Київський національний університет імені Тараса Шевченка
Факультету радіофізики, електроніки та комп’ютерних систем
Кафедра комп’ютерної інженерії

КРОС-ПЛАТФОРМНИЙ ЗАСТОСУНОК ДЛЯ ЕЛЕКТРОННОГО ЦИФРОВОГО ПІДПИСУ ДОКУМЕНТІВ

Дипломна робота магістра
студента 2-го року навчання
спеціальність: 123 «Комп'ютерна інженерія»
Богдана ПОЛІЩУКА

Науковий керівник
канд. техн. наук Євген СЛЮСАР,
асистент кафедри комп’ютерної інженерії

До захисту допускаю: Завідувач кафедрою Юрій БОЙКО

Ухвалено на засіданні кафедри “____” _____ 2023 р., протокол № ___

Київ – 2023

РЕФЕРАТ

Обсяг роботи за об’ємом складає 49 сторінок, містить 22 рисунків, 1 таблицю, 6 додатків, використано 14 інформаційних джерел.

Актуальність роботи полягає в тому, щоб створити крос-платформний застосунок для реалізації роботи електронного підпису за допомогою національного стандарту України ДСТУ 4145.

Головною метою роботи є продемонструвати роботу створеного застосунку для електронного підпису даних за допомогою національного стандарту України ДСТУ 4145 із функціями створення приватного та публічних ключів, підпису даних та перевірка на цілісність цих даних.

Ключові слова : хешування, електронний підпис, sha256.

Зміст

ПОСТАНОВКА ЗАДАЧІ 5
ВСТУП 6
1.Теоретичні відомості 8
1.1 Основні терміни 8
1.2 Електронний цифровий підпис 8
1.3 ДСТУ 4145-2002 11
1.4 Алгоритми хешування 13
1.4.1 SHA-256 13
1.4.2 ГОСТ 34.311 14
1.5 Криптостійкість 14
2.Практична частина 17
2.1 Вибір програмного забезпечення 17
2.1.1 Бек-енд частина 17
2.1.2 Фронт-енд частина 18
2.2 Створення застосунку 19
2.2.1 Створення публічного і приватного ключів 19
2.2.2 Підпис даних 22
2.2.3 Верифікація цифрового підпису 25
2.3 Крос-платформність 27
2.3.1 Запуск на платформі Android 28
2.3.2 Запуск на платформі iOS 30
ВИСНОВКИ 31
ПЕРЕЛІК ПОСИЛАНЬ 32

Додаток 1 – клас CreateX509 34
Додаток 2 – клас SignData 37
Додаток 3 – клас Verify 39
Додаток 4 – Сторінка Sign 43
Додаток 5 – Сторінка Create a key 45
Додаток 6 – Сторінка Verify 47

ПОСТАНОВКА ЗАДАЧІ

Розглянути принцип роботи алгоритму на еліптичних кривих та принцип стандарту ДСТУ 4145, провести порівняльну характеристику з схожим алгоритмом RSA.

Створити крос-платформний застосунок на платформі .NET MAUI для підпису електронним цифровим підписом із функціями створення приватного та публічних ключів, підпису даних та перевірка на цілісність цих даних. Перевірити коректність роботи застосунку з різними вхідними даним, зміненими даними та на платформах Windows, Android та iOS.

ВСТУП

В сучасному цифровому світі, де збільшується кількість електронних документів та інформації, необхідність у засобах забезпечення їх безпеки та цілісності зростає. Електронний підпис даних є ефективним засобом для забезпечення конфіденційності, цілісності та достовірності електронних документів та інформації. Застосування електронного підпису даних сприяє економії часу та коштів на оформлення документів, зменшенню ризику помилок та уникненню можливих спроб фальсифікації даних.

Забезпечення надійного функціонування комп'ютеризованих систем оброблення інформації вимагає відповідної перевірки цілісності та автентичності даних, які вони обробляють. Автентифікація є процесом, який дозволяє перевірити, чи має об'єкт чи суб'єкт очікувані властивості, що є важливим для забезпечення достовірності повідомлень. Перевірка цілісності повідомлення, тобто переконання про те, що повідомлення надійшло без порушення свого вмісту з очікуваного джерела, є одним з аспектів автентифікації. Цей процес здійснюється шляхом аналізу структури відповідних даних з використанням заздалегідь визначених алгоритмів.

У цій роботі буде розглядатися принцип роботи електронного підпису документів за алгоритмом на еліптичних кривих за стандартом ДСТУ-4145.

ДСТУ-4145 передбачає використання алгоритмів побудованих на базі еліптичних кривих для забезпечення електронного цифрового підпису (ЕЦП). Зокрема, стандарт встановлює вимоги до використання еліптичних кривих для генерації ключів підпису та перевірки підпису. Він був спеціально розроблений для того, щоб забезпечити перевірку того, що дані передалися без порушення цілісності.

Передбачається, що початковий обмін інформацією між користувачами відбувається в незахищеній середовищі, а передані пакети можуть бути перехоплені і модифіковані.

Алгоритми на еліптичних кривих має високу криптостійкість, навіть якщо зловмисник перехопить повідомлення, йому потрібно буде дуже багато часу, щоб дешифрувати повідомлення.

1.Теоретичні відомості

1.1 Основні терміни

Перш за все варто визначитись з термінологією що буде в подальшому використовуватись в даній роботі. Було виявлено, що певні терміни, що стосуються протоколу в загальних визначеннях не підходять для використання в цій роботі тому їх варто перевизначити щоб уникнути непорозумінь.

ДСТУ-4145 - це державний стандарт України, який визначає вимоги до електронного цифрового підпису (ЕЦП) та його застосування для забезпечення цілісності, автентичності та невідмовності електронних документів.

ECC - асиметричний криптографічний алгоритм, заснований на використанні алгебраїчної структури еліптичних кривих над кінцевими полями.

Електронний цифровий підпис (ЕЦП) – це математичний метод, який використовується для підтвердження автентичності та цілісності повідомлення.

RSA – асиметричний криптографічний алгоритмом.

1.2 Електронний цифровий підпис

Цифровий підпис — це математичний метод, який використовується для підтвердження автентичності та цілісності повідомлення, програмного забезпечення чи цифрового документа. Це цифровий еквівалент власноручного підпису чи печатки, але він забезпечує набагато більшу безпеку. Цифровий підпис призначений для вирішення проблеми фальсифікації та видавання себе за іншу особу в цифрових комунікаціях.

Призначення ЕЦП:

1. Аутентифікація особи, який підписав електронний документ.
2. Контроль цілісності документа: при будь-яких змінах документ підписання стане недійсним.
3. Захист від змін (підробок) документа.
4. Неможливість відмови від авторства (створити підпис може тільки власник закритого ключа, тому він потім не може відмовитися від своєї підписки під документом).
5. Доказове підтвердження авторства документа.

Цифрові підписи можуть надавати докази походження, ідентичності та статусу електронних документів, транзакцій або цифрових повідомлень. Підписувачі також можуть використовувати їх для підтвердження інформованої згоди.

Цифрові підписи базуються на криптографії з відкритим ключем, також відомої як асиметрична криптографія. За допомогою алгоритму відкритого ключа, такого як RSA або ECC, генеруються два ключі, створюючи математично пов’язану пару ключів, один приватний і один публічний.

Цифрові підписи працюють за допомогою двох криптографічних ключів із відкритим ключем, які взаємно перевіряють автентичність . Особа, яка створює цифровий підпис, використовує приватний ключ для шифрування даних, пов’язаних із підписом, тоді як єдиний спосіб розшифрувати ці дані – відкритий ключ підписувача.

Якщо одержувач не може відкрити документ за допомогою відкритого ключа підписанта, це означає, що з документом або підписом виникла проблема. Таким чином відбувається автентифікація цифрових підписів.

Технологія цифрового підпису вимагає від усіх сторін впевненості в тому, що особа, яка створила підпис, зберегла секретний ключ. Якщо хтось інший має доступ до приватного ключа підпису, ця сторона може створити шахрайські цифрові підписи від імені власника приватного ключа.

Схема електронного підпису включає в себе:

- алгоритм генерації ключової пари користувача, який випадково вибирає закритий ключ і вираховується відповідний йому відкритий ключ;
- формування підписів – для електронного документа за допомогою закритого ключа вираховується підпис
- перевірку (верифікацію) підпису – для документа і підпису за допомогою відкритого ключа визначається дійсність підпису.

В залежності від алгоритму функції, що формує підписку, може бути детермінованою або ймовірною. Детерміновані функції завжди знаходять однаковий підпис для одного і того ж документа. Вірогідні функції вносяться в підписку елемент випадковості, що підсилює криптостійкість алгоритмів ЕЦП. Але для ймовірних схем необхідний або апаратний генератор шуму, або криптографічний надійний генератор псевдовипадкових бітів, що ускладнює їх використання. Різняться одноразові схеми ЕЦП, в яких після перевірки підписи треба провести заміну ключів, і багаторазові схеми, які цього не вимагають. [4]

На рисунку 1.2 зображено процес підписання і верифікації даних за допомогою електронного цифрового підпису.

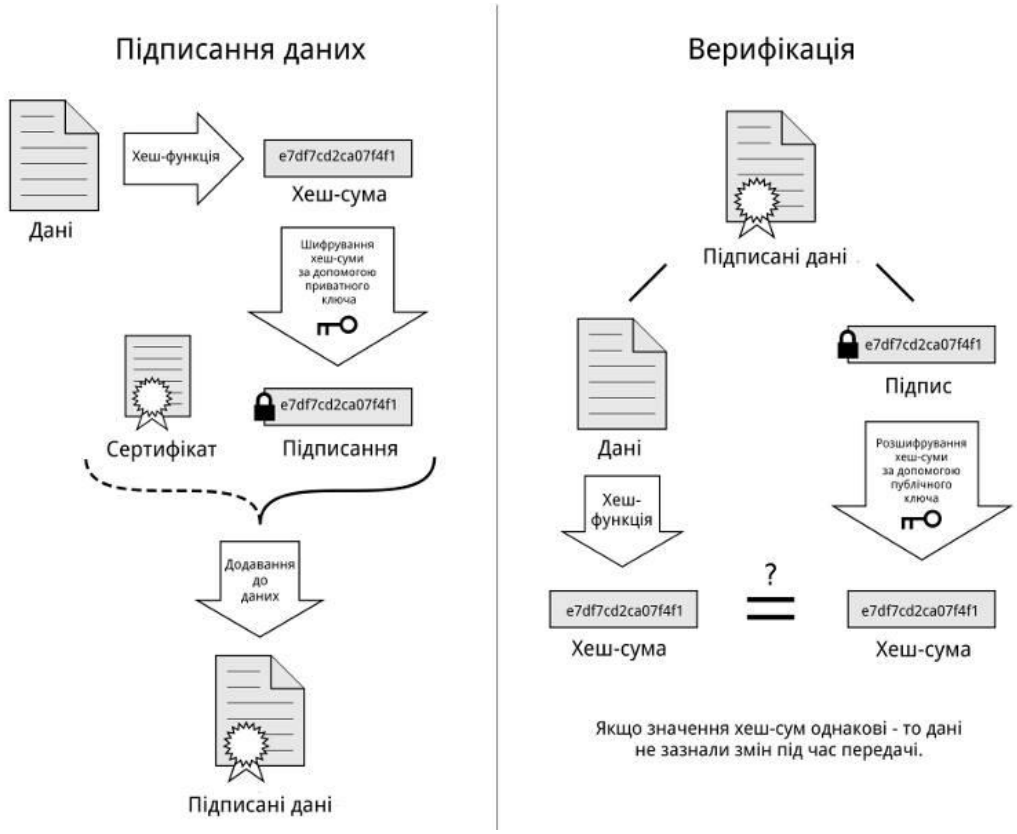


Рис. 1.2 - Ілюстрація цифрового підпису даних

1.3 ДСТУ 4145-2002

Основними процедурами алгоритму цифрового підпису, встановленими ДСТУ 4145-2002 є обчислення передпідпису, обчислення підпису та перевірка цифрового підпису. Стандарт описує алгоритми формування та перевірки електронного цифрового підпису, засновані на властивостях груп точок еліптичних кривих над полями $GF(2^m)$, де m - велике просте число, і правилах застосування цих правил до повідомлень, які пересилаються каналами зв'язки та/або обробляються у комп'ютеризованих системах загального призначення.

На рисунку 1.3 зображено блок-схема, яка ілюструє формування електронного цифрового підпису за алгоритмом ДСТУ 4145-2002.

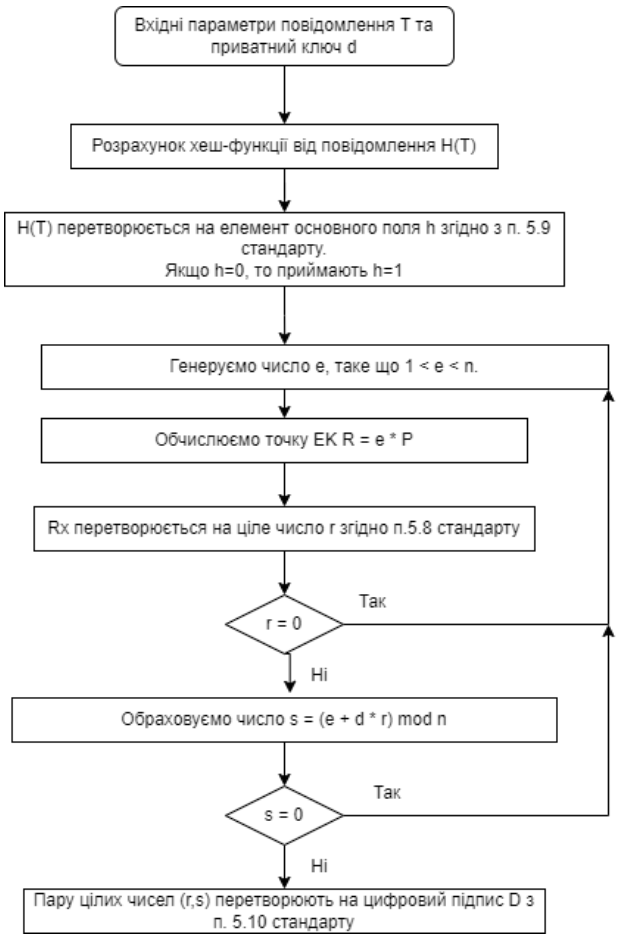


Рис. 1.3 Процес формування підпису

На рисунку 1.3.1 зображено процес верифікації електронного цифрового підпису за алгоритмом ДСТУ 4145-2002.

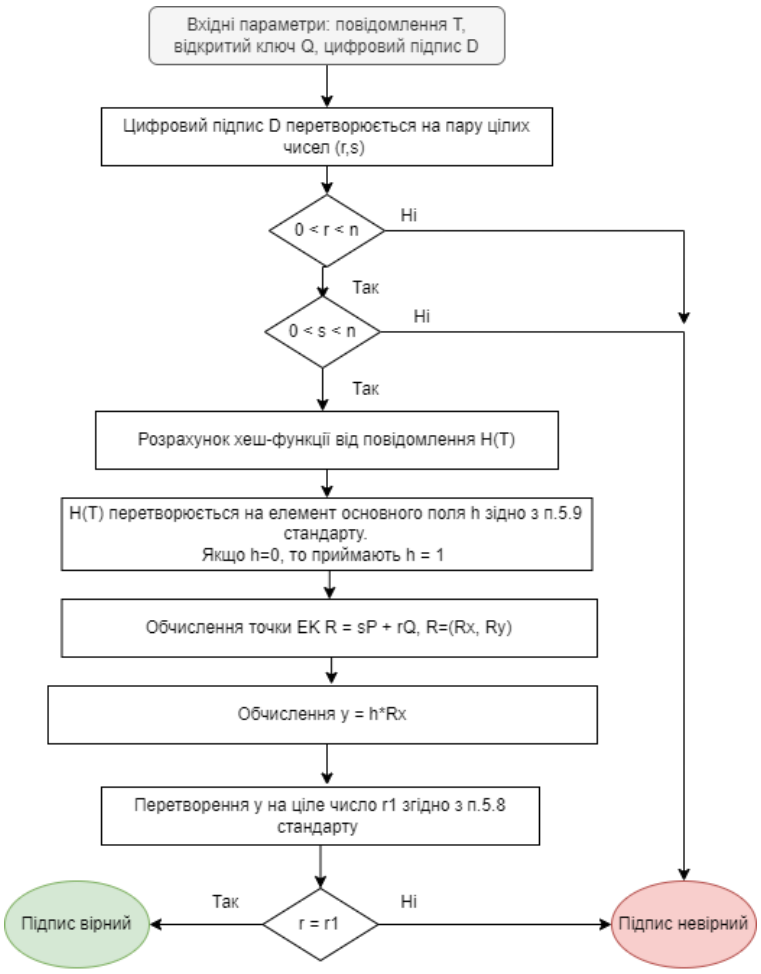


Рис 1.3.1 Процес верифікації підпису

1.4 Алгоритми хешування

Для того щоб алгоритм електронного підпису працював потрібно використовувати хеш-функцію. В даній роботі для підпису було використано SHA256 та ГОСТ 34.311.

1.4.1 SHA-256

SHA-256, що означає безпечний хеш-алгоритм 256 біт, є криптографічним алгоритмом хешування (або функцією), який використовується для перевірки цілісності повідомлень, файлів і даних. Це частина сімейства хеш-функцій SHA-2 і використовує 256-бітний ключ, щоб взяти фрагмент даних і перетворити його на новий, нерозпізнаний рядок

даних фіксованої довжини. Цей рядок випадкових символів і чисел, який називається хеш-значенням, також має розмір 256 біт. [3]

1.4.2 ГОСТ 34.311

ГОСТ 34.311 - це державний стандарт України, який визначає вимоги до захисту інформації, збереженої в електронному вигляді, від несанкціонованого доступу, модифікації та втрати.

Стандарт містить вимоги до захисту інформації, яка зберігається в електронних системах, включаючи вимоги до захисту інформації на різних етапах життєвого циклу даних: від збору та введення інформації до її знищення.

ГОСТ 34.311 встановлює вимоги до криптографічних методів захисту інформації, таких як електронний підпис, шифрування та хеш-функції. Також стандарт визначає вимоги до захисту інформації від несанкціонованого доступу за допомогою механізмів контролю доступу та автентифікації користувачів.

ГОСТ 34.311 є важливим документом для компаній та організацій, які зберігають чутливу інформацію в електронному вигляді, тому що дотримання стандарту допомагає забезпечити безпеку та конфіденційність цих даних. Крім того, дотримання вимог ГОСТ 34.311 є важливим для отримання сертифікатів відповідності та інших документів, які підтверджують відповідність захисту інформації вимогам національних та міжнародних стандартів.

1.5 Криптостійкість

Порівняння алгоритму еліптичних кривих із алгоритмом RSA.

Так як ДСТУ-4145 схожий за своїм алгоритмом на еліптичних кривих, тому було використано саме алгоритм на еліптичних кривих

RSA використовує метод розкладання на прості множники для досягнення одностороннього шифрування повідомлення. Метод розкладання на прості множники передбачає взяття двох великих випадкових простих чисел (навіть «трильйони» є занадто малим числом, щоб їх точно представити) і їх множення для створення відкритого ключа. Однак ви не можете розшифрувати повідомлення, не знаючи цих двох простих чисел. Отримання цих двох простих чисел є надзвичайно складним завданням. Що ж, група дослідників підрахувала, що знадобиться більше 1500 років обчислювального часу, щоб просіяти 768-бітний, 232-розрядний модуль RSA за допомогою «одноядерного процесора AMD Operton 2,2 ГГц з 2 ГБ оперативної пам'яті».

Важливою особливістю RSA є його простота. Він заснований на простих математичних принципах і може працювати швидше порівняно з ECC. Таким чином, RSA ідеально підходить для забезпечення внутрішньої безпеки організації.

ECC — це асиметричний криптографічний алгоритм, заснований на використанні алгебраїчної структури еліптичних кривих над кінцевими полями. Алгоритм ECC працює над проблемою дискретного логарифмування еліптичної кривої (ECDLP). Цей метод криптографії важче зламати, оскільки не існує відомого розв'язку математичної проблеми, заданої рівнянням, що створює еліптичну криву на графіку. Тому для хакерів залишається тільки один шлях: атака грубою силою — іншими словами, метод проб і помилок. Ця складність робить ECC більш безпечним порівняно з RSA.

Оскільки ECC — за структурою — більш безпечний порівняно з RSA, оскільки він пропонує оптимальний захист із меншою довжиною ключа. Як наслідок, це вимагає меншого навантаження на мережу та обчислювальну потужність, що перетворюється на кращий досвід користувача. Щоб надати деякі цифри, RSA може відповісти на 450 запитів на секунду із середнім часом відповіді 150 мілісекунд, тоді як ECC потрібно лише 75 мілісекунд, щоб відповісти на таку саму кількість запитів на секунду.

На таблиці 1.5 зображено порівняння алгоритмів

Безпека у бітах	Необхідна довжина ключа RSA	Необхідна довжина ключа ECC
80	1024	160-223
112	2048	224-255
128	3072	256-383
192	7680	384-511

Таблиця 1 Порівняння RSA та ECC

Як можна побачити із таблиці один алгоритм на еліптичних кривих має надійніший захист і йому потрібно меншу довжина ключа в порівнянні з RSA. RSA вимагає значно більшої довжини ключа порівняно з ECC. Тому, щоб реалізувати 256-бітне шифрування, нам доведеться використовувати ключ RSA довжиною 15360 біт. Це, звичайно, непрактично, оскільки потребує набагато більше обчислювальної потужності.

Обидва методи є поширеними та забезпечують захист від атак типу "людина посередині" (MitM). Однак RSA було визнано вразливим до деяких атак, і це питання «коли», а не «якщо» RSA в кінцевому підсумку вийде з ладу. Багато експертів вважають, що RSA більше не буде використовуватися до 2030 року. ECC, з іншого боку, знаходиться на стадії зрілості, і багато користувачів почали його використовувати. Якщо ви плануєте придбати сертифікат SSL, радимо вибрати сертифікат із опцією ECC, оскільки завжди краще бути на крок попереду зловмисників.

У 1998 році Certicom розпочала змагання з обчислення дискретних логарифмів на еліптичних кривих з бітовою довжиною від 109 до 369. На сьогоднішній день успішно зламані лише криві завдовжки 109 біт. Остання успішна спроба була здійснена у 2004 році.

В застосунку будуть використовуватися еліптичні криві з бітовою довжиною від 163 до 431 бітів.

2.Практична частина

Метою роботи є створення застосунку для цифрового підпису за стандартами ДСТУ використовуючи відкриті безкоштовні компоненти.

2.1 Вибір програмного забезпечення

Для створення застосунку, який буде дозволяти підписувати дані за допомогою електронного підпису було обрано програмне забезпечення .NET MAUI. Основна мова програмування – c# і мова розмітки - XAML.

Для вибору програмного забезпечення основною умовою було це, щоб кінцевий продукт був крос-платформний і тому було обрано платформа MAUI на платформі .NET.

2.1.1 Бек-енд частина

.NET MAUI є крос-платформовою платформою з відкритим вихідним кодом для розробки мобільних та класичних додатків за допомогою C# та XAML. Ця платформа є еволюцією Xamarin.Forms, яка була розширена для підтримки класичних сценаріїв та вдосконалена для підвищення продуктивності та розширюваності. Хоча є деякі подібності між Xamarin.Forms та .NET MAUI, остання має більше можливостей для створення багатоплатформних додатків з одного проекту, а також може використовувати спеціалізований вихідний код для конкретної платформи, якщо це необхідно. Основна мета .NET MAUI - забезпечити можливість реалізувати якомога більше коду та макета інтерфейсу в загальній базі для додатка.

За допомогою .NET MAUI можна розробляти програми, які можуть працювати на таких операційних системах як: Android, iOS, macOS та Windows з однієї загальної бази коду (Рисунок 2.1).

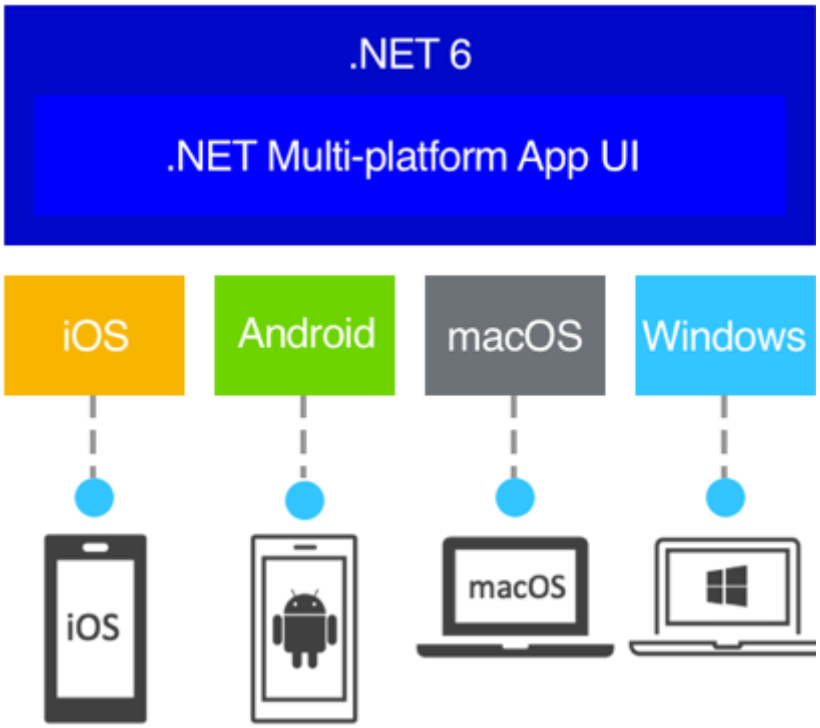
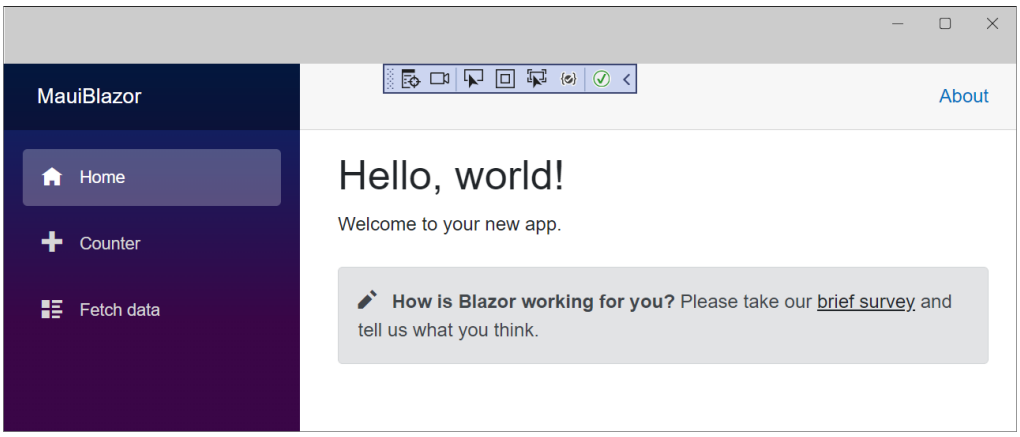


Рисунок 2.1

2.1.2 Фронт-енд частина

Blazor Pages - це фреймворк для створення інтерактивних програм на платформі .NET, який може працювати як на стороні сервера, так і на стороні клієнта. При створенні інтерфейсу користувача використовуються компоненти, що є схожими на ті, які використовуються в Angular, React та VueJS. Фреймворк використовує C# для програмування як на стороні клієнта, так і на стороні сервера, замість JavaScript. У свою чергу, для опису візуального інтерфейсу використовуються стандартні HTML та CSS. Приклад вигляду інтерфейсу Blazor на малюнку 1.



Малюнок 1

2.2 Створення застосунку

Для генерації електронного підпису методом еліптичних кривих було використано готову бібліотеку мовою c# [2]. Але для підпису даних і верифікації бібліотека мала такі недоліки:

1) Так як бібліотека знаходиться у альфа-версії, мною було протестовано і досліджено що бібліотека працювала не для всіх рекомендованих кривих, які описані у ДСТУ 4145-2002[1]. Виправивши помилки, які були знайдені - у моєму додатку буде змога використовувати різні рекомендовані еліптичні криві.
2) Бібліотека не мала хеш-функції, а це обов'язковий критерій для електронного підпису даних.

2.2.1 Створення публічного і приватного ключів

Кнопка "Створити і зберегти ключі" – генерує 2 ключі:

- Приватний ключ – це ключ який показує кількість разів, скільки була додана точка. Приватний ключ залишається у його власника і має бути надійно прихованим.
- Публічний ключ – ключ, який показує точку, яка додавалася, також у ньому міститься власне крива. Цей ключ використовується для передачі третім особам, які хочуть

засвідчитися чи документ правильно підписаний. Точки виглядають у двійковому форматі, так як числа дуже великі і комп'ютеру "легше" робити операції, коли дані знаходяться у двійковому форматі.

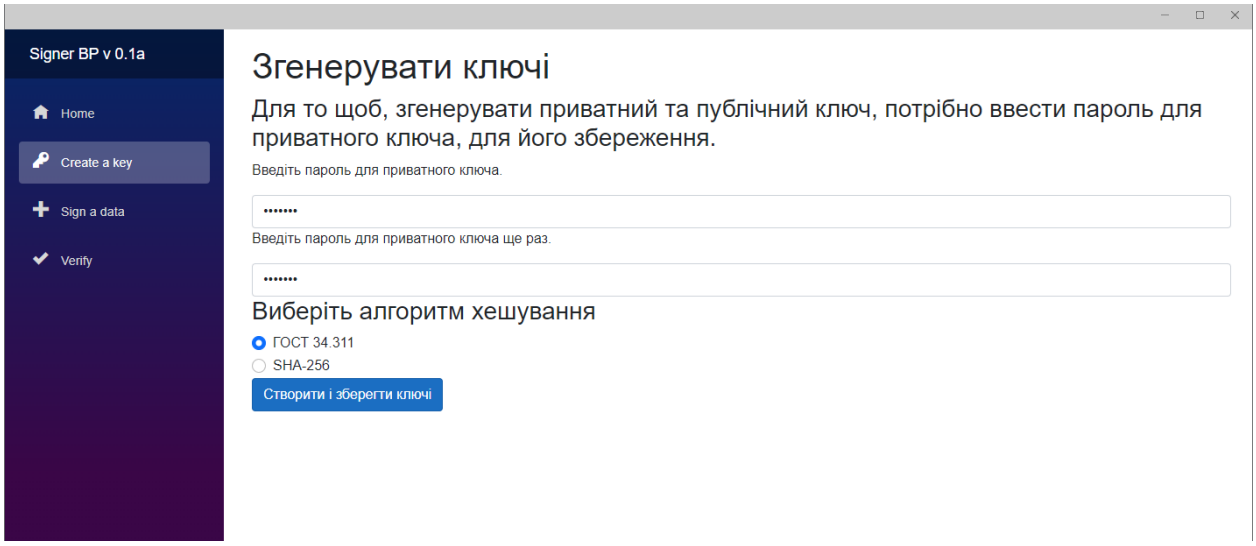


Рисунок 2.2.1

Приватний ключ буде збережений у форматі .p12. Було обрано саме цей формат так як він найпопулярніший і підтримується на багатьох платформах. Обов'язково умовою для створення приватного ключа є введення паролю для його захисту.

Публічний ключ зберігається в X.509 вигляді, а саме у форматі .crt.

Також є можливість обрати алгоритм хешування даних, а саме ГОСТ 34.311 і SHA-256.

На малюнку 3 зображені збережені ключі, створені застосунком, після натиску кнопки "Створити і зберегти ключі".

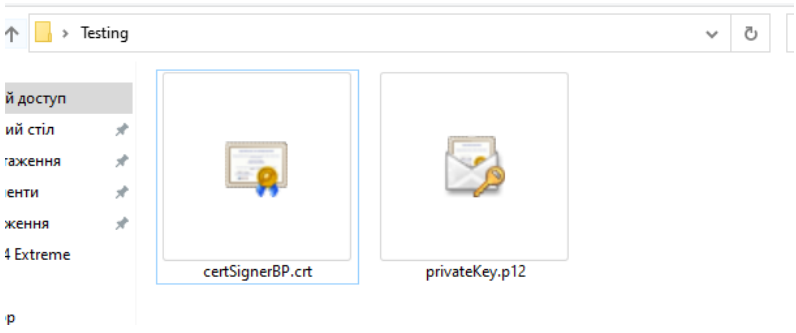


Рисунок 2.2.2

При спробі відкрити приватний ключ на платформі Windows відкриється майстер імпорту сертифікатів для успішного збереження приватного ключа у сховищі ключів, даний процес зображено на малюнку 4.

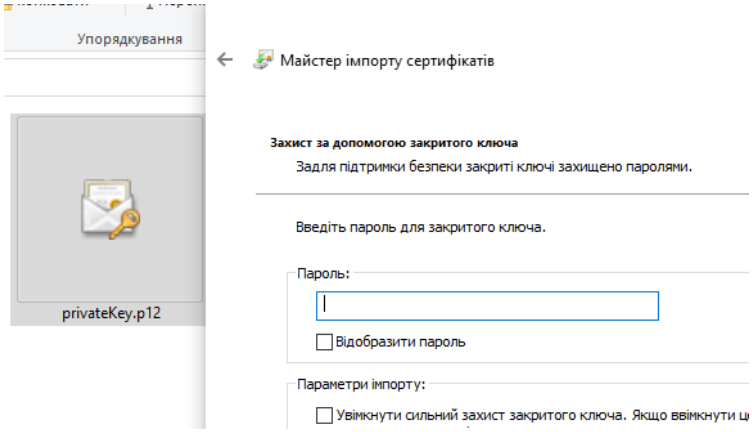


Рисунок 2.2.3

Аналогічно цей формат відкривається на платформі Android.

При спробі відкрити X.509 сертифікат, на платформі Windows він відкриється в нормальному для користувача вигляді. Це зображено на малюнку. Також можна побачити алгоритм підпису, який означає, що це алгоритм ДСТУ 4145 з хеш-функцією ГОСТ 34.311. Також є можливість вибрати іншу хеш-функцію.

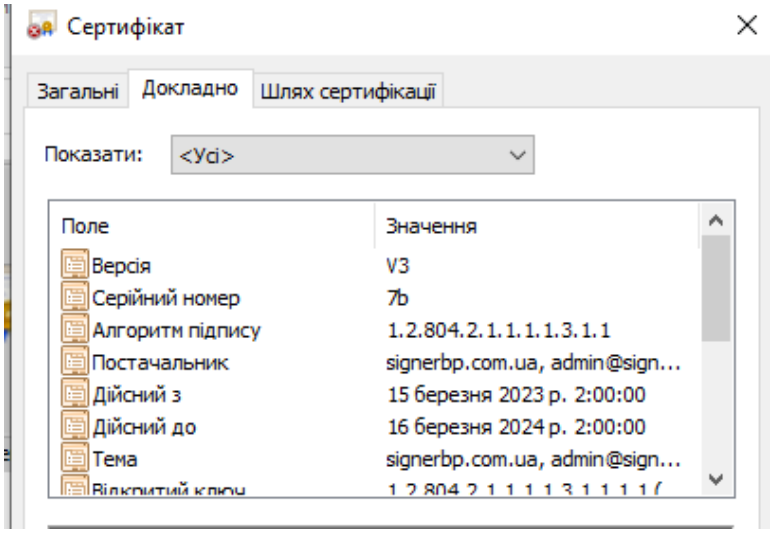


Рисунок 2.2.4

Як можна побачити на малюнку 5 сертифікат містить інформацію про:

- Версію сертифікату
- Серійний номер
- Алгоритм підпису, в даному випадку якщо ж перевірити в сторонніх джерелах
- Постачальник сертифікату – це організація яка випустила сертифікат
- Дійсність сертифікату
- Публічний ключ

2.2.2 Підпис даних

Коли ж уже ключі створені то можна підписувати і верифікувати дані. Наступним етапом після створення ключів є підпис даних. Для підпису даних потрібно мати особистий приватний ключ та власне самі дані. На малюнку 6 зображено процес підпису даних.

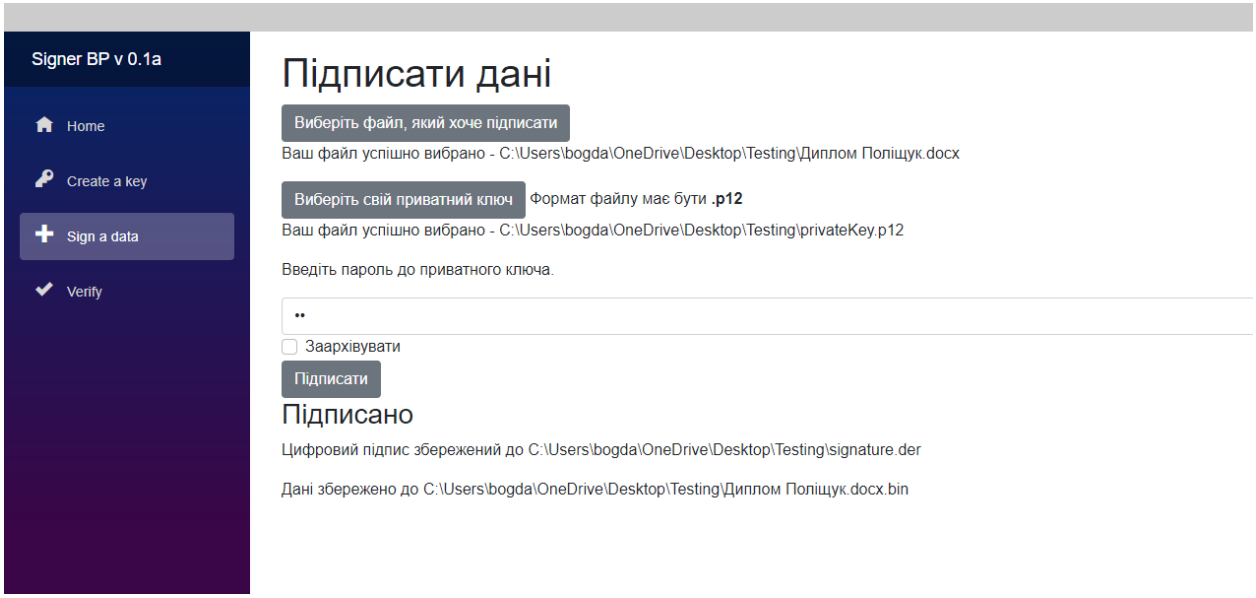


Рисунок 2.2.5

Також є можливість заархівувати дані і цифровий підпис для зручно передачі, даний процес зображено на рисунку 2.2.6.

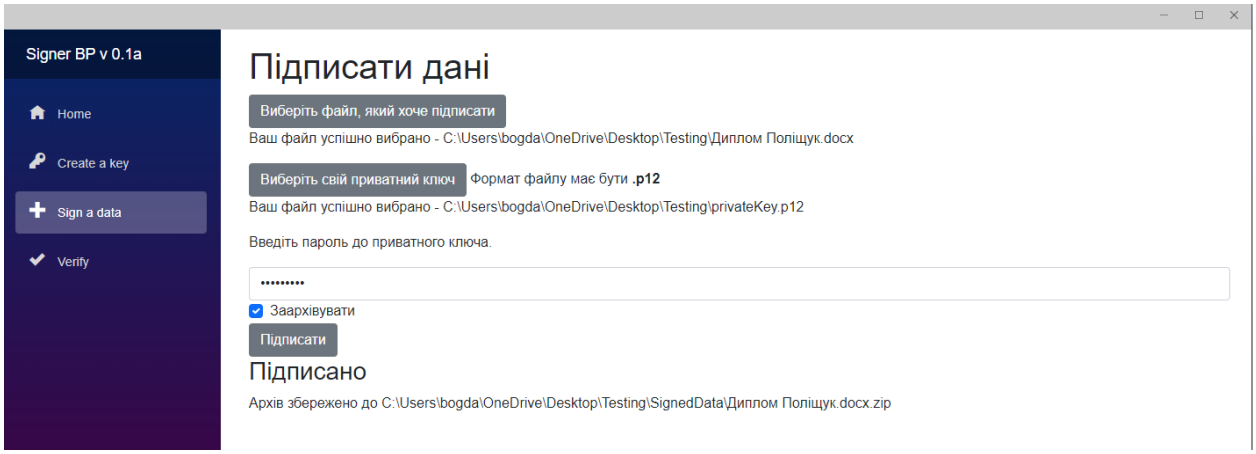


Рисунок 2.2.6

При відкриті архіву з форматом .zip в ньому буде знаходитися 2 файли, а саме дані у форматі масиву байтів з розширенням .bin та цифровий підпис з розширенням .der (рисунок 2.2.7).

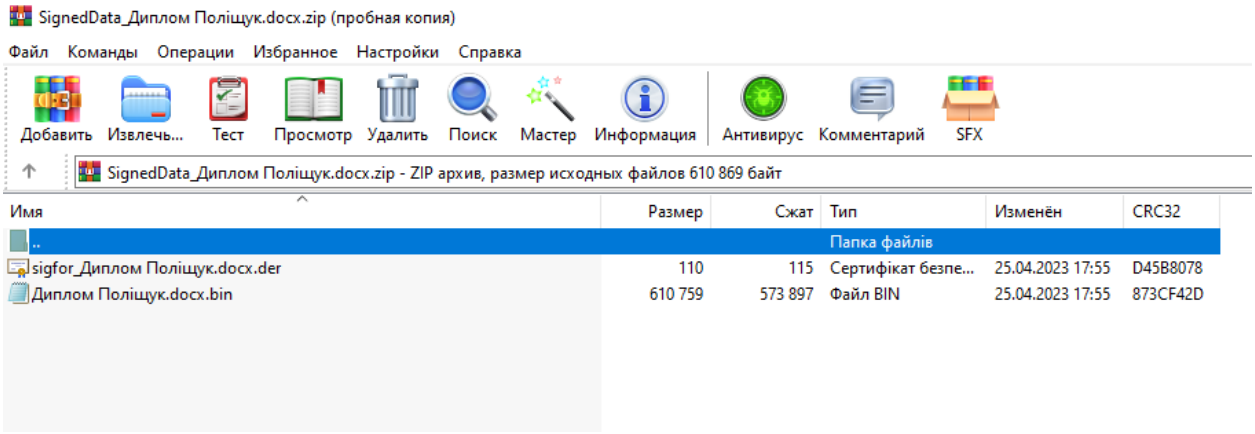


Рисунок 2.2.7

При введенні правильного паролю дані успішно підпишуться. Якщо пароль неправильний то виникне помилка (Рисунок 3). При успішному підпису даних створиться 2 файли: перший електронний відбиток (електронний підпис) має формат .der, другий це файл з даними, які підписуються.

Після підпису даних, цифровий підпис, дані і публічний ключ відправляються будь-яким способом до одержувача, після чого одержувач може за допомогою публічного ключа перевірити цілісність даних.

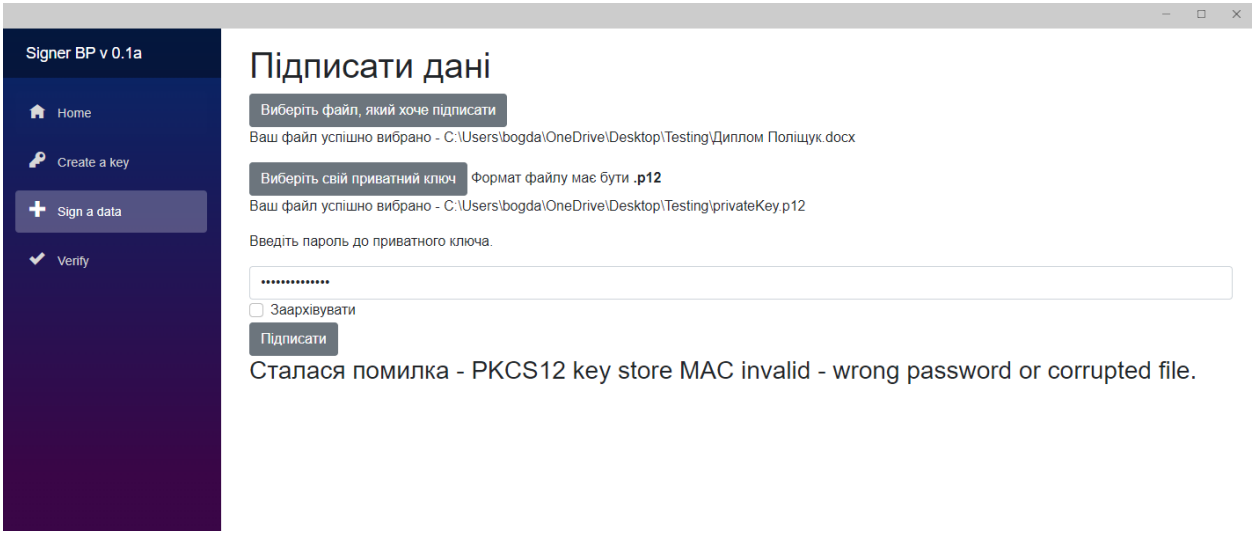


Рисунок 2.2.7

2.2.3 Верифікація цифрового підпису

Отже, передавши оригінальні дані разом з підписом і публічним ключем, одержувач зможе виконати перевірку валідності підпису. Процедура перевірки буде складатися з наступних кроків:

- Хешування оригінальних даних з використанням того ж алгоритму хешування, що і при створенні електронного підпису.
- Декодування електронного підпису, який було отримано з джерела, що перевіряється.
- Перевірка валідності підпису з використанням декодованого підпису, публічного ключа та хешованих оригінальних даних.

Якщо перевірка успішна, то підпис вважається валідним і ви можете довіряти джерелу, яке надіслало вам дані та підпис. Якщо перевірка невдала, то підпис вважається недійсним і вам слід розглянути дані з обережністю.

Іншими слова перевіряючий отримав дані з розширенням .bin, цифровий підпис з розширенням .der та X.509 сертифікат з розширенням .crt. Після отримання цих даних перевіряючий може обрати вкладку Verify у за стосунку та перевірити цілісність даних. Рисунок 2.2.8.

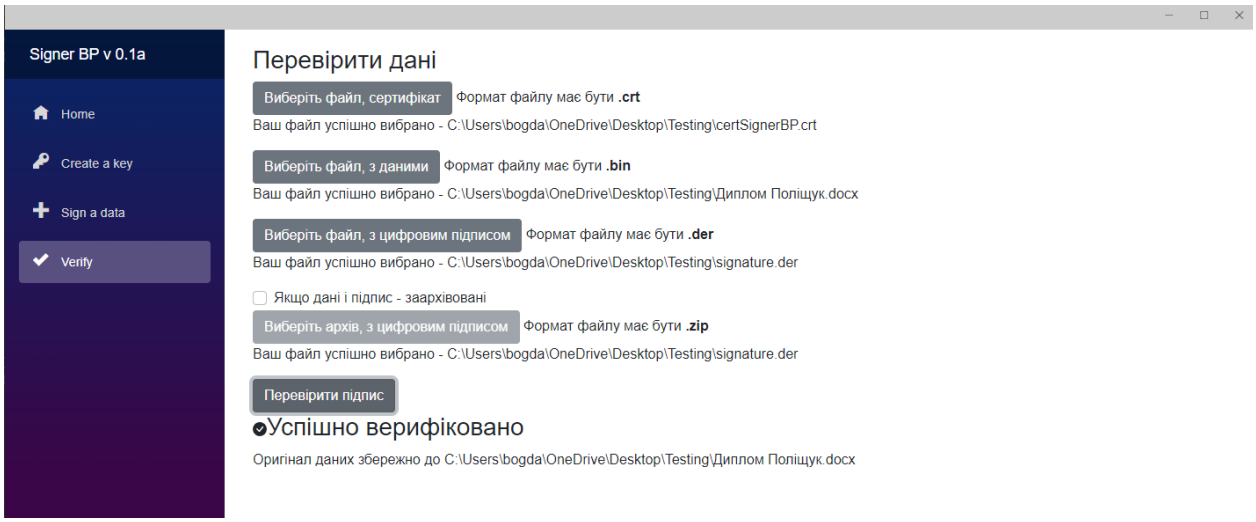


Рисунок 2.2.8

Але зважаючи, що потрібно відправляти аж 3 файли, щоб засвідчитися у валідності, було вирішено добавити перевірку архіву з даними і підписом, який можна створити при підписі даних у вкладці SignData. При першому відкриті вкладки Verify кнопка “Виберіть архів з цифровим підписом” недоступна за замовченням, тому потрібно обрати галочку і тоді кнопки “Виберіть файл з даними” і “Виберіть файл з цифровим підписом” стануть недоступними, так як застосунок очікує на архів, який вже має містити ці файли. Перевірка на цілісність зображена на рисунку 2.2.9.

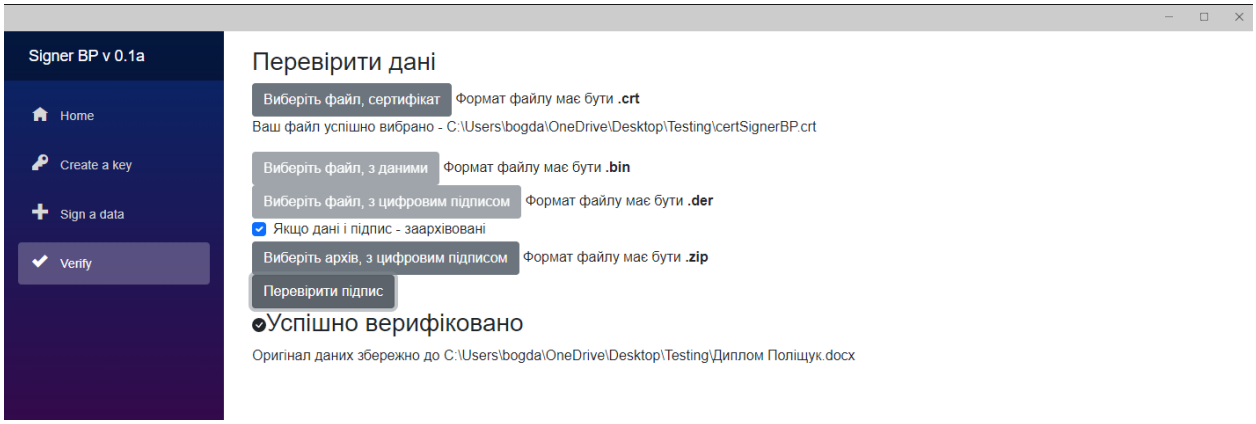


Рисунок 2.2.9

Після успішної перевірки файл з даними перетворюється з масиву байтів у оригіналильний файл з оригінальним розширення, це зображено на рисунку 2.2.10. Було перевірено файл “Диплом Поліщук.docx”

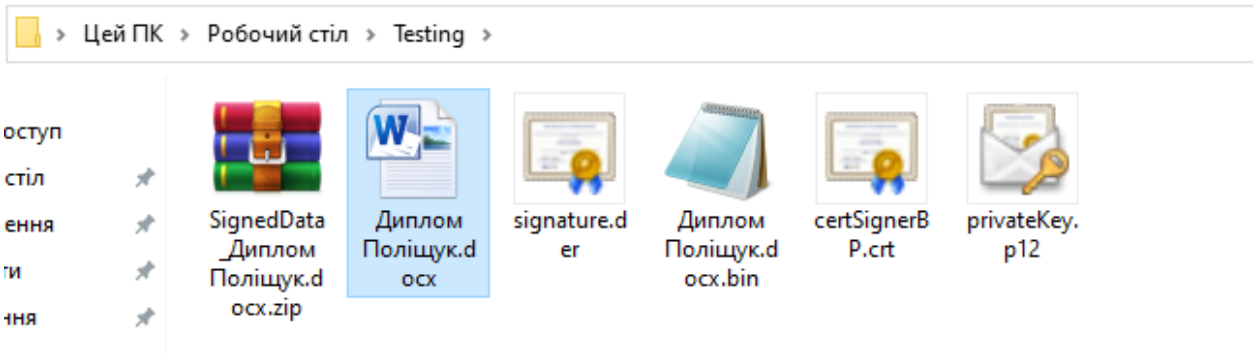


Рисунок 2.2.10

Якщо ж з якимось із трьох файлів – публічний ключ, цифровий підпис чи власне самі дані будуть якісь заміни чи спроба підмінити повідомлення то ці дані не пройдуть перевірку. Для прикладу було підмінено кілька символів

у архіві з даним і цифровим підписом у файлі з даними. На рисунку 2.3.1 було дописано рядок цифр.

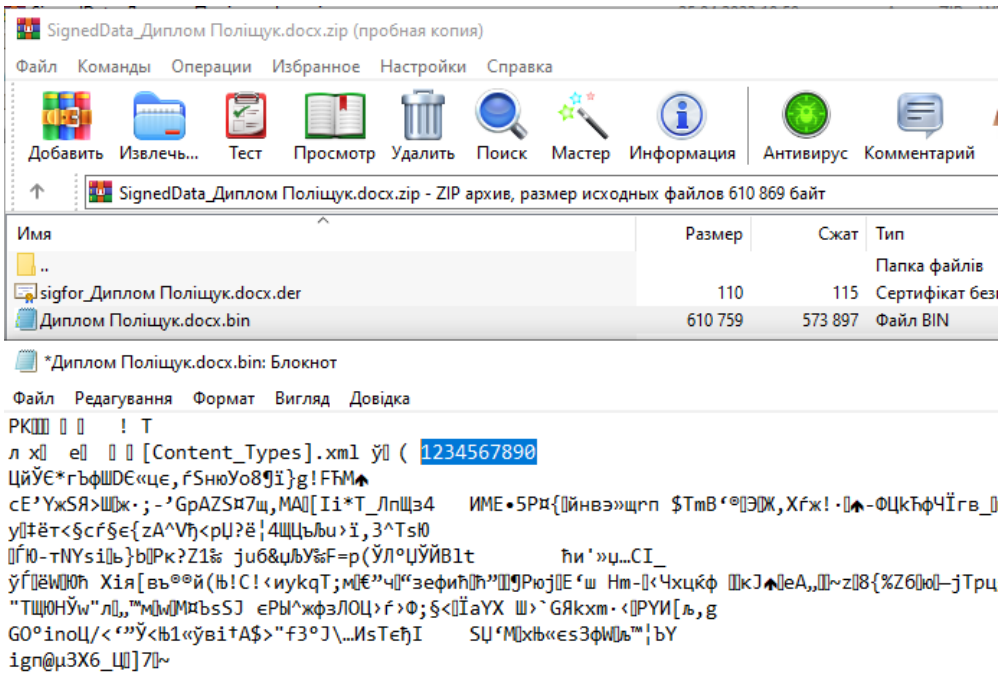


Рисунок 2.3.1

Тепер перевіримо знову цілісність даних. Як можна побачити на рисунку 2.3.2, що застосунок зміг визначити підміну даних і це видно у вигляді помилки, що дані не верифіковано.

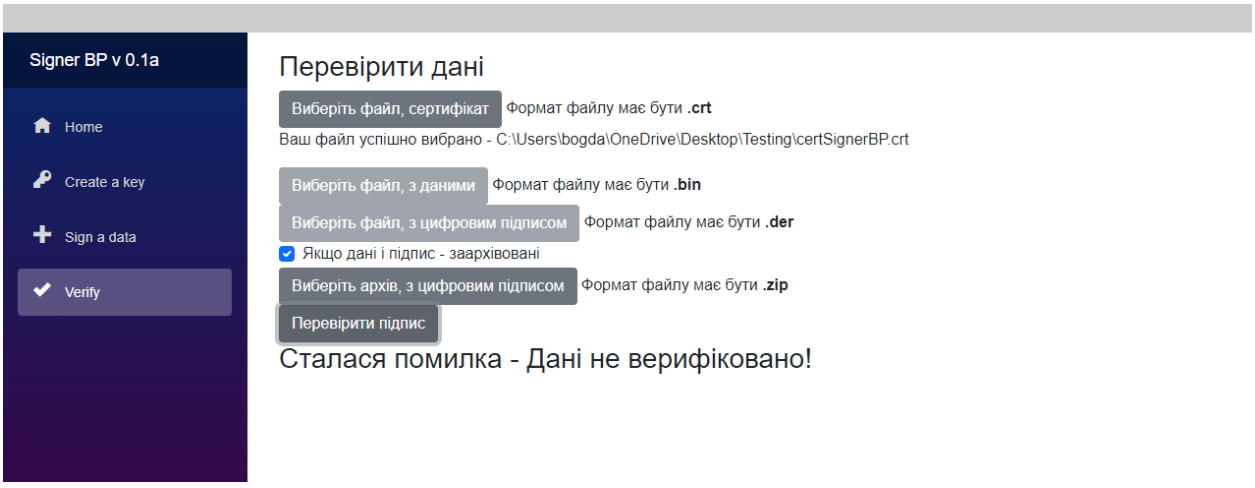


Рисунок 2.3.2

2.3 Крос-платформність

Як говорилося раніше, платформа .NET MAUI підтримує кросплатформеність з такими операційними системами як: Android, iOS, macOS та Windows.

2.3.1 Запуск на платформі Android

На платформі Android було успішно створено X.509 сертифікат і було його відкрито на рисунку 2.3.3. Як можна побачити дані відображаються правильно про компанію, яка створила цей само-підписаний сертифікат.

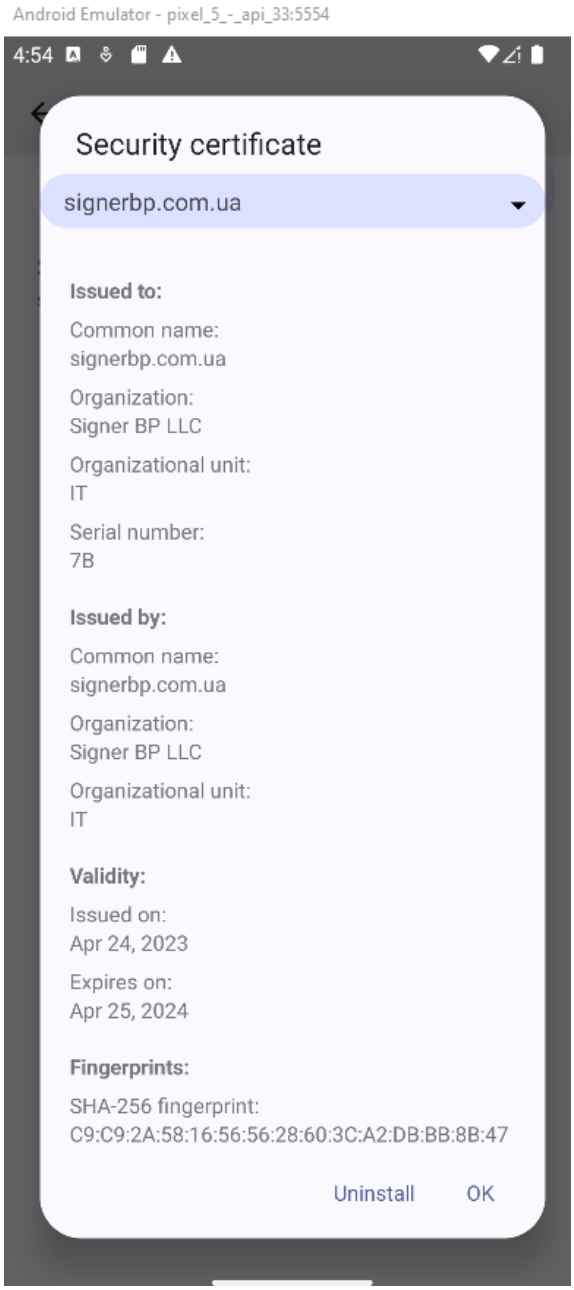


Рисунок 2.3.3

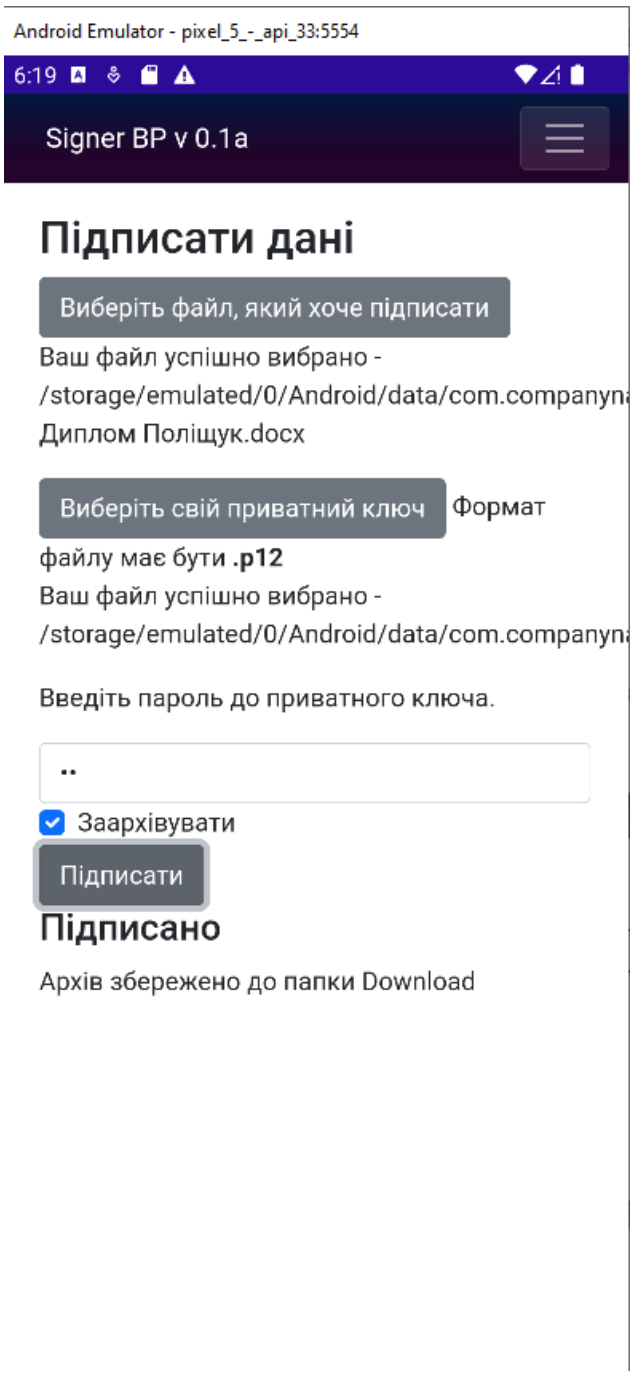


Рисунок 2.3.4

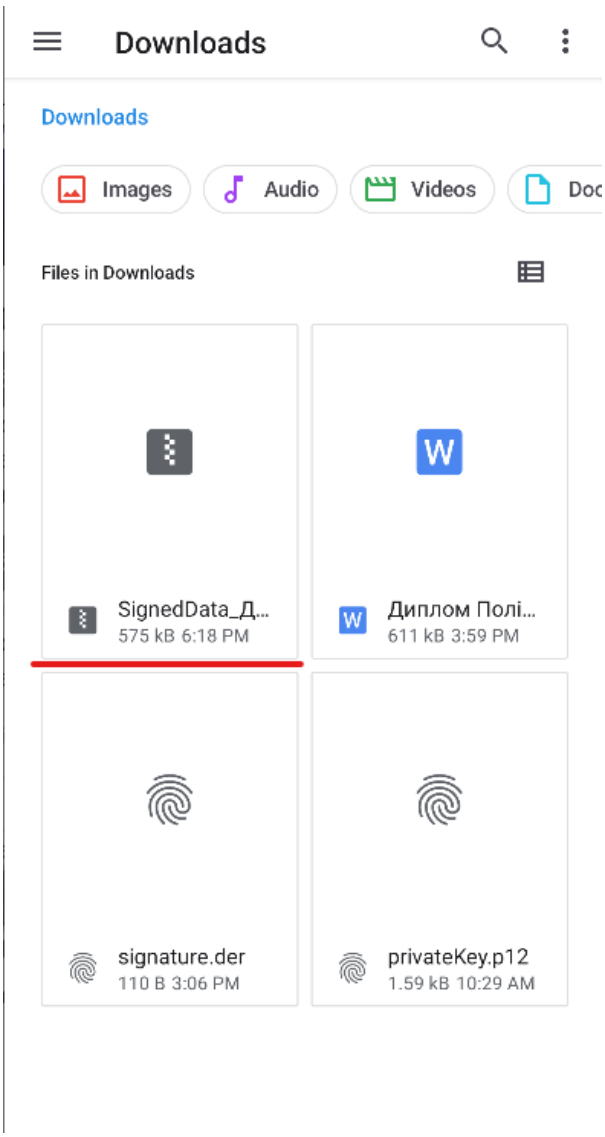


Рисунок 2.3.5

На рисунку 2.3.4 було успішно підписано файл з розширенням .docx. А на рисунку 2.3.5 наглядно видно, що підписані дані збереглися у архів.

Після успішного підпису даних було верифіковано дані на рисунку 2.3.6 на платформі Android.

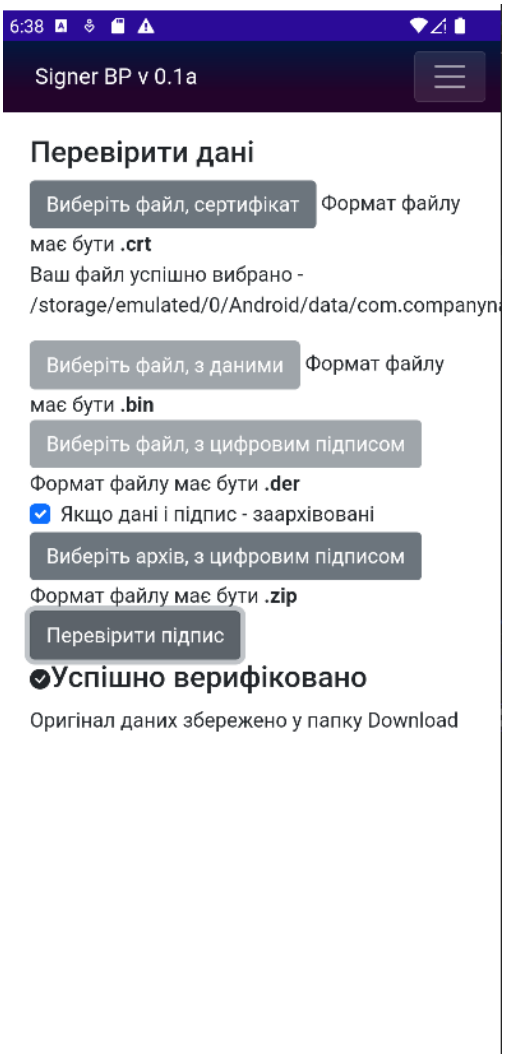


Рисунок 2.3.6

2.3.2 Запуск на платформі iOS

На жаль, тестування неможливе через платну підписку вартістю 99 американських доларів.

ВИСНОВКИ

В ході виконання роботи було створено застосунок платформі .NET MAUI за алгоритмом ДСТУ-4145 з функціями генерації приватного і публічних ключів, підпису даних електронним підписом та верифікація підписаних даних.

Також було протестовано додаток для перевірки правильного функціоналу з успішним результатом. Застосунок може підписувати документи з різними форматами та великими розмірами файлів. Отримано крос-платформність та перевірено функціонал на іншій платформі Android з позитивним результатом.

Застосування електронного підпису даних дозволяє зберігати інформацію у безпечному та надійному форматі, що є важливим у бізнес та правових процесах. Створення застосунку для електронного підпису даних є актуальним та важливим кроком у забезпеченні безпеки та цілісності електронної інформації

ПЕРЕЛІК ПОСИЛАНЬ

1) dstu_4145-2002 [Стандарт] http://www.ksv.biz.ua/GOST/DSTY_ALL/DSTU2/dstu_4145-2002.pdf
2) Ukrainian standard DSTU 4145-2002 [Бібліотека] https://sourceforge.net/projects/dstu4145-2002/
3) Інфрастуктури відкритих ключів. Електронний цифровий підпис. Теорія та приктика. / Ю.І. Горбенко, І.Д. Горбенко./ 2010 [Книга] 11 – 60с.
4) Розшифровані секрети. Методи і принципи криптології / Бауер Ф. / 2007 [Книга] 34 – 52с.
5) What is .NET MAUI? [Електронний ресурс] https://learn.microsoft.com/en-us/dotnet/maui/what-is-maui?view=net-maui-7.0
6) ECC vs RSA: Comparing SSL/TLS Algorithms [Електронний ресурс] https://cheapsslsecurity.com/p/ecc-vs-rsa-comparing-ssl-tls-algorithms/
7) SHA-256 hash calculator [Електронний ресурс] https://xorbin.com/tools/sha256-hash-calculator
8) SHA256 Class [Електронний ресурс] https://learn.microsoft.com/dotnet/api/system.security.cryptography.sha256?view=net-7.0
9) Доступно про криптографію на еліптичних кривих [Електронний ресурс] https://habr.com/ru/post/335906/
10) Вимоги до структури об'єктних ідентифікаторів для криптоалгоритмів, що є державними стандартами [Електронний ресурс] https://zakon.rada.gov.ua/laws/show/z1399-12#Text
11) Створення та реєстрація ЕЦП ДСТУ 4145-2002 [Електронний ресурс] https://auction.uice.com.ua/help/user/dsdstu.html
12) Build a .NET MAUI Blazor Hybrid app [Електронний ресурс] https://learn.microsoft.com/en-

us/aspnet/core/blazor/hybrid/tutorials/maui?view=aspnetcore-7.0&pivots=windows

13) Getting Started with .NET MAUI Blazor Application [Електронний ресурс] https://blazor.syncfusion.com/documentation/getting-started/maui-blazor-app

14) OID Repository [Електронний ресурс]

http://oid-info.com/

15) Ліцензована реалізація стандарту на Java, C++, ObjectPascal [Електронний ресурс] https://web.archive.org/web/20090922214538/http://www.bitis.com.ua/

16) Про затвердження вимог до форматів, структури та протоколів, що реалізуються у надійних засобах електронного цифрового підпису [Електронний ресурс] https://regulation.gov.ua/documents/id29825

Додаток 1 – клас CreateX509

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Org.BouncyCastle.Crypto.Digests;
using Org.BouncyCastle.Asn1.Dstu;
using Org.BouncyCastle.Asn1.Nist;
using Org.BouncyCastle.Asn1.Sec;
using Org.BouncyCastle.Asn1.X509;
using Org.BouncyCastle.Asn1;
using Org.BouncyCastle.Asn1.X9;
using Org.BouncyCastle.Math;
using Org.BouncyCastle.Math.EC;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.Security;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Crypto.Signers;
using Org.BouncyCastle.X509;
using Org.BouncyCastle.Pkcs;
using System.Security.Cryptography;

using Xamarin.Essentials;


namespace SignerBP.Data
{
    public class CreateX509
    {
        static SecureRandom SaltSecure;
        private static void GenerateSecureRNG()
        {
            SaltSecure = SecureRandom.GetInstance("SHA256PRNG");

            var tempObj = new RNGCryptoServiceProvider();
            byte[] passSeed = new byte[128];
            tempObj.GetBytes(passSeed);

            SaltSecure.SetSeed(passSeed);
        }
        public static void X509CertGenerateTests(string password)
        {

            GenerateSecureRNG();

            var OID_curve = DstuObjectIdentifiers.DstuP257;

            var curve_Params = new ECKeyGenerationParameters(OID_curve, SaltSecure);
            var generator_points = new ECKeyPairGenerator("ecdstu4145");
            generator_points.Init(curve_Params);

            var key = generator_points.GenerateKeyPair();

            var serialNumber = BigInteger.ValueOf(123);

            ECPublicKeyParameters ecPub = (ECPublicKeyParameters)key.Public;

            ECPrivateKeyParameters ecPriv = (ECPrivateKeyParameters)key.Private;
```

```
            var attr_for_X509 = new Dictionary<DerObjectIdentifier, string>();
            attr_for_X509[X509Name.C] = "UA";
            attr_for_X509[X509Name.O] = "Signer BP LLC";
            attr_for_X509[X509Name.L] = "Kyiv";
            attr_for_X509[X509Name.ST] = "Kyiv region";
            attr_for_X509[X509Name.OU] = "IT";
            attr_for_X509[X509Name.E] = "admin@signerbp.com.ua";
            attr_for_X509[X509Name.CN] = "signerbp.com.ua";


            var second_attr = new List<DerObjectIdentifier>();
            second_attr.Add(X509Name.C);
            second_attr.Add(X509Name.O);
            second_attr.Add(X509Name.L);
            second_attr.Add(X509Name.ST);
            second_attr.Add(X509Name.OU);
            second_attr.Add(X509Name.E);
            second_attr.Add(X509Name.CN);

            X509V3CertificateGenerator generator_X509 = new X509V3CertificateGenerator();
            generator_X509.SetSerialNumber(serialNumber);

            generator_X509.SetIssuerDN(new X509Name(second_attr, attr_for_X509));
            generator_X509.SetNotBefore(DateTime.Today.Subtract(new TimeSpan(1, 0, 0,
0)));
            generator_X509.SetNotAfter(DateTime.Today.AddYears(1));
            generator_X509.SetSubjectDN(new X509Name(second_attr, attr_for_X509));
            generator_X509.SetPublicKey(ecPub);
            generator_X509.SetSignatureAlgorithm("DSTU4145");
            generator_X509.AddExtension(
                X509Extensions.BasicConstraints,
                true,
                new BasicConstraints(true)
                );
            generator_X509.AddExtension(
                X509Extensions.KeyUsage,
                true,
                new KeyUsage(KeyUsage.CrlSign | KeyUsage.KeyCertSign)
                );
            generator_X509.AddExtension(X509Extensions.ExtendedKeyUsage,
                false,
                new ExtendedKeyUsage(
                      KeyPurposeID.IdKPServerAuth
                      //                    , KeyPurposeID.IdKPClientAuth
                      //                    , KeyPurposeID.IdKPEmailProtection
                      //                    , KeyPurposeID.IdKPCodeSigning
                      //                    , KeyPurposeID.IdKPOcspSigning
                      //                    , KeyPurposeID.IdKPTimeStamping
                      )
                );

            var hash_ = new Gost3411Digest("D-TEST");
            byte[] subjectKeyId = new byte[hash_.GetDigestSize()];
            byte[] pubKeyBytes = new
DerOctetString(ECCUtils.Dstu4145CompressPoint((F2mPoint)ecPub.Q).ToBigInteger().ToByteArr
ay()).GetDerEncoded();
            hash_.BlockUpdate(pubKeyBytes, 0, pubKeyBytes.Length);
            int len = hash_.DoFinal(subjectKeyId, 0);
            generator_X509.AddExtension(
                X509Extensions.SubjectKeyIdentifier,
                false,
                subjectKeyId);

            var genName = new GeneralName(new X509Name(second_attr, attr_for_X509));
```

```
            generator_X509.AddExtension(
                X509Extensions.AuthorityKeyIdentifier,
                false,
                new AuthorityKeyIdentifier(subjectKeyId, new GeneralNames(genName),
serialNumber)
                );

            X509Certificate generat = generator_X509.Generate(ecPriv, SaltSecure);
            generat.CheckValidity();
            generat.Verify(ecPub);

            byte[] derCert = generat.GetEncoded();

File.WriteAllBytes(@"C:\Users\bogda\OneDrive\Desktop\Testing\certSignerBP.crt", derCert);

            var store = new Pkcs12StoreBuilder().Build();
            store.SetKeyEntry(
                "TEST DSTU 4145 Certificate",
                new AsymmetricKeyEntry(ecPriv),
                new X509CertificateEntry[] { new X509CertificateEntry(generat) });

            var pkcs12output = new MemoryStream();
            store.Save(pkcs12output, password.ToCharArray(), SaltSecure);
            File.WriteAllBytes(@"C:\Users\bogda\OneDrive\Desktop\Testing\privateKey.p12",
pkcs12output.ToArray());

        }
    }
}
```

Додаток 2 – клас SignData

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Org.BouncyCastle.Crypto.Digests;
using Org.BouncyCastle.Asn1.Dstu;
using Org.BouncyCastle.Asn1.Nist;
using Org.BouncyCastle.Asn1.Sec;
using Org.BouncyCastle.Asn1.X509;
using Org.BouncyCastle.Asn1;
using Org.BouncyCastle.Asn1.X9;
using Org.BouncyCastle.Math;
using Org.BouncyCastle.Math.EC;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.Security;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Crypto.Signers;
using Org.BouncyCastle.X509;
using Org.BouncyCastle.Pkcs;
using System.Security.Cryptography;

namespace SignerBP.Data
{
    internal class SignData
    {
        private static ECPrivateKeyParameters privateKey;
        static SecureRandom Salt_Secure;
        private static void GenerateSecureRNG()
        {
            Salt_Secure = SecureRandom.GetInstance("SHA256PRNG");

            var tempObj = new RNGCryptoServiceProvider();
            byte[] Pass_Seed = new byte[128];
            tempObj.GetBytes(Pass_Seed);

            Salt_Secure.SetSeed(Pass_Seed);
        }
        public static bool ReadP12File(string pathToPrivateKey, string pathToFile, string
password)
        {
            byte[] file_P12 = File.ReadAllBytes(pathToPrivateKey);

            var store_p12 = new Pkcs12Store();
            store_p12.Load(new MemoryStream(file_P12), password.ToCharArray());
            privateKey = (ECPrivateKeyParameters)(store_p12.GetKey("TEST DSTU 4145
Certificate").Key);
            GenerateSecureRNG();
            ECDstu4145SignerMethod(pathToFile);
            return true;
        }
        public static void ECDstu4145SignerMethod(string pathToFile)
        {

            byte[] data = File.ReadAllBytes(pathToFile);


            var signer = new ECDstu4145DigestSigner(
                new ECDstu4145Signer(),
                CreateGost34311Digest());
```

```
            signer.Init(true, new ParametersWithRandom(privateKey, Salt_Secure));
            signer.BlockUpdate(data, 0, data.Length);
            byte[] sig = signer.GenerateSignature();

            File.WriteAllBytes(@"C:\Users\bogda\OneDrive\Desktop\Testing\signature.der",
sig);
            File.WriteAllBytes(@"C:\Users\bogda\OneDrive\Desktop\Testing\data.bin",
data);
        }
        private static Gost3411Digest CreateGost34311Digest()
        {
            return new Gost3411Digest("D-TEST");
        }
    }
}
```

Додаток 3 – клас Verify

```
using System;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Org.BouncyCastle.Crypto.Digests;
using Org.BouncyCastle.Asn1.Dstu;
using Org.BouncyCastle.Asn1.Nist;
using Org.BouncyCastle.Asn1.Sec;
using Org.BouncyCastle.Asn1.X509;
using Org.BouncyCastle.Asn1;
using Org.BouncyCastle.Asn1.X9;
using Org.BouncyCastle.Math;
using Org.BouncyCastle.Math.EC;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.Security;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Crypto.Signers;
using Org.BouncyCastle.X509;
using Org.BouncyCastle.Pkcs;

using System.IO.Compression;


namespace SignerBP.Data
{
    internal class Verify
    {
        public static void GetDataFromArchive(string pathToArchive)
        {
            byte[] data = { };
            string itemName = "";
            using (var archive = ZipFile.OpenRead(pathToArchive))
            {

                var entries = archive.Entries;
                foreach (var item in entries)
                {
                    if (Path.GetExtension(item.Name).TrimStart('.') == "bin")
                    {
                        using (var stream = item.Open())
                        using (var ms = new MemoryStream())
                        {
                            stream.CopyTo(ms);
                            data = ms.ToArray();
                        }
                        itemName = item.Name;
                    }
                }
            }
            itemName = itemName.Replace(".bin", "");
#if __ANDROID__

var downloadFolder =
Android.OS.Environment.GetExternalStoragePublicDirectory(Android.OS.Environment.Direc
toryDownloads).AbsolutePath;
```

```
var filePath = Path.Combine(downloadFolder, $"{itemName}");

File.WriteAllBytes(filePath, data);

#endif
            if (OperatingSystem.IsAndroid())
            {
            }
            else
            {

File.WriteAllBytes($@"C:\Users\bogda\OneDrive\Desktop\Testing\{itemName}", data);
            }


        }
        public static bool VerifyData(string pathToCrt, string pathToData, string
pathToSignature, string pathToArchive, bool isArchived)
        {
            byte[] data, sig;
            if (isArchived)
            {
                using (var archive = ZipFile.OpenRead(pathToArchive))
                {

                    var entries = archive.Entries;


                    var firstEntry = entries[0];
                    var extensionName =
Path.GetExtension(firstEntry.Name).TrimStart('.');
                    if (extensionName == "bin")
                    {
                        using (var stream = firstEntry.Open())
                        using (var ms = new MemoryStream())
                        {
                            stream.CopyTo(ms);
                            data = ms.ToArray();
                        }
                        firstEntry = entries[1];
                        using (var stream = firstEntry.Open())
                        using (var ms = new MemoryStream())
                        {
                            stream.CopyTo(ms);
                            sig = ms.ToArray();
                        }
                    }
                    else
                    {
                        using (var stream = firstEntry.Open())
                        using (var ms = new MemoryStream())
                        {
                            stream.CopyTo(ms);
                            sig = ms.ToArray();
                        }
                        firstEntry = entries[1];
                        using (var stream = firstEntry.Open())
                        using (var ms = new MemoryStream())
                        {
                            stream.CopyTo(ms);
```

```
                        data = ms.ToArray();
                    }
                }

            }
        }
        else
        {
            data = File.ReadAllBytes(pathToData);

            sig = File.ReadAllBytes(pathToSignature);
        }


        ECPublicKeyParameters publicKey;
        byte[] caCertBytes;

        if (OperatingSystem.IsAndroid())
        {
            using (var archive = ZipFile.OpenRead(pathToCrt))
            {

                var entries = archive.Entries;

                using (var stream = entries[0].Open())
                using (var ms = new MemoryStream())
                {
                    stream.CopyTo(ms);
                    caCertBytes = ms.ToArray();
                }
            }
        }
        else
        {
            caCertBytes = File.ReadAllBytes(pathToCrt);
        }

        var X509_cert = new X509CertificateParser().ReadCertificate(caCertBytes);
        var X509_params = X509_cert.GetPublicKey();
        byte[] dke = null;
        ECPublicKeyParameters caPubKey = null;
        if (X509_params is Dstu4145KeyParametersWithDKE)
        {
            dke = ((Dstu4145KeyParametersWithDKE)X509_params).DKE;
            caPubKey =
(ECPublicKeyParameters)((Dstu4145KeyParametersWithDKE)X509_params).Key;
        }
        else
            caPubKey = (ECPublicKeyParameters)X509_params;


        publicKey = caPubKey;
        var signer = new ECDstu4145DigestSigner(
           new ECDstu4145Signer(),
           CreateGost34311Digest());

        signer.Init(false, publicKey);
        signer.BlockUpdate(data, 0, data.Length);
        return signer.VerifySignature(sig);

    }
```

```
        private static Gost3411Digest CreateGost34311Digest()
        {
            return new Gost3411Digest("D-TEST");
        }

    }
}
```

Додаток 4 – Сторінка Sign

```
@page "/sign"
@using SignerBP.Data;



<h1>Підписати дані</h1>
<div>
    <button @onclick="OnPickFile" type="button" class="btn btn-secondary">Виберіть файл,
який хоче підписати</button>
    @{
        if (filePathFileToSign != null)
        {
            <p>Ваш файл успішно вибрано - @filePathFileToSign</p>
        }
    }
</div>
<div>
    <button @onclick="OnPickPrivaeKey" type="button" class="btn btn-secondary">Виберіть
свій приватний ключ</button> <span>Формат файлу має бути <b>.p12</b></span>
    @{
        if (filePathPrivateKey != null)
        {
            <p>Ваш файл успішно вибрано - @filePathPrivateKey</p>
        }
    }
</div>
<div>
    <p>Введіть пароль до приватного ключа.</p>
    <input type="password" class="form-control" name="Enter password" id="Password"
@oninput="OnInputEvent">
</div>
<div>
    <button @onclick="Sign" type="button" class="btn btn-secondary">Підписати</button>
</div>
@{
    if (success)
    {
        <h3>Підписано</h3>
        <p>Цифровий підпис збережений до
C:\Users\bogda\OneDrive\Desktop\Testing\signature.der</p>
        <p>Дані збережено до C:\Users\bogda\OneDrive\Desktop\Testing\data.bin</p>
    }
    else if (ErrorMessage != null)
    {
        <h3>Сталася помилка - @ErrorMessage</h3>
    }
}

@code {
    private string filePathFileToSign;
    private string filePathPrivateKey;
    private string password;
    private bool success = false;
    private string ErrorMessage;

    private async Task OnPickFile()
    {
        var result = await FilePicker.PickAsync();
        if (result != null)
        {
```

```
            filePathFileToSign = result.FullPath;
        }
    }
    private async Task OnPickPrivaeKey()
    {
        var result = await FilePicker.PickAsync(new PickOptions
            {
                PickerTitle = "Select a .p12 file",
                FileTypes = new FilePickerFileType(new Dictionary<DevicePlatform,
IEnumerable<string>>
            {
                { DevicePlatform.Android, new[] { "application/x-pkcs12" } },
                { DevicePlatform.iOS, new[] { "com.rsa.pkcs-12" } },
                { DevicePlatform.WinUI, new[] { ".p12" } }
            })
            });


        if (result != null)
        {
            filePathPrivateKey = result.FullPath;
        }
    }
    private void Sign()
    {
        try
        {
            success = SignData.ReadP12File(filePathPrivateKey, filePathFileToSign,
password);
        }catch (Exception e)
        {
            ErrorMessage = e.Message;
        }

    }
    private void OnInputEvent(ChangeEventArgs changeEvent)
    {
        password = (string)changeEvent.Value;
    }

}
```

Додаток 5 – Сторінка Create a key

```
@page "/creatingkey"
@using SignerBP.Data;
@using Xamarin.Essentials;

<h1>Згенерувати ключі</h1>

<h3>Для то щоб, згенерувати приватний та публічний ключ, потрібно ввести пароль для приватного ключа, для його збереження.</h3>

<p>Введіть пароль для приватного ключа.</p>
<input type="password" class="form-control" name="Enter password" id="Password" @oninput="OnInputEvent">

<p>Введіть пароль для приватного ключа ще раз.</p>
<input type="password" class="form-control" name="Enter password again" id="Password" @oninput="OnInputEventAgain">
@{
    if(password != password2)
    {
        <p>Будь ласка, введіть однакові паролі.</p>
    }
}

<div>
    <button @onclick="OnPickFile" type="button" class="btn btn-secondary">Виберіть директорію, куди зберегти клчюі</button>
    @{
        if (filePathFileToSave != null)
        {
            <p>Директорія вибрана - @filePathFileToSave</p>
        }
    }
</div>

<button class="btn btn-primary" @onclick="(() => InitSigner())">Створити і зберегти ключі</button>

@{
    if (valid)
    {
        <p>Приватний ключ збережений до C:\Users\bogda\OneDrive\Desktop\Testing\privateKey.p12</p>
        <p>Публічний ключ збережений до C:\Users\bogda\OneDrive\Desktop\Testing\certSignerBP.crt</p>
    }else if(ErrorMessage!=null)
    {
        <h3>Сталася помилка - @ErrorMessage</h3>
    }
}

@code {
    private string password = "";
    private string password2 = "";
    private string filePathFileToSave;
    private bool valid = false;
    private string ErrorMessage;


    private void OnInputEvent(ChangeEventArgs changeEvent)
    {
        password = (string)changeEvent.Value;
    }
```

```
    private void OnInputEventAgain(ChangeEventArgs changeEvent)
    {
        password2 = (string)changeEvent.Value;
    }
    private async Task OnPickFile()
    {
        var result = await FilePicker.PickAsync();
        if (result != null)
        {
            filePathFileToSave = result.FullPath;
        }
    }

    private void InitSigner()
    {
        try
        {
            if (password != password2)
                throw new PasswordException("Паролі різні");

            CreateX509.X509CertGenerateTests(password);
            valid = true;

        }catch (Exception e)
        {
            ErrorMessage = e.Message;
        }
    }
}
```

Додаток 6 – Сторінка Verify

```
@page "/verify"
@using SignerBP.Data;

<h3>Перевірити дані</h3>

<div>
    <button @onclick="OnPickCRT" type="button" class="btn btn-secondary">Виберіть файл,
сертифікат</button> <span>Формат файлу має бути <b>.crt</b></span>
    @{
        if (filePathCRT != null)
        {
            <p>Ваш файл успішно вибрано - @filePathCRT</p>
        }
    }
</div>
<div>
    <button @onclick="OnPickData" type="button" class="btn btn-secondary">Виберіть файл,
з даними</button> <span>Формат файлу має бути <b>.bin</b></span>
    @{
        if (filePathData != null)
        {
            <p>Ваш файл успішно вибрано - @filePathData</p>
        }
    }
</div>
<div>
    <button @onclick="OnPickSignature" type="button" class="btn btn-secondary">Виберіть
файл, з цифровим підписом</button> <span>Формат файлу має бути <b>.der</b></span>
    @{
        if (filePathFileSignature != null)
        {
            <p>Ваш файл успішно вибрано - @filePathFileSignature</p>
        }
    }
</div>
<div>
    <button @onclick="Sign" type="button" class="btn btn-secondary">Перевірити
підпис</button>
</div>
@{
    if (success)
    {
        <h3>Успішно верифіковано</h3>
        <p>Оригінал даних збережно до /storage/downloads/somefile.pdf</p>

    }
    else if (ErrorMessage != null)
    {
        <h3>Сталася помилка - @ErrorMessage</h3>
    }
}
@code {
    private string filePathCRT;
    private string filePathData;
    private string filePathFileSignature;
    private bool success = false;
    private string ErrorMessage;

    private async Task OnPickCRT()
    {
        var result = await FilePicker.PickAsync(new PickOptions
            {
```

```
                PickerTitle = "Select a .p12 file",
                FileTypes = new FilePickerFileType(new Dictionary<DevicePlatform,
IEnumerable<string>>
            {
                { DevicePlatform.Android, new[] { "application/zip" } },
                { DevicePlatform.iOS, new[] { "com.rsa.pkcs-12" } },
                { DevicePlatform.WinUI, new[] { ".zip" } }
            })
            });


        if (result != null)
        {
            filePathCRT = result.FullPath;
        }
    }
    private async Task OnPickData()
    {
        var result = await FilePicker.PickAsync(new PickOptions
            {
                PickerTitle = "Select a .bin file",
                FileTypes = new FilePickerFileType(new Dictionary<DevicePlatform,
IEnumerable<string>>
            {
                { DevicePlatform.Android, new[] { "application/octet-stream" } },
                { DevicePlatform.iOS, new[] { "com.rsa.pkcs-12" } },
                { DevicePlatform.WinUI, new[] { ".bin" } }
            })
            });


        if (result != null)
        {
            filePathData = result.FullPath;
        }
    }
    private async Task OnPickSignature()
    {
        var result = await FilePicker.PickAsync(new PickOptions
            {
                PickerTitle = "Select a .der file",
                FileTypes = new FilePickerFileType(new Dictionary<DevicePlatform,
IEnumerable<string>>
            {
                { DevicePlatform.Android, new[] { "application/octet-stream" } },
                { DevicePlatform.iOS, new[] { "com.rsa.pkcs-12" } },
                { DevicePlatform.WinUI, new[] { ".der" } }
            })
            });


        if (result != null)
        {
            filePathFileSignature = result.FullPath;
        }
    }
    private void Sign()
    {
        try
        {
            success = Data.Verify.VerifyData(filePathCRT, filePathData,
filePathFileSignature);
            if (success)
            {
```

```
                byte[] fileBytes = File.ReadAllBytes(filePathData);
                string fileText = Encoding.UTF8.GetString(fileBytes);

File.WriteAllText(@"C:\Users\bogda\OneDrive\Desktop\Testing\OriginalData.txt", fileText);
            }
            else
            {
                throw new Exception("Дані не верифіковано!");
            }
        }
        catch (Exception e)
        {
            ErrorMessage = e.Message;
        }

    }
}
```