Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____

Кафедра_____Програмної інженерії_____

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти – другий (магістерський)

_Дослідження та порівняльний аналіз архітектури центрів сертифікації ключів.
_____Реалізація послуги контролю цілісності._____

Виконав: студент 2 курсу, групи ІПЗм-18-4___
_____Білецький В.В._____

спеціальності 121 – Інженерія програмного забезпечення_____

Освітньо-наукової програми
__Інженерія програмного забезпечення_____

Керівник _____проф. Качко О.Г._____

Допускається до захисту Зав. кафедри, проф. _____ З.В.Дудар

Харків, 2020

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет_Комп'ютерних наук_____

Кафедра_Програмної інженерії_____

Рівень вищої освіти – другий (магістерський)

Спеціальність _121 – Інженерія програмного забезпечення_____

Тип програми освітньо-професійна програма

Освітня програма_Інженерія програмного забезпечення_

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

«_____»_____ 20 ___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Білецькому Владиславу Валерійовичу_____

1. Тема роботи Дослідження та порівняльний аналіз архітектури центрів сертифікації ключів. Реалізація послуги контролю цілісності._ затверджена наказом університету від "___" _____20 ___ р. № _____

2. Термін подання студентом роботи до екзаменаційної комісії
18 травня 2020 р.

3. Вихідні дані до роботи алгоритми контролю цілісності та криптографії, алгоритми перевірки цілісності документа, пояснювальна записка. Використовувати ОС Microsoft Windows, середовище об'єктно-орієнтованого проектування.

4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз проблемної галузі і постановка задачі, огляд архітектури центрів сертифікації ключів, порівняльній аналіз, методи перевірки цілісності.

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	проф. Качко О.Г.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка *
1.	Аналіз предметної галузі	15 березня 2020 р.	
2.	Огляд існуючих архітектур центрів сертифікації ключів	05 квітня 2020 р.	
3.	Аналіз методів контролю цілісності	20 квітня 2020 р.	
4.	Підготовка пояснювальної записки	01 травня 2020 р.	
5.	Спецчастина	15 травня 2020 р.	
6.	Підготовка презентації та доповіді	7 травня 2020 р.	
7.	Попередній захист	8 травня 2020 р.	
8.	Нормоконтроль, рецензування	13 травня 2020 р.	
9.	Занесення диплома в електронний архів	14 травня 2020 р.	
10.	Допуск до захисту у зав. кафедри	15 травня 2020 р.	
* заповнюється вручну після виконання чергового пункту			

Дата видачі завдання _25_ ____вересня_ 2019 р.

Студент _____

Керівник роботи _____ __проф. Качко О.Г._____

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить 49 сторінок, 26 рисунків, 15 джерел.

C#, ЦЕНТР СЕРТИФІКАЦІЇ КЛЮЧІВ, ОНОВЛЕННЯ СЕРТИФІКАТІВ, БАЗА ДАНИХ, JETBRAINS RIDER, NET CORE 3, MICROSOFT SQL SERVER 2016.

Метою роботи є дослідження сучасних архітектур центрів сертифікації ключів, їх порівняльний аналіз з подальшою розробкою сховища ключів з підтримкою заміни та перевірки цілісності.

Засобами і методами розробки є JetBrains Rider, ASP.NET Core, NET Core 3, C #, Microsoft SQL Server 2016, Postgre SQL, Docker.

В результаті розробки отримана програма «CertOnTheGo», яка може використовуватися користувачами для генерації, зберігання та оновлення сертифікатів. Програма має надавати можливості для зберігання та перевірки цілісності сертифікатів.

The Master’s certification work includes 49 pages, 26 figures, 15 sources.

C #, KEY CERTIFICATION CENTER, UPDATE CERTIFICATES, DATA BASIS, JETBRAINS RIDER, NET CORE 3, MICROSOFT SQL SERVER 2016.

The purpose of the work is to analyze several modern architectures of certification centers, compare their pros and cons and create certificate store which allows to store, replace certificates and check their integrity.

The tools and methods of development are JetBrains Rider, ASP.NET Core, NET Core 3, C #, Microsoft SQL Server 2016, Postgre SQL, Docker.

The result is a “CertOnTheGo” software that can be used by customers to generate, store, and update certificates. The program should be able to store and verify the integrity of certificates.

ЗМІСТ

ВСТУП

В даний час приватність та конфіденційність інформації стає все більш важливою, невід’ємною частиною нашого процесу користування мережею

Інтернет. Все більше можливостей є у системі, які здатні полегшити наше життя[5]:

– онлайн магазини, що дають можливість придбати товар не виходячи з дому;

– резервування квитків на транспорт та різні заходи, столів у кафе та ресторанах, номерів у готелях;

– оплата комунальних послуг без черг та багато іншого.

Що ж поєднує всі ці можливості? Їх поєднує конфіденційність інформації, яку користувач повинен передати стороні, що надає послуги. Ця інформація може бути будь-якого типу: текстовою (особисті дані, номер кредитної картки, дані страховки), аудіовізуальною (секретні записи розмов, відеоматеріали, тощо), бінарною (особисті файли). Уся ця інформація має бути отримана і прочитана лише адресатом, та в разі перехопленні або втрати не розкриватися викрадачеві. Для цього вже давно існує механізм шифрування даних, який дозволяє отримати ключ, зашифрувати повідомлення та бути впевненим, що інформацію захищено від сторонніх.

Здавалося б, коли є ключ та механізм – то проблема вирішена. Але ні, в цій схемі дуже багато нюансів, як, наприклад, доставка ключів до адресата, оновлення ключів, вилучення ключів, їх зберігання, перевірка цілісності.

Всю цю роботу бере на себе центр сертифікації ключів, і саме від нього залежить, наскільки безпечним буде використання особистих даних, ключів та шифрування.

Але для розробників програмного забезпечення може стати проблемою підтримання багатьох сертифікатів в системі, їх постійне оновлення та заміна в разі відзиву.

Іншою проблемою є те, що технічний прогрес відбувається значно швидше, ніж процес створення нових криптографічних рішень, проте настільки неоднорідно, що не кожен пристрій зміг би виконувати обчислення, необхідні для підтримання стандартів безпеки актуальними. Стандарти, які були прийняті вже двадцять років тому, вже не можуть забезпечити необхідного рівня захисту.

Протоколи, що використовують хешування SHA-256 та шифрування RSA-4096 вже не можуть бути гарантом захисту, бо через декілька років такий шифр можна буде зламати досить швидко[4].

Сучасні центри сертифікації ключів використовують саме ці механізми, бо вони є стандартними для всіх сфер, і не можуть бути замінені достатньо швидко. Кожного дня ми чуємо про те, що сервери компаній піддаються атакам, під час котрих інформація видаляється з серверів, потрапляє у відкритий доступ. А що ж буде, якщо під час такої атаки хакер замінить сертифікати безпеки на свої, згенеровані таким чином, щоб зловмисник міг розшифровувати дані? В такому випадку не тільки дані компанії, а й дані її користувачів будуть під загрозою.

Саме для цього і була створена система CertOnTheGo. Завдяки цій системі користувачі зможуть зберігати сертифікати в одному місці, отримувати їх по прямому посиланню, яке буде зберігатися за сертифікатом при оновленні чи заміні. Ця система буде використовувати посилене шифрування для усіх даних, що зберігаються у ній, що зробить неможливою підміну сертифікату та перехоплення конфіденційної інформації, що дасть можливість подовжити строки роботи загальноприйнятих механізмів захисту до того часу, доки усі структури не зможуть перейти на нові протоколи.

1 АНАЛІЗ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної галузі

На початку введемо поняття сертифікату. Сертифікат відкритого ключа – електронний або паперовий документ, що містить відкритий ключ, інформацію про власника ключа, сферу використання та що є підписаним центром сертифікації. Сертифікати можуть бути цифрові або паперові. Цифрові можуть видаватися за допомогою електронних засобів, форма сертифіката визначається стандартом X.509. Паперові сертифікати видаються центром сертифікації напряму власнику в офісі компанії центра сертифікації. Паперовий сертифікат повинен видаватися на підставі підтвердних документів і в присутності особи з подальшим засвідченням підписом працівника центру і носія закритого ключа. Сертифікати мають наступні поля[5]:

- відкритий ключ сертифікату;
- серійний номер сертифікату;
- термін дії;
- ім’я центру сертифікації;
- ім’я власника сертифікату;
- цифровий підпис.

Одними з найбільш великих постачальників сертифікатів є:

- Comodo (PositiveSSL Wildcard, Essential Wildcard SSL, Premium Wildcard SSL, SSL Wildcard);
- Thawte (SSL 123 Wildcard, Wildcard SSL Certificate);
- Symantec (Secure Site Wildcard);
- GeoTrust (QuickSSL Premium Wildcard, TrueBusinessID Wildcard);
- RapidSSL (WildcardSSL).
- GlobalSign (WildcardSSL).

Цифровий підпис гарантує неможливість підробки сертифікату, бо є результатом криптографічної хеш-функції від даних сертифікату[5],

зашифрованими закритим ключем центра сертифікації. Відкритий ключ сертифікату є загальновідомим, що дає змогу розшифрувати цифровий підпис сертифікату та перевірити, чи співпадають дані у сертифікаті з його підписом.

Тож які можливості нам дає сертифікат, і навіщо його використовувати? Сертифікат дає можливість перевірити, чи дійсно відкритий ключ, яким буде зашифровано повідомлення, належить стороні отримувача, а не є підробкою[4]. Для цього дані сертифікату перетворюються у хеш-суму за допомогою деякого відомого алгоритму хешування, після чого цифровий підпис сертифікату розшифровується відкритим ключем центру сертифікації, що гарантує правдивість даних сертифікату, та отримані дані порівнюються з хешем сертифікату. Якщо дані збігаються – то це значить, що інформація у сертифікаті співпадає з тою, що була у центру сертифікації, і відкритий ключ дійсно належить отримувачеві.

Сертифікати мають також галузь використання, що дає змогу виділити, який сертифікат і який ключ повинен бути використаний саме для вказаного випадку. Це дає додаткову захищеність, бо зіставивши тип ресурсу із типом сертифікату можна зрозуміти, чи правильно він використовується. Наприклад, якщо сертифікат має позначку про те, що він використовується у сфері торгівлі, а ресурс, на якому він використовується, позиціонується як банківський, то це повинно викликати тривогу та недовіру.

Формально механізм використання сертифікату можна пояснити наступним чином.

Нехай є дві сторони інформаційного обміну - A, B, що бажають обмінюватися повідомленнями конфіденційно, і третя сторона T (грає роль центра, що засвідчує правдивість даних, які сторони знають одне про одного), якій довіряють A і B.

Стороні A належить пара ключів (KA_o, KA_s), де KA_o - відкритий ключ, а KA_s - закритий (секретний) ключ боку A.

Стороні T належить пара ключів KT_o, KT_s).

А реєструється у Т (надсилає запит на підпис), вказуючи дані про себе і свій $КА_о$ Сторона Т за допомогою певних механізмів "засвідчує особистість" сторони А і видає стороні А сертифікат С, який встановлює відповідність між суб'єктом А і ключем $КА_о$.

Сертифікат С містить:

- ключ $КА_о$,
- ідентифікаційні дані суб'єкта А,
- ідентифікаційні дані, що засвідчує сторона Т,
- підпис сторони Т, який позначимо ST.

Підпис ST - це хеш (набір символів, хеш-сума / хеш-код), отриманий в результаті застосування хеш-функції до деталей сертифікату С, зашифрованого стороною Т з використанням свого закритого ключа $КТ_s$ і іншу інформацію.

А посилає стороні В свій сертифікат відкритого ключа С. В перевіряє цифровий підпис сертифікату ST, щоб переконатися, що він збирається використати відкритий ключ саме сторони А, а не підробку, яку йому надали зловмисники задля того, шоб розшифрувати та вкрасти його дані.

Для цього В:

- самостійно обчислює хеш від даних сертифіката С,
- розшифровує ЕЦП (електронний цифровий підпис) сертифіката ST за допомогою всім відомого $КТ_о$.
- отримавши інший хеш, перевіряє рівність цих двох хешів.

Якщо отримані хеші рівні – електронний цифровий підпис коректний, а це підтверджує, що $КА_о$ дійсно належить А.

Тепер В, знаючи відкритий ключ А і знаючи, що він належить саме А, може шифрувати цим відкритим ключем всі наступні повідомлення для А. І тільки А зможе їх розшифрувати, так як $КА_s$ відомий тільки А.

1.2 Виявлення проблем та актуалізація рішень

Нажаль зараз у відкритому доступі немає програмного забезпечення, що дозволило б зручно зберігати сертифікати, а також підтримувати їх цілісність, тому у більшості сертифікати зберігаються на звичайній файловій системі, є вразливими до відмови цієї файлової системи та від людського фактору, через який дані також можуть бути втрачені.

Частково цю проблему може вирішити використання хмарних сервісів, але вони не зручні для зберігання сертифікатів, та не дають змогу повернути файл з даними сертифікату в разі його видалення, пошкодження, або іншої події, після якої сертифікат буде неможливо використати.

Отже, потрібно порівняти різні типи зберігання сертифікатів.

1.2.1 Зберігання сертифікату на файловій системі

В цьому випадку недолік очевидний: жорсткий диск схильний до втрати своїх властивостей з часом, тому через декілька років може зруйнуватися, а дані не можна буде відновити. Також, файлова система вразлива до вірусів, що можуть викрасти дані, зруйнувати їх, тощо. Файлова система, а також пристрій зберігання, вразливі до фізичних утручань в їх роботу, тому перепад напруги, пожежа чи інші фактори також призведуть до втрати даних.

Ще один недолік – доступ до даних можна отримати тільки безпосередньо з комп’ютера, на якому вони зберігаються, що означає, що не можна таким чином отримати централізовану систему, яка б висилала сертифікати клієнтам.

Проте є і переваги. Якщо комп’ютер, що виконує роль сховища сертифікату, не має доступу до мережі інтернет, а тільки до внутрішніх ресурсів, а також

підтримується в гарному стані та дані регулярно дублюються, то таке сховище буде надійним, проте, все ще не зручним у використанні.

1.2.2 Зовнішні пристрої зберігання

Зовнішні пристрої зберігання частіше всього не мають рухомих частин, як жорсткі диски, тому не настільки зношуються із часом. Слід виділити оптичні диски, що майже не змінюються із часом, що дозволяє зберігати сертифікати протягом всього терміну дії не боючись того, що він не зможе бути прочитаний. Проте, оптичні диски можуть бути зруйновані подряпинами, зламані, загубленими з неакуратності. Як аналог можуть використовуватися flash-пристрої, які є більш компактними, але вони можуть деградувати з часом.

Перевагою такого методу зберігання є те, що у будь-який час можна зробити дублікат даних та віддати їх одному із розробників, для цього потрібен лише пристрій копіювання (дисковод для оптичних дисків, usb-порт, та інше).

Часто оптичні носії інформації використовуються як резервне сховище даних. Дані з серверів або інших носіїв переносяться на оптичний носій, що потім може бути схований до сейфу на роки, і гарантовано не втратить записану на нього інформацію. Таку резервну копію можна у будь-який час відновити. Це особливо актуально для інформації, яка не залежить від часу та є постійною.

Для зберігання сертифікатів таких спосіб резервування є сумнівним, бо сертифікат втрачає свою актуальність через деякий час, тому уся колекція резервних копій може стати непотрібною та неактуальною дуже скоро.

1.2.3 Хмарні сховища

Хмарне сховище є більш стійким до фізичних пошкоджень, бо інформація зберігається і дублюється на видаленому сервері. Також це вирішує ще й проблему того, що сертифікат може бути отриманий розробником із будь-якого місця. Але хмарні сховища часто перекодують інформацію або стискають її для того, щоб вона займала менше місця та швидше скачувалася користувачем. Нажаль, в разі сертифікату, для кого важливим є кожен байт, таке стискання неприпустиме. Також не всі хмарні сховища дозволяють зберігати будь-які типи файлів, що означає, що для зберігання сертифікатів список сервісів обмежений. Також особливістю таких сервісів є те, що неможливо прочитати інформацію із сертифікату, не завантаживши його на комп'ютер, бо формат цього файлу скоріше за все буде таким, що не підтримується переглядачем.

1.3 Аналіз способів зберігання інформації та способів їх використання

Окрім того, жоден із перелічених способів зберігання сертифікату не дає змогу використати його напряму із сховища, а потребує завантаження на комп'ютер розробника, потім на сервери, потім вбудовувати в програмне забезпечення, а потім заміняти його там у випадку, коли сертифікат дійде до дати закінчення дії.

Також жодне із цих рішень не дає можливості під'єднати систему до локальної інфраструктури приватних ключів (PKI), яка часто використовується в корпоративних мережах (найбільш розповсюджена зараз – це система Active Directory від Microsoft). Специфікації таких систем потребують виділених серверів для кількох рівнів сертифікації, сервер для відзиву ключів та систему зберігання. Нажаль, є ще один фактор – корпоративні політики – які роблять

неможливим використання сторонньої системи на підприємствах, що призводить до того, що адміністратори настроюють вище вказані сервери вручну для кожної мережі. Стандартна схема розгортки архітектури Active Directory приведена на рисунку 1.

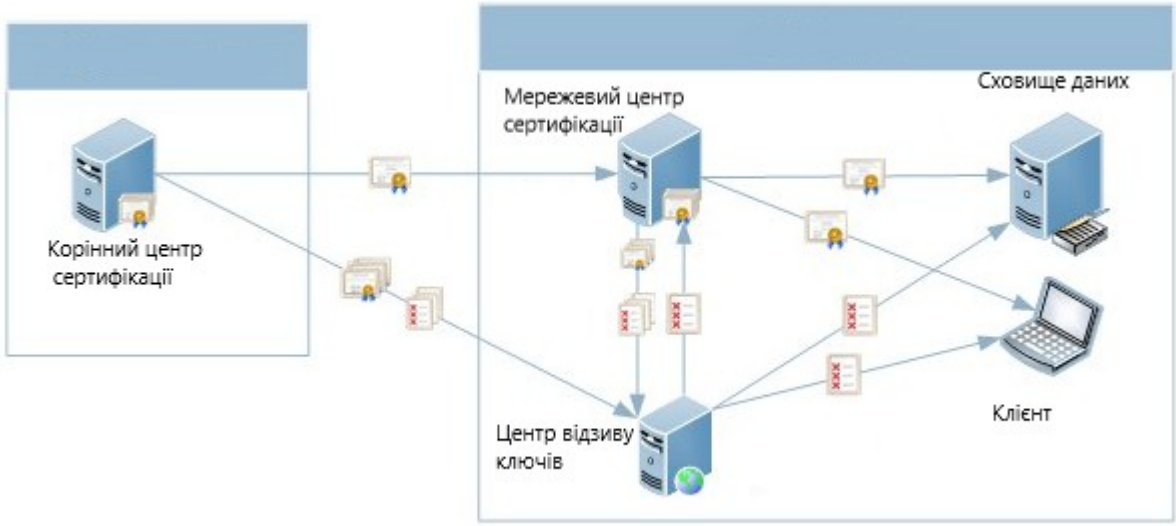

Рисунок 1 – Схема розгортки Active Directory

З рисунку видно, що для розгортання системи необхідно виділити принаймні 4 логічних сервери, аби підтримувати PKI. Також, система розраховує на те, що клієнт використовує сертифікат, що був отриманий з серверу, а значить, що в разі його відзиву користувач повинен замінити його сам.

Різноманітні варіації описаної інфраструктури відрізняються лише кількістю вказаних блоків, або перенесенням логіки цих блоків одне на одного (наприклад, сховище даних поєднується з центром відзиву, мережевий центр поєднаний з корінним, і так далі).

Проблемою є те, що такі системи не можуть витримати великого навантаження, саме через це їх запроваджують лише в внутрішніх корпоративних системах. Коли ж ми говоримо про тисячі сертифікатів та сотні тисяч користувачів, така система не зможе задовольняти їх потреби. Масштабування

таких систем не передбачене. Тому нова система, яка може прийти на заміну старій, повинна однаково легко розгортатися як для невеликої мережі, так і для високонавантажених мереж с великою кількістю користувачів.

1.4 Аналіз існуючої системи перевірки цілісності

В існуючих системах перевірка цілісності сертифікату відбувається лише за даними самого сертифікату. Як було вказано вище, поля сертифікату поєднуються в хеш-код, який порівнюється з хеш-кодом, що зашифровано у самому сертифікаті в вигляді цифрового підпису. Системи Active Directory, що запропоновані компанією Microsoft, перевіряють цілісність сертифікату саме таким чином. Але в даному випадку вся відповідальність лежить на вибраному алгоритмі шифрування. В даний момент, всі сертифікати для мереж шифрованого зв'язку (ssl, tls, тощо) використовують алгоритм SHA-256 для хешування та RSA-4096 для шифрування[2]. Нажаль, на разі зародження квантових обчислень ці методи можуть стати недостатніми.

Зламавши хеш та викликавши колізію можна підмінити дані сертифікату, що встановлено на комп'ютер користувача, після чого вся шифрована комунікація з використанням цього сертифікату стане відкритою для зловмисника[1]. Якщо ж сертифікат щоразу буде взятий з системи, яка знаходиться поза доступом зловмисника, сертифікат важко буде підмінити.

Але все ще можна зробити підробний сертифікат, зламавши ключ RSA-4096, та замінити такий сертифікат у системі, маючи доступ до бази даних. Ми не можемо замінити систему шифрування у самому сертифікаті, бо після цього його не можна буде використати[3]. Саме тому нова система, що буде запроваджена, повинна мати другий контур захисту, який не дозволить зробити несанкціонованих змін у базі даних, при цьому не перекладаючи навантаження на користувача. Для цього нова система буде мати інший, стійкий до квантових

обчислень алгоритм шифрування, яким буде повторно підписаний сертифікат. Відтепер, під час перевірки цілісності буде перевірятися не лише тіло сертифікату, а й додатковий підпис, який ніколи не буде доступним для користувачів поза системою та буде гарантувати відсутність стороннього втручання в систему.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

Перед початком роботи над програмною системою необхідно визначитися з вимогами до неї. У загальному випадку під вимогами розуміють сукупність властивостей, які повинна мати система, що реалізується. В ході роботи необхідно реалізувати систему, що дозволила б зберігати, використовувати, оновлювати сертифікати з централізованого серверу, забезпечуючи окрім основних функцій цілісність бази сертифікатів та кожного сертифікату окремо, а також можливість доступу з будь-якого пристрою.

2.1 Вимоги до програмного продукту

До програмного продукту існує ряд вимог, які необхідно дотримати при розробці. Так як вимоги до програмного продукту різняться за типом їх потрібно розглянути окремо.

2.1.1 Вимоги з безпеки використання

Як вже було сказано вище, сертифікат відкритого ключа не потребує захищеності його інформації від прочитання, а навіть навпаки, тому надавати доступ для читання можна будь-якому користувачу, якому потрібен даний сертифікат для перевірки його ключа. Однак сертифікат не може бути змінений стороннім користувачем, а тільки його власником.

Також важливим моментом є те, що кожен байт цифрового сертифікату має значення для сертифікату в цілому, бо з усіх даних (окрім даних цифрового

підпису) отримується хеш, що і складає цифровий підпис. Тому дані сертифікату повинні бути цілісними та не можуть бути спотворені, перекодовані, стиснуті, тощо. Окрім того, кожен користувач повинен мати змогу не тільки побачити сертифікат, а й перевірити його, звіривши вирахуваний хеш та дані підпису. Це означає, що кожен користувач повинен отримати одні і ті самі дані на кожен пристрій, незважаючи на операційну систему, тип платформи, тощо. Якщо простіше – під час доставки сертифікат не може бути перекодовано.

Ще один важливий фактор, зо було озвучено вище – це те, що сертифікат повинен бути захищений від випадкового видалення або спотворення даних в базі. Мається на увазі, що протягом деякого часу після видалення сертифікату користувачем повинна бути можливість повернути його без втрати даних.

Важливо, що система повинна функціонувати як у централізованому режимі (модель підписки, за якої користувачі системи будуть отримувати доступ до централізованої спільної системи), так і у приватному, за якого система буде розгорнута на локальних серверах клієнта (в такому разі можна передати у користування docker-image для серверу).

Для описаного в пункті 1.4 додаткового контуру захисту буде використано стійкий до крипто-атак з використанням квантових обчислень алгоритм NTRU-Prime, який, хоч все ще є молодим та мало вивченим алгоритмом, був офіційно затверджений для використання у сфері фінансів комітетом Accredited Standards Comitette X9[15].

2.1.2 Вимоги зі зручності користування

Як було сказано вище, система повинна давати можливість використати сертифікат напряму з неї, тобто давати можливість вбудувати її в будь-яке інше програмне забезпечення за необхідності. Також дані повинні коректно опрацьовуватися на будь-якій платформі.

Це, а також сказане про ідентичність даних у всіх користувачів, наводить на те, що система повинна працювати у режимі REST API, що дозволить надіслати запит на систему із будь-якого ресурсу та платформи. Для того, щоб інша програма могла коректно опрацьовувати дані, потрібно використати загальновідомий формат даних, який буде зручно використовувати будь-де.

У якості формату даних був вибраний формат JSON, бо він вирішує всі проблеми, описані вище. Так як фактично він є текстовим форматом – ми можемо задати його кодування, і цільове програмне забезпечення зможе коректно перевірити сертифікат. Окрім того, механізми роботи з цим форматом є на усіх платформах, а на тих, для яких немає готових рішень, можна зробити цей механізм власноруч без великих зусиль.

3 АРХІТЕКТУРА ТА ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. UML моделювання ПЗ

Моделювання програмного продукту було проведено з використанням діаграм UML. За допомогою цих діаграм було відображено роботу, функціонал та архітектуру системи (з доданням майбутнього функціоналу).

Діаграма варіантів використання (Use-Case) зображена на рисунку 2. Діаграма зображає можливі варіанти використання системи. Основними акторами системи є Власник сертифікату, Сторонній користувач системи та Механізм підтримки цілісності даних. Варіанти використання системи користувачами приведені на рисунку 2.

Опишемо варіанти використання користувачами:

Сторонній користувач системи може лише переглянути деякий сертифікат, на який у нього є посилання. Цей варіант розроблено для того, щоб розробник міг вбудувати посилання на сертифікат до свого програмного забезпечення, щоб користувач отримував його з серверу системи, а не з серверу цільового програмного забезпечення. Це допоможе замінювати сертифікати на всіх продуктах власника сертифікату в одну дію, просто змінивши сертифікат у системі.

Власник сертифікату може як отримати посилання на конкретний сертифікат (так само, як його отримує сторонній користувач), так і завантажити в систему новий сертифікат, виданий сторонньою організацією (центром сертифікації ключів). Окрім цього, власник може замінювати сертифікати, що дійшли кінця терміну експлуатації на нові, без заміни посилань на них.

Користувач, що має рівень доступу, може також додати новий корінний сертифікат, що може буде виданий локальним центрам сертифікації для видачі ними сертифікатів.

Локальні центри сертифікації також є акторами системи, та можуть брати сертифікат з корінного центру для генерації дочірніх сертифікатів або перевірки їх цілісності.

Також в системі буде присутній сервіс підтримання цілісності бази даних та фіксації змін DbHealthKeeper, але він не буде взаємодіяти з окремими мікросервісами, а лише з базами даних, тому його можна не включати в дану діаграму.

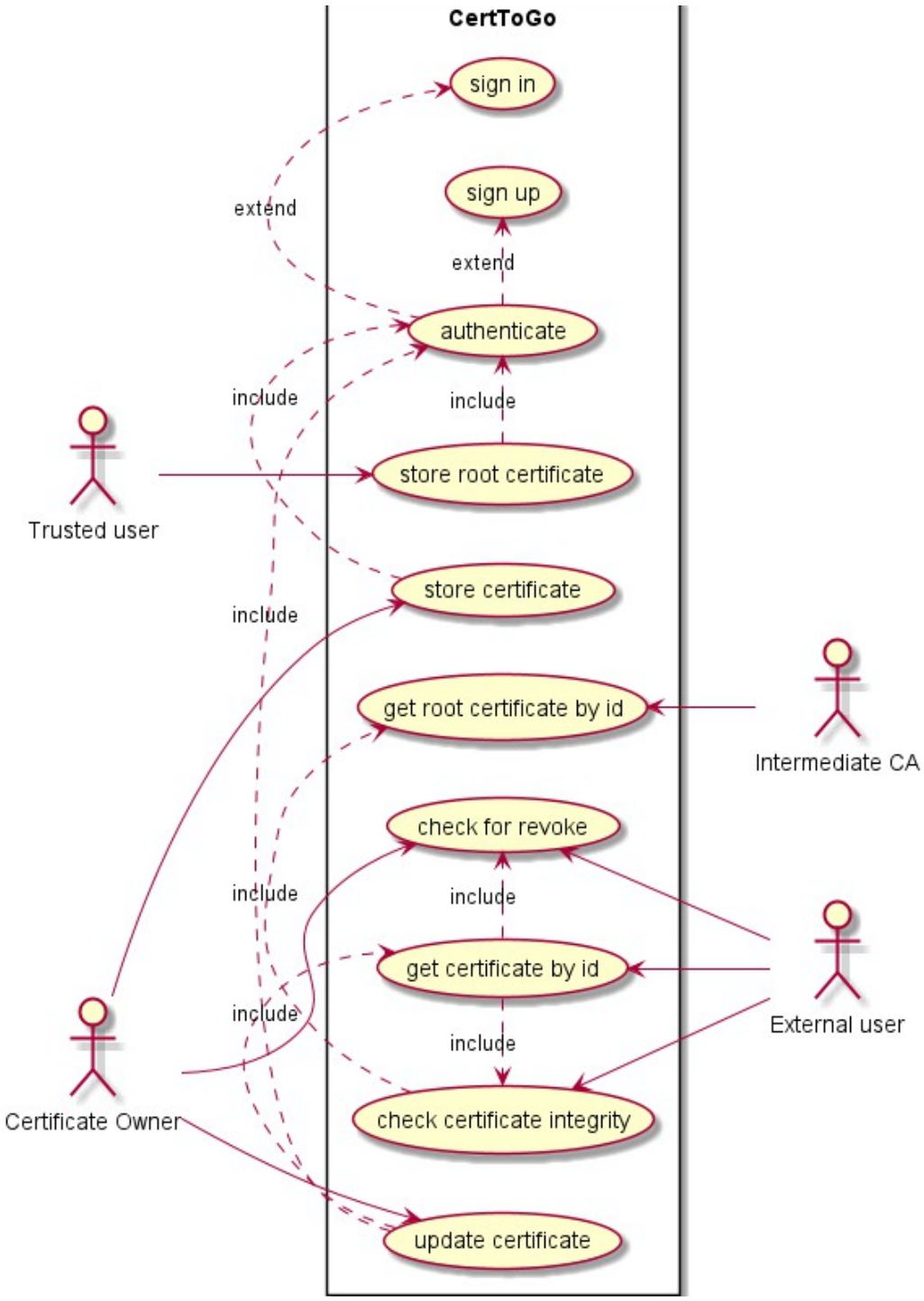

Рисунок 2 – Діаграма варіантів використання

3.2 Проектування архітектури ПЗ

Рисунок 3 зображує діаграму класів рівня API. Ця діаграма демонструє відношення декларативних елементів моделі[9]. Так як з боку API

функціональності небагато, то інтерфейс представлений лише двома класами-контролерами, що дозволяють взаємодіяти із системою. Це контролери Користувачів та Сертифікатів.

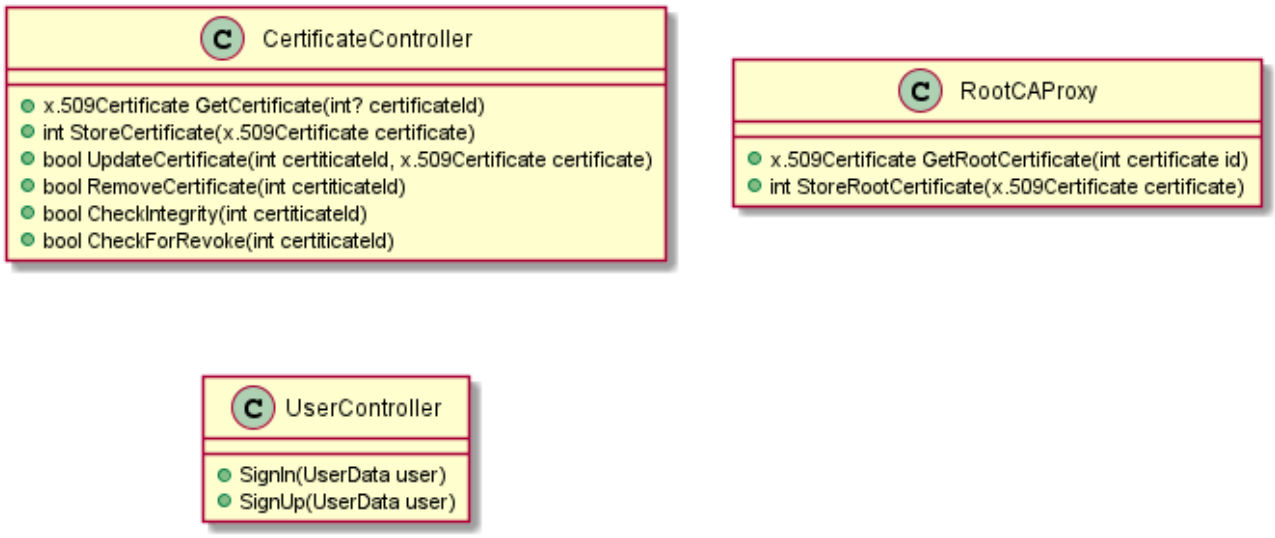

Рисунок 3 – Класи рівня API

Так як мікросервіси мають спільну логіку, вона була винесена в окремий проект, який розподілений на етапі компіляції між окремими сервісами. Це дозволило пере-використати написаний код для схожих запитів в базу даних та в інший мікросервіс, а також код перевірок цілісності. Також, це дозволило отримати єдиний інтерфейс для Root CA та Intermediate CA.

На рисунку 4 зображено діаграму компонентів. Вона відображає складові частини системи.

Як видно з діаграми, вся взаємодія між компонентами системи проходить, власне, через API. Доступ до бази даних проходить через сервер, а доступ до серверу проходить з будь-якого стороннього клієнту (на даному рисунку це веб-сайт, але клієнт може бути будь-чим).

Також видно, що система використовує два сховища даних: базу даних та файлову систему.

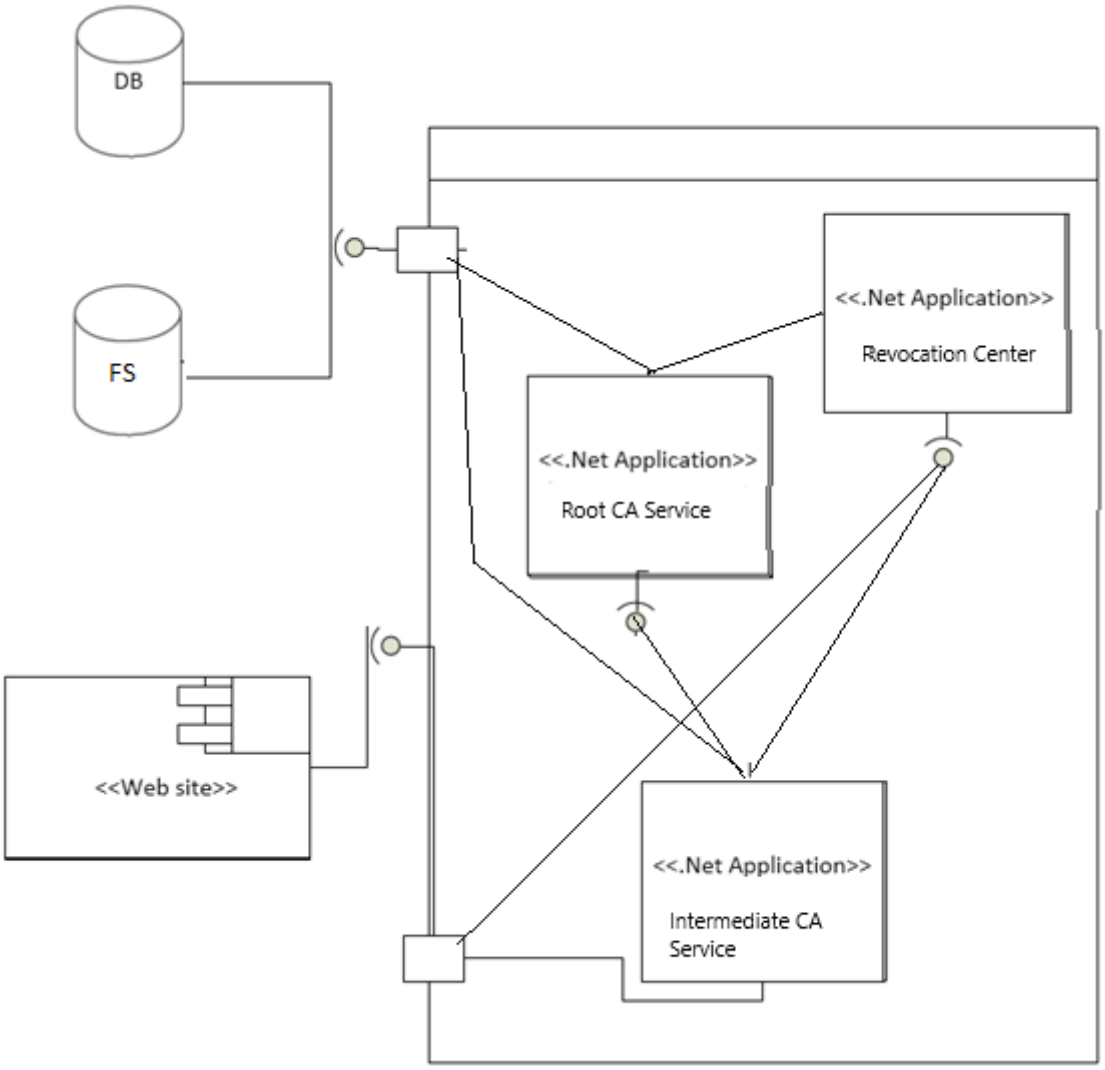

Рисунок 4 – Діаграма компонентів

Це зумовлено механізмом підтримки цілісності даних. База даних може дублюватися між різними версіями за необхідності, але резервні копії зберігаються окремо у вигляді файлів.

Це дасть змогу відновити дані з резервної копії у разі пошкодження даних, та навпаки, резервувати дані на вказану файлову систему у разі, коли це необхідно.

Тепер, маючи діаграму компонентів, ми можемо побудувати діаграму розгортання для завершальної фази проекту. Вона зображена на рисунку 5. Для системи обрано мікросервісну архітектуру, завдяки якій можна буде додати

стільки проміжних центрів сертифікації, скільки буде необхідно (корінний сервер видає сертифікати лише проміжним центрам, яких буде небагато, а от центри видають користувачам, і тому кількість одночасних запитів на них буде значно більшою). З діаграми видно, що корінний сервер не знає нічого про проміжні сервіси, тому її кількість може змінюватися «на льоту». Також кожен проміжний центр має власний сервіс відзиву сертифікатів. Окремий сервіс для підтримання бази даних створений для постійного моніторингу, створення резервних копій даних та відновлення даних з копії. Зважаючи на невеликий об‘єм даних, база даних може бути одна на всю систему. Кожен сервіс на діаграмі представляє собою додаток, написаний з використанням технології DotNetCore, та може буте розгорнутий як на різних серверах, так і на одному засобами віртуалізації (для тестування системи був використаний підхід контейнеризації через Docker)

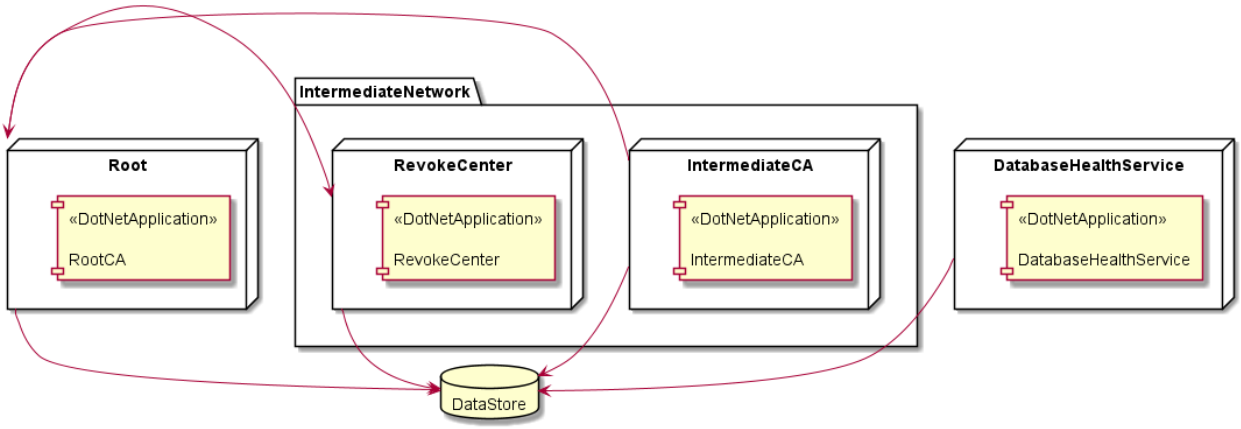

Рисунок 5 – Діаграма розгортки

Загальна взаємодія між об'єктами добре демонструється на прикладі діаграми послідовностей. На рисунку 6 зображено приклад даної діаграми для типової задачі системи – отримання сертифікату з системи.

З діаграми послідовностей можна побачити одразу декілька особливостей системи.

Перша – це те, що кожен із модулів відповідає тільки за один тип дій, як і повинно бути в об’єктно орієнтованому програмуванні за правилами SOLID (конкретно – правило єдиної відповідальності).

Друга – сертифікат перед відправкою клієнту проходить декілька етапів перевірок: перевірка на те, чи не був сертифікат відізваний, та на те, чи не був він пошкоджений. В будь-якому з цих випадків хазяїн сертифікату буде інформований про те, що сертифікат необхідно замінити.

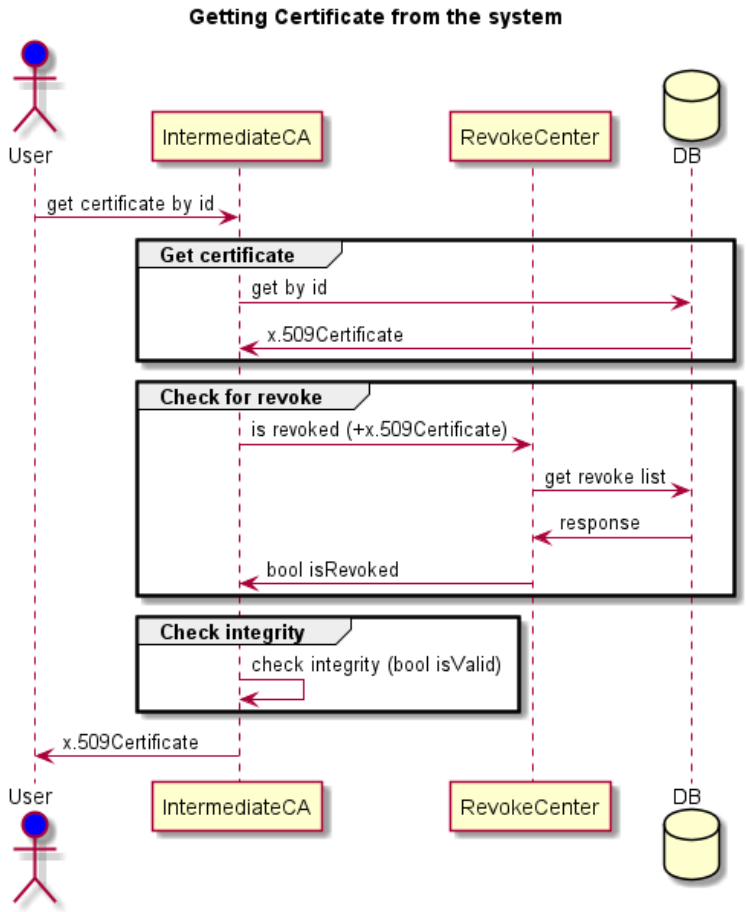

Рисунок 6 – Діаграма послідовності для отримання сертифікату

3.3 Проектування бази даних

Беручи до уваги те, що система повинна в цілому бути просто сховищем та надавати можливість використання сертифікату, можна виділити лише три

ключові сутності у базі даних: це будуть Користувач, Роль та Сертифікат. Відношення сутностей показані на рисунку 7. Розберемо значення полів у сутностей бази. Почнемо з Користувача.

Id – унікальний ідентифікатор користувача у системі[6]. Це поле є цілком внутрішнім по відношенню до системи, користувач ніколи не утримує свого ідентифікатора та він не має сенсу за межами системи.

Username – ім’я користувача, яке він вказав при реєстрації. Завдяки цьому полю система буде пропонувати ім’я власника сертифікату в нових згенерованих сертифікатах.

RoleId – id ролі, яка призначена користувачу. На даний момент проектується 2 ролі: користувач (може керувати сертифікатами проміжних центрів) та адміністратор (може керувати сертифікатами корінного центру).

PasswordHash – хеш-сума паролю, що користувач ввів при реєстрації. Сам пароль не зберігається в системі, його неможливо вкрасти. Кожного разу, коли користувач вводить пароль, система сама бере хеш-функцію від нього та порівнює отримані дані. Якщо був уведений вірний пароль – то хеш буде збігатися.

Email – служить для ідентифікації користувача за межами системи. Також, на цю адресу буде вислано нотифікацію, якщо будь-який з сертифікатів користувача було відізвано.

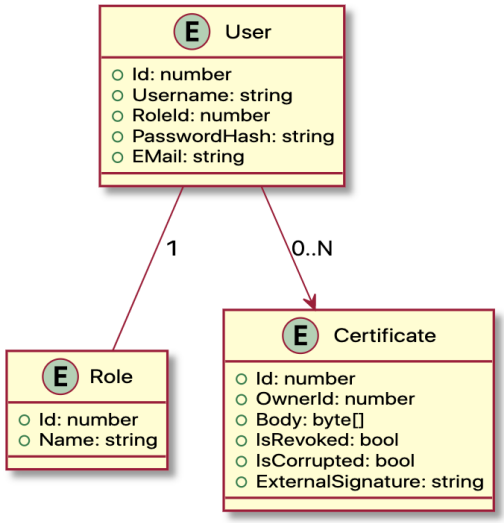

Рисунок 7 – Відношення сутностей в базі даних

Для сертифікату передбачені наступні поля:

- Id - унікальний ідентифікатор сертифікату у системі. За цим номером користувач може отримувати сертифікат.
- OwnerId – Ідентифікатор власника сертифікату
- Body – побайтове представлення самого сертифікату x.509
- IsRevoked – флаг, що показує, чи був сертифікат відізваний.
- IsCorrupted – флаг, що показує, чи був сертифікат модифікований.
- ExternalSignature – поле, по якому буде проводитися перевірка сертифікату всередині системи на предмет підміни. В ньому лежить хеш, закодований шифром NTRU-Prime.

Зв‘язок між користувачем та сертифікатом має тип «багато до одного», тобто один користувач може мати багато сертифікатів. Реалізовано це завдяки тому, що сертифікат зберігає id власника.

4 ОПИС ПРИЙНЯТИХ ПРОГРАМНИХ РІШЕНЬ

4.1 Серверна частина

Для розробки системи було використано фреймворк ASP.NET CORE. Більша частина додатку працює на протоколі REST.

Технологія Web API дозволяє будувати REST-подібні служби, які працюють у фоновому режимі та являють собою точки доступу до інформації для будь-яких клієнтів будь-то мобільний додаток, веб-клієнт, десктоп, чи навіть IoT-рішення[8]. Це доволі зручний підхід, так як забезпечує універсальність серверної частини системи. Один із Web API контролерів представлений на рисунку 8.

```
[HttpGet]
public IEnumerable<CertificateShortInfo> GetAllCertificates()
{
    return certificateService.GetAllCertificates(User.Identity.GetUserId()).Select(ToClientDto);
}

[HttpGet]
public CertificateDto GetCertificate(int id)
{
    return certificateService.GetCertificateById(id);
}

[HttpGet]
[Route("Delete")]
public bool DeleteCertificate(int id)
{
    var certificate = certificateService.GetCertificateById(id);
    if (certificate == null)
    {
        return true;
    }

    if (certificateService.IsCertificateBelongsToUser(User.Identity.GetUserId(), id))
    {
        return certificateService.RemoveCertificateById(id);
    }

    return false;
}
```

Рисунок 8 – Приклад WebApi контроллеру

Web API дозволяє обмінюватися інформацією за допомогою протоколів HTTP та більш сучасного HTTPS. При необхідності обслуговування IoT-рішення зазвичай використовують технологію WCF, що підтримує також інші протоколи передачі даних, але в даному випадку це призвело б до невиправданого ускладнення системи, так як вона потребує лише веб-клієнту.

REST (Representational State Transfer, «передача репрезентативного стану») – підхід до архітектури мережевих протоколів, який забезпечує доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP [11]. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0 [11].

Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами. При підході REST кількість методів і складність протоколу суворо обмежені, що призводить до того, що кількість окремих ресурсів має бути великою.

Наведемо декілька прикладів доступних методів для API:

– GET /rest/Certificates – повертає список всіх сертифікатів, які доступні користувачеві (додані ним);

– GET /rest/Certificates/certificate_Id – повертає файл сертифікату форматі X.509 certificate;

– DELETE /rest/Certificates/certificate_Id – видаляє сертифікат та всі його дочірні сертифікати з системи за поданим ідентифікатором;

– PUT /rest/Certificates/certificate_Id – замінює сертифікат тим, що буде відправлено в тілі запиту;

– POST /rest/Certificates – додає сертифікат до системи для даного користувача.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). В даному випадку використаний формат JSON, так як з ним гарно взаємодіє як Knockout.js фреймворк, так і Javascript загалом.

Загалом у веб-розробці зазвичай використовується формати XML або JSON, але останнім часом більша перевага надається JSON так як він є більш гнучким та зручним у використанні.

Задля того, щоб систему можна було масштабувати, зони відповідальності системи поділено між декількома мікросервісами, що можуть бути розгорнуті як на одному сервері, так і на декількох, завдяки чому можна буде підлаштувати систему під будь-яке навантаження. Усього спроектовано чотири основні типи сервісів:

Root CA – сервіс, що працює лише з корінними сертифікатами. Так як потреба в корінному сертифікаті буде значно меншою, ніж в проміжних сертифікатах, а також операції запису таких сертифікатів також виконуються рідше – цей сервіс буде мати лише одну розгорнуту копію.

Revocation Center – цей сервіс виконує перевірку сертифікату на відзив. В ньому є список сертифікатів, які були відізвані або скомпрометовані. Цей список постійно поповнюється, тому кожен раз, коли користувач отримує сертифікат – той може бути перевірено на відзив як його самого, так і його корінного сертифіката. Цей сервіс буде мати лише одну копію, бо навантаження на нього буде також невелике (перевірка на входження ключа в таблицю має обчислювальну складність O(1), тому максимальне навантаження буде лише тоді, коли потрібно буде виконати оновлення бази відзиву, а це швидкий процес). Однак за потреби даний сервіс може бути розгорнутий і в декількох копіях. В такому разі, посилання на розгорнуті сервіси потрібно буде розподілити між копіями Intermediate CA. Так як усі посилання будуть передаватися лише через конфігураційні файли, то таку процедуру можна буде автоматизувати за допомогою, наприклад, Kubernetes або інших подібних систем.

Intermediate CA – цей сервіс працює з проміжними сертифікатами. Він буде видавати проміжний сертифікат за посиланням, через нього система зможе виконувати запис нових проміжних сертифікатів, перевірку їх цілісності, перевірку наявності корінного сертифікату. Таких сервісів може бути безліч в залежності від навантаження системи. Кожен із них має єдине посилання на сервіс Root CA та Revocation Center. Під час отримання сертифікату сервіс одразу перевірить його цілісність через корінний сертифікат, а також перевірить, чи не було цей сертифікат відізвано.

Основна функціональність зібрана в класах CertificateService та DbHealthKeeperService.

CertificateService призначений для обробки інформації щодо сертифікату, запиту інформації з репозиторіїв та конвертації в формат представлення. Цей класс присутній в усіх варіантах центру (як в корінному, так і в проміжному центрі, а також в центрі відзиву).

DbHealthKeeperService призначений для резервного копіювання даних та підтримання цілісності бази даних сертифікатів. Механізм дії сервісу наступний.

Кожні 24 години сервіс робить запит по усім сертифікатам, що є в базі, на предмет того, чи змінився сертифікат з моменту останньої перевірки. Для цього він вираховує хеш-суму полів сертифікату, виконує розшифровку поля ExternalSignature для цього сертифікату, та порівнює отримані хеші.

Якщо хеш зпівпав – то дані не змінилися, і можна брати наступний сертифікат. Якщо ж вони не зпівпали – то можливі дві ситуації:

По-перше – користувач міг замінити сертифікат власноруч, і сервіс повинен зробити його резервну копію (для кожного сертифікату існує нескінченна кількість копій, що дозволяє зберігати не тільки теперішню, а й попередню версію сертифікату на випадок якщо вона знадобиться). В такому разі користувач, що завантажив сертифікат, буде проінформований про зміну, та повинен буде її підтвердити. Після цього буде зроблено новій хеш.

По-друге – в базу даних потрапили помилкові дані, була зроблена кібератака, був збій в роботі бази, тощо.

DbHealthKeeperService розгорнуто на окремому мікросервісі, який взаємодіє лише з базою даних, та відокремлений від бізнес-логіки.

Рівень доступу до даних використовує ORM EF Core, що забезпечує високу швидкість та високий рівень захищеності системи від SQL-ін’єкцій [12]. Зазвичай ORM працюють трохи повільніше складених вручну SQL-запитів[7], але в данному випадку Entity Framework цілком задовольняє потреби системи.

Також на рівні доступу до даних використовуються патерни «Репозиторій» та «Unit of Work» для підвищення рівня гнучкості системи[13].

4.2 Бази даних

Задля ізоляції зон відповідальності одне від одної, кожен з сервісів використовує свою, окрему базу даних. Такий підхід дозволить масштабувати навантаження на бази та виділяти їх в окремі сервери.

База даних Root CA Data зберігає лише дані про корінні сертифікати.

База даних Intermediate CA Data зберігає дані про всі проміжні сертифікати. У разі, коли буде розгорнуто більше одного Intermediate CA, є дві опції для цієї бази:

- Буде використовуватися одна, спільна база. Таке розділення не потребує фіксації користувачів на окремому сервісі. Це значить, що до якого б серверу Intermediate CA не звернувся користувач – він завжди зможе отримати актуальні дані про сертифікат.
- Буде використовуватися декілька баз. В такому разі для підтримання роботи системи необхідно зробити систему пошуку по всіх базах, та перенаправляти користувача до того серверу, в базі якого є дані про сертифікат. На даний момент така функціональність не підтримується, але може бути запропонована користувачам в вигляді опції для подальших оновлень системи.

База даних RevocationCenterData зберігає дані про відізвані сертифікати. В ній рідко відбувається процес запису, а обсяг даних не росте з кількістю серверів, тому вона може бути розгорнута в одному екземплярі.

База даних IdentityData зберігає дані про користувачів, що були зареєстровані в системі. Ця база має один екземпляр, до якого звертаються все розгорнуті сервери з Root CA та Intermediate CA. За такої умови, на який би сервер не потрапив користувач при реєстрації, він може здійснити вхід до будь-якого іншого сервера (може бути актуально як з різними Intermediate CA, так і коли користувач має права на доступ до баз Root CA та Intermediate CA одночасно).

4.3 Клієнтська частина

Хоч основна робота з системою виконується у вигляді REST – запитів, клієнтський інтерфейс також присутній. Завдяки цьому інтерфейсу можна зареєструватись у системі, внести новий сертифікат через просту форму завантаження (форма представлена на рисунку 9), переглянути сертифікати що були завантажені у систему користувачем, видалити їх або замінити.

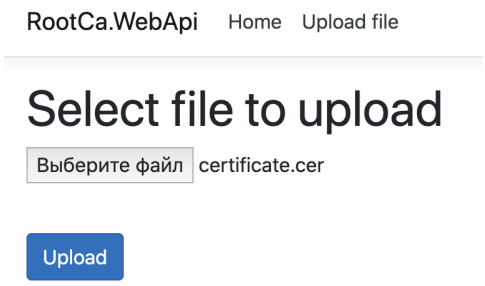

Рисунок 9 – Форма завантаження сертифікату

Рисунок 10 зображує форму, що виводить сертифікати користувача, а також дозволяє їх видалити чи замінити. Такий інтерфейс присутній при зверненні до Root CA або Intermediate CA.

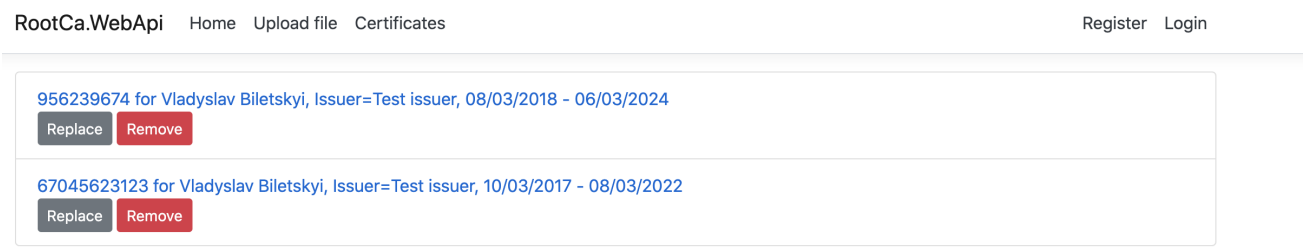

Рисунок 10 – Форма перегляду сертифікатів користувача

Натискання на будь-який з сертифікатів відкриє вікно перегляду сертифікату, де окрім присутніх даних також є публічний ключ сертифікату, та присутня можливість скачати даний сертифікат у форматі *.cer. Форма зображена на рисунку 11.

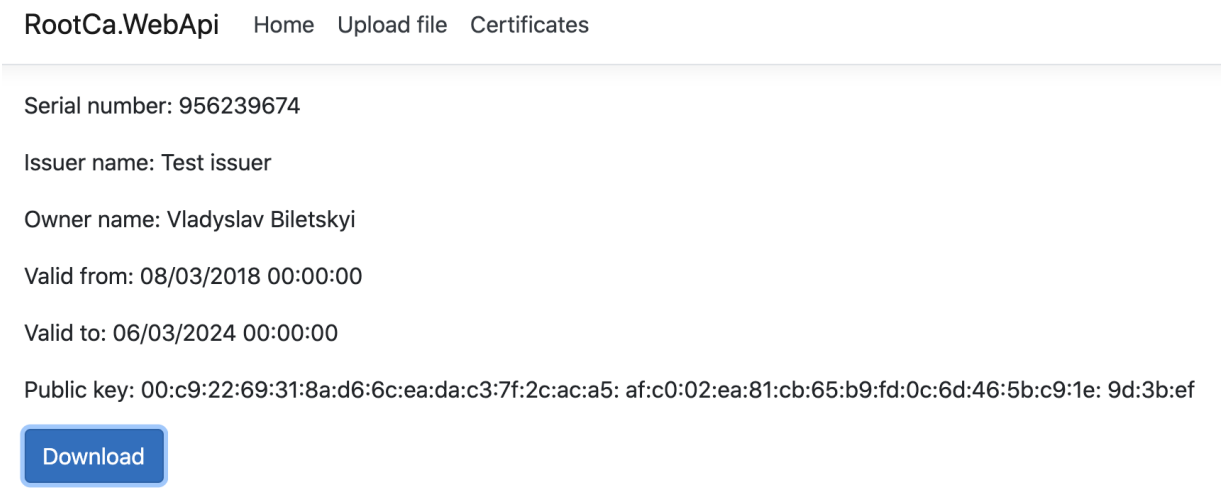

Рисунок 11 – Форма перегляду сертифікату

Слід зазначити, що дана кнопка використовує те ж посилання, що буде використовувати користувач в REST – запиті. Це значить, що її можна скопіювати та використовувати напряму. Завантаження сертифікату не потребує авторизації.

Revocation Center має лише одну функцію – перевірки на відзив. В нього є форма завантаження сертифікату, який буде перевірено одразу після завантаження. Після перевірки система сповістить, чи був відізваний даний сертифікат або його корінний сертифікат. Результат перевірки представлений на рисунку 12.

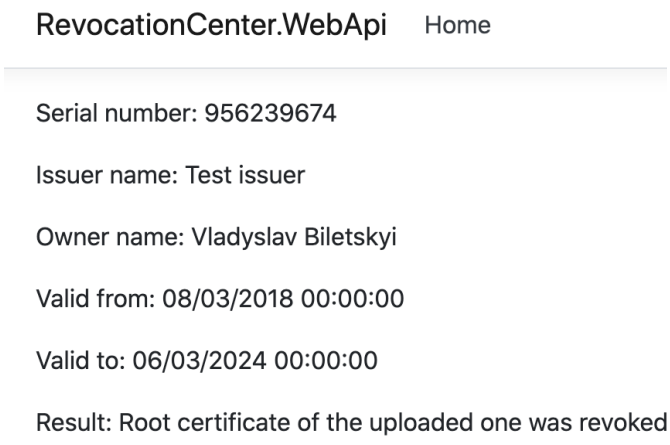

Рисунок 12 – Результат перевірки сертифікату.

Слід зазначити, що даний результат буде отримано лише через використання інтерфейсу користувача. В разі використання протоколу REST відповіддю буде структура з полями, що вказують, чи пройшов сертифікат перевірку (логічне значення true або false), та повідомлення про причини відхилення (коли сам сертифікат або його корінний сертифікат відізвано).

Кожна з вказаних дій може бути виконана за допомогою REST – запиту, а інтерфейс лише дублює функціональність запитів, дозволяючи користувачам тестувати свою інтеграцію. Також, інтерфейс може бути вимкнений за необхідністю за допомогою конфігураційного файлу.

DbHealthKeeper не може бути доступним з мережі, тому він не має свого інтерфейсу. Проте він має два відкритих методи, за якими сервіси можуть отримати дату останньої перевірки баз даних та примусово запустити цю

перевірку. На даний момент дата останньої перевірки відображається в інтерфейсі Root CA.

5 ТЕСТУВАННЯ РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Процес забезпечення якості програмного забезпечення проникає в усі стадії розробки інформаційних систем. Основними завданнями цього процесу є визначення методики забезпечення якості створюваного продукту, вбудовування процесу тестування в загальний процес розробки програмного забезпечення, а також вибір інструментальних засобів підтримки бізнес-процесів, пов'язаних із забезпеченням якості ПЗ. Тестування програмного забезпечення – це процес пошуку дефектів в системі, які відбуваються через помилки в програмі, які можуть привести до виходу з ладу результуючого продукту. Тестування програмного забезпечення має різні цілі і завдання, які включають в себе:

- виявлення дефектів;
- набуття впевненості і надання інформації про рівень якості;
- запобігання дефектів.

Під час розробки програмної системи були написані модульні та інтеграційні тести, які підтверджують якість кінцевого продукту. Модульне тестування (юніт-тестування) – це тестування мінімально можливого компоненту, наприклад, окремого класу або функції. Юніт-тестування підтверджує коректність роботи алгоритму, адже функція чи клас тестуються в ізоляції[14]. Під час інтеграційного тестування тестуються інтерфейси між компонентами та підсистемами, тобто взаємодія між класами, архітектурними рівнями системи.

Також було проведене конфігураційне та навантажувальне тестування програмної системи. Конфігураційне тестування проводиться для того, щоб забезпечити оптимальну роботу програми в різних браузерах з урахуванням їх версії, пропорцій екрану. Слід зауважити, що конфігураційне тестування було проведено як через WEB-інтерфейс, що посилав запити (макетний сайт), так і через REST-запити.

Навантажувальне тестування – це автоматизовані випробування програмної системи, що імітують різні навантажувальні моделі, з метою комплексної оцінки

продуктивності, перевірки якісної і безперебійної роботи системи, а також перевірки відповідності вимогам безпеки та ефективності. Навантажувальне тестування допомагає з'ясувати граничні умови навантаження на систему, при яких вона продовжує працювати стабільно і з прийнятним часом відгуку, а також оцінити здатність системи правильно функціонувати в разі перевищення планованих навантажень. Для виконання навантажувального тестування було використано додаток JMeter, що емулював 300 одночасних запитів від різних користувачів.

Під час навантажувального тестування серйозних недоліків виявлено не було, що свідчить про стабільність роботи розробленої веб-системи та можливість її використання багатьма користувачами одночасно.

Також для тестування функціональності системи було застосовано такий підхід, як димове тестування.

Smoke Testing (димове тестування) у тестуванні програмного забезпечення означає мінімальний набір тестів на явні помилки. Цей тест зазвичай виконується самим програмістом. Програму, що не пройшла такий тест, немає сенсу передавати на глибше тестування.

Перше своє застосування цей термін отримав у пічників, котрі збудувавши піч, закривали всі отвори, починали топити її й переконувалися, що дим іде виключно з передбачених місць. Повторне «народження» терміну відбулося в радіоелектроніці. Після переробки чи ремонту електронного пристрою його просто вмикали на короткий час. Якщо після цього із пристрою не йшов дим, вважалося, що тест пройдено.

Для більш складних корпоративних систем цього виду тестування зазвичай недостатньо і потрібне покриття функціоналу автотестами або написання тест кейсів та проводження ручного тестування. Обидва підходи потребують набагато більших витрат часу кваліфікованих кадрів. Димове тестування є достатнім для не складних систем та може доволі швидко проводитися розробниками, навіть без залучення тестувальників.

У процесі димового тестування зазвичай створюється мінімальний набір тест-кейсів, що називається Happy Path. Якщо всі тест-кейси у такому сценарії проходять з позитивним результатом, то димове тестування вважається завершеним позитивно, якщо хоча б один сценарій працює некоректно – функціонал потребує доопрацювання.

Загалом димове тестування було проведено з таким функціоналом системи, як: авторизація, реєстрація, додання нового уроку, планування та завершення уроку, додання композиції, планування та завершення композиції, додання пісні, планування та завершення пісні. Недоліків під час проведення димового тестування виявлено не було, що свідчить про високу якість результуючої системи та стабільність її функціоналу.

ВИСНОВКИ

В результаті виконання атестаційної роботи було досліджено сучасні архітектури центрів сертифікації ключів, їх переваги і недоліки. Досліджено алгоритми для забезпечення цілісності сертифікатів, які забезпечують сучасні засоби цифрового підпису. Було створено систему контролю та зберігання сертифікатів «CertOnTheGo» за допомогою технологій ASP.NET CORE, ASP.NET Web API, та мов програмування C# та JavaScript.

Результатом проекту є система, що складається чотирьох окремих мікросервісів, яка в перспективі може бути розширена за рахунок вбудови системи у клієнтські мережі користувачів системи.

Система дозволяє зберігати сертифікати користувача, гарантуючи їх цілісність, а також дозволяє вбудовувати посилання на сертифікат у інші продукти. Крім того, «CertOnTheGo» перевіряє кожен сертиіфкат на цілісність та виключати несанкціоновані зміни за допомогою посиленого з використанням стійкого до атак з використанням квантових обчислень алгоритму NTRU-Prime, слідкує за роботою бази, та може нотифікувати про відзив сертифікатів.

Для реалізації всього необхідного функціоналу була проаналізована предметна область, виявлені взаємовідносини між основними об'єктами системи. При аналізі предметної області проводились консультації з представниками предметної області (розробниками програмного забезпечення, що використовують сертифікати з відкритим ключем, працівниками сервіс-центру IT-компаній, які займалися розгорткою Microsoft Active Directory та подібних систем в локальних мережах компаній), після чого дані опитів були систематизовані та формалізовані. Для зберігання інформації була спроектована база даних, яка відображає всі зв'язки між сутностями даної предметної області. Система здатна задовольнити потреби розробників ПЗ, що є клієнтами системи, та в перспективі може бути впроваджена у використання в якості тестової, а згодом і основної системи зберігання сертифікатів у компаніях розробки програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. I.D. Gorbenko, O.G. Kachko, M.V. Yesina. Analysis of asymmetric NTRU Prime. IIT Ukraine encryption algorithm with regards to known attacks. Telecommunications and Radio Engineering, 2018, p 799-816.
2. Горбенко І., Кузнецов О., Потій О., Горбенко Ю., Качко О. Сутність. Проблеми та попередні результати міжнародного конкурсу зі створення стандартів асиметричних постквантових криптографічних перетворень. Матеріали конференції, с. 19 - 22.
3. Gorbenko I. ELECTRONIC SIGNATURE MECHANISMS. The Current State, the Existing Contradictions and Prospects of Practical Use for the Post-Quantum Period I. Gorbenko, A. Kuznetsov, Yu. Gorbenko, S. Kavun, O. Kachko, M. Yesina. ASC Academic Publishing Minden, Nevada, USA, 2017 – 165 p.
4. Горбенко І.Д., Качко О.Г., Єсіна М.В., Пономар В.А. Стан та проблемні питання розроблення та впровадження перспективного стандарту цифрового підпису. «КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ В НАУКОЄМНИХ ТЕХНОЛОГІЯХ» (КМНТ-2020), Харків-2020.
5. Нільс Фергюсон, Брюс Шнайер. Практична криптографія: розробка та впровадження захищених криптографічних систем. - М. : Діалектика, 2004. - 432 с.
6. Кренке Д. Теорія і практика побудови баз даних. 8-е вид. - СПб .: Пітер, 2003. - 800 с.
7. Querying Microsoft SQL Server 2012 – Exam 70-461 Training kit / I.Ben-Gan, D. Sarka, R.Talmage – Microsoft, 2012. – 720 p.
8. Microsoft ASP.NET 4 с примерами на C# 2010 для профессионалов. – 4-е изд. [Текст]: справочник / М. Мак-Дональд, А. Фримен, М. Шпушта – М.: ООО "И.Д. Вильямс", 2011. — 1424 с.
9. Буч, Г. Язык UML / Г. Буч – М.: ДМК Пресс, 2006. – 248 с.

10. Троелсен Е. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание / Е. Троелсен – М.: «Вильямс», 2013. – 1312 с.

11. REST API Tutorial [Електронний ресурс] / REST API Tutorial. – Режим доступу: https://www.restapitutorial.com/ – Назва з екрана.

12. M. Biehl Restful Api Design / M. Biehl – CreateSpace Independent Publishing Platform, 2016. – 294 p.

13. Lerman J. Programming Entity Framework / J. Lerman – 2nd Edition. – O'Reilly, 2010. – 920 p.

14. Савин, Р. Тестирование Дот Ком / Р. Савин – М.: Дело, 2007. – 312 с.

15. Business wire. Security innovation. NTRUEncrypt adopted X9 Standard Data. URL: www.businesswire.com/news/home/20110411005309/en/Security-Innovation's-NTRUEncrypt-Adopted-X9-Standard-Data (дата звернення: 01.05.2020)