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

Кваліфікаційна робота

на здобуття ступеня бакалавра

за спеціальністю 122 Комп'ютерні науки

на тему:

Швидка автентифікації в Інтернеті речей

Виконав студент 4-го курсу
Смельський Даніїл Сергійович

(підпис)

Науковий керівник:
Член-кореспондент НАН України, професор,
доктор фізико-математичних наук
Анісімов Анатолій Васильович

(підпис)

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

Студент _____

(підпис)

Київ - 2022

Реферат

Обсяг роботи 54 сторінки, 14 джерел, 1 таблиця

ПРОТОКОЛИ АВТЕНТИФІКАЦІЇ, ІНТЕРНЕТ РЕЧЕЙ, ПРОТОКОЛИ ІЗ НУЛЬОВИМ РОЗГОЛОШЕННЯМ, ШВИДКА АВТЕНТИФІКАЦІЯ

Об'єктом дослідження у даній роботі є оптимальні протоколи швидкої автентифікації для застосування в інтернеті речей. Об'єктом розробки є симулятор автентифікації з використанням протоколу Фіата-Шаміра.

Інструменти розроблення: операційна система macOS Monterey Version 12.3, ноутбук MacBook Air (M1, 2020) із 16 GiB оперативної памьяті, мова програмування C++, компілятор Clang 13.1.6, CMake, make, самописна бібліотека для роботи із великими числами та криптографічними обчисленнями, open-source бібліотека easy-profiler для замірів часу роботи програми.

Метою даної роботи є аналіз особливостей застосування протоколів автентифікації в інтернеті речей, а також порівняння кількох достатньо ефективних протоколів. Оптимальними протоколами для порівняння було обрано протокол Нідгема-Шредера, протокол Фейге-Фіата-Шаміра, а також протокол Шнорра.

Результати роботи: досліджено особливості застосування різних протоколів автентифікації в інтернеті речей, у порівнянні зі звичайними протоколами автентифікації. Розглянуто основні обмеження у сфері інтернету речей та як вони впливають на вибір протоколів автентифікації, а також додаткові корисні особливості даної сфери, які можуть бути застосовані для покращення процесу автентифікації. Також для кожного з обраних оптимальних протоколів було досліджено його ефективність, спираючись на кількість використаної памьяті, трафіку та складність обчислень. Реалізовано симулятор автентифікації із застосуванням протоколу Фіата-Шаміра та проаналізовано час роботи даного алгоритму.

Протоколи автентифікації дуже важливі, оскільки вони напряму використовуються для ідентифікації суб'єктів, що дає можливість отримати доступ до цінних та особистих даних, які мають бути надійно захищеними, оскільки до таких даних належать дані кредитних карт, цінні папери та документи, фотографії та інше. Особливості архітектури приладів інтернету речей додають певні обмеження на застосування звичайних протоколів, наприклад обмеження на пам'ять та потужність процесора, тому для приладів інтернету речей потрібно використовувати інші, більш ефективні алгоритми.

В даній роботі було досліджено достатньо оптимальні протоколи автентифікації, які не потребують багато памьяті, трафіку та потужності комп'ютера, тому вони можуть застосовуватись майже на будь-якому приладі, в залежності від його особливостей. Кожен з розглянутих протоколів забезпечує дуже високий захист від атак зловмисника, тому щодо безпеки таких протоколів можна не хвилюватись, оскільки вірогідність обійти такий протокол через відомі атаки дуже мала.

Зміст

Скорочення та умовні позначення

CA - Central authority

AUC - Authentication center

3-rd party - третя особа

IoT - Internet of Things

KiB - KibiByte

GiB - GibiByte

Вступ

Оцінка сучасного стану об’єкта розробки.

За статистикою у 2021-у році до міжнародної мережі було підключено близько 13.8 мільярдів активних приладів інтернету речей. Ще у 2010-у році їх було менше ніж мільярд, а у 2025-у році прогнозують близько 30.9 мільярдів активних приладів, і це тільки ті що підключені до міжнародної мережі інтернету. Кількість таких приладів росте надзвичайними темпами, як і популярність до них.

З появою сфери інтернету речей інженери зіткнулись із новими труднощами, пов'язаними зі специфікою цих приладів. Через, як зазвичай, відносно малу ціну та розмір, такі прилади мають досить обмежені ресурси, наприклад пам'ять та процесорну потужність, тому звичайні алгоритми та рішення для них можуть не підійти.

Однією із важливих проблем є безпека даних. Багато приладів передають через інтернет якісь особисті для людини дані, наприклад, це можуть бути дані вашої кредитної картки, коли ви оплачуєте якусь покупку онлайн на сайті, або відео з ваших камер відеоспостереження, або якісь дані про ваше самопочуття, такі як пульс, температура та інше. Навіть якщо ви не регулюєте процес передачі цих даних, ваш прилад зазвичай передає всі ці дані самостійно, щоб зберігати їх у вашому особистому кабінеті, або опрацьовувати на більш потужних машинах на сервері. Зловмисник, отримавши доступ до цих даних, може скористатись ними у власних цілях, і це завдасть вам великої шкоди, тому всі операції із такими даними треба дуже надмірно захищати.

Актуальність роботи та підстави для її виконання.

В першу чергу коли ваш прилад вмикається та підключається до сервера, серверу треба впевнитись в тому хто саме до нього підключається. Тобто, дізнатись що це за прилад, кому він належить, чи це точно саме цей прилад, чи точно саме цей користувач володіє ним зараз. Тому кожен прилад треба аутентифікувати.

Аутентифікація пов'язана не тільки із приладами інтернету речей. Коли ви заходите на будь-який сайт, де ви зареєстровані як певний користувач, вам також доводиться проходити етап аутентифікації. Зазвичай для цього користувач повинен ввести свій логін (або адресу електронної пошти) та пароль. Існує безліч інших методів аутентифікації, проте це найпопулярніший для звичайних сайтів.

Більшість із таких протоколів використовують різні криптографічні алгоритми, які потребують комп'ютерів із певною потужністю. Прилади інтернету речей можуть взагалі не підтримувати такі алгоритми, або підтримувати, але при цьому сильно перегріватись, споживати забагато електроенергії, запасів батареї, працювати недостатньо ефективно, та інше.

Тому для приладів інтернету речей потрібно розробляти нові протоколи, з урахуванням усіх особливостей та обмеженостей приладу.

Мета й завдання роботи.

Метою роботи є дослідити оптимальні протоколи автентифікації для сфери інтернету речей, визначити загальні особливості та недоліки протоколів для даної сфери, та проаналізувати кожен із протоколів на ефективність.

Можливі сфери застосування.

Пристрої інтернету речей використовуються дуже широко. Це, наприклад, технології розумного дому, медицина, транспорт, сільське господарство, геологія, енергоспоживання. Деякі напрямки навіть виділені в окремі сфери інтернету речей, наприклад інтернет машин, інтернет енергії, інтернет датчиків.

З іншого боку, сама технологія аутентифікації застосовується не тільки в інтернеті речей. Вона застосовується для будь-яких онлайн сервісів, соціальних мереж, приватних підприємств.

Розділ 1. Автентифікація

1.1 Поняття процедури автентифікації

Автентифікація - це процес перевірки даних на справжність. Частинним випадком автентифікації є ідентифікація - це процес перевірки того, що суб'єкт є тим за кого він себе видає.

Автентифікація зазвичай використовується у таких інтернет сервісах як:

- електронна пошта
- web-форум
- соціальні мережі
- інтернет-банкінг
- платіжні системи
- корпоративні сайти
- інтернет-магазини

Процес автентифікації виконується з самого початку роботи з будь-яким сервісом переліченим вище, або програмою, що підтримує особисті профілі. Для успішного проходження процедури автентифікації користувачу треба пред'явити свій ідентифікатор (зазвичай це адреса електронної пошти, або ім'я чи логін) та секретну інформацію, яку має знати тільки цей користувач, або група людей, що володіють цим профілем. У вигляді секретної інформації зазвичай використовують пароль - секретну фразу, або певну послідовність символів, проте існують й інші види секретних даних.

Після успішної автентифікації користувач проходить етап авторизації - це процес перевірки прав на володіння певними даними, або доступу до них. Цей процес зазвичай виконується самим сервісом, без комунікації із користувачем. Після успішної авторизації користувачу надається доступ до необхідних ресурсів, якими він володіє, наприклад доступ до профілю, особистих даних та банківських рахунків.

Етап автентифікації є дуже важливим, оскільки він захищає особисті дані користувача від посторонніх людей. Попри велику кількість різних протоколів автентифікації, кожен з них має свої недоліки, і жоден не гарантує стовідсоткову гарантію безпеки ваших даних, проте існують протоколи, що зменшують вірогідність викрадення володіння вашим профілем фактично до нуля.

1.2 Процедура автентифікації

Для проведення процедури автентифікації потрібно як мінімум два об'єкти: користувач та автентифікатор. Існують протоколи де до процесу автентифікації можуть бути залучені треті особи (сервіси), які можуть виступати як незалежні довірені об'єкти, щоб зробити весь процес надійнішим.

Оскільки автентифікація це процес перевірки даних на справжність, автентифікатору спочатку треба дізнатись якими даними володіє користувач. Ця передача даних відбувається на етапі реєстрації. Цей етап відбувається лише один раз, на самому початку роботи із програмою (сервісом), коли користувач створює особистий профіль. На цьому етапі потрібно максимально надійно передати секретні дані, що будуть застосовуватись під час етапу ідентифікації, до автентифікатора. Це може відбуватись різними способами, навіть офлайн. Для технології інтернету речей цей процес може здійснюватись навіть під час виробництва приладу.

Після реєстрації сервер має надійно зберегти секретні дані для ідентифікації користувача, наприклад зашифрувавши їх, або іншими методами. Якщо зловмисник отримає доступ до сервера та ненадійно збережених даних про користувачів, то він може дізнатись їх.

Кожен наступний раз коли користувач буде заново заходити у систему, йому буде потрібно автентифікуватись. Насправді це робиться не кожен раз, а із певною періодичністю, оскільки кожен раз вводити особисті дані може викликати незручності, проте чим частіше користувача просять автентифікуватись заново, тим надійнішою є система.

Під час самого процесу автентифікації користувачу треба доказати автентифікатору що він є тим за кого він себе видає. Цей процес може суттєво відрізнятись для різних протоколів, оскільки є багато факторів які треба враховувати при розробці протоколу, наприклад складність необхідних обчислень (загалом криптографічних), бізнес цілі у яких буде використовуватись автентифікація (наприклад публічний, або корпоративний сервер мають різні цілі), важливість та цінність даних.

Проте найбільш поширеним є протокол автентифікації, де користувачу треба передати свої секретні дані, можливо у якомусь зашифрованому вигляді, до сервера, який порівняє їх з оригінальними даними отриманими під час реєстрації. Сам по собі цей протокол не є достатньо надійним, оскільки існує безліч способів перехопити дані під час обміну ними із сервером, але він має багато удосконалень, які підвищують його надійність.

1.3 Різновид протоколів автентифікації

Протоколи автентифікації можуть сильно між собою відрізнятись, в залежності від цілей та особливостей їх застосування. Тому існує кілька загальних характеристик для класифікації таких протоколів.

1.3.1 Фактор автентифікації

Існує щонайменше три категорії даних, які можуть використовуватись для автентифікації:

- Знання - інформація яку знає суб'єкт. До цієї категорії відносяться паролі, секретна фраза, PIN-коди, приватний ключ та інше. Це найбільш поширена, основна категорія секретних даних що використовуються для автентифікації. Загалом ця категорія майже завжди використовується в онлайн сервісах та застосунках.
- Володіння - річ, якою володіє суб'єкт. До цієї категорії належать мобільний телефон, smart-card, флешпам'ять, електронна пошта. Ця

категорія зазвичай використовується як додаткова міра захисту, оскільки до неї важко отримати доступ.

- Властивість - це певна особливість якою володіє суб'єкт. До цього відносять різні біометричні дані, наприклад, відбиток пальця, сітківка ока, обличчя, ДНК та інші унікальні властивості людини.

Мультифакторна автентифікація - це метод автентифікації, у якому застосовується як мінімум два різні фактори ідентифікації. Для першого фактору зазвичай обирають знання, оскільки це найпростіший та найпоширеніший вид доказу. Проте щоб збільшити надійність даних додають застосування другого фактору, наприклад прив'язують профіль до номера мобільного телефону, або до програми на особистому телефоні, або, в корпоративних компаніях можуть використовувати ще більш надійний спосіб - smart-card, це електронні прилади що генерують код для аутентифікації.

Мультифакторна автентифікація забезпечує більшу надійність даних від дистанційних атак, оскільки для того, щоб успішно пройти автентифікацію, потрібно буде заволодіти чимось фізичним.

1.3.2 Напрямок автентифікації

Процедура автентифікації може використовуватись не тільки для автентифікації однієї особи, наприклад клієнта, а вона може так само використовуватись для обох осіб, щоб і клієнт і сервіс були впевнені, що вони не спілкуються із самозванцем. Існує 3 основні види автентифікації за напрямком:

- Одностороння (one-way) - коли лише одна з двох, або більше, осіб автентифікує себе перед іншими учасниками діалогу.
- Двостороння (two-way, або mutual authentication) - автентифікація у якій беруть участь дві особи, та обидві автентифікують себе один для одного.
- Тристороння (three-way) - метод автентифікації у якому третя особа (3-rd party), наприклад CA, допомагає провести процедуру

двосторонньої автентифікації. Така третя особа допомагає централізувати цей процес.

Двостороння автентифікація використовується, наприклад, у протоколі HTTPS, який використовує SSL та TLS сертифікати. Цей протокол автентифікує обидві сторони, і дає змогу як клієнту, так і серверу, бути впевненим із дуже великою ймовірність, що він спілкується із тим, із ким має спілкуватись. Також завдяки цьому протоколу відбувається не тільки процес автентифікації, а й процес встановлення безпечного, зашифрованого зв'язку.

1.3.3 Архітектура автентифікації

Загалом розрізняють два основні види архітектури для сервісу автентифікації:

- Децентралізована архітектура: дуже надійний вид архітектури, у якому немає єдиного центру автентифікації, а його роль виконують самі користувачі, через що деякі атаки стають складнішими, або навіть неможливим. Такий вид архітектури може застосовуватись, наприклад у блокчейні.
- Централізована архітектура: більш поширений вид архітектури автентифікації у якій виділяють єдиний (або групу з кількох) центр автентифікації. Ця архітектура має більшу сферу застосування та легша в реалізації ніж децентралізована, проте піддається деяким відомим атакам, а також має такий недолік як Single Point of Failure (єдина точка помилки). До цієї групи архітектур також входить архітектура у якій виділяють окремий сервіс для автентифікації, наприклад у протоколі Kerberos.

Розділ 2. Інтернет речей (Internet of Things)

2.1 Поняття інтернету речей

Інтернет речей - це концепція мережі, що складається із пов'язаних між собою фізичних пристроїв, кожен з яких має програмне забезпечення, яке реалізує як мінімум обмін даними з іншими пристроями цієї мережі завдяки стандартним протоколам зв'язку, а також взаємодію із фізичним світом. Ця мережа складається із підмереж - екосистем різних пристроїв.

Призначенням цієї концепції є підключення усіляких приладів, які можуть збирати будь-яку інформацію за допомогою датчиків, та використовуватись людиною в різних повсякденних цілях. Фактично, до такої мережі можна підключити все, що має необхідне програмне забезпечення, або може бути підключеним до нього, наприклад: відеокамери, холодильник, вікна, двері, водопровідна система, опалення, кондиціонер, або навіть одяг. Такі прилади набувають велику популярність загалом завдяки тому що вони компактні, здебільшого дешеві, легкі в застосуванні, та допомагають з автоматизацією різних процесів.

Особливість таких приладів в тому, що зазвичай до них підключаються різні фізичні датчики, які трансформують аналогові дані у цифрові, ці дані опрацьовуються програмним забезпеченням пристрою, і далі пристрій може ділитись результатами з іншими пристроями, а також трансформувати якісь дані із цифрового формату назад в аналоговий, і тим самим взаємодіяти з іншими фізичними пристроями.

Прикладами підключених датчиків можуть бути, наприклад:

- датчик температури, освітлення, вологості повітря, тиску, відстані
- хімічні датчики, газ та якість повітря
- біомедичні датчики
- радар, гіроскоп, акселерометр
- камери та оптичні сенсори

Чудовим прикладом інтернету речей є екосистема розумного дому. Це мережа що складається із різних розумних пристроїв здатних комунікувати один з одним, автоматично зчитувати та опрацьовувати дані з різних датчиків, та надсилати різні повідомлення користувачу, або навпаки опрацьовувати команди користувача. До такої мережі входять такі прилади як: камери відеоспостереження, автоматичні ліхтарі, датчики газу, диму та затоплення, та інші.

З ростом кількості пристроїв виокремились певні загальні підмережі інтернету речей, що мають певне призначення та набувають окрему популярність:

- Інтернет енергії (Internet of Energy) - термін що використовується для вдосконалення та автоматизації інфраструктури електроенергії для споживачів та виробників.
- Інтернет транспорту (Internet of Vehicles) - це набори транспортних засобів, оснащених датчиками, програмним забезпеченням і технологіями, які є посередниками між ними, з метою підключення та обміну даними через Інтернет відповідно до узгоджених стандартів.

Прилади інтернету речей не зобов'язані бути підключеними до загальної мережі інтернету, вони можуть підключатись до певної локальної мережі, або утворювати свою, децентралізовану мережу та, наприклад, спілкуватись на своїй частоті один з одним.

2.2 Особливості пристроїв інтернету речей

Головним елементом пристроїв інтернету речей завжди є мікроконтролер - мікросхема, призначена для управління електронними приладами. На мікроконтролері розташовані мікропроцесор, пам'ять, таймери, аналого-цифровий перетворювач, спеціальні чіпи та інше.

Також до мікросхеми можуть бути підключені будь-які датчики за допомогою спеціальних портів, або вони можуть бути інтегровані у сам чіп мікросхеми.

Більшість пристроїв інтернету речей мають бути компактними, щоб не займати багато місця, щоб їх було легше встановлювати, і щоб вони були дешевшими. Обмеження на розмір додає деякі труднощі для розробника, наприклад, архітектуру таких приладів складніше розробляти, оскільки треба акуратно помістити та об'єднати все необхідне. Щоб якось вдосконалити мікросхему, або додати новий датчик, можливо навіть доведеться повністю змінювати дизайн пристрою.

Оскільки такі пристрої зазвичай розраховані на великий ринок, для звичайних людей, вони повинні мати доступну ціну, тобто бути якомога дешевшими. Через це у таких пристроях варто використовувати лише те що необхідно, аби користувач не переплачував за те, що він не використовує.

Для пристроїв інтернету речей зазвичай використовуються спеціальні мікросхеми, які майже ніколи не використовуються в повноцінних персональних комп'ютерах, телефонах, ноутбуках та інших пристроях, що розраховані на складніші задачі.

В таких мікросхемах використовуються відносно дешеві мікропроцесори, які розраховані на прості задачі, що не потребують складних обчислень, великого навантаження системи, багато електроенергії та великого обсягу пам'яті. Це сильно зменшує ціну на пристрої, оскільки потужні процесори що використовуються у звичайних комп'ютерах коштують в рази більше, а також це сильно економить електроенергію, оскільки такі пристрої здебільшого мають працювати цілодобово.

На таких приладах варто приділяти окрему увагу кількості електроенергії яку він використовує, оскільки деякі з цих приладів можуть працювати не від мережі, а від акумулятора, батарейки, або сонячних панелей, які в свою чергу можуть працювати із різною ефективністю залежно від погоди та часу доби. Навіть якщо прилад має достатнє живлення, він може перегріватись, через що він може виключитись, зламатись, або навіть згоріти.

Через такі обмеження на потужність приладів інтернету речей, вони здатні підтримувати не всі алгоритми, бо деякі з них будуть працювати занадто повільно, або зовсім не будуть працювати. Через це такі алгоритми треба або сильно оптимізовувати та обмежувати під конкретні задачі, або розробляти нові аналоги алгоритму та шукати інший розв'язок проблеми, або змінювати архітектуру приладу та розширювати його функціонал, що понесе за собою ріст ціни на нього.

Проте з іншого боку прилади інтернету речей мають і свої переваги:

- Вони компактні та прості у вмонтуванні, через це їх легко кудись встановити не виділяючи на це багато місця, можливо навіть на потолок.
- Вони відносно дешеві, зазвичай в багато разів дешевші за персональні комп'ютери, через це вони досить доступні.
- Внаслідок простих компонент та спеціальної архітектури вони зберігають електроенергію.
- Такі прилади зазвичай максимально автоматизовані, вони збирають та автоматично опрацьовують дані з датчиків, самостійно роблять відповідні висновки та повідомляють про це користувачів, що спрощує їх життя.
- Користувач не переплачує за те що він не використовує.

2.3 Проблеми із безпекою

Прилади інтернету речей можуть обмінюватись різними повідомленнями та даними один з одним, або із різними сервісами в інтернеті. Коли користувач якось керує своїми приладами за допомогою, наприклад мобільного застосунку, то насправді користувач зазвичай спілкується не з приладом напряму, а із певним сервісом в інтернеті, який, в свою чергу, спілкується із самим пристроєм.

Прилад може сам опрацьовувати усі дані зчитані з датчиків, і передавати на сервер лише результати чи висновки, отримані з їх аналізу, або він може просто зчитувати дані з датчиків і одразу ж їх передавати, не опрацьовуючи локально. Останній варіант може використовуватись через особливості архітектури, або якщо йому просто невистачає потужності, щоб опрацьовувати ці дані, оскільки вона досить обмежена. Цей процес називають хмарними обчисленнями, оскільки сервер можна вважати хмарою - віддаленим сховищем даних, і він набирає дуже велику популярність, оскільки має багато привілей.

Деякі дані які прилад надсилає серверу можуть бути дуже приватними. Наприклад це може бути відео з камери відеоспостереження, або якісь біомедичні показники, або інша інформація, якою може скористатись зловмисник у своїх цілях. Такі дані треба добре шифрувати, застосовуючи надійні протоколи.

Також треба забезпечувати максимальну надійність керування пристроєм, щоб жоден зловмисник не зміг видати себе за власника пристрою, та керувати ним як він захоче. Тому потрібно застосовувати надійні методи автентифікації, щоб сервер був впевненим із яким саме пристроєм він спілкується, чи вірно передані дані з пристрою, та якому користувачу він передає ці дані. Та з іншого боку, пристрій має бути впевненим що він спілкується із конкретним сервером, а не зловмисником, що намагається себе видати за нього та отримати від пристрою всі необхідні дані.

Такі проблеми з безпекою притаманні не тільки приладам інтернету речей, вони також мають забезпечуватись і звичайним браузером, через який ви авторизуєтесь на сайтах та інше. На щастя, в наші часи вже існують дуже надійні протоколи, що забезпечують гарну безпеку передачі даних в інтернеті.

Проте, більшість таких протоколів можуть не підійти для приладів інтернету речей, через їх особливості. По-перше, такі протоколи часто використовують складні криптографічні обчислення, які потребують певну потужність комп'ютера, щоб виконуватись ефективно. Такі протоколи можуть працювати занадто повільно, або перегрівати прилад, якщо він має відносно слабке залізо. По-друге, складні протоколи використовують багато електроенергії, яка має використовуватись досить економно на простих приладах. По-третє, кількість приладів інтернету речей вже є більшою ніж кількість інших приладів, тому об'єми навантаження цих приладів на мережу є досить значними.

Розділ 3. Різновид протоколів автентифікації

Автентифікація була дуже важливим процесом ще задовго до початку розвитку комп'ютерних технологій та мереж. Автентифікація застосовувалась для того, щоб впевнитись чи є певна річ, чи певний документ справжніми, наприклад при продажі, або передачі. Для цього треба було автентифікувати власника цієї сутності та документ, що підтверджує право на власність. Такими перевірками й досі може займатись нотаріус - людина, спеціально уповноважена на виконання нотаріальних справ, серед яких є: свідчення вірності копій документів і виписок з них, свідчення справжності підписів на документах, вірності перекладів документів з однією мови на іншу, а також деякі інші дії.

Проте, із розвитком комп'ютерних технологій та переходу на цифрові дані, цей процес треба було автоматизовувати, щоб це відбувалось із мінімальним залученням людини до процесу, аби весь процес був легшим, надійнішим, та швидшим.

В залежності від особливостей середи та цілей використання, існують різні загальні види класифікації протоколів автентифікації. Деякі з них розраховані для використання на потужних комп'ютерах, а деякі на слабких, наприклад приладах інтернету речей. Деякі розраховані для використання у публічних сервісах, наприклад вебсайтах в інтернеті, а деякі можуть застосовуватись у корпоративних компаніях, або у групі таких компаній. Більшість із них розраховані на мінімальну участь людини, а деякі можуть взагалі працювати без втручання людини та цілком виконуватись автоматично.

3.1 Автентифікація за паролем

Це дуже розповсюджений вид автентифікації який зазвичай використовуються, наприклад, у соціальних мережах та онлайн сервісах для звичайних користувачів. Це досить простий протокол, але через свою простоту він є малонадійним, оскільки піддається деяким атакам.

Суть цього протоколу полягає в тому, що користувач при реєстрації в системі вказує свій ідентифікатор (зазвичай ідентифікатори мають бути унікальними, наприклад, це може бути адреса електронної пошти, номер телефону, ПІБ, або будь-який унікальний рядок), та пароль - певний рядок, що може складатись із різних символів, наприклад, букв латинського алфавіту (як великих, так і маленьких), цифр та спеціальних знаків. Ідентифікатор також часто можуть називати логіном (від англ. слова login).

Ідентифікатор це як правило публічна інформація, користувач може не перейматись щодо його безпеки. На першому етапі автентифікації сервера треба повідомити свій ідентифікатор, щоб він розумів із ким розмовляє.

Далі серверу треба впевнитись що ви є тим за кого себе видаєте, тобто що ви маєте доступ до профілю зареєстрованого під цим ідентифікатором. Для цього користувачу треба повідомити серверу свій пароль. Знання пароля для ідентифікатора це доказ того, що ви маєте до нього доступ, тому пароль завжди треба зберігати якомога надійніше та нікому його не розголошувати, бо будь-який зловмисник, дізнавшись пароль від вашого ідентифікатора, автоматично отримає доступ до вашого профілю та ваших даних.

В такому вигляді даний протокол майже ніколи вже не використовується, оскільки його досить просто обійти, наприклад, перехопивши пароль коли користувач передає його до сервера, щоб ідентифікувати себе. Тому цей протокол має багато вдосконалень, які запобігають більшій кількості атак.

Одним із варіантів вдосконалення такого протоколу є використання безпечного каналу зв'язку, наприклад, протокол HTTPS (HyperText Transfer Protocol Secure) - це вдосконалення дуже популярного протоколу передачі сторінок між клієнтом (наприклад браузером користувача) та сервером (вебсайтом), завдяки використанню SSL та TLS сертифікатів.

SSL (Secure Sockets Layer) - це механізм шифрування даних, який застосовують для обміну повідомленнями (даними) в інтернеті. TLS (Transport Layer Security) - удосконалення механізму SSL. Ці механізми надають захист від атак, побудованих на прослуховуванні мережевого з'єднання.

Встановлення надійного зв'язку за допомогою протоколу HTTPS надає велику надійність того, що спілкування клієнта із сервером не зможуть підслухати, тому клієнт може безпечно передавати свій пароль до сервера для проходження автентифікації. Проте, існує багато інших атак, які й досі можуть обійти цей протокол та отримати доступ до вашого профілю.

Оскільки сервер кожен раз при автентифікації користувача порівнює надісланий ним пароль із паролем, що був наданий при реєстрації, серверу потрібно у себе зберігати оригінальний пароль для кожного такого користувача. Якщо якимось чином будь-хто отримає доступ до бази даних, що зберігає усі паролі користувачів, то ця людина заволодіє доступом до профілів всіх цих користувачів. Цим зловмисником може бути як співробітник компанії, так і хакер, який обійшов захист сервера.

Для того, щоб уникнути атаки такого виду, пароль треба зберігати у зашифрованому вигляді, наприклад, застосовуючи хеш-функції. Хеш-функція - це функція, що може приводити дані будь-якого розміру та змісту, до даних (зазвичай цілих чисел) фіксованого, або обмеженого розміру, що самі по собі не мають жодного сенсу. Гарна хеш-функція має розподіляти усі можливі вхідні дані рівномірно на вихідні дані, тобто кожне можливе значення із вихідних даних повинно мати однакову вірогідність, проте хеш-функція, для одного й того самого числа, зобов'язана завжди повертати один і той самий результат. Цей факт використовується у різних структурах даних, наприклад, хеш-таблицях, а також у

криптографічних обчисленнях. Оскільки множина вихідних даних обмежена, а вхідні дані можуть мати будь-який розмір, це значить, що зазвичай неможливо визначити для якого вхідного значення було отримане конкретне вихідне значення. До того ж оскільки розподіл є рівномірним, це ускладнює процес пошуку будь-якого числа для якого результат хеш-функції буде відповідати певному числу.

Тоді, отримавши пароль від користувача, при кожній автентифікації сервер одразу ж застосовує для цього паролю певну хеш-функцію (кожен раз в точності одну й ту саму, з однаковими параметрами), та порівнює вже зашифрований результат із тим, що був збережений при реєстрації користувача. Якщо хеш-значення співпадають, то з великою ймовірністю можна сказати що людина знає справжній пароль, або вгадала будь-яке число для якого значення хеш-функції буде те самим, що і для справжнього пароля.

Тепер, будь-хто, дізнавшись зашифрований пароль, не зможе увійти в систему, оскільки він не знає справжнього пароля. Якщо він спробує застосувати зашифрований пароль, що був збережений в базі даних на сервері при реєстрації, для автентифікації, то до цього паролю буде застосована хеш-функція, яка утворить новий пароль, що все одно майже гарантовано буде відрізнятись від того що був збережений про реєстрації. Однак, зловмисник може застосувати іншу атаку - атаку повного перебору. Він може перебирати усі можливі паролі до тих пір поки не знайде той, хеш-функція до якого буде відповідати тій, що збережена в базі даних на сервері. Саме тому хеш-функцію треба обирати дуже надійну, розраховану на те, щоб було майже неможливо знайти два числа, що відповідають одному й тому самому хеш-значенню. Більш того, користувач має обирати надійний, бажано довгий пароль, що складається із випадкових символів, щоб до нього було неможливо здогадатись. І бажано для різних сервісів завжди застосовувати різний пароль, оскільки інакше, отримавши пароль від одного сервісу, зловмисник зможе отримати доступ одразу до кількох, що захищені тим самим паролем.

3.2 Виклик-відповідь

У протоколах автентифікації за паролем є свої недоліки. По-перше, серверу необхідно надійно захищати пароль кожного користувача, щоб до нього ніхто не отримав доступ. При цьому цей захист пароля треба реалізувати не тільки для бази даних із паролями, а й під час усіх обчислень, що використовують цей пароль. По-друге, користувачу кожен раз треба якось безпечно передавати пароль через інтернет. Для захисту зв'язку можна застосовувати протокол HTTPS, проте він має свої недоліки, і це не завжди можливо.

Проблему із передачею пароля через інтернет до сервера можна частково вирішити застосувавши ідею виклик-відповідь (challenge-response). Сама ідея полягає в тому щоб передавати не сам пароль, а якийсь результат, обчислений із застосуванням цього паролю.

Розглянемо наступний алгоритм:

1. Клієнт хоче увійти до свого профілю. Він надсилає повідомлення до сервера, де вказує свій ідентифікатор, щоб сервер міг зрозуміти хто він є.
2. Сервер обирає певне випадкове число та надсилає його користувачу. Це число заведено називати викликом, або nonce (number used only once - число, що використовується один раз).
3. Користувач отримує це число та обчислює певну функцію, відому обом сторонам, вхідними аргументами якої є пароль користувача та число, отримане від сервера. Важливо щоб ця функція була односторонньою, тобто щоб було майже неможливо отримати її вхідні аргументи, знаючи лише вихідне значення. Це може бути, наприклад хеш-функція.
4. Далі користувач надсилає це число до сервера. Цей етап називають відповіддю на виклик сервера.
5. Сервер отримує відповідь клієнта та звіряє, що результат отриманий клієнтом відповідає результату отриманому із застосуванням цього nonce та паролю користувача, збереженого в базі даних сервера.

Таким чином, користувач зміг себе автентифікувати, не передаючи через інтернет свій власний пароль. З усім тим, серверу й досі потрібно надійно зберігати пароль користувача у себе в базі даних, та зловмисник досі може дізнатись пароль користувача перебравши усі можливі варіанти та застосувати до них обрану функцію та nonce.

Цей протокол піддається атаці, коли зловмисник може перехопити відповідь користувача, надіслати її від себе серверу, видавши себе за користувача, та успішно автентифікуватись. Для цього потрібно на етапі автентифікації узгодити із сервером як будуть шифруватись наступні повідомлення, щоб зловмисник не міг нічого надалі зрозуміти.

Також важливо, щоб nonce кожен раз був обраний випадковим чином, оскільки інакше зловмисник може наступного разу перехопити відповідь користувача та видати себе за нього, не знаючи справжнього пароля, а знаючи лише відповідь на виклик. В якості nonce можна обрати поточну часову мітку в точності до мікросекунд. Тоді ця мітка буде досить випадковою, і користувач буде впевненим що її не міг підробити зловмисник, підставивши якесь старе, вже використане значення nonce.

3.3 Криптографічні протоколи автентифікації

3.3.1 Симетричне шифрування

Одним із недоліків автентифікації із використанням пароля є те, що користувач повідомляє серверу свої секретні дані - пароль, який після реєстрації сервера треба надійно зберігати, щоб зловмисник не міг його отримати.

Криптографічний ключ - це секретна інформація, що зазвичай задана цілим числом, яку використовують у криптографічних алгоритмах шифрування та дешифрування, встановленню підпису, його перевірка, та автентифікації. Також існують публічні ключі, які можна розголошувати, але вони завжди йдуть в парі із секретним ключем, і вони чимось пов'язані один з одним.

Симетричне шифрування - це спосіб шифруваннях даних, в якому одночасно застосовується один і той самий ключ для шифрування та дешифрування даних. Цей ключ є секретним, дізнавшись його значення та алгоритм шифрування, зловмисник зможе повністю розшифрувати повідомлення якими обмінюються сервер та клієнт. Обидві сторони мають узгодити алгоритм, який надалі буде застосовуватись для шифрування, і який буде використовувати даний секретний ключ.

Саме по собі симетричне шифрування не є достатньо зручним, оскільки секретний ключ треба якимось чином надійно передати від однієї сторони до іншої, щоб узгодити його. Тому його зазвичай використовують з іншими протоколами шифрування. Проте, такі алгоритми є досить швидкими, бо вони не потребують складних обчислень та велику потужність комп'ютера.

Одним із найбільш відомих алгоритмів симетричного шифрування є AES (від англ. Advanced Encryption Standard). Цей алгоритм був навіть прийнятим урядом США. Він добре проаналізований і зараз широко використовується, як це було з його попередником DES.

Алгоритми симетричного шифрування можна використовувати для двосторонньої автентифікації. Якщо обидві сторони вже знають секретний ключ, тоді, щоб одна сторона автентифікувала другу, перша сторона має відправити виклик, зашифрувавши його секретним ключем. Сторона отримувач має розшифрувати цей виклик, дописати до нього певну інформацію, наприклад поточний час, або деяку відповідь на виклик, зашифрувати це все разом та відправити до першої сторони. Це підтвердить те, що друга сторона знає секретний ключ, бо інакше було б дуже складно розшифрувати повідомлення із викликом та зашифрувати нове повідомлення. Даний алгоритм використовується, наприклад у досить відомому протоколі Kerberos.

Для підвищення безпечності протоколів часто вводять таке поняття як сесія - це певний проміжок часу, що починається в момент автентифікації. Коли даний проміжок часу закінчується користувачу треба пройти процедуру автентифікації заново. При цьому для кожної сесії параметри для шифрування обираються заново. Це зроблено для того, щоб якщо зловмисник якимось чином зможе підібрати секретний ключ, то він мав доступ до даних, не володіючи секретними ключами користувача, лише на обмежений проміжок часу.

3.3.2 Асиметричне шифрування

На відміну від симетричного шифрування, в асиметричному використовується два види ключів - відкритий та закритий, які мають математичний зв'язок. Відкритий ключ застосовується для дешифрування даних, а закритий для шифрування. Відкритий ключ зазвичай знаходять обчислюючи певну функцію для закритого ключа. Отримати значення відкритого ключа проста задача, але визначити закритий ключ знаючи відкритий майже неможливо. Слово майже застосовується лише для того, щоб підкреслити, що ймовірність дізнатись закритий ключ не є нульовою, проте вона дуже до цього близька. Такий вид шифрування ще часто називають шифруванням із відкритим ключем.

Власник пари ключів може зашифрувати будь-які дані своїм закритим ключем, і будь-хто, знаючи публічний ключ, зможе їх розшифрувати. Цей факт використовується в алгоритмах цифрового підпису, щоб підтвердити справжність документів. І це працює краще за звичайні підписи ручкою, оскільки такий вид підписів значно важче підробити (майже неможливо).

В більшості алгоритмів асиметричного шифрування публічний ключ можна так само застосовувати для шифрування даних, але ці дані можна розшифрувати лише знаючи приватний ключ. Проте, у таких алгоритмах і досі працює той факт що секретний ключ неможливо отримати знаючи публічний ключ, оскільки публічний ключ може знати фактично будь-хто.

Так само шифрування з відкритим ключем можна використовувати в алгоритмах автентифікації, оскільки такий підпис підтверджує те, що користувач є власником закритого ключа. Це також обходить деякі відомі атаки, наприклад атаку прослуховувача (man in the middle attack), бо для цього зловмиснику недостатньо перехопити якісь дані, йому доведеться дізнатись закритий ключ.

Процес автентифікації із використанням асиметричного шифрування виглядає наступним чином:

1. Клієнт надсилає привітальне повідомлення до сервера, де вказує свій публічний ключ.
2. Сервер генерує випробування (nonce), додає до нього часову мітку, зашифровує повідомлення використовуючи публічний ключ користувача, та надсилає це повідомлення користувачу.
3. Користувач розшифровує повідомлення використовуючи свій секретний ключ. Зловмисник не зможе цього зробити не знаючи цей ключ.
4. Далі користувач генерує відповідь до виклику, зашифровує це своїм секретним ключем та відправляє до сервера. Це шифрування підтвердить те, що відповідь надає точно саме цей користувач.
5. Сервер розшифровує повідомлення використовуючи публічний ключ користувача, та звіряє відповідь на випробування.

6. Якщо клієнт успішно автентифікований, сервер генерує секретний ключ для симетричного шифрування, шифрує його за допомогою публічного ключа користувача та надсилає йому.
7. Далі все спілкування сервера та клієнта шифрується за допомогою симетричного шифрування, згенерованого у рамках однієї сесії.

Однією із переваг автентифікації із відкритим ключем є те, що клієнту не обов'язково генерувати нову пару ключів для спілкування із різними серверами, оскільки жоден із серверів не зможе отримати будь-який доступ до інших, знаючи лише відкритий ключ. Серед інших переваг автентифікації із застосування відкритого ключа є те, що фактично користувачу не треба проходити етап реєстрації та передавати будь-які секретні дані в інтернеті, або офлайн, тому що публічний ключ є майже унікальним, коли ключі мають достатньо велику довжину. Ймовірність того, що два різні користувачі зможуть отримати однаковий публічний ключ для своїх закритих ключів є майже нульовою.

Алгоритми асиметричного шифрування засновані на математичній складності обчислення вхідних даних для односторонніх функцій. В основі алгоритму лежить певна математична функція, для якої неможливо за поліноміальний час підібрати можливі аргументи, щоб отримати конкретний результат.

Серед недоліків асиметричного шифрування виділяють довгий час роботи та достатню потужність комп'ютерів, оскільки такі алгоритми використовують складні криптографічні функції. Такі обчислення займають відносно багато часу та електроенергії, тому зазвичай даний вид шифрування застосовують лише для автентифікації, після чого створюється сесія та спілкування переходить до симетричного шифрування.

3.3.3 Алгоритм RSA

Концепція асиметричного шифрування із відкритим та закритим ключем була запропонована Уітфілдом Діффі та Мартіном Геллманом у 1976 році. Проте сам алгоритм ними не був дороблений, вони залишили відкритою проблему реалізації односторонньої функції, можливо через те, що факторизація в ті часи була недостатньо вивчена.

У 1977 році Рональд Райвест, Аді Шамір та Леонард Адлеман із Массачусетського технологічного інституту (MIT) опублікували повний алгоритм асиметричного шифрування, який в результаті було названо на їх честь - RSA.

Алгоритм RSA використовує пару ключів - публічний (відкритий) та приватний (закритий). Безпека алгоритму заснована на математичній складності факторизації довгих складених чисел за модулем.

Генерація ключів відбувається за наступним алгоритмом:

1. Вибираються два великі прості числа p та q. За довжину можна обрати 512 біт, це буде достатньо надійно.
2. Обчислюється їх добуток $n = pq$.
3. Обчислюється функція Ейлера: $\phi(n) = (p-1)(q-1)$.
4. Обирається ціле число e таке, що $1 < e < \phi(n)$ та e має бути взаємно простим із $\phi(n)$.
5. За допомогою розширеного алгоритму Евкліда обчислюється число d таке, що $ed \equiv 1 \ (mod\ \phi(n))$.

Число n називається модулем, а числа e, і d, - відкритою й секретною експонентами відповідно. Пара чисел $(n,\ e)$ є відкритою частиною ключа, а $(n,\ d)$ - секретною. Числа p, і q після генерації пари ключів можуть бути знищені, але в жодному разі не повинні бути розкриті.

Зашифрувати ціле число можна за допомогою формули:

$$c = m^e \ (mod\ n)$$

де m - це число що потрібно зашифрувати.

Щоб розшифрувати повідомлення c застосовують формулу із протилежним ключем:

$$m = c^{d}(mod\, n)$$

Даний алгоритм є дуже надійним, тому його часто застосовують у протоколах автентифікації із відкритим ключем, а також для асиметричного шифрування, проте він є досить повільним, тому після автентифікації, для шифрування повідомлень використовують симетричні алгоритми шифрування. Під час автентифікації, застосовуючи асиметричне шифрування, обом сторонам потрібно домовитись про секретний ключ, який буде надалі використовуватись в алгоритмі секретного шифрування. Час від часу треба цей ключ змінювати, це може бути реалізовано у вигляді сесій.

3.4 Протокол Нідгема-Шредера

3.4.1 Особливість протоколу

Даний протокол був запропонований Роджером Нідгемом та Майклом Шредером. Він узагальнює два різні комунікаційні протоколи, призначені для незахищених мереж:

- Протокол із симетричним ключем, заснований на алгоритмі симетричного шифрування. Даний протокол використовується для встановлення ключа сеансу, наприклад в протоколі Kerberos.
- Протокол із відкритим ключем, заснований на криптографічних алгоритмах із публічними ключами. Даний протокол використовують для взаємної автентифікації.

У початковій версії алгоритму пізніше було встановлено уразку до атаки людини посередині (man in the middle attack), розглянуту у статті Гевіна Лоу. Ним же даний протокол був трохи удосконалений, щоб унеможливити дану атаку.

3.4.2 Опис протоколу

У даному протоколі до процесу автентифікації долучають третю довірену особу (3-rd party), наприклад центр автентифікації (AUC), який зберігає публічний ключ кожного з користувачів.

Далі будемо використовувати наступні позначення:

- A - перший користувач
- B - другий користувач
- C - центр автентифікації (AUC)
- K_{PA}, K_{SA} - публічний та приватний ключ A, відповідно
- K_{PB}, K_{SB} - публічний та приватний ключ B, відповідно
- K_{PC}, K_{SC} - публічний та приватний ключ C, відповідно

Протокол виконується за наступним алгоритмом:

1. A відправляє C запит на отримання відкритого ключа B - K_{PB}

2. C відправляє A відкритий ключ B, та підписує його власним приватним ключем.
3. A генерує випадковий виклик NA та відправляє його B, шифруючи даний запит публічним ключем B. Також до цього запиту додається відправник A.
4. B отримує виклик від A. Тепер B відправляє C запит на отримання публічного ключа A.
5. C відправляє публічний ключ A до B, підписуючи цю відповідь своїм приватним ключем.
6. B генерує свій виклик NB, та разом із NA, відправляє його до A, шифруючи це публічним ключем A.
7. A розшифровує виклик B своїм приватним ключем, та відправляє цей виклик назад до B, шифруючи це його публічним ключем, проте в цей раз без виклику NA. Цей етап підтверджує те, що A може розшифрувати останнє повідомлення від B та виділити в ньому виклик NB.

Після цього обидві сторони можуть бути впевненими один в одному, та при цьому обидва виклики NA та NB відомі тільки їм. Таким чином також можна згенерувати ключ сесії.

3.4.3 Безпечність протоколу

Нажаль, версія протоколу розглянута вище вразлива до атаки людини посередині (man in the middle attack), де зловмисник може розпочати сесію з A, та потім, видаючи себе за A, встановити сесію із B. Розглянемо поетапно як це відбувається:

1. Спочатку A хоче почати сесію зі зловмисником D та відправляє йому виклик NA із відправником A, підписаний публічним ключем D (третій етап).

2. D розшифровує виклик, та перенаправляє його B, видаючи себе за A, та підписуючи публічним ключем B (підроблений третій етап).
3. B отримує виклик ніби-то від A, після чого він генерує свій виклик NB, підписує його публічним ключем A, та відправляє до зловмисника D, оскільки запит прийшов з його адреси (шостий етап).
4. Зловмисник перенаправляє даний виклик до A (підроблений шостий етап).
5. A розшифровує виклик NB, та відправляє його назад до зловмисника D, думаючи що A успішно авторизувався із ним (сьомий етап).
6. D розшифровує відповідь A, та отримує виклик B, після чого він відправляє його B, тим самим авторизовуючи себе ніби-то за A.

Після цього A думає, що він авторизувався із D, а B думає, що він авторизувався з A. Хоча насправді D одночасно авторизував себе для A, та авторизувався із B, видаючи себе за A.

Вперше дана атака на даний алгоритм була описана в роботі Гевіна Лоу у 1995 році. Тому Лоу запропонував свою модифікацію протоколу, яка унеможливлює дану атаку. Нову версію протоколу також називають протоколом Нідгема-Шредера-Лоу.

Модифікація Лоу полягає в тому, щоб на шостому кроці B до повідомлення, перед шифруванням додавав окрім обох викликів також і себе (тобто B), як відправника.

3.4.4 Ефективність

Даний протокол використовує достатньо мало памьяті, оскільки обом сторонам потрібно зберігати лише свою пару ключів, публічний ключ другої сторони та центру автентифікації, а також пару викликів. Кожен ключ має розмір близько з 512-1024 біт, отже сумарно вийде не більше одного KiB. Так само даний протокол використовує достатньо мало трафіку, оскільки він не інтерактивний та дану процедуру достатньо провести один раз за сесію із кожним користувачем, із яким треба авторизуватись.

З точки зору ефективності по часу, даний протокол не є достатньо оптимальним, оскільки для підпису зазвичай застосовують алгоритм RSA, який потребує багато операцій множення, які використовуються в алгоритмі асиметричного шифрування, при шифруванні та дешифруванні. Таким чином на одну процедуру шифрування та дешифрування, піде близько $2 \cdot 512$ операцій множення (де 512 це довжина ключа), якщо застосовувати алгоритм бінарного піднесення до степеня. Кожній зі сторін доведеться виконати шифрування та дешифрування 4 рази, отже сумарно виходить близько 4000 операції множення. Оскільки ключі складаються з багатьох біт, треба застосовувати бібліотеки для роботи із великими числами, щоб виконувати операцій множення, що також сповільнює весь процес.

Розділ 4. Протоколи із нульовим розголошенням (Zero knowledge proof)

4.1 Особливість протоколів із нульовим розголошенням

Протоколи з нульовим розголошенням (від англ. Zero knowledge proof) - це концепція протоколів, у яких один учасник, наприклад клієнт, може довести другому учаснику (серверу), що деякий факт є вірним, при цьому, не розголошуючи другому клієнту жодної інформації, окрім вірності даного факту.

Такі протоколи мають володіти трьома властивостями:

1. Повнота - якщо факт є вірним, тоді доказуючий впевнить в цьому перевіряючого, із наперед заданою точністю.
2. Коректність - якщо факт не є вірним, тоді, навіть зловмисник, не зможе впевнити перевіряючого що факт є вірним, окрім дуже малої ймовірності, якою можна знехтувати.
3. Нульове розголошення - незалежно від вірності факту та чесності перевіряючого, він не зможе дізнатись нічого окрім лише вірності заданого факту.

Даний протокол є ймовірнісним - це значить, що існує дуже мала, несуттєва вірогідність того, що зловмисник зможе обманути перевіряючого, та довести йому, що певний, насправді хибний, факт є вірним.

Є різні варіації таких протоколів - інтерактивні та неінтерактивні. У інтерактивних протоколах перевіряючий поступово надсилає виклик до доказуючого, який має на кожен виклик надати відповідь - чи є даний факт вірним. Такий процес повторюється певну кількість разів, поки перевіряючий не буде впевненим, що доказуючий знає доказ факту. У неінтерактивних варіаціях протоколу, доказуючий генерує один доказ, що не потребує викликів від перевіряючого.

Щоб краще пояснити ідею даного протоколу, у своїй роботі Жан-Жак Кіскатер приводить аналогію зі східною казкою про Алі-Бабу та сорок розбійників. Спочатку в казці розповідають що колись давно, в місті Багдад жив старець Алі-Баба, якого часто грабували коли він ходив по базару. Кожного разу старець переслідував злодія, проте злодій щоразу забігав у печеру, яка ділиться на два тунелі, кожен з яких закінчується тупиком. Алі-Баба не встигає помітити в який із тунелів біжить злодій, тому він обирає випадковий, йде до кінця, але кожного разу нікого там не знаходить. Першого разу Алі-Баба подумав що злодій обрав інший тунель та встиг повернутись до виходу, поки Алі-Баба йшов по другому тунелю. Проте коли це сталось сорок разів, із сорока злодіями, то Алі-Баба здогадався, що із тунелем щось не так. Наступного разу він сховався в кінці одного з тунелів, дочекався коли туди дійде злодій, та почув як злодій каже “Сезам, відкрийся”, після чого тупик тунелю пропав та на його місці відкрився прохід далі. Повторивши цю фразу, Алі-Баба з'ясував, що прохід поєднує тупики обох тунелів, тому злодій завжди міг втекти, який би прохід не обрав на початку. Експериментуючи із магічними словами, старець зміг замінити кодову фразу на нову, секретну, тому наступного ж дня злодій був пійманим. Алі-Баба записав цю історію у своїх рукописах, не розголошуючи саму кодову фразу, а залишивши підказки.

Через багато років вчені знайшли рукопис Алі-Баби, а також змогли розгадати секретну фразу та віднайти той самий тунель. Тунель не був вигадкою, а справді існував, і секретна фраза дійсно працювала. Про це одразу ж повідомили в новинах. Один із дослідників Мік Алі, що можливо є потомком Алі-Баби, вирішив продемонструвати, що він знає секретну фразу, не розголошуючи її нікому. Для цього він запросив знімальну групу, яка спочатку добре дослідила печеру, а потім всі з неї вийшли. Після чого, до печери зайшли тільки репортер та Мік, і зупинились біля розвилки. Репортер підкинув монетку і визначив в який із тунелів піде Мік, щоб з протилежного тунелю він вийшов. Далі Мік пішов до відповідного тунелю та вийшов із другого. Проте, щоб зменшити вірогідність того, що Міку якось пощастило, даний експеримент повторили сорок разів, і

кожен раз йому вдавалось пройти тунель та вийти з іншого кінця. Таким чином Мік зміг доказати те що він знає секретний код, не розголосивши його.

4.2 Застосування ZKP протоколів для автентифікації

Протоколи автентифікації із застосуванням пароля слабкі до ряду відомих атак, проте основним їх недоліком є те, що друга сторона, тобто сервер, має доступ до вашого пароля та зберігає його у себе. По-перше, зловмисник може отримати доступ до сервера, дістати пароль користувача, та застосовувати у своїх цілях. По-друге, можливо навіть сам сервер є зловмисником, і може використовувати пароль, який користувач йому довірив, у власних цілях, наприклад для автентифікації під ім'ям клієнта в інших сервісах.

Протоколи автентифікації із застосуванням публічного ключа є більш надійними, оскільки друга сторона не має жодного доступу до пароля клієнта - секретного ключа. Проте, такі протоколи можуть бути уразливими до так званих chosen-plaintext атак, коли зловмиснику відомий набір певних текстів, відповідних їм зашифрованих текстів, та функції шифрування. Фактично, зловмиснику можуть бути відомі всі дані задіяні в шифруванні, окрім ключів шифрування, які він може підібрати аналітичним методом. Таким чином, коли користувач шифрує певні дані за допомогою свого приватного ключа, то він розкриває певну інформацію про нього.

Протоколи автентифікації із нульовим розголошенням, на відміну від інших протоколів, не мають фактора зношування ключів із ростом кількості його використання, оскільки при їх застосуванні друга сторона не отримує жодної іншої інформації, окрім доказу того, що клієнт володіє секретним ключем, тому такі протоколи не уразливі до chosen-plaintext атак. Більшість таких протоколів менш ефективні та потребують багато ресурсів, проте деякі з них можуть бути кращими за інші протоколи із відкритим ключем.

4.3 Протокол Фіата-Шаміра

4.3.1 Особливість протоколу

Один із найбільш відомих протоколів із нульовим розголошенням був спочатку запропонований Амосом Фіатом та Аді Шаміром, та названий на їх честь. Пізніше цей протокол був удосконалений із залученням до роботи Уріеля Фейге.

Криптографічна стійкість даного протоколу заснована на математичній складності знаходження квадратного кореня за модулем достатньо великого складеного числа, факторизація якого невідома та складна для знаходження.

У даному протоколі застосовують третю особу - довірений центр, який зберігає усім відомий модуль та публічні ключі усіх користувачів, при цьому факторизацію модуля центр тримає в повному секреті.

4.3.2 Опис протоколу

Спочатку користувачу треба згенерувати свою пару ключів з публічного та приватного. Це відбувається за наступним алгоритмом:

1. Перш за все, певна довірена третя особа (наприклад Central authority) обирає два достатньо великі прості числа p та q, із великою різницею між ними, та обчислює параметр $n = pq$. Цей параметр вважають публічним модулем, який повідомляють усім учасникам, а числа p та q надійно зберігаються в секреті. Для надійності бажано, щоб число n складалось щонайменше із 512, або 1024 бітів.
2. Кожен користувач обирає випадкове ціле число s, взаємно-просте із n, де $s \in [1,\ n - 1]$, а потім користувач обчислює число $v = s^2 (mod\ n)$.
3. Після цього число v вважається публічним ключем користувача та може бути повідомлено будь-кому, його використовують в якості ідентифікатора, а число s є приватним ключем користувача, яке треба тримати в максимальному секреті.

Далі розглянемо як відбувається процес автентифікації, якщо A є доказуючою стороною, а B перевіряючою:

1. Спочатку A генерує випадкове число r, де $r \in [1, n-1]$, та відправляє його стороні B число $x = r^2 (mod\, n)$.
2. B обирає випадковий біт $e \in \{0, 1\}$ - виклик, та надсилає його A.
3. A обчислює відповідь на виклик $y = rs^e$, та надсилає його B.
4. B перевіряє, що $y^2 = xv^e$. Якщо це вірно, то B приймає даний доказ.

Щоб зменшити шанси на удачу дану процедуру повторюють t разів, кожен з яких підвищує шанси того, що A дійсно є тим за кого себе видає.

4.3.3 Безпечність протоколу

Вибір e із множини $\{0, 1\}$ використовується для того, щоб якщо сторона A дійсно знає секретний ключ, то вона точно зможе відповісти на даний виклик незалежно від значення e. Припустимо, що A це зловмисник, який не знає секретного ключа справжнього користувача та хоче обійти протокол. Тоді, якщо A припускає, що він отримає $e = 0$, то A завжди знає відповідь на виклик, оскільки це не потребує секретного ключа. Якщо A знає, що точно отримає $e = 1$, то A може обрати випадкове число r та надіслати на сервер $x = \frac{r^2}{v}$. Проблемою для зловмисника є те, що він не знає точно яке зі значень A він отримає з ймовірністю 100%, тому він зможе обманути перевірку лише із ймовірністю 50%. Якщо дану процедуру виконати t разів, то ймовірність того, що зловмисник зможе обманути перевірку всі t разів рівна 2^{-t}, таким чином B може бути впевненим в A, лише якщо всі t раундів були пройдені успішно. t зазвичай обирають відносно великим, близько 20-40.

Оскільки всі ітерації алгоритму не залежать один від одного, то даний протокол можна виконувати паралельно. Паралельна версія протоколу буде так само безпечною, проте, через певні технічні причини, властивість нульового розголошення буде втрачена.

4.3.4 Протокол Фейга-Фіата-Шаміра

Протокол Фіата-Шаміра був удосконалений у 1986 році, із залученням до роботи Уріеля Фейга. Даний протокол дуже схожий до попередньої версії, але він дозволяє зменшити кількість ітерацій, не змінюючи надійність протоколу.

На етапі підготовки модуль n обирається таким же чином як і раніше, проте тепер кожен учасник обирає не один секретний ключ s, а k різних цілих простих чисел $s_1, s_2, \ldots, s_k$, де $s_i \in [1, n-1]$, а також k випадкових бітів $b_1, b_2, \ldots, b_k$, і далі обчислює множину $v_1, v_2, \ldots, v_k$, де $v_i = (-1)^{b_i}(s_i^2)^{-1}(mod\ n)$. Значення $(v_1, v_2, \ldots, v_k)$ є публічним ключем учасника, і може бути повідомлено будь-кому, а значення $(s_1, s_2, \ldots, s_k)$ є секретним ключем і нікому більше не розголошується.

Етап автентифікації відбувається за наступним алгоритмом:

1. Спочатку A генерує випадкове число r, де $r \in [1, n-1]$, та відправляє стороні B число $x = r^2(mod\ n)$.
2. B обирає вектор з k випадкових бітів $\{e_1, e_2, \ldots, e_k\}$, де $e_i \in \{0, 1\}$, та надсилає його A.
3. A обчислює значення $y = r\prod_{i=1}^{k}(s_i^{e_i})\ (mod\ n)$ та відправляє B.
4. B обчислює $z = x\prod_{i=1}^{k}(v_i^{e_i})\ (mod\ n)$, та порівнює z із $\pm\ y$.

Таким чином, вірогідність того, що зловмисник зможе обійти одну ітерацію протоколу рівна 2^{-k}, а t ітерацій 2^{-kt}, що дозволяє зменшити кількість ітерацій до $t = 5$, при $k = 6$.

Також даний протокол можна використовувати для цифрового підпису.

4.3.5 Ефективність

У протоколі Фіата-Шаміра користувачу потрібно зберігати всього кілька чисел в постійній пам'яті: модуль n, секретний ключ s та публічний ключ v. На кожній ітерації користувачу також потрібно зберігати лише кілька чисел. Перевіряючому для всього процесу автентифікації потрібно ще менше пам'яті. Таким чином, якщо вважати що усі числа будуть мати 512 біт, то обом учасникам буде достатньо близько $5 \cdot 512$ біт, тобто менше одного КіВ. Також буде використано щонайменше $2 \cdot 512$ біт трафіку для обміну повідомленнями за одну ітерацію, та 512 біт для повідомлення публічного ключа стороні перевіряючого.

У протоколі Фейге-Фіата-Шаміра використовується більше пам'яті, оскільки користувачу тепер потрібно зберігати не по одному числу на кожен ключ, а одразу ж по k чисел. Тоді, для модуля розміром 512 біт, користувачу буде достатньо $(2k + 3) \cdot 512$ біт, а перевіряючому $(k + 3) \cdot 512$ біт, оскільки він не зберігає секретний ключ. При $k = 6$, це й досі буде менше одного КіВ для обох сторін. Для обміну повідомленнями буде використано близько $(k + 1) \cdot 512$ біт даних за ітерацію.

Найдорожчою операцією в обрахунках є множення двох чисел за модулем певного числа, оскільки числа досить довгі та не помістяться в один регістр процесора, а додавання є достатньо простою та швидкою операцією. Протокол Фіата-Шаміра потребує близько двох операцій множення за модулем для доказуючого та однієї такої операції для перевіряючого, а протокол Фейге-Фіата-Шаміра близько $k + 1$ операцій для доказуючого, та k операцій для перевіряючого. Враховуючи те, що k зазвичай достатньо обирати близько 5-10, то це дуже мала кількість.

4.4 Протокол Шнорра

4.4.1 Особливість протоколу

Протокол Шнорра є одним із найбільш ефективних та теоретично обгрунтованих схем автентифікації. Даний протокол є модифікацією протоколів Ель-Гамаля та Фіата-Шаміра. Криптографічна стійкість даного алгоритму заснована на математичній складності знаходження дискретного логарифма по модулю великого простого числа.

Більшість обчислень в даному протоколі можуть бути виконані під час попередніх обчислень, поки процесор не навантажений. Це дозволяє сильно зменшити час обрахунків необхідних для автентифікації.

Даний протокол також можна використовувати для цифрового підпису.

4.4.2 Опис протоколу

Перш за все користувачу треба згенерувати свою пару ключів. Це робиться так само як в алгоритмі цифрового підпису (DSA):

1. Спочатку обирається велике просте число q, що складається щонайменше зі 160 бітів.
2. Потім обирається просте число p, щонайменше із 1024 бітів, таке, що $p = kq + 1$, тобто, щоб q був дільником значення $p - 1$.
3. Далі треба визначити ціле число g, де $g \in [2, p - 1]$, таке, що $g^q \equiv 1 (mod\ p)$.
4. Користувач обирає випадкове ціле число w, менше за q.
5. Далі треба обчислити значення $y = g^{(q-w)} (mod\ p)$.

Таким чином користувач отримує свій публічний ключ (p, q, g, y) та приватний ключ w. Перші 3 етапи можуть бути винесені на обробку довіреній особі - CA.

Далі розглянемо сам процес автентифікації:

1. Користувач обирає випадкове число r, де $r \in [1,\ q\ -\ 1]$, та обчислює $x = g^r (mod\ p)$.
2. Користувач відправляє значення x серверу.
3. Сервер обирає випадкове число e із діапазону $[0,\ 2^t - 1]$, та відправляє його клієнту.
4. Клієнт обчислює $s = r + we\ (mod\ q)$, та відправляє це значення до сервера.
5. Сервер перевіряє, що $x = g^s y^e (mod\ p)$.

Пункт 1 може бути виконано завчасно, до початку автентифікації, оскільки він не потребує жодної комунікації із сервером, і ніяк не прив'язаний до часу. Тому цей етап можна виконувати наперед, можливо, коли процесор ніяк не завантажений.

4.4.3 Безпечність протоколу

Безпечність протоколу сильно залежить від вибору параметра t. Шнорр рекомендує використовувати значення t щонайменше 72, тоді кожному відомому на цей час алгоритму знаходження дискретного алгоритму, знадобиться близько 2^{72} кількості кроків, що є дуже великим числом, враховуючи потужність сучасних комп'ютерів.

За припущенням про складність знаходження дискретного логарифма, даний протокол є стійким до атак пасивного противника. Також, за припущенням про те, що даний протокол є протоколом із нульовим розголошенням, він стійкий до атак активного противника, коли зловмисник може видавати себе за автентифікатора та збирати дані, виконуючи процес автентифікації із клієнтом, а потім використати ці дані для знаходження секретного ключа клієнта. Проте, на поточний час, ніхто не зміг доказати те, що даний протокол гарантовано відповідає протоколу із нульовим розголошенням.

4.4.4 Ефективність

Даний алгоритм, так само як і попередній, потребує достатньо малий об'єм памьяті, оскільки користувачу та серверу потрібно зберігати всього лише кілька чисел, хоч і різного розміру, але сумарно виходить менше за один КіВ. Так само даний протокол використовує дуже малий об'єм трафіку, оскільки користувачу та серверу треба обмінятись всього парою чисел.

Під час автентифікації користувачу треба виконати всього одну операцію множення, оскільки найважчі операції були виконані перед автентифікацією. Проте, серверу під час автентифікації доведеться виконати більше обчислень, оскільки йому треба піднести два числа до великого степеня. Якщо використовувати бінарний алгоритм піднесення до степеня, то на етапі автентифікації, серверу доведеться виконати близько $2 \cdot (160 + 72) = 464$ операцій множення.

Найбільшу кількість обчислень клієнту доведеться виконати до початку автентифікації. Для піднесення один раз числа до степеня, завдяки алгоритму бінарного піднесення до степеня, можна обійтись всього $2 \cdot 160 = 320$ операціями множення. Проте, ці обчислення можна виконати задовго до початку автентифікації, у проміжки часу коли процесор є мінімально завантаженим.

Розділ 5. Реалізація протоколу Фіата-Шаміра

5.1 Інструменти реалізації

Для реалізації протоколу було обрано мову програмування C++, оскільки програми на даній мові, у порівнянні з іншими, є досить швидкими та мають відносно малий розмір. Ці критерії є дуже важливими для приладів інтернету речей, оскільки ресурси, у тому числі й пам'ять, там є обмеженими.

Для компіляції програми використовувався компілятор Clang версії 13.1.6. Для зборки програми використовувалась кросплатформенна система автоматизації зборки CMake, яка генерує make-файли для зборки програми.

Для реалізації даного протоколу також було розроблено самописну бібліотеку для роботи із великими числами, а також окремий модуль для певних криптографічних обчислень.

Для замірів часу роботи програми було використано open-source бібліотеку EasyProfiler, що дозволяє замірювати середній час роботи окремих функцій та блоків програми.

Компіляцію, зборку та тестування протоколу було виконано на комп'ютері MacBook Air (M1, 2020) з операційною системою macOS Monterey Version 12.3. Даний комп'ютер має 16 GiB оперативної памьяті та процесор Apple M1.

5.2 Опис програмної реалізації

Розроблена програма симулює процес автентифікації користувача.

Спочатку створюється окрема сутність CentralAuthority, яка відповідає третій, довіреній особі, яка генерує публічний модуль n, а також зберігає публічні ключі усіх користувачів за їх ідентифікатором.

Клас CentralAuthority при створенні генерує два великі прості числа p та q, певного розміру, наприклад 256 біт, та обчислює публічний модуль $n = pq$. Після чого, використовуючи публічні методи даного класу, користувачі можуть дізнатись даний модуль, визначити свої ключі, та додати себе до бази даних CA, використовуючи публічний метод ***registerUser***, який приймає ідентифікатор користувача та його публічний ключ.

Клас User створений для емуляції роботи користувача. При створенні користувач, використовуючи модуль отриманий від CentralAuthority, генерує свою пару ключів. Спочатку приватний ключ s, а потім публічний ключ $v = s^2 (mod\ n)$. Найважчою процедурою на даному етапі є згенерувати випадкове просте число в проміжку $[1, n - 1]$, оскільки для цього треба генерувати випадкові числа, та перевіряти їх на простоту, а це досить важка та відносно довга операція, проте це достатньо зробити лише один раз.

У класу User є два публічні методи, які використовуються для авторизації користувача:

- Метод ***initAuthentication*** призначений для того, щоб згенерувати випадкове число $r \in [1, n - 1]$, зберегти його у себе, та повернути число $x = r^2 (mod\ n)$ користувачу, який бажає авторизувати даного користувача. У справжній програмі, автентифікацію зазвичай бажає здійснити сам користувач, тому він сам генерує це число та відправляє серверу значення x, проте, оскільки дана програма є симулятором, для зручності було обрано додати даний метод, щоб ініціатива була за сервером.

- Метод ***processChallenge*** приймає одне ціле число (можливо логічну змінну) e - виклик. Даний метод має викликатись сервером для того, щоб користувач підтверджував знання секретного ключа. При отриманні даного виклику, користувач спочатку перевіряє чи зберігав він значення r. Якщо так, то далі він обчислює відповідь на виклик $y = rs^e (mod\ n)$, інакше повертає помилку. Після цього, користувач очищує значення r, щоб наступного разу згенерувати нове число r та x . Далі користувач повертає серверу значення y.

Симуляція процесу автентифікації відбувається у файлі ***main.cpp.*** Спочатку ініціалізується бібліотека для криптографічних обчислень, щоб створити генератор випадкових чисел. Далі створюється клас CentralAuthority, який генерує публічний модуль n. Після цього можна створити користувачів - сутності класу User, та додати їх до бази даних CA. І далі можна викликати функцію ***VerifyUser***, яка приймає посилання на CentralAuthority, сутність користувача якого треба авторизувати, та його ідентифікатор.

У функції ***VerifyUser*** сервер спочатку дізнається публічний ключ користувача. Далі, протягом t раундів йде перевірка типу виклик-відповідь (challenge-responce), де сервер кожен раз генерує випадковий біт $e \in \{0, 1\}$, дізнається x від користувача, завдяки методу ***initAuthentication***, та відповідь y, завдяки методу ***processChallenge.*** Після чого сервер перевіряє, що $y^2 \equiv xv^e\ (mod\ n)$.

5.3 Приклад роботи програми

При створенні сутності CentralAuthority, генерується модуль N, розміром 64 біти:

1. Module N = 11003118673559796013

Далі створюється користувач, з ідентифікатором Alice, та публічним ключем '68748072161083131 38', приватний ключ при цьому не розголошується.

2. New user 'Alice' with public key '6874807216108313138' was registered.

Далі сервер починає автентифікацію користувача, виконуючи серію з $t = 30$ раундів. Якщо користувач не проходить хоча б один раунд, то програма повертає помилку, інакше каже про успіх. На даному етапі значення t можна змінити на інше, проте значення $t = 30$ є достатньо оптимальним.

3. User 'Alice' passed t=30 tests and has successfully authorized.

5.4 Аналіз результатів

Для замірів часу роботи алгоритму було використано open-source бібліотеку easy-profiler. Дана бібліотека дозволяє заміряти середній час роботи окремих функції та блоків програми.

Нижче наведено табличку із часом роботи певних блоків програми, в залежності від кількості біт у параметрах p та q:

Блок програми	32 біти	64 біти	128 бітів	256 бітів
Один раунд автентифікації	0.487 ms	1.312 ms	4.018 ms	12.683 ms
initAuthentication	0.192 ms	0.501 ms	1.489 ms	4.669 ms
processChallenge	0.076 ms	0.204 ms	0.643 ms	2.002 ms

За результатами видно, що час роботи дуже сильно залежить від довжини модуля $n = pq$, але час роботи все одно є дуже задовільним для інтернету речей, оскільки це дуже швидко. На слабких комп'ютерах цей час буде в кілька разів більшим, але це досі має бути менше секунди, що є дуже ефективно.

Використану пам'ять вже було досліджено при аналізі алгоритму.

Висновки

Сфера інтернету речей стає популярнішою із кожним роком, і кількість приладів, підключених до цієї мережі вже складає десятки мільярдів. Так само швидко росте й область застосування даної сфери, оскільки такі прилади можуть бути впроваджені будь-де: у розумних будинках, фермах, магазинах, машинах, офісах і так далі.

У даній сфері навіть виділяються цілі екосистеми, наприклад інтернет енергії - мережа для споживачів та виробників електроенергії, інтернет машин - мережа, що пов'язує транспортні засоби, а також екосистема розумного дому, яка пов'язує різні камери відеоспостереження, датчики (води, газу, тиску), ліхтарі та інше.

Більшість таких приладів обмінюються в інтернеті особистою інформацією користувача, наприклад прилади розумного дому можуть передавати різну інформацію про будинок, медичні прилади можуть передавати особисті дані про стан здоров'я користувача, а деякі з приладів можуть застосовуватись для інтернет-банкінгу та передавати дані про кредитну картку користувача та інше. Тому із розвитком даної тенології так само росте і кількість атак зловмисників, які хочуть отримати доступ до особистих даних інших людей. Через це треба забезпечувати високу надійність того, що доступ до даних зможе отримати лише власник цих даних.

Процес автентифікації полягає в тому, щоб ідентифікувати користувача та надати йому доступ до певного сервісу, або до його особистих даних. Завдяки автентифікації певний сервер, або навіть інший прилад може автентифікувати користувача та впевнитись в тому що він спілкується із тим із ким потрібно, після чого всі повідомлення можна шифрувати, щоб забезпечити їх безпечний обмін в інтернеті. Тому даний процес має велике значення у безпеці даних користувача та є дуже важливим.

Проте, більшість приладів інтернету речей з певних причин не можуть підтримувати основні протоколи автентифікації, які застосовуються на звичайних комп'ютерах та серверах. Це може бути через особливості мережі у якій здійснюється обмін повідомленнями, або через особливості заліза приладів інтернету речей, бо зазвичай такі прилади мають дуже обмежені ресурси, пам'ять, енергію та потужність процесора, тим часом як більшість протоколів автентифікації використовують складні криптографічні обчислення.

Для приладів інтернету речей треба розробляти окремі, обмежені, або удосконалені протоколи автентифікації, які будуть враховувати особливості кожного різного приладу. Такі протоколи мають використовувати відносно прості обчислення, які можуть бути швидко виконані на процесорі, та не потребують багато ресурсів приладу.

Разом з тим, такі протоколи мають бути дуже надійними та не уступати протоколам які використовуються на звичайних комп'ютерах, щоб забезпечити достатню надійність, оскільки зловмисник, маючи потужний комп'ютер, може обійти слабкі протоколи.

З іншого боку, такі прилади мають і корисні особливості. Процес автентифікації на більшості таких приладів є автоматизованим, тому вони можуть використовувати протоколи, що не потребують втручання людини, хіба що на етапі реєстрації, або налаштування приладу. Також такі прилади можуть мати окремі мікросхеми, призначені для забезпечення більшої надійності обміну даними в інтернеті. Наприклад, деякі з таких мікросхем захищають користувача від, так званих, фізичних атак, коли зловмисник може заволодіти приладом та переглянути дані які він зберігає. Такі мікросхеми можуть виконувати криптографічні обчислення, наприклад шифрування та дешифрування, зберігаючи всі дані в безпечному сховищі памьяті, куди ніхто не може отримати доступ.

У даній роботі було розглянуто приклади достатньо ефективних протоколів автентифікації, які можуть використовуватись на слабких пристроях, проте забезпечувати високу надійність. Ці протоколи потребують дуже мало пам'яті та використовують дуже мало відносно складних обчислень, тому вони можуть бути виконані навіть на слабких процесорах, а також вони не будуть відбирати багато електроенергії та трафіку, завдяки чому можуть бути використані також і на приладах з обмеженою електроенергією.

Безпечність даних протоколів пов'язана зі складністю обчислень певних математичних проблем, таких як складність факторизації великих чисел за модулем та пошук дискретного логарифма великого складеного числа. Ймовірність того, що зловмисник зможе обійти даний протокол настільки мала, що нею можна знехтувати, тому дані протоколи є дуже надійними.

Протокол Фейге-Фіата-Шаміра потребує набагато менше ресурсів ніж інші протоколи, що робить його ще більш ефективним за інші протоколи. Завдяки тому, що даний протокол забезпечує дуже високий рівень надійності, його можна так само використовувати для звичайних комп'ютерів, що не мають особливих обмежень на ресурси.

Висновком даної роботи є факт того, що протокол Фейге-Фіата-Шаміра є оптимальним для забезпечення надійного процесу автентифікації для приладів інтернету речей. Він швидкий, використовує мало памьяті, складних обчислень, трафіку та електроенергії, тому може бути використаний і на слабких пристроях.

Перелік джерел посилання

1. IoT and non-IoT active device connections worldwide from 2010 to 2025, https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/
2. Authentication, Wikipedia, https://en.wikipedia.org/wiki/Authentication
3. New features of Authentication Scheme for the IoT: A Survey, IoT S&P 19, November 15, 2019, London, United Kingdom
4. Authentication Protocols for Internet of Things: A Comprehensive Survey, Department of Computer Science, 6 November 2017
5. The Top IoT Authentication Methods and Options, KeyFactor, https://www.keyfactor.com/blog/the-top-iot-authentication-methods-and-options/
6. An Overview of Different Authentication Methods and Protocols, Richard Duncan, October 23, 2001
7. IoT security system with modified Zero Knowledge Proof algorithm for authentication, Egyptian Informatics Journal, Volume 22, Issue 3, September 2021, Pages 269-276
8. How to Explain Zero-Knowledge Protocols to Your Children, Quisquater Jean Jacques, Mariam Muriel, Michael Guillou Louis, Marie Annick, Gaid, Anna, Gwenole, Soazig, 1998
9. Needham–Schroeder protocol, Wikipedia, Needham–Schroeder protocol, https://en.wikipedia.org/wiki/Needham-Schroeder_protocol
10. An attack on the Needham-Schroeder public key authentication protocol, Lowe, Gavin, November 1995
11. Feige-Fiat-Shamir and Zero Knowledge Proof, Medium, https://medium.com/asecuritysite-when-bob-met-alice/feige-fiat-shamir-and-zero-knowledge-proof-cdd2a972237c
12. Schnorr Identification and Signatures, David Mandell Freeman, October 20, 2011
13. EasyProfiler open-source library: https://github.com/yse/easy_profiler
14. Код програми: https://github.com/smelskiyd/FiatShamirAuthentication