**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

**КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ**

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____2021 р.

На правах рукопису
004.056.5:510.22(043.3)

# ДИПЛОМНА РОБОТА

## ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

## ОСВІТНЬОГО СТУПЕНЯ «БАКАЛАВР»

**Тема**: Система багатофакторної автентифікації та розмежування доступу до інформаційних систем

**Виконавець:** Н.О. Кірносов
**Керівник:** к.т.н. О.О. Висоцька
**Нормоконтролер:** к.т.н. О.О. Висоцька

**Київ 2021**

**НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ**

**Факультет:** Кібербезпеки, комп'ютерної та програмної інженерії

**Кафедра:** Комп'ютеризованих систем захисту інформації

**Освітній ступінь:** Бакалавр

**Спеціальність:** 125 «Кібербезпека»

**Освітньо-професійна програма**: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ С.В. Казмірчук
«_____»_____2021 р.

**ЗАВДАННЯ**

**на виконання дипломної роботи**

**здобувача вищої освіти Кірносова Нікіти Олександровича**

1. Тема: *Система багатофакторної автентифікації та розмежування доступу до інформаційних систем*
затверджена наказом ректора від «26» квітня 2021 р. № 652/ст.

2. Термін виконання: з 10.05.2021 р. по 20.06.2021 р.

3. Вихідні дані: проаналізувати існуючі системи автентифікації та розмежування доступу; виявлення їх переваг і недоліків; розробити методику, алгоритм та програмне забезпечення системи багатофакторної автентифікації.

4. Зміст пояснювальної записки: аналіз існуючих систем багатофакторної автентифікації та розмежування доступу до інформаційної системи; розробка програмного додатку багатофакторної автентифікації.

# КАЛЕНДАРНИЙ ПЛАН

## виконання дипломної роботи

| № з/п | Етапи виконання дипломної роботи | Термін виконання етапів | Примітка |
|---|---|---|---|
| 1. | Уточнення постановки задачі | 21.04.2021 | *Виконано* |
| 2. | Аналіз літературних джерел | 30.04.2021 | *Виконано* |
| 3. | Обґрунтування рішення | 15.05.2021 | *Виконано* |
| 4. | Збір інформації | 16.04.2021 | *Виконано* |
| 5. | Аналіз та дослідження систем багаофакторної автентифікації та розмежування доступу до інформаційних систем | 17.05.2021 | *Виконано* |
| 6. | Розробка та дослідження програмного додатку багатофакторної автентифікації | 20.05.2021 | *Виконано* |
| 7. | Тестування програмного додатку багатофакторної автентифікації | 25.05.2021 | *Виконано* |
| 8. | Оформлення і друк пояснювальної записки | 28.05.2021 | *Виконано* |
| 9. | Перевірка на антиплагіат | 07.06.2021 | *Виконано* |
| 10. | Оформлення презентації | 08.06.2021 | *Виконано* |
| 11. | Отримання рецензій від рецензентів | 14.06.2021 | *Виконано* |

Здобувач вищої освіти _____ Н.О. Кірносов

(підпис, дата)

Керівник дипломної роботи _____ О.О. Висоцька

(підпис, дата)

# РЕФЕРАТ

Дипломна робота складається зі вступу, двох розділів, загальних висновків до кожного розділу, списку використаних джерел, додатків і 47 рисунків, дві таблиці.

Список використаних джерел містить 20 найменувань і займає 2 сторінки. Загальний обсяг роботи 63 сторінки.

Метою дипломної роботи є розробка система багатофакторної автентифікації та розмежування доступу до інформаційної системи.

В роботі були оцінені матеріальні можливості та потреби програми, яку потрібно захистити від несанкціонованого доступу, був проведенний експеримент з алгоритмами багатофакторної автентифікації та розмежування доступу за допомогою Angular 7.

В роботі також був проведений аналіз та порівняння існуючих методів автентифікації, систем багатофакторної автентифікації та розмежування доступу.

Також було продемонстровано веб-додаток з двохфакторною автентифікацією. Завдяки роботі було коротко поясненно як відбувається автентифікація в інформаційних системах.

Було показано вихідні коди всіх алгоритмів щоб користувач міг використовувати будь-яку частину коду так само, як і будь-який компонент програмного забезпечення.

# ЗМІСТ

# ВСТУП

**Актуальність** теми полягає у тому, що багатофакторна автентифікація та розмежування доступу сьогодні є одними з основних способів захисту інформаційних систем від несанкціонованого доступу до інформації і завжди потребують удосконалення методів побудови.

**Метою дипломної роботи є** розробка система багатофакторної автентифікації та розмежування доступу до інформаційної системи.

Досягнення мети потребує розв'язання таких **задач**:

-Проаналізувати методи автентифікації користувачів інформаційних систем, способи реалізації багатофакторної автентифікації та технології розмежування доступу до інформаційних систем.

-Розробити систему багатофакторної автентифікації та розмежування доступу до інформаційної системи.

-Провести тестування розробленої системи багатофакторної автентифікації та розмежування доступу до інформаційної системи.

**Об'єктами дослідження дипломної роботи є:** процеси автентифікації, зокрема багатофакторної автентифікації, процеси розмежування доступу до інформаційної системи.

**Предметами дослідження є** методи автентифікації, зокрема багатофакторної автентифікації, технології розмежування доступу до інформаційної системи.

**Практична цінність результатів** полягає в тому, що в роботі розроблено систему багатофакторної автентифікації та розмежування доступу до інформаційної системи, яка за допомогою аналізу таких двох факторів автентифікації, як довгостроковий пароль і одноразовий пароль, дозволяє забезпечити санкціонований доступ користувачів до ресурсів інформаційних систем.

# РОЗДІЛ 1
# АНАЛІЗ СУЧАСНИХ МЕТОДІВ І ЗАСОБІВ ІДЕНТИФІКАЦІЇ, АВТЕНТИФІКАЦІЇ ТА РОЗМЕЖУВАННЯ ДОСТУПУ В ІНФОРМАЦІЙНИХ СИСТЕМАХ

## 1.1 Що таке Інформаційні Системи

Інформаційна система (ІС) - це матеріальна система, яка організовує, зберігає та перетворює інформацію. Основним предметом і продуктом праці в такій системі є інформація.

Таким чином, можна сказати, що інформаційна система - це система, призначена для зберігання, обробки, пошуку, розповсюдження, передачі та надання інформації.

Інформаційна система - це організаційно впорядкований набір документів та інформаційних технологій, включаючи ті, що використовують комп'ютерні технології та комунікації, реалізуючи різні інформаційні процеси.

Інформаційна система - сукупність засобів збору, передачі, обробки та зберігання інформації, а також персонал, який виконує такі дії; організаційно організований набір документів (ІР) і інформаційних технологій, що реалізують інформаційні процеси (ІР); організаційно-функціональна структура, яка реалізує обробку інформації. Як було сказано вище, основними представниками інформаційних систем є архіви, бібліотеки, музеї, різні інформаційні відділи та організації.

Розробка ІС починається зі створення організаційних підсистем, при цьому підсистема науково-технічної підготовки виробництва відноситься до функціональних інформаційних підсистем.

Інформаційний фонд системи, як правило, являє собою базу або сукупність баз даних, створених у вигляді табличних, ієрархічних та мережевих структур.

Мережеві IP-адреси використовують два способи спілкування з кінцевими користувачами:

1. Розподіл часу - кожен учасник мережі, здається, користується своїм комп’ютером. Основним завданням розробників та адміністраторів мережі є захист даних від несанкціонованого доступу та забезпечення взаємної ізоляції учасників;

2. Надання групових рішень - організація взаємодії користувачів у процесі прийняття рішень. Цей метод поєднує в собі комунікаційні, обчислювальні та технології прийняття рішень для групи людей для реалізації складних неструктурованих завдань.

Будь-який ІР повинен мати такі властивості:

1) функціональність - будь-який ІР-об'єкт повинен містити функціонально повний і максимально незалежний набір операцій обробки даних;

2) зв'язність, коли об'єкт реалізує набір взаємопов'язаних функцій - методів, що працюють з однаковими даними, деякі з яких приховані для системи в цілому;

3) маскування - доступність для системи лише об'єктних параметрів, що представляють собою набори вхідних і вихідних інтерфейсів об'єкта. Важливою властивістю ІС є мінімізація кількості її інформаційних посилань, які залежать від вартості модифікації системи, коли вона працює в змінених умовах, та вирішення змінених функціональних проблем.

За призначенням функціонування інформаційна ІС поділяється на: державну, юридичну (законодавчу), ділову, фінансову, науково-технічну, освітню, соціальну, розважальну та інші. В цьому випадку, наприклад, фінансова інформація ділиться на: бухгалтерську, банківську, податкову та іншу, а медична (як і інші) може містити всі перераховані вище функції.

Областями застосування є бізнес, професійна діяльність, інформація для споживачів і електронна комерція.

За рівнем управління розрізняють стратегічні, тактичні й оперативні інформаційні системи.

За ступенем застосування технічних засобів ІС діляться на автоматизовані і неавтоматизовані. Дана автоматизація означає автоматизацію від окремих

процесів і завдань до рівня автоматизації підприємств, установ і їх сукупності в масштабі території (регіону), тобто являє собою клас систем, орієнтованих на автоматизацію окремих функцій або процесів, і клас інтегрованих системи і комплекси обробки і доставки даних, автоматизація функцій і процесів управління, підтримка прийняття рішень і ін.

За видами інформації - документальна, фактична та документально-фактологічна ІС.

До документальних ІС належать інформаційно-пошукові системи (МКС), інформаційно-логічні та інформаційно-семантичні системи.

Фактографічні ІС діляться на дві категорії:

1) системи обробки даних (DDS),

2) автоматизовані інформаційні системи (АІС) та автоматизовані системи управління (АСУ).

Документально-фактологічна ІС містить:

1) автоматизовані документально-фактичні інформаційно-пошукові системи науково-технічної інформації (ADFIPS STI).

2) автоматизовані системи пошуку документальної та фактичної інформації в автоматизованій системі нормативно-методичного забезпечення управління (ADFIPS в ASNMOU).

Існують також такі ІС, як: бухгалтерський облік, банківська справа, ІС ринку ринку цінних паперів, ІС управління (ISS), системи підтримки прийняття рішень (DSS), експертні системи (ES), гібридні експертні системи (HPP), моніторинг ІС (IMS) тощо.

У АІС розміщуються різні типи інформації: бібліографічні дані (записи); фактичні дані (записи); повнотекстові документи (записи); довідкові дані (включаючи покажчики); математичні або числові (цифрові, табличні) дані; графічні дані; мультимедійні дані.

ІС можна класифікувати за видами оброблюваної інформації: текстові процесори та редактори (текст); Графічні процесори та редактори (графіка); Системи управління базами даних (СУБД), табличні процесори, алгоритмічні

мови програмування (дані); Експертні системи (знання), мультимедійні системи (об'єкти реального світу, включаючи будь-яку інформацію) тощо. Звичайно, ця класифікація є доволі довільною. Таким чином, сучасний текстовий процесор може забезпечити наявність та взаємодію практично будь-якого типу інформації, гіпертексту та комунікаційних можливостей. Інша справа, наскільки це задовольнить відповідних користувачів.

Інформаційні технології засновані на широкому використанні інформаційних ресурсів інформаційних систем, що забезпечують ефективне надання інформаційних послуг користувачам, шляхом пошуку та передачі їм необхідних даних.

Інформаційні ресурси - це сукупність даних, отриманих і накопичених в процесі розвитку науки та практичної діяльності людей для багатоцільового використання в суспільному виробництві та управлінні.

Інформаційні ресурси архівів, інформаційних систем та організацій, а також бібліотек є основою НДДКР. Отже, до інформаційних організацій належать: архіви, бібліотеки, інформаційні служби, музеї тощо.

Існують ІР: національні, територіально-адміністративні утворення, інформаційні служби, бібліотеки, інші організації та їх підрозділи, а також персоналії.

Загалом, за ступенем доступності, інформаційні ресурси поділяються на: загальнодоступні та з обмеженим доступом користувачів (останні є ЦСП і секретні - закритий ІР).

Сучасні електронні інформаційні ресурси - це електронні тексти та документи, розміщені на різних електронних носіях даних, на сайтах та порталах, в електронних бібліотеках та офісах.

## 1.2 Системи розмежування доступу

Після виконання ідентифікації та автентифікації підсистема захисту встановлює повноваження (набір прав) суб’єкта для подальшого контролю дозволеного використання об’єктів ІВ.

Як правило, повноваження суб'єкта представлені: списком доступних користувачеві ресурсів та правами доступу до кожного ресурсу зі списку.

Існують такі методи контролю доступу:

1) диференціація за списками;

2) використання матриці встановлення влади;

3) диференціація за рівнями секретності та категоріями;

4) диференціація паролів.

При диференціації доступу за списками встановлюються кореспонденції: для кожного користувача - список ресурсів та права доступу до них, або для кожного ресурсу - список користувачів та їх права доступу до цього ресурсу.

Списки дозволяють встановлювати права з точністю до користувача. Тут легко додати права або явно заборонити доступ. Списки використовуються в підсистемах безпеки операційних систем та систем управління базами даних.

Використання матриці авторизації передбачає використання матриці доступу (таблиця авторизації). У цій матриці рядки - це ідентифікатори суб’єктів, які мають доступ до ІС, а стовпці - об’єкти (ресурси) ІС. Кожен елемент матриці може містити ім'я та розмір наданого ресурсу, право доступу (читання, запис тощо), посилання на іншу інформаційну структуру, яка визначає права доступу, посилання на програму, яка управляє правами доступу тощо.

Цей метод забезпечує більш уніфікований та зручний підхід, оскільки вся інформація про дозволи зберігається в одній таблиці, а не у вигляді списків різних типів. Недоліками матриці є її можлива громіздкість і субоптимальність (більшість комірок порожні).

Поділ доступу за рівнями секретності та категоріями - це поділ ресурсів ІВ за рівнями секретності та категоріями.

При розмежуванні за ступенем секретності існує кілька рівнів, наприклад:

спільний, конфіденційний, секретний, цілком секретний. Повноваження кожного користувача встановлюються відповідно до максимального рівня

секретності, до якого він допускається. Користувач має доступ до всіх даних, рівень конфіденційності яких не перевищує вказаний для нього. Наприклад, користувач, який має доступ до "секретних" даних, також має доступ до "конфіденційних" та "спільних" даних.

При диференціації за категоріями встановлюється і контролюється ранг категорії користувачів. Відповідно, усі ресурси ІС поділяються за рівнями важливості, а певний рівень відповідає категорії користувачів.

Диференціація паролів, очевидно, передбачає використання методів доступу суб'єктів до об'єктів за паролем. При цьому використовуються всі методи захисту паролем. Очевидно, що постійне використання паролів створює незручності для користувачів та затримки часу. Тому ці методи застосовуються у виняткових ситуаціях.

На практиці зазвичай поєднуються різні методи обмеження доступу. Наприклад, перші три методи покращені захистом паролем.

### 1.3 Огляд технологій автентифікації користувачів в інформаційних системах

Сьогодні існує декілька технологій ідентифікації та автентифікації користувачів в інформаційно-комунікаційних системах.

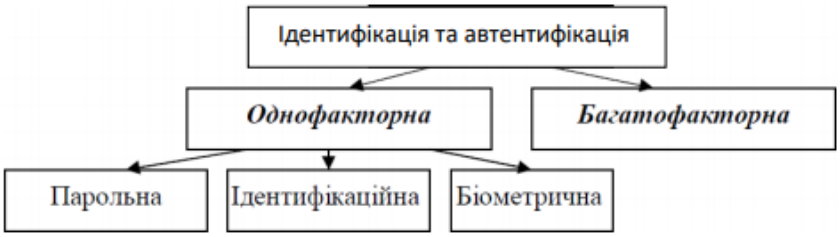


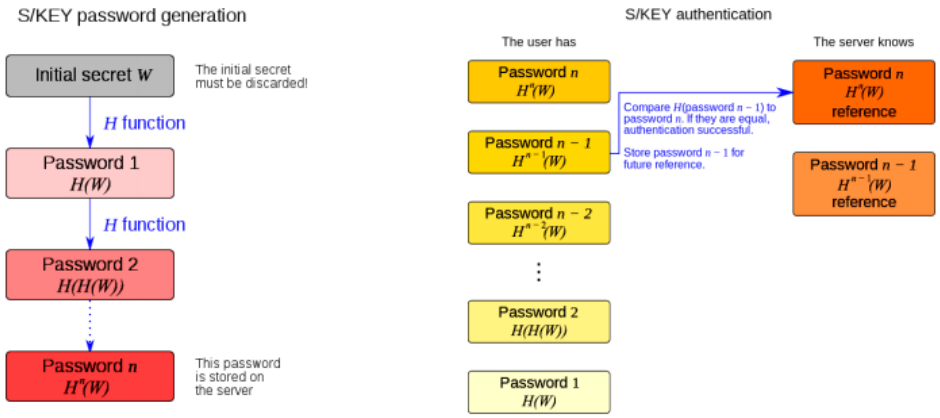

Рис 1.1 Види ідентифікації та автентифікації

Кожен має свої переваги та недоліки, завдяки чому одні технології придатні для використання в одних комп’ютерних системах, інші - в інших. Однак у багатьох випадках чітко визначеного рішення не існує. Тому як розробники програмного забезпечення, так і користувачі повинні самостійно вирішувати, який метод ідентифікації застосовувати у своїй ІКС.

### 1.4 Парольна автентифікація

Останнім часом ідентифікація / автентифікація по паролю була чи не єдиним способом ідентифікувати користувача. І це зовсім не дивно. Справа в тому, що ідентифікацію за паролем найпростіше реалізувати і використовувати. Суть його в наступному. Кожен зареєстрований користувач системи отримує набір особистих даних (зазвичай з використанням пари логін-пароль). Далі, кожен раз, коли ви намагаєтеся увійти в систему, він повинен надавати свою інформацію. Ну, тому що він унікальний для кожного користувача, на основі його система робить висновок про людину і ідентифікує його. При правильному використанні паролі можуть забезпечити рівень безпеки, прийнятний для багатьох організацій. Однак, незважаючи на всі їхні характеристики, їх слід вважати найслабшою засобом перевірки. Слабкий захист паролем - одна з основних причин уразливості комп'ютерних систем для спроб несанкціонованого доступу. Але тепер символічний пароль - це найпоширеніший спосіб ідентифікації і автентифікації користувачів, який залишиться таким ще довгий час. Подальші заходи істотно підвищать надійність парольного захисту: - введення технічних обмежень: установка мінімальної довжини пароля, використання в паролі різних груп символів (пароль повинен містити літери, цифри, знаки пунктуації і т. Д.); - управління умовами паролів, їх періодична зміна - обмеження кількості невдалих спроб входу в систему; - використання програмних генераторів паролів (така програма, заснована на простих правилах, може генерувати тільки милозвучні паролі, які запам'ятовуються). Для більш детального розгляду принципів побудови пральних систем сформулюємо кілька основних визначень. Ідентифікатор користувача - якийсь унікальний обсяг інформації, що дозволяє розрізняти окремих користувачів парольної системи (ідентифікувати їх). ID також часто називають ім'ям користувача або ім'ям облікового запису користувача. Пароль користувача - це секретний обсяг інформації, відомий тільки користувачеві і системі паролів, який користувач може запам'ятати і відправити для проходження процедури автентифікації. Одноразовий пароль дозволяє користувачеві пройти автентифікацію один

раз. Ви можете використовувати кілька паролів для повторної перевірки. Обліковий запис користувача - це набір ідентифікаторів користувача і пароля. База даних користувачів системи паролів містить облікові записи всіх користувачів цієї системи паролів. Під кодовою системою ми розуміємо програмно-апаратний комплекс, який реалізує системи ідентифікації і автентифікації користувачів АСУ ТП під час сну з одноразовими (рис. 1.2.а) або множинними (рис. 1.2.б) паролями.




Як правило, такий комплекс функціонує разом із підсистемами контролю доступу та реєстрації подій. У деяких випадках система паролів може виконувати ряд додаткових функцій, включаючи створення та розповсюдження короткочасних (сесійних) криптографічних ключів. Основними компонентами системи прання є:

- інтерфейс користувача;
- інтерфейс адміністратора;
- модуль зв'язку з іншими підсистемами безпеки;
- база даних рахунків.

Система паролів - це «передова лінія оборони» всієї системи безпеки. Деякі його елементи (включаючи ті, що реалізують інтерфейс) можуть бути розташовані у місцях, відкритих для потенційного зловмисника. Таким чином, система паролів стає однією з перших цілей атаки, коли зловмисник вторгається в захищену систему. Нижче наведено типи загроз безпеці систем паролів.

1). Розкриття налаштувань облікового запису через:

- вибір в інтерактивному режимі;

- шпигунство;

- умисна передача пароля його власником іншій особі;

- захопити базу паролів системи;

- перехоплення пральний інформації, переданої по мережі;

- зберігання пароля в доступному місці.

2). Втручання в роботу компонентів пральний системи через:

- впровадження програмних закладок;

- виявлення і використання помилок, допущених на етапі розробки;

- збій в системі паролів.

Деякі з цих типів загроз пов'язані з наявністю так званого людського фактора, який проявляється в тому, що користувач може:

- виберіть пароль, який легко запам'ятати, а також легко підібрати;

- запишіть складний для запам'ятовування пароль і покладіть запис в доступне місце;

- введіть пароль так, щоб його бачили сторонні;

- передати пароль іншій особі навмисно або під впливом помилки.

Крім вищесказаного, необхідно підкреслити наявність «парадоксу людського фактора». Справа в тому, що користувач часто прагне бути більш противником системи паролів, як, до речі, будь-якої системи безпеки, робота якої впливає на її умови роботи, а не союзником системи безпеки, тим самим послаблюючи її. Важливим аспектом стабільності пароля є спосіб зберігання паролів в базі даних облікового запису.

Доступними параметрами зберігання паролів:

- в відкриту; - у вигляді згорток (хеширования);

- зашифровано якимось ключем.

Найцікавішими є другий і третій методи, що мають ряд особливостей. Хешування (використання незворотної хеш-функції для будь-якої інформації перетворює її в унікальний код) не забезпечує захисту від вгадування пароля у словнику, якщо зловмисник отримує базу даних. Вибираючи алгоритм

хешування, який буде використовуватися для обчислення зведених паролів, необхідно забезпечити різницю між значеннями зведених даних, отриманих на основі різних паролів користувачів. Крім того, ви повинні надати механізм, який гарантує, що зведення є унікальними, якщо двоє користувачів вибирають однакові паролі. Для цього при обчисленні кожної згортки зазвичай використовується якась «випадкова» інформація, наприклад, видана генератором псевдовипадкових чисел. При шифруванні паролів особливо важливим є спосіб створення та зберігання ключа шифрування для бази даних облікових записів. Існує кілька можливих варіантів: ключ генерується програмно і зберігається в системі, що дозволяє автоматично перезавантажити його; ключ генерується програмним забезпеченням і зберігається на зовнішньому носії, з якого він зчитується кожного разу при його запуску; ключ генерується на основі пароля, вибраного адміністратором, який вводиться в систему кожного разу при його запуску. У другому випадку необхідно забезпечити неможливість автоматичного перезапуску системи, навіть якщо вона виявляє носій за допомогою ключа. Для цього ви можете попросити адміністратора підтвердити продовження процедури завантаження, наприклад, натиснувши клавішу на клавіатурі. Найбезпечніше зберігання паролів забезпечується хешуванням та подальшим шифруванням отриманих згорток, тобто комбінацією другого та третього методів. З огляду на те, що користувачі часто вибирають недостатньо надійні паролі, можна зробити висновок, що отримання бази даних облікового запису або перехоплення зведеного значення пароля, передане через мережу, створює серйозну загрозу безпеці для системи паролів. У більшості випадків автентифікація відбувається в розподілених системах і передбачає передачу інформації про параметри облікових записів користувачів по мережі. Якщо інформація, передана по мережі під час процесу автентифікації, не захищена належним чином, існує ризик того, що її може перехопити зловмисник і використовувати для компрометації системи захисту паролем. Відомо, що багато комп’ютерні системи дозволяють перевести мережевий адаптер у

режим прослуховування, адресований іншим одержувачам мережевого трафіку в мережі, на основі передачі пакетів даних.

Згадаймо основні види захисту мережевого трафіку: фізичний захист мережі; остаточне шифрування; шифрування пакетів. Поширені наступні способи передачі паролів через мережу:

- відкрито;

- зашифровані;

- у формі звивин;

- без прямої передачі інформації про пароль ("з нульовим розголошуванням"). Перший метод досі використовується у багатьох популярних програмах (наприклад, TELNET, FTP). У безпечній системі вона може використовуватися лише разом із захистом мережевого трафіку.

Коли паролі передаються в зашифрованому вигляді або у вигляді звивин через мережу з відкритим фізичним доступом, можливі такі загрози безпеці системи паролів:

- перехоплення та повторне використання інформації;

- перехоплення та відновлення паролів;

- зміна переданої інформації з метою введення контрольної сторони в оману;

–Зловмисник імітує дії довіреної сторони, щоб ввести користувача в оману.

Схеми автентифікації нульових знань або нульових знань вперше з’явилися в середині 1980-х - на початку 1990-х. Їх основна ідея полягає в тому, щоб дати змогу одному з пари суб’єктів довести істинність будь-якого твердження іншому, не надаючи йому жодної інформації про зміст самого висловлювання. Наприклад, перший суб'єкт (що «верифікується») може переконати другого ("верифікатор"), що він знає певний пароль, фактично не передаючи жодної інформації про сам пароль. Ця ідея знайшла своє відображення в терміні «нульовий доказ знань». Що стосується захисту паролем, це означає, що якщо зловмисник виявляється замість валідатора, він не отримує жодної інформації про підтверджене твердження і, зокрема, про

пароль. Загальна схема процедури автентифікації з нульовим розкриттям складається з послідовності обміну інформацією (ітерацій) між двома учасниками процедури, після чого верифікатор із заданою ймовірністю робить правильний висновок про істинність твердження, що перевіряється. Зі збільшенням кількості ітерацій зростає ймовірність правильного визнання істинності (або хибності) висловлювання.

Інший спосіб поліпшити стабільність систем паролів, пов'язаних із передачею паролів по мережі, - це використання одноразових (one - time) паролів. Загальний підхід до використання одноразових паролів базується на послідовному використанні хеш-функції для обчислення наступного одноразового пароля на основі попереднього. Спочатку користувач отримує упорядкований список одноразових паролів, останній з яких також зберігається в системі автентифікації. При кожній реєстрації користувач вводить наступний пароль, і система обчислює його згортку і порівнює її зі стандартним, що зберігається в системі. Якщо є збіг, користувач успішно автентифікується, і введений ним пароль зберігається для використання в якості посилання під час наступної реєстрації. Захист від прослуховування мережі за такою схемою заснований на властивостях незворотності хеш-функції. Найвідомішими практичними реалізаціями схем одноразових паролів є програмний пакет S / KEY та система OPIE, розроблена на його основі.

Головною перевагою автентифікації пароля є простота впровадження та використання. Крім того, впровадження автентифікації за допомогою пароля є економічно вигідним: цей процес впроваджено в більшості програмних продуктів. Таким чином, система захисту інформації проста і доступна. Основним недоліком ідентифікації паролів слід вважати величезну залежність надійності ідентифікації від самих користувачів, точніше від вибраних ними паролів. Справа в тому, що більшість людей використовують ненадійні ключові слова, про які легко вгадати. Сюди входять занадто короткі паролі, відомі комбінації символів тощо. Тому деякі експерти з

інформаційної безпеки рекомендують використовувати довгі паролі, що складаються з випадкової комбінації літер, цифр та різних символів.

### 1.5 Апаратна автентифікація

Цей принцип ідентифікації заснований на визначенні особи суб'єкта, ключа, який використовується виключно для нього.

Зрозуміло, що мова йде не про звичні для більшості людей ключі, а про спеціальні електронні ключі. В даний час найбільш поширені два типи пристроїв: різні картки (безконтактні картки, смарт-карти, магнітні картки тощо) і так звані токени (токени), які підключені безпосередньо до одного з комп'ютерних портів. Кожен апаратний (електронний) ідентифікатор - це фізичний пристрій, зазвичай невеликий для перенесення. Системи електронної ідентифікації та автентифікації включають:

Рис 1.3 Токени

1. Переносні токени:

- асинхронний - користувач вводить рядок в пристрій, отримує відповідь і вводить його в комп'ютер;

- ПІН-код / асинхронний - асинхронний спосіб доповнюється введенням ПІН-коду в пристрої;

- синхронно - наприклад, токен синхронізується за часом з сервером і генерує пароль для цього користувача в той момент, який вже введений в систему;

- ПІН / синхронно.

2. Різні карти - це пристрої, схожі на портативні автентифікатори, але більш складні за складом. Карти бувають:

- пасивні (карти пам'яті);

- активні (смарт-карти).

До останніх відносяться ЦП, мініатюрна операційна система, годинник, постійна пам'ять (RAM), криптографічний пам'ять (RAM), незалежна пам'ять або EEPROM (електрично стирається програмована постійна пам'ять) для зберігання цифрових ключів. За допомогою смарт-карти розраховуються одноразові паролі і здійснюється взаємодія з пристроєм через картрідер.

Після введення ПІН-коду картрідер сам запитує смарт-карту, і подальший процес відбувається без втручання людини, тому ви можете використовувати досить довгі ключі. Карт дуже багато, і працюють вони за різними принципами. Наприклад, безконтактні карти (також звані безконтактними картами) досить прості у використанні, що дозволяє ідентифікувати користувачів як в комп'ютерних системах, так і в системах доступу в приміщення. Найнадійнішими вважаються смарт-карти - аналоги звичайних банківських карт багатьох людей. Крім того, існують більш дешеві, але менш захищені від злому карти: магнітні, штрих-коди і т. Д. Що таке USB-ключ, розглянемо на прикладі цифрового підпису від Aladin Software. Цифровий підпис - це особистий засіб автентифікації і зберігання даних, яке підтримує цифрові сертифікати і електронні цифрові підписи (ЕЦП) на обладнанні.


Рис 1.4 Цифрові підписи

Цифровий підпис може бути у формі USB-ключа або стандартної смарт-карти.

Цифровий підпис підтримує та інтегрується з усіма основними системами та програмами, що використовують технологію смарт-карт або PKI (інфраструктура відкритих ключів). Основна мета: - двохфакторна автентифікація користувачів при доступі до захищених ресурсів (комп’ютери, мережі, додатки) - безпечне зберігання приватних ключів цифрових сертифікатів, криптографічних ключів, профілів користувачів, встановлення додатків тощо в енергонезалежній пам'яті ключів; - апаратна реалізація криптографічних операцій у довіреному середовищі (генерація ключів шифрування, симетричне та асиметричне шифрування, обчислення хеш-функцій, генерація ЕЦП).

Цифровий підпис як засіб автентифікації підтримується більшістю сучасних операційних систем, бізнес-додатків і продуктів інформаційної безпеки.

Можливості застосування:

- сувора автентифікація користувачів при доступі до серверів, баз даних, розділів веб-сайтів;

- безпечне зберігання секретної інформації: паролі, ключі шифрування, закриті ключі цифрових сертифікатів;

- захист електронної пошти (електронний підпис і шифрування, доступ);

- системи електронної комерції, «клієнт-банк», «домашній банк»;

- захист комп'ютера;

- захист мереж і каналів передачі даних шляхом побудови VPN (virtual private network - віртуальні приватні мережі);

- клієнт-банк, home -банк.

Цифровий підпис забезпечує: автентифікацію користувача за допомогою криптографічних методів; безпечне зберігання ключів шифрування та цифрових підписів, а також приватних ключів цифрових сертифікатів для доступу до захищених корпоративних мереж та інформаційних ресурсів; мобільність користувачів і можливість безпечної роботи з конфіденційними даними в ненадійному середовищі (наприклад, на чужому комп’ютері) через

те, що ключі шифрування та цифрові підписи генеруються цифровим ключем в апаратному забезпеченні і не можуть бути перехоплені; безпечне використання - цифровим ключем може користуватися лише власник, який знає PIN-код ключа; впровадження як західних, так і російських, а також вітчизняних стандартів шифрування та цифрового підпису; Зручність роботи - ключ виконаний у вигляді брелоків із світловою індикацією режимів роботи і безпосередньо підключений до портів USB, які зараз оснащені на 100% комп'ютерами, не вимагає спеціальних зчитувачів, джерел живлення, проводів тощо. .; використання одного ключа для вирішення багатьох різних завдань - вхід в комп'ютер, вхід у мережу, захист каналу, шифрування інформації, ЕЦП, захищений доступ до захищених розділів веб-сайтів, інформаційних порталів тощо.

Безконтактні смарт-карти підрозділяються на безконтактні і смарт-карти відповідно до міжнародних стандартів ISO / IEC 15693 та ISO / IEC 14443. Більшість безконтактних смарт-карт засновані на технології RFID. Основними компонентами безконтактних пристроїв є мікросхема і антена. Ідентифікатори можуть бути як активними (з акумуляторами), так і пасивними (без харчування). Ідентифікатори мають унікальні 32- або 64-бітові серійні номери. Системи безконтактної автентифікації не є криптографічно безпечними, за винятком спеціальних рекомендованих систем. USB-ключі працюють з USB-портом комп'ютера. Вони зроблені у вигляді брелків. Кожен ключ має 32- або 64-бітний серійний номер.

USB-ключі, представлені на ринку:

− eTokenR2, eToken Pro – компанія Aladdin Knowledge Systems;

− iKey10xx, iKey20xx,iKey 3000 – компаніяRainbow Technologies;

− ePass 1000 ePass 2000 – фірма Feitian Technologies;

− ruToken – розробка компанії «Актив» і фірми «АНКАД»;

− uaToken – компанія ТОВ«Технотрейд».

USB-ключі є наступниками смарт-карт, тому структура USB-ключів і смарт-карт ідентична. Основна перевага використання апаратної

ідентифікації - досить висока надійність. І дійсно, в пам'яті токенів можуть зберігатися ключі, які досить складно знайти. Крім того, вони реалізують безліч різних механізмів безпеки, а вбудований мікропроцесор дозволяє електронному ключу не тільки брати участь в процесі ідентифікації користувача, а й виконувати деякі інші корисні функції. Основна небезпека при використанні апаратної ідентифікації полягає в можливості крадіжки зловмисниками токенів або карт у зареєстрованих користувачів. Також вони можуть бути загублені, передані іншій людині, дубльовані. Другий недолік цієї технології - ціна. В цілому за останній час вартість як електронних ключів, так і програмного забезпечення, здатного з ними працювати, значно знизилася. Однак введення в експлуатацію такої системи ідентифікації все ж потребують певних інвестицій. Однак кожному зареєстрованому користувачу необхідно надати особисті токени. Крім того, з часом, коли деякі типи ключів можуть зношуватися, вони можуть бути втрачені і т.д. Тобто ідентифікація обладнання вимагає певних експлуатаційних витрат.

Основні переваги й недоліки різних типів цифрових ключів наведено у таблиці 1.1

Таблиця 1.1

| Продукт | Основні переваги | Основні недоліки |
|---|---|---|
| USB- токени | Мобільність - токен можна використовувати на будь-якому комп'ютері з USB-портом. Підтримка великої кількості додатків IT-безпеки. Токен належить конкретному користувачеві. | Потребує встановлення ПЗ користувача. |

| | | |
|---|---|---|
| Смарт-карти | Високий рівень безпеки. Компактність. Підтримка великої кількості застосунків. | Потребує встановлення ПЗ користувача. Потребує зчитувального пристрою |
| USB-токени з вбудованим чіпом | Високий рівень безпеки. Мобільність. Підтримка великої кількості застосунків. Очевидна приналежність токена користувачеві. | Потребує встановлення ПЗ користувача. |
| OTP токени | Мобільність. Простота в користуванні. Не потребує встановлення ПЗ користувача | Обмежене коло підтримуваних застосунків . Потребує сервера автентифікації. Обмежений час роботи через використання акумулятора. |
| Гібридні токени | Мобільність. Підтримка великої кількості програм. Не вимагає встановлення користувацького програмного забезпечення для використання одноразових паролів (OTP). Право власності на токен очевидне для користувача | . Обмежений час роботи через використання акумулятора.( Крім випадків, коли користувач може замінити її самостійно). |
| Програмні токени | Не потребує апаратного пристрою. | Секретний ключ слабко захищений. Обмежене коло підтримуваних застосунків. Потрібно сервер автентифікації. |

## 1.6 Біометрична автентифікація

Біометрія - це ідентифікація людини за унікальними біологічними характеристиками, властивими лише їй. Тобто, можна сказати, що спочатку були розроблені біометричні технології для точного встановлення особистості людини.

Тому рішення використовувати їх у сфері інформаційної безпеки виглядає цілком логічним. Більше того, цей напрямок розвивається дуже активно. Сьогодні використовується більше десятка різних біометричних характеристик. Більше того, для найбільш поширених з них (відбитки пальців і райдужної оболонки ока) існує безліч сканерів, які в принципі різні.

Тож у користувачів, які вирішили використовувати біометричну ідентифікацію, є з чого вибрати. Біометрична ідентифікація / автентифікація - це метод ідентифікації особи на основі певних конкретних біометричних ознак, властивих конкретній людині. Сучасний рівень розвитку комп’ютерних технологій дозволив використовувати такі ознаки як основу для ідентифікації людини та прийняття рішення про можливість доступу до ресурсів комп’ютерних систем.

Серед біометричних ідентифікаційних механізмів можна виділити наступні:

1) за статичними характеристиками - те, що практично не змінюється з часом, починаючи з народження людини (фізіологічні особливості);

2) за динамічними ознаками - поведінкові характеристики, тобто такі, що побудовані на ознаках, характерних для підсвідомих рухів у процесі відтворення дії.

Динамічні ознаки можуть змінюватися з часом, але не різко, а поступово.

Серед статичних методів у завданнях ідентифікації користувача комп'ютерних систем використовуються:

1. Ідентифікація по відбитку пальця (рис. 1.5). Цей метод заснований на унікальності папілярного малюнка на пальцях рук. Ідентифікація будується так: за допомогою сканера виходить

зображення відбитка пальця, потім це зображення по складним алгоритмом перетворюється в спеціальний цифровий код. Потім цей код порівнюється з довідковими кодами, які зберігаються в базі даних.

Рис. 1.5 Дактилоскопічний сканер

2. Визначення по розташуванню вен на долоні. Пристрій, що зчитує інформацію, в даному випадку - інфрачервона камера. В результаті при формуванні цифрового коду на вході в програму з'являється візерунок з вен на руці людини. Не потребує контакту людини з пристроєм для сканування. Має високі показники надійності і надійності.

3. Ідентифікація по сітківці ока (рис. 1.6). В цьому випадку сканується малюнок кровоносних судин очного дна, який має фіксовану структуру, що не змінюється з плином часу. Зрозуміло, що така закономірність спостерігається тільки при певних умовах: при скануванні людина дивиться на далекий джерело світла і спеціальна камера сканує його очне дно, що в свою чергу може викликати у людини неприємні відчуття. Вважається одним з найнадійніших біометричних методів.

4. Ідентифікація за райдужкою ока (рис. 1.6).

Візерунок райдужки у кожної людини унікальний. У цьому методі важлива не тільки виділена камера, але й надійне програмне забезпечення. Зрештою, саме за допомогою програмного забезпечення

на малюнку виділяється потрібна нам схема райдужки. Цей метод є одним з найбільш точних серед біометричних методів.

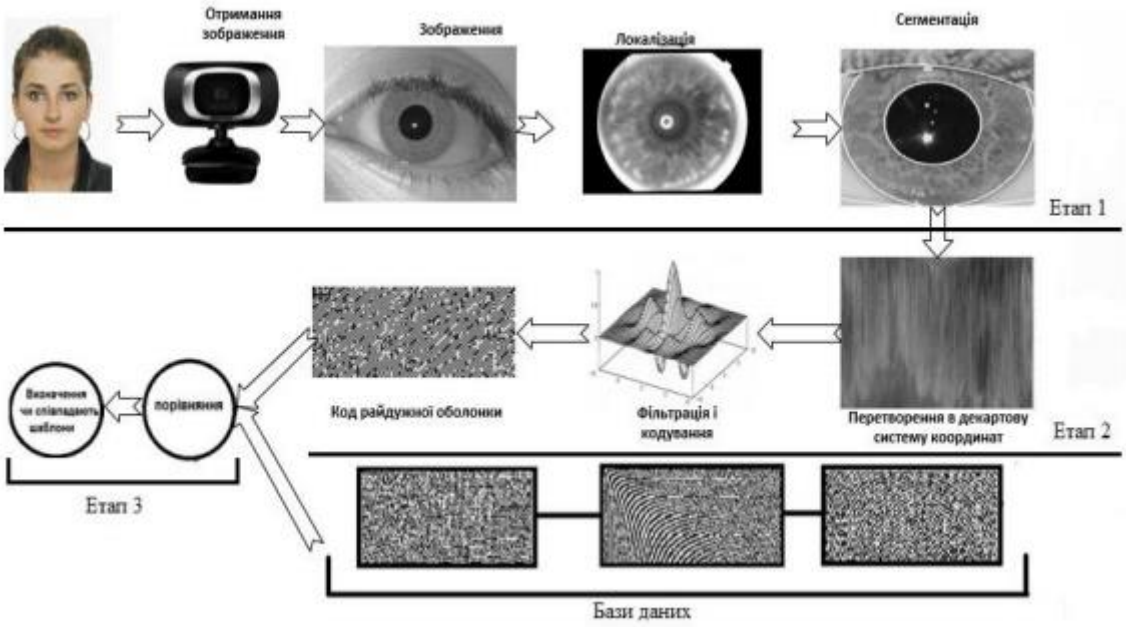



Рис. 1.6 Автентифікація за райдужкою ока

5. Ідентифікація за формою руки. Цей метод заснований на розпізнаванні геометричних особливостей кисті. Спеціальний сканер генерує тривимірний малюнок руки. При аналізі цього показника проводяться вимірювання, за допомогою яких формується відповідний цифровий код.

6. Ідентифікація за формою обличчя (рисунок 1.7).

На практиці використовуються як 2D, так і 3D зображення. Більше того, двовимірне розпізнавання обличчя сьогодні є одним з найбільш неефективних методів біометрії, тому воно має обмежений спектр застосування або використовується лише разом з іншими методами. Розпізнавання за тривимірним зображенням обличчя чимось схоже на метод ідентифікації за формою руки. Тут також побудовано тривимірне зображення обличчя. Спеціальне програмне забезпечення витягує з цього зображення контури очей, губ та інших частин обличчя. Далі проводяться точні вимірювання між зазначеними контурами. Саме з цих даних будується цифровий код.

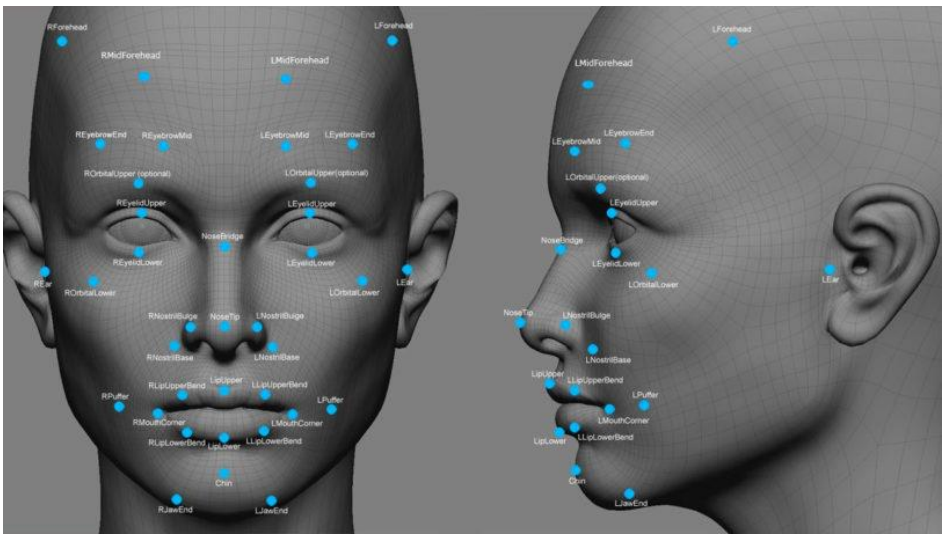



Рис. 1.7 Ідентифікація за формою обличчя

До динамічних методів, що використовуються для ідентифікації користувача, відносяться наступні:

1. Голосова ідентифікація. В даний час існує безліч програм розпізнавання голосу. У методі голосової ідентифікації важливі частотні характеристики людського голосу. Цифрова модель побудована на частотних характеристиках.

2. Ідентифікація почерку. При ідентифікації цим методом, як правило, перевіряється підпис людини. Перевіряються такі динамічні характеристики, як: графічні параметри, сила тиску на поверхню, швидкість підписання підпису. На основі цих характеристик будується цифровий код.

3. Розпізнавання за допомогою клавіатурного почерку. Цей метод схожий на ідентифікацію рукописним способом, але замість підпису людині потрібно ввести кодове слово. Цифровий код базується на динаміці набору слова чи фрази.

При теоретичному різноманітності можливих біометричних методів на практиці мало. Існує три основні методи - розпізнавання відбитків пальців, зображення особи (двомірне або тривимірне), райдужна оболонка ока і сітківка. На сьогоднішній день все біометричні технології є ймовірними і часто цей факт є підставою для критики біометрії.

Важко не погодитися з тим, що біометричні технології надійніше і зручніше тих засобів захисту, які широко використовуються досі. Однак, незважаючи на активну роботу в останні роки з розробки та вдосконалення методів ідентифікації користувачів для контролю доступу до ресурсів інформаційних систем, надійність і стійкість існуючих систем недостатні для сьогоднішніх потреб. Головна перевага біометричних технологій - найвища надійність. І дійсно, всім відомо, що двох людей з однаковими відбитками пальців в природі просто не існує. Однак сьогодні є кілька способів обдурити сканери відбитків пальців.

Наприклад, бажані відбитки пальців можна перенести на плівку або використовувати фотографію пальця зареєстрованого користувача. Однак треба визнати, що сучасні пристрої набагато більш стійкі до такої фальсифікації.

Головний недолік біометричної ідентифікації - вартість обладнання. Адже для кожного комп'ютера, що входить в цю систему, потрібно купувати власний сканер. Звичайно, останнім часом ціни на біометричні пристрої постійно знижуються. Крім того, нещодавно з'явилися миші і клавіатури з вбудованими сканерами відбитків пальців.

У цих системах для визначення особистості користувача використовується тільки один фактор. Однак такі процеси сьогодні не можна назвати надійними.

Останнім часом широкого поширення набула комплексна або багатофакторна ідентифікація.

### 1.7 Багатофакторна автентифікація

Багатофакторну ідентифікацію / автентифікацію не можна виділити в окремій формі. Тим не менш, у таких системах для визначення ідентифікації користувача в ICS використовується відразу кілька параметрів. Причому ці параметри можна комбінувати в будь-якому порядку. Однак сьогодні, у переважній більшості випадків, використовується лише одна пара: захист паролем та маркер. У цьому випадку користувач може не боятися хакера,

який вгадує свій пароль (він не буде працювати без електронного ключа), а також крадіжки токена (він не буде працювати без пароля). У деяких системах використовуються найнадійніші процедури ідентифікації, які одночасно використовують паролі, токени та біометричні характеристики людини. Впровадження комбінованих систем збільшує кількість ідентифікаційних ознак і, отже, підвищує безпеку.

На сьогодні існують комбіновані системи наступних типів:

– системи на базі безконтактних смарт-карт і USB-ключів;

– системи на базі гібридних смарт-карт;

– біоелектронні системи. Безконтактні смарт-карти і USB-ключі.

Антена та мікросхема вбудовані в корпус брелока USB для створення безконтактного інтерфейсу. Це дозволить організувати контроль доступу до приміщень та до комп'ютера за допомогою одного ідентифікатора. Ця схема використання ідентифікатора може виключити ситуацію, коли працівник, залишаючи робоче місце, залишає USB-ключ у роз'ємі комп’ютера, що дозволить працювати під його ідентифікатором. У випадку, коли неможливо вийти з кімнати без використання безконтактного ідентифікатора, цієї ситуації можна уникнути. Сьогодні найпоширенішими двома ідентифікаторами цього типу є:

– RfiKey –компанія Rainbow Technologies;

– eToken PRO RM – компаніяAladdin Software SecurityR.D.

Цифрові підписи, такі як: eToken RM - USB-ключі та смарт-карти eToken PRO, доповнені пасивними RFID-мітками. Технологія RFID (ідентифікація радіочастот, ідентифікація радіочастот) є найпопулярнішою технологією безконтактної ідентифікації сьогодні. Розпізнавання радіочастот здійснюється за допомогою так званих RFID-міток, прикріплених до об'єкта, що несуть ідентифікаційну та іншу інформацію. З сімейства USB-ключів eToken лише eToken PRO / 32K може бути доповнений тегом RFID. Гібридні смарт-карти. Гібридні смарт-карти містять різнорідні чіпи. Один чіп підтримує контактний інтерфейс, інший - безконтактний. Як і у випадку з

гібридними USB-ключами, гібридні смарт-карти вирішують дві проблеми: доступ до кімнати та доступ до комп’ютера. Додатково на картку можна нанести логотип компанії, фотографію працівника або магнітну смужку, що дає змогу повністю замінити звичні місця та перейти до єдиного «електронного пропуску». Смарт-карти цього типу розробляються багатьма компаніями: HID Corporation, Axalto, GemPlus, Indala, Aladdin Knowledge Systems та ін. У Росії Aladdin Software Security RD розроблена технологія виробництва гібридних смарт-карт eToken Pro / SC RM . У них мікросхеми з контактним інтерфейсом eToken Pro вбудовані в безконтактні смарт-карти. Смарт-карти EToken PRO можуть бути доповнені пасивними RFID-мітками, виготовленими HID / ISOProx II, EM-Marine (125 кГц), Cotag (122/66 кГц), Angstrem / Kiba-002 (13,56 МГц), Mifare та іншими компаніями. Вибір варіанту поєднання визначається замовником. Біоелектронні системи (рисунок 1.8). Як правило, для захисту комп'ютерних систем від несанкціонованого доступу використовується комбінація двох систем - біометричної та контактної на основі смарт-карт або USB-ключів. Найчастіше системи розпізнавання відбитків пальців використовуються як біометричні системи.

Рис. 1.8 Багатофакторний біометричний термінал

Коли друк відповідає шаблонам, доступ дозволений. До недоліків цього методу ідентифікації можна віднести можливість використання фіктивного відбитка. Підвищення надійності і точності автоматизованих систем ідентифікації користувачів може бути досягнуто за рахунок поєднання

використання біометричних характеристик разом з класичними методами ідентифікації користувача (наприклад, захист паролем, PIN-кодом, використання різних карт і т. Д.). системи, що використовують кілька біометричних характеристик користувача для прийняття рішення про доступ до інформаційних систем (наприклад, спільне використання функцій клавіатури, рукописного введення, голоси, динаміки користувача з «мишею» або використання декількох відбитків пальців і т. д..). Деякі виробники вже почали інтегрувати два методу розпізнавання осіб, включаючи двох-і тривимірні зображення.

### 1.8 Міжнародні стандарти по криптографічним протоколам ідентифікації/автентифікації

Основним міжнародним стандартом по криптографічним протоколам автентифікації є стандарт Міжнародної організації по стандартизації та Міжнародної електротехнічної комісії ISO / IEC 9798 - Information technology – Security techniques - Entity authentication mechanisms, що складається з п'яти частин:

ISO / IEC 9798-1 - «General Model»;

ISO / EEC 9798-2 - «Mechanisms using symmetric encipherment algorithms »;

ISO / EEC 9798-3 - «Entity authentication using a public-key algorithm »;

ISO / IEC 9798-4 - «Mechanisms using a cryptographic check function»;

ISO / IEC 9798-5 - «Mechanisms using zero knowledge techniques»

У цих протоколах заявник і перевіряючий мають симетричний секретний ключ або парно-виборчі ключі зв'язку. Для їх отримання можна використовувати довірений сервер реального часу.

Стандартом ISO / IEC 9798-2 передбачені три способи автентифікації:

1. Одностороння автентифікація на основі мітки часу. Якщо заявник та рецензент мають системний годинник, подавати запит немає необхідності. заявник може негайно надіслати повідомлення із позначкою часу, включеною до нього, а верифікатор може перевірити позначку часу на показаннях його годинника.

2. Одностороння автентифікація за допомогою випадкових чисел. На запит верифікатор надсилає випадкове число, сформоване за допомогою генератора псевдовипадкових чисел. Отримавши запит, заявник обчислює відповідь на нього - він шифрує отримане випадкове число та (за необхідності) ідентифікатор, використовуючи алгоритм симетричного шифрування. верифікатор розшифровує отриманий зашифрований текст і перевіряє структуру запиту. Якщо це правильно, він приймає заявника.

3. Взаємна автентифікація за допомогою випадкових чисел. Різниця між цим протоколом і попереднім полягає в тому, що тут кожен з учасників по черзі виконує ролі перевіряючого та заявника, що перетворює справжність один на один. Протокол взаємної автентифікації - це, по суті, два односторонні протоколи автентифікації, «упаковані» у три передачі повідомлень. Через їх симетрію такі протоколи називаються протоколами рукостискання. цей протокол дозволяє замінити шифр на хеш-функцію на ключ, як зазначено у стандарті ISO / IEC 9798-4. Для підвищення надійності протоколу до переданого повідомлення можна додати мітки часу.

4. Протоколи «запит - відповідь» з використанням асиметричних криптосхем. Такі протоколи можна розділити на дві групи: протоколи з використанням ЕЦП і протоколи з використанням відкритого шифрування. Стандарт ISO / IEC9798-3 рекомендує:

1) Протоколи, що використовують схеми цифрового підпису

В описах протоколів містяться сертифікати сертифікатів відкритих ключів для цифрового підпису відповідних учасників протоколу, тобто структури даних, що містять їх ідентифікатори, відкриті ключі та іншу службову інформацію, засвідчені цифровим підписом сертифікаційний орган. Спосіб сертифікації відкритих ключів буде розглянуто більш докладно далі. Протоколи, подібні до описаних вище, також використовуються в стандарті

MCE X.509. Він описує протоколи автентифікації, які зведені за протоколами обміну ключами.

2) Протоколи із застосуванням схем типу зашифрованого шифрування

Погрози безпеки ICS можна розділити на мережеві атаки (інформація, що надходить від віддаленого клієнта) та локальні атаки, які походять від шкідливого програмного забезпечення, вже встановленого в системі клієнта, наприклад, троянських програм, руткітів тощо. Часто оцінки безпеки автентифікації зосереджуються насамперед на мережевих атаках, припускаючи, що користувальницький термінал (тобто настільний комп'ютер, ноутбук або мобільний пристрій) є захищеною платформою. Однак нерідкі випадки, коли зловмисник отримує повний доступ до ПК жертви за допомогою прихованих комунікаційних процесів, що залишилися від шкідливого програмного забезпечення, яке використовує недопрацьовані діри в безпеці в ліцензійному програмному забезпеченні.

Типовими методами атак на протоколи автентифікації є:

1. Самозванство(impersonation)- один користувач намагається видати себе за іншого.

2. Повторна передача (a replay attack) - повторна передача колишніх автентифікаційних даних будь-яким користувачем.

3. Підміна боку автентифікаційного обміну (Interleaving attack) - зловмисник в ході атаки має можливість модифікувати проходячий через нього трафік.

4. Відображення передачі (reflection attack) - один з варіантів атаки підміни, коли в рамках даного сенсу зловмисник пересилає назад перехоплену інформацію.

5. Вимушена затримка (forced delay) - зловмисник перехоплює деяку інформацію і передає її через деякий час.

6. Атака з вибіркою тексту (chosen-text attack) - зловмисник перехоплює трафік і намагається отримати інформацію про довготривалі ключі.

## 1.9 Порівняння способів автентифікації

Аналіз методів автентифікації, наведений у таблиці 2, дозволяє порівняти вибір методу з точки зору рівня безпеки та з точки зору витрат на реалізацію. Одним з найбільш безпечних рішень є використання апаратного криптографічного маркера або біометричної автентифікації (відбиток пальця, сітківка ока). Однак реалізація деяких рішень цього класу пов'язана з великими витратами і не використовується широко. Найбільш безпечним і в той же час найменшим методом автентифікації, що використовується у веб-додатках, є автентифікація за допомогою комбінації довготривалого і одноразового паролів.

Табл. 2 Приклади реалізації способів автентифікації

| Секрет | Фактори автентифікації | Витрати на реалізацію | Рівень захищеності |
|---|---|---|---|
| Довготривалий пароль Фактор знання Дуже низькі Низький | Фактор знання Дуже низькі Низький | Дуже низькі | Низький |
| Одноразовий пароль, отриманий по SMS | Фактор володіння | Середні | Середній |
| Довготривалий пароль і одноразовий пароль, отриманий по SMS | Фактор знання і фактор володіння | Середні | Високий |
| Відбиток пальця | Фактор властивості суб'єкта | Високі | Високі |
| Відбиток пальця і одноразовий пароль, отриманий по SMS | Фактор володіння і фактор властивості суб'єкта | Високі | Дуже високий |

## 1.10 Типи ідентифікаторів

Cookie

Після успішної автентифікації клієнту надсилається ідентифікатор сеансу клієнта - фрагмент даних. Клієнтська частина кожного запиту на сервер надсилатиме ідентифікатор для автентифікації та авторизації в системі.

Одним із типів ідентифікатора є Cookie - невеликий фрагмент даних, що надсилається сервером і зберігається на комп'ютері користувача. Кожного разу, коли клієнт (веб-браузер) намагається відкрити сторінку відповідного сайту, він надсилає цю частину даних на сервер як частину запиту HTTP. Він використовується для збереження даних на стороні користувача, включаючи дані автентифікації.

Файли cookie для кожного автентифікованого суб’єкта зберігаються на задній частині системи (як правило, в базі даних) та перевіряються на відповідність запитам. Якщо файлb cookie неправильно налаштовано, автентифікація має такі недоліки:

1. Низька захищеність від атаки XSS: якщо система має вразливість XSS, то зловмисник може викрасти Cookie.

2. Низька захищеність від атаки CSRF: зловмисник міг

ініціювати дії від імені користувача без крадіжки файлів cookie.

Токен

Токен - це частина даних, яку створює сервер і передає клієнту, який згодом використовує цей маркер для підтвердження своєї ідентичності. Маркер може містити дані про тему, які можна зашифрувати, а також криптографічний підпис сервера для перевірки цілісності. Цей підхід має такі недоліки:

Низька захищеність від атаки XSS

Порівняння типів ідентифікаторів

При реалізації ідентифікатора типу токен, на нього можуть бути записані допоміжні дані як для клієнтської сторони, так і для серверної. Однак на клієнтській стороні маркер зберігається в незахищеному сховищі у браузері, коли файли cookie можуть бути захищені від читання з клієнтської сторони програми за допомогою спеціальних прапорів, що зменшує ризик витоку

ідентифікатора, і в той же час виключає можливість додавання додаткових даних до ідентифікатора.

Існуючі системи автентифікації і авторизації

PassportJS

PassportJS - проміжне програмне забезпечення для платформи NodeJS, що реалізує різні способи автентифікації на стороні сервера, звані «стратегіями», наприклад, такі як:

1. Автентифікація через сторонні сервіси за допомогою протоколу
2. Автентифікація за допомогою токенів.
3. Автентифікація за логіном і паролем.

Недоліки цього підходу:

1. Відсутня можливість настройки багатофакторної автентифікації.
2. Не забезпечуються механізми захисту ідентифікатора сесії на клієнтській стороні.
3. Багато «стратегії» уразливі до атак.

Auzy

Auzy - проміжне програмне забезпечення для платформи NodeJS, що реалізує доступ до системи за допомогою сесій і дозволяє робити настроювання параметрів за допомогою конфігураційного файлу, а саме:

1. Ім'я заголовка для зберігання ідентифікатора сесії.
2. Час життя ідентифікатора сесії.

Недоліки цього підходу:

1. Відсутня можливість настройки багатофакторної автентифікації.
2. Не забезпечуються механізми захисту ідентифікатора сесії на клієнтській стороні.

Обрані для реалізації механізми автентифікації

На підставі порівняльного аналізу розглянутих рішень в роботі був запропонований підхід, який використовує наступний способи автентифікації:

1. Автентифікація за довгостроковим паролем.

2. Автентифікація за довгостроковим паролем і одноразовим паролем, що приходить по SMS.

3. Автентифікація за довгостроковим паролем і одноразовим паролем, отриманим за допомогою алгоритму TOTP через додаток Google Authenticator.

Крім того, в підході для забезпечення безпечного зберігання ідентифікатора сесії було прийнято рішення використовувати випадково згенеровані Cookie з подальшим збереженням їх в базі даних і додатковими параметрами, а саме:

1. З додаванням прапора HttpOnly - прапор, який забороняє зловмисникові прочитати значення Cookie, реалізувавши атаку XSS.

2. З додаванням прапора Secure - прапор, який забороняє встановлювати браузеру Cookie в HTTP-запит, чи не захищений протоколом HTTPS для захисту від атаки «людини по середині».

З додаванням заголовка SameSite = Strict - заголовок, який забороняє відправляти Cookie в запиті з іншого ресурсу, що є захистом від атаки CSRF.

# РОЗДІЛ 2

# Розробка системи багатофакторної автентифікації та розмежування доступу

## 2.1 Опис системи багатофакторної автентифікації та розмежування доступу

Автентифікація за довгостроковим паролем і одноразовим паролем з програми Google Authenticator

Компоненти алгоритму

У кожного суб'єкта в системі існує свій логін - персональне ім'я входу, довготривалий пароль - послідовність символів і заздалегідь заданий секрет, за допомогою якого на основі часу кожні 30 секунд генерується новий одноразовий пароль (time-based one-time password). Даний секрет записується в додаток Google Authenticator, а додаток відображає поточний пароль.

## 2.2 Алгоритм

Алгоритм автентифікації за довгостроковим паролем і одноразовим паролем, отриманим за допомогою програми Google Authenticator виглядає наступним чином:

1. Ввод логіну і довгострокового паролю на клієнтській частині.

2. Клієнтська частина відправляє дані, введені суб'єктом, на серверну частину.

3. Серверна частина обробляє запит і перевіряє справжність введеного логіна і довготривалого пароля.

4. Якщо введені дані вірні, то перехід на крок 5.

Інакше, на клієнтську частину відправляється відповідь з маркером помилки.

5. Серверна частина відправляє на клієнтську частину повідомлення: "One-time password required"

6. Ввод одноразового паролю з програми Google

Authenticator на клієнтській частині.

7. Клієнтська частина відправляє одноразовий пароль, логін і довготривалий пароль на серверну частину.

8. Серверна частина обробляє запит і перевіряє справжність введеного логіна, довготривалого пароля і одноразового пароля за допомогою секрету.

9. Якщо отриманий одноразовий пароль не збігається зі згенерованим за допомогою секрету одноразовим паролем, то перехід на крок 10.Інакше, перехід на крок 11.

10. Серверна частина відправляє на клієнтську частину маркер помилки і повідомлення: "Incorrect code"

11. Серверна частина генерує ідентифікатор сесії, зберігає його в базі даних і відправляє на клієнтську частину.

12. Клієнтська частина отримує відповідь від серверної частини і, якщо немає помилки, то встановлює секрет в наступні запити на серверну частину.

**2.3 Програмна реалізація**

Мова і засоби розробки

Як мова розробки програми була обрана платформа мови програмування JavaScript NodeJS, а в якості фреймворка для запуску веб-сервера був обраний модуль Express. Цей вибір обгрунтований наступними факторами:

1. Кросплатформеність, що дозволяє використовувати модуль в продуктах на різних ОС.

2. Наявність безлічі модулів (бібліотек), що дозволяють зменшити кількість вихідного коду і час розробки;

Як середовище розробки був обраний інструмент Visual Studio Code IDE 1.40.2, який є найбільш актуальною версією програми на момент написання роботи. Основною перевагою даного середовища є підтримка безлічі плагінів для зручної роботи, а також вільний доступ.

**2.4 Ресурси**

1. Node JS (LTS)

2. Google Authenticator

Після установки перерахованих вище ресурсів займемося створенням API для додатка.

Порядок розробки

Крок 1: серверний додаток

Для створення API-сервісів ми будемо використовувати невеликий фреймворк для Node.js, який називається Express.js. Створимо папку 'back-end' для нашого серверного додатка. Потім перейдемо в неї в терміналі командного рядка і встановимо необхідні залежності.

```
> mkdir back-end
> cd back-end
> npm init -y
> npm install --save express body-parser cors qrcode speakeasy
```

Ми створили папку 'back-end' і ініціалізували проект Node.js, встановивши наступні залежності:

1. express - це невеликий настроюється фреймворк для створення API сервісів.

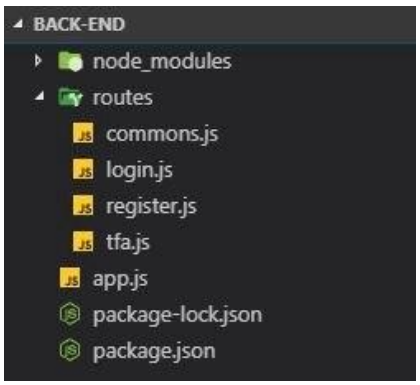
2. body-parser - для аналізу методів HTTP.

3. cors - пакет використовується для інтеграції клієнтської частини веб-додатки з API сервісами.

4. qrcode - відповідає за генерацію QR-код у вигляді зображень base64.

5. speakeasy - генератор секретних ключів за алгоритмом T-OTP, який використовує Google Authenticator.

Тепер створимо кілька API-сервісів, в яких головним виконуваним файлом буде app.js. Для спрощення вивчення матеріалу ми опустимо код, який відповідає за взаємодію з базою даних.




Структура папок для серверної частини

API-сервіси реалізують функціонал входу в систему, реєстрації та TFA (двохфакторну автентифікацію):

Сервіс входу в систему

Включає в себе базову функціональність для входу за допомогою логіна, пароля і коду автентифікації.

```
import { Component, OnInit } from '@angular/core';
import { LoginServiceService } from 'src/app/services/login-service/login-service.service';

@Component({
const express = require('express');
const speakeasy = require('speakeasy');
const commons = require('./commons');
const router = express.Router();

router.post('/login', (req, res) => {
    console.log(`DEBUG: Received login request`);

    if (commons.userObject.uname && commons.userObject.upass) {
        if (!commons.userObject.tfa || !commons.userObject.tfa.secret) {
            if (req.body.uname == commons.userObject.uname && req.body.upass == commons.userObject.upass) {
                console.log(`DEBUG: Login without TFA is successful`);

                return res.send({
                    "status": 200,
                    "message": "success"
                });
            }
            console.log(`ERROR: Login without TFA is not successful`);

            return res.send({
                "status": 403,
                "message": "Invalid username or password"
            });

        } else {
            if (req.body.uname != commons.userObject.uname || req.body.upass != commons.userObject.upass) {
                console.log(`ERROR: Login with TFA is not successful`);

                return res.send({
                    "status": 403,
                    "message": "Invalid username or password"
                });
            }
            if (!req.headers['x-tfa']) {
                console.log(`WARNING: Login was partial without TFA header`);

                return res.send({
                    "status": 206,
                    "message": "Please enter the Auth Code"
```

```
                });
            }
            let isVerified = speakeasy.totp.verify({
                secret: commons.userObject.tfa.secret,
                encoding: 'base32',
                token: req.headers['x-tfa']
            });

            if (isVerified) {
                console.log(`DEBUG: Login with TFA is verified to be successful`);

                return res.send({
                    "status": 200,
                    "message": "success"
                });
            } else {
                console.log(`ERROR: Invalid AUTH code`);

                return res.send({
                    "status": 206,
                    "message": "Invalid Auth Code"
                });
            }
        }
    }

    return res.send({
        "status": 404,
        "message": "Please register to login"
    });
});

module.exports = router;
```

Не будемо використовувати базу даних для зберігання даних користувачів. Тому реалізуємо це на стороні сервера.

```
let userObject = {};

module.exports = {
    userObject
}
```

Сервіс реєстрації

Реєстрація користувача в додатку буде полягати в додаванні логіна і пароля в об'єкт userObject. А також у видаленні існуючої в ньому інформації. Модулі входу і реєстрації створювалися винятково для демонстрації, тому додаток буде підтримувати тільки одного користувача.

```
const commons = require('./commons');
const router = express.Router();

router.post('/register', (req, res) => {
    console.log(`DEBUG: Received request to register user`);

    const result = req.body;

    if ((!result.uname && !result.upass) || (result.uname.trim() == "" || result.upass.trim() == "")) {
        return res.send({
            "status": 400,
            "message": "Username/ password is required"
        });
    }

    commons.userObject.uname = result.uname;
    commons.userObject.upass = result.upass;
    delete commons.userObject.tfa;

    return res.send({
        "status": 200,
        "message": "User is successfully registered"
    });
});

module.exports = router;
```

сервіс TFA

Сервіс призначений для реалізації двохфакторної автентифікації поряд з верифікацією коду T-OTP, згенерованого Google Authenticator. Він буде включати в себе функціональність для отримання налаштувань TFA, а також включення або відключення TFA для userObject.

```
const express = require('express');
const speakeasy = require('speakeasy');
const QRCode = require('qrcode');
const commons = require('./commons');
const router = express.Router();

router.post('/tfa/setup', (req, res) => {
    console.log(`DEBUG: Received TFA setup request`);

    const secret = speakeasy.generateSecret({
        length: 10,
        name: commons.userObject.uname,
        issuer: 'NarenAuth v0.0'
    });
    var url = speakeasy.otpauthURL({
        secret: secret.base32,
        label: commons.userObject.uname,
        issuer: 'NarenAuth v0.0',
        encoding: 'base32'
    });

    QRCode.toDataURL(url, (err, dataURL) => {
        commons.userObject.tfa = {
            secret: '',
            tempSecret: secret.base32,
            dataURL,
            tfaURL: url
        };
        return res.json({
            message: 'TFA Auth needs to be verified',
            tempSecret: secret.base32,
            dataURL,
            tfaURL: secret.otpauth_url
        });
    });
});

router.get('/tfa/setup', (req, res) => {
    console.log(`DEBUG: Received FETCH TFA request`);

    res.json(commons.userObject.tfa ? commons.userObject.tfa : null);
});
```

```
router.delete('/tfa/setup', (req, res) => {
    console.log(`DEBUG: Received DELETE TFA request`);

    delete commons.userObject.tfa;
    res.send({
        "status": 200,
        "message": "success"
    });
});

router.post('/tfa/verify', (req, res) => {
    console.log(`DEBUG: Received TFA Verify request`);

    let isVerified = speakeasy.totp.verify({
        secret: commons.userObject.tfa.tempSecret,
        encoding: 'base32',
        token: req.body.token
    });

    if (isVerified) {
        console.log(`DEBUG: TFA is verified to be enabled`);

        commons.userObject.tfa.secret = commons.userObject.tfa.tempSecret;
        return res.send({
            "status": 200,
            "message": "Two-factor Auth is enabled successfully"
        });
    }

    console.log(`ERROR: TFA is verified to be wrong`);

    return res.send({
        "status": 403,
        "message": "Invalid Auth Code, verification failed. Please verify the system Date and Time"
    });
});

module.exports = router;
```

Згадані вище сервіси включені в один виконуваний файл 'app.js', розташований в кореневій папці. Цей код запустить HTTP-сервер, створений за допомогою express.js на локальному хості з портом 3000.

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const cors = require('cors');
const login = require('./routes/login');
const register = require('./routes/register');
const tfa = require('./routes/tfa');

app.use(bodyParser.json());
app.use(cors());

app.use(login);
app.use(register);
app.use(tfa);

app.listen('3000', () => {
    console.log('The server started running on http://localhost:3000');
```

Ми реалізували серверну частину коду веб-додатки.

```
> ng serve

** Angular Live Development Server is listening on localhost:4200,
```

Наступним кроком буде створення простого додатка, що використовує ці сервіси на Angular 7.

Крок 2: додаток на базі Angular 7

Спочатку потрібно встановити Angular. Після цього ми створимо додаток з назвою 'front-end' і встановимо залежність від 'bootstrap' (посилання на bootstrap.min.css в styles.css), перейшовши в папку front-end.

```
> npm install -g @angular/cli
> ng new front-end
> cd front-end
> npm install --save bootstrap
> ng serve
```

Після цього створимо кілька компонентів і сервісів, які потрібні додатком. Для цілей демонстрації ми створимо LoginService і два guards - 'Auth Guard' і 'Login Guard.'

```
> ng g s services/login-service/login-service --spec=false
> ng g g guards/AuthGuard
> ng g g guards/Login
```

Guards, які ми створюємо, відносяться до типу CanActivate. Сервіс входу в систему буде включати в себе HTTP-запити до сервісів, створеним на стороні сервера.

AuthGuard обмежить навігацію користувача тільки домашньою сторінкою, якщо той не увійшов в систему.

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';
import { LoginServiceService } from 'src/app/services/login-service/login-service.service';

@Injectable({
  providedIn: 'root'
})
export class AuthGuardGuard implements CanActivate {

  constructor(private _loginService: LoginServiceService, private _router: Router) {
  }
  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    if (this._loginService.getAuthStatus()) {
      return true;
    }

    this._router.navigate(['/login'])

    return false;
  }
}
```

LoginGuard не дозволить користувачеві заходити на сторінку авторизації, якщо користувач вже увійшов в систему.

```
import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';
import { LoginServiceService } from 'src/app/services/login-service/login-service.service';

@Injectable({
  providedIn: 'root'
})
export class LoginGuard implements CanActivate {
  constructor(private _loginService: LoginServiceService, private _router: Router) {
  }
  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot) {
    if (!this._loginService.getAuthStatus()) {
      return true;
    }

    this._router.navigate(['/home'])
    return false;
  }

}
```

Ми завершили створення основи для нашого застосування, створивши служби і guards. Тепер розробимо кілька компонентів.

```
> ng g c components/header --spec=false
> ng g c components/home --spec=false
> ng g c components/login --spec=false
> ng g c components/register --spec=false
```

Після створення необхідних компонентів додатка ми налаштуємо маршрутизацію для додатка, зв'язавши відповідні guards для активації маршрутів. Також, видалимо код, доданий за замовчуванням в app.component.html, і вставимо компонент загального заголовка і висновок маршрутизатора.

```
<app-header></app-header>
<router-outlet></router-outlet>
```

Компонент заголовку

Це загальний компонент для інших компонентів, який включає в себе панель навігації програми. Видимість посилань в заголовку контролюється методом getAuthStatus () сервісу LoginService. У фоновому режимі запросимо файл * .ts для компонента заголовка.

Компонент входу в систему

Призначений для отримання логіну, паролю та коду AuthCode (якщо включений TFA) від користувача і його перевірки на стороні сервера. Якщо

дані вірні, то користувач буде переміщений в HomeComponent. Ми також будемо перевіряти статус, отриманий від серверної частини коду, для відображення повідомлень користувачеві.

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router'
import { LoginServiceService } from 'src/app/services/login-service/login-service.service';

@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.css']
})
export class LoginComponent implements OnInit {

  tfaFlag: boolean = false
  userObject = {
    uname: "",
    upass: ""
  }
  errorMessage: string = null
  constructor(private _loginService: LoginServiceService, private _router: Router) {
  }

  ngOnInit() {
  }

  loginUser() {
    this._loginService.loginAuth(this.userObject).subscribe((data) => {
      this.errorMessage = null;
      if (data.body['status'] === 200) {
        this._loginService.updateAuthStatus(true);
        this._router.navigateByUrl('/home');
      }
      if (data.body['status'] === 206) {
        this.tfaFlag = true;
      }
      if (data.body['status'] === 403) {
        this.errorMessage = data.body['message'];
      }
      if (data.body['status'] === 404) {
        this.errorMessage = data.body['message'];
      }
    })
  }
}
```

Компонент реєстрації

Ми зареєструємо одного користувача в усьому додатку.Якщо загубиться логін і пароль для додатка, або секретний ключ TFA, просто потр ввести нове ім'я користувача та пароль на сторінці.

Сторінка входу і реєстрації

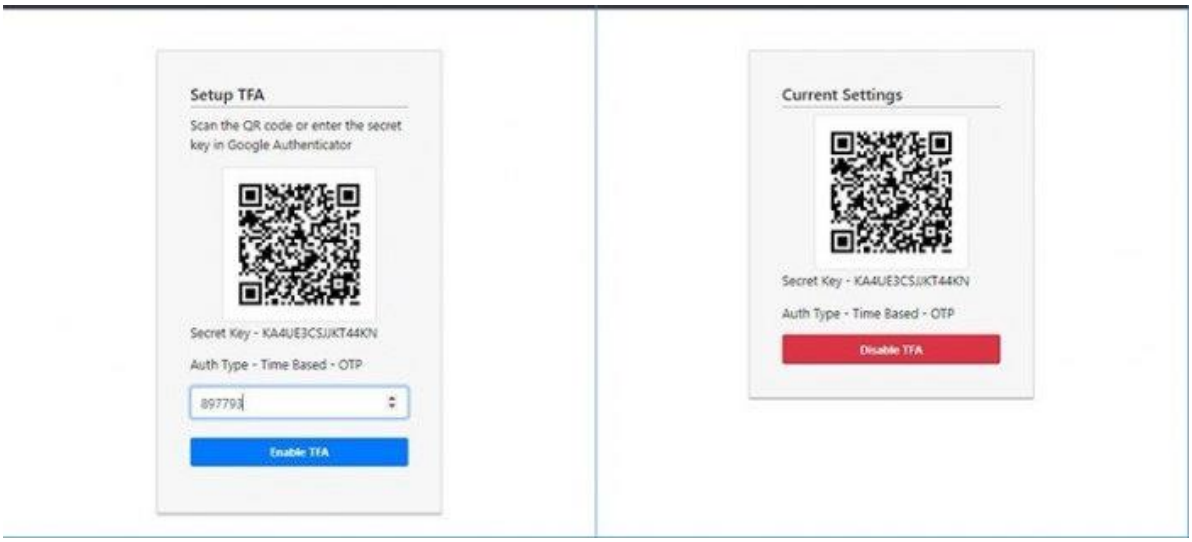
Як тільки користувач зареєструвався і увійшов в систему з логіном і паролем, йому буде надана можливість включення і відключення двохфакторної автентифікації в HomeComponent.

Компонент Home

Дозволить користувачеві налаштовувати і перевіряти TFA. Як тільки користувач потрапить на цю сторінку, він зможе відсканувати QR-код в

додатку Google Authenticator. Після сканування T-OTP (елемент TFA), пов'язаний з userObject, буде включений в додаток Google Authenticator. AuthCode буде відображатися в додатку на тимчасовій основі. Такий самий пароль необхідний для перевірки і включення TFA для userObject.

Якщо користувач увімкнув TFA, то відобразяться поточні настройки з QR-кодом і секретним ключем. А також опція відключення TFA, пов'язаного з userObject.




Установка і поточні налаштування TFA

Функції тимчасової роботи (middleware) - це функції, які мають доступ до об'єкта запиту (request), об'єкту відповіді (response) і до наступної функції тимчасової роботи в циклі "запит-відповідь" додатка. Наступна функція тимчасової роботи, як правило, позначається змінної next.

Функції тимчасової роботи можуть виконувати такі завдання:

1. Виконання будь-яких команд.
2. Внесення змін до об'єкти запитів і відповідей.
3. Завершення циклу "запит-відповідь".
4. Виклик наступної функції тимчасової роботи з стека функцій.

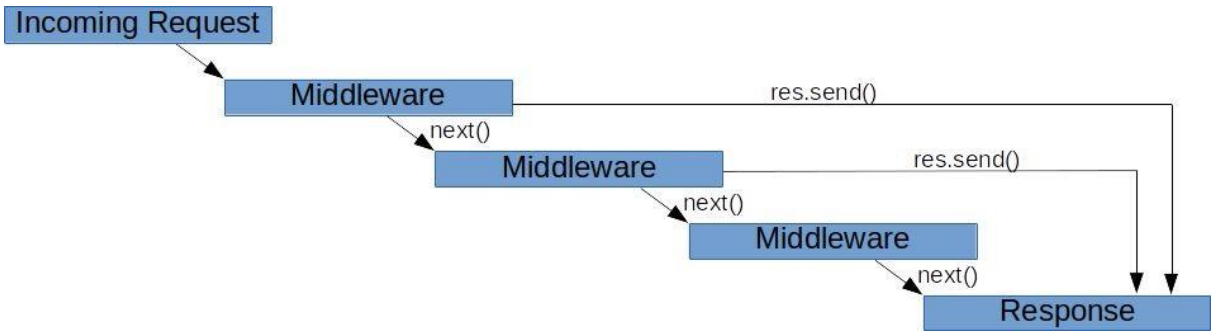



Рис. 2.1 Стек функцій тимчасової роботи сервера Express

**2.5 Інструкція**

У реалізації модуля автентифікації і авторизації передбачена настройка користувачем функцій модуля. У модулі заздалегідь оголошений об'єкт структури конфігураційного файлу і правила, за якими він повинен бути визначений розробником:

Тип глобальної автентифікації (поле authType, тип "String"):

1. "login-password" - автентифікація за логіном і довготривалого паролю.

2. "login-password-ga-otp" - автентифікація за логіном, довготривалого паролю і одноразовим паролем, отриманим за допомогою програми Google Authenticator.

Шляхи (routes) для запитів на сервер (поле routes, тип "Object"), мають структуру "/ {route}":

1. login - шлях для автентифікації в системі (тип "String" / "Array").

2. logout - шлях для закриття сесії в системі (тип "String" / "Array").

3. insecure - масив шляхів, які не потребують автентифікації (тип "String" / "Array").

4. secureFactor - масив шляхів, що вимагають додаткової перевірки фактора автентифікації (тип "Array").

Інші, не зазначені тут, шляхи перевірятимуть ідентифікатор сесії користувача.

Функції для доступу до записів користувачів (поле user, тип "Object"): 1. getLoginData (login) - функція, яка приймає змінну login, рівну логіну користувача в системі і повертає об'єкт, з даними користувача

{Id, login, password, otpsalt}

2. checkPassword (inputPassword, dbPassword) - функція, яка приймає введений пароль користувачем inputPassword і пароль з бази даних dbPassword, і порівнює їх. Повертає true або false.

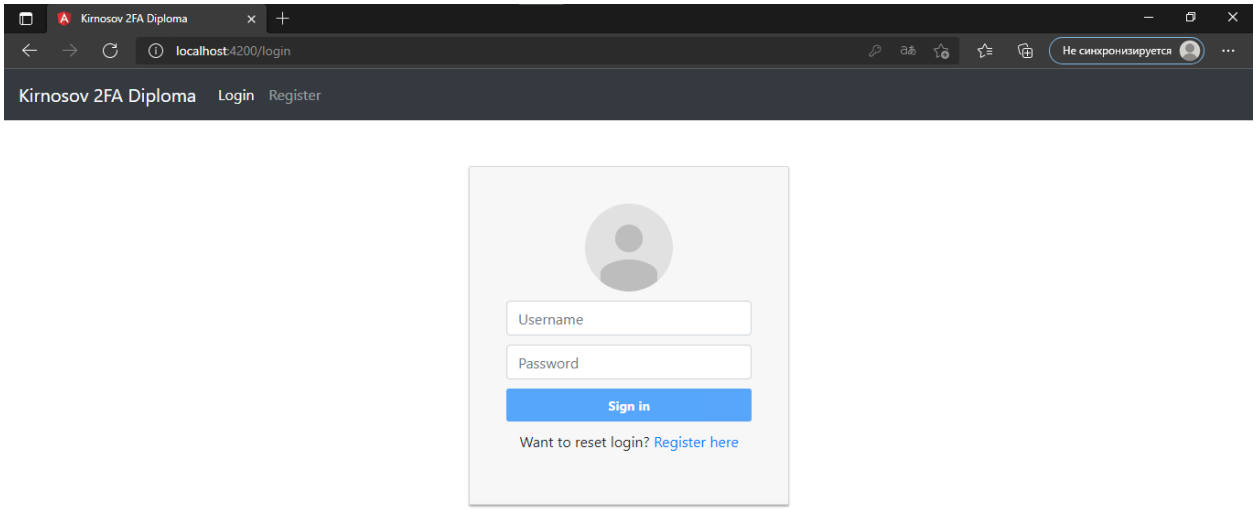
Далі для перегляду сесій користувачів (поле session, тип
"Object"):
1. ttl - час життя сесії користувача (тип "String").
2. timesToCheck - максимальна кількість перевірок одноразового пароля (тип "Number").
3. get (token) - функція, яка приймає ідентифікатор сесії
(Секрет) і повертає об'єкт для даної сесії:

{Id, token}

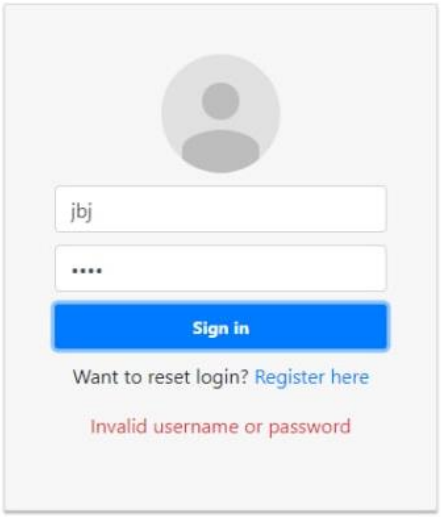
4. upsert ({id, token}) - функція, яка приймає об'єкт з полями
id і token, створює новий запис в таблиці сесій.
5. delete (id) - функція, яка по id сесії видаляє запис з таблиці сесій.
6. getOtpSalt (id) - функція, яка приймає id користувача і повертає об'єкт з секретом для автентифікації за допомогою
додатки Google Authenticator:{Otpsalt}

## 2.6 Тестування програмного додатку

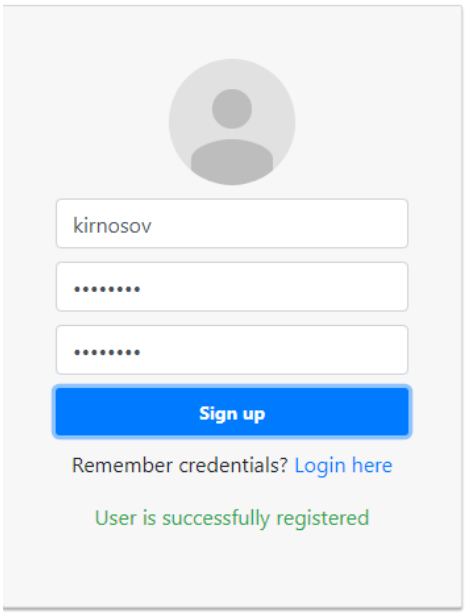
Вмикаємо хост локального сервера та підключаємось до веб-додатку через стандартний браузер комп'ютера. Підключившись, бачимо інтерактивний інтерфейс додатку, за допомогою якого в нас є можливість пройти автентифікацію (login), або зареєструватись.



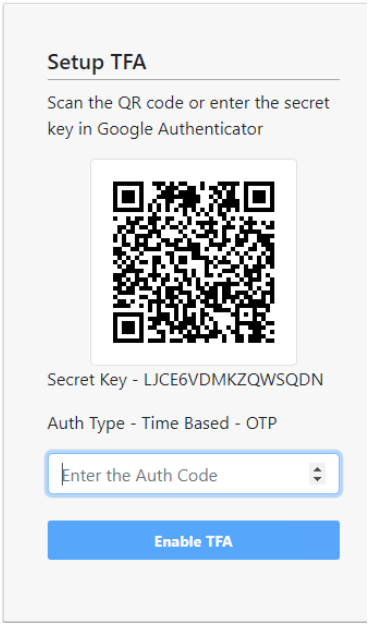

Спробуємо одразу перевірити працездатність системи, набравши довільні дані у віконцях логіну та паролю. Отримуємо результат:



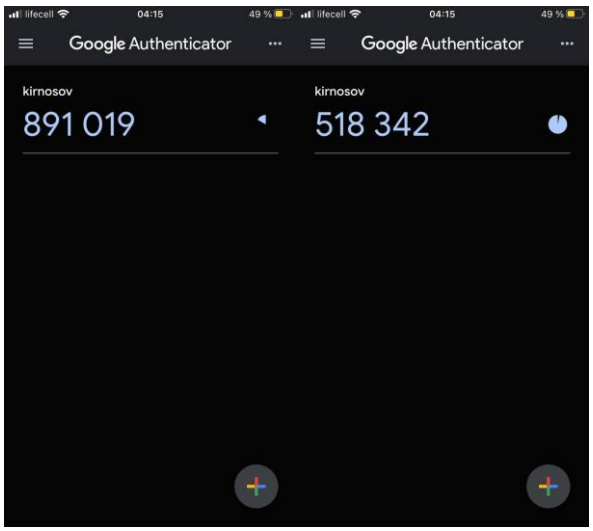

Переходимо до процесу реєстрації, переходимо через посилання «Register here», бачимо, що увімкнулося вікно реєстрації. Вводимо свої дані, а саме логін, пароль та підтвердження пароля, після чого система сповіщає нас, про успіх операції.




Переходимо до авторизації та вносимо свої дані. Після введення правильного логіну та паролю система переходить до налаштування другого фактору автентифікації, а саме одноразового паролю, що генерується за допомогою додатка на смартфон «Google Authenticator».

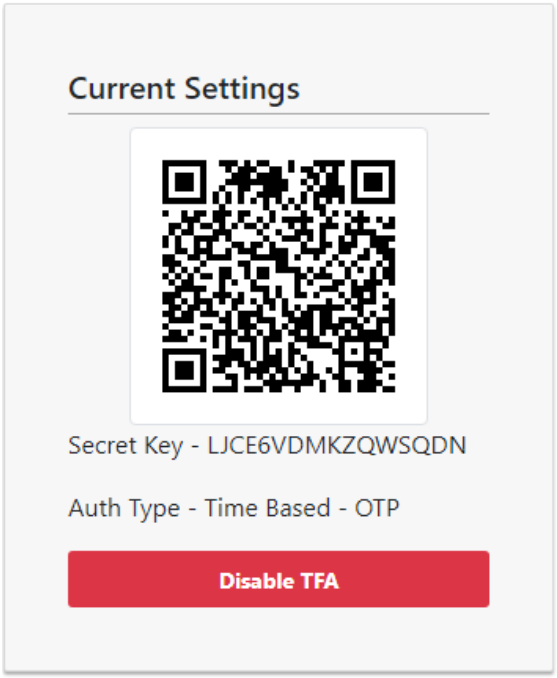



Вводимо секретний ключ, або скануємо щойно згенерований QR-код в додаток на свій смартфон та вносимо одноразовий пароль, що генерується кожні 30 секунд у свій браузер.

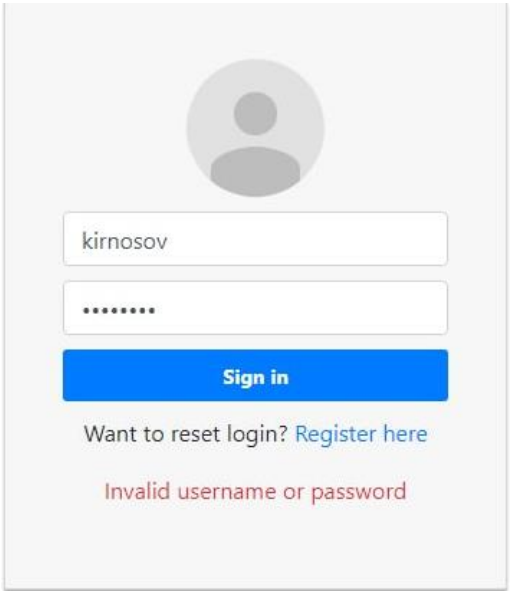



Після внесення секретного ключа, одноразовий пароль для авторизації цього облікового запису тепер завжди буде доступний на смартфоні.

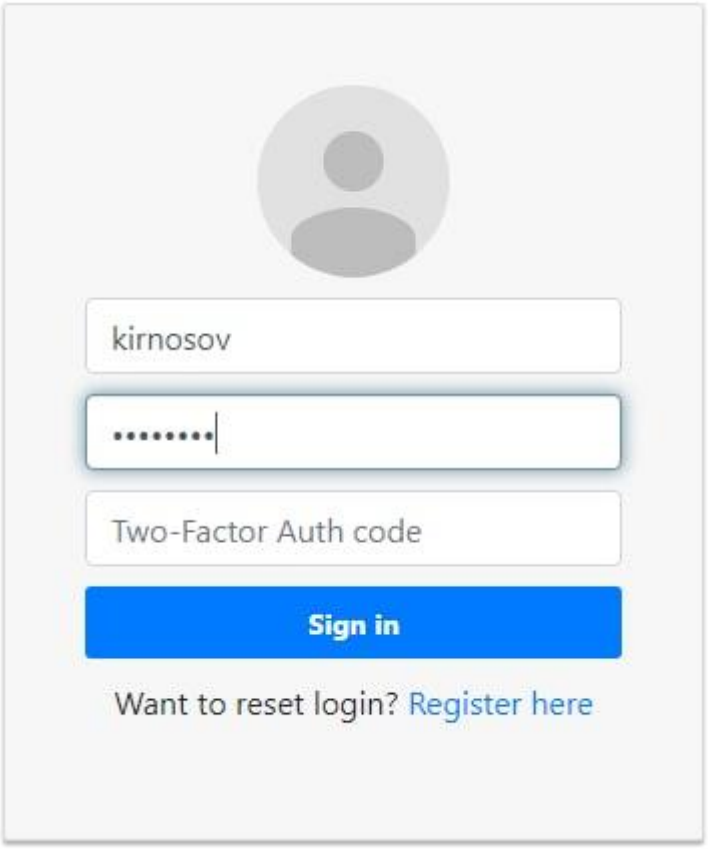
Після налаштування одноразового паролю для нашого облікового запису ми отримуємо доступ до системи, а також можливість відключити другий фактор автентифікації до акаунту.



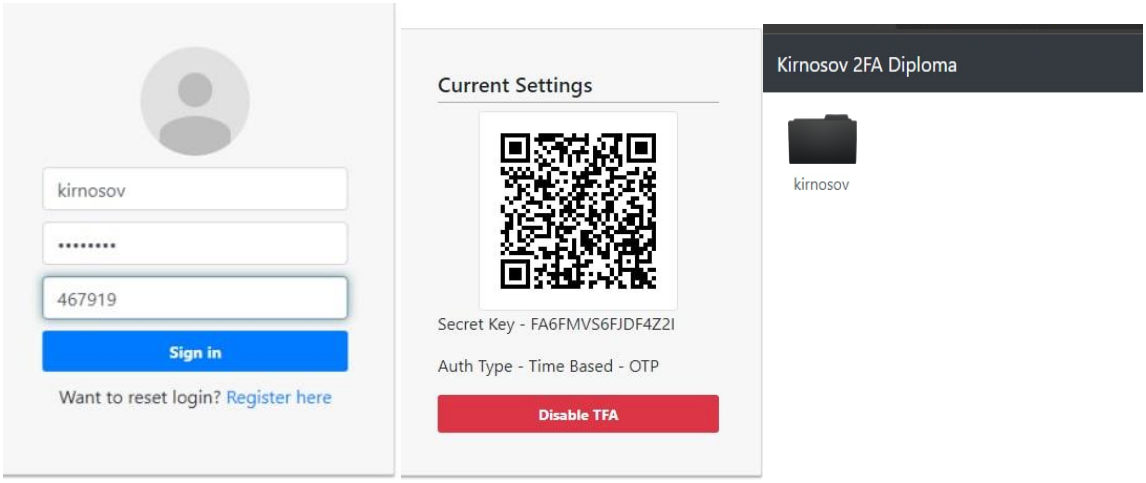

Спробуємо вийти з облікового запису та зайти ще раз, щоб перевірити як працює двохфакторна автентифікація на вже налаштованому обліковому записі. З метою перевірки вводимо логін та довільний пароль.



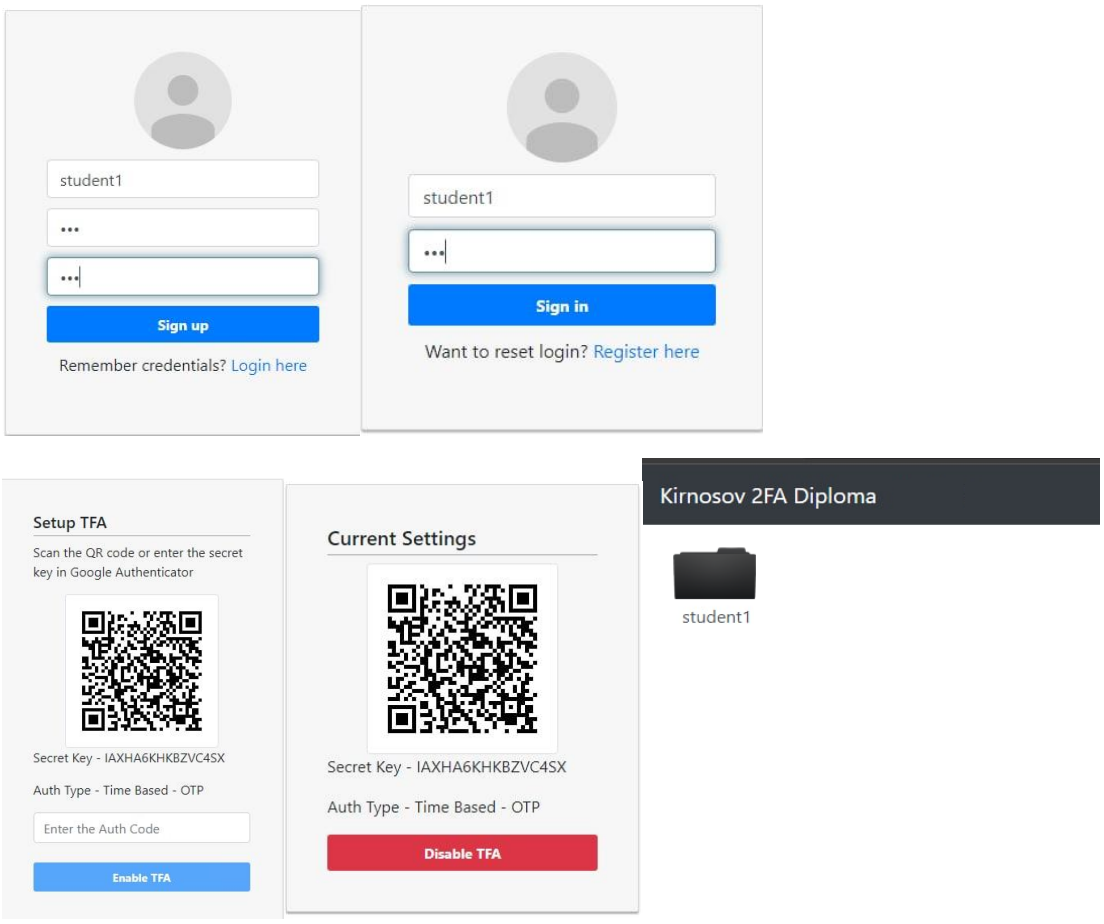

Вводимо відомий нам пароль, на що система в цей раз вже вимагає код.




Спробуємо ввести довільний код, щоб перевірити працездатність системи, відгуку від системи на нього немає. Вводимо код, згенерований мобільним додатком «Google Authentificator» та знову отримуємо доступ.




Також, спробуємо зареєструвати та авторизувати другий акаунт з метою перевірки системи.




Отримуємо доступ та переконуємося, що всі функції Веб-додатку розмежування доступу з подвійною автентифікацію за допомогою одноразового паролю працюють, та додаток готовий до впровадження в інформаційні системи.

# ВИСНОВКИ

Проаналізувавши існуючі системи багатофакторної автентифікації та розмежування доступу до інформаційних сиситем , можна зробити висновок, що переважно використовуються парольні типи автентифікації. Проте з бурхливим розвитком інформаційних систем постало питання підвищення рівня захищеності інформаційних систем від НСД, тож перспективнішими в цьому плані є біометричні способи автентифікації.

Результатом виконаної роботи є аналіз та порівняння сучасних факторів автентифікації до інформаційних систем та розробка програмного модуля двохфакторної автентифікації за допомогою довгострокового паролю та одноразового паролю Google Authentictor. Під час виконання роботи було:

1. Проведено аналіз сучасних методів автентифікації та розмежування доступу до інформаційних систем, який став базою для розробки програмного веб-додатку з системою двохфаторної автентифікації за допомогою довгострокового паролю та тимчасового паролю Google Authenticator.

2. Розроблено веб-додаток з системою двохфаторної автентефікації за довгостроковим паролем і одноразовим паролем, при використанні платформи мови програмування JavaScript NodeJS та модулю Express в якості фреймворка для запуску веб-сервера, що дало змогу забезпечити санкціонований доступ користувачів до ресурсів інформаційних систем .

3. Проведено тестування розробленої системи двохфаторної автентефікації за довгостроковим паролем і одноразовому паролю для перевірки її спроможності для вирішення поставленої задачі.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

1. ДСТУ 3396.0-96 Захист інформації. Технічний захист інформації. Основні положення.

2. НД ТЗІ 1.1-003-99. Термінологія у галузі захисту інформації в комп'ютерних системах від несанкціонованого доступу, затверджений наказом Департаменту спеціальних телекомунікаційних систем та захисту інформації СБ України від 28.04.99 р. № 22.

3. НД ТЗІ 1.1-004-99. Загальні положення щодо захисту інформації в

комп'ютерних системах від несанкціонованого доступу, затверджений наказом ДСТСЗІ СБ України від 28.04.99 р. № 22. Про електронні документи та електронний документообіг: закон України від 22 травня 2003 р. № 851-IV // Урядовий кур'єр — 2003. — №119 – 2 липня. – С. 1–6.

4. Про електронний цифровий підпис: закон України від 22 травня 2003 р. №

851-IV

// Урядовий кур'єр — 2003. — №119 – 2 липня. – С. 1– 6.

5. Анин Б. Защита компьютерной информации. — СПб.: БХВ-Петербург, 2000. — 384 с.

6. Гайворонський М.В., Новіков О.М. Безпека інформаційно-комунікаційних систем. - К.: Видавнича група BHV,2009. – 608 с., іл.

7. Домарев В.В. Безопасность информационных технологий. Системный подход. – К.: ООО «ТИД «ДС», 2004. – 992 с.

8. Дронь М.М., Малайчук В.П., Петренко О.М. Основи теорії захисту інформації: Навч. посібник. – Д.: Вид-во Дніпропетр. ун-ту, 2001. – 312 с.

9. Конахович Г.Ф., Корченко О.Г., Юдін О.К., Захист інформації в мережах передачі даних: Підручник. – К.: Видавництво ТОВ НВП «ІНТЕРСЕРВІС», 2009. – 714с., іл.

10. Сарбуков А. Автентификация В Компьютерных Системах / А. Сарбуков

А. Грушо // Системы безопасности. – 2003. - №5 (53). – С. 25-29.

11. Auzy [Электронный ресурс] // URL: https://github.com/alexey-detr/auzy (дата обращения: 20.01.2020).

12. ExpressJS. Фреймворк веб-приложений NodeJS [Электронный ресурс] // URL: https://expressjs.com/ru/ (дата обращения: 20.01.2020).

13. ExpressJS middleware. Использование промежуточных обработчиков [Электронный ресурс] // URL: https://expressjs.com/ru/guide/using-middleware.html (дата обращения: 20.01.2020).

14. Google Authenticator [Электронный ресурс] // URL: https://te-st.ru/entries/google-authenticator/ (дата обращения: 20.01.2020).

15. Multi-factor authentication [Электронный ресурс] // URL: https://www.onelogin.com/learn/what-is-mfa (дата обращения: 20.01.2020).

16. NodeJS [Электронный ресурс] // URL: https://nodejs.org/en/ (дата обращения: 20.01.2020).

17. OAuth 2.0 [Электронный ресурс] // URL: https://oauth.net/2/ (дата обращения: 20.01.2020).

18. OWASP. Broken authentication and session management [Электронный ресурс] // URL: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2- Broken_Authentication (дата обращения: 20.01.2020).

19. OWASP. Cross Site Scripting (XSS) [Электронный ресурс] // URL: https://owasp.org/www-community/attacks/xss/ (дата обращения: 20.01.2020).

20. PassportJS. Authentication middleware for NodeJS [Электронный ресурс] // URL: http://www.passportjs.org/ (дата обращения: 20.01.2020).

# ДОДАТКИ

**Лістинг програми:**

Back-end\App.js

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const cors = require('cors');
const login = require('./routes/login');
const register = require('./routes/register');
const tfa = require('./routes/tfa');

app.use(bodyParser.json());
app.use(cors());

app.use(login);
app.use(register);
app.use(tfa);

app.listen('3000', () => {
    console.log('The server started running on http://localhost:3000');
});
```

Back-end\Routes\register.js

```
const express = require('express');
const commons = require('./commons');
const router = express.Router();

router.post('/register', (req, res) => {
    console.log(`DEBUG: Received request to register user`);

    const result = req.body;

    if ((!result.uname && !result.upass) || (result.uname.trim() == "" || result.upass.trim() == "")) {
        return res.send({
            "status": 400,
            "message": "Username/ password is required"
        });
    }

    commons.userObject.uname = result.uname;
    commons.userObject.upass = result.upass;
    delete commons.userObject.tfa;

    return res.send({
        "status": 200,
        "message": "User is successfully registered"
    });
});

module.exports = router
```

Back-end\Routes\common.js

```
let userObject = {};

module.exports = {
    userObject
};
```

Back-end\Routes\login.js

```
const express = require('express');
const speakeasy = require('speakeasy');
const commons = require('./commons');
const router = express.Router();

router.post('/login', (req, res) => {
    console.log(`DEBUG: Received login request`);

    if (commons.userObject.uname && commons.userObject.upass) {
        if (!commons.userObject.tfa || !commons.userObject.tfa.secret) {
            if (req.body.uname == commons.userObject.uname && req.body.upass == commons.userObject.upass) {
                console.log(`DEBUG: Login without TFA is successful`);

                return res.send({
                    "status": 200,
                    "message": "success"
                });
            }
            console.log(`ERROR: Login without TFA is not successful`);

            return res.send({
                "status": 403,
                "message": "Invalid username or password"
            });

        } else {
            if (req.body.uname != commons.userObject.uname || req.body.upass != commons.userObject.upass) {
                console.log(`ERROR: Login with TFA is not successful`);

                return res.send({
                    "status": 403,
                    "message": "Invalid username or password"
                });
            }



            if (!req.headers['x-tfa']) {
                console.log(`WARNING: Login was partial without TFA header`);

                return res.send({
                    "status": 206,
                    "message": "Please enter the Auth Code"
                });
            }
            let isVerified = speakeasy.totp.verify({
                secret: commons.userObject.tfa.secret,
                encoding: 'base32',
                token: req.headers['x-tfa']
            });
```

```
            if (!req.headers['x-tfa']) {
                console.log(`WARNING: Login was partial without TFA header`);

                return res.send({
                    "status": 206,
                    "message": "Please enter the Auth Code"
                });
            }
            let isVerified = speakeasy.totp.verify({
                secret: commons.userObject.tfa.secret,
                encoding: 'base32',
                token: req.headers['x-tfa']
            });

            if (isVerified) {
                console.log(`DEBUG: Login with TFA is verified to be successful`);

                return res.send({
                    "status": 200,
                    "message": "success"
                });
            } else {
                console.log(`ERROR: Invalid AUTH code`);

                return res.send({
                    "status": 206,
                    "message": "Invalid Auth Code"
                });
            }
        }
    }

    return res.send({
        "status": 404,
        "message": "Please register to login"
    });
});

module.exports = router;
```

Back-end\Routes\tfa.js

```
const express = require('express');
const speakeasy = require('speakeasy');
const QRCode = require('qrcode');
const commons = require('./commons');
const router = express.Router();

router.post('/tfa/setup', (req, res) => {
    console.log(`DEBUG: Received TFA setup request`);

    const secret = speakeasy.generateSecret({
        length: 10,
        name: commons.userObject.uname,
        issuer: 'NarenAuth v0.0'
    });
    var url = speakeasy.otpauthURL({
        secret: secret.base32,
        label: commons.userObject.uname,
        issuer: 'NarenAuth v0.0',
        encoding: 'base32'
    });
    QRCode.toDataURL(url, (err, dataURL) => {
        commons.userObject.tfa = {
            secret: '',
            tempSecret: secret.base32,
            dataURL,
            tfaURL: url
        };
        return res.json({
            message: 'TFA Auth needs to be verified',
            tempSecret: secret.base32,
            dataURL,
            tfaURL: secret.otpauth_url
        });
    });
});
```

```
router.get('/tfa/setup', (req, res) => {
    console.log(`DEBUG: Received FETCH TFA request`);

    res.json(commons.userObject.tfa ? commons.userObject.tfa : null);
});

router.delete('/tfa/setup', (req, res) => {
    console.log(`DEBUG: Received DELETE TFA request`);

    delete commons.userObject.tfa;
    res.send({
        "status": 200,
        "message": "success"
    });
});

router.delete('/tfa/setup', (req, res) => {
    console.log(`DEBUG: Received DELETE TFA request`);

    delete commons.userObject.tfa;
    res.send({
        "status": 200,
        "message": "success"
    });
});

router.post('/tfa/verify', (req, res) => {
    console.log(`DEBUG: Received TFA Verify request`);

    let isVerified = speakeasy.totp.verify({
        secret: commons.userObject.tfa.tempSecret,
        encoding: 'base32',
        token: req.body.token
    });

    if (isVerified) {
        console.log(`DEBUG: TFA is verified to be enabled`);

        commons.userObject.tfa.secret = commons.userObject.tfa.tempSecret;
        return res.send({
            "status": 200,
            "message": "Two-factor Auth is enabled successfully"
        });
    }

    console.log(`ERROR: TFA is verified to be wrong`);

    return res.send({
        "status": 403,
        "message": "Invalid Auth Code, verification failed. Please verify the system Date and Time"
    });
});

module.exports = router;
```