми, але тепер в обидва боки.

При використанні випадкових чисел він набуде вигляд:

$$A \leftarrow B: r_B$$

$$A \rightarrow B: E_k(r_B, r_A, B^*)$$

$$A \leftarrow B: E_k(r_A, r_B)$$

Як і в односторонньому випадку, В перевіряє розшифроване повідомлення, і, додатково, отримує r_A, яке після цього надсилає А. А

отримує повідомлення і перевіряє випадкові значення, завершуючи процес аутентифікації.

2.2.3 Виклик-відповідь з відкритим ключем

Криптографія з відкритим ключем, або асиметрична криптографія, - це криптографічна система, яка використовує пару ключів: відкритий (який може бути відомий іншим) і закритий (який не повинен бути відомий нікому, крім власника).

Техніки криптографії з відкритим ключем можуть бути використані для ідентифікації в виклик-відповідь протоколах декількома способами: наприклад, суб'єкту автентифікації може бути запропоновано розшифрувати повідомлення, зашифроване його публічним ключем, або поставити на виклик свій цифровий підпис.

Потенційні небезпеки такої системи включають можливу вразливість до chosen-text атаки, а також поступову деградацію безпеки із використанням через накопичення інформації.

Далі розглянуто можливі протоколи:

Ідентифікація через розшифровку ПК-зашифрованого повідомлення.

$$A \leftarrow B\colon h(r), B, P_A(r, B)$$

$$A \rightarrow B\colon r$$

Тут B надсилає A *свідка* $h(r)$, де h - одностороння функція, що дозволяє продемонструвати знання r без його передачі, ідентифікатор B, а також виклик $P_A(r, B)$, де P_Aпозначає асиметричне шифрування. A розшифровує повідомлення та отримує r^* та B^*. A перериває виконання протоколу, якщо $h(r) \neq h(r^*)$ або $B^* \neq B$. Інакше A надсилає B отримане r, і у разі його коректності протокол успішно завершується. Вважається, що B ідентифікував A.

Іншим варіантом асиметричного протоколу аутентифікації з відкритим ключем є *протокол Нідгема-Шредера*[6]. Оригінальний протокол був вразливий до man-in-the-middle атаки, і пізніше модифікований Гевіном Лоу[7]. Модифікований протокол отримав назву *протокол Нідгема-Шредера-Лоу.*

$$A \rightarrow B: P_B(r_A, A)$$

$$A \leftarrow B: P_A(r_A, r_B, B)$$

$$A \rightarrow B: r_B$$

В оригінальному алгоритмі друга операція не містила ідентифікатор B, що дозволяло зловмиснику уособити A. Протокол потребує попереднього знання сторонами публічних ключів.

Виклик-відповідь автентифікація за допомогою цифрового підпису.

Автентифікація такого типу може бути як одностороння, так і двостороння, а також спиратися на часові помітки або випадкові числа.

Одностороння автентифікація з часовими мітками:

$$A \rightarrow B: cert_A, t_A, B, S_A(t_A, B),$$

де $cert_A$ позначає сертифікат, що містить відкритий ключ A, а S_A позначає алгоритм підписання, що використовується A.

Одностороння автентифікація з випадковими числами може бути застосована для позбавлення залежності від часових міток. У цьому випадку, B необхідно спочатку надіслати випадкове число r_B до A, і після цього a використовує його при підписанні.

Протокол *двосторонньої ідентифікації з випадковими числами* має вигляд:

$$A \leftarrow B: r_B$$

$$A \rightarrow B: cert_A, r_A, B, S_A(r_A, r_B, B)$$

$$A \leftarrow B: cert_B, A, S_B(r_B, r_A, A).$$

2.3 Використання zero-knowledge протоколів для автентифікації

Недоліком простих парольних протоколів є те, що коли А (хто в контексті zero-knowledge протоколів називається прувером (англ. prover)) повідомляє перевіряючому В свій пароль, В після цього може видати себе за А. Протоколи "виклик-відповідь" покращують цю ситуацію: А відповідає на виклик В залежно від часу, щоб продемонструвати знання секрету, надаючи інформацію, яку В не може використовувати безпосередньо. Проте, це все ще може розкрити деяку часткову інформацію про секрет; зловмисний верифікатор може бути здатний стратегічно вибирати виклики, щоб отримати відповіді, що надають таку інформацію, як у випадку chosen-text атаки.

Zero-knowledge протоколи побудовані таким чином, що вони дозволяють ідентифікувати доказуючого без розкриття будь-якої інформації, яку б верифікатор не був здатний добути до виконання протоколу. Ідея полягає в тому, що лише інформація про те, що прувер дійсно знає секрет, має бути передана.

Загалом задача zero-knowledge протоколів полягає у наведенні доказу твердження без передачі будь-якої інформації, що дозволяє уникнути великої кількості потенційних небезпек у криптографічних системах.

Zero-knowledge протоколи зазвичай є інтерактивними, де прувер та верифікатор обмінюються декількома повідомленнями. Задачею прувера є переконати верифікатора, що він дійсно знає секрет, а задачею верифікатора - встановити, чи прувер каже правду. У інтерактивному сценарії докази є ймовірнісними, а не абсолютними.

2.3.1 Proof of knowledge та zero-knowledge proof

Інтерактивні докази, використовувані для ідентифікації, можуть бути сформульовані як докази знання (proof of knowledge). A володіє деякими секретом s і намагається переконати B в тому, що він знає s, правильно відповідаючи на питання (з використанням загальновідомих вхідних даних і узгоджених функцій), для відповіді на які потрібно знання s.

Для того, щод інтерактивний доказ вважався proof of knowledge, він повинен мати наступні характеристики:

Повнота. Інтерактивний доказ є повним якщо, з умови чесності обох прувер та верифікатора, протокол завершується успішно з дуже високою ймовірністю. Точне значення ймовірності може варіюватися в залежності від використання.

Надійність. Інтерактивне доказ є надійним, якщо існує поліноміальний алгоритм M такий, що якщо нечесний прувер (видає себе за A) може з істотною ймовірністю успішно виконати протокол з B, то M може бути використаний для вилучення з цього пруверу знань (по суті еквівалентних секрету A), які з дуже великою ймовірністю дозволяють успішно виконувати наступні протоколи.

Proof of knowledge алгоритм вважається zero-knowledge, якщо існує такий поліноміальний алгоритм, який при введенні твердження, що потребує доказу, може надати результати, які не можна відрізнити від тих, що надав би прувер, без спілкування із прувером.

З твердження вище випливає, що прувер не відкриває ніякої інформації, яка не могла б бути обчислена на основі лише відкритих знань. Таким чином, виконання цього протоколу не підвищує шансів уособлення.

Zero-knowledge властивість може бути *абсолютною* або *обчислювальною.* У цьому випадку, властивість є обчислювальною якщо не є можливим вилучити секрет з результатів виконання протоколу у

осмислений проміжок часу. За замовчуванням у криптографії zero-knowledge має на увазі обчислювальну властивість.

2.3.2 Порівняння zero-knowledge протоколів із іншими асиметричними протоколами автентифікації

У цьому розділі розглянемо деякі характеристики zero-knowledge протоколів порівняно до інших протоколів, що використовують публічні ключі.

	ZK-протоколи	PK-протоколи
Деградація із часом	Відсутня; безпечність цих протоколів не зменшується внаслідок постійного використання, і вони не вразливі до chosen-text атак	Присутня; вразливі до chosen-text атак
Шифрування	Деякі техніки можуть запобігати його використання	Завжди використовується
Обчислювальна ефективність	Ефективність варіюється залежно від техніки; деякі протоколи є дуже ефективними внаслідок інтерактивної природи	В середньому більш обчислювально ефективні
Залежність від недоказаних припущень	Залежні	Залежні

Табл. 1 Порівняння ZK- та інших PK-протоколів

Як можна побачити, основна перевага ZK-протоколів полягає у відсутності деградації із часом та невразливості до chosen-text атак. Але й інші важливі характеристики, як то обчислювальна ефективність, більш ефективних ZK-протоколів є не гіршими за PK-протоколи.

2.3.3 Протокол Фіата-Шаміра

Базова версія протокола Фіата-Шаміра є сугубо історичною. Більш сучасна та ефективна версія протоколу носить назву *протокол Фейга-Фіата-Шаміра*[8]. Різниця між протоколами заключається в тому, що оригінальний протокол задає лише одне однобітове питання на кожній ітерації протоколу, тоді як більш сучасна версія може надавати декілька викликів, значно покращуючи ймовірність при однакомій кількості ітерацій.

Одноразові початкові налаштування.

1. Деяка довірена сутність T (наприклад, онлайн-сервіс), обирає два випадкові великі прості числа p та q та обчислює $n = pq$, після чого публікує n, але зберігає p та q у секреті.
2. Кожен заявник A обирає набір з k секретних чисел $s_1, s_2, \ldots, s_k$ взаємно простих з n. Обчислюються $v_i \equiv s_i^{-2} (mod\ n)$.
3. A ідентифікує себе за допомогою свого відкритого ключа $(v_1, v_2, \ldots, v_k; n)$ і зберігає свій закритий ключ $s_1, s_2, \ldots, s_k$. Через обчислювально складність визначення квадратичних лишків вважається, що неможливо отримати його закритий ключ використовуючи відкритий.

Процедура.

1. A обирає випадкове число r, випадковий біт b, обчислює $x \equiv (-1)^b \cdot r^2$ та надсилає верифікатору B.
2. Верифікатор обирає набор чисел $a_1, a_2, \ldots, a_k$, де $a_i \in \{0, 1\}$ та надсилає їх A.
3. A обчислює $y = r s_1^{a_1} s_2^{a_2} \ldots s_k^{a_k} (mod\ n)$ та надсилає це число до B.
4. B перевіряє $y^2 = \pm\ x v_1^{a_1} v_2^{a_2} \ldots v_k^{a_k} (mod\ n)$, а також що $x \neq 0$

Ця процедура повторюється t разів та можлива за умови, що B має доступ до відкритого ключа A. Інакше A повинен надіслати йому цей ключ перед початком процедури. Якщо усі t ітерацій завершилися успішно, B вважає A ідентифікованим.

Ймовірність того, що для кожної ітерації зловмисник E, що намагається видати себе за A, залежить від кількості питань у виклику, тобто від величини k. Так як E не знає чисел $s_1, s_2, ..., s_k$, для уособлення їй необхідно вгадати усі числа a_i, що обере B, вибрати випадкове число y, обчислити x та надіслати його в якості першого кроку процедури, а після отримання чисел a_i просто повернути y. Так як усі a_i є одно бітовими, ймовірність вгадати їх дорівнює $\frac{1}{2^k}$. Після t ітерацій ця імовірність буде ставити $\frac{1}{2^{kt}}$, що при $k = 5$ та $t = 8$ приблизно дорівнює одному шансу на трильйон.

2.3.4 Протокол Шнорра

Альтернативою до протокола Фіата-Шаміра є *протокол Шнорра*[9]. Протокол Фіата-Шаміра використовує складність обчислення квадратичних лишків, тоді як протокол Шнорра спирається на дискретний логарифм.

Ця схема дозволяє проводити попередні обчислення, скорочуючи обчислення в реальному часі для прувера до одного множення по модулю простого q; таким чином, вона особливо підходить для користувачів з обмеженими обчислювальними можливостями. Подальше покращення ефективності обчислень досягається за рахунок використання підгрупи порядку q мультипликативной групи цілих чисел по модулю p, де q ділить (p-1); це також зменшує необхідну кількість переданих бітів. Нарешті,

протокол був розроблений таким чином, щоб було потрібно всього три проходи і не потребувалася висока пропускна здатність каналу зв'язку.

Далі розглянуто сам протокол:

Одноразові початкові налаштування.

1. Обирається велике (достатньо для того, щоб обчислення дискретного логарифму було практично неможливим) просте число p, таке щоб p-1 ділилося на інше просте число q.
2. Обирається ціле число$\beta \in [1, p - 1]$ з мультиплікативним порядком q.
3. Усі сторони мають доступ до системних параметрів (p, q, β)і до публічного ключа довіреної сутності T, що дозволяє верифікувати цифровий підпис T S_T.
4. Обирається параметр t, $2^t < q$.
5. Кожен користувач A отримує унікальний ідентифікатор I_A.
6. Кожен A обирає закритий ключ a і обчислює $v = \beta^{-a} mod\ p$.
7. A передає v до T і отримує $cert_A = (I_A, v, S_T(I_A, v))$.

Процедура.

1. A обирає випадкове число r, обчислює $x = \beta^r modp$, і надсилає x та $cert_A$до B.
2. B отримує публічний ключ A з $cert_A$та надсилає A випадкове ціле число $e \in [1, 2^t]$ - виклик в цьому протоколі.
3. A надсилає B результат обчислення $y = ae + rmodq$. y є відповіддю.
4. B перевіряє $x = \beta^y v^e modp$ і визнає A ідентифікованим, якщо це рівняння виконується.

Як і в випадку протокола Фіата-Шаміра, ймовірність того, що зловмисник може уособити користувача, залежить від параметру t. У протоколі Шнорра ця ймовірність дорівнює 2^{-t}. Таким чином, для забезпечення надійності протоколу необхідно обрати досить велике t. Тк як значення t обмежене значенням q, для побудови безпечної схеми необхідно велике q. Достатнім вважається $q \geq 2^{160}$.

2.3.5 Протокол альфа-бета

Протокол альфа-бета базується на складності факторизації великих чисел. Загальна ідея полягає в тому, що для достатньо великих чисел обчислювальна складність факторизації надто висока, щоб існувала істотна ймовірність того, що зловмисник може знайти фактори числа, тоді як користувач з доступом до свого закритого ключа має таку можливість.

Цей протокол дозволяє використання ряда zero-knowledge методів, як розглянуто далі. Протокол не потребує додаткових ітерацій, і більша кількість обчислень виконується на стадії налаштування, що робить його досить обчислювально ефективним.

Далі розглянуто безпосередньо протокол:

Стадія налаштування.

Одноразова взаємодія з довіреною сутністю Т.

Спочатку обирається параметр t, що визначає рівень безпеки. Заявник А обирає два випадкові числа s_1 та s_2довжини t такі, що $gcd(s_1, s_2) = 0$. Ці числа передаються Т, яка виконує операцію α.

Альфа:

1. Випадково обираються прості p та q,

   $2^t < p < s1, 2^t < q < s1$ та число r більше за 2^t.

2. Обчислюється $n = pq$,

   $z_1 = r^{-1}s_2^{-1}q^{-1} mod\, s_1 + ks_1$,

   $z_2 = rs_2 mod\, s_1 + k_2 s_1$, $2^t < k_1, k_2$,

   де k_1, k_2- випадкові числа.

3. Кроки 1 та 2 можуть бути виконані m разів, генеруючи m різних значень n та пар z_1, z_2.

4. Усі згенеровані значення n_i передаються А та В (можуть бути у вільному доступі), а відповідні пари z секретно зберігаються В.

Процедура.

Процедура для цього протоколу є дуже простою.

1. Верифікатор В обирає випадкове значення і та передає пару $(n_i, w_i = z_{1i} \cdot z_{2i})$ пруверу А.

2. А обчислює $\tilde{p}_i = z_{1i} z_{2i} n_i$ та $p_i = z_{1i} z_{2i} n_i mods_1$- *операція*β. Після цього А може обчислити фактори n_i, бо $p_i = \tilde{p}_i mods_1, q_i = n_i \div p_i$ та продемонструвати В їх знання декількома способами. *Прямий спосіб* полягає в тому, щоб просто надіслати В отримане число р. Тоді це n_i вважається використаним і не може використовуватися при подальших виконаннях протоколу. *Непрямий спосіб* полягає у використанні zero-knowledge технік, як, наприклад, квадратичних лишків: В надсилає А $x^2 modn$, А повертає найменше можливе значення х.

3. Якщо А успішно продемонстрував В знання факторів, автентифікація закінчується успішно, інакше - ні.

Протокол є повним, бо за умови чесних сутностей А та В А може легко факторизувати n, $z_{1i} z_{2i} n_i mod s_1 = pqr^{-1} s_2^{-1} q^{-1} r s_2 mod\, s_1 = p$

Протокол також є zero-knowledge, бо не відкривається ніяка інформація, яка допомогла би дістати секрет прувера.

Для ідентифікація зловмиснику потрібно вгадати фактор p, що можна зробити вгадавши секрет s_1. Ймовірність цього залежить від значення t, яке визначає множину значень s_1 та s_2. Так, при $t = 6$ вони можуть приймати значення між 100000 та 999999, тобто 900000 значень. Таким чином, ймовірність того, що зловмисник зможе автентифікуватися, дорівнює $(9 \times 10^{t-1})^{-1}$.

2.3.6 Порівняння zero-knowledge протоколів

Наведені вище протоколи Фіата-Шаміра, Шнорра та альфа-бета вирішують одну й ту ж саму проблему подібними методами. Вони мають переваги та недоліки порівняно один з одним, і використовуються в різних ситуаціях. Далі наведено порівняльна характеристика цих протоколів на основі кількох важливих для протоколів автентифікації факторів, таких як:

- обчислювальна складність;
- гарантія безпеки;
- залежність від третьої сторони та необхідність довіри до неї.

Характеристика	Фіат-Шамір	Шнорр	Альфа-бета

Обчислювальна складність (верифікатор)	Потребує дуже незначну кількість обчислень від верифікатора	Потребує відносно велику кількість обчислень від верифікатора	Потребує дуже незначну кількість обчислень від верифікатора
Обчислювальна складність (прувер)	Потребує відносно більше обчислень від заявника	Потребує лише одну операцію модульного множення від заявника	При прямому способі доказу потребує лише одну операцію модульного множення; інакше - більше
Залежність безпеки від припущень	Залежить від нерозв'язності добуття квадратичних лишків по модулю	Залежить від нерозв'язності дискретних логарифмів по модулю	Залежить від нерозв'язності факторизації великих чисел
Залежність від третьої сторони	Так	Так	Так
Використання пам'яті	Потребує k секретів та t ітерацій для досягнення задовільного рівня безпеки	Відносно мале використання пам'яті	Потребує порівняно великий об'єм пам'яті для збереження m пар значень від верифікатора

Табл. 2 Порівняння zero-knowledge протоколів автентифікації

Як можна побачити, кожен з протоколів має свої переваги та недоліки, і в залежності від умов використання різних протоколів може бути більш доцільним.

РОЗДІЛ 3 РЕАЛІЗАЦІЯ АЛЬФА-БЕТА ПРОТОКОЛУ

3.1 Інструменти реалізації

Зважаючи на результати дослідження у попередньому розділі найбільш цікавими для реалізації є zero-knowledge протоколи. З поміж них було обрано протокол альфа-бета, як один з більш гнучких та цікавих підходів до автентифікації.

Реалізовано протокол було мовою C++, бо ця мова історично пов’язана з криптографічними задачами та задачами, що потребують високу швидкість виконання/оптимізацію витрат ресурсів загалом. Крім того, C++ має потужну стандартну бібліотеку і велику кількість ресурсів, що можуть бути використані у проекті.

Для сборки програми використовувалася утилита make, для компіляції компілятор g++ (GCC 7.5.0). У якості робочого середовища використовувався редактор коду Visual Studio Code, що пропонує багато можливостей найбільш сучасних інтегрованих середовищ, але є набагато більш легким і потребує менше ресурсів.

3.2 Структура програми

Програма, відповідно до мови виконання, побудована за принципами ООП. Кожна сутність, що приймає участь у виконанні протоколу, реалізована у вигляді класу. Таким чином, у програмі присутні наступні класи:

- **Prover** - клас, що відповідає чесному користувачу-заявнику. Цей клас генерує випадкові числа (s_1, s_2), передає їх до довіреного сервісу, що далі генерує публічний ключ, та зберігає ці значення для подальшої аутентифікації за

допомогою операції β. Розмір (s_1, s_2)дорівнює заданому t та варіюється.

- **Verifier** - клас, що відповідає верифікатору. Об'єкт цього класу зберігає набір значень $(z_{1i}, z_{2i}), i = 1 \ldots m$, а також n_i. За необхідності значення n_iможна не зберігати, адже вони знаходяться в загальному доступі, але якщо пам'ять не є проблемою, то зберігати значення n разом із парами (z_1, z_2) зручніше. Клас Verifier також відповідає за надсилання "виклику" до прувера та перевірку його відповіді.
- **Provider** - клас, що відповідає довіреній третій стороні, яка отримує пару (s_1, s_2) та генерує n_i та (z_{1i}, z_{2i}). Цей клас виконує операцію α, викладає усі n_iу вільний доступ та передає (z_{1i}, z_{2i}) верифікатору. Операція α при цьому виконується m разів в залежності від зазначеного m.
- **Eavesdropper** - клас, що відповідає нечесному користувачу, що намагається видати себе за іншого. Припускається, що зловмисник має доступ лише до відкритої інформації, і тому може лише намагатися вгадати фактор.

Розберемо основні методи (безпосередньо альфа та бета). Метод альфа приймає на вхід значення t, а також пару випадкових чисел (s_1, s_2) довжини t. Далі цей метод генерує необхідні випадкові прості числа та обчислює значення n, z_1та z_2, після чого повертає їх та, за необхідністю, повторюється. Метод бета є набагато простішим і полягає в прстому обчисленні добутка n, z_1та z_2по модулю s_1.

3.3 Робота програми

Програма має набір вхідних опцій, що дозволяють регулювати параметри системи, такі як параметр безпеки t та кількість виконань альфа/згенерованих провайдером чисел m. Також є можливість задати, яка інформація буде доступна при виконанні. Повний список параметрів приведен на рис. 2.

```
Possible parametrs: [-h] [-p] [-P] [-v] [-V] [-e] [-E] [-t T] [-m M]
[-h]            Prints this
[-p] [-P]       View as prover
[-v] [-V]       View as verifier
[-e] [-E]       View as eavesdropper
[-t T]          Security settings, T is unsigned integer
[-m M]          How many numbers n are generated, M is unsigned integer
```

Рис. 2 Перелік можливих параметрів програми

- Параметр -h друкує повідомлення на рис. 2.
- -p та -P встановлюють доступну інформацію як таку для прувера.
- Аналогічно -v та -V встановлюють доступну інформацію як таку для верифікатора.
- -e або -E скривають всю інформацію, що не є відкритою.
- -t T дозволяє задати параметр безпеки, що визначає довжині секрету прувера. Тут T - невід’ємне цілочисельне значення.
- -m M задає кількість виконань операції α, що є важливим при використанні прямого способу демонстрації прувером знання секрету, бо після кожної такої автттентифікації nне може використовуватися в майбутньому.

Після запуску програми обчислюються числа s_1 та s_2 заданої довжини t та виконується операція альфа. Відкриті дані, а також дані, що доступні йому, виводяться користувачу.

```
Secret:
6458567400, 6102941369
Generated public key:
5479944514241849369 20317919466011023039 7719365523147284309 957221040589163789 6714742631220685319
```

Рис. 3 Інформація, доступна пруверу

Після цього виконання протоколу продовжується до успішного або невдалого завершення. За умови виконання протоколу чесними сторонами він завжди завершується успішно, як показано на рис. 4.

```
Prover got numbers 20243168945498220487 and 36488504406788306381 from verifier
Verifier got number 2667683383 as a factor of 957221040589163789 from prover
Authentication succesful!
```

Рис. 4 Приклад успішного завершення виконання протоколу

3.4 Підсумки реалізації

Розглянемо витрати часу на виконання протоколу, початкове налаштування, а також на спроби вгадати секрет, не знаючи його, в залежності від параметра безпеки t.

t	Час виконання протоколу	Час одноразової комунікації з T	Час на подбір числа зловмисником
3	$\sim 30ms$	$\sim 30ms$	$\sim 50ms$
5	$\sim 30ms$	$\sim 50ms$	$\sim 5000ms$
7	$\sim 30ms$	$\sim 70ms$	$\sim 200000ms$
100	$\sim 50ms$	$\sim 90000ms$	-
300	$< 100ms$	$\sim 540000ms$	-
500	$\sim 120ms$	$10528843ms$	-

Табл. 3 Витрати часу при виконанні протоколу

Можна побачити, що час, необхідний зловмиснику на знаходження секрета, зростає дуже швидко. Це є наслідком того, що факторизація числа

є дуже обчислювально складною задачею і не існує способів швидко відшукати фактори без додаткової інформації, яку зловмисник не може отримати через використання zero-knowledge технік.

Іншим цікавим моментом є те, що швидкість виконання протоколу залишається майже однаковою навіть при значному зростанні t, і лише час на попередню обробку значно збільшується. І дійсно, якщо подивитися на алгоритм, усі складні обчислення належать до операції альфа, яка виконується під час одноразового контакту з T. При використанні прямого способу доведення ідентичності пруверу необхідно виконати єдину операцію множення по моделю, а верифікатору - одну перевірку. Хоча такий підхід і приводить до того, що кожне із значень n може використовуватись лише один раз, він дозволяє проводити автентифікацію дуже ефективно.

При використанні непрямого способу доведення (наприклад, через квадратичні лишки) витрати часу на виконання протоколу дещо зростають як ціна за можливість використовувати кожне значення n декілька разів. Таким чином, протокол дозволяє балансувати затрати часу й ресурсів на стадії налаштування (якщо значення n швидко закінчуються, необхідно знов звернутися до їх джерела) та безпосередньо під час виконання протоколу.

Повний код програми розміщено у [10].

ВИСНОВКИ

У даній роботі було досліджено та проаналізовано існуючі протоколи автентифікації та реалізовано один з zero-knowledge протоколів - альфа-бета протокол.

1. Проаналізовано існуючі зараз типи аутентифікаційних протоколів, їх переваги та недоліки, області застосування та можливі вразливості. В цілому найбільш безпечними та одночасно досить ефективними виявилися інтерактивні zero-knowledge протоколи автентифікації з відкритим ключем. Це можна пояснити тим, що вони запобігають ряду можливих вразливостей через те, що вони не розкривають ніякої корисної інформації потенційним супротивникам, використовуючи для автентифікації лише інформацію про те, що користувач дійсно володіє секретом.
2. Проведено дослідження та порівняння існуючих zero-knowledge протоколів, виявлено їх відносні недоліки т переваги. Так, протокол Шнорра дозволяє мінімізувати обчислення прувера, протокол Фіата-Шаміра, навпаки, мінімізує обчислення на боці верифікатора. Протокол альфа-бета може мінімізувати обидва, але за ціною великої кількості попередніх обчислень та необхідності повторювати ці попередні обчислення при використанні прямого способу доведення знання.
3. Реалізовано альфа-бета протокол автентифікації. Саме цей zero-knowledge протокол було обрано через те, що він є одним з найбільш гнучких протоколів свого класу, і дозволяє використовувати різні способи доведення знання секрету.

Досліджено роботу протоколу та зроблено висновок, що вона є адекватною.

В цілому можна сказати, що область автентифікації, незважаючи на свою ключову роль у комп'ютерних системах, все ще має багато невирішених проблем, а деякі математичні відкриття можуть легко перевернути вже існуючі знання та скомпрометувати навіть добре захищені системи. У сьогоденній реальності, коли значна частина людського життя залежить від інтернету та комп’ютерних систем, вопрос автентифікації стоїть дуже гостро і привертає увагу людей як з добрими, так і з поганими намірами. З часом нові типи атак і нови протоколи будуть розроблені, і можливо ті протоколи автентифікації, що зараз вважаються одними з найкращих, будуть навіть не близько такими.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. https://www.ojp.gov/pdffiles1/nij/225333.pdf
2. Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone. Handbook of applied cryptography. CRC Press, October 1996 - 816p.
3. B. Lloyd, W. Simpson. PPP Authentication Protocols. October 1992 https://datatracker.ietf.org/doc/html/rfc1334
4. G. B. Purdy; A high security log-in procedure. Cohmun. ACM 17(B), 442-445 (1974).
5. https://www.techopedia.com/definition/26138/challenge-response-authentication
6. Needham, Roger; Schroeder, Michael. Using encryption for authentication in large networks of computers. December 1978
7. Lowe, Gavin. An attack on the Needham-Schroeder public key authentication protocol. November 1995
8. Feige, Uriel; Fiat, Amos; Shamir, Adi. Zero-knowledge proofs of identity. *Journal of Cryptology,* June 1988.
9. C. P. Schnorr. Efficient identification and signatures for smart cards. 1989
10. https://github.com/NotGodsSlave/diploma2021

ДОДАТОК А Скріншоти з кодом програми

```
class Prover
{
public:
    Prover(int t, gmp_randstate_t state);
    void setNumbers(int t, gmp_randstate_t state);
    mpz_class gets1();
    mpz_class gets2();
    mpz_class beta(mpz_class z1, mpz_class z2, mpz_class n);
private:
    mpz_class s1, s2;
};
```

```
mpz_class Prover::beta(mpz_class n, mpz_class z1, mpz_class z2)
{
    std::cout << "Prover got numbers " << z1 << " and " << z2 << " from verifier\n";
    return (n * z1 * z2) % s1;
}
```

```
class Verifier
{
public:
    Verifier();
    void getData(std::vector<mpz_class> n, std::vector<mpz_class> z1, std::vector<mpz_class> z2);
    std::tuple<mpz_class,mpz_class,mpz_class> giveRandomChallenge(bool straight);
    bool verify(mpz_class p);
private:
    std::vector<mpz_class> n;
    std::vector<mpz_class> z1;
    std::vector<mpz_class> z2;
    mpz_class current_check;
};
```

```
std::tuple<mpz_class,mpz_class,mpz_class> Verifier::giveRandomChallenge(bool straight)
{
    srand(time(NULL));
    int i = rand()%n.size();
    std::tuple<mpz_class,mpz_class,mpz_class> res (n[i],z1[i],z2[i]);
    current_check = n[i];
    if (straight)
    {
        n.erase(n.begin()+i);
        z1.erase(z1.begin()+i);
        z2.erase(z2.begin()+i);
    }
    return res;
}
```

```
class Provider
{
public:
    Provider(){};
    mpz_class generatePrimeBetween(gmp_randstate_t state, mpz_class low, mpz_class high);
    std::tuple<mpz_class,mpz_class,mpz_class> alpha(mpz_class s1, mpz_class s2, unsigned int t, gmp_randstate_t state);
    void setup(std::vector<mpz_class> &n, std::vector<mpz_class> &z1, std::vector<mpz_class> &z2,
               int m, mpz_class s1, mpz_class s2, unsigned int t, gmp_randstate_t state);
};
```

```
tuple<mpz_class,mpz_class,mpz_class> Provider::alpha(mpz_class s1, mpz_class s2, unsigned int t, gmp_randstate_t state)
{
    mpz_class low;
    mpz_ui_pow_ui(low.get_mpz_t(),2,t);
    mpz_class q, p, r, k1, k2;

    //generating all the primes we need for this operation
    q = generatePrimeBetween(state,low,s1);
    p = generatePrimeBetween(state,low,s1);
    r = generatePrimeBetween(state,low,s1);
    k1 = generatePrimeBetween(state,low,s1);
    k2 = generatePrimeBetween(state,low,s1);
    //n = p*q
    mpz_class n = p*q;

    //inverting values we need inverted
    mpz_class r_invert, q_invert, s2_invert;
    mpz_invert(r_invert.get_mpz_t(),r.get_mpz_t(),s1.get_mpz_t());
    mpz_invert(q_invert.get_mpz_t(),q.get_mpz_t(),s1.get_mpz_t());
    mpz_invert(s2_invert.get_mpz_t(),s2.get_mpz_t(),s1.get_mpz_t());

    //calculating z1
    mpz_class z1 = (r_invert * q_invert * s2_invert) % s1 + k1 * s1;

    //calculating z2
    mpz_class z2 = (r * s2) % s1 + k2 * s1;
    tuple<mpz_class,mpz_class,mpz_class> res (n,z1,z2);
    return res;
}
```

```
class Eavesdropper
{
public:
    mpz_class guess(int t, gmp_randstate_t state);
    mpz_class beta(mpz_class n, mpz_class z1, mpz_class z2, unsigned int t, gmp_randstate_t state);
};
```

```
mpz_class Eavesdropper::guess(int t, gmp_randstate_t state)
{
    mpz_class low, high;
    mpz_ui_pow_ui(low.get_mpz_t(),10,t-1);
    mpz_ui_pow_ui(high.get_mpz_t(),10,t);
    mpz_class diff = high-low;

    mpz_class guessed;
    mpz_urandomm(guessed.get_mpz_t(),state,diff.get_mpz_t());
    guessed += low+1;
    return guessed;
}
```

```
void Prover::setNumbers(int t, gmp_randstate_t state)
{
    mpz_class low, high;
    mpz_ui_pow_ui(low.get_mpz_t(),10,t-1);
    mpz_ui_pow_ui(high.get_mpz_t(),10,t);
    mpz_class diff = high-low;

    mpz_urandomm(s1.get_mpz_t(),state,diff.get_mpz_t());
    s1 += low+1;

    mpz_urandomm(s2.get_mpz_t(),state,diff.get_mpz_t());
    s2 += low+1;
    mpz_class gc = gcd(s1,s2);
    while (gc != 1)
    {
        mpz_urandomm(s2.get_mpz_t(),state,diff.get_mpz_t());
        s2 += low+1;
        gc = gcd(s1,s2);
    }

    //std::cout << "Secret numbers for A are: " << s1 << ' ' << s2 << '\n';
}
```