Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ДОПУСТИТИ ДО ЗАХИСТУ:
в.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Іван ПАРХОМЕНКО
«____» червня 2023 р.

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи

галузь знань 12 Інформаційні технології
(шифр і назва галузі знань)

спеціальність 125 Кібербезпека
(код і назва спеціальності)

освітній ступень бакалавр

освітня програма Кібербезпека
(назва освітньої програми)

на тему: Програмний засіб керування одноразовими паролями на базі TOTP-алгоритму

Виконавець: студент IV курсу, групи КБ-42

_____ **Володимир МЕЛЬНИК**
(підпис) (Ім'я ПРІЗВИЩЕ)

	Ім'я, прізвище	**Підпис**
Керівник	Яніна ШЕСТАК	

Нормоконтроль	Сергій ДАКОВ	

Київ 2023

Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
в.о. завідувача кафедри
кібербезпеки та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

спеціальності 125 Кібербезпека
(код і назва спеціальності)

освітньої програми Кібербезпека
(назва освітньої програми)

студенту **КБ-42** **Володимир Віталійович Мельник**
(група) (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи Програмний засіб керування одноразовими паролями на базі TOTP-алгоритму

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Тема кваліфікаційної роботи затверджена на засіданні кафедри кібербезпеки та захисту інформації протокол №3 від 20.10.2022 р.

2. ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Алгоритми хешування, технології проектування та програмування, інформаційні ресурси.

3. ЗМІСТ РОЗРАХУНКОВО-ПОЯСНЮВАЛЬНОЇ ЗАПИСКИ

Поняття автентифікації, розвиток автентифікації, мультифакторна автентифікація, алгоритми генерації одноразових паролів, безпека при використанні одноразових паролів, розробка засобу керування одноразовими паролями.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Практична цінність Розробка засобу для зберігання секретних ключів та

генерації одноразових паролів.

5. ДАТА ВИДАЧІ ЗАВДАННЯ
Дата видачі завдання: 24 жовтня 2022 року

Завдання видала _____ Яніна ШЕСТАК
(підпис) (ім’я, прізвище)

Завдання прийняв до виконання _____ Володимир МЕЛЬНИК
(підпис) (ім’я, прізвище)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Найменування етапів випускної кваліфікаційної роботи	Термін виконання робіт (початок- кінець)	Відмітка про виконання
1	Уточнення постановки завдання	24.10.2022 – 01.11.2022	*виконано*
2	Аналіз літератури	02.11.2022 – 08.11.2022	*виконано*
3	Огляд технологій автентифікації	09.11.2022 – 14.12.2022	*виконано*
4	Аналіз алгоритмів генерації одноразових паролів	15.12.2022 – 28.12.2022	*виконано*
5	Визначення вимог, для захищеності роботи одноразових паролів	29.12.2022 – 14.01.2023	*виконано*
6	Базова реалізація алгоритму TOTP	15.01.2023 – 26.01.2023	*виконано*
7	Розробка засобу керування одноразовими паролями	27.01.2023 – 09.04.2023	*виконано*
8	Порівняльний аналіз розробленого засобу з іншими рішеннями	10.04.2023 – 30.04.2023	*виконано*
9	Оформлення пояснювальної записки	01.05.2023 – 31.05.2023	*виконано*
10	Підготовка до захисту кваліфікаційної роботи	01.06.2023 – 12.06.2023	*виконано*

Завдання видала _____ Яніна ШЕСТАК
(підпис) (ім’я, прізвище)

Завдання прийняв до виконання _____ Володимир МЕЛЬНИК
(підпис) (ім’я, прізвище)

Термін подання кваліфікаційної роботи до ЕК 12 червня 2023 року

РЕФЕРАТ

Пояснювальна записка: 64 с., 13 рис., 2 табл., 2 додатка, 28 джерел.

Мета роботи: Розробка програмного засобу, який дозволить захищати секретні ключі та генерувати одноразові паролі, для посилення безпеки автентифікації.

Об’єкт дослідження: процес автентифікації за допомогою одноразових паролів.

Предмет дослідження: алгоритми генерації одноразових паролів.

Методи дослідження: порівняння, системний аналіз, класифікація.

Практичне значення роботи полягає у розробці програмного засобу для керування одноразовими паролями, який може використовуватись для підвищення безпеки облікових даних в застосунках або веб-сервісах.

Результати здійснених у кваліфікаційній роботі досліджень можуть бути використані для посилення захисту облікових записів в інтернет сервісах.

Ключові слова: автентифікація, мультифакторна автентифікація, одноразовий пароль, мобільний пристрій, криптографічний алгоритм, програмний засіб, алгоритм хешування, Android Keystore.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

API – Application Programming Interface
CTAP – Client to Authenticator Protocol
CTSS – Compatible Time-Sharing System
GCM – Galois/Counter Mode
HMAC – Hash-based Message Authentication Code
HOTP – HMAC-based One Time Password
IV – Initialization Vector
MFA – Multi-Factor Authentication
MIT – Massachusetts Institute of Technology
MITM – Man In The Middle
OATH – Open Authentication
OCRA – OATH Challenge-Response Algorithm
OTP – One Time Password
RFC – Request For Comments
RFID – Radio-Frequency Identification
TOTP – Time-based One Time Password
U2F – Universal 2nd Factor
ОС – Операційна Система

ЗМІСТ

ВСТУП

Актуальність. Безпека автентифікації завжди була проблемою в інформаційних системах. По своїй природі, автентифікація є процесом перевірки, що "ти є, ким ти представляєшся". Використовується автентифікація майже в усіх веб-сервісах, локальних мережах тощо, для розмежування доступу до даних, акаунтів та іншої конфіденційної інформації.

Стандартна парольна автентифікація, не завжди є надійним фактором автентифікації. Пароль можна підглянути, підібрати, або виманити у користувача через фішингові сторінки. Компрометація паролю призведе до отримання зловмисником повного доступу до акаунта. Одним з варіантів вирішення цієї проблеми є впровадження декількох факторів автентифікації. Таким чином втрата, злам чи будь-яка інша компрометація одного з факторів, не розкриє інші фактори і не дозволить отримати доступ до акаунта. Мультифакторна автентифікація є актуальною і широко використовується в веб-сервісах, особливо тих що зберігають інформацію, розголос якої може спричинити значної шкоди користувачам (поштові сервіси, банківські акаунти, хмарні сховища тощо).

Одним з найбільш поширених способів додавання мультифакторної автентифікації до вже наявної системи є використання одноразових паролів як другого фактору. Такий спосіб дозволяє дешево та достатньо надійно захищати облікові записи користувачів без надмірних зусиль з боку користувача.

Мета дослідження. Метою роботи є розробка програмного засобу, який дозволить захищати секретні ключі та генерувати одноразові паролі, для посилення безпеки автентифікації, шляхом використання додаткового фактору автентифікації.

Для досягнення зазначеної мети кваліфікаційної роботи, та розкриття теми поставлені окремі завдання:

- провести аналітичний огляд наявних практик щодо використання факторів автентифікації;
- провести аналіз алгоритмів генерації одноразових паролів;

- визначити необхідні вимоги, для забезпечення захищеності роботи алгоритму TOTP;
- розробити мінімальну реалізацію алгоритму TOTP;
- розробити захищений додаток, для керування та збереження секретних значень і автоматичної генерації одноразових паролів на базі алгоритму TOTP;
- провести порівняння розробленого програмного засобу з існуючими рішеннями.

Об'єкт дослідження. Процес автентифікації за допомогою одноразових паролів.

Предмет дослідження. Алгоритми генерації одноразових паролів.

Практична цінність одержаних результатів. Результати дослідження можуть бути корисними для фахівців з криптографії та використані, для розробки нових методів генерації одноразових паролів або для покращення автентифікації в нових чи існуючих системах.

РОЗДІЛ 1
АНАЛІЗ АВТЕНТИФІКАЦІЇ

1.1 Поняття автентифікації

Автентифікація — процедура встановлення належності користувачеві інформації в системі пред'явленого ним ідентифікатора [1].

В інформаційних та інформаційно-комунікаційних системах для надання доступу до функцій системи та даних певному користувачу виконуються декілька процедур.

Перший етап — ідентифікація. Користувач пред'являє певний унікальний ідентифікатор, такий як логін, який система може розпізнати і знайти в базі даних, тощо. В залежності від реалізації, якщо ідентифікатор не знаходиться системою, користувач може отримати помилку на цьому або наступному етапі.

Другий етап — автентифікація. На цьому етапі користувач підтверджує системі, що йому належить наданий ідентифікатор, шляхом надання паролю або іншого аналогу, що знає чи має лише користувач.

Останній, третій етап — авторизація. Після успішної ідентифікації та автентифікації, система надає користувачу доступ чи повноваження до частини функцій та даних, які дозволені для даного користувача.

З перелічених етапів, чи не найважливішим процесом є саме автентифікація. Неправильна автентифікація користувача, може як помилково заборонити авторизацію справжнього користувача, так і помилково надати користувачу більше повноважень ніж необхідно. Особливу загрозу становлять зловмисники, що цілеспрямовано намагаються зламати або обійти систему автентифікації, для отримання несанкціонованого доступу до інформації.

1.2 Розвиток автентифікації

В 1961 році була представлена перша операційна система загального призначення з поділом часу — Compatible Time-Sharing System (CTSS) [2]. Розроблена в Массачусетському технологічному інституті вона працювала на комп'ютерній системі IBM 709. Оскільки системою могли користуватися декілька користувачів, кожен мав свою теку, для зберігання файлів в пам'яті. Це, в свою чергу, створило проблему розмежування доступу до цих файлів — лише власник повинен мати можливість переглядати, записувати і видаляти свої файли.

Дослідники MIT задля вирішення цієї проблеми встановили для кожного користувача системи свій пароль, таким чином чи не вперше в світі використавши парольну автентифікацію в комп'ютерній системі [2]. Текстові паролі було легко зберігати в пам'яті комп'ютера і процес перевірки паролів не вимагав великих обчислювальних здатностей, тому це було досить очевидне рішення для захисту файлів користувачів.

Проте, з точки зору безпеки, дана система не була хорошим захистом. Дослідники в MIT не приділяли багато уваги безпеці. Серед вразливостей і недоліків, що були присутні в CTSS, наведено деякі:

1. паролі зберігались в файлі в незашифрованому вигляді;
2. будь-хто з правами суперкористувача міг проглянути, чи зробити копію файлу з паролями;
3. будь-хто з фізичним доступом до накопичувачів міг прочитати парольний файл;
4. не було вимог щодо паролів (мінімальна довжина, наявність спеціальних символів тощо).

Незабаром з'явилась і перша особа, що змогла отримати доступ до файлу з паролями. Алан Шерр, дослідник в MIT, хотів збільшити час впродовж якого він міг користуватися CTSS. Знайшовши шлях до файлу, він зміг обійти будь-які обмеження на доступ до цього файлу і роздрукувати його. Про цей випадок він зізнався лише через 25 років.

Ще один цікавий випадок пов'язаний з CTSS стався пізніше в 1960-х роках. Системний адміністратор редагував парольний файл, паралельно йому, інший адміністратор редагував повідомлення, що висвічувалося в терміналі кожного дня під час логіну користувача. Під час одночасного редагування, в програмі сталася помилка, яка спричинила наступне: тимчасові буфери редагування цих двох терміналів були замінені один одним. В результаті цього впродовж певного часу в терміналі кожного користувача при логіні можна було побачити список всіх паролів [3].

В 1970-х роках була випущена стаття "Password Security: A Case History", написана Робертом Моррісом та Кеном Томпсоном [3]. В статті послідовно описується історія використання та зберігання паролів в UNIX системі, недоліки та вразливості підходу, можливі рішення для посилення захисту паролів. Зокрема стаття згадує можливість перебору паролів грубою силою та наводить час за який можна перебрати всі паролі довжини n що використовують різні множини символів (таблиця 1.1), як аргумент для встановлення більш складних паролів.

Таблиця 1.1

Час необхідний для перебору паролів на комп'ютері PDP-11/70

	Набори символів				
n	26 літер нижнього регістру	36 літер нижнього регістру і цифр	62 літери і цифри	95 друкованих символів	128 ASCII символів
1	30 мс.	40 мс.	80 мс.	120 мс.	160 мс.
2	800 мс.	2 с.	5 с.	11 с.	20 с.
3	22 с.	58 с.	5 хв.	17 хв.	43 хв.
4	10 хв.	35 хв.	5 год.	28 год.	93 год.
5	4 год.	21 год.	318 год.	-	-
6	107 год.	-	-	-	-

Стаття також описує експеримент, спрямований на визначення звичок користувачів, у випадку якщо потрібно створити пароль без будь-яких вимог до нього. Серед 3289 паролів 86% мали менше 6 символів в них або були словами чи

іменами, які можна знайти в словнику. Як можливі покращення до тодішнього стану шифрування паролів, автори наводять: додавання вимог до довжини чи символів в паролі, додавання солі (випадкового числа) до паролів перед їх шифруванням, використання повільнішого алгоритму шифрування тощо.

Також в 1970-х роках, британський криптограф Джеймс Елліс продемонстрував можливість захищеної комунікації, яка не потребувала б обміну симетричним ключем для шифрування повідомлень [4]. Елліс назвав такий метод "non-secret encryption", дослівно — не засекречене шифрування. Цей метод шифрування та обміну повідомленнями розумів під собою що вся комунікація і всі надіслані зашифровані повідомлення можуть бути прочитані третьою стороною, але при цьому вміст повідомлень не міг би бути визначений.

Те що в своїй роботі описував Елліс, пізніше стало відомим як асиметричне шифрування або ж шифрування з відкритим ключем. Першими розробленими асиметричними алгоритмами були RSA та алгоритм Діффі-Геллмана.

Алгоритм RSA вперше був описаний математиком Кліфордом Коксом в секреті в центрі урядового зв'язку Великої Британії. Його робота не була розкрита до 1997 року [5]. Кількома роками пізніше, в 1977, незалежно від Кокса, декілька дослідників в MIT, Рональд Рівест, Аді Шамір і Леонард Адлеман, також дійшли до цього алгоритму і опублікували його опис [6].

Алгоритм RSA ґрунтується на проблемі факторизації великих чисел. Для роботи, алгоритм обирає два великих простих числа (512 біт чи більше), але у відкритому вигляді передає лише їх добуток. Маючи ці два числа не є складним процесом обчислити відкритий та закритий ключі, але маючи лише добуток цих чисел, розділити його на множники є складною задачею при будь-яких обчислювальних потужностях, за умови достатньо великих чисел.

Інший алгоритм — Діффі-Геллмана, є алгоритмом для узгодження криптографічних ключів, також винайдений в Великій Британії в секреті Малколмом Вільямсоном і пізніше публічно представлений Вітфілдом Діффі і Мартіном Геллманом [7]. Алгоритм ґрунтується на складності обчислення дискретного

логарифма, і може використовуватися лише для розподілу секретних ключів, але не для шифрування повідомлень.

Також в своїй роботі Діффі та Геллман описують концепт "односторонньої автентифікації" або цифрового підпису, що також є важливою сферою в криптографії. За допомогою асиметричних криптосистем, користувач А може "розшифрувати" повідомлення яке він бажає надіслати своїм приватним ключем, і тоді користувач Б, що бажає отримати це повідомлення і підтвердити що воно надійшло від першого користувача, може "зашифрувати" отримане повідомлення відкритим ключем користувача А.

Починаючи з 1980-х років, почався розвиток одноразових паролів. Оскільки інформаційні і комп'ютерні системи масово використовували паролі як єдиний фактор автентифікації, дослідники та хакери постійно шукали способи зламати чи обійти їх. Один із найбільших ризиків з парольною системою полягає в тому, що якщо зловмисник зможе вгадати, викрасти або перехопити пароль певного користувача, він потенційно матиме постійний доступ до акаунту, даних та іншої конфіденційної інформації доти, доки користувач не змінить пароль. Постійні збільшення вимог з безпеки до парольної автентифікації привели до нової концепції. Якби користувач постійно мав би інший пароль кожного разу при вході у систему, це б зменшило деякі ризики присутні в статичних паролях.

Описаний концепт отримав загальну назву "одноразовий пароль" (OTP). Одним з перших стандартів генерації одноразових паролів був S/KEY. Розроблений в Bellcore, алгоритм мав на меті надати можливість безпечної автентифікації з незахищених чи ненадійних пристроїв, в яких використання постійного довготривалого паролю не є бажаним. Алгоритм базується на хеш-функціях, для генерації паролів та захисту. Кожен пароль може бути використаним лише один раз, після чого система що проводить автентифікацію повинна приймати лише наступний в черзі пароль.

1.3 Використання декількох факторів для автентифікації

Створення стандартів для генерації одноразових паролів стало першим кроком, для поширення застосування мультифакторної автентифікації (MFA) в різних сферах. Поняття мультифакторної або двофакторної автентифікації означає використання декількох методів (факторів) автентифікації, для підтвердження справжності користувачів. Серед очевидних переваг безпеки — хакеру буде недостатньо зламати лише один фактор для успішного проходження автентифікації, поки інші фактори залишаються незламаними. Оскільки різні алгоритми та технології використовуються, для різних факторів автентифікації, MFA також ускладнює викрадання, підбір чи виманювання парольних даних у користувачів.

Хоча також варто віддати належне звичайній парольній автентифікації. Оскільки дана технологія використовується впродовж досить довгого періоду, дослідники та спеціалісти з безпеки створили достатньо вимог і рекомендацій до систем автентифікації для їх захисту. Парольна автентифікація пройшла чимале випробування віком. При правильному використанні алгоритмів хешування та маючи достатню ентропію кожного пароля, зламати пароль не є легко виконуваним завданням. Але, цих вимог не завжди дотримуються як користувачі, так і адміністратори чи розробники систем автентифікації.

Для зменшення ризиків, щодо використання лише одного фактору автентифікації, організації, що мали необхідність в дуже високій безпеці, такі як банківські системи, державні установи, великі підприємства тощо використовували мультифакторну автентифікацію задовго до створення відкритих стандартів. Здебільшого організації створювали власні закриті методи автентифікації, для додаткових факторів що використовувались разом зі звичайним паролем.

З розвитком технологій, все більше і більше організацій додавали мультифакторну автентифікацію до своїх сервісів. Найбільш зручним і простим в реалізації методом були одноразові паролі. Банківські системи й досі надсилають одноразові цифрові коди через SMS-повідомлення, для підтвердження транзакцій чи інших дій з банківським акаунтом. Деякі онлайн сервіси дозволяють сканувати QR-

код через спеціальну програму-генератор, що генерує одноразові коди кожні 30 секунд, які потім використовують під час логіну.

Починаючи з 2010-х років, біометрична автентифікація стала ще більш доступною, з появою перших смартфонів з наявним сканером відбитків пальців. Підтримка від технологічних організацій і написання програмних інтерфейсів надали можливості для відносно легкої інтеграції біометрії в наявні програми. Це дозволило водночас посилити безпеку, оскільки біометричні дані вкрай складно підробити, полегшити і пришвидшити процес автентифікації для кінцевих користувачів — не потрібно запам'ятовувати паролі, мати при собі фізичний ключ тощо.

На сьогоднішній день, більшість організацій і користувачів мають змогу користуватися двофакторною автентифікацією. Проте велика частка користувачів самостійно не обирає підключення інших факторів автентифікації, при наявності такої опції в сервісі. Згідно з дослідженням компанії Duo Labs, станом на 2019 рік лише 53% опитуваних жителів США та Великої Британії користувалися двофакторною автентифікацією. Це є покращенням порівняно зі схожим дослідженням за 2017 рік, де лише 28% опитуваних користувались двофакторною автентифікацією. На запитання щодо онлайн акаунтів, які користувачі вважали найбільш вартими захисту, 85% опитуваних надали високий пріоритет банківським та фінансовим акаунтам. Всі ж інші категорії, такі як комунікації, соціальні мережі, здоров'я, розваги тощо, отримали високий пріоритет від 32% чи меншого відсотку опитуваних (рисунок 1.1) [8].

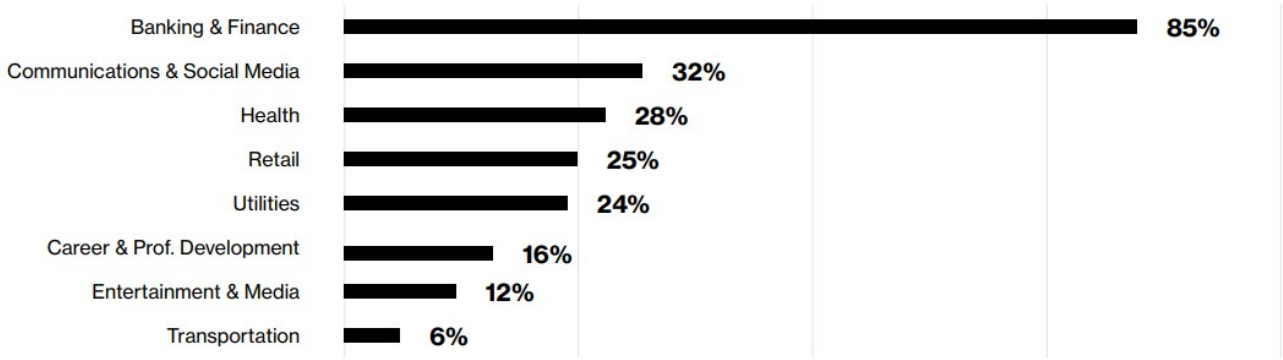

Рисунок 1.1 — Пріоритетність захисту онлайн акаунтів за типом

Поширення використання декількох факторів автентифікації серед користувачів є актуальною проблемою для організацій. Надаючи опцію в налаштуваннях, для використання двофакторної автентифікації за допомогою одноразових кодів, яка не увімкнена за замовчуванням, може захистити лише користувачів, які піклуються про безпеку. Проте для користувачів, які так і не вмикають дане налаштування, нічого не змінюється. Очевидним рішенням проблеми може бути примусове використання двофакторної автентифікації без можливості її відключення. З іншого боку, таке нововведення негативно відобразиться на використанні сервісу користувачами, збільшивши рівень їх незадоволення.

Важливо знайти баланс — не занадто ускладнювати процес автентифікації, зберігаючи легкість в користуванні, і водночас зберігати достатню безпеку акаунтів від зламу і крадіжок. Вищенаведена біометрична автентифікація є хорошим прикладом такого методу автентифікації. Розробка та використання подібних методів або полегшення використання існуючих, позитивно вплине на обидва аспекти онлайн сервісів.

В 2019 році було опубліковано дослідження Google в їхньому блозі що мало на меті перевірити наскільки альтернативи до двофакторної автентифікації на базі одноразових паролів, що Google використовує для акаунтів, є успішними в захисті від крадіжок акаунтів (account hijacking) [9]. При спробі увійти в акаунт, Google використовує два типи запитів для додаткового підтвердження, що саме власник намагається зайти в свій акаунт:

1. Device-based challenges — запити на основі пристрою, зазвичай мобільного телефону. Використовуються, якщо користувач попередньо увійшов в свій акаунт на телефоні або прив'язав номер телефону до акаунта.

2. Knowledge-based challenges — запити на основі знання (додаткова поштова адреса, останнє місцезнаходження тощо). Цей тип має менший пріоритет, порівняно з попереднім, через відносно меншу надійність.

На рисунку 1.2 представлено статистику, щодо запобігання крадіжок акаунтів, яку було отримано як результат проведеного дослідження. Серед досліджуваних способів підтвердження користувачів наведено 3 запити на основі пристрою та 3

запити на основі знання. Серед можливих атак на акаунти користувачів наведено автоматичні атаки, фішингові атаки та таргетовані атаки. Як можна побачити, запити на основі пристрою можуть протидіяти переважній більшості атак. Найбільшу ефективність поміж ними мають ключі безпеки та сповіщення на пристрої. Розглядаючи запити на основі знань, вони здебільшого ефективні проти автоматичних атак. Проте хоча б посередній захист проти фішингових та таргетованих атак може надати лише додаткова поштова адреса, що працює в 68-79 відсотках випадків. Найгірші показники має перевірка останнього місцезнаходження — лише 10 відсотків фішингових атак не впорались з нею, в той час як всі таргетовані атаки пройшли перевірку успішно.

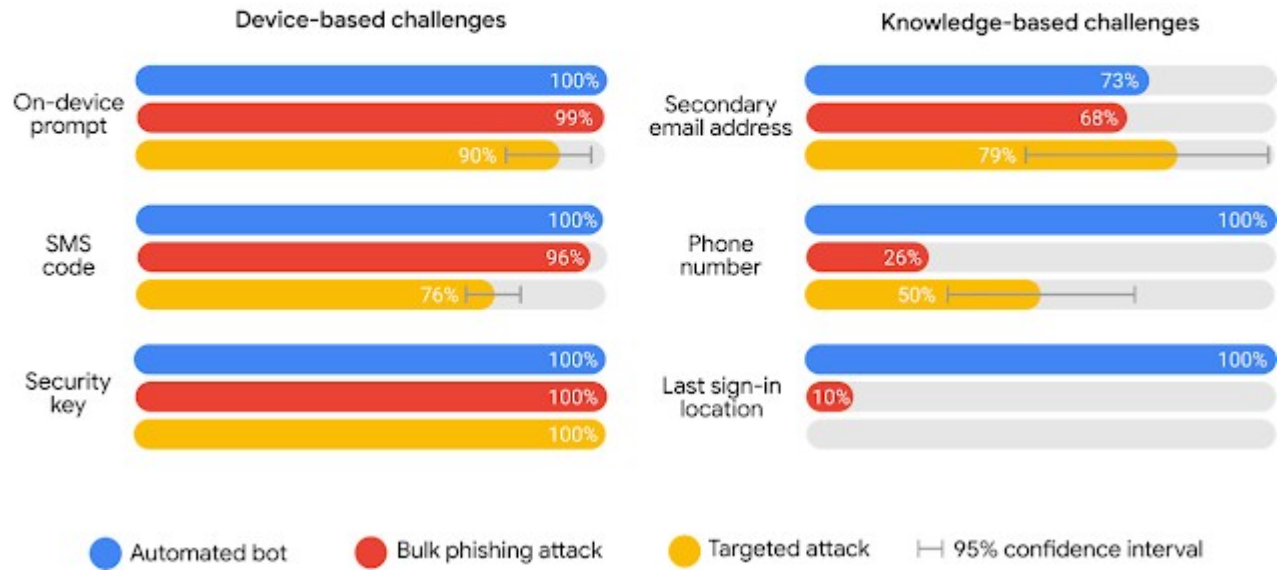

Рисунок 1.2 — Успішність запобігання крадіжкам акаунтів за різними запитами

Проаналізувавши розглянуті дані, можна зробити висновок, що компанія Google створила достатньо збалансовану систему захисту, що може протистояти спробам входу в акаунт від сторонніх осіб, при цьому не вимагаючи особливо важких зусиль від користувачів.

1.4 Різновиди факторів автентифікації

Для того щоб мультифакторна автентифікація надавала дійсно посилений захист в порівнянні з однофакторною автентифікацією, необхідно щоб фактори були різні за своїми характеристиками, структурою тощо. В іншому випадку, якщо використовувати два однакових фактори, як от два паролі, складність атаки не надто збільшиться, отримати чи підібрати другий пароль можна в той же спосіб, що й попередній. Або, знайшовши вразливість в певному компоненті системи автентифікації, завдяки якій можна обійти один фактор автентифікації, використання того ж компоненту вдруге означатиме повний обхід захисту.

Зазвичай виділяють три фактори, що можуть бути використані для автентифікації [10]:

1. Фактор знання — щось, що користувач знає.
2. Фактор володіння — щось, що користувач має.
3. Фактор властивості — щось, що чим користувач є.

Перший фактор, знання, має на меті перевірити певну секретну інформацію, яку має знати лише користувач. Цією інформацією може бути пароль, парольна фраза, цифровий код, тощо. Різноманітні запитання, що використовуються в деяких системах, про персональну інформацію на кшталт імені домашньої тварини чи школу, в якій користувач навчався, також відносяться до цього фактору.

Через свою простоту парольна автентифікація є найбільш легкою в імплементації і тому найбільш розповсюдженою. Від користувача вимагається лише створення та зберігання в секреті чи запам'ятовування паролю. Через те що майже будь-який інтернет сервіс вимагає пароль для створення акаунта, для зберігання паролів були створені менеджери паролів — спеціальні програмні додатки, що надійно зберігають паролі в зашифрованому вигляді, а деякі також надають можливість створення випадкових паролів для більшої безпеки.

Проте через відносну простоту парольна автентифікація є однією з найненадійніших і найбільш легко зламуваних. Серед основних недоліків можна виокремити декілька:

1. Створення користувачами легких до перебору паролів. Якщо система автентифікації не встановлює вимоги до складності паролів, частина користувачів буде використовувати легкі для запам'ятовування паролі, як своє ім'я, прізвище, дату народження тощо або довільну комбінацію з наведених варіантів.

2. Недотримання користувачами встановлення різних паролів для різних акаунтів. Ще один людський фактор, спричинений надмірною кількістю паролів які необхідно пам'ятати.

3. Простота перехоплення паролю через фішингові чи MITM атаки.

4. Недотримання організаціями коректного зберігання паролів. При витоку даних з серверів певної компанії, хакери можуть отримати паролі, якщо вони не були збережені з використанням односторонніх криптографічних функцій. Такі вкрадені бази паролів потім можуть використовуватися як словники, для отримання доступу до інших акаунтів. Лише в першій половині 2019 року, було вкрадено більше 4 мільярдів записів у 3800 витоках даних [11].

Другий фактор, володіння, перевірка якогось унікального предмета, яким володіє користувач. Перевагами даного фактору є більша складність підробки та можливість крадіжки лише у фізичному вигляді. Серед недоліків можна виокремити декілька. По-перше, для зчитування інформації з фізичного предмета, необхідні спеціальні пристрої, що під'єднуються до комп'ютера чи смартфона, а також необхідне відповідне програмне забезпечення, для роботи обраного зчитувача. З першого пункту випливає другий — надлишкові витрати, що збільшують вартість системи автентифікації. По-третє, необхідно врахувати можливість псування, поламки, втрати, загалом перетворення пристрою на нефункціональний.

До пристроїв, які відносяться до фактору володіння, належать ключі безпеки, RFID мітки, магнітні картки та інші фізичні токени (рисунок 1.3). За своєю структурою, апаратні засоби чи програмні додатки, які генерують одноразові паролі, також відносяться до фактору володіння. Секретні значення необхідні для роботи алгоритмів, що зберігають пристрої генерації паролів, можна вважати як своєрідні токени і лише користувач має їх в своїй власності. Відсутність необхідності в

додатковому обладнанні, для перевірки одноразових паролів є одною з причин, що вплинули на популярність цього методу автентифікації.

Рисунок 1.3 — USB-ключ Yubico та RFID картка Zipato

Третій фактор, властивості, загалом розуміє під собою використання біометричних ідентифікаторів для автентифікації. Як унікальний ідентифікатор може використовуватися відбиток пальця, райдужна оболонка ока, сітківка ока, ДНК, лице, голос та інші особливості людського тіла. З точки зору безпеки, підробка більшості біометричних даних, за умови ідеальної роботи пристроїв сканування, є надскладною задачею і можливою лише за умови попереднього сканування цих даних у власника. Для користувача ж, такий спосіб автентифікації не вимагає жодних зусиль на кшталт запам'ятовування паролів чи носіння з собою додаткових пристроїв, як в попередніх двох факторах.

На жаль, біометрична автентифікація має численну кількість недоліків. Так само як і фактор володіння, для сканування біометричних даних необхідні спеціальні пристрої. Через специфіку біометричного сканування, такі пристрої є дорожчими в порівнянні зі сканерами цифрових носіїв. Сканування не цифрової інформації є менш точним, і може бути покращене лише кращим апаратним забезпеченням, що в свою чергу ще більше збільшує ціну системи. Додатково, під час перевірки біометричних даних, виникає проблема некоректного позитивного (false positive) результату — коли система розпізнає вхідні дані, які не зберігаються в

системі або не мають відкривати доступ до системи чи інформації, як коректні, надаючи доступ неавторизованій особі. При теоретичній компрометації біометричних даних, що зберігалися в певній системі, замінити частину тіла не є можливим, на відміну від паролів чи фізичних ключів безпеки. І на останок, у разі виникнення ситуації, коли зловмисник буде вимагати від користувача необхідні дані, для автентифікації під його особою, біометричні дані можна буде зняти хіба з трупа, що в такому випадку становитиме пряму загрозу життю та здоров'ю людини, що користується біометричною автентифікацією.

1.5 Технології спрямовані на досягнення безпарольної автентифікації

З усіх незручностей в інтернеті, необхідність створювати та запам’ятовувати незліченну кількість паролів мабуть займає перше місце в списках більшості людей. В середньому на кожну людину припадає більше ста паролів від різних вебсайтів, соціальних мереж, шопінгу, різноманітних платформ для ігор чи бізнесу [12]. І це занадто велика кількість паролів. Не дивно що багато користувачів використовують прості або однакові паролі для більшості акаунтів.

Хоча можна використовувати менеджери паролів для генерації та зберігання всіх паролів, щоб спростити процес, проте це не звільняє користувача від необхідності підтримувати та оновлювати цей список і запобігати втраті доступу до менеджеру чи до паролю від нього. Водночас необхідно бути впевненим, що облікові дані для менеджера паролів ніколи не будуть скомпрометовані.

FIDO Alliance — асоціація, місія якої посилити автентифікацію онлайн. Досягнення цієї мети здійснюється за допомогою:

1. Розробки технічних специфікацій, що визначають відкриті, масштабовані, сумісні механізми, які замінюють використання паролів для безпечної автентифікації користувачів онлайн-сервісів.

2. Експлуатації галузевих програм, для забезпечення успішного впровадження специфікацій у всьому світі.

3. Подання технічних специфікацій до визнаних організацій із розробки стандартів для офіційної стандартизації [13].

В жовтні 2014 року, FIDO Alliance представила стандарт протоколу Universal 2nd Factor, для використання в якості другого фактору автентифікації. Протокол U2F дозволяє онлайн-сервісам посилити безпеку існуючої інфраструктури паролів, додавши надійний другий фактор для входу користувача, в той же час зберігаючи його конфіденційність. Користувач, як і раніше, входить до системи за допомогою імені користувача та пароля. Надійний другий фактор в цьому випадку дозволяє сервісу спростити свої паролі (наприклад до 4-значного PIN-коду) без послаблення безпеки.

Під час реєстрації та автентифікації користувач пред'являє другий фактор, просто натискаючи кнопку на USB або NFC пристрої. Користувач може використовувати свій U2F пристрій у всіх онлайн-сервісах, які підтримують протокол, використовуючи вбудовану підтримку у веб-браузерах. Коли користувач реєструє пристрій U2F в акаунті на певному вебсайті, пристрій створює нову пару ключів, яку можна використовувати тільки на цьому вебсайті, надаючи йому відкритий ключ для асоціації з даним акаунтом [14].

Протокол U2F є лише одним фактором автентифікації і презентується як додаток до вже існуючої системи автентифікації, тому не може бути використаний для повної її заміни. Проте цей стандарт став поштовхом для інших стандартів та протоколів, що наближають технології до повністю безпарольної автентифікації. Серед них — специфікації Client To Authenticator Protocol (CTAP) і WebAuthn, обидві з яких входять до FIDO2 project.

Протокол CTAP, представлений FIDO Alliance, дозволяє вебсайтам чи додаткам, що запускаються на певній платформі чи комп'ютерній системі, під'єднуватися до зовнішнього пристрою автентифікатора, такого як смартфона або ключа безпеки. Автентифікатор може під'єднуватися до комп'ютера будь-яким захищеним каналом передачі даних, тому для цього можуть використовуватися USB, NFC або Bluetooth. Також, під час запиту парольних даних сервісом, автентифікатор

має підтвердити взаємодію з користувачем шляхом натискання кнопки, введення пін-коду або паролю, біометричного сканування або іншої дії [15].

Специфікація протоколу CTAP визначає дві його версії: CTAP1/U2F і CTAP2. Версія CTAP1/U2F призначена для сумісності з U2F автентифікаторами. А автентифікатори, що підтримують протокол CTAP2, також називаються FIDO2 або WebAuthn автентифікаторами.

WebAuthn це API, що дозволяє веб сервісам створювати та використовувати надійні парольні дані на основі відкритих ключів для автентифікації користувачів. Стандарт Web Authentication був опублікований організацією World Wide Web Consortium спільними зусиллями з FIDO Alliance. WebAuthn визначає вимоги до інтернет сервісів, браузерів і клієнтів для автентифікації за допомогою зовнішніх автентифікаторів, а також описує процес за яким веб сайти можуть реєструвати та автентифікувати користувачів за цим API [16].

Реєстрація автентифікаторів FIDO використовуючи вищенаведені технології відбувається в декілька етапів:

1. Користувачеві пропонується вибрати доступний автентифікатор, який є сумісним з інтернет сервісом.

2. Користувач розблоковує автентифікатор за допомогою пристрою для зчитування відбитків пальців, кнопки на другому пристрої, надійно введеного PIN-коду або іншим способом.

3. Пристрій користувача створює нову пару ключів — відкритого і закритого, унікальну для локального пристрою, інтернет сервісу та облікового запису користувача.

4. Відкритий ключ надсилається до інтернет сервісу і пов’язується з обліковим записом користувача. Закритий ключ і будь-яка інформація про метод локальної автентифікації ніколи не залишають локальний пристрій.

Автентифікація відбувається за схожим алгоритмом, але на заміну створенню нової пари ключів, клієнт підписує запит надісланий сервісом своїм закритим ключем, який потім перевіряється відкритим ключем на сервері [17]. Таким чином ні

сервіс, ні браузер, ні навіть клієнтський комп'ютер не отримують закритий ключ в жоден момент часу.

Висновки за розділом 1

Проблема автентифікації бере початок ще з 1960-х років, коли було створено комп'ютерну систему з декількома користувацькими акаунтами. Маючи численні недоліки, такі як зберігання паролів в незашифрованому вигляді, використання користувачами занадто коротких паролів та інші, система не була достатньо захищеною і тому була експлуатована дослідником. Цей приклад показав, що потрібно приділяти більше уваги захисту автентифікації і зберіганню парольних даних.

З часом, технології розвивались, надаючи нові можливості для кращої та безпечнішої автентифікації. Серед них: алгоритми асиметричного шифрування, цифровий підпис, одноразові паролі, біометрія тощо. Поєднавши два або більше способи автентифікації з'являється технологія мультифакторної автентифікації, що ще більше посилила безпеку.

Проте, за винятком коли мультифакторна автентифікація активується за замовчуванням, користувачі ж самі не завжди вмикають її в налаштуваннях. Постає нова проблема — досягнення балансу між безпекою і зручністю в користуванні. Розробка та поширення нових зручних і безпечних способів автентифікації має бути пріоритетним завданням. FIDO Alliance є одною з організацій, які прагнуть допомогти вирішити це питання, в даному випадку розвиваючи технології безпарольної автентифікації, хоча й значного поширення вони поки не набули.

РОЗДІЛ 2
ДОСЛІДЖЕННЯ ОДНОРАЗОВИХ ПАРОЛІВ

2.1 Концепт одноразових паролів

Одноразовими паролями називаються паролі, що можуть бути використані лише для однієї автентифікації. Після використання або проходження певного часу, одноразові паролі перестають бути ефективними.

В порівнянні зі звичайними постійними паролями, одноразові паролі, як концепт, вирішують деякі проблеми. Найбільш очевидною є неможливість повторного використання пароля зловмисником. Таким чином підглядання, кейлогери та будь-які інші перехоплення пароля втрачають свою ефективність, якщо спроби використання паролю йдуть після успішної автентифікації цим паролем справжнього користувача. Оскільки одноразові паролі здебільшого генеруються або випадковим чином, або за допомогою математичних функцій, а не користувачем, можна виключити можливість перебору найбільш популярних паролів зловмисником.

При правильній реалізації системи автентифікації, зловмисник взагалі не повинен мати можливість послідовно вводити велику кількість паролів з метою вгадати правильний, оскільки система повинна оновлювати пароль або забороняти вхід іншими способами після певної кількості неправильних спроб або визначеного часового проміжку.

Проте, як і будь-яка система, одноразові паролі мають певні недоліки, які можна визначити ще на стадії концепту. Одноразові паролі майже не захищають користувачів від фішингу, якщо пароль використовується одразу після його перехоплення. При надсиланні паролів за допомогою SMS повідомлень, існують атаки, спрямовані на перехоплення цих повідомлень або перереєстрацію SIM картки та подальшого отримання на неї повідомлень. Для паролів, що генеруються на основі певного секретного значення, перехоплення цього секретного значення

означатиме повну компрометацію майбутніх одноразових паролів створених на основі цього значення.

2.2 Способи генерації та розповсюдження одноразових паролів

Щоб користувач отримав одноразовий пароль, який необхідно ввести при автентифікації, існує два найпоширеніших методи:

1. генерація паролю на стороні сервера та його надсилання на електронну пошту користувача, телефон тощо;
2. генерація паролю математичними методами на стороні користувача.

В першому випадку, процес генерації одноразового паролю не є важливим. Для цього методу необхідно лише мати певний захищений канал з користувачем, щоб надіслати пароль. Зазвичай, при реєстрації на інтернет сервісі, користувач змушений надати свою електронну пошту або номер телефону, до яких неявно очікується, що має доступ лише користувач. Ці способи зв'язку й використовуються для передачі паролю.

Додаткова перевірка користувача шляхом надсилання йому коду на пошту широко використовується багатьма інтернет сервісами, здебільшого у випадку, якщо користувач заходить в свій акаунт з іншого пристрою чи IP адреси. Незручна особливість такого другого фактору автентифікації, що велика частка сервісів не надає можливості вимкнути цю перевірку по електронній пошті або замінити її іншим способом, що може негативно вплинути на певних користувачів.

Другий метод розповсюдження одноразових паролів спирається на математичні алгоритми, для генерації паролів на стороні користувача. Зазвичай для роботи цих алгоритмів необхідно мати хоча б одне спільне секретне значення або ключ, яким мають володіти як клієнт, так і сервер. Передача цього ключа здійснюється одноразово, в процесі ініціалізації математичного алгоритму, після чого генерація одноразових паролів може почати виконуватися.

Оскільки для отримання паролів не потрібен ніякий канал зв'язку, генерація паролів може здійснюватися повністю офлайн, що додає захист від можливих

перехоплень. Більше того, згенеровані паролі не розкривають достатньо інформації щодо наступних паролів.

2.3 Алгоритми генерації одноразових паролів

Загалом, можна виділити декілька схем для генерації одноразових паролів:

1. Генерація паролів на основі попереднього паролю, створюючи таким чином ланцюжок паролів, які необхідно використовувати в визначеному порядку.

2. Генерація паролів на основі лічильника або часу.

3. Генерація паролів у відповідь на запит (challenge).

Кожну схему буде детальніше розглянуто в контексті наявних алгоритмів генерації одноразових паролів, які реалізують її.

2.3.1 Алгоритм S/KEY

В 1981 році Леслі Лампортом була запропонована схема одноразових паролів, яка має на меті прибрати два шляхи через які зловмисник може дізнатися пароль користувача:

1. Отримавши доступ до інформації, що зберігається всередині системи, (наприклад, прочитавши файл паролів системи).

2. Шляхом перехоплення комунікації користувача з системою, наприклад, підслуховування лінії, що з’єднує термінал користувача з системою, або спостереження за виконанням програми перевірки пароля [18].

Для роботи алгоритму необхідна певна хеш-функція узгоджена з системою перевірки і з клієнтом — генератором одноразових паролів. Алгоритм працює наступним чином:

1. Обирається певне секретне значення, яке може бути згенеровано випадково, або введено користувачем.

2. Секретне значення послідовно хешується визначену кількість разів, наприклад 1000.

3. Останнє, тисячне, згенероване хеш значення надсилається системі автентифікації.

Після цього, при кожній автентифікації, системі надсилається секретне значення хешоване на один раз менше, ніж зберігається в системі. Тобто якщо система має 1000-е хеш значення, при першій автентифікації їй надсилається 999-е хеш значення. Система автентифікації може перевірити, що це дійсно попереднє значення застосувавши хешування ще один раз. Якщо отримане хеш значення збігається зі збереженим, автентифікація проходить успішно, а в системі зберігається нове попереднє хеш значення.

Алгоритм S/KEY це система одноразових паролів розроблена в Bellcore, яка працює подібно до схеми Лампорта. Стандарт наведено в відкритому доступі у вигляді RFC 1760. В якості хеш-функції використовується MD4.

Кожен згенерований одноразовий пароль має довжину 64 біта. Для зручності вводу, одноразовий пароль може бути перетворено в 6 коротких слів, обраних зі словника, що містить 2048 слів, отже кожне слово кодує 11 біт інформації. Для коректної роботи необхідно щоб сервер і клієнт мали однакові словники. В стандарті також наведено словник за замовчуванням, який містить англійські слова довжиною від одного до чотирьох символів [19].

Безпека алгоритму алгоритму базується на сукупності декількох властивостей. Основною властивістю є складність розрахування обернених хеш-функцій, оскільки всі одноразові паролі, що надсилаються через мережу і відповідно можуть бути перехоплені, є хеш значеннями розрахованими з наступного паролю, що буде використано для наступної автентифікації. Секретне значення і будь-який з невикористаних згенерованих паролів не мають бути розкритими, оскільки знаючи їх можна буде згенерувати всі чи деякі з одноразових паролів.

Серед інших проблем S/KEY є використання вразливого алгоритму хешування MD4 (хоча деякі реалізації використовують більш безпечні алгоритми, як програма skey в OpenBSD [20]) та відсутність солі при хешуванні. Ці фактори дозволяють розраховувати з меншою складністю обернені значення хеш-функції та використовувати райдужні таблиці.

Використання алгоритму S/KEY рідко зустрічається в сучасних системах. Одним з застосувань є використання в ОС заснованих на BSD та для автентифікації через SSH. Здебільшого непопулярність цього алгоритму спричинена частішим використанням інших алгоритмів генерації одноразових паролів.

2.3.2 Алгоритм HOTP

Алгоритм HOTP (HMAC-Based One-Time Password) був представлений в 2005 році в RFC 4226. HOTP презентується як простий алгоритм генерації одноразових паролів, що є легким в використанні кінцевими користувачами та який можливо реалізувати на дешевому апаратному забезпеченні, такому як Java смарт-карти або SIM карти [21].

В RFC 4226 наводиться список вимог з шести пунктів, яким мав відповідати алгоритм:

1. Алгоритм повинен базуватися на послідовності або лічильнику: одна з цілей полягає в тому, щоб алгоритм був вбудований у пристрої, такі як смарт-карти, USB ключі і SIM-карти.

2. Алгоритм має бути економним в реалізації на апаратному забезпеченні шляхом мінімізації вимог до акумулятора, кількості кнопок, обчислювальної потужності та розміру екрану.

3. Алгоритм повинен працювати з токенами, які не підтримують цифрове введення, але може також використовуватися з більш складними пристроями.

4. Значення, яке відображається на токені, повинно легко читатися та вводитися користувачем: для цього потрібно, щоб згенероване значення було прийнятної довжини. Довжина HOTP повинна бути хоча б 6 цифр. Також бажано щоб значення було лише цифровим, для легкого вводу, зокрема на телефонах.

5. Повинні бути зручні механізми, для ресинхронізації лічильника.

6. Алгоритм повинен використовувати надійне секретне значення. Довжина повинна бути хоча б 128 біт, рекомендовано — 160 біт.

Перед роботою алгоритму необхідно визначити декілька параметрів, які мають бути однаковими у клієнта та сервера: хеш-функцію (за замовчуванням — SHA-1), секретне значення, початкове значення лічильника, довжина згенерованого паролю (6-10 цифр, за замовчуванням — 6). Процес генерації одноразового паролю є достатньо простим (формула 2.1):

1. До поточного значення лічильника застосовується функція HMAC, що використовує попередньо визначену хеш-функцію. Як ключ вона приймає секретне значення.

2. Результат функції HMAC далі "обрізається" (Truncate) до 31-бітового значення. Детальніше, цей процес перетворення описано в специфікації.

3. На останньому етапі, з отриманого значення беруться останні 6-10 цифр (довжина одноразового паролю) в десятковому форматі.

$$HOTP(K, C) = Truncate(HMAC\text{-}SHA\text{-}1(K, C)) \quad (2.1)$$

де Truncate – функція обрізання хеш-суми;

K – секретний ключ;

C – лічильник.

Значення лічильника збільшується на один після кожної автентифікації. Сервер та клієнт ведуть окремі лічильники, які можуть десинхронізуватися. В випадку якщо клієнтський лічильник має більше значення ніж серверний, сервер може автоматично синхронізувати свій лічильник, без додаткових дій від користувача. Для цього, якщо отриманий сервером одноразовий код не збігається зі згенерованим на стороні сервера, сервер генерує коди для декількох наступних значень лічильника. Якщо один з кодів збігається, сервер перезаписує поточне значення лічильника на значення використане для генерації клієнтського коду, закінчуючи цим процес синхронізації. Якщо жодне значення не підходить, сервер знову запитує введення одноразового коду від клієнта.

Для забезпечення безпеки автентифікації за алгоритмом HOTP, важливі декілька параметрів:

- кількість спроб, для введення пароля;

- кількість наступних значень лічильника, що перевіряються при десинхронізації;
- кількість цифр, в одному паролі.

Якщо секретне значення не є скомпрометованим, найбільш ефективною атакою на HOTP є грубий перебір значень. Успіх такої атаки напряму залежить від вищенаведених параметрів.

2.3.3 Алгоритм TOTP

Алгоритм TOTP (Time-based One-Time Password) — ще один метод генерації одноразових паролів, який широко використовується в багатьох онлайн-системах для мультифакторної автентифікації. Специфікація була вперше опублікована в RFC 6238. Алгоритм презентується як розширення алгоритму HOTP, та генерує одноразові паролі, які є дійсними лише впродовж короткого часу [22].

TOTP майже повністю базується на алгоритмі HOTP, однак, на відміну від нього, алгоритм TOTP генерує одноразові паролі на основі поточного часу, а не за значенням лічильника. Для цього обидві сторони (сервер та клієнт) отримують поточний UNIX час. Після цього ця часова мітка ділиться на попередньо визначений часовий інтервал (за замовчуванням 30 секунд), отримуючи числове значення, яке використовується як лічильник в алгоритмі HOTP. Далі процес генерації продовжується за алгоритмом HOTP без змін (формула 2.2).

$$TOTP = HOTP(K, T) \tag{2.2}$$

де K – секретний ключ;

T – кількість часових інтервалів.

В RFC 6238 наводиться список вимог, яким мав відповідати алгоритм TOTP, серед яких:

1. Клієнтський пристрій та сервер повинні знати або мати змогу отримати поточний UNIX час (кількість секунд, що минули з 1 січня 1970 року за UTC) для генерації одноразового пароля.

2. Клієнт та сервер повинні володіти однаковим секретним значенням, або мати змогу його згенерувати.

3. Алгоритм HOTP повинен використовуватися як ключовий, при розробці алгоритму TOTP.

4. Клієнт та сервер повинні використовувати однаковий часовий інтервал.

Безпека алгоритму TOTP є здебільшого такою ж як і в алгоритму HOTP. Як було вже описано в попередньому розділі, найкращою атакою на цей алгоритм є грубий перебір. Проте TOTP додатково ускладнює такий перебір часовим фактором. Постійна зміна одноразових паролів, без додаткових дій від користувача, зменшує можливість зловмисника вгадати пароль в порівнянні з алгоритмом HOTP, в якому паролі змінюються лише після успішної автентифікації. Недовга валідність одноразових паролів також ускладнює підглядання або перехоплення паролів.

Алгоритм TOTP досить широко використовується в інтернет сервісах для двофакторної автентифікації. Здебільшого, якщо вебсайт пропонує додати двофакторний автентифікатор, мається на увазі саме цей алгоритм. Існує велика кількість безкоштовних програм, які дозволяють зручно додавати секретні значення за QR-кодами, зберігати їх і генерувати одноразові паролі. Серед найбільш використовуваних: Google Authenticator, Microsoft Authenticator і Twilio Authy Authenticator.

2.3.4 Алгоритм OCRA

В 2011 році була опублікована специфікація RFC 6287, що описує алгоритм автентифікації на основі запит-відповідь (challenge-response). Алгоритм дістав назву OCRA (OATH Challenge-Response Algorithm). Цей алгоритм також заснований на алгоритмі HOTP і надає можливості для односторонньої та взаємної автентифікації, а також створення електронних підписів [23].

В RFC 6287 описані основні вимоги, які ставилися перед розробкою алгоритму. Велика увага приділяється зручності використання враховуючи специфіки алгоритму HOTP і апаратних можливостей:

1. Алгоритм повинен підтримувати автентифікацію на основі запиту-відповіді.

2. Алгоритм повинен підтримувати електронні підписи на основі симетричного ключа. По суті, це різновид запиту-відповіді, де запит виводиться з даних, які потрібно підписати.

3. Алгоритм повинен підтримувати автентифікацію сервера, за допомогою якої користувач може перевірити, що комунікація відбувається зі справжнім сервером.

4. Алгоритм HOTP повинен використовуватися як ключовий при розробці.

5. Довжина та формат вхідного запиту повинні бути налаштовуваними.

6. Довжина та формат згенерованої відповіді повинні бути налаштовуваними.

7. Запит може бути згенеровано за допомогою перевірки цілісності. Це дозволить токенам виконувати просту перевірку помилок, під час вводу користувачем значення запиту.

8. Унікальне секретне значення повинне існувати для кожного токена, яке спільно використовується токеном і сервером автентифікації. Ключі повинні бути згенеровані випадковим чином або отримані за допомогою алгоритму виведення ключів.

9. Алгоритм може включати в обчислення додаткові атрибути, такі як часову мітку або інформацію про сесію. Ці дані можна використовувати індивідуально або всі разом.

Одностороння автентифікація за алгоритмом OCRA здійснюється за наступним алгоритмом. Сервер надсилає запит клієнту. Клієнт генерує відповідь, та надсилає її серверу. Сервер перевіряє відповідь, після чого надсилає клієнту "OK", якщо перевірка пройшла успішно або "NOK", якщо відповідь неправильна.

Взаємна автентифікація вимагає, щоб клієнт спочатку надіслав свій запит серверу. Сервер при отриманні запиту, генерує відповідь та надсилає її разом зі своїм запитом клієнту. Клієнт перевіряє відповідь сервера і, лише у випадку якщо вона є

правильною, надсилає свою відповідь. Сервер перевіряє відповідь і у разі успіху надсилає "OK" клієнту. На цьому етапі взаємна автентифікація закінчується.

2.4 Програмна реалізація алгоритмів HOTP та TOTP мовою програмування Rust

В цьому розділі представлено розроблений прототип додатку, що реалізує алгоритми генерації одноразових паролів TOTP і, як наслідок, HOTP. Мова програмування Rust була обрана через її зручність, для розробки таких математичних алгоритмів, автоматичне керування пам'яттю та можливість використання бібліотек розроблених іншими користувачами.

2.4.1 Реалізація алгоритму HOTP

Створення реалізації було почато з написання функції, для генерації хеш-суми за алгоритмом HMAC-SHA1 з наданого секретного ключа та лічильника у вигляді масивів байтів. Для цього було використано відкриту бібліотеку *ring* створену Браяном Смітом [24]. На рисунку 2.1 показано етапи створення ключа та генерації хеш-суми (Tag).

```
fn hmac_sha1(key: &[u8], counter: &[u8]) -> Tag {
    let hmac_key = Key::new(HMAC_SHA1_FOR_LEGACY_USE_ONLY, key);
    let tag = hmac::sign(&hmac_key, counter);
    return tag;
}
```

Рисунок 2.1 — Вихідний код функції генерації хеш-суми HMAC-SHA1

Далі необхідно створити функцію Truncate, що має скорочувати попередньо утворений хеш до 6-10 десяткових цифр, які задаються змінною digits. На рисунку 2.2 представлено вихідний код, для виконання цієї задачі.

```rust
fn truncate_hmac(hmac_tag: &[u8], digits: u32) -> u32 {
    assert_eq!(hmac_tag.len(), 20);
    assert!((6..=10).contains(&digits));
    let offset = (hmac_tag[19] & 0xf) as usize;
    let code = (hmac_tag[offset] as u32 & 0x7f) << 24
        | (hmac_tag[offset + 1] as u32 & 0xff) << 16
        | (hmac_tag[offset + 2] as u32 & 0xff) << 8
        | (hmac_tag[offset + 3] as u32 & 0xff);
    return code % 10u32.pow(digits);
}
```

Рисунок 2.2 — Вихідний код функції Truncate

Таким чином було створено всі необхідні частини, для роботи алгоритму HOTP. Використання цих функцій відповідає формулі 2.1.

2.4.2 Реалізація алгоритму TOTP

Оскільки основна частина алгоритму TOTP використовує HOTP, необхідно було лише виконати розрахування кількості часових інтервалів з заданої мітки часу (рис. 2.3).

```rust
fn totp(key: &[u8], time_s: u64) -> u32 {
    let counter = time_s / 30;
    let hmac_result = hmac_sha1(key, counter.to_be_bytes().as_ref());
    return truncate_hmac(hmac_result.as_ref(), 6);
}
```

Рисунок 2.3 — Вихідний код алгоритму TOTP

Використання чи адаптування цих функцій може бути виконане таким шляхом, який підходить до системи автентифікації. Для даного прототипу було розроблено мінімальний користувацький інтерфейс, для швидкого і простого генерування одноразових кодів, зневажаючи деякі аспекти безпеки.

На рисунку 2.4 представлено вихідний код консольного інтерфейсу. Після запуску програма просить користувача ввести секретний ключ в кодуванні Base32. Далі отримується поточний UNIX час в секундах, який додатково виводиться на екран, та генерується одноразовий TOTP код. Приклад взаємодії з даним додатком представлено на рисунку 2.5.

```rust
fn main() {
    let mut user_code = String::new();
    println!("Enter secret key:");
    std::io::stdin().read_line(&mut user_code).expect("Read error");
    let user_key = base32::decode(base32::Alphabet::RFC4648 { padding: false }, &user_code.trim()).unwrap();

    let unix_time = SystemTime::now().duration_since(SystemTime::UNIX_EPOCH).unwrap();
    let unix_time_seconds = unix_time.as_secs();
    println!("Unix time in seconds: {unix_time_seconds}");
    println!("TOTP code: {}", totp(&user_key, unix_time_seconds));
}
```

Рисунок 2.4 — Вихідний код функції main з консольним інтерфейсом

```
$ ./totp.exe
Enter secret key:
ABCDE
Unix time in seconds: 1679829150
TOTP code: 407370
```

Рисунок 2.5 — Запуск програми та генерація коду для секретного ключа ABCDE в Base32 форматі

Повний файл програми та тести до алгоритмів наведені в додатку А.

2.5 Вимоги до системи автентифікації

Будь-які криптографічні алгоритми, з точки зору безпеки, мають сенс лише, якщо система та протоколи які використовуються також безпечні. Для правильного використання алгоритмів HOTP і TOTP необхідно визначити деякі вимоги (які можуть застосовуватися і до деяких інших алгоритмів генерації одноразових паролів), яким має відповідати система автентифікації.

Найперше, що варто розглянути, це мінімізація можливості перебору одноразових паролів. Для алгоритму HOTP є декілька параметрів, що прямо впливають на шанс вгадати зловмисником правильний пароль (формула 2.3).

$$Sec = sv/10^d \tag{2.3}$$

де Sec – шанс правильно вгадати одноразовий пароль;

s – кількість наступних паролів (вікно), що перевіряються при десинхронізації;

v – кількість спроб введення пароля;

d – кількість цифр в одноразовому паролі.

Збільшення параметрів s і v може покращити користування сервісом, але водночас дасть зловмиснику більше шансів вгадати пароль. Конфігурування даних параметрів на доцільному рівні є необхідним для безпеки. Після закінчення кількості спроб введення паролю, потрібно вжити певних заходів, для запобігання подальшим спробам введення одноразових паролів. Наприклад, акаунт користувача може бути заблоковано, а самого користувача про це проінформовано. Також можливе застосування блокувань вводу на деякий час після неправильної спроби. Час блокування має зростати після кожної неправильної спроби і повинен застосовуватися до всіх сесій, що намагаються автентифікуватися.

Система автентифікації, що використовує HOTP, повинна використовувати мультифакторну автентифікацію. Таким чином, алгоритм HOTP необхідно використовувати, як додатковий фактор автентифікації, а не як єдиний фактор. Комунікація між сервером та користувачем повинна виконуватися через захищений канал, для забезпечення конфіденційності та запобігання певним атакам. Секретні ключі, що генеруються сервером, повинні надійно зберігатися до закінчення їхньої дії.

2.6 Вимоги до користувачів системи автентифікації

Користувачі сервісу, що використовує одноразові паролі як один з факторів автентифікації, також повинні притримуватися певних вимог та рекомендацій, для забезпечення компрометації їхнього облікового запису. Зважаючи на те, що втрата одним користувачем свого облікового запису не вплине на безпеку системи автентифікації, безпека свого облікового запису, здебільшого, є інтересом самого користувача. Тим не менш, для будь-якого сервісу, безпека окремого акаунта стає проблемою, якщо акаунт має певні адміністративні повноваження і може вносити зміни в важливі компоненти системи.

Користувачі повинні надійно зберігати секретний ключ, що отримується при увімкненні автентифікації за одноразовими паролями та застосовується для їх генерації. Якщо користувач втратить даний ключ, він також втратить можливість

автентифікації за даним фактором, що може означати втрату доступу до облікового запису. Додатково, рекомендується зберігати секретні ключі в зашифрованому вигляді, якщо вони зберігаються в електронному форматі. В такому випадку, шкідливе програмне забезпечення, яке може отримати повний доступ до комп'ютера, не зможе дізнатися секретний ключ. Проте, шифрування все ще не захистить від можливого видалення ключа.

У випадку, якщо для генерації одноразових паролів використовуються фізичні токени, користувачі, відповідно, повинні дбати про їх зберігання та запобігати їх втраті чи поламці.

Користувачі не повинні розсекречувати свій секретний ключ або невикористані одноразові паролі третім сторонам. Ця вимога також включає запобігання підгляданням при прогляданні або вводі одноразових паролів. Користувачі мають дбати про те, щоб не стати жертвою фішингу.

Також не варто забувати про інші фактори автентифікації, що використовуються разом з одноразовими паролями. Користувачі не повинні зневажати безпекою інших факторів. Використання мультифакторної автентифікації, не означає, що можна розкривати наприклад свій постійний пароль, тримаючи в секреті свої одноразові паролі і вважаючи, що зловмисник не зможе їх зламати. Додаткові фактори мають посилювати безпеку всього процесу автентифікації, а не заміняти собою інші фактори.

Висновки за розділом 2

Одноразові паролі вирішують деякі проблеми, присутні в постійних паролях. Наприклад повторне використання пароля зловмисником не є можливим, а випадкова або математична (псевдовипадкова) генерація паролів ускладнює їх перебір за словниками.

Одноразові паролі можливо створювати або випадково, з подальшим надсиланням їх користувачеві, або за допомогою математичних алгоритмів, що є більш цікавим способом. Розглянуті алгоритми математичної генерації одноразових

паролів включають S/KEY, HOTP, TOTP та OCRA. Вони працюють за різними схемами генерації і мають різні сильні сторони та недоліки.

Розроблена програмна реалізація алгоритмів HOTP і TOTP показує, що дані алгоритми є достатньо простими в роботі та реалізації використовуючи будь-яку мову програмування.

Система автентифікації, що використовує одноразові паролі повинна притримуватися певних вимог, для запобіганню атакам. Серед деяких параметрів, що прямо стосуються одноразових паролів, важливі декілька: кількість цифр в одноразовому паролі, кількість спроб введення пароля та кількість наступних паролів що перевіряють при десинхронізації. Система автентифікації і користувач повинні надійно зберігати свої секретні ключі та запобігати їх компрометації.

РОЗДІЛ 3
РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ КЕРУВАННЯ ОДНОРАЗОВИМИ ПАРОЛЯМИ

3.1 Визначення функціоналу та вимог до програмного засобу

Розробка це складний процес, що має враховувати безліч деталей та нюансів впродовж всього процесу розробки, особливо з точки зору безпеки, оскільки програмний засіб матиме доступ до секретних ключів користувача. Перед початком розробки повноцінного засобу, для керування одноразовими паролями користувачів, необхідно визначити основні вимоги до його функцій, а також продумати яким чином досягатиметься захист секретних ключів.

В якості необхідного функціоналу програми було визначено наступний:

1. додавання та видалення одноразових паролів;
2. редагування інформації про одноразові паролі;
3. автоматичне оновлення паролів з часом;
4. можливість скопіювати поточний одноразовий пароль до буферу обміну.

Наведених функцій буде достатньо для мінімально зручного користування програмним засобом. Додатково можна розробити сканування QR-кодів, за допомогою яких розповсюджуються секретні ключі в більшості випадків, і експортування та імпортування всіх ключів, що зберігаються. Проте, виконання даних функцій вимагатиме отримання дозволів на використання камери та доступу до пам'яті користувача.

Для захисту секретних ключів та самих одноразових паролів від шкідливого програмного забезпечення, необхідно також визначити деякі вимоги. Досягти повного захисту не є можливим, проте все ще можна посилити захист відомими способами та засобами. До функцій безпеки належать:

1. зберігання секретних ключів в області пам'яті з обмеженим доступом;
2. шифрування секретних ключів;

3. відсутність зв'язку з інтернетом та дозволів для цього.

3.2 Вибір цільової платформи, мови програмування та фреймворків

Є дві основні платформи, для яких можна розробляти програмне забезпечення: комп'ютери та смартфони. Комп'ютери та ноутбуки, в порівнянні зі смартфонами, є менш мобільними та зазвичай залишаються в одному місці. Зберігання одноразових паролів лише на даних пристроях було б незручним. На противагу комп'ютерам, смартфони постійно увімкнені та залишаються поряд з користувачем де б він не був. Тому зважаючи на переваги що надають мобільні пристрої, в якості апаратних засобів, для яких буде розроблятися програмний засіб, було обрано смартфони.

Існують дві операційні системи на яких працює переважна більшість мобільних телефонів: Android та IOS. Оскільки розробка для ОС IOS можлива лише на комп'ютерах під управлінням macOS, а для ОС Android — на всіх трьох основних комп'ютерних ОС (Windows, macOS, Linux), як цільову систему для програмного засобу було обрано ОС Android.

Основними підтримуваними мовами для розробки для Android Framework є Java та Kotlin. Існують також інші фреймворки, такі як Flutter, React Native, Kotlin Multiplatform Mobile, для розробки мобільних додатків як для Android так і для IOS. Проте вони можуть не надавати повний доступ до API, які надає кожна система. Тому вибір зупинився на мові Kotlin.

Розробка на мовах Java та Kotlin також може доповнюватися бібліотеками написаними мовами C, C++, або іншими мовами, які можуть бути скомпільовані в бінарний код для мобільних архітектур. Це надає можливість в подальшому замінити частини коду, що виконують складні розрахунки (наприклад криптографічні перетворення), на більш швидкі бібліотеки написані на низькорівневих мовах.

3.3 Опис зберігання та шифрування секретних ключів

Генерація одноразових TOTP паролів неможлива без використання спільного секретного ключа сервером та клієнтом. Коли користувач вмикає автентифікацію за одноразовими паролями, сервер надає йому секретний ключ, який користувач має надійно зберігати, не розкриваючи його стороннім особам. Оскільки один менеджер одноразових паролів може використовуватися для безлічі акаунтів, всі секретні ключі мають надійно зберігатися цим програмним менеджером. Постає проблема способу, яким буде забезпечуватися захист та збереження ключів.

Оскільки кожен секретний ключ і серія одноразових паролів, що він генерує, належить чітко визначеному акаунту, має свою назву та набір параметрів для генерації, вони можуть зберігатися у вигляді таблиці. Система Android надає вбудовані функції для роботи з SQLite таблицями. Додатково, існують офіційні бібліотеки розроблені Google, для спрощеної роботи з базами даних, одною з яких є Room [25].

Для кожної встановленої Android програми, система створює окрему директорію, доступ до якої має лише дана програма. Ні користувач, ні інші програми не можуть переглядати, читати чи записувати дані, які зберігаються в цих каталогах. Записуючи секретні ключі в базу даних, яка зберігається у власній директорії програми, виконується вимога, щодо обмеження доступу до ключів, тим самим посилюючи безпеку.

Проте, у випадку якщо зловмисник все таки зможе отримати доступ до внутрішньої пам'яті, зберігати ключі в відкритому доступі не є раціональним захистом. Шифрування секретних ключів вирішує цю проблему. Android API має підтримку багатьох алгоритмів шифрування, а починаючи з версії 10 було додано алгоритм AES в режимі GCM (Galois/Counter Mode) [26]. Перевага GCM над іншими режимами блокового шифрування полягає в тому, що GCM виконує автентифіковане шифрування, чим забезпечує цілісність даних разом з конфіденційністю.

Наступна проблема — це зберігання ключа який використовується для шифрування даних. Адже якщо ключ шифрування можна буде отримати порівняно

простіше, ніж намагатися дешифрувати секретні ключі від одноразових паролів, тоді шифрування не матиме сенсу. Є два способи, якими можливо забезпечити безпечне зберігання ключа шифрування:

1. Запитувати ключ шифрування в користувача, при кожному запуску програмного засобу.

2. Використовувати Android Keystore — системне API, для зберігання і захисту ключів.

В першому випадку, відповідальність за збереження ключа повністю переходить на користувача, від програмного засобу вимагається лише недопущення витоку або зберігання введеного ключа в постійній пам'яті. В залежності від користувача, такий спосіб може спрацювати або якнайкраще, або гірше ніж альтернативи. Перевага даного способу полягає в неіснуванні ключа шифрування в пам'яті телефону і неможливість його отримати будь-якими іншими програмами, крім випадку, якщо до оперативної пам'яті пристрою буде отримано повний доступ в той час як вона зберігає ключ шифрування.

В другому випадку, програмний засіб самостійно створює ключ шифрування і зберігає його в контейнері Android Keystore. Система Android Keystore захищає ключі від несанкціонованого використання різними способами. Вона зменшує ризик несанкціонованого використання ключів ззовні пристрою, запобігаючи отриманню ключів з процесів програми та з пристрою загалом. Ключі не покидають системний процес, який виконує всі необхідні криптографічні операції, що вимагає програма з цим ключем. Система вимагає від програм вказувати призначення ключів і забороняє їх використання в інших сценаріях. Якщо Android пристрій підтримує Trusted Execution Environment, ключі можуть зберігатися на цьому безпечному обладнанні, яке вони не покидають. Таким чином, навіть при компрометації Android системи, зловмисник зможе максимум використовувати ключі на даному пристрої, проте не зможе експортувати їх [27].

3.4 Реалізація зберігання секретних ключів та алгоритму TOTP мовою програмування Kotlin

Для зберігання секретних ключів до кожного одноразового пароля, насамперед необхідно визначити які саме дані будуть асоційовані з кожним об'єктом. Кожен одноразовий пароль має хоча б два поля: назву та секретний ключ. Для шифрування, більшість режимів блокового шифрування вимагає мати вектор ініціалізації (IV), який, в цілях безпеки, необхідно мати різний для кожного окремого шифрування. Додатково, задля забезпечення можливості редагування даних, кожній структурі необхідно присвоїти унікальний ідентифікатор, який залишатиметься незмінним впродовж всього існування одноразового пароля. Створену структуру для зберігання цих даних представлено на рисунку 3.1.

```
data class EncryptedTotpKey(
    val id: Int,
    val name: String,
    val secret: ByteArray,
    val iv: ByteArray,
) {
```

Рисунок 3.1 — Вихідний код структури для зберігання секретних ключів

Створення ключа шифрування використовуючи Android Keystore, що показано на рисунку 3.2, вимагає певних налаштувань та надає можливість визначити обмеження, щодо використання ключа. Найперше що вказується — це призначення ключа для симетричного шифрування або дешифрування. Далі вказуються налаштування для алгоритму AES в режимі GCM, такі як довжина ключа в 256 біт і відсутність доповнення (padding). Встановлення setRandomizedEncryptionRequired в значення false означає, що при шифруванні буде використовуватися IV наданий програмою, а не системою Android.

Алгоритм додавання нового секретного ключа до бази даних (рисунок 3.3) створює випадковий IV, шифрує секретний ключ (plainSecret) і додає його до бази даних (repository).

```
override fun generateRandomKey(alias: String): SecretKey {
    val keyGenerator = KeyGenerator.getInstance("AES", "AndroidKeyStore")
    keyGenerator.init(
        KeyGenParameterSpec.Builder(
            alias,
            KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT
        )
            .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
            .setKeySize(256)
            // false to use custom IV
            .setRandomizedEncryptionRequired(false)
            .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
            .build()
    )
    return keyGenerator.generateKey()
}
```

Рисунок 3.2 — Функція для створення ключа шифрування в Android Keystore

```
val random = SecureRandom()
val iv = ByteArray(encryptor.ivSize)
random.nextBytes(iv)
repository.addKey(EncryptedTotpKey(0, name, encryptor.encrypt(plainSecret, iv), iv))
```

Рисунок 3.3 — Код функції, що додає новий секретний ключ до сховища

Оскільки програмний засіб розробляється лише для генерації одноразових паролів за алгоритмом TOTP, відокремлювати алгоритм HOTP в окрему функцію не є необхідним. Імплементацію хеш-функції HMAC було взято з бібліотеки Apache Commons Codec [28], що розповсюджується за ліцензією Apache License 2.0.

Програмна реалізація алгоритму TOTP мовою Kotlin, мало чим відрізняється від реалізації на мові Rust. Тим не менш, деякі відмінності в синтаксисі та стандартній бібліотеці є присутніми. Наприклад, стандартна бібліотека Kotlin не має функції для розбиття цілого числа на масив байтів, тому дану функцію, toBigEndianBytes, довелось створювати додатково. Програмна реалізація на момент написання, використовує незмінюваний часовий інтервал в 30 секунд і алгоритм хешування HMAC на базі SHA-1 (рисунок 3.4).

Вихідний код файлів, що містять розглянутий в цьому підрозділі код наведено в додатку Б.

```kotlin
override fun generate(secret: ByteArray, unixTime: Duration): Int {
    val seconds = toBigEndianBytes(unixTime.inWholeSeconds / 30)
    val hmac = HmacUtils(HmacAlgorithms.HMAC_SHA_1, secret).hmac(seconds)
    return truncate(hmac)
}

private fun toBigEndianBytes(long: Long): ByteArray {
    return ByteArray(8) { i -> (long ushr (8 * (7 - i))).toByte() }
}

private fun truncate(hmac: ByteArray, digits: Int = 6): Int {
    val offset = (hmac.last() and 0xf).toInt()
    val code = (hmac[offset].toUInt() and 0x7fu shl 24) or
            (hmac[offset + 1].toUInt() and 0xffu shl 16) or
            (hmac[offset + 2].toUInt() and 0xffu shl 8) or
            (hmac[offset + 3].toUInt() and 0xffu)
    return code.toInt() % (10f.pow(digits).toInt())
}
```

Рисунок 3.4 — Вихідний код реалізації алгоритму TOTP мовою Kotlin

3.5 Кінцевий результат

Після закінчення розробки графічного інтерфейсу та пов'язаних внутрішніх компонентів, роботу над програмним засобом було завершено. Вимоги щодо функціоналу, описані у підрозділі 3.1, були повністю виконані. Програмний засіб дозволяє додавати, видаляти та редагувати одноразові паролі. Автоматичне оновлення відбувається якнайшвидше після проходження інтервалу часу в 30 секунд.

На рисунку 3.5 представлено графічний інтерфейс розробленого засобу. Візуально, інтерфейс є досить простим. Натискання на бірюзову кнопку "плюс" відкриває віконце додавання нового секретного ключа. Довге натискання на кожен одноразовий пароль дозволяє редагувати або видалити секретний ключ чи назву цього пароля. Скопіювати поточний одноразовий пароль можливо за допомогою відповідної іконки з правого боку.

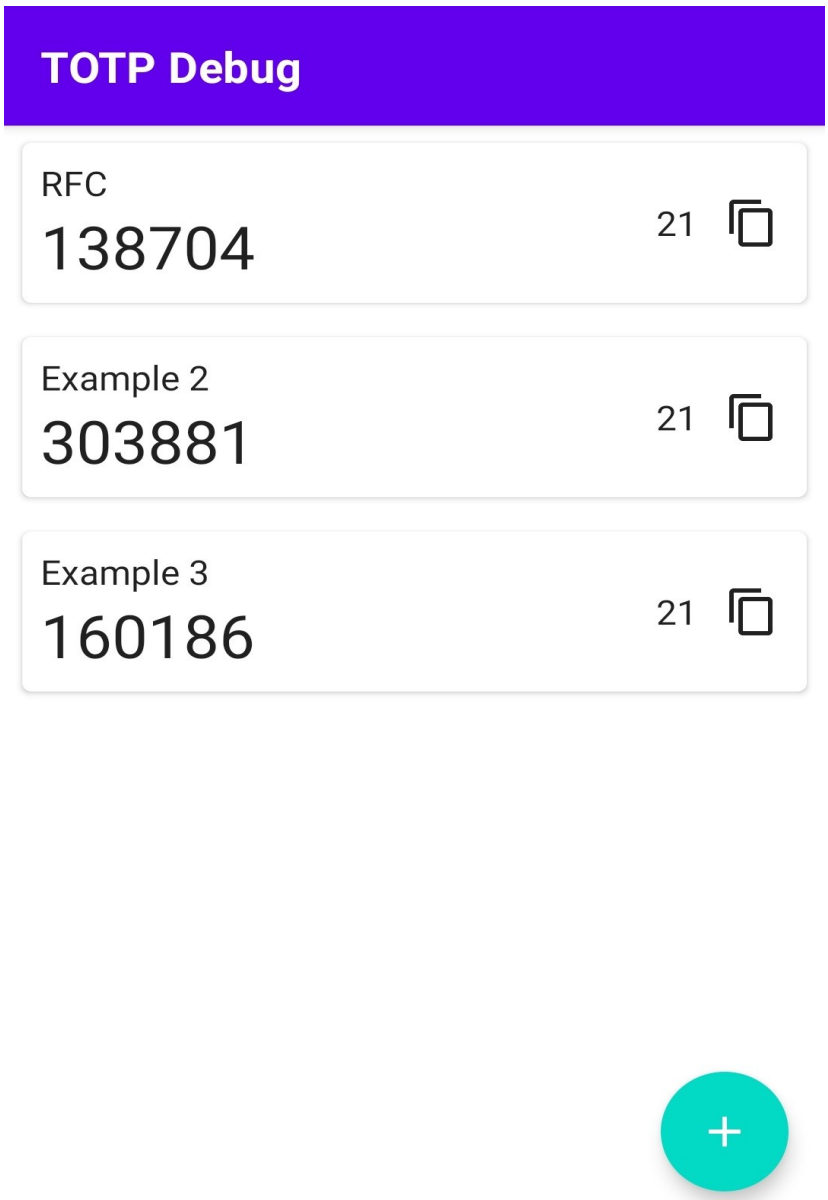

Рисунок 3.5 — Інтерфейс розробленого програмного засобу

3.6 Безпека та вразливості програмного засобу

Основним об'єктом який необхідно захищати в розробленому програмному засобі є секретні ключі від одноразових паролів. В цілях захисту, перед збереженням кожного секретного ключа, вони зашифровуються алгоритмом автентифікованого шифрування, який гарантує цілісність та конфіденційність даних. Для захисту ключа шифрування, використовується системні бібліотеки Android Keystore, які забороняють доступ до ключа всім (програмам та користувачам), крім додатка, який його створив.

Для запобігання неавторизованого доступу до бази даних, в якій зберігаються секретні ключі, база даних зберігається в особистій теці програми. Доступ до цієї теки забороняється іншим програмам та всім звичайним користувачам.

Варто зазначити, що у разі отримання прав root користувача сторонньою програмою або користувачем, доступ до бази даних може бути отримано, відкрита інформація в ній — прочитана і вся інформація в базі — змінена або видалена. Проте дозволу на доступ до ключів шифрування, що зберігаються в Android Keystore, навіть root користувачем все ще отримати неможливо.

Доступ до ключів шифрування в Keystore отримується програмою без додаткових умов. Це дає змогу зловмиснику, який отримає фізичний доступ до розблокованого смартфона, прочитати і розшифрувати секретні ключі, що зберігаються на той момент в базі даних. Таким чином, єдиний захист при фізичному доступі до системи, але не повному доступі до пам'яті пристрою, це екран блокування.

Програмний засіб не отримує жодних дозволів від системи Android, тобто програмному засобі відмовлено в доступі до: камери, інтернету та приватних файлів користувача. Це надає користувачу інформацію навіть без переглядання вихідного коду, що програма не може виконувати будь-які дії які б вимагали додаткових дозволів. На додаток, відсутність використання камери або інтернету, протидіє потенційним експлойтам, які могли б отримати доступ до даних програмного засобу або виконання коду від імені програми.

3.7 Порівняння розробленого засобу з аналогами інших розробників

В інтернеті існує безліч програм, що також надають можливості генерації одноразових паролів. Оскільки багато таких засобів існували та використовувались впродовж останніх декількох років, вони є більш розвиненими ніж розроблений засіб. Порівнявши їхній функціонал до функціоналу розробленого засобу, можна визначити які функції є відсутніми на зараз, але їх варто було б додати до

розробленого програмного засобу, для покращення або полегшення використання засобу користувачами.

Для порівняння було обрано два популярних засоби генерації одноразових паролів: Google Authenticator і LastPass Authenticator.

Google Authenticator є одним з найпопулярніших автентифікаторів, з більш ніж ста мільйонами завантажень. Випущено в березні 2012 року. Даний програмний засіб найчастіше рекламується самою компанією Google для використання двофакторної автентифікації в акаунтах Google, а також багатьма іншими інтернет сервісами, які рекомендують Google Authenticator, для своєї двофакторної автентифікації.

LastPass Authenticator — менш популярний автентифікатор з понад одним мільйоном завантажень. Розроблено як додаток до менеджеру паролів LastPass і в основному позиціонується для використання з акаунтами LastPass. Тим не менш засіб не обмежується лише акаунтами LastPass і підтримує додавання одноразових паролів від будь-яких акаунтів.

Після проведення аналізу цих програмних засобів, було виокремлено декілька категорій щодо особливостей та можливостей які мають ці засоби. Ці дані представлено в таблиці 3.1.

Таблиця 3.1

	Власна розробка	Google Authenticator	LastPass Authenticator
Підтримка OTP алгоритмів	TOTP	TOTP, HOTP	TOTP
Додаткові налаштування генерації	Лише стандартні	Лише стандартні	Кількість цифр в паролі, алгоритм хешування, інтервал часу
Синхронізація, експорт чи збереження паролів	Ручний експорт або збереження	Експорт за QR-кодом	Синхронізація і збереження в акаунті LastPass, експорт за QR-кодом або файлом
Відкритий код	Так	Лише до версії 5.0	Ні
Блокування програми	Ні	Ні	Так

З таблиці можна визначити, що LastPass Authenticator підтримує найбільшу кількість налаштувань та функцій з-поміж розглянутих програмних засобів. Проте, цей засіб є єдиним, що не має хоч якогось відкритого коду. Google Authenticator має досить малу кількість налаштувань, але має підтримку алгоритму HOTP.

3.8 Можливості подальшого вдосконалення програмного засобу

Порівнявши можливості інших додатків, можна визначити певні покращення для розробленого засобу керування одноразовими паролями:

1. Підтримка налаштувань генерації одноразових паролів. Кожен окремий одноразовий пароль може генеруватися за різний час, з різною кількістю цифр в ньому або за іншим алгоритмом хешування. Хоча більшість інтернет сервісів використовують стандартні налаштування, якщо не буде можливості їх змінювати, жоден сервіс не буде використовувати інші налаштування.

2. Можливість експорту та імпорту одноразових паролів. На випадок виходу з ладу мобільного пристрою, важливо зберігати копії всіх секретних ключів для їх відновлення. Також, ці функції можливо використовувати для передачі ключів на інший смартфон. Для посилення безпеки, варто мати опцію збереження секретних ключів у зашифрованому вигляді.

3. Можливість блокування програмного засобу за паролем або біометричною автентифікацією. Наявність додаткового захисту збереже конфіденційність одноразових паролів у випадку фізичного доступу до смартфона.

Також можливо розширити підтримку операційних систем та платформ шляхом адаптування вихідного коду програмного засобу для цільових систем. Маючи програмний засіб для системи Android, можливо з помірними зусиллями адаптувати код для підтримки системи IOS, що дозволить покрити більшість мобільних пристроїв.

Для подальшого розвитку засобу керування одноразовими паролями, можлива розробка окремого апаратного засобу, що працював би з власним акумулятором незалежно від смартфона чи комп'ютера. Такий пристрій повинен мати достатньо

низьке електроспоживання та реалізовувати зберігання секретних ключів без компрометації їх безпеки, у разі викрадення пристрою.

Висновки за розділом 3

Перед розробкою програмного засобу керування одноразовими паролями, необхідно було визначити вимоги щодо функціоналу засобу та безпеки зберігання даних про одноразові паролі. Після виконання цього завдання, постає проблема платформи для якої буде створюватися програмний засіб, оскільки кожна система має різні використання, особливості та програмні бібліотеки.

Як платформу для програмного засобу було обрано смартфони на базі операційної системи Android. Для ефективного та безпечного зберігання ключів шифрування, якими будуть шифруватися секретні ключі для TOTP алгоритму, Android фреймворк надає бібліотеку Android Keystore. Після продумування алгоритму зберігання даних та написання необхідних функцій, було створено простий графічний інтерфейс. Протестувавши реалізовані функції, роботу над програмним засобом було завершено.

В кінцевому результаті, програмний засіб дозволяє додавати секретні ключі для одноразових паролів, редагувати та видаляти їх. Згенеровані паролі можна легко скопіювати а їх оновлення з часом відбувається автоматично. Здебільшого, цих функцій є достатньо для мінімального користування програмним засобом, але порівняння з іншими програмами що виконують схожі функції, показує, яких функцій не вистачає для покращення чи полегшення користування розробленим засобом.

ВИСНОВКИ

Безпечна автентифікація в інформаційних системах є актуальною проблемою, що існує з часів створення перших комп'ютерних систем. Численні дослідження та технологічний прогрес породили багато рішень з безпечної автентифікації користувачів і цей процес продовжується й до сьогодні. Одним з розповсюджених способів автентифікації є двофакторна автентифікація з використанням одноразових паролів.

В кваліфікаційній роботі було розглянуто деякі з відкритих алгоритмів генерації одноразових паролів, а саме S/KEY, HOTP, TOTP та OCRA, кожен з яких використовує різні схеми для генерації. Для алгоритму TOTP було розроблено мінімальну програмну реалізацію, для демонстрації роботи алгоритму. Також було розглянуто, як різні параметри алгоритму TOTP впливають на складність зламу одноразового пароля.

В результаті проведених досліджень, було розроблено програмний засіб, для керування одноразовими паролями на базі алгоритму TOTP. Розроблений засіб є зручним в користуванні і дозволяє безпечно зберігати секретні ключі в зашифрованому вигляді, автоматично генерувати та оновлювати одноразові паролі.

В результаті виконання кваліфікаційної роботи були виконані завдання, наведені в меті роботи:

- проведено аналітичний огляд наявних практик щодо використання факторів автентифікації;
- проведено аналіз алгоритмів та стандартів генерації одноразових паролів;
- визначено необхідні вимоги та параметри, для забезпечення захищеності роботи алгоритму TOTP;
- розроблено мінімальну реалізацію алгоритму TOTP мовою програмування Rust;

- розроблено захищений додаток для ОС Android, для керування та збереження секретних значень і автоматичної генерації одноразових паролів на базі алгоритму TOTP;
- проведено порівняння розробленого програмного засобу з двома існуючими рішеннями, що виконують схожі функції.

Всі поставлені задачі було виконано в повному обсязі. Мету роботи було досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Правила забезпечення захисту інформації в інформаційних, телекомунікаційних та інформаційно-телекомунікаційних системах [Електронний ресурс]: постанова КМУ від 29.03.2006 р. № 373 – Режим доступу: https://zakon.rada.gov.ua/laws/show/373-2006-п

2. The World's First Computer Password? It Was Useless Too [Електронний ресурс]: Wired – Режим доступу: https://www.wired.com/2012/01/computer-password/

3. Morris, R. and Thompson, K. Password security: A case history [Електронний ресурс]: Bell Laboratories. – Режим доступу: https://wolfram.schneider.org/bsd/7thEdManVol2/password/password.pdf

4. Ellis, J. THE POSSIBILITY OF SECURE NON-SECRET DIGITAL ENCRYPTION [Електронний ресурс]. – Режим доступу: https://cryptocellar.org/cesg/possnse.pdf

5. Dr Clifford Cocks CB [Електронний ресурс]: University of Bristol – Режим доступу: http://www.bristol.ac.uk/graduation/honorary-degrees/hondeg08/cocks.html

6. A Method for Obtaining Digital Signatures and PublicKey Cryptosystems [Електронний ресурс]: Communications of the ACM – Режим доступу: http://people.csail.mit.edu/rivest/pubs.html#RSA78

7. Diffie, W. and Hellman, M. New Directions in Cryptography [Електронний ресурс]: IEEE TRANSACTIONS ON INFORMATION THEORY – Режим доступу: https://ee.stanford.edu/~hellman/publications/24.pdf

8. State of the Auth. Experiences and Perceptions of Multi-Factor Authentication [Електронний ресурс]: Duo Labs Report – Режим доступу: https://duo.com/assets/ebooks/state-of-the-auth-2019.pdf

9. New research: How effective is basic account hygiene at preventing hijacking [Електронний ресурс]: Google Security Blog – Режим доступу: https://security.googleblog.com/2019/05/new-research-how-effective-is-basic.html

10. Digital Identity Guidelines [Електронний ресурс]: NIST Special Publication 800-63-3 – Режим доступу: https://csrc.nist.gov/publications/detail/sp/800-63/3/final

11. Data Breaches Expose 4.1 Billion Records In First Six Months Of 2019 [Електронний ресурс]: Forbes – Режим доступу: https://www.forbes.com/sites/daveywinder/2019/08/20/data-breaches-expose-41-billion-records-in-first-six-months-of-2019/

12. Struggling with password overload? You're not alone [Електронний ресурс]: techradar – Режим доступу: https://www.techradar.com/news/most-people-have-25-more-passwords-than-at-the-start-of-the-pandemic

13. Alliance Overview [Електронний ресурс]: FIDO Alliance – Режим доступу: https://fidoalliance.org/overview/

14. Universal 2nd Factor (U2F) Overview [Електронний ресурс]: FIDO Alliance Proposed Standard 09 October 2014 – Режим доступу: https://fidoalliance.org/specs/fido-u2f-v1.0-ps-20141009/fido-u2f-overview-ps-20141009.pdf

15. Client To Authenticator Protocol [Електронний ресурс]: FIDO Alliance Proposed Standard 27 September 2017 – Режим доступу: https://fidoalliance.org/specs/fido-v2.0-ps-20170927/fido-client-to-authenticator-protocol-v2.0-ps-20170927.pdf

16. Web Authentication: An API for accessing Public Key Credentials [Електронний ресурс]: W3C Recommendation, 8 April 2021 – Режим доступу: https://www.w3.org/TR/2021/REC-webauthn-2-20210408/

17. How FIDO Works [Електронний ресурс]: FIDO Alliance – Режим доступу: https://fidoalliance.org/how-fido-works/

18. Lamport, L. Password Authentication with Insecure Communication [Електронний ресурс]: Communications of the ACM – Режим доступу: http://lamport.azurewebsites.net/pubs/password.pdf

19. The S/KEY One-Time Password System [Електронний ресурс]: Network Working Group. Request for Comments: 1760 – Режим доступу: https://www.rfc-editor.org/rfc/rfc1760

20. SKEY(1) [Електронний ресурс]: OpenBSD manual page server – Режим доступу: https://man.openbsd.org/skey.1

21. HOTP: An HMAC-Based One-Time Password Algorithm [Електронний ресурс]: Network Working Group. Request for Comments: 4226 – Режим доступу: https://www.rfc-editor.org/rfc/rfc4226

22. TOTP: Time-Based One-Time Password Algorithm [Електронний ресурс]: Internet Engineering Task Force (IETF). Request for Comments: 6238 – Режим доступу: https://www.rfc-editor.org/rfc/rfc6238

23. OCRA: OATH Challenge-Response Algorithm [Електронний ресурс]: Internet Engineering Task Force (IETF). Request for Comments: 6287 – Режим доступу: https://www.rfc-editor.org/rfc/rfc6287

24. ring [Електронний ресурс]: Github – Режим доступу: https://github.com/briansmith/ring

25. Room [Електронний ресурс]: Android Developers – Режим доступу: https://developer.android.com/jetpack/androidx/releases/room

26. Cipher [Електронний ресурс]: Android Developers – Режим доступу: https://developer.android.com/reference/javax/crypto/Cipher

27. Android Keystore system [Електронний ресурс]: Android Developers – Режим доступу: https://developer.android.com/training/articles/keystore

28. Apache Commons Codec [Електронний ресурс]: Apache Commons – Режим доступу: https://commons.apache.org/proper/commons-codec/

ДОДАТКИ

ДОДАТОК А

Вихідний код прототипу програмного додатка для генерації TOTP паролів

```
use std::time::SystemTime;

use base32;
use hmac::{HMAC_SHA1_FOR_LEGACY_USE_ONLY, Key, Tag};
use ring::hmac;

fn main() {
    let mut user_code = String::new();
    println!("Enter secret key:");
    std::io::stdin().read_line(&mut user_code).expect("Read error");
    let user_key = base32::decode(base32::Alphabet::RFC4648 { padding:
false }, &user_code.trim()).unwrap();

    let unix_time =
SystemTime::now().duration_since(SystemTime::UNIX_EPOCH).unwrap();
    let unix_time_seconds = unix_time.as_secs();
    println!("Unix time in seconds: {unix_time_seconds}");
    println!("TOTP code: {}", totp(&user_key, unix_time_seconds));
}

fn hmac_sha1(key: &[u8], counter: &[u8]) -> Tag {
    let hmac_key = Key::new(HMAC_SHA1_FOR_LEGACY_USE_ONLY, key);
    let tag = hmac::sign(&hmac_key, counter);
    return tag;
}

fn truncate_hmac(hmac_tag: &[u8], digits: u32) -> u32 {
    assert_eq!(hmac_tag.len(), 20);
    assert!((6..=10).contains(&digits));
    let offset = (hmac_tag[19] & 0xf) as usize;
    let code = (hmac_tag[offset] as u32 & 0x7f) << 24
        | (hmac_tag[offset + 1] as u32 & 0xff) << 16
        | (hmac_tag[offset + 2] as u32 & 0xff) << 8
        | (hmac_tag[offset + 3] as u32 & 0xff);
    return code % 10u32.pow(digits);
}

fn totp(key: &[u8], time_s: u64) -> u32 {
    let counter = time_s / 30;     // where 30 = step in seconds
    let hmac_result = hmac_sha1(key, counter.to_be_bytes().as_ref());
    return truncate_hmac(hmac_result.as_ref(), 6);
}
```

```
#[cfg(test)]
mod tests {
    use super::*;

    // Example from the original RFC 4226
    #[test]
    fn truncate_test() {
        assert_eq!(
            truncate_hmac([0x1fu8, 0x86, 0x98, 0x69, 0x0e, 0x02, 0xca,
0x16, 0x61, 0x85,
                                0x50, 0xef, 0x7f, 0x19, 0xda, 0x8e,
0x94, 0x5b, 0x55, 0x5a].as_slice(), 6),
            872921,
        )
    }

    #[test]
    fn totp_test() {
        let key = "12345678901234567890".as_bytes();
        let times = [59u64, 1111111109, 1111111111, 1234567890,
2000000000, 20000000000];
        let outputs = [94287082u32, 07081804, 14050471, 89005924,
69279037, 65353130];

        for (time, output) in
times.as_ref().iter().zip(outputs.as_ref()) {
            assert_eq!(totp(key, *time * 1000), output %
10u32.pow(6));
        }
    }
}
```

ДОДАТОК Б

Окремі файли вихідного коду програмного засобу для Android

Файл AesGcmSecretEncryptor.kt

```
package totp_android.data.crypto

import totp_android.domain.crypto.SecretEncryptor
import javax.crypto.Cipher
import javax.crypto.SecretKey
import javax.crypto.spec.GCMParameterSpec

class AesGcmSecretEncryptor(
    private val secretKey: SecretKey,
) : SecretEncryptor {
    private val cipher: Cipher =
Cipher.getInstance("AES/GCM/NoPadding")
    override val ivSize: Int
        get() = 12

    override fun encrypt(plainSecret: ByteArray, iv: ByteArray?):
ByteArray {
        init(Cipher.ENCRYPT_MODE, iv!!)
        return cipher.doFinal(plainSecret)
    }

    override fun decrypt(encryptedSecret: ByteArray, iv: ByteArray?):
ByteArray {
        init(Cipher.DECRYPT_MODE, iv!!)
        return cipher.doFinal(encryptedSecret)
    }

    private fun init(mode: Int, iv: ByteArray) {
        cipher.init(mode, secretKey, GCMParameterSpec(128, iv))
    }
}
```

Файл TotpCodeGeneratorImpl.kt

```
package totp_android.data.crypto

import org.apache.commons.codec.digest.HmacAlgorithms
import org.apache.commons.codec.digest.HmacUtils
import totp_android.domain.crypto.TotpCodeGenerator
import kotlin.experimental.and
import kotlin.math.pow
import kotlin.time.Duration
```

```
class TotpCodeGeneratorImpl : TotpCodeGenerator {
    override fun generate(secret: ByteArray, unixTime: Duration): Int
{
        val seconds = toBigEndianBytes(unixTime.inWholeSeconds / 30)
        val hmac = HmacUtils(HmacAlgorithms.HMAC_SHA_1,
secret).hmac(seconds)
        return truncate(hmac)
    }

    private fun toBigEndianBytes(long: Long): ByteArray {
        return ByteArray(8) { i -> (long ushr (8 * (7 -
i))).toByte() }
    }

    private fun truncate(hmac: ByteArray, digits: Int = 6): Int {
        val offset = (hmac.last() and 0xf).toInt()
        val code = (hmac[offset].toUInt() and 0x7fu shl 24) or
                (hmac[offset + 1].toUInt() and 0xffu shl 16) or
                (hmac[offset + 2].toUInt() and 0xffu shl 8) or
                (hmac[offset + 3].toUInt() and 0xffu)
        return code.toInt() % (10f.pow(digits).toInt())
    }
}
```

Файл AndroidKeyStoreRepository.kt

```
package totp_android.data.crypto

import android.security.keystore.KeyGenParameterSpec
import android.security.keystore.KeyProperties
import totp_android.domain.crypto.SecretKeyRepository
import java.security.KeyStore
import java.security.KeyStore.SecretKeyEntry
import javax.crypto.KeyGenerator
import javax.crypto.SecretKey

class AndroidKeyStoreRepository : SecretKeyRepository {
    private val keyStore =
KeyStore.getInstance("AndroidKeyStore").apply { load(null) }

    override fun generateRandomKey(alias: String): SecretKey {
        val keyGenerator = KeyGenerator.getInstance("AES",
"AndroidKeyStore")
        keyGenerator.init(
            KeyGenParameterSpec.Builder(
                alias,
                KeyProperties.PURPOSE_ENCRYPT or
KeyProperties.PURPOSE_DECRYPT
            )
                .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
```

```
                .setKeySize(256)
                // false to use custom IV
                .setRandomizedEncryptionRequired(false)
                .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDIN
G_NONE)
                .build()
        )
        return keyGenerator.generateKey()
    }

    override fun getKey(alias: String): SecretKey? {
        if (!keyStore.containsAlias(alias)) return null
        val entry = keyStore.getEntry(alias, null) as SecretKeyEntry
        return entry.secretKey
    }

    override fun deleteKey(alias: String) {
        keyStore.deleteEntry(alias)
    }

    override fun getAliases(): List<String> {
        return keyStore.aliases().toList()
    }
}
```

Файл UseCases.kt

```
package totp_android.domain.usecases

import totp_android.domain.crypto.SecretEncryptor
import totp_android.domain.crypto.TotpCodeGenerator
import totp_android.domain.entities.EncryptedTotpKey
import totp_android.domain.repository.TotpKeyRepository
import java.security.SecureRandom
import kotlin.time.Duration

class AddNewTotpUseCase(
    private val repository: TotpKeyRepository,
    private val encryptor: SecretEncryptor,
) {
    suspend operator fun invoke(plainSecret: ByteArray, name: String)
{
        val random = SecureRandom()
        val iv = ByteArray(encryptor.ivSize)
        random.nextBytes(iv)
        repository.addKey(EncryptedTotpKey(0, name,
encryptor.encrypt(plainSecret, iv), iv))
    }
}

class GenerateTotpCodeUseCase(
```

```
    private val generator: TotpCodeGenerator,
    private val encryptor: SecretEncryptor,
    private val getUnixTime: () -> Duration,
) {
    operator fun invoke(totpKey: EncryptedTotpKey): Int {
        val secret = encryptor.decrypt(totpKey.secret, totpKey.iv)
        return generator.generate(secret, getUnixTime())
    }
}
```