Хмельницький національний університет
Факультет програмування
та комп'ютерних і телекомунікаційних систем
Кафедра кібербезпеки та комп'ютерних систем і мереж

# КВАЛІФІКАЦІЙНА РОБОТА

*бакалавр*
Освітній рівень

Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів
Назва теми

*КвРКБ.170156.17.02.07 ПЗ*
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *125 «Кібербезпека»*
Шифр, назва

Освітня програма *«Кібербезпека»*
Назва

Виконав: студент IV курсу, група КБ–17–1 О.В. Камінський
Підпис Ініціали, прізвище

Керівник _____ В.М. Чешун
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ І.В. Муляр
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри кібербезпеки та
комп'ютерних систем і мереж _____ Ю.П. Кльоц
Підпис Ініціали, прізвище

«23» червня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ПРОГРАМУВАННЯ ТА КОМП'ЮТЕРНИХ І ТЕЛЕКОМУНІКАЦІЙНИХ СИСТЕМ

Кафедра КІБЕРБЕЗПЕКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 125 КІБЕРБЕЗПЕКА

Освітня програма ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА ПІДГОТОВКИ БАКАЛАВРІВ

ЗАТВЕРДЖУЮ

Завідувач кафедри _____

Клюх Ю.Ч

5.02 •2021 р.

**ЗАВДАННЯ**
**НА КВАЛІФІКАЦІЙНУ РОБОТУ**

Камінському Олександру Володимировичу

Прізвище, ім'я, по батькові студента

1 Тема роботи Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів

Керівник роботи _____

_____

Прізвище, ім`я, по батькові, науковий ступінь, вчене звання

Затверджено наказом ректора університету від05022021р. №11 додаток 9

2 Строк подання студентом роботи на кафедру:_____

3 Вихідні дані до роботисистема захисту електронного обігу інформації«Askod»_____

_____

4Зміст пояснювальної записки (перелік питань, які потрібно розробити) Аналіз об'єкта захисту, обґрунтування вибору засобів для побудови системи безпеки, проектування модулю захисту,реалізація роботи_____

5 Перелік графічного матеріалу (із зазначенням обов'язкових креслень) «Алгоритм початкової авторизації», «Алгоритм роботи модифікованого захисту», «Алгоритм роботи перевірки токену авторизації», «Алгоритм роботи першого етапу автенфікації», «Алгоритм роботи другого етапу автенфікації»_____

6 Консультанти розділів курсового проекту (роботи)

| Розділ | Прізвище, ініціали та посада консультанта | Підпис, дата | |
|---|---|---|---|
| | | завдання видав | завдан прийн |
| Нормоконтроль | Муляр І.В., доцент кафедри КбКСМ | | |
| Антиплагіат | Муляр І.В., доцент кафедри КбКСМ | | |

7 Дата видачі завдання _____ _____ 20___р.

## КАЛЕНДАРНИЙ ПЛАН

| №з/п | Назва етапів (розділів) дипломного проекту (роботи) | Термін виконання етапів проекту (роботи) | Пр |
|---|---|---|---|
| 1 | Вибір та затвердження теми кваліфікаційної роботи. | Січень | |
| 2 | Аналіз об'єкта захисту. | Січень–лютий | |
| 3 | Проектування та розробка загальної архітектури і структури системи захисту. | Лютий–березень | |
| 4 | Програмна реалізація запропонованого рішення та тестування системи; аналіз результатів і оцінювання прийнятих рішень. | Квітень | |
| 5 | Написання тексту пояснювальної записки та розробка графічних матеріалів. | Травень | |
| 6 | Остаточне коригування кваліфікаційної роботи з урахуванням зауважень керівника. | | |
| 7 | Оформлення кваліфікаційної роботи як документа відповідно до вимог. | | |
| 8 | Отримання супровідних документів. Нормоконтроль. | Червень | |
| 9 | Підготовка до захисту та захист кваліфікаційної роботи. | | |

Студент _____ О.В.Камінський
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ В.М. Чешун
Підпис Ініціали, прізвище

# АНОТАЦІЯ

Тема кваліфікаційної роботи: «Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів».

Автор роботи: Камінський Олександр Володимирович.

Керівник роботи: Чешун Віктор Миколайович.

Обсяг – 51 с., 12 рис., 2 додатка, 17 джерел.

Графічна частина: 9 презентаційних слайдів, 5 плакати.

СИСТЕМА ЗАХИСТУ ВІД ВИТОКУ ДАНИХ.

Основної метою є захист існуючої системи дукоментообігу «Askod» додатковим модулем захисту двофакторної автенфікації. Який повинен розширити функції захисту системи.

Даний модуль повинен посилити надійність входу в особистий кабінет співробітника системи «Askod».

У роботі було проаналізовано та досліджено види двофакторноїавтенфікації, криптосистему RSA, криптосистему AES, хеш-функцію SHA-3,обфускацію коду.

В рамках кваліфікаційної роботи була розроблено для користувача інтерфейс для двофакторної системи автентифікації особистого кабінету співробітника системи «Askod» і саму систему двофакторної автентифікації.

Підпис студента

07.06.2021
Дата

| Формат | Зона | Позиц | Позначення | Найменування | Кільк. | Прим. |
|---|---|---|---|---|---|---|
| А4 | | 1 | | Завдання на дипломний проект | 1 | |
| А4 | | 2 | | Анотація | 1 | |
| А4 | | 3 | КвРКБ.170156.17.01.07 ПЗ | Захист системи електронного обігу інформації АТ «Ощадбанк» Пояснювальна записка | 1 | |
| А2 | | 4 | КвРКБ.170156.17.01.07 Е8 | Перший етап автенфікації Алгоритм роботи | 1 | |
| А2 | | 5 | КвРКБ.170156.17.01.07 Е8 | Другий етап автенфікації Алгоритм роботи | 1 | |
| А2 | | 6 | КвРКБ.170156.17.01.07 Е8 | Перевірка токену автенфікації Алгоритм роботи | 1 | |

| Зм. | Арк. | № Докум. | Підп. | Дата | КвРКБ.170156.17.01.07 ВП | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Камінський О.В. | | | Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів Відомість проекту | Літера | Аркуш 1 / Аркушів 2 |
| Перев. | | Чешун В.М. | | | | н | |
| Н. контр. | | Муляр І.В. | | | | ХНУ, КБ-17-1 | |
| Затв. | | Кльоц Ю.П. | | | | | |

| Формат | Зона | Позиц | Позначення | Найменування | Кільк. | Прим. |
|---|---|---|---|---|---|---|
| А2 | | 7 | КвРКБ.170156.17.01.07 Е8 | Алгоритм роботи | 1 | |
| | | | | модифікованого | | |
| | | | | захисту | | |
| | | | | Алгоритм роботи | | |
| | | | | | | |
| А2 | | 8 | КвРКБ.170156.17.01.07 Е8 | Алгоритм початкової | 1 | |
| | | | | авторизації | | |
| | | | | Алгоритм роботи | | |

| Зм. | Арк. | № Докум. | Підп. | Дата | КвРКБ.170156.17.01.07 ВП | Арк. |
|---|---|---|---|---|---|---|

## ЗМІСТ

# ВСТУП

З кожним роком у світі з’являється все більше і більше сучасних технологій і росте величезна кількість нових застосунків. І в більшості застосунків потрібні методи автентифікації від несанкціонованого доступу. Але більшість наявних методів автентифікації недостатньо, щоб повною мірою захистити облікові записи від несанкціонованого доступу зловмисниками. У кожного теперішнього методу автентифікації є ряд своїх недоліків. І для кращого захисту від атаки несанкціонованого доступу, була придумана двофакторна автентифікація.

І у зв’язку з революційним відкриттям такої технології як 2FA, яка має всі властивості для кращого захисту застосунків і всіляких транзакцій і даних від зловмисників. Стало можливо нове рішення для методів автентифікації, використовуючи 2FA.

Для захисту застосунків та облікових записів від несанкціонованого доступу застосовують механізми автентифікації. Але більшість відомих механізмів автентифікації не гарантують повний захист від несанкціонованого доступу. Тому актуальним є дослідження методики автентифікації, двофакторної автентифікації на основі технології 2FA.

Метою і завданням даної роботи є впровадження додаткового модуля захисту для системи електронного обігу документів в АТ «Ощадбанк».

Проведення аналізу програми для роботи з електронним документообігом «Askod», принципи її роботи та зв’язок з інформаційною безпекою. Перевірка надійності методів автентифікації у застосунку.

Ознайомлення та опрацювання електронних джерел про загрози безпеки несанкціонованого доступу, методів автентифікації та технології 2FA.

Захист полягає у тому, що була запропонована автентифікації на основі RSA, який повинен покращити захист програми.

Запропоноване рішення може використовуватися у реальних програмних рішеннях розробників застосунків для забезпечення автентифікації.

# 1 АНАЛІЗ ОБ'ЄКТА ЗАХИСТУ

У цьому розділі буде описано призначення модуля системи, можливі загрози безпеки, засновані на несанкціонованому доступі до системи, а так само представлені механізми автентифікації в розподілених системах.

## 1.1 Призначення системи

Двофакторна система автентифікації (2FA) особистого кабінету співробітника призначена для забезпечення кращого захисту, ніж поточна версія.

На цей час в розробці нова версія модуля захисту системи, яка забезпечує більше функціональності, ніж стара версія. Тому безпека нової версії повинна бути вищою [1].

Основним недоліком поточної версії процедури автентифікації користувача є можливість підбору пароля користувача, крім того, за допомогою підібраного пароля можна зайти в особистий кабінет необмежену кількість разів.

Нова версія автентифікації зажадає від користувача, крім логіна і пароля, підтвердження особи користувача, так званого другого фактора автентифікації (чим володіє тільки користувач) [2].

## 1.2 Аналіз вимог до модуля

Система повинна відповідати наступним вимогам:

– можливість зміни ключів шифрування для окремого користувача;

– стійкість системи до витоку ключа для одного або більшості користувачів;

– стійкість системи до витоку токену;

– неможливість введення кілька разів з використанням одного і того ж токену;

– у разі неправильного введення даних на другому етапі автентифікації, повторення відбувається тільки на цьому етапі автентифікації, а не весь процес;

– надання інформації про невдалі входи адміністратору інформаційної безпеки.

Якщо ключ шифрування просочується від одного або декількох з користувачів, всі інші користувачі не повинні отримати шкоди через це, тому існує вимога зміни ключа шифрування для одного або багатьох користувачів, не впливаючи на інших користувачів. У гіршому випадку, коли зловмисник отримує ключові дані від усіх користувачів системи, він не повинен мати можливості перехоплювати та розшифровувати токени користувачів.

Якщо маркер автентифікації було перехоплено, ви повинні заблокувати можливість повторного введення того самого маркера, і зловмисник не повинен підкушувати такий маркер автентифікації самостійно.

Система повинна забезпечити необхідний рівень безпеки, а саме, в разі витоку одного токену автентифікації, зловмисник не повинен отримувати ніякої інформації про токени автентифікації інших користувачів.

Для досягнення цієї мети необхідно досягти наступних викликів:

– проаналізувати предметну область;

– сформулювати вимоги до програмного продукту;

– розробити архітектуру системи;

– розробити алгоритми функціонування системи;

– впровадити двофакторні системи автентифікації для особистого кабінету співробітника.

Впровадження двофакторної автентифікації підвищить надійність входу в систему, а головне - значно знизить ймовірність того, що в систему зможе проникнути зловмисник.

## 1.3 Типи двофакторної автентифікації

### 1.3.1. SMS-код

Найпоширеніший другий етап автентифікації. В основі цього методу автентифікації становить технологія OTP (one time password) - одноразовий пароль. На першому рівні автентифікації користувач вводить свої персональні дані (логін і пароль), потім він відправляє на телефон SMS з кодом, який необхідно ввести на другому етапі автентифікації.

Переваги цього типу автентифікації:

– генерація нових кодів кожного разу, коли ви входите в систему;

– прив'язатися до номера телефону.

Недоліки:

– неможливість увійти в систему при відсутності стільникової мережі;

– існує можливість заміни номера через послугу оператора або співробітників салонів мобільного зв'язку;

– у разі авторизації та отримання кодів на одному пристрої (смартфоні) захист перестає бути двофакторний.

Цей тип автентифікації не можна використовувати в системі, що розробляється, оскільки вартість занадто висока для її реалізованості. Вартість одного SMS-повідомлення для державного університету становить близько 0,5 гривень, а кількість входів в особистий кабінет за один день може перевищувати 1000. Крім того, не всі користувачі системи мають у своєму розпорядженні мобільний телефон.

### 1.3.2. Програми автентифікації

Це схоже на опцію SMS-коду, але спеціальні коди генеруються безпосередньо на пристрої за допомогою спеціального додатка (Authy, Google Authenticator). Під час початкового налаштування користувач отримує первинний ключ, на основі якого різні алгоритми генерують одноразові паролі, з обмеженою тривалістю від 30 до 60 секунд.

Переваги:

– вам не потрібен стільниковий сигнал для авторизації, вам не потрібно підключення до Інтернету, при першому налаштуванні;

– підтримці декількох облікових записів в одному додатку.

Недоліки:

– якщо зловмисники отримають доступ до первинного ключа, вони зможуть генерувати майбутні паролі;

– у разі авторизації та отримання кодів на одному пристрої (смартфоні) захист перестає бути двофакторний.

Цей метод автентифікації не можна використовувати, оскільки система «Askod» розрахована на всіх працівників банку, але не всі працівники мають у своєму розпорядженні мобільний телефон, а тим більше смартфон. Додатковим обмеженням цього методу автентифікації є установка та реєстрація додаткових додатків. Крім того, тривалість цього одноразового пароля занадто коротка.

1.3.3. Вхід через мобільні додатки

Цей тип автентифікації є комбінацією попередніх. Для авторизації системи нема потреби вводити одноразові паролі, потрібно підтвердити вхід з мобільного пристрою встановленим додатком сервісу (Google). Пристрій має закритий ключ, який перевіряється при кожному вході в обліковий запис. Наприклад, при вході у свій обліковий запис Google потрібно ввести логін і пароль, після чого підтвердити запит на своєму смартфоні повідомленням про вхід, після чого буде виконана авторизація.

Переваги:

– нема потреби вводити додаткові коди при авторизації;

– незалежність від стільникової мережі;

– підтримка декількох облікових записів в одному додатку.

Недоліки:

– якщо закритий ключ перехоплено, зловмисники можуть прикидатися власником ключа;

– якщо ви використовуєте один і той же пристрій входу, значення двофакторної автентифікації втрачається.

Цей метод автентифікації не можна використовувати, оскільки система «Askod» розрахований на всіх працівників, але не всі працівники мають у своєму розпорядженні мобільний телефон, а тим більше смартфон. Додатковим обмеженням цього методу автентифікації є установка та реєстрація додаткових додатків. Також не всі користувачі мають облікові записи в різних сервісах, які надають можливість такої автентифікації.

1.3.4. Апаратні токени

Один з найнадійніших способів двофакторної автентифікації. Оскільки апаратні маркери є окремими пристроями, вони не зможуть втратити свій двофакторний компонент. Апаратні токени можуть бути декількох видів: USB-брелок, який підключається безпосередньо до комп'ютера і генерує ключі (рисунок 1.1), або токен з екраном, що показує згенерований ключ, що вводиться (рисунок 1.2).

Переваги:

– нема потреби в SMS або додатках;
– мобільний телефон не потрібен;
– повністю незалежний пристрій.

Недоліки:

– купувати потрібно окремо;
– не всі служби підтримуються;
– кілька облікових записів повинні носити кілька жетонів.


Рисунок 1.1 – USB-токен


Рисунок 1.2 - Токен з кодом

Попри недоліки цього методу автентифікації, система "Askod" вони не такі значні. Кожен користувач системи може мати лише один обліковий запис, тому кожен користувач матиме один маркер для авторизації. Однак з двох типів токенів перевага надається токену, який генерує код одним натисканням кнопки на пристрої, а не маркером USB. Не всі користувачі банку мають хороші комп'ютери в своєму розпорядженні, і маркер USB іноді може не вистачити потужності, і не завжди є вільний порт USB для підключення.

## 1.4 Аналіз чинної системи захисту програми "Askod"

АСКОД - це комплексне рішення, що забезпечує нагромадження і систематизацію неструктурованої інформації, підтримку орієнтованих на документи ділових процесів і можливості аналізу інформації

Можливості системи електронного документообігу аскод автоматизація:

– службового документообігу;
– процесів опрацювання вхідної, вихідної та внутрішньої кореспонденції;
– процесів опрацювання звернень громадян, запитів, заявок;
– процесів надання адміністративних послуг.

Можливості системи електронного документообігу:

– автоматизація процесів: ведення реєстрів;
– процесів обліку договорів і контролю їх виконання;

– архівного зберігання документів;

– застосування: електронного цифрового підпису;

– технології штрих-кодування документів;

– формування основних звітів щодо документообігу та контролю виконання;

– контекстний пошук;

– розмежування прав доступу;

– пошук за різними атрибутами;

– протоколювання дій користувачів у захищеному журналі.

Програма дозволяє визначати:

– види послуг;

– перелік документів, що є необхідним для надання послуги;

– шаблони документів;

Також відстежувати:

– повноту надання необхідних документів для отримання послуги;

– хід виконання послуги; контроль надання послуги.

Теперішня система захисту "Askod". Ця система призначена для роботи з електронним обігом документів яка для роботи в банку підходить.

Під час проведення аналізу чинної системи захисту "Askod", було виявлено наступні недоліки системи входу. Як видно на рисунку 1.3 під час авторизації в програму потрібно вести тільки логін і пароль, що робить її вразливою до в зламу. Зловмисник зможе перехопити данні входу і використати їх.

Ще один недолік системи якщо зловмисник знає логін працівника він може підбирати пароль, а про це не буде ніяких повідомлень знати. Що свідчить про низький рівень захисту інформації, а також низький рівень передачі та її захисту.

Тому було прийняте рішення про необхідну розробку захищеного способу входу в програму.

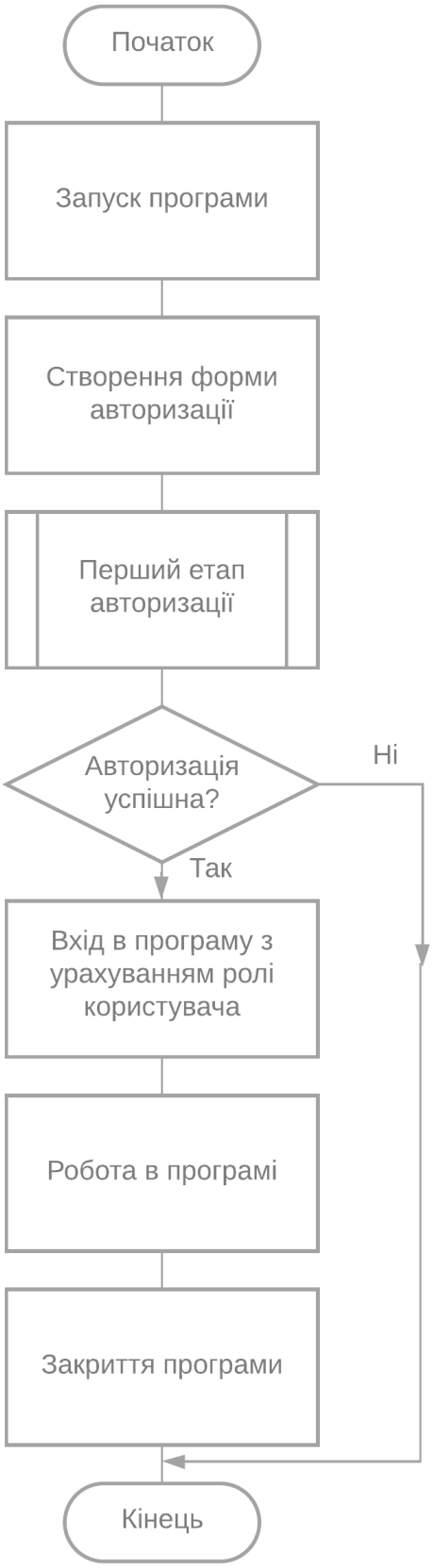



Рисунок 1.3 – Загальна схема роботи програми

## 1.5 Загрози інформаційної безпеки, їх класифікація та джерела

Інформаційна безпека в найширшому сенсі цього слова являє собою сукупність засобів захисту інформації від випадкового або ненавмисного впливу. Це поняття може бути розкрито за допомогою моделей безпеки. Моделі являють собою поділ усіх видів порушень на кілька груп таким чином, щоб будь-яке порушення яке стосується хоча б до однієї з цих груп і якщо інформаційна система може протистояти кожній з цих груп порушень, то вона є безпечною. Найпоширенішою і відомішою є модель, описана в стандарті ISO 27001.

Цілісність - дані повинні залишатися незмінними після використання, обробки або передачі, повинні зберігати свою структуру і вихідну форму.

Конфіденційність - доступ до інформації повинен бути обмежений. Під час операцій над інформацією вона повинна бути доступна тільки тим користувачам, у яких є права доступу до неї[4].

Доступність - інформація повинна бути доступна в будь-який момент для осіб, які мають права доступу до неї.

Стандарт ISO 27001 націлений на забезпечення трьох принципів управління інформації, які перераховані вище, а заходи, які ним передбачені, спрямовані на захист даних від кіберзлочинів, незаконного втручання, крадіжки інформації та інших загроз.

Крім моделі з трьох груп (цілісність конфіденційність і доступність) існують і інші, наприклад, гексада (шестикутна зірка - підкреслює, що модель складається з шести груп) Паркера. У цій моделі розглядаються 6 груп можливих порушень, крім трьох відомих (цілісність конфіденційність і доступність) також розглядаються:

Дійсність - фізичний контроль над системою знаходиться у володінні у тих осіб, у яких права доступу [5].

Корисність - зручність використання інформаційної системи, всі методи захисту не повинні перешкоджати користувачеві в роботі з конфіденційною інформацією.

Перевірка даних - неможливість користувача видати себе за іншого і правдивість інформації про авторство і джерелах документів [6].

Модель STRIDE є поширеною моделлю і має шість груп порушень. Найменування даної моделі є абревіатурою назв шести груп, які входять в неї:

– підміна даних - можливість користувача видати себе за іншого і таким чином отримати несанкціонований доступ до конфіденційної інформації;

– зміна даних;

– відмова від відповідальності - можливість користувача відмовитися від відповідальності у виконанні будь-яких дій шляхом введення оператора системи в оману;

– порушення конфіденційності;

– порушення доступності;

– підвищення прав доступу - несанкціоноване підвищення повноважень користувачем з метою отримання доступу до конфіденційних даних [7].

До неформальних засобів можна віднести нормативні правові документи, правила, вказівки, заходи з навчання персоналу і морально-етичні норми. Вони діляться на:

Нормативні - до даного виду захисту інформації відносяться нормативні правові документи та закони, що регулюють відносини у сфері захисту інформації.

Адміністративні - має на увазі організаційні заходи, спрямовані на поліпшення компетенцій і підвищення відповідальності співробітників в області інформаційної безпеки.

Морально-етичні - не є обов'язковими і не регулюються законодавчо. Сформовані принципи поведінки, які сприяють підвищенню рівня захищеності інформації в організації. При недотриманні даних норм є ризик втрати авторитету [8].

До формальних засобів відносять вже технічні засоби захисту, які фізично або технічно перешкоджають злому або розкрадання інформаційних носіїв. Їх ділять на:

Апаратні - це технічні пристосування, які вбудовуються в інформаційну систему і перешкоджають доступу і дозволяють замаскувати інформацію від сторонніх осіб. До подібних пристроїв можна віднести електричні, оптичні, що пригнічують шум пристрою.

Фізичні - будь-які засоби захисту, які перешкоджають фізичного контакту з носіями інформації або з пристроями, що дозволяють отримати до неї доступ. Це можуть бути кодові замки, сталеві двері, відеоспостереження, датчики руху і так далі.

Програмні - програмне забезпечення, що захищає інформаційну систему від зовнішніх і внутрішніх загроз. Найчастіше представляють комплекс засобів з антивірусного ПО, Firewall, і безлічі інших програм, що дозволяють розмежовувати або обмежувати доступ різних осіб до ІС.

Криптографічні – являють собою набір криптографічних і стенографічних методів, що дозволяють передавати конфіденційну інформацію по внутрішній і зовнішній мережі в зашифрованому вигляді [9].

## 1.6 Висновки та постановка задачі

В цьому розділі розглянув призначення особистого кабінету співробітника банку, необхідність розробки розширеного способу входу в особистий кабінет, а також недоліки наявної системи входу.

Були розглянуті вимоги до двофакторної системи автентифікації, що розробляється. Основними вимогами є забезпечення захисту кінцевих даних користувачів, а також необхідний рівень захисту переданої інформації, а також її зберігання.

Розглядалися різні типи двофакторної автентифікації: SMS-код, програми автентифікації, перевірка мобільного входу та апаратні токени.

SMS-код, один з найпоширеніших видів двофакторної автентифікації не підходить для реалізації через надмірну вартість тому, що не всі користувачі системи банку мають у своєму розпорядженні мобільний телефон.

Ви не можете використовувати автентифікатори для реєстрації входу за допомогою мобільних додатків і додатків, оскільки не всі користувачі банку мають обліковий запис смартфона та соціальних мереж.

Не доцільно використовувати токени USB, оскільки можуть виникнути технічні проблеми, наприклад, користувачі системи банку можуть забути або втратити їх.

# 2 ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ ДЛЯ ПОБУДОВИ СИСТЕМИ

## 2.1 Опис криптосистеми RSA

При розробці двофакторної системи автентифікації для особистого кабінету співробітника слід враховувати той факт, що частина створених токенів автентифікації відбудеться на стороні замовника. Токен повинен бути захищений від вільного доступу, а саме зашифрований, але оскільки він генерується на стороні клієнта, використовувати симетричне шифрування не можна (в браузері в налагодженні можна подивитися на ключі шифрування, розшифрувати справжній токен і підробити його в майбутньому).

Розв'язання цієї проблеми є асиметрична система шифрування, в цьому випадку RSA.

Криптосистема RSA була створена в 1977 році та розшифровується як Rivest, Shamir, Aldeman, на імена людей, які її створили.

Ця криптосистема використовує два ключі: один для шифрування даних (відкритий ключ), інший для розшифрування повідомлень (закритий ключ). При створенні електронного цифрового підпису функції ключів змінюються: закритий ключ використовується для створення підпису, відкритий ключ використовується для перевірки правильності підпису [10].

Асиметричні системи шифрування засновані на одномовних функціях, які працюють так.

1. Якщо ви знаєте значення x, то обчислити F(x) відносно легко.

2. Нехай буде відомо y'F(x), але важко обчислити x. Можна обчислити x, але зробити це в розумні терміни неможливо.

Криптовалюти з відкритим ключем мають деякі недоліки в порівнянні з симетричними криптосистемами.

1. Швидкість алгоритмів криптовалют з відкритим ключем набагато нижче, ніж швидкість симетричних алгоритмів.

2. Довжина ключів в асиметричних системах набагато довше, ніж в симетричних криптосистемах.

Однак криптовалюти з відкритим ключем мають більш зручний розподіл ключів (однією з головних проблем симетричних крипто систем є обмін ключами), в мережах з більшою кількістю користувачів кількість ключів набагато менше.

В основі RSA становить завдання факторування (розкладання на мультиплікатори) роботи двох простих великих чисел.

Спочатку потрібно згенерувати ключі шифрування (відкриті, закриті) за наступним алгоритмом.

1. Виберіть два великих, простих чисел $p$ і $q$, $p$ не дорівнює $q$.

2. Модуль обчислюється $N=p*q$.

3. Обчислюється значення функції Вайлера з модуля $N$ (Формула 2.1) :

$$\varphi(N)=(p-1)*(q-1) \qquad (2.1)$$

4. Номер E, який називається відкритим експонентом, підбирається таким чином, щоб були вибрані $1<e<\varphi(N)$ і НОД($e$, $\varphi(N)$)=1. Обчислюється число d (таємний експонент), наприклад, виконується рівність: $d*e=1(\text{mod } \varphi(N))$

Пара (наприклад e $N$) називається відкритим ключем RSA, а пара ($d$,$N$) називається закритим ключем RSA [11].

Нехай він шифрує повідомлення M за допомогою відкритої клавіші ($e$,$N$).

$$C=M^e \bmod (N) \qquad (2.2)$$

де $C$ — зашифроване повідомлення.

Повідомлення на схемі нижче розшифровується

$$D=C^d \ mod \ (N) \qquad (2.3)$$

де $C$ - отримав зашифроване повідомлення, пара $(d,N)$ - закритий ключ RSA.

На практиці такий алгоритм використовувати не можна, оскільки він не відповідає певним умовам.

## 2.2 Опис криптосистеми AES

Щоб забезпечити безпечне зберігання ключів шифрування для створення маркера автентифікації, слід зберігати ключі шифрування в зашифрованому вигляді. Серверна сторона недоступна для користувачів системи, тому можна використовувати симетричну систему шифрування AES.

AES (Advanced Encryption Standart), також відомий як Rijndael, є симетричним алгоритмом шифрування блоків. Розмір ключа 128/192/256 біт, розмір блоку 128 біт. Розроблений у 2001 році.

Шифрування містить в собі такі функції:

1. ExpandKey - це функція обчислення всіх круглих клавіш;

2. SubBytes - це функція байт-змішування, яка використовує таблицю з символами узагальнення (таблиця 2.1);

3. ShiftRows - це функція, яка дозволяє циклічне зміщення рядків до різних значень;

4. MixColumns - функція, яка змішує дані всередині стовпця;

5. AddRoundKey - додати на модулі 2 поточний стан і круглий ключ;

Алгоритм шифрування такий:

1. розбийте вихідне повідомлення на 128-бітні блоки;

2. запустити expandKey;

3. запустити AddRoundKey для першого раунду;

4. запустити під байти;

5. запустити ShiftRows;

6. запустити MixColumns;

7. запустити AddRoundKey для наступного раунду;

8. якщо раунд не останній, то збільште раунд на 1 і перейдіть до кроку 4, інакше до кроку 9. Кількість раундів визначається розміром ключа;

9. запустити subBytes;

10. запустити ShiftRows;

11. AddRoundKey для останнього раунду.

Таблиця 2.1 – Таблиця підставлення

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 63 | 7c | 77 | 7b | F2 | 6b | 6f | c5 | 30 | 01 | 67 | 2b | fe | d7 | ab | 76 |
| 1 | ca | 82 | c9 | 7d | fa | 59 | 47 | f0 | ad | d4 | a2 | af | 9c | a4 | 72 | c0 |
| 2 | b7 | fd | 93 | 26 | 36 | 3f | f7 | cc | 34 | a5 | e5 | f1 | 71 | d8 | 31 | 15 |
| 3 | 04 | c7 | 23 | c3 | 18 | 96 | 05 | 9a | 07 | 12 | 80 | e2 | eb | 27 | b2 | 75 |
| 4 | 09 | 83 | 2c | 1a | 1b | 6e | 5a | a0 | 52 | 3b | d6 | b3 | 29 | e3 | 2f | 84 |
| 5 | 53 | d1 | 00 | ed | 20 | fc | b1 | 5b | 6a | cb | be | 39 | 4a | 4c | 58 | cf |
| 6 | d0 | ef | aa | fb | 43 | 4d | 33 | 85 | 45 | f9 | 02 | 7f | 50 | 3c | 9f | a8 |
| 7 | 51 | a3 | 40 | 8f | 92 | 9d | 38 | f5 | bc | b6 | da | 21 | 10 | ff | f3 | d2 |
| 8 | cd | 0c | 13 | ec | 5f | 97 | 44 | 17 | c4 | a7 | 7e | 3d | 64 | 5d | 19 | 73 |
| 9 | 60 | 81 | 4f | dc | 22 | 2a | 90 | 88 | 46 | ee | b8 | 14 | de | 5e | ob | db |
| A | e0 | 32 | 3a | 0a | 49 | 06 | 24 | 5c | c2 | d3 | ac | 62 | 91 | 95 | e4 | 79 |
| B | e7 | c8 | 37 | 6d | 8d | d5 | 4e | a9 | 6c | 56 | f4 | ea | 65 | 7a | ae | 08 |
| C | ba | 78 | 25 | 2e | 1c | a6 | b4 | c6 | e8 | dd | 74 | 1f | 4b | bd | 8b | 8a |
| D | 70 | 3e | b5 | 66 | 48 | 03 | f6 | 0e | 61 | 35 | 57 | b9 | 86 | c1 | 1d | 9e |
| E | e1 | f8 | 98 | 11 | 69 | d9 | 8e | 94 | 9b | 1e | 87 | e9 | ce | 55 | 28 | df |
| F | 8c | a1 | 89 | 0d | bf | e6 | 42 | 68 | 41 | 99 | 2d | 0f | b0 | 54 | bb | 16 |

Поточний стан є шифром.

Розшифровка відбувається таким же чином, тільки у зворотному порядку.

Таблиця 2.2 показує кількість можливих комбінацій клавіш в залежності

від розміру ключа.

Таблиця 2.2 - Кількість можливих комбінацій

| Розмір ключа, біт | Кількість комбінацій |
|---|---|
| 56 (DES) | $7.2*10^{16}$ |
| 128 (AES) | $3.4*10^{38}$ |
| 192 (AES) | $6.2*10^{57}$ |
| 256 (AES) | $1.1*10^{77}$ |

Таблиця 2.3 - Час для зламу ключа

| Розмір ключа, біт | Час для зламу |
|---|---|
| 56 | 399 секунд |
| 128 | $1.02*10^{18}$ |
| 192 | $1.872*10^{37}$ |
| 256 | $3.31*10^{56}$ |

В таблиці 2.3 приведений час, яке може знадобитися для комп'ютера, щоб зламати ключ. Як видно з таблиці чим більший розмір ключа, тим більше потрібний час для його зламу [12].

## 2.3 Опис хеш-функції SHA-3

Нехай $\{0,1\}^m$ кількість всіх двійкових ліній довжини m (m - викид хеш зображення) $\{0,1\}^*$ і багато всіх двійкових ліній кінцевої довжини. Потім хеш-функція набуває перетворенням виду:

$$h:\{0,1\}^{¿}\rightarrow\{0,1\}^m \qquad (2.4)$$

Хеш-функція називається криптографічно стійкою, якщо вона відповідає

умовам:

1. Незворотність. Для заданої хеш-функції не можна обчислити *x*, для якої *H(x)=y*.

2. Стійкість до першокласних зіткнень. Для даної y неможливо підібрати такий c, що *H(y)=H(c)*.

3. Стійкість до зіткнень другого роду. Неможливо підібрати y і c, такі як *H(y)=H(c).*

SHA-3 (Secured Hash Algorithm) або Keccak використовується як хеш-функція. Розроблено та затверджено як стандарт хешування у 2015 році. Загальна схема хешування представлена на рисунку 2.1.

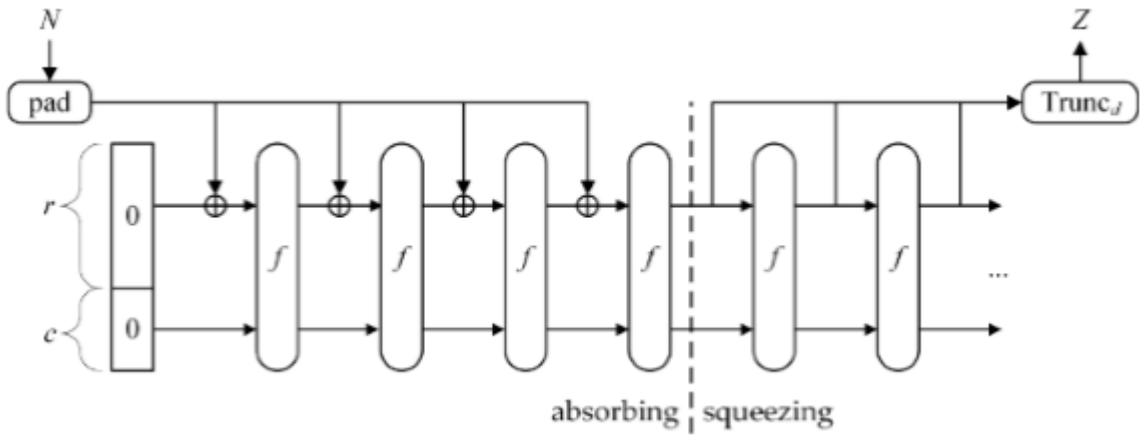



Рисунок. 2.1. Схема хеш-функції SHA-3.

Початкове повідомлення доповнюється довжиною декількох r, згідно з правилом N=N||0x01|| 0x00||…||0x00||0x80. Початкове повідомлення буде розбито на блоки довжиною r. Після отриманих даних *N* один раз застосовується функція f, яка змішує ці дані. Це стадія поглинання або поглинання.

На стадії стискання або пресування виконуються ті ж операції, але після кожної продуктивності функції f частина біта повертається у якості функції хешування до отримання необхідної довжини[14].

В якості стандарту прийнята реалізація Keccak-1600, т.е. r+c=1600. В якості значення r і c були вибрані наступні: SHA-224 (r=1156, c=448), SHA-256 (r=1088, c=512), SHA-384 (r=832, c=768), SHA-512 (r=576, c=1024)[15].

## 2.4 Обфускація коду

Токен авторизації створюється на стороні клієнта, отже, вихідний код доступний для всіх користувачів. Для того, щоб було складно проаналізувати, зрозуміти роботу, модифікацію алгоритму, необхідно використовувати обфускацію.

Обфускація програми - це перетворення O програми p, на вид O(p), в якому зберігається функціональність оригінальної програми або зберігається вихідний код програми, але важко проаналізувати роботу алгоритмів, а також змінити алгоритми [13].

Наприклад, проста програма, написана мовою програмування javaScript, яка зображається на консолі: «Hello World!»:

```
console.log("Hello World!");
```

Після процесу обфускації він може виглядати наступним чином:

```
var _0xb154 = [
  "\x48\x65\x6C\x6C\x6F\x20\x57\x6F\x72\x6C\x64\x21", "\x6C\x6F\x67"
  ]; console[_0xb154[1]](_0xb154[0])
```

Ця програма, незважаючи на складність коду, все ж виводить в консоль повідомлення "Hello World!".

Можна виділити наступні види обфускації.

1. Лексична обфускація.
2. Перетворення даних.
3. Трансформаційний менеджмент.
4. Профілактична обфускація.

Лексична обфускація полягає в заміні імен змінних і функцій, видаленні або зміні коментарів в програмному коді, видаленні прогалин і відступів, що використовуються для поліпшення візуального сприйняття програмного коду. Наприклад, код:

```
bool flag=false;
```

```
for(int count=0;count<100;++count)
{
if(count==99)
flag=true;
}
```

Після лексичної обфускації перетворюється в код, менш читабельний такого роду:

```
bool a=false;for(int b=0;b<100;++b){if(b==99)a=true;}
```

У міру збільшення кількості змінних в коді стає важко зрозуміти та проаналізувати такий код. Однак такий спосіб обфускації є найбільш ненадійним.

Перетворення даних – це зміна та створення нових типів даних.

Нижче наведено деякі з таких типів

1. Обфускація зберігання. Йдеться про використання незвичайних типів даних або створення нових, зміну сприйняття теперішніх. Наприклад:

– зберігання індексу елементів масиву як змінної, що не є інгібуючим мільйоном;

– перетворення статичних даних в виклик, що генерує необхідне значення (замініть статичний рядок викликом функції);

– розділення змінних (замість bool F=true; if(F) використовувати bool f1,f2=false; if(!(f1&&f2)))

2. З'єднання обфускації. Це розділення залежних даних або зв'язок незалежних даних. Прикладом може бути поділ одного масиву на декілька підмасивів, об'єднання декількох масивів в один, зменшення або збільшення розміру масиву, зміна ієрархії успадкування класів, створення додаткових класів або помилкове розділення чинних.

3. Обфускація перестановки змінних. Вона складається зі зміни

послідовності оголошених змінних, перестановці методів, певних полів у структурах [11].

Трансформаційне управління є порушенням природного перебігу програми.

Ви можете розбити на наступні групи.

1. Обчислювальна обфускація. Зміни, пов'язані з основною структурою управління потоком. Наприклад:

– додавання умов до циклів, які не впливають на кількість ітерацій циклу;

– перейменування імен певних об'єктів зі стандартних бібліотек;

– додавання надлишкових операцій (перетворення виразу A=A+B на вираз типу A=A+B*x-y, де x та y — 1 та 0 відповідно);

– розділення коду на окремі області, які працюють паралельно [13].

2. З'єднання обфускації. Розділення фрагментів програмного коду для усунення логічних зв'язків між ними. Наприклад:

– вставте код функції в місця, де він викликається, і видаліть його з програмного коду;

– також можна скористатися зворотною процедурою заміни частини коду викликом функції.

– об'єднання різних частин коду, які запускають різні операції разом.

– створення кількох альтернативних функцій;

– поділ циклу на кілька, якщо цикл кузова складається з декількох незалежних операцій

3. Послідовність обфускації. Він полягає в перестановці блоків, циклів, виразів.

Профілактична обфускація служить для захисту коду від деобфускації програмами, які засновані на пошуку найскладніших структур, виявленні невикористаних фрагментів коду, аналізі динамічних і статистичних даних.

Якість змін, внесених під час обфускації, можна визначити за наступними критеріями:

– ефективність обфускації - це складність відновлення коду програми;

– прозорість це взаємодія інформаційного коду з іншими;

– вартість - сума надлишкового коду в обфускованих;

– сталий розвиток складність злому програми автоматизованими засобами деобфускації.

Попри переваги безпеки коду, обфускація має ряд недоліків. Після обфускації код стає непридатним для налагодження, оскільки для зловмисника і для забудовника [14].

Існує також ненульова ймовірність того, що код не буде працювати після обфускації.

## 2.5 Висновки

Розробка двофакторної автентифікації підвищить надійність входу в систему, а головне - значно знизить ймовірність того, що систему зможуть в зламати.

Для досягнення цієї мети необхідно проаналізувати предметну область, сформулювати вимоги до програмного продукту, розробити архітектуру системи, розробити алгоритми функціонування системи, впровадити двофакторні системи автентифікації для особистого кабінету співробітника системи «Askod».

Розглянуто алгоритм шифрування RSA, який базується на однотипних функціях, таких як розкладання складеного числа на множники. Загальна схема шифрування RSA не використовується на практиці через свою вразливість, тому схема шифрування ускладнюється тим, що вона додає випадкове значення, яке не дозволяє підібрати вихідний код, а також ознака того, чи була функція сформована правильно або була підроблена зловмисником.

Криптосистема RSA буде використовуватися на стороні клієнта, оскільки будь-хто, хто використовує вбудовану в браузер систему налагодження, зможе отримати ключів шифрування. Криптосистема RSA використовує відкритий

ключ на стороні користувача, доступний кожному.

Однією з вимог при розробці даної системи було забезпечення безпеки зберігання ключів шифрування в базі даних, в цьому використовується симетрична криптовалюта AES, оскільки серверна частина недоступна користувачам, а тому немає необхідності у відкритому доступі до ключа шифрування.

Для захисту від підробок використовується функція SHA-3. Він відповідає вимогам криптографічної міцності, а також дозволяє знайти значення хеш-функції різної довжини.

Для захисту коду необхідна обфускація. Незважаючи на те, що обфускація захищає код від можливості розібрати та проаналізувати нападника, це не дозволяє ефективно дрібно розібрати код.

# 3 ПРОЕКТУВАННЯ СИСТЕМИ БЕЗПЕКИ

## 3.1 Основний алгоритм роботи програми

Загальна схема роботи алгоритму представлена на рисунку 3.1

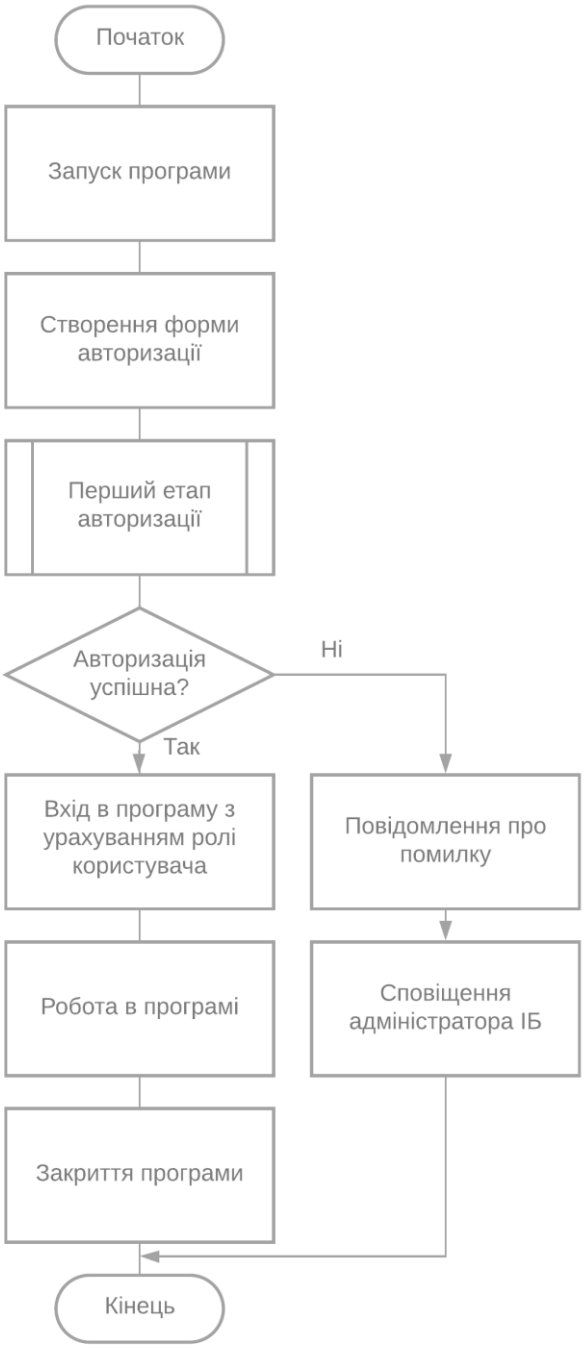



Рисунок 3.1 - Загальна схема алгоритму роботи додатку.

В загальний алгоритм був доданий модуль який після 5 некоректних вводів

логіну і пароля сповіщає адміністратора ІБ, блокує дане підключення. Що не дає методом підбору підібрати пароль для входу в програму.

Адміністратор вже зможе переглянути інформацію про підключення. В разі виявлення зловмисника зможе його заблокувати.

## 3.2 Модифікований алгоритм автентифікації


Рисунок 3.2 - Модифікований алгоритм автентифікації

На рисунку 3.2 розташований модифікований алгоритм автентифікації. Який містить в собі 2 етапи авторизації які забезпечують потрібний захист програми.

## 3.3 Алгоритм першого етапу автентифікації

Алгоритм роботи першого етапу автентифікації зображений на рисунку 3.3

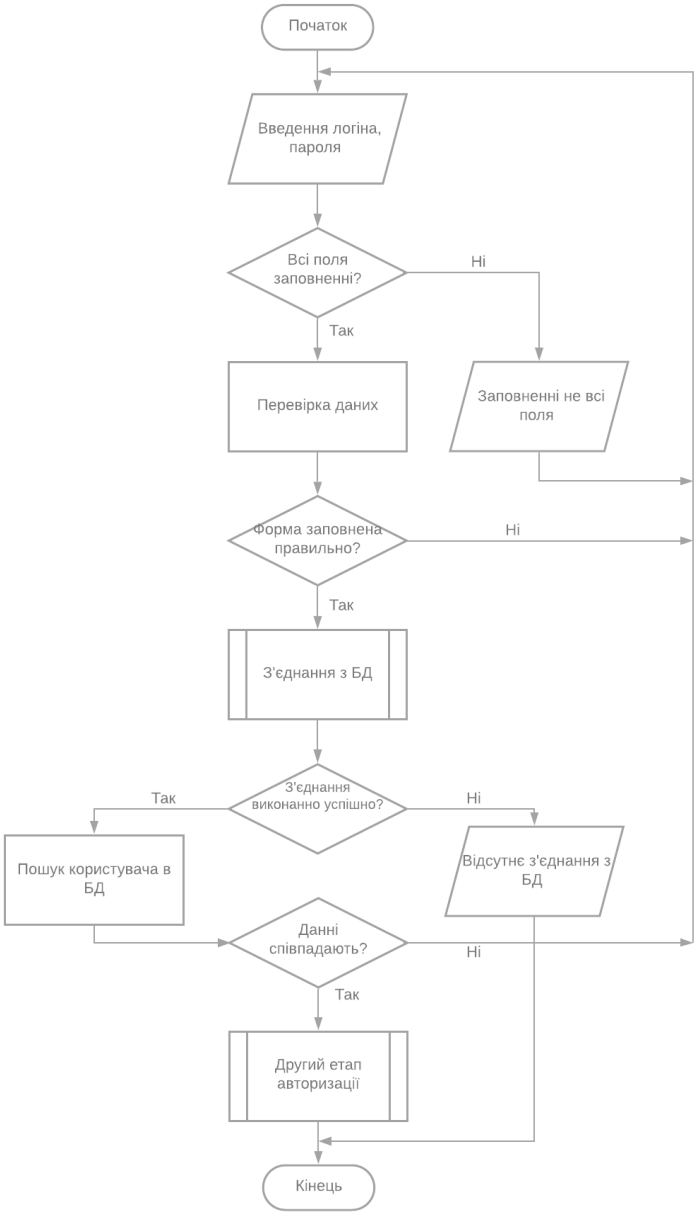



Рисунок 3.3 - Алгоритм першого етапу автентифікації

Робота даного алгоритму складається з наступного:

1. ведення логіну і паролю користувачем;
2. програма перевіряє чи заповнені всі поля, а саме поле логіну і паролю;
3. перевіряє чи коректно заповнені поля логіну і паролю;
4. захищене з’єднання з базою даних;
5. якщо з’єднання успішне то:
    a. пошук користувача в базі даних;
    b. перевірка ведених даних з даними що є в базі;
    c. перехід до другого етапу автентифікації.
6. якщо з’єднання не успішне то закриття програми.

При вході в особистий кабінет співробітника банку, співробітнику необхідно ввести свій логін і пароль, в разі правильності логіна і пароля, потрібно пройти другий етап автентифікації.

## 3.4 Алгоритм другого етапу автентифікації

Другий етап автентифікації показаний на рисунку 3.4.

Використовуючи дані першого етапу автентифікації (логін), з сервера запитується відкритий ключ шифрування який знаходиться в базі даних, попередньо розшифровуючи його. Потім користувач вводить секретний код, згенерований з допомогою токену автентифікації.

Потім програма обчислює значення хеш-функції яка складається з секретного коду, логіна та поточної дати, а потім шифрує значення секретного коду, дати, входу та значення хеш-функції.

Це повідомлення надсилається на сервер, де обчислюється його правильність. Якщо розшифроване повідомлення правильне, користувачеві буде дозволено доступ до системи, інакше їм доведеться повторити другий етап автентифікації.

Якщо користувач використавши 3 спроби входу не може пройти перевірку

інформація про нього, а саме: його ір-адреса, місце знаходження. Буде відправленне адміністратору з інформаційної безпеки. Який буде розглядати чи це працівник чи зловмисник, в разі чого блокувати цю людину.

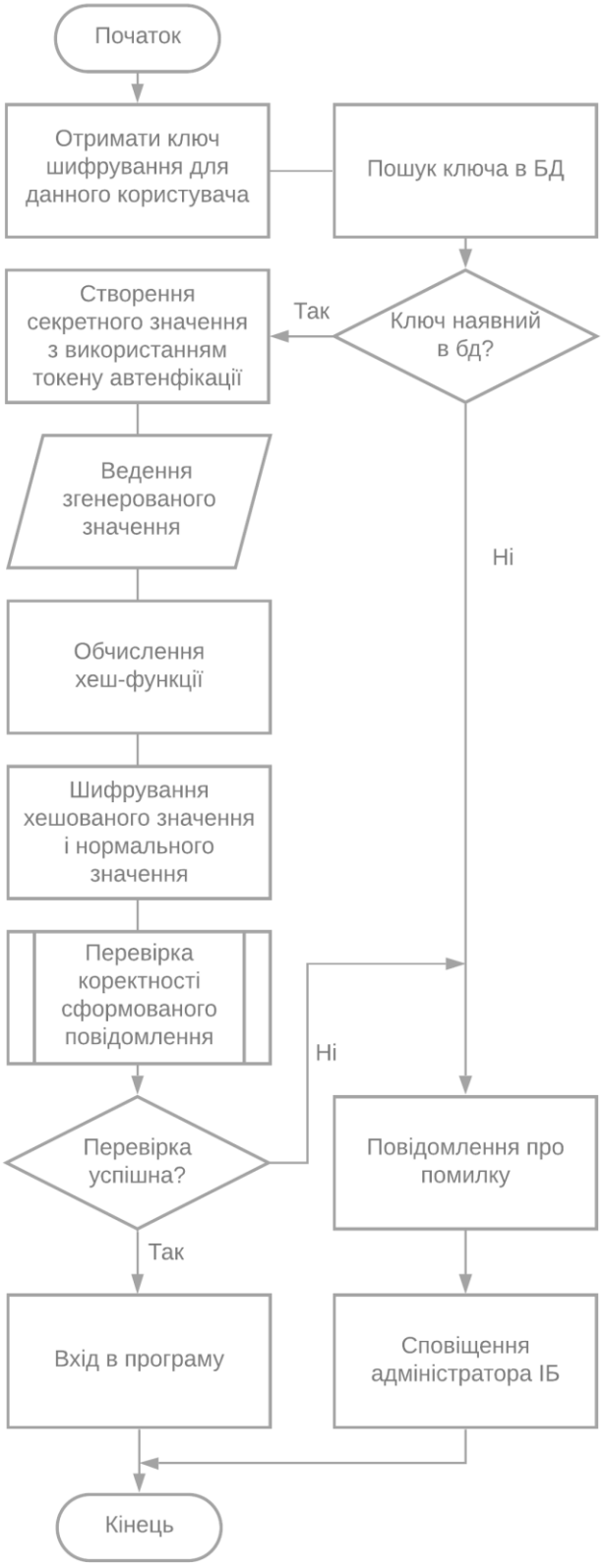



Рисунок 3.4 - Алгоритм другого етапу автентифікації

3.5 Генерація токену

Щоб пройти другий етап автентифікації на стороні серверу, використовуючи мову програмування JavaScript, потрібно вести токен автентифікації.

Використовуючи вхідні дані, ви отримуєте токен наступного виду:

Token=Login+EK+C(Numb+ Time-based +Login+H(Numb+ Time-based+Login)),

Де Token є токеном, правильність якого буде перевірено на стороні сервера.

E() є функцією шифрування, в цьому випадку використовується криптосистема RSA, ключі якої зберігаються на сервері в зашифрованому вигляді. Ключі шифруються криптосистемою AES, секретний ключ якої однаковий для всіх користувачів (у користувачів немає цього ключа).

K – відкритий ключ шифрування криптосистеми RSA для певного користувача.

Time-based — це поточна дата й час, потрібні для керування «часом життя» токену. Однією з вимог при розробці цієї системи є неможливість використовувати один і той же токен кілька разів.

Login - це логін користувача, який він використовував на першому етапі автентифікації.

H() є хеш-функцією для перевірки того, що частина повідомлення не була підроблена.

HMAC розшифровується як Hash-based Message Authentication Code, або код автентифікації повідомлень, який використовує хеш-функції. MAC - це механізм, що дозволяє верифікувати відправника повідомлення. Алгоритм MAC генерує MAC-тег, використовуючи секретний ключ, відомий тільки відправнику і одержувачу.

Отримавши повідомлення, ви можете згенерувати MAC-тег самостійно і порівняти два теги. Якщо вони збігаються все в порядку, втручання в процес

передачі повідомлення не було. Як бонус цим же способом можна перевірити, чи не пошкодилося повідомлення при передачі. Звичайно, відрізнити втручання від пошкодження не вийде, але досить самого факту пошкодження повідомлення.

Хеш це результат застосування до повідомлення хеш-функції. Хеш-функції беруть дані і роблять з нього рядок фіксованої довжини. Хороший приклад всім відома функція MD5, яка широко використовувалася для перевірки цілісності файлів.

Сам по собі MAC - це не конкретний алгоритм, а лише загальний термін. HMAC в свою чергу вже конкретна реалізація. Якщо точніше, то HMAC-X, де X - одна з криптографічних хеш-функцій. HMAC приймає два аргументи: секретний ключ і повідомлення, змішує їх певним чином, застосовує обрану хеш-функцію два рази і повертає MAC-тег.

Все це має відношення до одноразових паролів, ми майже дійшли до найголовнішого.

Згідно зі специфікацією, HOTP обчислюється на основі двох значень:

K - секретний ключ, який знають клієнт і сервер. Він повинен бути мінімум 128 біт довжиною, а краще 160, і створюється коли, ви налаштовуєте 2FA.

C - лічильник.

Лічильник це 8-байтовое значення, синхронізоване між клієнтом і сервером. Нове воно, у міру того як ви генеруєте нові паролі. У схемі HOTP лічильник на стороні клієнта використовується кожен раз, коли ви генеруєте новий пароль. На стороні сервера кожен раз, коли пароль успішно проходить валідацію. Оскільки можна згенерувати пароль, але не скористатися ним, сервер дозволяє значенням лічильника трохи забігати вперед у межах встановленого розміру. Однак якщо ви занадто загралися з генератором паролів в схемі HOTP, доведеться синхронізувати його повторно.

Цілком очікувано, K використовується в якості секретного ключа. Лічильник, в свою чергу, використовується в якості повідомлення. Після того, як HMAC функція згенерує MAC-тег, загадкова функція Truncate витягує з

результату вже знайомий нам одноразовий пароль, який ви бачите в своєму токені.

Time-based означає, що замість статичного значення, як лічильник використовується поточний час. Або, якщо точніше, "інтервал" (time step). Або навіть номер поточного інтервалу. Щоб обчислити його, ми беремо Unix-час (кількість мілісекунд з півночі 1 січня 1970 року по UTC) і ділимо на вікно валідності пароля (зазвичай 30 секунд). Сервер зазвичай допускає невеликі відхилення через недосконаліть синхронізації годин. Зазвичай на 1 інтервал вперед і назад в залежності від конфігурації.

Очевидно, це набагато безпечніше, ніж схема HOTP. У схемі, зав'язаною на час, валідний код змінюється кожні 30 секунд, навіть якщо він не був використаний. В оригінальному алгоритмі валідний пароль визначено поточним значенням лічильника на сервері + вікном допуску. Якщо ви не авторизуєтеся, пароль не буде змінений нескінченно довго. Більше про TOTP можна почитати в RFC6238.

Оскільки схема з використанням часу - це доповнення до оригінального алгоритму, нам не потрібно буде вносити зміни до початкової реалізації. Ми скористаємося requestAnimationFrame і будемо на кожен фрейм перевіряти, до цього часу ми знаходимося всередині тимчасового інтервалу. Якщо немає - генеруємо новий лічильник і обчислюємо HOTP заново.

Щоб роздобути одноразові паролі, нам знадобиться виконати наступні кроки.

Згенерувати HMAC-SHA1 хеш з параметрів K і C. Це буде 20-байтовая рядок.

Витягнути 4 байти з цього рядка певним чином.

Здійснити конвертацію витягнуті значення в число і поділити його на 10 ^ n, де n = кількість цифр в одноразовому паролі (зазвичай n = 6). Ну і, нарешті, взяти залишок від цього поділу. Це і буде наш пароль.

## 3.6 Алгоритм перевірки коректності токену автентифікації

Алгоритм перевірки токену автентифікації показаний на рисунку 3.5.

Коли ви перевіряєте, що повідомлення має правильний формат, а саме з нього ви можете витягти логін користувача. Використовуючи логін цього користувача, секретний ключ шифрування RSA витягується з бази даних. За допомогою секретного ключа повідомлення розшифровується.

Перевіряється правильність розшифрованого повідомлення (обсяг даних і самі дані повинні бути правильними). Значення хеш-функції розраховується з параметрів, переданих в повідомленні, і порівнюється з переданим значенням хеш-функції, якщо вони однакові, дані не були замінені.

Ім'я користувача, зашифроване в повідомленні та не зашифрованому імені користувача, перевіряється. Після цього потрібно перевірити, що між часом надсилання та часом, який знадобився для отримання повідомлення, пройшло не більше 30 сукунд. Завдяки цьому не можна вводити один і той же токен кілька разів.

Потім перевіряється правильність згенерованого коду. Для перевірки правильності коду використовується алгоритм для створення коду на сервері, відповідно до відомої кількості успішного створення коду на сервері. Електронний токен генерує секретний код за допомогою тих же алгоритмів, але враховує загальну кількість генерацій секретного коду. У самому пристрої і на сервері таке ж значення використовується як відправна точка для подальшого створення секретного коду, який при необхідності може бути синхронізований адміністратором.

Якщо коди, згенеровані на сервері і за допомогою електронного пристрою, однакові, користувачеві дозволяється доступ до системи з відповідними правами.

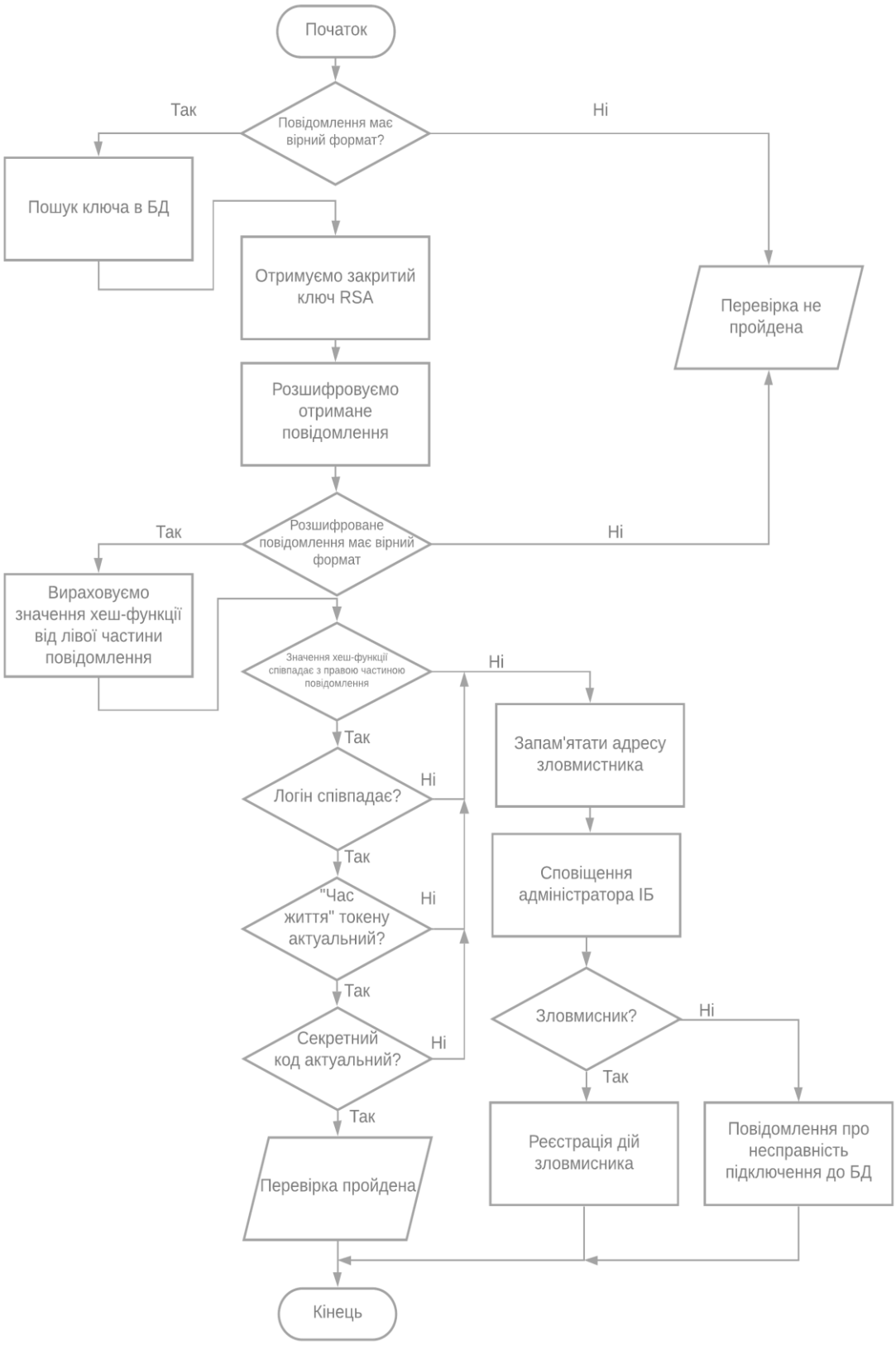



Рисунок 3.5. Алгоритм перевірки токена автентифікації

## 3.7 Технічне завдання на розроблювану систему

### 3.7.1 Загальні відомості

Назва розроблюваної системи: «Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів»

### 3.7.2 Мета і призначення розроблюваної системи

Основною метою є захист чинної системи документообігу «Askod» додатковим модулем захисту двофакторної автентифікації. Який повинен розширити функції захисту системи.

Даний модуль для посилення надійності входу в особистий кабінет співробітника системи «Askod».

Програма має наступні можливості.

1. Перевірка коректності введеного логіна і пароля.

2. Перевірка коректності введеного згенерованого коду.

3. Здійснює двофакторну автентифікацію входу в особистий кабінет співробітника системи «Askod».

### 3.7.3 Структура модулю

Модуль складається з декількох файлів. У файлі «HomeService.cs» описані функції для перевірки коректності секретного коду, а також функції необхідні для отримання відкритого ключа шифрування криптосистеми RSA.

У файлі «AccountController.cs» описані методи контролера, необхідні для відображення сторінки авторизації, а також виклик функцій отримання ключа шифрування і перевірки коректності секретного коду.

У файлі «_LoginForm.cshtml» описана розмітка форми входу в особистий кабінет співробітника системи «Askod».

У файлі «Encrypt.JS» описані функції необхідні для реалізації шифрування на стороні клієнта з використанням мови програмування JavaScript.

У файлі «SHA.js» описані функції необхідні для реалізації обчислення значення хеш-функції SHA-3.

У файлі «2FA.js» знаходиться код реалізації створення токена автентифікації в обфусцифікаційному вигляді.

3.7.4 Опис функціонування системи

Робота програми відбувається відповідно до алгоритму, представленим на рис. 3.1.

У файлі «HomeService.cs» описані функції для перевірки коректності секретного коду, а також функції необхідні для отримання відкритого ключа шифрування криптосистеми RSA.

У файлі «AccountController.cs» описані методи контролера, необхідні для відображення сторінки авторизації, а також виклик функцій отримання ключа шифрування і перевірки коректності секретного коду.

У файлі «_LoginForm.cshtml» описана розмітка форми входу в особистий кабінет співробітника системи «Askod».

У файлі «Encrypt.JS» описані функції необхідні для реалізації шифрування на стороні клієнта з використанням мови програмування JavaScript.

У файлі «SHA.js» описані функції необхідні для реалізації обчислення значення хеш-функції SHA-3.

У файлі «2FA.js» знаходиться код реалізації створення токена автентифікації в обфусцифікаційному вигляді.

Якщо всі перевірки пройдені, то користувачеві дозволено доступ в систему з відповідною йому роллю.

3.7.5 Захист модулю

Для початку ми виробляємо простий виклик-відповідь. Сервер посилає нам випадковий виклик. Наш пристрій підписує виклик і повертає підпис сервера, після чого сервер звіряє підпис.

Сам по собі виклик-відповідь не вирішує проблеми фішингу, оскільки ви авторизуєтеся на rnail.ru замість mail.ru, то ваш підпис все ще може бути використана для входу в ваш особистий кабінет. Для захисту від цього браузер до виклику додає URL, з якого був зроблений запит на підпис, і ID каналу TLS, після чого залежна сторона звіряє ці дані.

На цей момент наш пристрій підписує все однією парою ключів. Це створює проблему для приватності, в зв'язку з тим що публічний ключ буде скрізь однаковий. Для прикладу скажемо якби ви були зареєстровані на сумнозвісному AshleyMadison.com, то зловмисник міг би зв'язати злитий публічний ключ і ваші інші акаунти і потенційно завдати фізичну і моральну шкоду вам і ваши близьким.

Щоб зберегти приватність при реєстрації, залежна сторона передає ID додатки (AppID) і насіння (випадкове число). На основі цих даних пристрій генерує унікальну реєстраційно-залежну пару ключів. Як пристрій генерує пару не описано в протоколі, а повністю віддано на розсуд виробника пристрою. Наприклад, кожен Yubikey має свій майстер ключ, який в зв'язці з HMAC і генератор випадкових чисел генерує нове число.

Користуючись тим, що пара ключів унікальна для кожної реєстрації, стає можливим використовувати спільно одне U2F пристрій для безлічі акаунтів.

Через те, що U2F це тільки протокол, то він може мати різні інтерпретації, в вашому коді і ПО. Деякі інтерпретації можуть бути не стійкими до клонування. Для захисту від цього U2F пристрій має вбудований лічильник. Кожен підпис і реєстрація збільшує стан лічильника на одиницю. Стан лічильника підписується і повертається залежній стороні. Якщо U2F пристрій було скопійовано, то стан лічильника скопійованого пристрою швидше за все буде менше ніж стан лічильника оригінального пристрою, що викличе помилку під час верифікації.

Різні інтерпретації протоколу можуть бути небезпечні. Щоб уникнути цього, кожне U2F пристрій має вбудований персональний сертифікат, який встановлюється приблизно на кожні сто тисяч пристроїв. Кожен підпис і реєстрація додатково підписується сертифікатом, публічний ключ якого знаходиться в публічній директорії.

Наприклад, якщо ви заходите на форум про кошенят, то ви можливо не сильно хвилюєтеся про те, наскільки безпечні U2F пристрої ваших користувачів, а якщо відвідуєте сайт банку, то можливо ви дозволите підключення тільки тим пристроям, які сертифіковані FIDO альянсом.

У ситуації, коли користувач знаходиться далеко від свого пристрою, зловмисник за допомогою шкідливого програмного забезпечення може спробувати атакувати ваш пристрій методом повного перебору або іншими видами атак. Для захисту від цього U2F стандарт вимагає щоб всі інтерпретації, які були створенні користувачем. Користувач зобов'язаний підтвердити своє рішення на двофакторну автентифікацію. Цією дією може бути просте натискання на кнопку, введення пін-коду, зняття відбитків пальців або інше.

## 3.8 Висновки

Розроблено алгоритми функціонування системи, а саме алгоритм першого етапу автентифікації, другий етап автентифікації, алгоритм перевірки правильності токенів, також наведено схему створення токенів.

На першому етапі автентифікації користувачеві необхідно ввести логін і пароль. Сервер перевіряє правильність введених даних, якщо вони правильні, користувачеві потрібно пройти другий етап автентифікації. Якщо вони не правильні то надсилаються данні для адміністратора інформаційної безпеки.

Під час другого етапу автентифікації користувачеві необхідно згенерувати секретний код і ввести його в текстове поле і натиснути кнопку "Війти". Потім на стороні клієнта буде згенерований токен авторизації, який включає логін, секретний код, дату, значення хеш-функції з логіна, дати та секретного коду. Логін потрібен для визначення ключів шифрування, дата для перевірки правильності тривалості токену (30 секунд).

При генерації токену використовується криптосистема RSA, відкритий ключ якої знаходиться в базі в зашифрованому вигляді. Для шифрування криптосистеми RSA використовується криптосистема AES, ключі якої недоступні користувачам, оскільки вони використовуються лише на зворотному боці.

Хоча криптосистема RSA повільніша за криптовалюту AES, а довжина

ключа набагато довша, використовувати криптовалюту AES для шифрування токену неможливо. Ключі шифрування на клієнті та серверах у криптосистемі AES однакові, але будь-хто має доступ до клієнтської сторони, тому ви можете перехопити повідомлення будь-якого користувача та підробити його, знаючи ключ. Додатковим захистом від підробки є наявність хеш-функції, яку можна перевірити, щоб переконатися, що повідомлення не було підроблено.

Під час перевірки токену автентифікації слід переконатися, що повідомлення, яке отримує сервер, має правильний формат. Логін користувача, який містить секретний ключ шифрування криптосистеми RSA, потім витягується з отриманого повідомлення, частина якого використовувалася для шифрування даних. Цей ключ зберігається зашифрованим, тому його слід спочатку розшифрувати. За допомогою цього ключа розшифровується вихідне повідомлення, після чого перевіряється правильність розшифровки. Перевіряється дійсність значення хеш-функції, обчисленого на сервері, а також значення функції, переданого в повідомлення.

В самому кінці перевіряється правильність секретного згенерованого коду. Завдяки тому ж алгоритму створенні секретного коду на сервері і за допомогою електронного пристрою, правильність коду легко встановити, а в разі помилок пристрій можна синхронізувати з сервером.

Якщо всі перевірки зроблені, користувачеві дозволяється доступ до системи, з відповідною роллю.

.

# 4 РЕАЛІЗАЦІЯ РОБОТИ

## 4.1 Результат роботи програми

На рисунку 4.1. представлена форма першого етапу автентифікації. Користувачеві необхідно ввести логін і пароль, натиснути кнопку увійти. Якщо логін і пароль правильні, то користувачеві необхідно ввести код згенерований пристроєм (рисунок 3.2), якщо логін і пароль введені неправильно, то з'явиться відповідне повідомлення (рисунок 3.3).

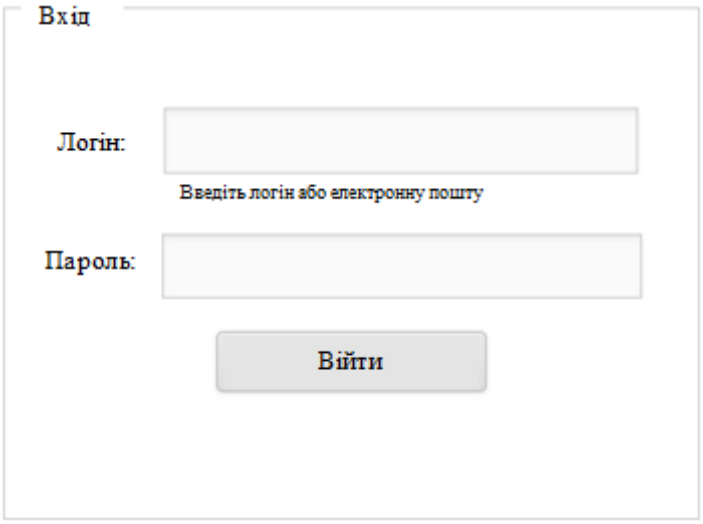



Рисунок 4.1 - Вікно першого етапу автентифікації

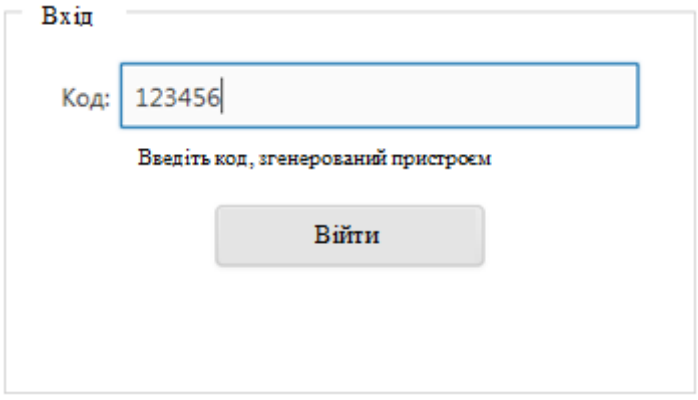



Рисунок 4.2 - Вікно введення, згенерованого коду.

Якщо введений код правильний, то користувачеві доступний вхід в систему з відповідною йому роллю.

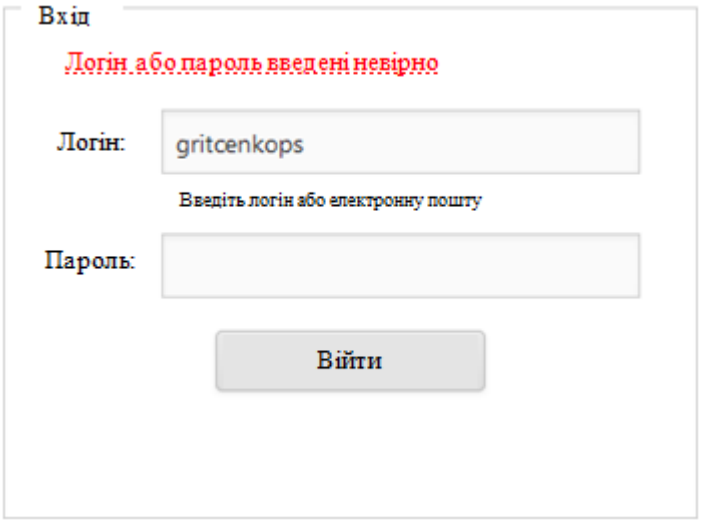



Рисунок 4.3 - Повідомлення про помилку

Для створення токена автентифікації, необхідно отримати відкритий ключ шифрування криптосистеми RSA. Для отримання такого ключа необхідно знайти його в базі і розшифрувати використовуючи криптосистему AES з ключами (наведені випадкові, що не дійсні ключі):

ключ=10223740DD127EE303E3C95090065D12CB0B83F361BF713D5AD7A FEB B5BEA0A8;

вектор ініціалізації= E5FA1B493ABB7A97F7EEFCDB553BFE66

Для даного користувача ключ RSA в зашифрованою формі має такий вигляд (приведена частина ключа):

EB39EB1589…ECD5D3C2C4E4A4B8391E897F5248333B3DA8FC5C66D25

Довжина ключа в зашифрованому вигляді 3360 символів. У розшифрованому вигляді ключ має вигляд:

<RSAKeyValue>

<Modulus>qyBMRKWzj…ZinPpWQ==</Modulus>

<Exponent>AQAB</Exponent>

<P>1UmgEWTuK0iLc…S3sqGDne8pBsrg6908=</P>

<Q>zWU63LXQM8I…F0XvkePcVw68g6tc=</Q>

<DP>yTKAuqfZPB9…MqEdQHyXCL9QlxJmU=</DP>

<DQ>WoxyMrMEo…kSNDHZqnceb+k=</DQ>

<InverseQ>q7yoE…Xdv6lCRis3zk=</InverseQ>

<D>gc1i7KM/24GCf…loaKtN2kFJVf5pqgCJsry457FG6jQ==</D>

</RSAKeyValue>

З даного ключа витягується модуль і експонента у вигляді рядка містить шістнадцяткові значення. За допомогою цих рядків на клієнті шифрується токен.

Для запобігання даних від підробки використовується хеш-функція SHA-3, значення якої обчислюється на стороні клієнта і на стороні сервера. Приклад значення хеш-функції для частини токена:

частина токену=gritcenkops2021-06-15T12:44:44.731Z123456;

хеш=5833f90f05ae7aa76a9301e7eb14b987de1da1367de74a3e1c408ee5e0f6c8a13d6201a9c7296dfcdcc74c51f9e2c9718e8679f971c2310f4049532432v232s412dv421r413e5se4g325fcf4fa38b6c

Далі обчислюється весь токен, який має наступний вигляд:

gritcenkops2021-06-15T12:44:44.731Z|123456|5833f90f05ae7aa76a9301e7e-b14b987de1da1367de74a3e1c408ee5e0f6c8a13d6201a76fa89gcx76896cxz6787v8z67zx67zx8c8v9xc6b7xc689c7296dfcdcc74c51f9e2c9dasf352f718e8679f971c2310f40495fcf4fa38b6c

На стороні клієнта токен шифрується за допомогою раніше отриманого відкритого ключа криптосистеми RSA.

Зашифровану частину токена має наступний вигляд:

VoiwcAABOY+F0p0wkG7eL5DR3od9ozkMqjkhqbXvDAMA/PDVQZr0x99SMn+Dv+ikchrhGeeAV9sidRiVxzHnzYBn5m1bTPN8iq9zFEC28OwolVmrUMu8PNWV5/fGkoTfcA4apbYypHC3ll1Bh6fJ24UionizIIIlcmhuj5s2K5dfM86e8Nt0IHMZuwJG1LrzuJc1UBBdJWBKfYdGAVgJADKvbnhGy+Zqqi65EHcqCS2hn6/RyP13HXYvRrSvSMFHq5FAYo7wpGIIQ5OBGnnxb5l8x86fw1rFkYf6Ezi4mWzjTtOCq6uHzN4n59wO VAeLn9AbFZxejiVN//3Hu8UoA==

Перед зашифрованою частиною токена вставляється ім'я користувача і весь цей токена передається на сервер, де перевіряється його коректність. Для цього для користувачу потрібно ввести його секретний ключ системи RSA

(попередньо розшифровується сам ключ у програмі), розшифровується токен, перевіряється коректність значення хеш-функції, "час життя" токена, коректність секретного коду.

## 4.2 Висновки

Розроблено для користувача інтерфейс для двофакторної системи автентифікації особистого кабінету співробітника системи «Askod».

На першому етапі користувачеві необхідно ввести логін і пароль, в разі якщо вони коректні, то користувачеві необхідно пройти другий етап автентифікації, в іншому випадку необхідно повторно ввести логін або пароль.

Для даного користувача на сервері знаходиться ключ шифрування криптосистеми RSA, який зберігається в зашифрованому вигляді. Ключі шифрування RSA зашифровані криптосистемою AES, для забезпечення надійності зберігання. Після розшифровки ключів системи RSA на клієнтську сторону відправляється відкритий ключ, який складається з модуля і експоненти.

На другому етапі автентифікації користувачеві необхідно ввести секретний код, згенерований пристроєм для автентифікації.

Обчислюється значення хеш-функції від імені користувача, поточної дати і часу, секретного коду. Генерується токен, що складається з імені користувача, поточної дати і часу, секретного коду і значення хеш-функції, обчисленого раніше. Цей токен шифрується за допомогою криптосистеми RSA відкритим ключем, який був отриманий перед другим етапом автентифікації. До зашифрованого токені додається ім'я користувача, для того щоб на сервері можна було розшифрувати даний токен.

На сервері з отриманого токена витягується ім'я користувача, знаходиться закритий ключ шифрування криптосистеми RSA, розшифровується токен, перевіряється правильність переданого значення хеш-функції, а також коректність секретного згенерованого коду. Завдяки однаковому алгоритму

створення секретного коду на сервері і за допомогою електронного пристрою, правильність коду легко встановити, а в разі помилок, пристрій можна синхронізувати з сервером.

Під час не проходження перевірки токену інформація про спробу входу відправляється адміністратору, а також фіксується в базі даних.

# ВИСНОВКИ

Використання паролів найпоширеніша форма автентифікації. Більшість людей, що користуються мережею Інтернет, використовують паролі для доступу до різних сервісів, таких як поштовий ящик, інтернет-банкінг, онлайн магазини та ін. Однак паролі одна з найменш захищених форм автентифікації. Їх можна легко забути, якщо вони становлять собою безглуздий набір цифр і букв, в іншому випадку їх можна зламати. Двофакторна автентифікація дозволяє посилити захищеність входу в особистий кабінет чнонз те, що зловмисникові крім логіна і пароля жертви, буде потрібно щось, чим володіє тільки дана людина.

Розглянуто види двофакторної автентифікації: SMS-код, програми автентифікатор, перевірка входу за допомогою мобільних додатків і апаратні токени. Основними недоліками розглянутих видів автентифікації є висока вартість реалізації, відсутність смартфона у користувачів системи «Askod» або персонального акаунту у соціальних мережах.

На основі аналізу видів двофакторної автентифікації, а також чинного законодавства й архітектури нового особистого кабінету співробітника системи «Askod» здійснено вибір засобів і технологій для реалізації поставленого завдання. Для реалізації поставленого завдання вибрана мова програмування C # для серверної сторони, на клієнтській стороні використовується JavaScript, програма має архітектуру MVC.

Розроблена система крім основних функціональних вимог задовольняє також вимогам безпеки та вимогам продуктивності, а саме неможливість зайти в систему кілька разів з використанням одного і того ж токена.

Клієнтська сторона доступна всім, отже, використовувати симетричні системи шифрування не можна, тому для шифрування токена на клієнтській стороні використовується криптосистема RSA. Однак, спрощену схему шифрування RSA використовувати також можна через її уразливості, тому використовується ускладнена схема.

Серверна сторона недоступна користувачам, тому нема потребив шифруванні ключів з використанням асиметричних криптосистем. Проаналізовано криптосистема AES, за допомогою якої зашифровані ключі криптосистеми RSA, які зберігаються в базі. Для серверної сторони використання криптосистеми AES для шифрування ключів гарантує швидкодію, а також менший розмір ключа, в порівнянні з криптосистемою RSA.

Наведено схему роботи алгоритму хешування SHA-3, який використовується для підтвердження того факту, що токен не був змінений зловмисником. Використання інших алгоритмів хешування недоцільно зважаючи на наявність вразливостей в них, за допомогою яких можна підібрати потрібне значення хеш-функції.

Токен авторизації створюється на стороні клієнта, отже, вихідний код доступний всім користувачам. Наведено спосіб захисту коду від аналізу, модифікації і розуміння роботи алгоритмів, а саме обфускація.

Наведено загальний алгоритм роботи програми і алгоритми роботи першого, другого етапів автентифікації, а також алгоритм перевірки токена. Розглянуто схему створення токена автентифікації. Завдяки однаковому алгоритму створення токена на сервері і за допомогою електронного пристрою, перевірка секретного коду здійснюється без особливих складнощів. Пристрій для створення коду можна синхронізувати з сервером, в разі виникнення проблем.

На прикладі розглянуто процес створення токена, а також наведені проміжні дані етапів розшифровки ключа криптосистеми RSA. Наведено загальний вигляд ключа шифрування, значення хеш-функції для конкретного токена, а також зашифровані токени.

Перевагою розробленої системи є можливість налаштування системи в разі зміни тих чи інших правил, можливість зміни ключів шифрування для конкретного користувача, стійкість системи в цілому до витоку токена або ключа шифрування одного або декількох користувачів, фіксування всіх невдалих спроб входу в базі даних. Також фіксується адреса комп’ютера в мережі. Цю інформацію може переглядати адміністратор інформаційної безпеки та на основі

цих даних проводити блокування користувачів.

Недоліком розробленої системи, є складність налагодження клієнтської частини через наявність заплутаного коду який був створений за допомогою методу обфускації.

Даний модуль захисту введений в систему співробітників банку «Askod». Зазначені недоліки не впливають на функціонал системи, але є необхідними для безпечного використання даної системи.

## ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ


1. “Системний аналіз категорії “загроза” в інформаційній безпеці” Сьомак Р.В. 2019.

2. Засоби захисту інформації А. Г. Чубенко, М. В. Лошицький, Д. М. Павлов, С. С. Бичкова, О. С. Юнін. Київ : Ваіте, 2018. С. 175; 273.

3. Качинський А.Б. Безпека складних систем.-К.: ТОВ “Видавництво “Юстон”, 2017.498 с.

4. Інформаційна безпека людини як споживача телекомунікаційних послуг: Монографія / І. В. Арістова, Д. В. Сулацький ; НДІ інформатики і права НАПрН України. К. : Право України ; Х. : Право, 2013. 184 с.

5. Кормич Б. А. Інформаційна безпека: організаційно-правові основи: Навч. посібник. К.: Кондор, 2004. 384 с.

6. Кормич Б. А. Організаційно-правові основи політики інформаційної безпеки України: Автореф. дис. д-ра юрид. наук: 12.00.07. Х.: НХУ України, 2004. 150 с.

7. Сідак В. С., Артемов В. Ю. Забезпечення інформаційної безпеки в країнах НАТО та ЄС: Навчальний посібник. К.: КНТ, 2007. 87с.

8. Харченко В. С. Інформаційна безпека. Глосарій. К.: КНТ, 2005. 65 с.

9. Цимбалюк В.С. Проблеми державної інформаційної політики: гармонізація міжнародного і національного інформаційного права // Правове, нормативне та метрологічне забезпечення системи захисту інформації в Україні. К.: НТУУ «КПІ», 2001. № 4. 98 с.

10. Кузюрін М. М., Фомін С. А. Ефективні алгоритми та складність обчислень. 2008 165 с.

11. Немирівський А. С., Юдін Д. Б. Складність і ефективність методів оптимізації. М.: Наука, 1979 135 с.

12. Генадій Гулак, Володимир Бурячок, Павло Складанний - Швидкий алгоритм генерації підстановок багатоалфавітної заміни. 64 с.

13. «Енциклопедія кібернетики», відповідальний ред. В. Глушков, 1 тт., 1973, Алгоритм. 42 с.

14. Микола Глибовець. Основи комп'ютерних алгоритмів. Видавничий дім «Києво-Могилянська Академія», 2003. 452 с.

15. Кваліфікаційна робота бакалавра . Методичні вказівки щодо її виконання для студентів спеціальності 125 “Кібербезпека” освітнього рівня “бакалавр”. Джулій В.М., Муляр І.В., Чешун В.М 94 с.

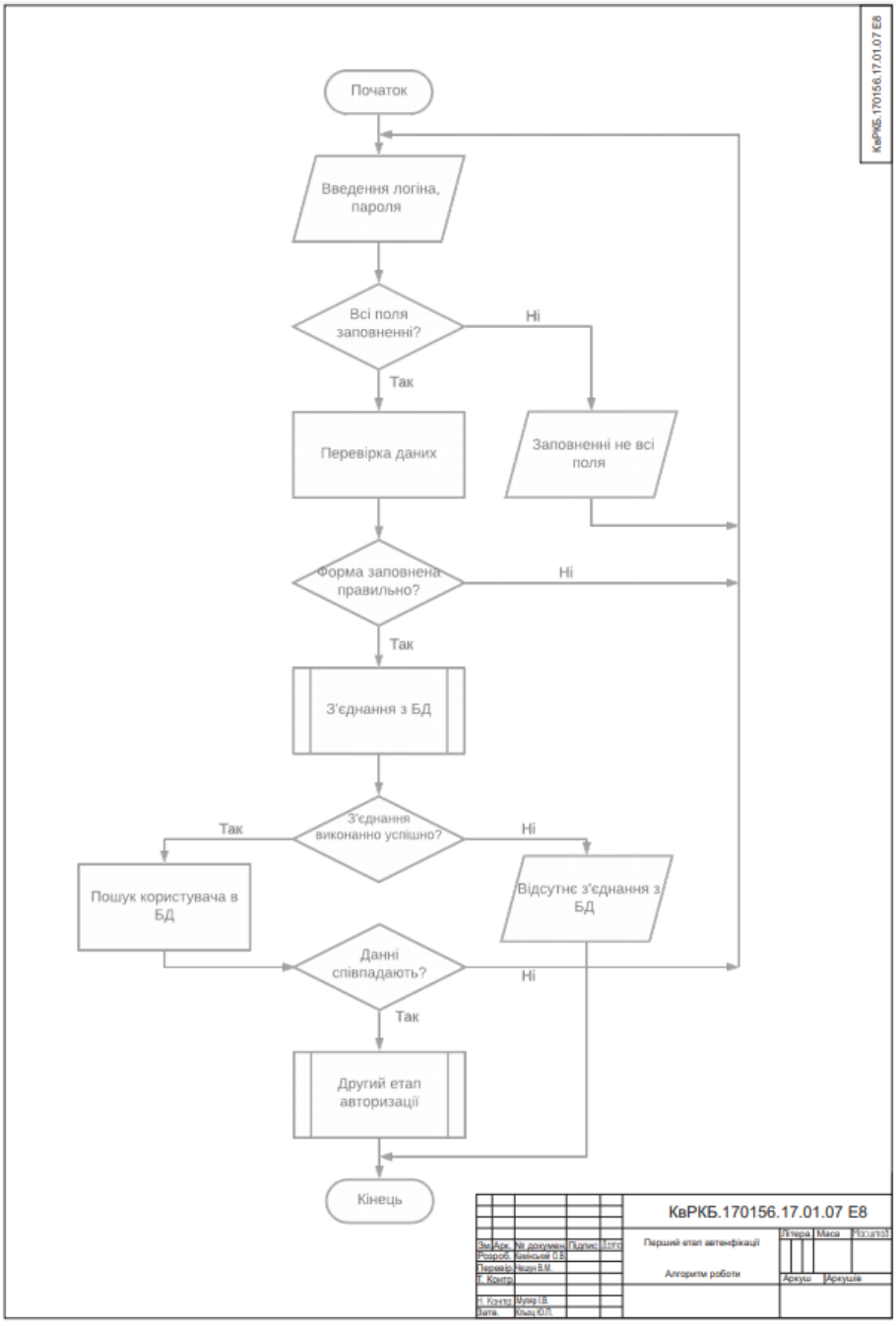
16. Методи і алгоритми захисту інформаційних ресурсів комп'ютерних систем: навчальний посібник / В. М. Джулій, Ю. П. Кльоц, І. В. Муляр, В. М. Чешун. Хмельницький: ХмНУ, 2020. 196 с.

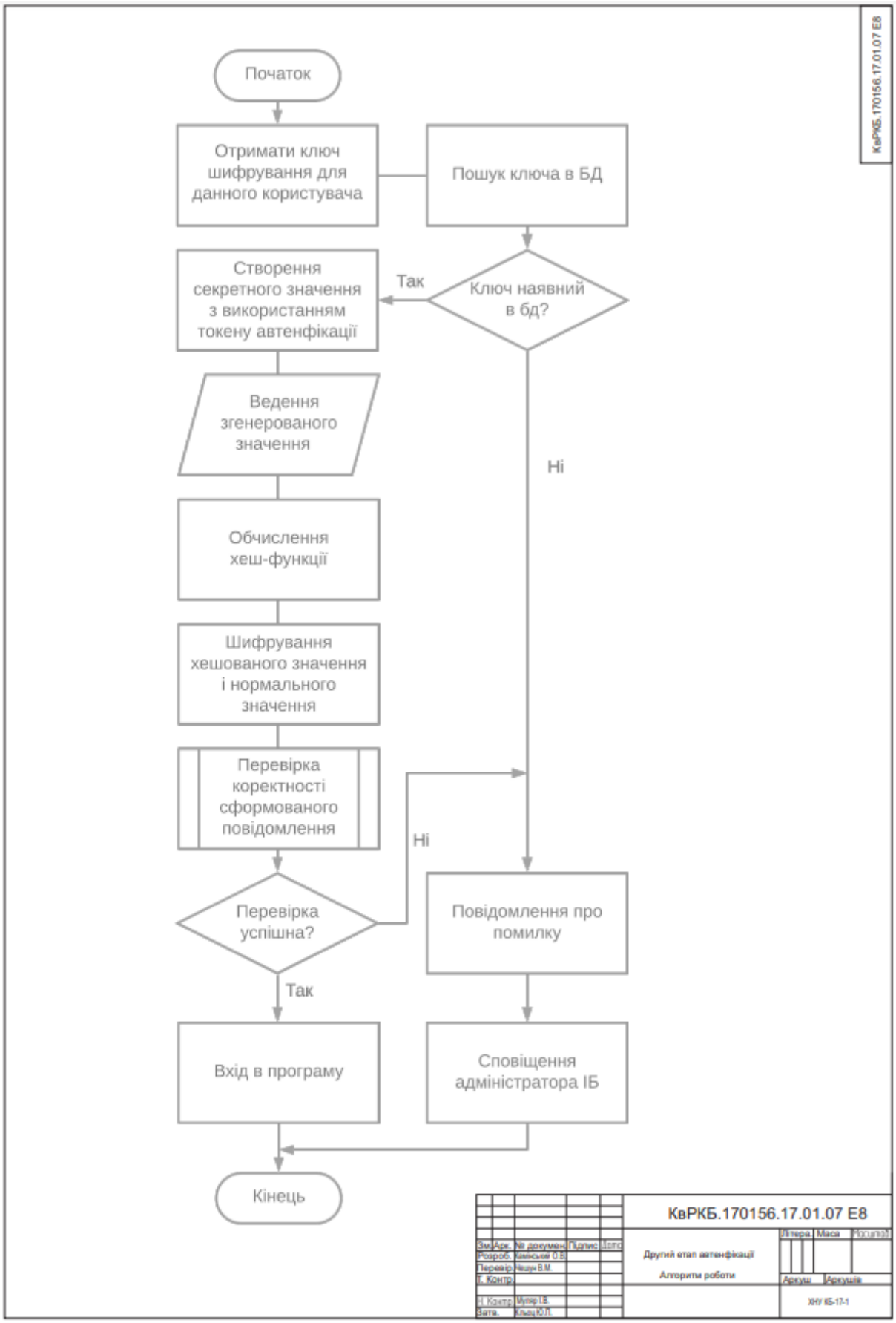
17. Сімейство функцій Keccak, [Єлеткронний ресурс]. - Режим доступу: http://keccak.noekeon.org, вільний - Дата обробки 14.05.2021.

ДОДАТОК А

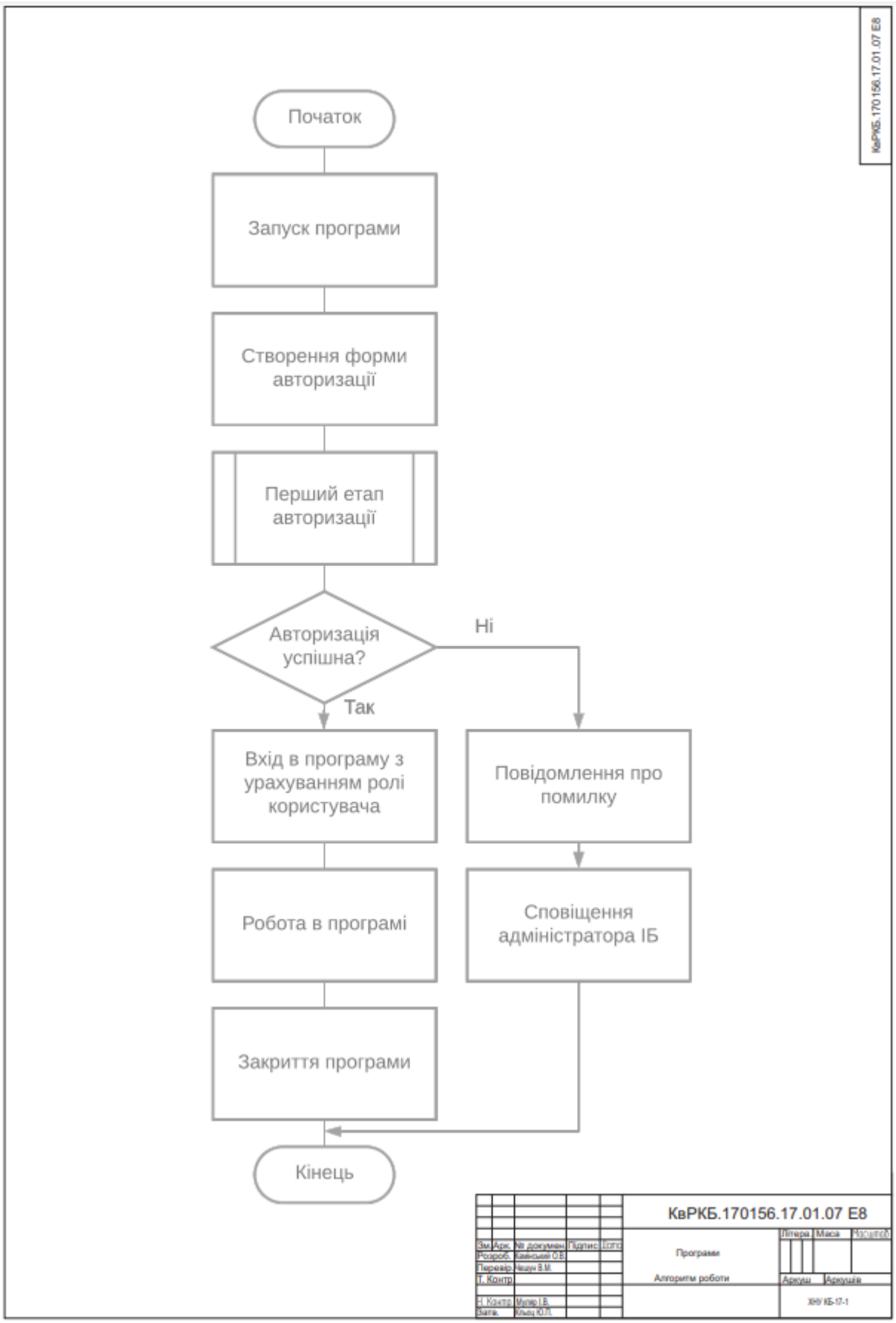
(Обов’язковий)

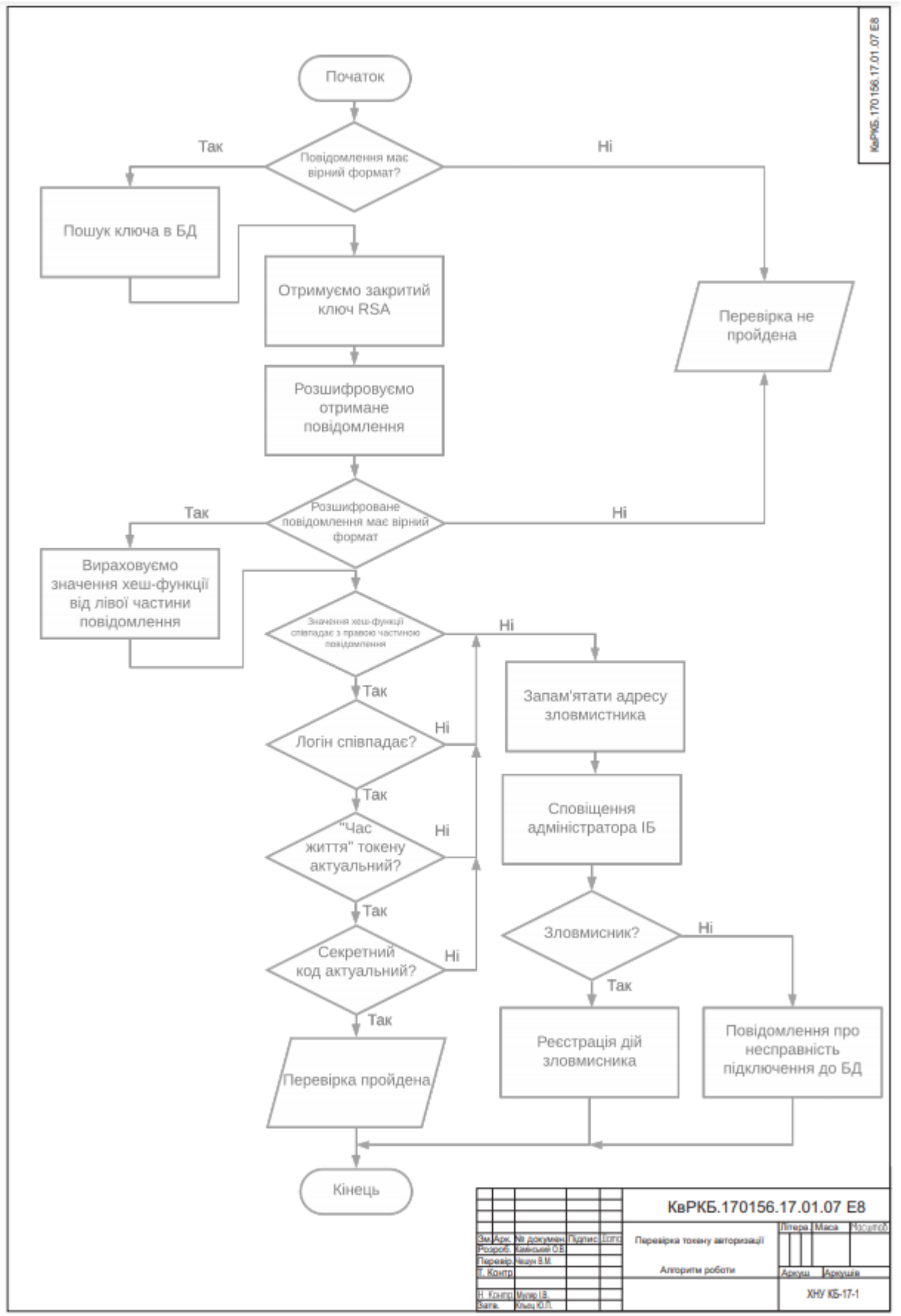
Копія графічної частини




КвРКБ.170156.17.01.07 Е8
Початок
Отримати ключ шифрування для данного користувача
Пошук ключа в БД
Створення секретного значення з використанням токену автенфікації
Так
Ключ наявний в бд?
Ведення згенерованого значення
Ні
Обчислення хеш-функції
Шифрування хешованого значення і нормального значення
Перевірка коректності сформованого повідомлення
Ні
Перевірка успішна?
Повідомлення про помилку
Так
Вхід в програму
Сповіщення адміністратора ІБ
Кінець
КвРКБ.170156.17.01.07 Е8
Другий етап автенфікації
Алгоритм роботи
Літера
Маса
Масштаб
Аркуш
Аркушів
ХНУ КБ-17-1
Зм.
Арк.
№ докумен.
Підпис
Дата
Розроб.
Камінський О.В.
Перевір.
Чешун В.М.
Т. Контр.
Н. Контр.
Мулар І.В.
Затв.
Кльоц Ю.Л.


КвРКБ.170156.17.01.07 Е8
Початок
Запуск програми
Створення форми авторизації
Перший етап авторизації
Другий етап авторизації
Вхід в програму з урахуванням ролі користувача
Робота в програмі
Закриття програми
Кінець
КвРКБ.170156.17.01.07 Е8
Модифікованої програми
Алгоритм роботи
Літера
Маса
Масштаб
Аркуш
Аркушів
ХНУ КБ-17-1


КвРКБ.170156.17.01.07 Е8
Початок
Запуск програми
Створення форми авторизації
Перший етап авторизації
Авторизація успішна?
Ні
Так
Вхід в програму з урахуванням ролі користувача
Повідомлення про помилку
Робота в програмі
Сповіщення адміністратора ІБ
Закриття програми
Кінець
КвРКБ.170156.17.01.07 Е8
Програми
Алгоритм роботи
Літера
Маса
Масштаб
Аркуш
Аркушів
ХНУ КБ-17-1


КвРКБ.170156.17.01.07 Е8
Початок
Повідомлення має вірний формат?
Так
Ні
Пошук ключа в БД
Отримуємо закритий ключ RSA
Перевірка не пройдена
Розшифровуємо отримане повідомлення
Розшифроване повідомлення має вірний формат
Так
Ні
Вираховуємо значення хеш-функції від лівої частини повідомлення
Значення хеш-функції співпадає з правою частиною повідомлення
Ні
Так
Запам'ятати адресу зловмистника
Логін співпадає?
Ні
Так
Сповіщення адміністратора ІБ
"Час життя" токену актуальний?
Ні
Так
Зловмисник?
Ні
Так
Секретний код актуальний?
Ні
Так
Реєстрація дій зловмисника
Повідомлення про несправність підключення до БД
Перевірка пройдена
Кінець
КвРКБ.170156.17.01.07 Е8
Перевірка токену авторизації
Алгоритм роботи
Літера
Маса
Масштаб
Аркуш
Аркушів
ХНУ КБ-17-1
Зм. Арк. № докумен. Підпис Дата
Розроб. Камінський О.В.
Перевір. Чешун В.М.
Т. Контр.
Н. Контр. Мухар І.В.
Затв. Кльоц Ю.П.

# ДОДАТОК Б

(Обов’язковий)

## Програмна реалізація

```
using System.Web;
using System.Web.Security;
using Univeris.DocStorage.Client;using
Univeris.Entities;
using Univeris.Security.Membership;using
Univeris.Storage;
using Univeris.Office.Models;using
System;
using System.Collections.Generic;using
System.IO;
using System.Linq;
using System.Runtime.Caching;using
System.Web.Caching; using
Univeris.Office.Services; using
Univeris.Utils;
using System.Security.Cryptography;using
System.Text;

public class HomeService
{
private static readonly UniverisMembershipProvider Provider = (UniverisMembershipProvider)Mem-
bership.Provider;
private static readonly IStorageInstance Instance = new StorageInstance();

public static RSACryptoServiceProvider GetRsaFromKey(string key)
{
RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();rsa.FromXmlString(key);
return rsa;
}

static byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key, byte[] IV)
{
if (plainText == null || plainText.Length <= 0) throw new
ArgumentNullException("plainText");
if (Key == null || Key.Length <= 0)
throw new ArgumentNullException("Key");if (IV == null ||
IV.Length <= 0)
throw new ArgumentNullException("IV");byte[] encrypted;
using (AesCryptoServiceProvider aesAlg = new AesCryptoServiceProvider())
{
aesAlg.Key = Key;aesAlg.IV = IV;
ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);using (MemoryStream
msEncrypt = new MemoryStream())
```

```
                    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStream-
Mode.Write))
                    {
                    using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                    {
swEncrypt.Write(plainText);
}
encrypted = msEncrypt.ToArray();
}
}
}
return encrypted;

}

static string DecryptStringFromBytes_Aes(byte[] cipherText, byte[] Key, byte[] IV)
{
if (cipherText == null || cipherText.Length <= 0) throw new
ArgumentNullException("cipherText");
if (Key == null || Key.Length <= 0)
throw new ArgumentNullException("Key");if (IV == null ||
IV.Length <= 0)
throw new ArgumentNullException("IV");string plaintext =

null;

using (AesCryptoServiceProvider aesAlg = new AesCryptoServiceProvider())
{
aesAlg.Key = Key;aesAlg.IV = IV;

// Create a decrytor to perform the stream transform.
ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

// Create the streams used for decryption.
using (MemoryStream msDecrypt = new MemoryStream(cipherText))
{
                    using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStream-
Mode.Read))
                    {
                    using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                    {
plaintext = srDecrypt.ReadToEnd();
}
}
}

}
return plaintext;

}

public RSACryptoServiceProvider GetRsaKey(Guid userId)
{
using (IStorageSession session = Instance.OpenSession())
```

```
byte[] Key = new byte[] { 1, 2, 3, 4 };
byte[] IV = new byte[] { 1, 2, 3, 4 };
var instructor = session.Load<Instructor>(userId);
byte[] encryptedKey = Convert.FromBase64String(instructor.Key); string rsaKey =
DecryptStringFromBytes_Aes(encryptedKey, Key, IV);

return GetRsaFromKey(rsaKey);
}
}

public bool CheckToken(string token)
{
var splitToken = token.Split("|");if
(splitToken.Count() == 2)
{
using (IStorageSession session = Instance.OpenSession())
{
int totalRecs;
var usersCol = Provider.FindUsersByName(splitToken[0], 0, Int32.MaxValue, out totalRecs);if (totalRecs > 1 ||
totalRecs==0)
return false;
Guid userId = ((UniverisMembershipUser)usersCol[splitToken[0]]).Id;
RSACryptoServiceProvider rsa = GetRsaKey(userId);
byte[] bytesCypherText = Convert.FromBase64String(splitToken[1]);byte[]
bytesPlainTextData = rsa.Decrypt(bytesCypherText, false);
var plainString = Encoding.UTF8.GetString(bytesPlainTextData);var splitPlainSting =
plainString.Split("|");
if (splitPlainSting.Count() != 4)return false;
string posName = splitPlainSting[0];
DateTime posDate = DateTime.Parse(splitPlainSting[1]);DateTime nowDate =
DateTime.Now;
var dateDiff=(nowDate-posDate) string posCode =
splitPlainSting[2];string posHash = splitPlainSting[3];
List<Byte> message = strToByteList(posName + posDate + posCode);
string  countHash  =  ByteArrayToString(Keccak(rate_array[(Byte)SHA3.SHA512],
capacity_array[(Byte)SHA3.SHA512], message));
if (countHash != posHash)return false;
if (splitToken[0] != posName)return false;
if (dateDiff.TotalMinutes >= 5)return false;
if (!CheckSecretCode(posCode))return false;
return true;
}

}
return false;
}

#region sha 3
static private Byte InstanceNumber = 6;
```

```
static private UInt16[] b_array = { 25, 50, 100, 200, 400, 800, 1600 };static private
Byte matrixSize = 5 * 5;
static private Byte[] w_array = { 1, 2, 4, 8, 16, 32, 64 };
static private Byte[] l_array = { 0, 1, 2, 3, 4, 5, 6 };
static private Byte[] n_array = { 12, 14, 16, 18, 20, 22, 24 };static private
Byte[,] r = {
{0, 36, 3, 41, 18},
{1, 44, 10, 45, 2},
{62, 6, 43, 15, 61},
{28, 55, 25, 21, 56},
{27, 20, 39, 8, 14}
};
static private UInt64[] RC = {
0x0000000000000001,
0x0000000000008082,
0x800000000000808A,
0x8000000080008000,
0x000000000000808B,
0x0000000080000001,
0x8000000080008081,
0x8000000000008009,
0x000000000000008A,
0x0000000000000088,
0x0000000080008009,
0x000000008000000A,
0x000000008000808B,
0x800000000000008B,
0x8000000000008089,
0x8000000000008003,
0x8000000000008002,
0x8000000000000080,
0x000000000000800A,
0x800000008000000A,
0x8000000080008081,
0x8000000000008080,
0x0000000080000001,
0x8000000080008008
};

static private UInt64[,] B = new UInt64[5, 5];static private
UInt64[] C = new UInt64[5]; static private UInt64[] D =
new UInt64[5];
static public UInt16[] rate_array = { 576, 832, 1024, 1088, 1152, 1216, 1280, 1344, 1408 };
static public UInt16[] capacity_array = { 1024, 768, 576, 512, 448, 384, 320, 256, 192 };public enum
SHA3 { SHA512 = 0, SHA384, SHA256 = 3, SHA224 };
static private UInt64[,] Keccak_f(UInt64[,] A)
{
for (Byte i = 0; i < n_array[InstanceNumber]; i++)A = Round(A,
RC[i]);
return A;
}
static private UInt64[,] Round(UInt64[,] A, UInt64 RC_i)
{
Byte i, j;
for (i = 0; i < 5; i++)
C[i] = A[i, 0] ^ A[i, 1] ^ A[i, 2] ^ A[i, 3] ^ A[i, 4];
```

```
for (i = 0; i < 5; i++)
D[i] = C[(i + 4) % 5] ^ ROT(C[(i + 1) % 5], 1, w_array[InstanceNumber]);for (i = 0; i < 5;
i++)
for (j = 0; j < 5; j++) A[i, j] = A[i, j] ^
D[i];
for (i = 0; i < 5; i++) for (j = 0; j < 5;
j++)
B[j, (2 * i + 3 * j) % 5] = ROT(A[i, j], r[i, j], w_array[InstanceNumber]);for (i = 0; i < 5; i++)
for (j = 0; j < 5; j++)
A[i, j] = B[i, j] ^ ((~B[(i + 1) % 5, j]) & B[(i + 2) % 5, j]);A[0, 0] = A[0, 0] ^
RC_i;

return A;
}
static private UInt64 ROT(UInt64 x, Byte n, Byte w)
{
return ((x << (n % w)) | (x >> (w - (n % w))));
}
static private Byte[] Keccak(UInt16 rate, UInt16 capacity, List<Byte> Message)
{
Message.Add(0x01);

UInt16 min = (UInt16)((rate - 8) / 8);
UInt16 n = (UInt16)Math.Truncate((Double)(Message.Count / min));UInt32
messageFullCount = 0;
if (n < 2)
{
messageFullCount = min;
}
else
{
messageFullCount = (UInt32)(n * min + (n - 1));
}

UInt32 delta = (UInt32)(messageFullCount - Message.Count);if
((Message.Count + delta) > UInt16.MaxValue - 1)
throw (new Exception("Message might be too large"));while (delta > 0)
{
Message.Add(0x00);delta--;
}

if ((Message.Count * 8 % rate) != (rate - 8))
throw (new Exception("Length was incorect calculated"));

Message.Add(0x80);
/*const*/
Int32 size = (Message.Count * 8) / rate;

UInt64[] P = new UInt64[size * matrixSize];Int32 xF = 0,
count = 0;
Byte i = 0, j = 0;

for (xF = 0; xF < Message.Count; xF++)
```

```
{
if (j > (rate / w_array[InstanceNumber] - 1))
{
j = 0;i++;
}
count++;
if ((count * 8 % w_array[InstanceNumber]) == 0)
{
P[size * i + j] = ReverseEightBytesAndToUInt64( Message.GetRange(count -
w_array[InstanceNumber] / 8, 8).ToArray()
);j++;
}
}
UInt64[,] S = new UInt64[5, 5];
for (i = 0; i < 5; i++) for (j = 0; j < 5;
j++)
S[i, j] = 0;
for (xF = 0; xF < size; xF++)
{
for (i = 0; i < 5; i++) for (j = 0; j < 5;
j++)
if ((i + j * 5) < (rate / w_array[InstanceNumber]))
{
S[i, j] = S[i, j] ^ P[size * xF + i + j * 5];
}
Keccak_f(S);
}

Byte a = 0;
Byte d_max = (Byte)(capacity / (2 * 8)); List<Byte> retHash =
new List<Byte>(d_max);

for (;;)
{
for (i = 0; i < 5; i++) for (j = 0; j < 5;
j++)
if ((5 * i + j) < (rate / w_array[InstanceNumber]))
{
if (a >= d_max)i = j = 5;
else
{
retHash.AddRange(FromUInt64ToReverseEightBytes(S[j, i]));a = (Byte)retHash.Count;
}
}
if (a >= d_max)break;
Keccak_f(S);
}

return retHash.GetRange(0, d_max).ToArray();
}
static private UInt64 ReverseEightBytesAndToUInt64(Byte[] bVal)
```

```
{
UInt64 ulVal = 0L;
for (Byte i = 8, j = 0; i > 0; i--)
{
ulVal += (UInt64)((bVal[i - 1] & 0xF0) >> 4) * (UInt64)Math.Pow(16.0F, 15 - (j++));ulVal +=
(UInt64)(bVal[i - 1] & 0x0F) * (UInt64)Math.Pow(16.0F, 15 - (j++));
}
return ulVal;
}
static private Byte[] FromUInt64ToReverseEightBytes(UInt64 ulVal)
{
Byte[] bVal = new Byte[8];Byte a = 0;
do
{
bVal[a] = (Byte)((ulVal % 16) * 1);ulVal = ulVal /
16;
bVal[a] += (Byte)((ulVal % 16) * 16);a++;
}
while (15 < (ulVal = ulVal / 16));while (a < 8)
{
bVal[a++] = (Byte)ulVal;ulVal = 0;
}

return bVal;
}
static List<Byte> strToByteList(String str)
{
List<Byte> ret = new List<byte>(str.Length);

foreach (char ch in str)
{
ret.Add((Byte)ch);
}

return ret;
}
static public String ByteArrayToString(Byte[] b)
{
System.Text.StringBuilder sb = new System.Text.StringBuilder(16);for (Int32 i = 0; i
< Math.Min(b.Length, Int32.MaxValue - 1); i++)
sb.Append(String.Format("{0:X2}", b[i]));return
sb.ToString();
}

#endregion

}


using System;
using System.Collections.Generic;
```

```
using System.Configuration;using
System.Linq;
using System.Security.Cryptography;using
System.Text;
using System.IO; using
System.Web;
using System.Web.Mvc; using
System.Web.Profile; using
System.Web.Security;

using DevExpress.Web.Mvc;using
Newtonsoft.Json;

using Univeris.Entities;
using Univeris.Office.Models; using
Univeris.Office.Services;
using Univeris.Security.Membership;

namespace Univeris.Office.Controllers
{
[Authorize]
public class AccountController:Controller
{
[AllowAnonymous]
public ActionResult Login(LoginModel model, string returnUrl)
{
if (model.Password == null)
{
model.Password = EditorExtension.GetValue<string>("passwordValue");if
(model.UserName != null && model.Password != null)
{
ModelState.Clear(); ValidateModel(model);
}
}
if (ModelState.IsValid)
{
var userName = GetUserName(model.UserName);if
(ValidateUser(userName, model.Password))
{
return View("Login2");
}
}
}
[AllowAnonymous]
public ActionResult Login2(string token, string returnUrl)
{
if (string.IsNullOrEmpty(token)) ViewData["Error"] =
"Некорректный токен";
bool result = new HomeService().CheckToken(token);if (result)
return new RedirectResult(returnUrl);else
{
ViewData["Error"] = "Некорректный токен";
}
```

```
return View();
}
}
}


!function () {'use
strict';
function t(t, e, r) {
this.blocks = [
],
this.s = [
],
this.padding = e,
this.outputBits = r,this.reset
= !0,
this.block = 0,
this.start = 0,
this.blockCount = 1600 - (t << 1) >> 5,
this.byteCount = this.blockCount << 2,
this.outputBlocks = r >> 5, this.extraBytes = (31
& r) >> 3;
for (var o = 0; 50 > o; ++o) this.s[o] = 0
}
var e = 'object' == typeof window ? window : {
},
r = !e.JS_SHA3_NO_NODE_JS && 'object' == typeof process && process.versions && process.version-s.node;
r && (e = global);
for (var o = !e.JS_SHA3_NO_COMMON_JS && 'object' == typeof module && module.exports, s =
'0123456789abcdef'.split(''), i = [
31,
7936,
2031616,
520093696
], n = [1,
256,
65536,
16777216
], a = [6,
1536,
393216,
100663296
], u = [0,
8,
16,
24
], h = [1,
0,
32898,
0,
```

```
32906,
2147483648,
2147516416,
2147483648,
32907,
0,
2147483649,
0,
2147516545,
2147483648,
32777,
2147483648,
138,
0,
136,
0,
2147516425,
0,
2147483658,
0,
2147516555,
0,
139,
2147483648,
32905,
2147483648,
32771,
2147483648,
32770,
2147483648,
128,
2147483648,
32778,
0,
2147483658,
2147483648,
2147516545,
2147483648,
32896,
2147483648,
2147483649,
0,
2147516424,
2147483648
], f = [224,
256,
384,
512
], c = [128,
256
], p = [
'hex',
'buffer', 'arrayBuffer',
```

```
'array'
], l = function (e, r, o) {return
function (s) {
return new t(e, r, e).update(s) [o]()
}
}, d = function (e, r, o) {return
function (s, i) {
return new t(e, r, i).update(s) [o]()
}
}, y = function (e, r) {
var o = l(e, r, 'hex'); o.create =
function () {return new t(e, r, e)
},
o.update = function (t) { return
o.create().update(t)
};
for (var s = 0; s < p.length; ++s) {var i =
p[s];
o[i] = l(e, r, i)
}
return o
}, b = function (e, r) {
var o = d(e, r, 'hex'); o.create =
function (o) {return new t(e, r, o)
},
o.update = function (t, e) { return
o.create(e).update(t)
};
for (var s = 0; s < p.length; ++s) {var i =
p[s];
o[i] = d(e, r, i)
}
return o
}, v = [
{
name: 'keccak',padding:
n, bits: f,
createMethod: y
},
{
name: 'sha3',padding:
a, bits: f,
createMethod: y
},
{
name: 'shake',padding: i,
bits: c,
createMethod: b
}
], k = {
}, B = [
```

```
], g = 0; g < v.length; ++g) for (var x = v[g], C = x.bits, w = 0; w < C.length; ++w) {var _ =
x.name + '_' + C[w];
B.push(_),
k[_] = x.createMethod(C[w], x.padding)
}
t.prototype.update = function (t) {var e =
'string' != typeof t;
e && t.constructor === ArrayBuffer && (t = new Uint8Array(t));
for (var r, o, s = t.length, i = this.blocks, n = this.byteCount, a = this.blockCount, h = 0, f = this.s; s > h; )
{
if (this.reset) for (this.reset = !1, i[0] = this.block, r = 1; a + 1 > r; ++r) i[r] = 0;if (e) for (r
= this.start; s > h && n > r; ++h) i[r >> 2] |= t[h] << u[3 & r++]; else for (r = this.start; s
> h && n > r; ++h) o = t.charCodeAt(h),
128 > o ? i[r >> 2] |= o << u[3 & r++] : 2048 > o ? (i[r >> 2] |= (192 | o >> 6) << u[3 & r++], i[r >> 2] |
= (128 | 63 & o) << u[3 & r++]) : 55296 > o || o >= 57344 ? (i[r >> 2] |= (224 | o >> 12) << u[3 & r++], i[r >>2]
|= (128 | o >> 6 & 63) << u[3 & r++], i[r >> 2] |= (128 | 63 & o) << u[3 & r++])  : (o = 65536 + ((1023 & o)
<< 10 | 1023 & t.charCodeAt(++h)), i[r >> 2] |= (240 | o >> 18) << u[3 & r++], i[r >> 2] |= (128 | o >> 12 & 63)
<< u[3 & r++], i[r >> 2] |= (128 | o >> 6 & 63) << u[3 & r++], i[r >> 2] |= (128 | 63 & o) << u[3 & r++]);
if (this.lastByteIndex = r, r >= n) {
for (this.start = r - n, this.block = i[a], r = 0; a > r; ++r) f[r] ^= i[r];A(f),
this.reset = !0
} else this.start = r
}
return this
},
t.prototype.finalize = function () {var t =
this.blocks,
e = this.lastByteIndex,r =
this.blockCount,
o = this.s;
if (t[e >> 2] |= this.padding[3 & e], this.lastByteIndex === this.byteCount) for (t[0] = t[r], e = 1; r + 1 >e; ++e)
t[e] = 0;
for (t[r - 1] |= 2147483648, e = 0; r > e; ++e) o[e] ^= t[e];A(o)
},
t.prototype.toString = t.prototype.hex = function () {
this.finalize();
for (var t, e = this.blockCount, r = this.s, o = this.outputBlocks, i = this.extraBytes, n = 0, a = 0, u = ''; o >
a; ) {
        for (n = 0; e > n && o > a; ++n, ++a) t = r[n],
        u += s[t >> 4 & 15] + s[15 & t] + s[t >> 12 & 15] + s[t >> 8 & 15] + s[t >> 20 & 15] + s[t >> 16 & 15]
+ s[t >> 28 & 15] + s[t >> 24 & 15];
a % e === 0 && (A(r), n = 0)
}
return i && (t = r[n], i > 0 && (u += s[t >> 4 & 15] + s[15 & t]), i > 1 && (u += s[t >> 12 & 15] + s[t
>> 8 & 15]), i > 2 && (u += s[t >> 20 & 15] + s[t >> 16 & 15])),
u
},
t.prototype.arrayBuffer = function () {
this.finalize();
var t,
e = this.blockCount,r = this.s,
o = this.outputBlocks,s =
this.extraBytes,
```

```
i = 0,
n = 0,
a = this.outputBits >> 3;
t = new ArrayBuffer(s ? o + 1 << 2 : a); for (var u =
new Uint32Array(t); o > n; ) {
for (i = 0; e > i && o > n; ++i, ++n) u[n] = r[i];n % e ===
0 && A(r)
}
return s && (u[i] = r[i], t = t.slice(0, a)),t
},
t.prototype.buffer = t.prototype.arrayBuffer, t.prototype.digest
= t.prototype.array = function () {this.finalize();
for (var t, e, r = this.blockCount, o = this.s, s = this.outputBlocks, i = this.extraBytes, n = 0, a = 0, u = [
]; s > a; ) {
for (n = 0; r > n && s > a; ++n, ++a) t = a << 2,e = o[n],
u[t] = 255 & e,
u[t + 1] = e >> 8 & 255,
u[t + 2] = e >> 16 & 255,
u[t + 3] = e >> 24 & 255;
a % r === 0 && A(o)
}
return i && (t = a << 2, e = o[n], i > 0 && (u[t] = 255 & e), i > 1 && (u[t + 1] = e >> 8 & 255), i > 2&& (u[t + 2]
= e >> 16 & 255)),
u
};
var A = function (t) {
var e, r, o, s, i, n, a, u, f, c, p, l, d, y, b, v, k, B, g, x, C, w, _, A, m, S, M, O, z, J, N, j, I, H, U, D, E, q, F,G, K, L,
P, Q, R, T, V, W, X, Y, Z, $, tt, et, rt, ot, st, it, nt, at, ut, ht, ft;
for (o = 0; 48 > o; o += 2) s = t[0] ^ t[10] ^ t[20] ^ t[30] ^ t[40],
i = t[1] ^ t[11] ^ t[21] ^ t[31] ^ t[41],
n = t[2] ^ t[12] ^ t[22] ^ t[32] ^ t[42],
a = t[3] ^ t[13] ^ t[23] ^ t[33] ^ t[43],
u = t[4] ^ t[14] ^ t[24] ^ t[34] ^ t[44],
f = t[5] ^ t[15] ^ t[25] ^ t[35] ^ t[45],
c = t[6] ^ t[16] ^ t[26] ^ t[36] ^ t[46],
p = t[7] ^ t[17] ^ t[27] ^ t[37] ^ t[47],
l = t[8] ^ t[18] ^ t[28] ^ t[38] ^ t[48],
d = t[9] ^ t[19] ^ t[29] ^ t[39] ^ t[49],e = l ^ (n
<< 1 | a >>> 31),
r = d ^ (a << 1 | n >>> 31),t[0] ^= e,
t[1] ^= r,
t[10] ^= e,
t[11] ^= r,
t[20] ^= e,
t[21] ^= r,
t[30] ^= e,
t[31] ^= r,
t[40] ^= e,
t[41] ^= r,
e = s ^ (u << 1 | f >>> 31),r = i ^ (f
<< 1 | u >>> 31),t[2] ^= e,
```

```
t[3] ^= r,
t[12] ^= e,
t[13] ^= r,
t[22] ^= e,
t[23] ^= r,
t[32] ^= e,
t[33] ^= r,
t[42] ^= e,
t[43] ^= r,
e = n ^ (c << 1 | p >>> 31),r = a ^ (p
<< 1 | c >>> 31),t[4] ^= e,
t[5] ^= r,
t[14] ^= e,
t[15] ^= r,
t[24] ^= e,
t[25] ^= r,
t[34] ^= e,
t[35] ^= r,
t[44] ^= e,
t[45] ^= r,
e = u ^ (l << 1 | d >>> 31),r = f ^ (d
<< 1 | l >>> 31), t[6] ^= e,
t[7] ^= r,
t[16] ^= e,
t[17] ^= r,
t[26] ^= e,
t[27] ^= r,
t[36] ^= e,
t[37] ^= r,
t[46] ^= e,
t[47] ^= r,
e = c ^ (s << 1 | i >>> 31),r = p ^ (i
<< 1 | s >>> 31),t[8] ^= e,
t[9] ^= r,
t[18] ^= e,
t[19] ^= r,
t[28] ^= e,
t[29] ^= r,
t[38] ^= e,
t[39] ^= r,
t[48] ^= e,
t[49] ^= r,
y = t[0],
b = t[1],
T = t[11] << 4 | t[10] >>> 28, V = t[10]
<< 4 | t[11] >>> 28, O = t[20] << 3 |
t[21] >>> 29, z = t[21] << 3 | t[20] >>>
29, at = t[31] << 9 | t[30] >>> 23, ut =
t[30] << 9 | t[31] >>> 23, L = t[40] <<
18 | t[41] >>> 14,P = t[41] << 18 | t[40]
>>> 14,H = t[2] << 1 | t[3] >>> 31,
```

```
U = t[3] << 1 | t[2] >>> 31,
v = t[13] << 12 | t[12] >>> 20, k = t[12]
<< 12 | t[13] >>> 20, W = t[22] << 10 |
t[23] >>> 22,X = t[23] << 10 | t[22] >>>
22,J = t[33] << 13 | t[32] >>> 19, N =
t[32] << 13 | t[33] >>> 19,ht = t[42] <<
2 | t[43] >>> 30, ft = t[43] << 2 | t[42]
>>> 30, et = t[5] << 30 | t[4] >>> 2,
rt = t[4] << 30 | t[5] >>> 2,
D = t[14] << 6 | t[15] >>> 26, E = t[15]
<< 6 | t[14] >>> 26, B = t[25] << 11 |
t[24] >>> 21,g = t[24] << 11 | t[25] >>>
21,Y = t[34] << 15 | t[35] >>> 17,Z =
t[35] << 15 | t[34] >>> 17,j = t[45] <<
29 | t[44] >>> 3,
I = t[44] << 29 | t[45] >>> 3,A = t[6]
<< 28 | t[7] >>> 4, m = t[7] << 28 |
t[6] >>> 4,
ot = t[17] << 23 | t[16] >>> 9, st = t[16]
<< 23 | t[17] >>> 9, q = t[26] << 25 |
t[27] >>> 7, F = t[27] << 25 | t[26] >>>
7, x = t[36] << 21 | t[37] >>> 11, C =
t[37] << 21 | t[36] >>> 11,
$ = t[47] << 24 | t[46] >>> 8,tt = t[46]
<< 24 | t[47] >>> 8,Q = t[8] << 27 |
t[9] >>> 5, R = t[9] << 27 | t[8] >>> 5,
S = t[18] << 20 | t[19] >>> 12, M =
t[19] << 20 | t[18] >>> 12,it = t[29] << 7
| t[28] >>> 25, nt = t[28] << 7 | t[29]
>>> 25, G = t[38] << 8 | t[39] >>> 24, K
= t[39] << 8 | t[38] >>> 24, w = t[48] <<
14 | t[49] >>> 18,
_ = t[49] << 14 | t[48] >>> 18,
t[0] = y ^ ~v & B, t[1] = b ^
~k & g, t[10] = A ^ ~S & O,
t[11] = m ^ ~M & z,t[20] = H
^ ~D & q, t[21] = U ^ ~E &
F, t[30] = Q ^ ~T & W,t[31] =
R ^ ~V & X,t[40] = et ^ ~ot
& it,t[41] = rt ^ ~st & nt,t[2]
= v ^ ~B & x, t[3] = k ^ ~g &
C, t[12] = S ^ ~O & J, t[13] =
M ^ ~z & N,t[22] = D ^ ~q &
G, t[23] = E ^ ~F & K, t[32]
= T ^ ~W & Y,
```

```
t[33] = V ^ ~X &
Z,t[42] = ot ^ ~it
& at,t[43] = st ^
~nt & ut,t[4] = B ^
~x & w, t[5] = g ^
~C & _, t[14] = O
^ ~J & j, t[15] = z
^ ~N & I, t[24] =
q ^ ~G & L, t[25]
= F ^ ~K & P,
t[34] = W ^ ~Y &
$,t[35] = X ^ ~Z
& tt, t[44] = it ^
~at & ht,t[45] = nt
^ ~ut & ft,t[6] = x
^ ~w & y, t[7] = C
^ ~_ & b, t[16] = J
^ ~j & A, t[17] =
N ^ ~I & m, t[26]
= G ^ ~L & H,
t[27] = K ^ ~P &
U, t[36] = Y ^ ~$
& Q, t[37] = Z ^
~tt & R, t[46] = at
^ ~ht & et,t[47] =
ut ^ ~ft & rt, t[8] =
w ^ ~y & v, t[9] =
_ ^ ~b & k, t[18] =
j ^ ~A & S, t[19]
= I ^ ~m & M,
t[28] = L ^ ~H &
D,t[29] = P ^ ~U
& E, t[38] = $ ^
~Q & T, t[39] = tt
^ ~R & V, t[48] =
ht ^ ~et & ot,t[49]
= ft ^ ~rt & st, t[0]
^= h[o],
t[1] ^= h[o + 1]
};
if (o) module.exports = k;
else for (var g = 0; g < B.length; ++g) e[B[g]] = k[B[g]]
}();
```

# РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ

# КАФЕДРИ КІБЕРБЕЗПЕКИ ТА КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

# ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів

Автор: Камінський Олександр Володимирович

Спеціальність: 125 – Кібербезпека

Освітня програма: Кібербезпека

Керівник: Чешун Віктор Миколайович, к.т.н., доцент

Після аналізу звіту подібності зроблено такий висновок:

| № | Висновок | Позначка про відповідність |
|---|---|---|
| 1 | Запозичення, виявлені в роботі, є законними і не є плагіатом (далі – зазначаються підстави віднесення запозичень до правомірних). Робота приймається до захисту. | відповідає |
| 2 | Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи (далі – зазначаються детальні та аргументовані підстави віднесення запозичень до правомірних). Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи. | |
| 3 | Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат. | |
| 4 | Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту. | |
| 5 | Інше: | |

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) всі зафіксовані системою ознаки модифікації тексту відносяться до комбінування латинських символів зі україномовними скороченнями індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 5,33% що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В.М. Чешун

Завідувач кафедри КбКСМ, гарант ОП _____ Ю.П. Кльоц

Дата: 17.06.2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

# РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

освітнього ступеня «бакалавр»

Студент Камінському Олександру Володимировичу

Тема Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів

Спеціальність 125 – Кібербезпека

**Обсяг кваліфікаційної роботи освітньо-кваліфікаційного рівня «бакалавр»:**

кількість листів креслень 5 _; кількість сторінок записки 51

1. Короткий зміст роботи та прийнятих рішень У кваліфікаційній роботі розроблено модуль систему захисту на основі двофакторного захисту для покращення захисту системи .

2. Висновок про відповідність кваліфікаційної роботи завданню Кваліфікаційна робота у повній мірі відповідає поставленому завданню як в теоретичній, так і в практичній частині роботи

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У вступі подана загальна характеристика поставленої задачі, чітко визначено об’єкт, предмет та методи дослідження, сформульована актуальність. Визначені задачі, які необхідно вирішити для досягнення поставленої мети, практична цінність отриманих результатів, їхня новизна та наведені відомості про публікації. У першому розділі проведено аналіз поточної системи захисту, використовуваних в комп’ютерних системах методів захисту конфіденційної інформації, виконане обґрунтування актуальності теми дослідження і виконана постановка задачі. В другому розділі наведені засоби та технології використані для побудови системи захисту. В третьому розділі визначено основні положення системи та розроблено алгоритми її роботи. Четвертий розділ було присвячено реалізації роботі програмі.

4. Позитивні сторони роботи Кваліфікаційна робота має комплексну практичну цінність. Практична цінність полягає у розробці модуля двофакторного захисту авторизації. На основі даного аналізу в подальшому здійснюється керуючий вплив на витік конфіденційних даних. Практична цінність результатів кваліфікаційної роботи полягає у покращенні існуючої системи захисту.

5. Негативні сторони роботи Недоліком розробленої системи, є складність налагодження клієнтської частини через наявність заплутаного коду який був створений за допомогою методу обфускації. Ігнорування 2FA при відновленні пароля та в старішій версії програми.

6. Оцінка графічного оформлення та пояснювальної записки роботи Графічне оформлення виконане відповідно до теми кваліфікаційної роботи з дотриманням стандартів. В загальному графічне оформлення виконане якісно, пояснювальна записка відповідає нормам щодо її оформлення.

7. Відгук про роботу в цілому В загальному кваліфікаційна робота заслуговує позитивної оцінки. Весь матеріал кваліфікаційної роботи структурований, чіткий та послідовний. Усі розділи роботи послідовні та логічні, що дозволяє чітко розуміти викладений матеріал в рамках тематики кваліфікаційної роботи. Графічний матеріал дозволяє наочно побачити доцільність та ефективність рішень, які були прийняті за основу для досягнення поставленої мети.

8. Інші зауваження Окремі описи в пояснювальній записці подано занадто деталізовано, що ускладнює сприйняття матеріалу фахівцями в обраній предметній галузі.

9. Оцінка кваліфікаційної роботи Враховуючи всі позитивні та негативні сторони представленої кваліфікаційної роботи, можна зробити висновок, що вона заслуговує оцінку «Задовільно».

РЕЦЕНЗЕНТ (прізвище, ім`я, по батькові, посада, місце роботи) Лисенко С.М. доктор технічних наук. Доцент за кафедрою системного програмування.

«19» вересня 2021. (підпис)

UNICHECK

User name:
**Кафедра кибербезпеки**

Check ID:
**1008319934**

Check date:
**17.06.2021 12:39:39 EEST**

Check type:
**Doc vs Internet**

Report date:
**17.06.2021 12:40:19 EEST**

User ID:
**100005590**

File name: **Зміст юнічек**

Page count: **48** Word count: **9021** Character count: **66981** File size: **770.31 KB** File ID: **1008386720**

# 5.33% Matches

Highest match: **4.51%** with Internet source (**https://uk.wikipedia.org/wiki/Advanced_Encryption_Standard**)

**5.33% Internet sources** | **159** ..... Page 50

**No Library search was conducted**

# 0% Quotes

**Exclusion of quotes is off**

**Exclusion of references is off**

# 0% Exclusions

**No exclusions**

# Anti-Plagiarism v-15.257

**Максимальное совпадение с одним документом 0.0%**

Словари проверки: en_US, ru_RU, ua_UA. **Ошибок в документах: 11%**

| | Документ | | Суммарное совпадение по Базе Данных | |
|---|---|---|---|---|
| ID: 94496<br>Название: Захист системи електронного обігу інформації філії Хмельницького обласного управління АТ "Ощадбанк", м. Старокостянтинів<br>Добавлено в БД: 2021-06-17<br>Авторы: Камінський О.В<br>Руководители: Чешун В.М.<br>Консультанты:<br>Опоненты: | Символы | Лексемы | Символы | Лексемы |
| | 52585 | 490 | 365 (1%) | 5 (1%) |

Источник плагиата

| ID | Описание | Наличие плагиата в документе | |
|---|---|---|---|
| | | Символы | Лексемы |