Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Факультет інформатики
Кафедра інформатики

КРИПТОГРАФІЧНІ АЛГОРИТМИ В МОБІЛЬНИХ ДОДАТКАХ ПІД УПРАВЛІННЯМ IOS

Текстова частина до курсової роботи
за спеціальністю 122 «Комп'ютерні науки»

Керівник курсової роботи
к. ф-м. н., доцент
Нагірна А. М.

_____
(підпис)
"____"_____ 2022 р.

Виконав студент МП КН-1
Сабадишин М.О.

_____
(підпис)

"____" _____ 2022 р.

Київ-2022

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав. Кафедри інформатики,
доцент, к.ф-м.н.
Гороховський С.С.
_____
(підпис)
“____” _____ 2022 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

Студента Сабадишина Максима Олександровича факультету інформатики 1 курсу

ТЕМА криптографічні алгоритми в мобільних додатках під управлінням iOS

Зміст ТЧ до курсової роботи:
Календарний план
Вступ
Аналіз предметної області та дослідження її актуальності
Теоретичні відомості щодо криптографії та використовуваних алгоритмів
Практика використання та реалізації алгоритмів
Висновки по роботі
Список джерел

Дата видачі “____” _____ 2022 р. Керівник _____
(підпис)

Календарний план виконання роботи:

Тема: криптографічні алгоритми в додатках під управлінням iOS

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової	01.11.2021	
2.	Ознайомлення з предметною областю	05.01.2022	
3.	Пошук корисних ресурсів	10.01.2022	
4.	Ознайомлення зі знайденими ресурсами	20.01.2022	
5.	Опис актуальності предметної області	31.01.2022	
6.	Аналіз проблем криптографії в мобільних застосунках	05.02.2022	
7.	Ознайомлення з основними концептами криптографії	20.02.2022	
8.	Опис загальних концептів криптографії	15.04.2022	
9.	Вивчення та опис актуальних криптографічних алгоритмів в мобільній розробці	30.04.2022	
10.	Пошук існуючих підходів в індустрії	05.05.2022	
11.	Аналіз бібліотек по роботі з криптографією на iOS	20.05.2022	
12.	Написання програмного додатку з прикладом використання бібліотек та аналізом швидкодії	30.05.2022	
13.	Оформлення роботи	07.06.2022	

Студент Сабадишин М.О.

Керівник Нагірна А.М.

“____” _____

ЗМІСТ

Анотація

Метою цієї курсової роботи є аналіз криптографії в рамках індустрії розробки мобільних додатків, аналіз актуальних криптографічних алгоритмів та їх застосування у сфері, аналіз фреймворків, що надають функціонал криптографічних алгоритмів.

Проведено огляд криптографії як науки загалом, описано її основні концепти, причини актуальності.

Проведено дослідження щодо проблем, які криптографія вирішує на мобільних застосунках, оглянуто ті алгоритми, які використовуються при розробці мобільних додатків.

Досліджено існуючі додатки та описані сценарії використання криптографічних алгоритмів у них, проаналізовано існуючі фреймворки, що надають функціонал криптографічних алгоритмів, оглянуто їх прикладне використання.

Ключові слова: криптографія, криптографічні алгоритми, мобільний додаток, iOS.

Вступ

Актуальність теми та її практичне значення

Мобільні пристрої в останні роки мають стабільний ріст в кількості активованих девайсів, з показником в 6.2 мільярди в минулому році та прогнозованим ростом ще на 300 мільйонів в поточному [1]. Для розуміння росту менш, ніж половина населення Землі в 2016 році мала у власності смартфон. Проте за цей час рівень поширення смартфонів зростав, досягши 78.05 відсотків в 2020 році та все ще тримаючи тенденцію на зростання [2].

Незважаючи на такий результат все ще з упевненістю можна бачити перспективи подальшого збільшення ринку, адже в таких багатонаселених країнах як Індія та Китай спостерігається порівняно низький показник володіння мобільним пристроєм.

Також варто відмітити і фінансовий ріст на глобальному ринку протягом останніх років, який в основному спричинений збільшеною середньою ціною продажу смартфону [3].

Фактори збільшення кількості користувачів, а також фінансовий ріст ринку вказують на те, що мобільні пристрої є вигідною нішею для роботи. При цьому також зростає кількість нових програмних застосунків в магазинах додатків [4].

Така популярність мобільних пристроїв та велика кількість їх можливостей як в плані заліза так і в плані доступності створили тренд на перенесення багатьох важливих сервісів у простір смартфонів. Переважна більшість людей має мобільний пристрій завжди при собі і саме з ним стає зручніше виконувати різні операції чи мати доступ до даних.

Сьогодні стало обов’язком представникам банківської сфери мати свої мобільні додатки: Україна є лідером в діджиталізації сфери державних послуг розробивши та підтримуючи «Дію» [5]. Багато бізнес процесів протікають в просторі мобільного зв’язку, важливі переговори проводяться по відеозв’язку використовуючи планшети, договори пересилаються використовуючи

месенджери, а оплати за різні послуги проводяться не покидаючи інтерфейсу додатку.

Ця безумовна зручність полегшила життя багатьом людям. Проте часто вони навіть не задумуються про приватність та безпеку тих даних, які вони без питань передають своєму пристрою. Сьогодні мобільні девайси знають про своїх власників абсолютно все: від паролей у всі сервіси до паспортних даних чи планів на майбутнє.

Саме тому на плечі розробників лягає відповідальність в безпечному збереженні та передачі важливих даних, і результати роботи спеціалістів в сфері криптографії є важливим механізмом у забезпеченні такої надійності даних.

Актуальність та важливість роботи полягає в тому, що в ній розглянуто теоретичну складову алгоритмів у криптографії, проаналізовано сучасні рішення по забезпеченню безпеки даних користувачів мобільних додатків, а також створено програмний застосунок, в якому імплементовано найпопулярніші алгоритми та рішення в коді, а також показано їх роботу.

Структура роботи

Дана робота містить три основні розділи.

Перший розділ складається з аналізу предметної області, дослідження актуальності використання криптографії в мобільній розробці. Також розглянуто випадки витоків даних та наслідки від них.

У другому розділі зроблено огляд алгоритмів з криптографії, проаналізовано актуальність використання тих чи інших в сучасній розробці.

Третій розділ містить опис практичної імплементації алгоритмів в коді, опис існуючих рішень та аналіз складності їх підключення і використання.

Розділ 1. Аналіз предметної області та дослідження її актуальності

1.1 Актуальність досліджень в сфері криптографії

Безпека даних та їх передачі займала важливе місце в думках людей ще до появи цифрових технологій. Такі питання як до прикладу те, як безпечно передати лист отримувачу щоб в разі крадіжки його не можна було прочитати, з'явились в думках людей ще до часів цифрового буму. Проблеми, які поставали тоді, вирішувались відносно простим закодуванням та розкодуванням даних.

Таким чином актуальність даної проблеми тримається вже протягом десятків, а то і сотень років, а задача криптографії полягає в тому, щоб цю безпеку надавати. Той факт, що її існування було ще за часів Юлія Цезаря підтверджує, що спроби досягнути безпеки даних були ще задовго до цифрової революції, а тому ця галузь науки не є новою і спроби її покращити продовжуються вже протягом багатьох років.

Вплив цифрових технологій з кожним роком збільшується, а тому все більше інформації зберігається та передається мережею. Саме тому увага до криптографії та досліджень в сфері сьогодні також зростає. Ponemon Institute має власне дослідження щодо трендів криптографії, яке ілюструє збільшення уваги до сфери. Згідно даних в опитуванні, які зібрані в проміжку з 2005 до 2021 року, спостерігаємо лінійне збільшення кількості компаній, які мають стратегію кодування даних по всій своїй структурі [6]. Також пряма кореляція з тим, що з 2005 року і до сьогодні можна спостерігати, що кількість компаній без стратегії стабільно зменшується.

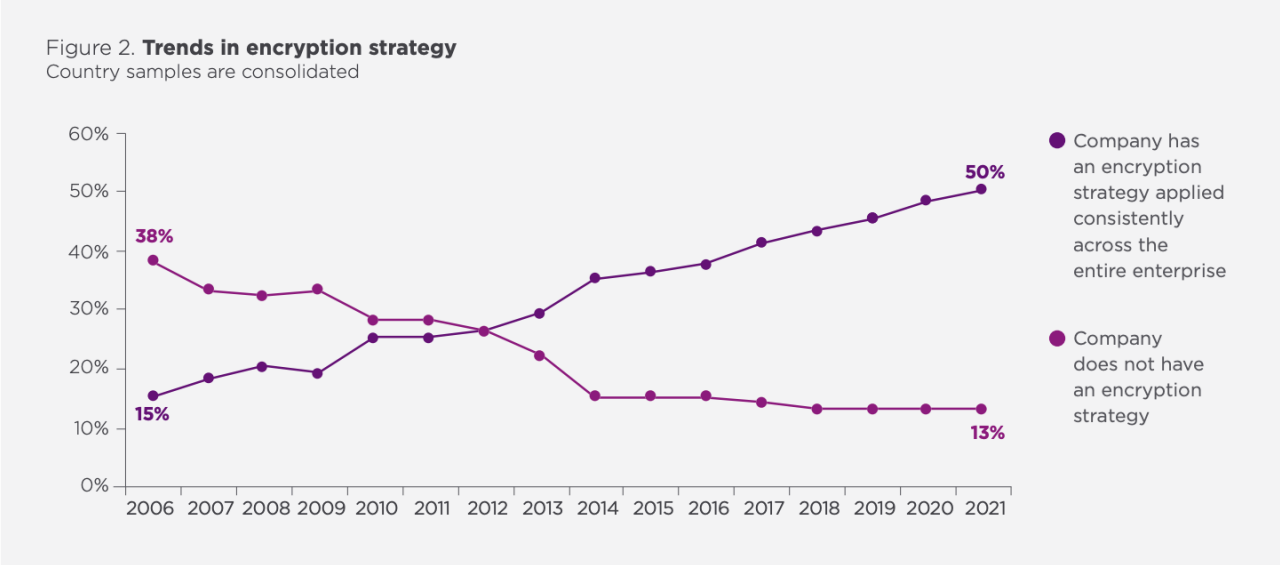

Рисунок 1.1 Графік трендів в наявності стратегій кодування [6]

Зі збільшенням популярності цифрових сервісів збільшується і увага до них від зловмисників, адже закодована інформація часто має високу цінність і свою аудиторію зацікавлених покупців. Згідно Purplesec, тільки за час пандемії COVID-19 кількість кіберзлочинів зросла на 600% [7]. Багато інструментів використовується для того аби отримати доступ до конфіденційної інформації починаючи від соціальної інженерії, яка за даними Purplesec є методом 98% кіберзлочинів [7], і закінчуючи зворотньою розробкою, яка за необхідних навичок та вмінь є серйозним інструментом.

Такий постійний пошук способів доступитись до існуючих алгоритмів шифрування є серйозним стимулом щодо необхідності в ідентичному до зловмисників темпі продовжувати дослідження в галузі криптографії, а також створення нових способів безпечного зберігання та передачі інформації.

Крім попередніх факторів, фінансова складова бізнесу відіграє одну з найбільших ролей у переважній більшості компаній, адже отримання прибутку є однією з важливих основ у житті кожної такої одиниці. Саме тому будь які збитки не є бажаними і керування ризиками щодо них є важливим елементом в управлінні бізнесом.

Витоки даних є одними з таких ризиків, який може принести велику суму збитків для компанії та має бути мінімізованим. За останніми дослідженнями IBM, які вона проводить щорічно протягом останніх сімнадцяти років, видно, що середня ціна витоку даних зросла на 11.9% відносно 2015 року [8].

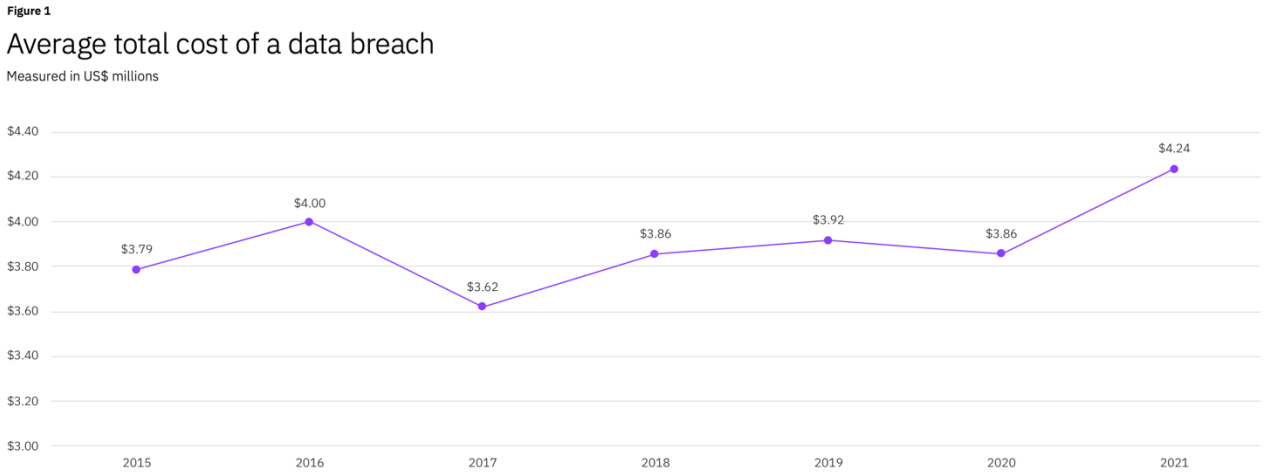

Рисунок 1.2 Середня ціна витоку даних [8]

Також з цього ж дослідження цікаво відзначити, що середній час на пошук та виправлення витоку також зростає. Порівнюючи 2015 та 2021 роки бачимо, що цей показник зріс на 12 днів [8].

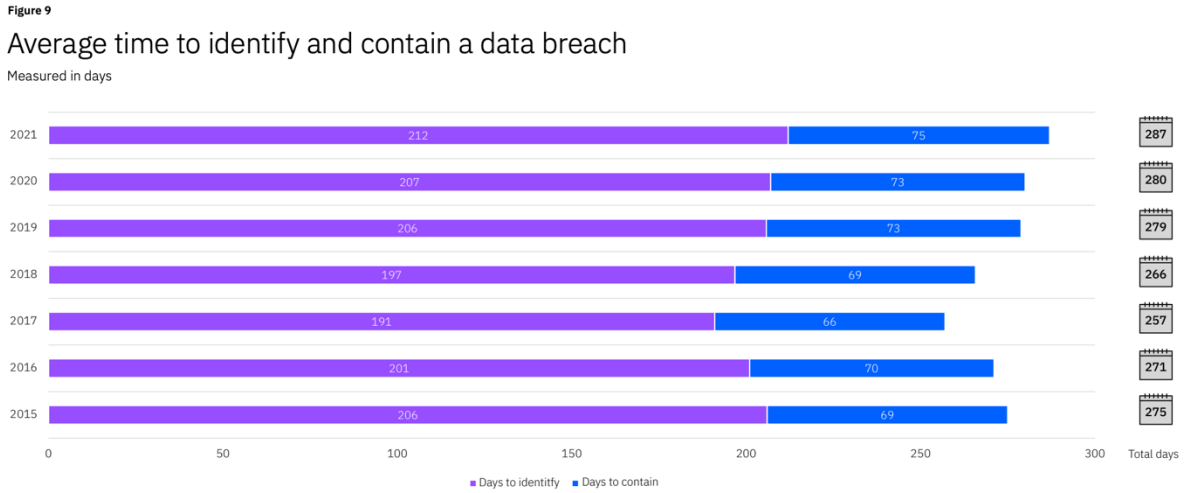

Рисунок 1.3 Середній час на знаходження та виправлення витоку [8]

Ponemon Institute після власних обрахунків зробив висновок, що впроваджуючи нові підходи в криптографії, а також використовуючи актуальні рекомендації по кібербезпеці, бізнеси зможуть в середньому заощаджувати 1.4 мільйони доларів на кожній атаці [9].

З цього дослідження можемо зробити висновок, що такий інцидент як витік даних не тільки коштує компанії репутації, а ще й завдає суттєвих фінансових збитків. Саме тому в інтересах бізнесу має бути забезпечення надійного керування даними, в чому розвиток криптографії надає пряму підтримку.

1.2 Основні причини та відомі випадки витоків даних

Враховуючи безпосередній вплив витоків даних на актуальність криптографії, а також враховуючи той факт, що ціль криптографії як науки в тому, щоб дані зберігались та передавались безпечно, доцільно буде розглянути основні причини та випадки витоків даних в індустрії.

Безпека даних – багатостороння проблема і її цілісність може бути порушеною різними підходами. Незважаючи на велику кількість можливих шляхів для втручання, в центрі тієї, яка найбільше використовується, стоїть людина.

Варто відзначити, що криптографія хоч і може гарантувати максимальну безпеку і суттєво знизити шанс витоку даних, проте зарадити фактору людської помилки не може. Цим часто і користуються зловмисники.

За статистикою від Purplesec, 33% витоків відбулися за допомогою соціальної інженерії. Це суттєвий показник, проте фокус даного дослідження лежить в площині впливу криптографії, а тому, незважаючи на його важливість, вплив соціальної інженерії в роботі розглянуто не буде.

Способи отримання цікавих зловмисникам даних не закінчуються на соціальній інженерії. Згідно звіту від Identity Theft Resource Center, найбільшу частку серед досліджених ними витоків даних мали ті, що були спричинені прямими кібератаками [10].

Частка таких атак відносно інших суттєво зросла останній рік, що може свідчити про те, що динаміка розвитку інструментів протидії шифруванню є швидшою, аніж розробка нових підходів.

ATTACK VECTOR TRENDS	2021	2020	2019
Cyberattacks	**1,613**	**878**	**928**
Phishing/Smishing/BEC	537	383	490
Ransomware	321	158	83
Malware	139	104	112
Non-secured Cloud Environment	23	51	15
Credential Stuffing	14	17	3
Unpatched software flaw	4	3	3
Zero Day Attack	4	1	n/a
Other - not specified	436	161	222
NA	106	n/a	n/a
Human & System Errors	**179**	**152**	**231**
Failure to configure cloud security	54	57	56
Correspondence (email/letter)	66	55	89
Misconfigured firewall	13	4	4
Lost device or document	12	5	19
Other - not specified	34	31	63
Physical Attacks	**51**	**78**	**118**
Document Theft	9	15	19
Device Theft	17	30	57
Improper Disposal	5	11	14
Skimming Device	1	5	4
Other - not specified	19	17	24
Unknown	**12**	**n/a**	**2**

Рисунок 1.4. Розподіл напрямків атак, що спричинили витоки інформації [10]

Витоки даних трапляються з компаніями різних розмірів, а збитки сягають великих сум. Розглянемо декілька відомих випадків і проаналізуємо потенційний вплив криптографії на них.

Великий витік даних відбувся у британської компанії Equifax, що займається кредитуванням [11]. Наслідком стало викрадення інформації про близько 145 мільйонів громадян Сполучених Штатів Америки та більше, ніж 10 мільйонів громадян Великої Британії. Можливість такого витоку стала реальною завдяки ігноруванню компанією рекомендацій безпеки, а також недійсному цифровому сертифікату.

Зловмисники мали доступ до системи з травня до липня 2017 року. В даному інциденті розуміння криптографії допомогло б запобігти таким наслідкам. Наявність HTTPS надала би можливість перевіряти трафік мережі на наявність встановлених некоректних сертифікатів і відповідно дала б змогу ідентифікувати зловмисників швидко.

Інший випадок з витоком даних відбувся в державній структурі США. На початку 2018 року міністерство оборони США надіслало електронний лист з помилковим списком отримувачів [12]. Відповідно до цього, лист було отримано і звичайними людьми.

Це привело до витоку даних військових корпусу морської піхоти США включаючи їх контакти, ідентифікаційні коди, а також інформацію про банківські рахунки. Правильний підхід до безпеки таких даних полягає в тому, щоб кодувати вихідний лист задля унеможливлення його прочитання людиною, що не є прямим адресатом.

Поганим прикладом також може слугувати інцидент з додатком NQ Mobile Vault. У ньому не відбулося прямого витоку даних користувачів проте він є зразком того, що користувачам не можна просто довіряти маркетинговим заявам компаній, а компаніям варто більш уважно підходити до безпеки своїх додатків, особливо коли це є основним їх функціоналом.

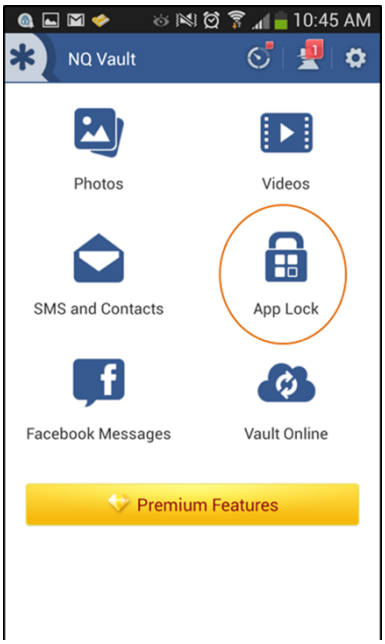

Рисунок 1.5 Інтерфейс NQ Mobile Vault[13]

NQ Mobile Vault надавав користувачам можливість шифрувати їх дані задля того аби вони не могли бути прочитані третьою стороною. Для цього користувач обирав що саме він хоче закодувати, налаштовував пароль доступу, і мав вводити його при бажанні переглянути той чи інший вміст.

Даний додаток був доволі популярним і отримував нагороди як кращий в своїй категорії, проте ніхто не дізнавався механізм шифрування даних, який використовувався. Ентузіасти провели аналіз підходів використаних у ньому і той сервіс, що мав би надійно шифрувати дані користувача, використовував слабкий за складністю алгоритм для кодування.

Замість використання будь-якого, навіть базового алгоритму, NQ Mobile Vault шифрували використовуючи звичайну побітову XOR заміну. Більше того, шифрувались лише перші 128 байт даних. На сьогоднішній день сервіси, що відтворюють медіа контент, вміють заміняти пошкоджені байти інформації, а тому з певною ймовірністю могли б замінити зашифровані 128 байт на правильні, відтворивши таким чином зашифрований користувачем контент.

Даний приклад є ще одним показником того, що навіть ті сервіси, які запевняють в гарантіях безпеки і користуються популярністю, не завжди її насправді надають, а також підкріплює впевненість в актуальності криптографії та її правильних підходів.

Наведені приклади є не найбільшими в історії, але і далеко не єдиними. Вони показують, що важливо дотримуватись стратегії безпеки в команді, впроваджувати нові підходи по кодуванню важливих даних, а також завжди моніторити стан системи щоб помітити неприродню її поведінку в разі доступу до неї зловмисниками.

1.3 Проблеми, що вирішує криптографія на мобільних застосунках

Як і увага до криптографії, популярність мобільних пристроїв також різко зростає, а програмні застосунки наявні на платформах покращуються в своїх можливостях, таким чином полегшуючи користувачам життя.

Сьогодні, маючи один лише телефон користувач може використати послуги банку, відсканувати власні документи, купити квитки на літак, або ж переглянути свої особисті нотатки. Пристрої є носіями великої кількості інформації про користувача: його місцезнаходження, його діяльності на сайтах або ж в соціальних мережах, паролі від сервісів.

Потреба в приватності та конфіденційності унеможливлює ігнорування криптографії в тому чи іншому вигляді під час процесу розробки мобільного застосунку. Необхідність використання криптографії підсилюється і тим, що зловмисник може викрасти не тільки програмне забезпечення, але і сам девайс фізично.

Незважаючи на те, що криптографія як наука з'явилась задовго до мобільних пристроїв як таких, знання з неї рідко є у спеціалістів з мобільної розробки. Частково саме через це і трапляються випадки як, наприклад, згаданий раніше з NQ Mobile Vault.

Майже кожен додаток сьогодні так чи інакше зберігає певні особисті дані користувачів. Проаналізувавши проблематику та частоту витоків даних можна стверджувати, що надійне зберігання інформації про користувача, а також правильна автентифікація є необхідними для безпечної роботи продукту.

Автентифікація – це не завжди логін користувача в додаток, а більш обширне поняття, яке можна описати як процес, що перевіряє та підтверджує певну інформацію. До прикладу, коли було створено той чи інший документ, хто є користувачем даного девайсу, звідки походить файл, та інші. Така перевірка зменшує ризики пропустити дії зловмисників. Криптографічні інструменти вирішують дану проблему за допомогою цифрових підписів.

Цифровий підпис – це математична схема для перевірки автентичності цифрового повідомлення [14]. Є різні способи його створення проте цільове призначення полягає саме в тому, щоб унеможливлювати підробку цифрових документів та автентифікувати їх походження.

Звісно, авторизація користувачів є ще однією важливо точкою, де криптографічні алгоритми забезпечують правильну роботу мобільного додатку.

Розробники повинні передбачати різні сценарії користування додатком, включаючи викрадення або втрату мобільного пристрою, або ж загалом доступ до смартфону третьою стороною, а саме тому необхідно мати стратегію автентифікації користувача.

Криптографічні підходи та правильна архітектура додатку є необхідними складовими в таких ситуаціях і допомагають мінімізувати ризики для користувача. В деяких додатках така система авторизації є абсолютно необхідною. До прикладу, мобільний банкінг та криптогаманці.

Як вже було згадано раніше, мобільний банкінг стрімко розвивається та отримує нових користувачів. Все більше банків доєднуються до тренду та створюють власні рішення. Проте далеко не всі структури винаймають професійних розробників, незважаючи на необхідність мати надійний захист як даних користувача, так і правильну систему доступу до них.

Криптовалюти є також великим трендом останніх років, а криптогаманець можна тримати на власному пристрої. Доступ зловмисника до такого гаманця може спричинити втрату всіх вкладень, проте правильне застосування криптографічних алгоритмів від авторизації до проведення платежів є механізмом, що зменшує можливість такого сценарію.

Криптографічні підходи вирішують не тільки проблему авторизації користувача в мобільному додатку, а також і зберігання чутливої інформації в ньому. Розробниками операційних систем вже було написано багато рішень для таких юзкейсів проте вони все ще іноді ігноруються при розробці стороннього програмного забезпечення, і всі дані зберігаються у звичайному форматі.

Криптографічні алгоритми також вирішують потенційні проблеми в комунікації мобільного застосунку зі сторонніми елементами, такими як модулі аналітики, бекенд, та інші сторонні допоміжні сервіси. Не рідко ключі доступу зберігаються не зашифрованими в коді.

Є багато можливих інструментів як отримати вихідний код застосунку. Хоч сам по собі код є зашифрованим, проте існують методики зворотньої розробки, що дозволяють отримувати вихідний код. Таким чином, зберігаючи

ключі доступу в коді без шифрування розробники ризикують отримати витік даних в майбутньому.

1.4 Висновки до розділу 1

В першому розділові було оглянуто актуальність досліджень та роботи в сфері криптографії, а також причини необхідності знань в галузі.

Також враховуючи те, що нехтування безпекою застосунків неодноразово призводило до витоків даних, було оглянуто відомі причини їх появи, середні витрати компаній на такі випадки, а також проаналізовано як правильне використання криптографічних алгоритмів допомогло б уникнути збитків.

Після цього було оглянуто те які проблеми криптографія вирішує в площині мобільних застосунків.

Розділ 2. Теоретичні відомості щодо криптографії та використовуваних алгоритмів

2.1 Загальні поняття з криптографії

Криптографію можна описати як набір підходів призначений для зберігання даних та їх передавання від відправника до отримувача в такий спосіб, щоб лише отримувач мав змогу прочитати інформацію в її початковому вигляді. Шифрування інформації відбувається за допомогою обрахунків, які в свою чергу покладаються на математичні гіпотези і загально мають назву алгоритми.

Початок певної задачі в криптографії починається з того, що присутній вхідний текс, що є незашифрованим, так званий «plaintext». В результаті роботи певного криптографічного алгоритму та ключу, що є застосовані до вхідного тексту, ми отримуємо на виході текст, що є зашифрованим. В процесі життя зашифрованого ключа, також передбачається його розшифрування отримувачем. Можна розглядати процес шифрування тексту як:

$$C = Ek(P),$$

де C – це зашифрований текст, E – це метод шифрування, k – визначений ключ, а P – це вхідний текст.

Процес дешифрування позначається як

$$P = Dk(C),$$

де P – це розшифрований текст, D– це метод дешифрування, k – визначений ключ, а C – зашифрований текст.

Також справедливою є така рівність:

$$Dk(Ek(P)) = P$$

В основі криптографії лежать чотири основні принципи: шифрування, автентифікація, цілісність, та невідмовність.

Шифрування беззаперечно є одним з найважливіших принципів. Він передбачає необхідність того, щоб повідомлення або ж будь-яка інформація були закодованими або такими, яких неможливо прочитати задля забезпечення

безпеки. Даний принцип створює необхідність для розкодування отриманої інформації отримувачем з використанням спеціального цифрового ключа.

Автентифікація як принцип криптографії передбачає в собі ідентифікацію джерела повідомлення. Після визначення джерела стає зрозуміло з чим саме необхідно налаштувати канал комунікації, а також зробити його захищеним. Даний принцип стає можливим у випадку коли тим чи іншим способом відбувається валідація особи відправника за допомогою обміну ключами.

Не менш важлива і цілісність інформації, що надсилається. Даний принцип позначає що криптографічні алгоритми мають забезпечувати цілісність даних, а саме щоб отримувач отримував повідомлення з очікуваним контентом від правильного відправника. Отримувач має також бути впевненим, що інформація не була жодним чином змінена або ж порушена протягом процесу її передачі йому. Зазвичай хеш функція є механізмом, що гарантує зазначену цілісність даних.

Значення невідмовності як принципу полягає в тому, що суб’єкт, який надсилає інформацію не має можливості заперечувати факт надсилання. Коректну роботу даного принципу забезпечують цифрові підписи, що унеможливлюють заміну джерела даних на інше. Його користь полягає в тому, що джерело даних завжди відоме та достовірне.

Однією з важливих характеристик криптоалгоритмів є стійкість. Стійкість алгоритму визначається в його захищеності від зовнішніх атак. На сьогоднішній день з точки зору оцінки стійкості шифри базуються на принципі Керкгоффза [16]. Даний принцип передбачає, що таємність шифру передбачається не таємністю самого алгоритму, а радше таємністю ключа. Це важлива особливість і шифри, які базуються саме на таємності алгоритму називають обмеженими.

Основним недоліком такого підходу є ймовірність того, що зловмисник має більший простір для дій з метою виявлення потенційних вразливостей. До прикладу, в разі якщо це програмний продукт, то аналіз або ж зворотня розробка його вихідного коду дає змогу зрозуміти використані підходи та застосувати їх для дешифрування цікавої третій стороні інформації.

Саме тому правильним вважається підхід в криптоаналізі коли аналіз надійності алгоритму проводиться з урахуванням того, що зловмисник має всю інформацію про систему і невідомим залишається лише значення самого ключа, використаного в алгоритмі.

2.2 Типи криптографічних алгоритмів

Існують різні підходи до класифікації криптографічних алгоритмів проте найбільш розповсюдженим можна вважати підхід виходячи з кількості використаних ключів.

Згідно нього, криптографічні алгоритми поділяються на три типи: симетричні, асиметричні, та хеш-функції.

Симетричні алгоритми також називають алгоритмами з приватним або ж секретним ключем. Даний алгоритм використовує лише один ключ який відіграє роль у зашифруванні для відправника, а також для розшифрування контенту у випадку використання ключа отримувачем. Використовують даний тип шифрів переважно для приватності та конфіденційності.

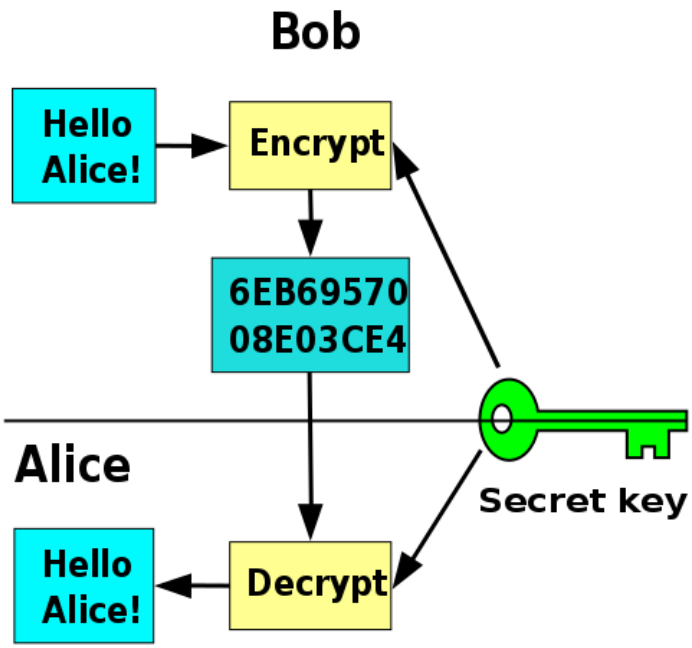

Рисунок 2.1 Принцип роботи алгоритму з симетричним ключем [16]

В даному підході і відправник і той, хто отримує інформацію мають знати ключ, який в свою чергу є секретом. Однією з найважчих задач в даному типові шифрування є саме передача ключа.

Симетричні ключі, в свою чергу, загалом поділяються на потокові та блокові шифри.

Потокові шифри працюють в рамках одного біту на ітерацію та імплементують підхід який дозволяє ключеві постійно змінюватись. Двома з основних типів потокових шифрів є самосинхронізуючі шифри та синхронізуючі шифри.

Шифри з самосинхронізацією обраховують кожен біт в потоці за допомогою фукнції з попередніх n бітів потоку. Суть саме самосинхронізації в назві полягає в тому, що процес розшифрування може залишатись синхронізованим з процесом шифрування лише знаючи актуальне значення в потоці.

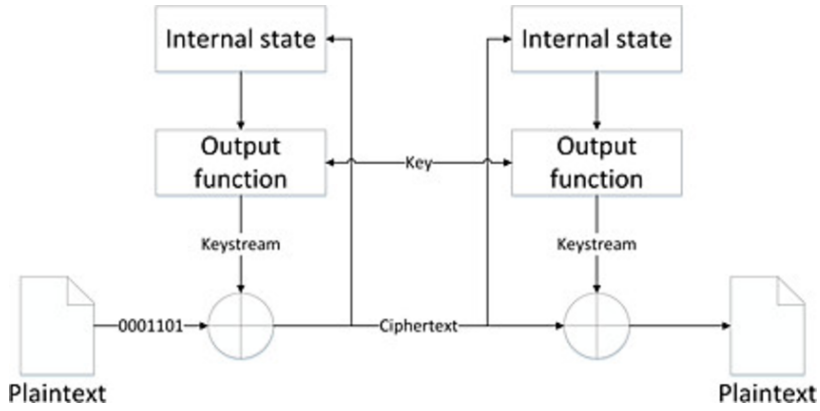

Рисунок 2.2 Схема роботи шифрів з самосинхронізацією [17]

Синхронізовані потокові шифри, в свою чергу, генерують ключовий потік незалежно від потоку повідомлення, але використовуючи той самий підхід до генерації ключового потоку для відправника та отримувача.

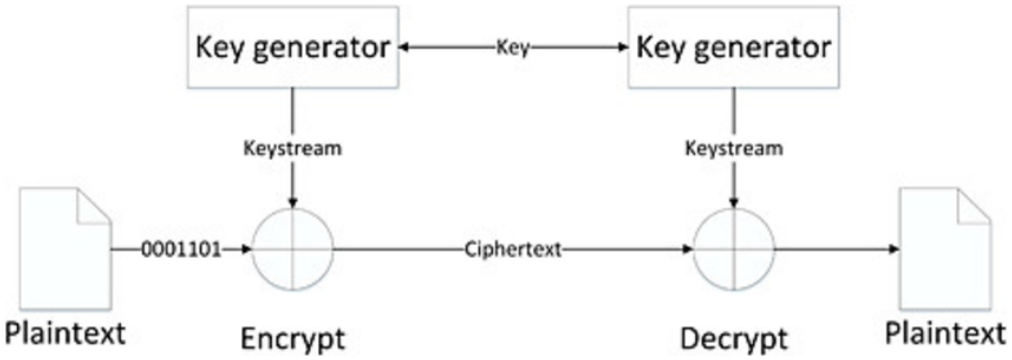

Рисунок 2.3 Схема роботи синхронізованих шифрів [17]

Варто також відзначити, що потокові шифри не мають детермінованого процесу шифрування та декодування. А саме один і той самий ключ не дає ідентичного результату при багаторазовому використанні.

Механізм роботи блокового шифру полягає в тому, що зашифровується лише один блок інформації фіксованого розміру за раз. Блокові шифри є детермінованими, а саме кожна вхідна інформація буде закодованою в ідентичну зашифровану інформацію за умови використання однакового ключа.

Найвідомішим представником блокового шифрування є шифр Фейстеля, який ще називають мережею Фейстеля. Даний шифр комбінує в собі елементи заміни, переміщення, та розширення ключів і як перевагу має те, що шифрування та декодування за своєю суттю є майже ідентичними. В певних випадках щоб виконати одну з операцій необхідно застосувати зворотню імплементацію іншої, що таким чином суттєво зменшує ресурси необхідні для реалізації такого підходу.

Як вже було згадано раніше, основною проблемою в симетричному шифруванні є передача ключа сторонам, що задіяні у процесі. Дану проблему вирішує собою новий підхід, який було названо асиметричним шифруванням.

Деякі спеціалісти в сфері криптографії зазначають, що поява цього виду шифрування, який також називають шифруванням з публічним ключем, є найважливішою подією в розвитку криптографії за останні 400 років.

В підході асиметричного шифрування використовуються два ключі – публічний та приватний. Приватний ключ зберігається у однієї зі сторін і не має бути поширеним. Інший же ключ є публічним, а тому може передаватись будь-кому і будь де, розкриття його значення не є критичним. В загальному випадку, асиметричне шифрування передбачає, що обидва ключі мають математичну залежність один від одного проте знання одного з них не є достатнім для обрахування іншого.

За визначенням, в комунікації беруть участь дві сутності A та B. Сутність A має публічний ключ e і відповідний йому приватний ключ d. В захищених системах обрахування ключа d маючи ключ e є складною за ресурсами задачею. Публічним ключем є трансформацією шифрування E_e , в той час як приватний ключ визначає асоційовану трансформацію розшифрування D_d.

Сутність B, яка надсилає повідомлення m до A отримує зашифрований текст $c = E_e(m)$ і передає його A. Для розшифрування даного повідомлення A застосовує трансформацію з розшифрування і отримує початковий текст $m = D_d(c)$ [18].

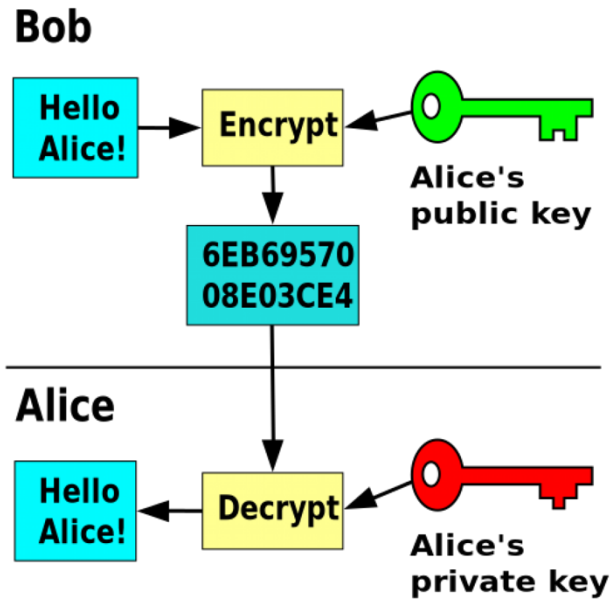

Рисунок 2.4 Принцип роботи алгоритму з асиметричним ключем [16]

Задачею публічного ключа є шифрування вхідного тексту, в той час як приватний ключ відповідає за отримання вхідного тексту з зашифрованого. Такий підхід дозволяє безпечно шифрувати дані в публічних каналах при цьому пропонуючи цілісність даних та правильного відправника і отримувача. Також відсутня необхідність в обміні ключами.

Незважаючи на те що концепт шифрування з публічним ключем надає багато переваг, проте така безпека даних все ж вимагає більшої кількості ресурсів при використанні. Через великий розмір ключів, а також необхідність використання двох ключів процес шифрування і розшифрування займає більше часу. Виходячи з цього можна вважати, що асиметричне шифрування підходить краще для роботи з маленькими обсягами даних. Зазвичай підходи асиметричного шифру використовуються для налаштування безпечного каналу комунікації, а вже для передачі інформації відбувається перехід на симетричне шифрування, яке вимагає менше ресурсів.

Хеш-функції є ще одним важливим типом криптографічних алгоритмів. Вони являють собою певну детерміновану функцію, яка на вході має стрічку бітів довільної довжини, а на виході завжди бітова стрічка сталої довжини n.

Хеш-функція з входом m позначається як H(m) і продукує значення, яке називають хешем. Вхідні ж дані називають прообразом функції. Для того щоб довільна функція могла називатись хеш-функцією, мають виконуватись певні властивості, а саме стійкість до пошуку першого прообразу, стійкість до пошуку другого прообразу, а також стійкість до колізій.

Стійкість до пошуку першого прообразу визначає відсутність ефективного поліноміального алгоритму обчислення зворотної функції для заданої хеш-функції. Інакше це позначають як те, що хеш-функція є односторонньою функцією.

Стійкість до пошуку другого прообразу має під собою властивість, що маючи повідомлення і його згортку знайти таке повідомлення, яке відрізняється від початкового проте має ідентичну згортку.

Стійкість до колізій визначає відсутність ефективного алгоритму що виконується за поліноміальний час і знаходить колізії. Колізія – це пара відмінних одне від одного повідомлень, які мають ідентичні значення хешу.

Ще одним важливим елементом, що забезпечує ефективність хеш-функцій, є те, що значення хешу має сильно змінюватись навіть при мінімальній зміні вхідного повідомлення.

2.3 Аналіз актуальних алгоритмів при розробці мобільних додатків

Всі види криптографічних алгоритмів активно використовуються і в мобільній розробці. Криптографія виконує багато задач в даній предметній області, а тому для різних проблем використовуються різні способи їх вирішення з урахуванням ресурсів, необхідних для виконання задач.

Розглянемо ті підходи, які сьогодні є найпоширенішими в розрізі задач криптографії на мобільних пристроях.

Мобільні застосунки тісно пов’язані з такими проблемами як цифрові підписи, верифікація паролей та повідомлень. У розрізі таких задач часто використовуються хеш-функції. Можна розглянути приклад верифікації паролей і як вирішується дана задача. Коли користувач його вводить у додатку, він не надсилається в розшифрованому вигляді, а хешується на девайсі, після чого відправляється на сервер для верифікації. На самому сервері паролі також зберігаються лише у значенні хешу. Таким чином, ми не здійснюємо зберігання чи відправку незашифрованих паролей, покращуючи безпеку, а також зберігаючи можливість перевірити введений користувачем пароль на валідність.

В розробці під iOS фреймворки орієнтуються на сім’ю алгоритмів SHA-2, а саме SHA-256, SHA-384, SHA-512, серед яких найпопулярнішим є SHA-256. Він повертає хеш розміром 256 біт і є суттєво більш надійним, аніж його попередник SHA-1 та аналог MD5. З точки зору швидкодії він є повільнішим на близько 20-30%. SHA-2 також є формально визначеним алгоритмом Національним Інститутом Стандартів та Технологій.

Такі алгоритми як SHA-1 та MD5 також можуть використовуватись проте більше для зворотної підтримки, а не як першочерговий вибір, адже не гарантують достатнього рівня безпеки при їх використанні. MD5 генерує хеш розміром 125 біт і раніше активніше використовувався в мобільній розробці, проте зі знаходженням вразливостей його замінили аналоги і він не є рекомендованим для використання. SHA-1 є першою ітерацією алгоритмів сім'ї SHA і генерує хеш розміром 160-біт. Його популярність використання так само спала в моменті коли було знайдено вразливості в алгоритмі.

Варто відзначити існування ще одного алгоритму хешування, який використовується лише на iOS і поки що лише в Apple. Його назва NeuralHash. Поява алгоритму викликана бажанням Apple боротись з розповсюдженням медіа, що містять в собі випадки сексуального та фізичного насилля над дітьми.

Задача, що стоїть перед ним полягає в тому, щоб ідентифікувати фото з таким контентом у користувачів Apple девайсів, і коли кількість підозрілих фото перебільшує певний поріг, Apple надсилає звіт у NCMEC(National Center for Missing and Exploited Children).

Задля того щоб зберегти приватність користувачів, у своєму підході, який було названо CSAM(Children Sexual Abuse Material), Apple не матиме доступ до самого зображення, адже мова йде про всі фото власників iPhone, які зберігаються в хмарі.

Для того щоб мати можливість шукати протизаконний вміст без перегляду самого фото було вирішено створити базу даних хеш значень зображень, які містять насилля, і для кожного користувача так само створити хеш значення його фото порівнюючи їх з тими, які є в попередньо створеній базі даних.

Існуючі алгоритми хешування не підійшли під задачу, адже одна з вимог хеш функцій полягає в тому, щоб мінімальна зміна у значенні вхідної стрічки суттєво змінював хеш значення.

З цього випливає, що мінімальна зміна в картинці як то зміна розширення або ж гірша якість призводила б до зміни бітового відображення, і в свою чергу сильно змінювала б результат застосування хеш функції до неї.

Саме тому було створено NeuralHash, особливістю якого є менша чутливість до змін, що в свою чергу не реагує на незначні відмінності у зображеннях і генерує для них однаковий хеш. Алгоритм використовує згорткову нейронну мережу для обчислення значення хешу.

Дана нейронна мережа навчається на парах картинок, де правильні пари – це картинки, де присутня одна з простих трансформацій, як от обрізання, зміна розміру, чи зміна куту нахилу зображення. А невірні пари складаються з абсолютно різних зображень. Суть тренування мережі полягає в тому, щоб навчити її правильним парам надавати однаковий хеш, а неправильним – різний. Таким чином отримується результат того, що малі перетворення в картинці ігноруються.

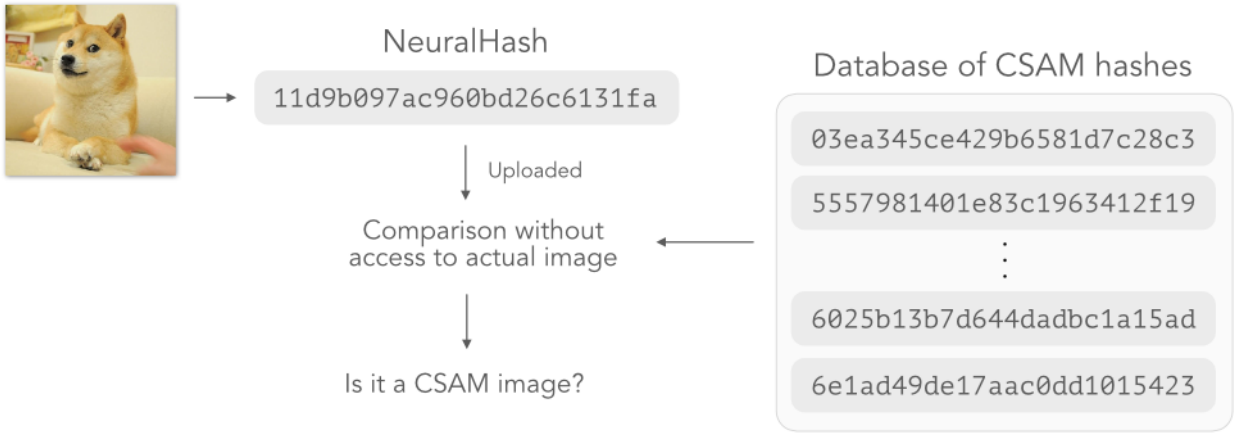

Рисунок 2.5 Алгоритм роботи NeuralHash [19]

Проте ентузіасти знайшли можливість експортувати модель для навчання і виявили, що в багатьох випадках трапляються колізії хешу для абсолютно різних зображень, як от до прикладу фото собаки та кота. Такі колізії потенційно можуть створювати невірні рішення щодо контенту на фотографіях на стороні серверів Apple та потенційно починати моніторінг за непричасними до насилля людьми.

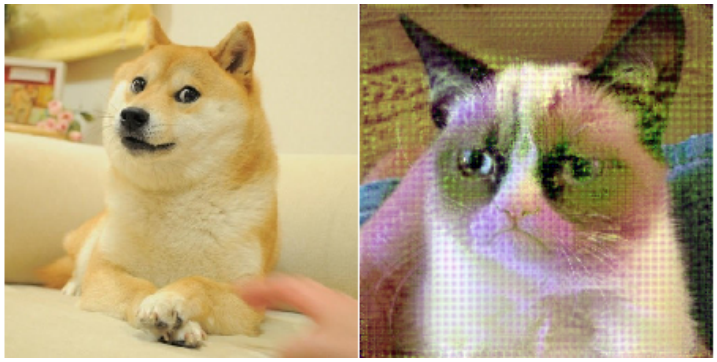

Рисунок 2.6 Зображення з однаковим значенням хешу в NeuralHash[19]

Незважаючи на існуючі недоліки алгоритму варто відмітити, що це цікавий вектор розвитку для такої компанії як Apple.

Так само важливими є і алгоритми шифрування з приватним ключем. На практиці, вони часто поєднуються з підходами по асиметричному шифруванні, перекриваючи недоліки один одного. Роль симетричного шифрування в такому сценарії полягає саме в шифруванні та розшифруванні даних.

На сьогоднішній день еталонним вважається шифр AEAD. Authenticated Encryption with Associated Data (AEAD) – це шифр, що дозволяє одночасно автентифікувати шифр та забезпечити його цілісність. AEAD по своїй суті відповідає чотирьом принципам: секретність, автентичність, симетричність, рандомізація.

В основі секретності лежить відсутність будь-якої інформації про зашифрований текст у відкритому доступі, окрім його довжини.

Автентичність полягає в тому, що не є можливим змінити зашифрований текст так, щоб це не було помічено.

Симетричність як властивість є ознакою того, що для шифрування та розшифрування повідомлення використовується один і той самий ключ.

Рандомізація гарантує випадковість шифрування. А саме, два повідомлення з однаковим вхідним текстом не матимуть ідентичний зашифрований текст. Це дозволяє уникнути випадку, де хакери мають можливість підібрати вхідний текст під шифр.

Конкретно два його представники, що трапляються найчастіше – це AES-GCM та ChaCha20-Poly1305.

AES-GCM, або ж повна назва Advanced Encryption Standard with Galois Counter Mode, вперше був представлений Національним інститутом стандартів та технологій США. Це блоковий шифр, що надає швидке шифрування та цілісність даних.

Під час своєї роботи, AES-GCM приймає ініціалізаційний вектор, додаткову автентифікаційну інформацію, секрет, та вхідну стрічку на шифрування. Алгоритм працює з вхідними та вихідними даними розміру 128 біт, 192 біти, та 256 біт. Від довжини ключа шифру залежить кількість ітерацій трансформації і, відповідно, час та ресурси які витрачаються на виконання операції.

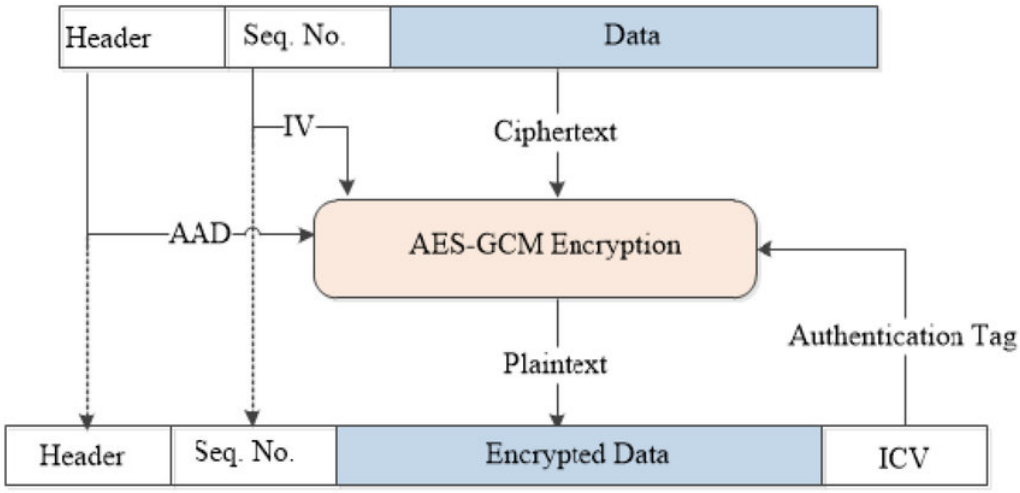

Рисунок 2.7 Схема AES-GCM шифру [20]

ChaCha20-Poly1305 – це алгоритм, що комбінує в собі потоковий шифр ChaCha20 з кодом автентифікації повідомлення Poly1305. Він характеризується швидкодією і приймає на вхід 256-бітовий ключ, а також 96 бітовий криптографічний нонс. Нонс – це випадкове значення, що використовується один раз і часто є представлено часовою міткою [21].

Даний алгоритм зазвичай показує кращу швидкодію, аніж його конкурент AES-GCM на системах, де в ЦП не присутнє розширення системи команд AES. Дане розширення було запропоновано Intel та присутнє на архітектурах

процесорів x86 [22]. Таким чином, ChaCha20-Poly1305 часто є рекомендацією для використання виходячи з того, що він має схожий рівень безпеки і швидкодію, особливо на мобільних пристроях, які зазвичай не мають розширення системи команд AES, адже побудовані на інших архітектурах процесорів.

Розглянемо популярні в мобільній розробці підходи при використанні алгоритмів з публічним ключем. ECC алгоритми на сьогоднішній день є основним вибором у випадку використання асиметричних шифрів. Його перевагою над RSA, який був найпопулярнішим вибором з моменту свого створення у 1978 в мобільній розробці вважають набагато менші за розміром ключі, які при цьому надають відносно однаковий рівень безпеки. Основні фреймворки роботи з криптографічними алгоритмами надають готову підтримку лише ECC. Найчастіше перед вибором стоять такі різновиди алгоритму як P256/P384/P521 ECC алгоритми або ж Curve25519 ECC алгоритм, які за результатом виконання задачі не дуже відрізняються між собою.

2.4 Висновки до розділу 2

В другому розділі було розглянуто загальні поняття з криптографії, її основні принципи та підходи.

Також було проаналізовано основні типи криптографічних алгоритмів, такі як симетричні, асиметричні алгоритми, та хеш-функції. Розглянуто їх особливості та випадки використання.

В розділі також було оглянуто які конкретні криптографічні алгоритми використовуються у сфері мобільної розробки.

Розділ 3. Практика використання та реалізації алгоритмів

3.1 Відомі існуючі підходи в індустрії

Як вже було оглянуто в попередніх розділах, ринок мобільних застосунків на базі платформи iOS зростає, а тому з'являється більше запитів по захищеності даних користувачів. Аналіз того як сьогодні вирішуються криптографічні проблеми в мобільних застосунках ускладнюється тим, що їх вихідний код не є присутнім у відкритому доступі, а враховуючи важливість безпеки даних лише мала частина додатків ділиться своїми підходами та практиками.

Проаналізуємо та розглянемо як вирішуються криптографічні проблеми у певних додатках на iOS.

Bear – додаток для Apple платформ, що виконує функцію записника. З початком розробки додатку приватність нотаток користувача стояла приорітетом у Bear, а тому вони не мали доступ до них і не передавали їх на бекенд сервіси, а зберігали локально на пристрої.

В певний момент з ростом аудиторії з'явився запит на можливість більш просунутого захисту нотаток, а тому було почато роботу над криптографічними підходами, а також іншими підходами по додатковому захисту даних користувача.

Після імплементації системи безпеки в додатку кожна нотатка зберігається в зашифрованому вигляді і є розшифрованою лише після автентифікації користувача. У якості алгоритму шифрування Bear використовує AES-GCM-256, про який було згадано в розділі з описом популярних на сьогодні алгоритмів шифрування.

Задля додаткової безпеки інформаці що так чи інакше стосується користувача не зберігається у відкритій базі даних на пристрої, а використовує Keychain.

Keychain – це зашифрована база даних, що надається Apple, та призначена для невеликих обсягів інформації [23]. Розробники, у випадку використання Keychain, ізольовані від необхідності займатись ручним шифруванням даних, за

них це робить система. Таким чином, зберігання чутливих даних там дозволяє створити ще один рівень захисту через виділене API.

Крім вище перелічених заходів для забезпечення безпеки даних користувачів, Bear також імплементував всі важливі процеси управління ключами, такі як: генерація ключів, розтягування ключів, та кешування ключів.

Криптографія широко розповсюджена в такій сфері мобільних додатків як месенджери. Сьогодні – безпека даних користувачів в цьому сегменті є дуже вигідною перевагою над конкурентами, а тому використовується великими гравцями на ринку.

Одним з таких є WhatsApp. В основі криптографічних підходів застосунку лежить протокол “Signal”, який в свою чергу базується на комбінації шифрів з публічним та приватним ключами. У випадку WhatsApp симетричні шифри відповідають за конфіденційність та цілісність в той час як асиметричні шифри забезпечують інші цілі по безпеці, такі як невідмовність та автентифікація [24].

Розглянемо детальніше використання криптографії у додатку. У якості шифрів з публічним ключем використовується три пари.

Пара ідентичності – довготривала пара Curve25519 ключів, що генерується на момент встановлення.

Підписаний попередній ключ – середньотривала пара Curve25519 ключів, що також генерується на моменті встановлення і є підписаною ключем ідентичності. Періодично змінюється.

Одноразові попередні ключі – черга Curve25519 пар ключів для одноразового використання, яка також генерується на моменті встановлення і поповнюється за потребою.

WhatsApp також використовує ключі сесії, які також діляться на три типи.

Першим є кореневий ключ, що являє собою 32 байтне значення, яке використовується при створенні ланцюгового ключа.

Ланцюговий ключ в свою чергу є другим з ключів сесії і також представляє собою 32-байтне значення, яке використовується для генерації ключа повідомлення.

Ключ повідомлення – 80-байтне значення, яке відповідає за шифрування основної одиниці контенту у WhatsApp – повідомлення користувача. З 80 використаних байт 32 використовуються для AES-256 ключа, ще 32 для HMAC-SHA256 ключа, і інші 16 для вектору ініціалізації.

Також в додатку існує і інший тип ключу, який отримав назву ключ прив'язки секрету. Він складається з 32-байтного значення, що генерується на пристрої, який є кандидатом на прив'язку, і має бути переданим по безпечному зв'язку до основного пристрою. Поточна реалізація включає в себе необхідність сканування QR коду і таким чином передачі ключа. Використовується коли користувач бажає прив'язати новий пристрій до свого акаунту.

Протягом реєстрації користувач передає свої публічні ключі про які було згадано вище на сервер і вони є збереженими там, при цьому будучи пов'язаними з ідентифікатором користувача.

Для комунікації користувачів застосунку між собою, відправник має встановити зашифровану сесію з кожним девайсом отримувача. Також, задля того щоб в разі використання декількох девайсів відправником, така сесія будується і для них. Це допомагає уникнути перестворення сесії у разі якщо було використання іншого пристрою.

Для створення сесії відправник запитує набір публічних ключів отримувача з серверу і валідує кожен девайс. Після цього відправник встановлює закодовану сесію з кожним індивідуальним девайсом:

1. Відправник зберігає ключ ідентичності як $I_{recipient}$, підписаний попередній ключ як $S_{recipient,}$ і одноразовий попередній ключ як $O_{recipient.}$
2. Відправник генерує ефемерну Curve25519 пару ключів і зберігає її в $E_{initiator.}$
3. Відправник завантажує власний ключ ідентичності як $I_{initiator.}$
4. Відправник обраховує мастер секрет за формулою

$$master\_secret = ECDH(I_{initiator}, S_{recipient}) \,||\, ECDH(E_{initiator}, I_{recipient}) \,||\, ECDH(E_{initiator}, S_{recipient}) \,||\, ECDH(E_{initiator}, O_{recipient})$$

де ECDH – протокол, що на вхід приймає пару відкритий/закритий ключ на еліптичних кривих та повертає загальний секретний ключ

5. Відправник потім використовує функцію отримання ключа HKDF для створення кореневого ключа, а також ланцюгових ключів з master_secret.

До моменту поки отримувач не прочитає повідомлення, у всі надіслані в цей період СМС обов'язково додається така інформація така як ефемерна пара ключів та ключ ідентичності від відправника, що дозволить потім побудувати сесію зі сторони отримувача.

По прочитанню повідомлення отримувач обраховує відповідний master_secret використовуючи його приватні ключі та публічні ключі, отримані в повідомленні.

Після цього він видаляє одноразовий попередній ключ, використаний відправником.

Маючи налаштовану сесію, протягом неї WhatsApp використовує ключ повідомлення для кодування та HMAC-SHA256 для автентифікації. У разі потреби нового ключа повідомлення він обраховується за формулою:

$$\text{Message Key} = \text{HMAC-SHA256}(\text{Chain Key}, 0x01)$$

$$\text{Chain Key} = \text{HMAC-SHA256}(\text{Chain Key}, 0x02)$$

Дані операції гарантують отримання нового ключа, а також те, що новий ключ повідомлення не може бути використано для знаходження поточного чи одного з попередніх значень ланцюгового ключа.

Таким чином WhatsApp гарантує конфіденційність та безпеку переписки, а також особистості кожного з учасників діалогу.

3.2 Опис та пояснення вибору інструментів розробки

В якості аналізу сучасних підходів до використання криптографічних алгоритмів в мобільній розробці було написано програму, що наочно демонструє процес роботи з інструментами в рамках роботи з Apple пристроями.

На сьогоднішній день існує два варіанта мов програмування для написання додатків під платформи Apple – Objective-C та Swift. Objective-C з'явився першим в 1984 році і з того часу був основним інструментом для розробки додатків для Apple пристроїв. Його характеризує C-подібний синтаксис, який має доволі високий поріг входу. Проте в 2014 році Apple представила нову мову програмування – Swift. Він має зручний та легкий для читання синтаксис, а його підтримка у спільноті на сьогоднішній день лише збільшується. Swift має статичну типізацію, що зменшує можливість помилки, а також автоматичне управління пам'яттю, зменшуючи необхідний для розробника об'єм роботи.

Незважаючи на те, що зусилля Apple спрямовані на підтримку та розвиток Swift, Objective-C все ще користується популярністю у розробників, особливо враховуючи ту велику кількість застосунків, що були написані в часи популярності Objective-C та потребують підтримки.

Згідно щомісячного індексу TIOBE спостерігаємо, що різниця між Swift та Objective-C все ще невелика, і обидві мови покращили свої показники відносно попереднього зрізу і Swift займає 12 місце, в той час як Objective-C – 16.

May 2022	May 2021	Change	Programming Language	Ratings	Change
1	2	↑	Python	12.74%	+0.86%
2	1	↓	C	11.59%	-1.80%
3	3		Java	10.99%	-0.74%
4	4		C++	8.83%	+1.01%
5	5		C#	6.39%	+1.98%
6	6		Visual Basic	5.86%	+1.85%
7	7		JavaScript	2.12%	-0.33%
8	8		Assembly language	1.92%	-0.51%
9	10	↑	SQL	1.87%	+0.16%
10	9	↓	PHP	1.52%	-0.34%
11	17	⇈	Delphi/Object Pascal	1.42%	+0.22%
12	18	⇈	Swift	1.23%	+0.08%
13	13		R	1.22%	-0.16%
14	16	↑	Go	1.11%	-0.11%
15	12	↓	Classic Visual Basic	1.03%	-0.38%
16	21	⇈	Objective-C	1.03%	+0.24%

Рисунок 3.1 Останні результати індексу TIOBE [25]

Враховуючи стрімкий розвиток Swift, регулярні оновлення, а також те, що основні криптографічні фреймворки та нові практики підтримуються саме у розробці на Swift, він був використаний для проведення аналізу.

Для написання додатків для Apple пристроїв вибір середовища розробки не є таким різноманітним, як в інших платформ. Код на Swift можна писати будь-де, використовуючи звичайні редактори тексту, складність у його компіляції. Розробка під Apple платформи строго контрольована самою компанією, а тому проводити збірку застосунку навіть для тесту можна лише на девайсах від Apple, і робити це за допомогою інструментів XCode.

Відсутність можливості збирати додатки під управлінням iOS в інших середовищах розробки погіршує конкуренцію на ринку та створює певну монополію. З конкурентів є лише AppCode, що створений компанією JetBrains та закриває багато потреб розробників, надаючи цікавий інструмент для розробки під iOS. Проте враховуючи, що для збірки йому все рівно необхідний XCode, а також зручність роботи з IDE від Apple, вибір середовища розробки для застосунку було зупинено на XCode.

XCode – це створене Apple інтегроване середовище розробки, що містить в собі повний набір інструментів для розробки, тестування, та розповсюдження програмного забезпечення. Крім цього, вбудована підтримка роботи зі сторонніми фреймворками, система контролю версій Git, та документація по розробці. Той факт, що багато корисних в щоденній роботі речей є зібрано в одному продукті полегшує робочі процеси. Він підтримує різні мови програмування проте основний акцент зроблено саме на розробку під платформи Apple.

Інтерфейс XCode відповідає принципам Single-Window Interface, вся робота розробника розміщується в одному вікні і замінюються лише робочі елементи, а не саме вікно.

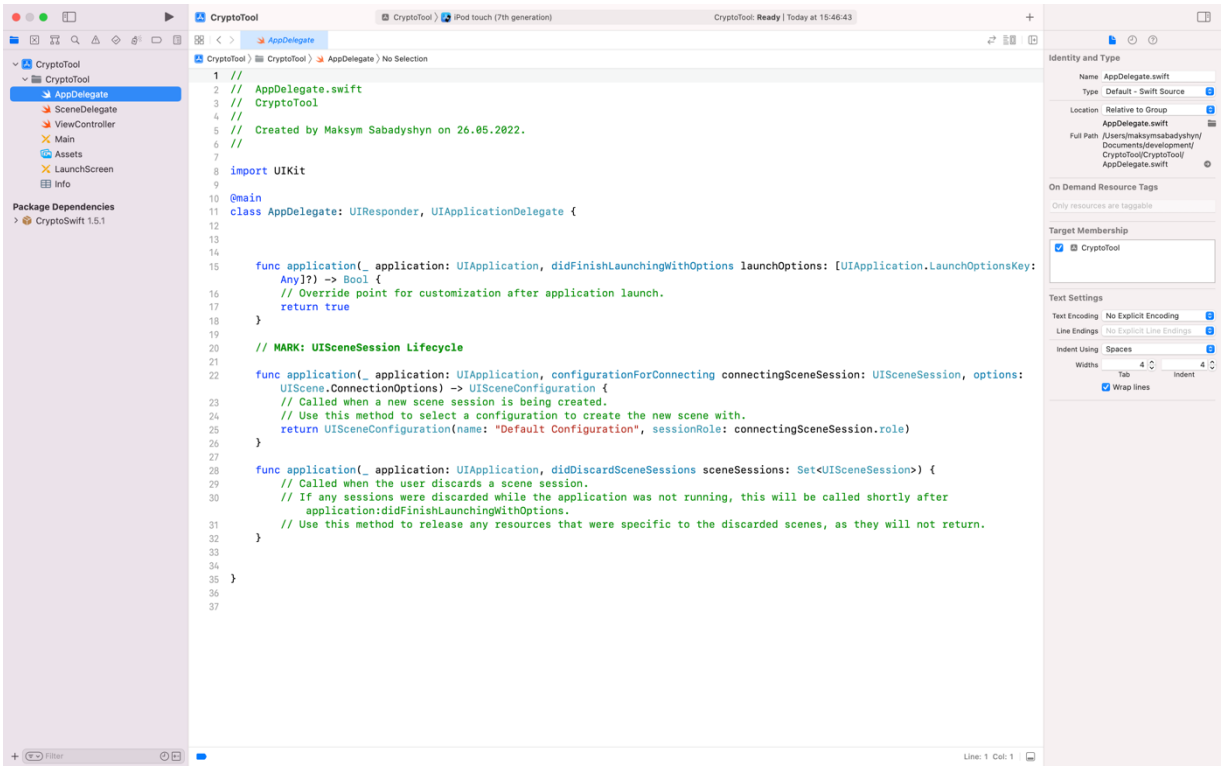

Рисунок 3.2 Інтерфейс XCode

Такий підхід надає зручність в розробці та дозволяє сфокусуватись на робочих моментах. Підсумовуючи обрані для додатку технології маємо, що написаний він за допомогою мови програмування Swift в середовищі інтегрованої розробки XCode.

3.3 Аналіз основних підходів до використання криптографічних алгоритмів в мобільній розробці

В попередніх розділах було розглянуто актуальні в мобільній розробці алгоритми. Задля демонстрації основних інструментів та технологій за допомогою яких надається можливість використовувати згадані популярні криптографічні алгоритми було розроблено додаток.

Незважаючи на існуючу можливість написання власних рішень по криптографічним алгоритмам, часто такий підхід є не оптимальним. Причинами цього є те, що надані вже алгоритми є оптимізованими під систему, а також більш захищеними та досконалими. В той час власне рішення вимагає витрат по часу та зазвичай не виправдовує вкладених ресурсів.

Саме тому існує багато готових фреймворків, що надають можливості криптографічних алгоритмів в мобільних додатках.

На сьогоднішній день існує два найбільш популярних та використовуваних фреймворки для роботи з криптографічними алгоритмами: CryptoKit та CryptoSwift. Обидва мають велику аудиторію користувачів з постійними оновленнями проте між ними існує фундаментальна різниця, що впливає на конкурентноспроможність: CryptoKit – це фреймворк розроблений Apple в той час як CryptoSwift – проект з відкритим вихідним кодом від сторонніх розробників.

CryptoKit є відносно новою розробкою компанії та вперше був представлений на щорічній конференції для розробників WWDC у 2019 році. Він мінімально підтримується з версії iOS 13.0, що на сьогоднішній день встановлена у ~1% користувачів, що дозволяє використовувати фреймворк у готових мобільних застосунках [26].

До CryptoKit вже існували фреймворки по роботі з криптографією, такі як Security та CommonCrypto. Їх основним недоліком було те, що за основу брались бібліотеки написані на C, що в свою чергу ускладнювало роботу з бібліотеками. Перевагами CryptoKit в порівнянні з попередніми способами роботи з криптографією є те, що бібліотека написана з якісним та зрозумілим API, повною підтримкою Swift і відповідним до мови синтаксисом, і додаванням вищого рівня абстракції, що дозволяє працювати з нею тим, хто не суттєво знайомий з криптографією.

У можливості CryptoKit входять хешування даних та порівняння вже існуючих хешів, симетричне та асиметричне шифрування даних. З доступних алгоритмів для хешування є MD5, SHA-1, SHA-2. MD5 та SHA-1 не є рекомендованими і містяться в незахищеному модулі, проте доступні для використання.

З симетричні алгоритмів шифрування можна виділити актуальні сьогодні HMAC, AES-GCM, та ChaCha20-Poly1205. Присутні також імплементації

асиметричних шифрів: Curve25519 та NIST. З недоліків можна відзначити відсутність певних відомих алгоритмів, такі як RSA.

CryptoSwift – це проект з відкритим вихідним кодом, що створений незалежним розробником [27]. Він представляє собою колекцію криптографічних алгоритмів, що написані на Swift.

Незважаючи на те, що проект створений одним розробником, він отримав підтримку від спільноти і продовжує розвиватись. Присутні функції хешування MD5, SHA-1, SHA-2, SHA-3, шифри AES, ChaCha20, та інші.

Враховуючи його популярність він та CryptoKit будуть проаналізовані на предмет наявності важливих криптографічних алгоритмів та досвід роботи з бібліотеками.

Також варто відзначити інший продукт, що надає послуги безпеки – Themis [28]. Themis – це комплексний продукт, що надає високорівневі послуги безпеки протягом життєвого циклу програми. Дана бібліотека, будучи підключеною до проекту, абстрагує розробника від необхідності використовувати криптографічні алгоритми та опрацьовує такі випадки автоматично. Зважаючи на те, що Themis не є бібліотекою криптографічних алгоритмів її не було розглянуто детально.

Процес налаштування роботи з CryptoKit простий, адже він за замовчуванням є присутнім в середовищі розробки. Функціонал бібліотеки стає доступним після імпортування модулю у бажаний файл.

Так як CryptoSwift є стороннім модулем, його необхідно підключати окремо. У розробці під Apple присутні три найпопулярніші менеджерів сторонніх залежностей – CocoaPods, Carthage, Swift Package Manager. У кожного є свої переваги та недоліки проте у рамках даного аналізу було обрано Swift Package Manager, адже він зручно інтегрується в середовище розробки і не вимагає багатьох окремих кроків для підключення залежностей.

Створений з метою аналізу популярних фреймворків проект є простим за суттю. У ньому присутні два класи – CryptoSwiftImpl та CryptoKitImpl, що демонструють використання відповідних алгоритмів. Клас main викликає

відповідні функції. Задля кращого розуміння відмінностей між фреймворками проведено порівняльний аналіз швидкодії.

Розглянемо роботу з CryptoKit. Дана бібліотека надає можливість використання алгоритму ChaCha20Poly, імплементуємо його в коді.

```
func chachaEncrypt(message: String, key: SymmetricKey) -> Data {
    let message = Array(message.utf8)
    return try! ChaChaPoly.seal(message, using: key).combined
}

func chachaDencrypt(sealedBox: Data, key: SymmetricKey) -> String {
    let sealedBox = try! ChaChaPoly.SealedBox (combined: sealedBox)
    let decryptedData = try! ChaChaPoly.open(sealedBox, using: key)
    return String(data: decryptedData, encoding: .utf8) ?? ""
}
```

Рисунок 3.3 Імплементація шифрування та розшифрування в CryptoKit використовуючи ChaCha20Poly

Шифрування є простим на не вимагає додаткових зусиль, незважаючи на рівень захищеності та складності алгоритму, адже CryptoKit приховує дані деталі. Для шифрування необхідно передати попередньо створений ключ та повідомлення, що шифрується, а також викликати функцію ChaChaPoly.seal(message: String, using: SymmetricKey).

Задля розшифрування необхідно отримати зашифровану інформацію, а також ключ, та викликати функцію ChaChaPoly.open(SealedBox, using: SymmetricKey). Отриманий результат повертається у форматі Data, тому необхідно додатково перевести його в String.

CryptoSwift має реалізований звичайний ChaCha20, реалізуємо його.

```
func chachaEncrypt(message: String, passwordKey: String, saltKey: String, iv: Array<UInt8>) -> (Array<UInt8>, Array<UInt8>) {
    let message = Array(message.utf8)

    let password: Array<UInt8> = Array(passwordKey.utf8)
    let salt: Array<UInt8> = Array(saltKey.utf8)

    let key = try! HKDF(password: password, salt: salt, variant: .sha256).calculate()

    return (try! ChaCha20(key: key, iv: iv).encrypt(message), key)
}

func chachaDecrypt(encrypted: Array<UInt8>, key: Array<UInt8>, iv: Array<UInt8>) -> String {
    let decryptedBytes = try! ChaCha20(key: key, iv: iv).decrypt(encrypted)
    let decryptedString = String(bytes: decryptedBytes, encoding: .utf8)
    return decryptedString ?? ""
}
```

Рисунок 3.4 Імплементація шифрування та розшифрування в CryptoSwift використовуючи ChaCha20

Варто відмітити, що у порівнянні з CryptoKit, CryptoSwift потребує більше налаштувань для того аби здійснити бажану операцію. У функцію кодування передається вхідне повідомлення, сіль та пароль, а також вектор ідентичності. Генерується ключ та вже після цього відпрацьовує функція закодування.

Для розкодування необхідно передати зашифрований масив, а також ключ та вектор ідентичності. CryptoSwift повертає масив байтів як результат, проте його можна перевести в String для зручного читання.

Розглянемо також імплементацію алгоритму хешування на обох бібліотеках використовуючи SHA256.

```
func hash(fromValue value: String) -> Data {
    let valueData = value.data(using: .utf8)!
    let hash = SHA256.hash(data: valueData)
    return Data(hash)
}
```

Рисунок 3.5 Реалізація алгоритму хешування використовуючи CryptoKit

Реалізація використовуючи CryptoKit проста: першим кроком йде конвертація у байти і вже потім передача виділеній функції hash(data: Data), яка повертає дайджест, що можна конвертувати в об’єкт Data.

Розглянемо реалізацію за допомогою CryptoSwift.

```
func hash(fromValue value: String) -> Data {
    let calculatedHash = CryptoSwift.SHA2(variant: .sha256).calculate(for: Array(value.utf8))
    return Data(calculatedHash)
}
```

Рисунок 3.6 Реалізація алгоритму хешування використовуючи CryptoSwift

Вихідний код також є зрозумілим і доволі простим. Повертається об’єкт типу Data. Оцінивши зрозумілість синтаксису бачимо, що загалом в обох випадках він не є складним за розумінням чи комплексним за об’ємом.

Порівняємо ефективність роботи двох підходів. Для цього створено окрему функцію, що генерує стрічку заданої довжини випадковим чином. Для цього створено набір допустимих символів і в конструкторі String n раз взято випадковий елемент з набору.

```
func randomString(length: Int) -> String {
  let letters = "0123456789abcdefghijklmnopqrstuvwxyz"
  return String((0..<length).map{ _ in letters.randomElement()! })
}
```

Рисунок 3.7 Генерація випадкових стрічок тексту

Проведемо n ітерацій і оцінимо швидкодію алгоритму, створивши таблицю з результатами. Для поточної задачі було проведено 20 ітерацій. Результати наведено в таблиці 3.1.

	SHA-256			ChaCha20/ChaChaPoly		
	min	avg	max	min	avg	max
CryptoKit	0.383	0.4	0.5	0.377	0.389	0.416
CryptoSwift	1.859	1.9	1.976	2.531	2.646	2.806

Таблиця 3.1 Результати аналізу швидкодії бібліотек

Ознайомившись з таблицею спостерігаємо значну різницю у швидкодії між CryptoKit та CryptoSwift з перевагою бібліотеки від Apple. Така різниця пояснюється тим, що Apple оптимізують алгоритми під залізо, що використовується системою, а також має більше ресурсів на їх покращення.

Також варто відзначити, що важливий на сьогодні популярний алгоритм асиметричного шифрування Curve25519 не є доступний у CryptoSwift, проте є реалізований розробниками CryptoKit, що є додатковою перевагою у використанні. З точки зору мобільної розробки алгоритм використовується для цифрових підписів. Приведемо реалізацію даного алгоритму у рамках створеної програми для аналізу фреймворків.

```swift
func signData(message: String) -> (Data, Curve25519.Signing.PublicKey) {
    let privateKey = Curve25519.Signing.PrivateKey()
    let publicKey = privateKey.publicKey

    let spk = try! Curve25519.Signing.PublicKey(rawRepresentation: publicKey.rawRepresentation)

    let data = message.data(using: .utf8)!
    let signature = try! privateKey.signature(for: data)

    return (signature, spk)
}

func checkDataSignature(message: String, signature: Data, publicKey: Curve25519.Signing.PublicKey) {
    let messageData = message.data(using: .utf8)!

    if publicKey.isValidSignature(signature, for: messageData) {
        print("This message has valid signature.")
    }
}
```

Рисунок 3.8 Реалізація Curve25519 за допомогою CryptoKit

Для підпису створюється приватний ключ, з нього створюємо публічний, і генеруємо підпис, який повертаємо як результат функції разом з ключем. Для перевірки підпису використовуємо існуючу функцію isValidSignature(signature: Data, for: String).

Враховуючи складність криптографії на еліптичних кривих присутність такого функціоналу допомагає зменшити необхідну на розробку кількість ресурсів.

Отже, і CryptoKit і CryptoSwift мають свою аудиторію проте проводячи порівняння двох бібліотек бачимо, що CryptoKit виграє у наявності більшої кількості алгоритмів та швидкодії.

3.4 Висновки до розділу 3

В третьому розділі було оглянуто існуючі приклади вирішення криптографічних проблем в комерційних додатках, а саме Bear та WhatsApp.

Також було оглянуто актуальні для розробки технології та інструменти, досліджено популярність мови Swift та переваги XCode як середовища інтегрованої розробки.

В кінці розділу було оглянуто два найпопулярніших підходи до роботи з криптографічними алгоритмами: CryptoKit та CryptoSwift. Після огляду обох фреймворків була написана допоміжна програма, що демонструвала приклади реалізації алгоритмів, а також вимірювала швидкодію.

В результатах аналізу виявлено, що CryptoKit є зручнішим та більш швидким інструментом у роботі з криптографічними алгоритмами при розробці мобільних додатків під iOS.

Висновки по роботі

В роботі було досліджено причини актуальності роботи з криптографією сьогодні серед яких виділено те, що за час пандемії кількість кіберзлочинів стрімко зросла разом з середньою вартістю збитків для компанії.

Також оглянуто тенденції до створення компаніями власних стратегій кодування та зберігання зашифрованої інформації, що мають позитивний тренд на збільшення таких практик в індустрії.

Враховуючи важливість недопущення витоків інформації, а також прямого впливу на них криптографії, в роботі було досліджено основні причини відомих витоків даних. Знайдено, що це траплялось в основному через ігнорування компаніями сертифікатів безпеки, а також зберігання та надсилання інформації в не зашифрованому вигляді.

Проаналізовано актуальність криптографії у мобільних застосунках, оглянуто проблеми, які вона вирішує. Такі проблеми як автентифікація та авторизація користувача, зберігання конфіденційності його даних, та цифрові підписи є основними задачами, які вирішуються за допомогою криптографічних алгоритмів. Задачі в сфері банкінгу, криптовалюти, а також комунікація з такими сторонніми сервісами як бекенд є важливими точками використання шифрування та зашифрованого зв’язку.

Також було оглянуто теоретичну складову криптографії як науки. Розглянуто та пояснено основні концепти так як типи криптографії. Виділено три основні: алгоритми з відкритим ключем, алгоритми з приватним ключем, та хеш функції.

Розглянуто актуальні алгоритми у розробці, а також нові підходи, що не є у широкому використанні, як NeuralHash від Apple.

У роботі також проаналізовано дві найпопулярніші бібліотеки для роботи з криптографічними алгоритмами – CryptoKit від Apple та CryptoSwift від сторонніх розробників.

Написано програмний застосунок, шо порівнює легкість використання кожної з бібліотек, а також перевіряє швидкодію на алгоритмі хешування та шифрування даних.

Серед проблем що виникали в процесі була важкість знайти відкриту інформацію про практики криптографії на комерційному ринку. Дана проблема була вирішена після зміни вектору пошуку, який змістився на пошук компаній, які надають такі послуги і діляться своїми дослідженнями.

В результаті роботи досліджено, що важливість криптографії для мобільних застосунків під управлінням iOS є великою, а багато з компаній все ще ігнорують базові заходи безпеки в коді. Це в свою чергу призводить до витоків даних і зменшення довіри користувачів.

Кінцевий результат роботи також демонструє, що на iOS сьогодні існує якісна база бібліотек, які надають вищий рівень абстракції для роботи з криптографічними алгоритмами. Після аналізу криптографічних алгоритмів в роботі також виділяємо найпопулярніші з них в кожному сегменті: SHA-2 для хешування даних, ChaCha20Poly серед алгоритмів з приватним ключем, та Curve25519 як алгоритм шифрування з відкритим ключем.

Робота показує важливість подальших досліджень в галузі з перспективи мобільної розробки та створює фундамент для майбутніх досліджень.

Список джерел

1. Smartphone users 2026 | Statista [Електронний ресурс] // Statista. – Режим доступу: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. – Назва з екрана.
2. Smartphone users 2026 | Statista [Електронний ресурс] // Statista. – Режим доступу: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/. – Назва з екрана.
3. Smartphone average selling price worldwide 2016-2021 | Statista [Електронний ресурс] // Statista. – Режим доступу: https://www.statista.com/statistics/788557/global-average-selling-price-smartphones/. – Назва з екрана.
4. Number of apps from the Apple App Store 2022 | Statista [Електронний ресурс] // Statista. – Режим доступу: https://www.statista.com/statistics/268251/number-of-apps-in-the-itunes-app-store-since-2008/. – Назва з екрана.
5. Дія – Державні послуги онлайн [Електронний ресурс] // Державні послуги онлайн | Дія. – Режим доступу: https://diia.gov.ua. – Назва з екрана.
6. 2021 Global Encryption Trends Study [Електронний ресурс] // Entrust. – Режим доступу: https://www.entrust.com/lp/en/global-encryption-trends-study. – Назва з екрана.
7. 2022 Cyber Security Statistics Trends & Data [Електронний ресурс] // PurpleSec. – Режим доступу: https://purplesec.us/resources/cyber-security-statistics/. – Назва з екрана.
8. Cost of a Data Breach Report 2021 [Електронний ресурс] // IBM - Deutschland | IBM. – Режим доступу: https://www.ibm.com/security/data-breach. – Назва з екрана.

9. Home | Ponemon Institute [Електронний ресурс] // Ponemon Institute. – Режим доступу: https://www.ponemon.org. – Назва з екрана.

10. 2021 Annual Data Breach Report [Електронний ресурс] // Identity Theft | Resource Center. – Режим доступу: https://www.idtheftcenter.org/publication/2021-annual-data-breach-report-2/. – Назва з екрана.
11. Jason T. A Case Study Analysis of the Equifax Data Breach 1 A Case Study Analysis of the Equifax Data Breach [Електронний ресурс] / Thomas Jason. – 2019. – Режим доступу: https://doi.org/10.13140/RG.2.2.16468.76161.. – Назва з екрана.
12. US Marines Confirm 21,000 Details Exposed in Data Breach [Електронний ресурс] // Infosecurity Magazine. – Режим доступу: https://www.infosecurity-magazine.com/news/marines-21000-breach/. – Назва з екрана.
13. NQ Mobile Vault™ Mobile App | The Best Mobile App Awards [Електронний ресурс] // The Best Mobile App Awards | Recognizing the best Android, iPhone, iPad and Windows Mobile Apps. – Режим доступу: https://bestmobileappawards.com/app-submission/nq-mobile-vault-2. – Назва з екрана.
14. What is Digital Signature: How it works, Benefits, Objectives, Concept - Employee Onboarding [Електронний ресурс] // Employee Onboarding. – Режим доступу: http://www.emptrust.com/blog/benefits-of-using-digital-signatures/. – Назва з екрана.
15. P P. F. A. Kerckhoffs' Principle [Електронний ресурс] / Petitcolas Fabien A. P // SpringerLink. – Режим доступу: https://link.springer.com/referenceworkentry/10.1007/978-1-4419-5906-5_487. – Назва з екрана.
16. Symmetric key encryption - Wikimedia Commons [Електронний ресурс] // Wikimedia Commons. – Режим

доступу: https://commons.wikimedia.org/wiki/File:Symmetric_key_encryption.svg. – Назва з екрана.

17. A survey of lightweight stream ciphers for embedded systems [Електронний ресурс] / Harry Manifavas [та ін.]. – Режим доступу: https://www.researchgate.net/publication/291012336_A_survey_of_lightweight_stream_ciphers_for_embedded_systems. – Назва з екрана.
18. Handbook of Applied Cryptography [Електронний ресурс] / A. Menezes, P. van Oorschot, S. Vanstone // Chapter 8. – [Б. м.], 1996. – Режим доступу: https://www.garykessler.net/library/crypto/hac_chap08.pdf. – Назва з екрана.
19. Lim S. K. Apple's NeuralHash – How it works and how it might be compromised [Електронний ресурс] / Swee Kiat Lim. – Режим доступу: https://towardsdatascience.com/apples-neuralhash-how-it-works-and-ways-to-break-it-577d1edc9838. – Назва з екрана.
20. Fadele A. Open Access ROBUST DATA SECURITY FRAMEWORK FOR IoT NETWORK USING INTEGRATED TECHNIQUES / Alaba Fadele, Abayomi Jegede, Christopher Eke // International Journal of Applied Mathematics and Machine Learning. – 2020.
21. Lutkevich B. What is a Nonce? - Cryptographic Nonce from SearchSecurity [Електронний ресурс] / Ben Lutkevich // SearchSecurity. – Режим доступу: https://www.techtarget.com/searchsecurity/definition/nonce. – Назва з екрана.
22. AbdAllah E. G. Advanced Encryption Standard New Instructions (AES-NI) Analysis: Security, Performance, and Power Consumption [Електронний ресурс] / Eslam G. AbdAllah, Yu Rang Kuang, Changcheng Huang // ICCAE 2020: Proceedings of the 2020 12th International Conference on Computer and Automation Engineering. – 2020. – С. 167–172. – Режим доступу: https://doi.org/10.1145/3384613.3384648. – Назва з екрана.
23. Apple Developer Documentation [Електронний ресурс] // Apple Developer. – Режим

доступу: https://developer.apple.com/documentation/security/keychain_services. – Назва з екрана.

24. WhatsApp. WhatsApp Encryption Overview [Електронний ресурс] / WhatsApp. – 2016. – № 6. – Режим доступу: https://scontent.fiev25-1.fna.fbcdn.net/v/t39.8562-6/271639644_1080641699441889_2201546141855802968_n.pdf?_nc_cat=108&ccb=1-7&_nc_sid=ad8a9d&_nc_ohc=O2F6mAPL1mMAX-X7GHx&_nc_ht=scontent.fiev25-1.fna&oh=00_AT8qO7Qn2UNOoX2fESzaaSzXEAX44yAK_HuyLNbgT_Xn1A&oe=62A53C3D. – Назва з екрана.
25. TIOBE Index - TIOBE [Електронний ресурс] // TIOBE. – Режим доступу: https://www.tiobe.com/tiobe-index/. – Назва з екрана.
26. App Store - Support - Apple Developer [Електронний ресурс] // Apple Developer. – Режим доступу: https://developer.apple.com/support/app-store/. – Назва з екрана.
27. CryptoSwift [Електронний ресурс] // CryptoSwift. – Режим доступу: https://cryptoswift.io. – Назва з екрана.
28. Cossack Labs. Themis â cross-platform cryptographic library [Електронний ресурс] / Cossack Labs // Cossack Labs. – Режим доступу: http://www.cossacklabs.com/themis. – Назва з екрана.