МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ С.В. Казмірчук

«_____»_____ 20__ р.

На правах рукопису
УДК 004.056.5:510.22(043.3)

МАГІСТЕРСЬКА АТЕСТАЦІЙНА РОБОТА

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Криптографічний модуль захисту повідомлень в середовищі ОС Android

Автор: П.Ю. Геменчук

Науковий керівник: к.т.н., доцент А.В. Ільєнко

Нормоконтролер: асистент С.В. Єгоров

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Навчально-науковий інститут: інноваційних освітніх технологій

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Магістр

Спеціальність: 125 «Кібербезпека»

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

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ С.В. Казмірчук

«__»_____ 20__ р.

ЗАВДАННЯ

на виконання магістерської атестаційної роботи

магістранта Геменчука Петра Юрійовича

2. Тема: «*Криптографічний модуль захисту повідомлень в середовищі ОС Android*» затверджена наказом ректора від «__» _____ 20__ № _____/ст.

3. Термін виконання з 25.11.2019 р. по 26.02.2020 р.

4. Вихідні дані: проаналізувати існуючи засоби захисту інформації сучасних ОС; провести дослідження ефективності криптографічних методів захисту інформації; розробити модуль криптографічного захисту інформації задля імплементації послуги цілісності та конфеденційності інформації;

Зміст пояснювальної записки: аналіз існуючих реалізацій захисту інформації в різних її станах; аналіз засобів захисту сучасних Unix-подібних ОС; аналіз існуючих методів криптографічного захисту; розробка криптографічного модуля захисту повідомлень в середовищі ОС Android; опис архітектури ПЗ, схеми криптографічного захисту та її складових; тестування основної частини функціональності ПЗ.

КАЛЕНДАРНИЙ ПЛАН

виконання магістерської роботи

№ п/п	Етапи виконання магістерської роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	25.11.2019	*Виконано*
2.	Аналіз літературних джерел	01.12.2019	*Виконано*
3.	Обґрунтування вибору рішення	10.12.2019	*Виконано*
4.	Збір інформації	15.12.2019	*Виконано*
5.	Виконання 1-го розділу АР	28.12.2019	*Виконано*
6.	Виконання 2-го розділу АР	16.01.2020	*Виконано*
7.	Програмна реалізація	25.01.2020	*Виконано*
8.	Тестування розробленого програмного забезпечення	31.01.2020	*Виконано*
9.	Оформлення і друк пояснювальної записки		*Виконано*
10.	Оформлення презентації		*Виконано*
11.	Отримання рецензій від рецензента		*Виконано*
12.	Захист в ЕК	26.02.2020	*Виконано*

Магістрант П. Геменчук

(підпис, дата)

Науковий керівник А. Ільєнко

(підпис, дата)

РЕФЕРАТ

Магістерська атестаційна робота складається зі вступу, трьох розділів, загальних висновків, списку використаних джерел, додатків і має 107 сторінок основного тексту, 6 рисунків, 2 таблиці, 19 сторінок додатків. Список використаних джерел містить 27 найменувань і займає 3 сторінки. Загальний обсяг роботи 126 сторінок.

Мета роботи — програмна реалізація криптографічного модулю захисту повідомлень у середовищі ОС Android.

Об’єктом дослідження є процедура шифрування і дешифрування та процедура формування і верифікації ЕЦП.

Предметом дослідження є криптографічні методи, алгоритмічні та математичні підходи функціонування криптографічних методів.

Метод дослідження. Проведені дослідження базуються на сучасних методах теорії побудови захищених інформаційних мереж та криптографічних методах забезпечення цілісності і конфеденційності інформації.

Завданням даної роботи є: дослідження відомих засобів захисту ОС Android; дослідження криптографічних методів захисту інформації та практичні підходи забезпечення цілісності та конфеденційності інформації; розробка програмної реалізації модулю контролю захисту повідомлень у середовищі ОС Android; проведення тестування роботи криптографічного модуля та обґрунтувати доцільність його використання щодо захисту повідомлень у середовищі ОС Android.

Наукова новизна обумовлена вирішенням задачі щодо захисту інформації, зокрема інформації, що зберігається, оброблюється та передається застосовуючи сучасні комунікаційні засоби, з забезпеченням цілісності та конфеденційності криптографічними засобами, зокрема використання комбінованої криптосистеми з алгоритмами ЕС та AES. Практичне значення роботи полягає в створенні удосконаленого криптографічного модуля захисту повідомлень у середовищі ОС Android мовою програмування Java.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

AES — Advanced Encryption Standart

API — Application Programming Interface

ASN.1 — Abstract Syntax Notation One

CMS — Cryptographic Message Syntax

CRL — Certificate Revocation List

CRC — Cyclic Redundancy Check

DAC — Discrete Access Control

DES — Data Encryption Standart

DLP — Discrete Logariphm Problem

DOS — Denial of Service

ECDSA — Ecliptic Curve Digital Signature Algorithm

IoT — Internet Of Things

IP — internel protocol

IPC — Inter Process Communication

JSON — JavaScript Object Notation

MAC — Mandatory Access Control

MITM — The Man in the middle

NFC — Near Field Communication

NoSQL — Non Structured Query Language

OCSP — Online Certificate Status Protocol

PKCS — Public Key Cryptography Standarts

PKI — Public Key Infrastructure

RFC — Request For Comments

RPC — remote procedure call

RSA — Rivest Shamir Adleman

SQL — Sqtructured Query Language

TCP — transmission control protocol

TSP — Timestamp Protocol

UEFI — Unified Extensible Firmware Interface

UDP — User datagram Protocol

UI — User Interface

UTF — Unicode Transform Format

АЦП — аналогово-цифровий перетворювач

ІБ — інформаційна безпека

МП — мова програмування

ПЗ — програмне забезпечення

ОП — оперативна пам’ять

ОС — операційна система

РЗЛЗЗ — Регістр зсуву з лінійним зворотнім зв’язком

ФС — файлова система

ЦАП — цифрово-аналоговий перетворювач

ВСТУП

Сучасне інформаційне суспільство, що формується внаслідок бурхливого розвитку науки, техніки та технологій, здобуває властиві риси глибоких знань, високої динаміки розвитку, передового виробництва, усебічного розвитку особистості. Разом з появою нових технологій пропорційно змінюється суспільне життя окремих людей — з’являються сучасні побутові прилади, телевізори, комп’ютери, мобільні пристрої, IoT пристрої тощо. Окрему роль, як найвпливовішого елемента на повсякденне життя сучасної людини, мають мобільні пристрої, які, з сучасними версіями ПЗ, можуть замінювати будь-яке десктопне ПЗ та виконувати, будь-які необхідні окремому користувачу, функції.

Все розмаїття цих пристроїв має програмну та аппаратну складову, що взаємодіють між собою та потребують ретельного інформаційного захисту в різних станах своєї роботи. Тому захист інформації - забезпечення конфеденційності, цілісності, доступності, є ключовим моментом для забезпечення безпечної взаємодії з насиченим інформаційним простором.

Внутрішній аппаратний захист пристрою як правило реалізовується розробником пристрою, що вмотивований захистом своєї репутації, та захищеним рівнем роботи процессора і підтримкою його роботи в ядрі поточної ОС інформаційної системи. Рівень захисту режиму багатокористувацької системи забезпечується поточною ОС, над якою працює багато розробників. Найвразливішою ланкою є мережева взаємодія пристроїв, оскільки в перших двох випадках зловмиснику необхідний локальний доступ до системи, а у випадку з мережею зловмиснику достатньо контролювати(пасивно сніфати) будь-яку транзитну точку, або взаємодіяти на канальному рівні. Зважаючи на значимість, захист мережевих комунікацій має одну з найважливіших ролей в кібербезпеці.

Мессенджери - ПЗ за допомогою яких можна обмінюватися зручними швидкими та, як правило, короткими повідомленнями між користувачами. Комерційного ПЗ, яке виконує функції мессенджерів багато, серед яких Viber,

FB-Messenger, Skype, Telegram, тощо. Всі вони крім своїх безпосередніх функцій виконують функції закладені розробником згідно його бізнесу, насамперед фінансової його складової, на яку згоджуються користувачі використовуючи ці месенджери. Наприклад збирають інформацію про місцеположення користувача, мають можливість читати їхні повідомлення та передавати їх спецслужбам, сканувати вподобання з контексту повідомлень задля побудови корегентного рекламного контенту та його подальшого продажу, тренування своїх нейронних мереж, продажу персональних даних користувачів тощо. Забезпечуючи цілісність та доступність в рамках свого сервісу, вони, через фінансові інтереси, не забезпечують в повній мірі конфеденційнійності.

У світі мобільних платформ існують дві найпопулярніші ОС - Android та IOS. Обидві підтримуються комерційними розробниками, Google та Apple відповідно. Обидва розробника забезпечують гідний рівень захисту додатків. Зокрема Android використовує MAC, ccgroups, sandbox, права для нав’язування взаємодії об’єктів з суб’єктами, розмежування ресурсів, приватно-доступного простору ФС, розмежовування прав тощо.

ПЗ яке є лейтмотивом цієї роботи використовує наступні криптографічні інструменти: комбіновану (симетричну та асиметричну) криптографію для шифрування та підписування користувацьких повідомлень; симетричну криптографію як допоміжну задля шифрування приватних ключів в локальній БД SQLite пристрою; хешування та «підсолення» паролю локального користувача з якого утворюється ключ для симетричного шифрування полів з приватними ключами в БД локального пристрою; захищений генератор псевдовипадкових чисел тощо. Для опису абстрактного синтаксису даних використовуються мова ASN.1, криптограми пакуються згідно стандартів PCKS#7, та його підмножини CMS, що описує стандартизовану структуру, яка в свою чергу поєднує захищені дані та відомості, які потрібні для корректної роботи з ними. Серед додаткових відомостей, що додаються в CMS-контейнери можна окремо виділити наступні: інформація про алгоритм хешування та

підпису; інформація про алгоритм шифрування та його отримувачів, які скориставшись своїми публічними ключами зможуть розшифрувати симетричні ключі для розшифровки контенту повідомлення, сертифікати користувачів(підписувачів), додаткові ланцюги сертифікатів (наприклад включення кореневих сертифікатів), спискі відізваних сертифікатів. В криптоконтейнерах з ПЗ фігурують два типи CMS: Enveloped Data та Signed Data для забезпечення можливості шифрування повідомлення та верифікації віддаленої сторони стороною що є приймачем повідомлення відповідно.

Централізованого сервера який ідентифікує користувачів та зберігає інформацію про повідомлення немає. Замість цього використовується сервіс Google Firebase який, якщо використовується в додатку, персоналізує окремий мобільний пристрій через FirebaseToken та дає можливість для мобільного додатку створити RealTime Database, яка є NoSQL базою і зберігає дані в форматі JSON та може слугувати додатку сховищем даних — тим самим в поточній архітектурі ПЗ забезпечує псевдо-централізовані функції. Мережевою взаємодією між додатками є подійна модель, яку впроваджує хмарна БД Firebase. Коли пристрій с додатком опиняється в мережі, виникає подія і пристрій може встановити зв“язок з хмарною БД. Побудована схема хмарної БД передбачає створення в хмарній БД гілки(шляху в БД) з персоніфікованими ідентифікаторами (FirebaseToken) пристроїв. Коли інший додаток пише повідомлення він кладе криптоконтейнер з даними в гілку з відомим йому ідентифікатором FirebaseToken іншої сторони, яку додаток знає з X509 контейнера(з поля Subject Alternative Name). Таким чином додаток напряму не створює сокетів для виконання мережевих операцій, не відкриває TCP або UDP порти для прослуховування, тощо.

Мотивом вибору тематики дипломної роботи є реалізація забезпечення в якомога повніший мірі цілісності, конфеденційності та доступності повідомлень користувачів, без впливу фінансового чинника, крім того з псевдо-централізацією.

РОЗДІЛ 1. СУЧАСНИЙ СТАН ЗАБЕЗПЕЧЕННЯ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ

1.1 Аналіз поточного стану забезпечення захисту інформації

Термін «інформація» (від лат. informatio — роз“яснення) в самому широкому сенсі означає відомості про об“єкти та явища навколишнього світу, їхні параметри, властивості або стани, які можуть сприймати як суб“єктивні субстанції(біологічні організми) так і об“єктивні(керуючі машини) в процесі своєї життєдіяльності або роботи. Зважаючи на складність природи біологічних механізмів та їх фізико-хімічних характеристик, інформація, в контексті розгляду суб“єктивними інформаційними приймачами, є складним об“єктом, якому притаманнє функціонування усвідомлення, обміну, перетворення, зберігання, оновлення, накопичення, фільтрування та маніпуляції з її властивостями відносно особливостей суб“єкту. Для об“єктивних приймачів інформація є об“єктом з яким вони маніпулюють відносно закладених в їхне функціонування алгоритмів та цілей, які були поставлені при закладанні цих алгоритмів, а також впливів, що несе функціонування об“єктивного приймача інформації на інші компоненти та зовнішню систему в складі якої він працює. Для суб“єктивних приймачів інформації сприйняття інформації є комфортним в аналоговій формі, та частково в цифровій формі. Для об“єктивних приймачів інформації сприйняття інформації відбувається тільки в цифровій формі. В залежності від критеріїв інформація може класифікуватися: за способом сприйняття, за областью виникнення, за формою надання, за призначенням, тощо. В залежності від способу надання інформація розмежовується на візуальну, аудиальну, смакову, тактильну тощо. За областью виникнення інформацію можна поділити на: елементарну, біологічну, соціальну тощо. За формою надання інформація може бути: текстовою, машинною, графічною, числовою, звуковою тощо. За критерієм призначення інформація може бути поділена на: спеціальну, масову, особисту, статистичну, соціальну, тощо.

Найважливішими властивостями інформації для будь-яких приймачів інформації є об“єктивність, достовірність, повнота, точність, актуальність, цінність, зрозумілість, адекватність, новизна. Об“єктивністю інформації можна вважати міру залежності її від суб“єктивної складової та психологічних особливостей субстанції яка її несе. Тобто об“єктивною є інформація, що не залежить від будь-якої думки або розсуду та несе об“єктивну дійсність,. Достовірністю інформації можна вважати показник якості її повноти та загальної точності, відображення інформацією істинного стану речей, з врахуванням наступних критеріїв: відсутність неправдивих або спотворенних даних, зменшення значення реальних фактів, низька ймовірність помилкового застосування одиниць даних(букви, символа, біта тощо). Повнота інформації — це міра достатності отриманних даних, на основі яких можливо прийняти якесь рішення. Точність інформації відображається ступенем її близькості до реального стану об“єкта, процесса, явища тощо. Актуальність інформації відображає ступінь відповідності інформації щодо поточного часу. Цінність інформації — це міра пристосування інформації до потреб конкретних випадків та полягає в множині задач, за допомогою якої їх можно вирішити. Зрозумілість інформація проявляється в мірі, наскільки адресанти інформації сприймають форму її вираження. Адекватність інформації — це міра відповідності інформаціїї очікуваному вмісту. Новизна інформації — це властивість інформації, коли її ентропія має позитивне значення для отримувача інформації, або ймовірність виникнення непердбачуванності інформації, що збільшує інформативність інформації. Властивості інформації можуть комбінуватися по відношенню до інших, обопільно доповнюватися або суперечити іншим: чим більше інформація об“єктивна тим вона і достовірніша; неповна інформація не може бути недостовірною; достовірність, актуальність, повнота — набір, що гарантує цінність; адекватність — властивість схожа с об“єктивністю та достовірністю.

З інформацією, в незалежності від її виду, можна виконувати різні операції, основними з яких є: захист, збір та накопичення, фільтрація та перетворення. Збір і накопичення інформації проводиться з метою збільшення значень її властивостей, насамперед, забезпечення її повноти, достовірності та актуальності. Фільтрація інформації проводиться для відсіювання зайвої інформації, наприклад коли інформація не володіє в достатній мірі своїми основними властивостями — достовірністю та повнотою. Перетворення інформації проводиться у випадках коли, при деяких обставинах, необхідно змінити спосіб її надання, виду, форми тощо. Наприклад, оскільки світові явища мають аналогову природу, а комп“ютерізований простір потребує цифрову форму, з“являється необхідність перетворення аналогової форми інформації в цифрову, де виникає необхідність підбирання оптимального для окремого випадку значення частоти дискретизації, яке вагомо впливає на точність інформації. Пристрої, що виконують перетворення аналогової форми інформації в цифрову називаються АЦП(Аналогово-цифровий перетворювач). Оптимальне значення частоти дискретизації, а з ним необхідна точність інформації, може бути знайдено через теорему Котельнікова. Для зворотньої операції використовуються ЦАП(цифрово-аналоговий перетворювач).

Сучасна економічна наука розподіляє всю історію людства на три етапи: доіндустріальне суспільство(феодальне та рабовласницьке); індустріальне(капіталістичне); постіндустріальне суспільство. В доіндустріальному суспільстві владу мали власники землі, в індустріальному суспільстві влада належала власникам виробництва(заводів, фабрик) , в постіндустріальному — владу мають ті хто володіє інформацією. Тому інформація для сьогодення є найціннішим та найкоштовнішим ресурсом, що виступає в ролі основного важіля для розвитку суспільства, а інформаційні системи та технології є засобом для збільшення продуктивності та ефективності праці людей. Зважаючи на сьогоденну роль інформації, необхідність її захисту з часу появи перших комп“ютерів і комп“ютерних

технологій та мереж, залишається однією з найактуальніших проблем сучасності. Кожен користувач будь-якого комп“ютерізованого пристрою — гаджета, десктопного або лептопного комп“ютера, пристрою IoT, смартфону — так чи інакше стикається з цією проблемою. Використання смарт-карт для спрошення транзакцій банківських установ, компьютерізація автомобілів, експлуатація «розумних будинків» - короткий список того, що потребує ретельної скрупульозної методики щодо захисту інформації. Внаслідок комп“ютерізації інфраструктури багатьох державних установ та установ міжнародного рівня в різних країнах, проблематика захисту інформації набуває загальнонаціонального та міжнародного масштабу. Отже в сьогоденні гострою залишається проблематика впровадження ефективних методів щодо захисту персональних даних, державної, військової, комерційної інформації. А вістя методик захисту інформації, насамперед, повинно ефективно сприяти на запобіжні заходи щодо інформації — її втрати, неавторизованого використання та модифікації, і впроваджувати по відношенню до інформації, що обробляється властивості: конфеденційність, цілісність та достовірность. Конфеденційність можна розглядати як статус, що приєднується до інформації та підкреслює її особливу, в плані, захисту роль. Конфеденційна інформація повинна бути відома тільки допущеним та авторизованим суб“єктам системи(користувачам, та їхнім процесам), для інших суб“єктів системи ця інформація повинна бути невідомою. Цілістність інформації — властивість інформації,що проявляється в можливості збереження інформацією своєї структури змісту під час передачи та зберігання.

Стан інформації можна розглядати як її властивість з деякої множини, завдяки якій до інформації може бути застосований додатковий абстрактний функціонал. Кожен з цих станів інкапсулює в собі деяку функціональну процедуру, результатом якої є поява у інформації відповідної властивості. Процедури, що породжують властивості станів інформації та мають надважливе значення класифікуються наступним чином: зберігання, обробка та передача

інформації. Зберігання інформації — це процедура, внаслідок якої інформація має змогу утримувати свій поточний стан. Обробка інформації представляє собою процес інтерпретації(усвідомлення) інформації з можливою зміною її змісту або форми. Передача інформації — це процедура, внаслідок якої інформація може бути відтворена на іншій стороні, що формально називається приймачем. Захист інформації повинен бути оболонкою, що впроваджує оптимальні умови проходження всіх інформаційних процесів, що породжують властивості станів. Засади щодо ефективності захисту інформації повинні ґрунтуватися на сучасних методиках, а реалізація — відповідати вимогам окремих випадків.

Форма інформації впливає на складність процесу зберігання інформації. Цифрову інформацію зберігати набагато легше оскільки вона є послідомністю цифр, яка, в двійниковій системі(0 та 1), може бути закодована будь-яким прийнятним чином через використання: електронних триггерів, полярності намагнічування, каскада перемикачів, напівпровідникових схем тощо. Зберігання ж аналогової інформації є складним процесом, що використовує допоміжні електричні схеми на основі конденсаторів, що схематично виконанні під окремий випадок використання. Зберігання інформації на самих низьких рівням, як правило, пов“язане з спеціальними пристроями — носіями інформації. Носіями інформації можуть бути: кеш процессора, оперативна пам“ять, жорсткі накопичувачі(флеш-накопичувачи), енергонезалежна пам“ять (NVRAM), тощо. Для архітектури ПК, кожен з цих носіїв інформації має різне призначення та характеристикі. Кеш процессора є недовготривалою швидкісною пам“ятю, що виступає посередником між регістрами процессора та ОП, через яку проходить вся інформація, що перебуває в обробці, оскільки архітектура компьютерів та набір її команд для процессора під час переміщення інформації передбачає обов“язковим операндом регістр, тобто транзакцію переміщення ОП — ОП за одну команду процессора здійснити неможливо. ОП — недовготривалий тип пам“яті, в якому як правило, знаходяться:

функціональні та службові частини ОС — планувальник, диспетчер системних викликів, система для забезпечення файлового вводу/виведення, диспетчер-аллокатор ОП, мережева підсистема(як правило TCP/IP), інтерфейси до мережевих пристроїв, численні драйвери пристроїв, різноманітні кеши, обробники хардварних та програмних переривань; контекст процесів, що виконуються в ОС — збережений стан регістрів, відкриті ресурси(файли, сокети, IPC, пристрої), стани потоків процесу, вказівник на останню виконану інструкцію процессором для процеса, тощо. В реальному режимі роботи процессора оперативна пам“ять є простим массивом, з яким процес працює сегментно. В захищеному режимі — вона має сторінкову структуру, з аллокацією за участю процессора та його аппаратної частини MMU(Memory Management Unit). Жорсткі накопичувачі — це довготривалий тип пам“яті, який зберігає левову частку користувацьких даних та службових файлів ОС. Для ядра ОС жорсткий накопичувач — це пристрій енергонезалежної пам“яті, ввід/виведення з яким відбувається частинами деякого фіксованного значення(блоками), через драйвери файлових систем, що впроваджують деяку логіку щодо розподілу доступного простору накопичувача. NVRAM — це узагальнений тип енергонезалежної довготривалої пам“яті, що використовується у світі мікроархітектур(мікроконтролерів, смарткарт, тощо), та, як правило, є частиною формфактора мікроархітектури.

В залежності від носія інформації можуть бути використані різні методики захисту інформації. Модулі ОП можуть мати вбудовану аппаратну коррекцію помилок(ECC — error correction code), що досягається додаванням збиткових бітів до корисної інформації(наприклад коди Хеммінга). Захист інформації на жорстких накопичувачах може бути впроваджений аппаратно(аппаратне шифрування), але частіше є програмним, що впроваджується драйвером файлової системи. Логіка драйверів файлових систем щодо захисту інформації може використовувати: журналювання подій(для відтворення неуспішних транзакцій вводу/виведення); перевірку

цілісності контенту файлів та їхніх метаданних(CRC(Cyclic Redundancy Check), хешування, тощо); захищення службової інформації самого драйвера; самотестування щодо корректної роботи; маркування ущкоджених блоків, тощо. В додаток до вищезгаданого функціоналу, для механічних жорских дисків драйверу треба по можливості мінімізовувати вектор обертання шпинделя для збільшення тривалості експлуатації пристрою. Програмний захист жорсткого накопичувача може включати в себе шифрування на рівні файлової системи та на рівні блокового пристрою. Для збільшення параметра доступності інформації може: використовуватися аппаратний NAS(Network Attached Storage); побудовані різні рівні RAID-массиву(Redundant Array of Independent Disk) як аппаратно так і програмно; використовуватися кластерна файлова система. Пам“ять NVRAM слугує накопичувачем для мікроархітектури та може зберігати службову та конфеденційну інформацію взалежності від призначення пристрою. Оскільки вона є частиною самого мікропристрою — доступ до неї обмежений аппаратними засобами. Наприклад, сучасний електроний паспорт є смарт-картою з RISC-процессором та захищенною NVRAM пам“яті, в якій зберігаються приватні ключі користувача, та яку неможливо здублювати фізично. Процессорні пристрої пам“яті, як правило, не мають захисту інформації оскільки мають службову роль та працюють в динамічному надшвидкісному режимі, та тестуються при завантажені системи процедурою POST BIOS(Power On Self Test, Basic Input Output System) або вбудованими програмами для тестів UEFI.

Результатом процесу обробкі інформації може стати отримання нової інформації або зміна форми представлення існуючої інформації внаслідок отримання інформації системою з її вхідного контуру. Реалізація захисту інформації під час процесу її обробки є прорегативою комп“ютерної системи, що її обробляє. Перш за все алгоритми комп“ютерної системи не повинні обробляти інформацію, що не відповідає формату узгодженних вхідних данних, іншими словами — комп“ютерна система повинна чинити опір фаззингу.

Іншим важливим чинником є забезпечення прийнятного часу обробки інформації відносно кількості зусиль, які необхідно здійснити системі(виражається в функції «о велике» наприклад O(log(n)), яка може бути узагальнено обчислена для системи), оскільки в іншому випадку вона може бути вразливою до атак відмови обслуговування(DOS — denial of service) результатом якої є унеможливлення використання її потужностей для легітимних операцій обробки інформації. Для реалізації більшого степені захищеності даних в обчислювальній системі повинен бути передбачений прийом тільки достовірних даних з її вхідного контуру, зміст якого полягає суворій приналежності інформації к суб“єкту, який є джерелом інформації. Ще одним важливим параметром обчислювальної системи є перевірка цілісності інформації з її вхідного контуру, результатом якої є гарантія що дані не змінилися під час інших обчислювальних операцій, тобто не мало місце їх випадкове або навмисне пошкодження або руйнування їх семантичної складової.

Рис 1. Етапи процесу обробки інформації

Під час процесу передачі даних між двома точками виникає поняття канала зв“язку, що описує середовище передачі з деякими його властивостями(насамперед максимально можливою швидкісті передачі, емності каналу зв“язку) з якими повинні бути узгоджені обидві взаємодіючі точки. В результаті програмних чи аппаратних помилок, особливостей функціонування або порушеннях фізичних процессів, що супроводжують функціонування середовища передачі в каналі зв“язку можуть з“явитися завади або шум.

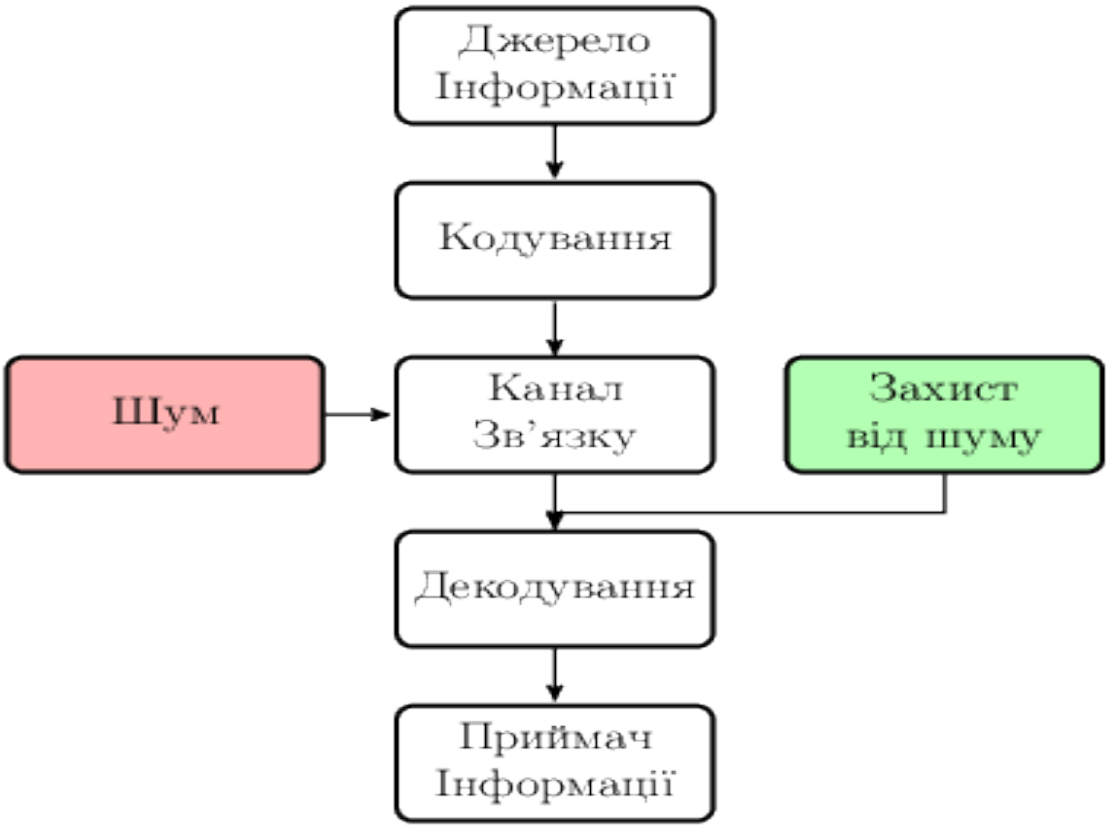


Рис 2. Процес передачі інформації між двома точками

Кодуванням в процесі передачі даних можна вважати одну з технік перетворення інформації завдяки якій учасники системи будуть мати змогу захистити відправлені дані, та перевірити отримані дані. Це може бути один із методів або комбінація з них: шифрування, зтиснення або захист від шуму. В якості захисту від шуму може бути застосований метод із збитковими бітами. В якості шуфрування може виступати будь-яке перетворення інформації внаслідок

якого кардинальним чином змінюється контент інформації в обох напрямках, а його алгоритм відомий обом сторонам. Зтисненням можна вважати алгоритм аналізу частоти використання окремих складових контенту інформації та співставлення цих складових відповідним складовим з деякої множини, зважаючи на їхню частоту в початковій інформації, в результаті роботи якого довжина інформації може стати значно меншою ніж початкова.

Захист інформації під час її передачи є однією з найскладніших проблем сьогодення. Складність цього процесу характерізується насамперед через необхідність взаємодії елемента системи, що є джерелом інформації з елементом системи, що виступає в ролі приймача, через відкриті канали зв“язку. Розглядаючи взаємодію компонентів через протокол IPv4, для компонентів системи немає гарантій щодо стану захищеності кожної транзитної точки маршруту і неперадбачуваним є: кількість маршрутизаторів(або серверів що таку функцію забезпечують) через які проходить їхній обопільний трафік; факт фрагментації трафіку будь-якими маршрутизаторами задля узгодження MTU(Maximum Transport Unit) каналу що їх з“єднує; факт зміни маршруту мережного слідування трафіку. Протоколи мережевого та транспортного рівня OSI мають вбудовані засоби забезпечення цілісності(обов“язкове поле для контрольної суми в IPv4-пакеті, обов“язкове поле для контрольної суми в TCP-пакеті, необов“язкове поле контрольної суми в UDP-пакеті) однак в плані серьйозного захисту інформації розглядатися не можуть, а слугують лише як захист від можливих завад в середовищах передачи через коллізії, аппаратні та програмні помилки, тощо. Внаслідок таких умов можуть виникнути проблеми, що компрементують рівень безпеки під час процесу передачі інформації через мережу: перехват інформації внаслідок чого цілісність інформації залишается незмінною, а її конфеденційність порушена; модифікація інформації — початкове повідомлення змінюється (повністю або частково) та відправляється адресату(сніффінг та ip-инжектінг); підміна авторства повідомлення; перехват з подальшим його вилученням. Задля таких дій повинні бути успішними

наступні атаки: спуфінг(spoofing) - дозволяє виконати підміну якогось компонента (наприклад успішний ARP-spoofing може призвести до підміни MAC-адреси шлюза за замовчуванням на канальному рівні, як результат весь локальний домен мережі буде слати свій трафік через інший хост(хост зловмисника), або атака MITM(«man in the middle»)); сніффінг(sniffing) або перехват трафіку — як правило,результат успішної атаки MITM, коли трафік контролюється зловмисником пассивно; hijacking — передбачає перехват трафіку з подальшим його редагуванням(наприклад tcp-hijacking), а за необхідності і вилученням частин трафіку.

1.2 Сучасні методи захисту інформації

Розвиток потужності архітектур систем зберігання і обробки інформації, навантаження їх різноманітними задачами(електронного документообігу, операцій з банківськими транзакціями, тощо) та маніпулювання з різними класами інформації під час виконання цих задач створили необхідність в пропорційному розвитку ефективності захисту інформації відповідно до складності цих архітектур та інформаційних задач, які вони вирішують. Руйнування інформаційного ресурсу, тимчасова його недоступність або несанкціоноване його використання можуть призвести до негативного впливу на репутацію компанії та інших небажаних наслідків. Таким чином загрози щодо захисту інформації зробили засоби забезпечення інформаційної безпеки обов“язковими властивостями будь-якої інформаційної системи. Проектування архітектури сучасних ОС як багатокористувацької та багатозадачної створило необхідність включення засобів захисту інформації як частини дистрибуції самої операційної системи.

Інформаційною безпекою можна вважати стан захищеності інформації та інформаційної інфраструктури від випадкових або навмисних впливів, що покликані нанести негативні наслідки суб“єктам інформаційних відносин. Ці впливи прийнято називати загрозами, а їхнє реальне втілення — атаками. Класифікація загроз може відбуватися відносно різних критеріїв. Видами загроз

слід вважати класифікацію, яка викристалізовує вплив загрози на інформацію (на синтаксичному, семантичному або прагматичному рівні): порушення фізичної цілісності; порушення логічної структури; порушення змісту; порушення конфеденційності; порушення права власності. Природа виникнення загрози, показує характер її джерела, що може бути випадковим(відмови, помилки, стихійні лиха, тощо) - спонтаними незалежними від людей обставинами, або навмисним(зловмисні дії людей). Передумови появи загроз показують характер ймовірності появи загрози, серед них можна виділити об“єктивні(якісні або кількісні недостачі компонентів системи, концептуально застарілі елементи системи) та суб“єктивні(зловмисні дії, шпіонаж тощо). Джерелом загрози є безпосередній виконавець загрози, який може бути: людиною; технічним пристроєм; алгоритмом або программою; зовнішнім середовищем. Простір загроз можна розглядати як три чисельні групи: загрози конфеденційності, загрози цілісності та загрози доступності. Загроза конфеденційності реалізується якщо інформація, що представляє потенційну або діючу цінність оскільки, вона не повинна бути доступною стороннім особам, стає доступною для них. Загрози цілісності інформації виражаються в несанкціонованному або ненавмисному редагуванні її змісту. Загрози доступності викристалізовуються в ситуаціях коли захищена інформація стає заблокованною для свого власника, особи що нею володіє, або користувача.

Поняттю загрози суміжним є поняття вразливості, що є деякою невдалою характеристикою системи, внаслідок виявлення та експлуатації якої у зловмисника з“являється можливість провести атаку. Збиток також суміжне поняття для загрози, що означає реальний або прогнозований результат реалізації загрози(атаки) в обумовленому еквіваленті. Причини виникнення вразливостей можуть бути різноманітними: як наслідок складності інформаційних систем; зворотня сторона збиткової функціональності, що в свою чергу корисна для користувачів; незлагодженість робочого процесу розробників інформаційної системи та їхні помилки; неналежне тестування

функціоналу інформаційної системи; неналежне адміністрування інформаційної системи. Загроза стає можливою виключно завдяки вразливості, тому вкрай важливим є процес її виявлення та «пропатчення»(patch - заплатка). Протидія загрозам безпеки є основною метою засобів захисту інформаційних та коммунікаційних систем. Виявлення загроз повинно бути прорегативою сторони, що захищається, однак на практиці домінуючою є атакуюча сторона, спеціалісти якої, як правило, мають менше аніж спеціалісти ІБ, обов“язків. Хоча задля узагальнення інформації про виявленні та відомі вразливості інформаційний світ поклав багато зусиль, зокрема база даних CVE(Common Vulnerabilities and Exposures) та сигнатурна база вірусів, залишаються багато вразливостей що є невідомими(0-day) і успішно можуть експлуатуватися зловмисниками, що через власні канали можуть обмінюватися знайденою вразливістю. В результаті таких обставин інформаційна безпека, в більшості випадків є захистом від відомих загроз та залишается неефективною під час 0-day атак(або атак нульового дня).

В практичній діяльності мотивація для організації щодо забезпечення та фінансування сфери свого захисту описується набором категорій: загроза — вразливість — збиток. Захист інформації — виключно практична сфера діяльності, що може в фінансовому плані коштувати дорого, тому необхідним стає аналізу цього набору категорій для окремого випадку, внаслідок якого власнику інформації необхідно застосувати варіанти її захисту відносно специфіки окремого випадку: фінансового чинника; вподобаннь відносно технічних особливостей; ймовірності виникнення окремих категорій загроз; наявності обізнаного консультуючого персоналу. Додає складності в питання фінансування тей факт, що не завжди для окремого об“єкта сповна визначенні реальні загрози, а наслідки цих загроз проявляються не відразу та не завжди. Узагальнений параметр, що враховує цінність ресурса, рівень загрози та степень вразливості є рівнем ризику, і фактично означає ймовірність настання небажанної події та узагальнений збиток після того як подія відбулася.

В основі побудови організаційнійного захисту лежать деякі важливі принципи, що є узагальненими та перевіреними часом практиками. Первинна роль захисту припадає на регламентацію підзвітного використання носіїв інформації, які зберігають її, оскільки без них вона не існує. Другим постулатом є необхідність в ізоляції інформації, що захищается від несанкціонованого доступу, в незалежності від її форми, від потенційних зловмисників. Третім пунктом можна відзначити стратегію градуювання персоналу відносно їхньої обізнаності та посади, в результаті якої співробітники не мають доступу до інформації, що не стосується їхньої трудової діяльності. Четвертою щеблиною є запровадження режиму обмеженої інформованості, зміст якого полягає в юридичній відповідальності співробітників, що мають доступ до конфеденційної інформації, щодо розголощення відомостей та деталей, що їм відомі. П“ятим пунктом є мінімізація контактів між співробітниками з несуміжних сфер діяльності та здіснення контролю, спостереження та нагляду за співробітниками. Шостим пунктом є запровадження звітності та контролю відносно кожного звернення до конфеденційних даних, для забезпечення ще більшої надійності. Сьомим пунктом є запровадження політики об“єднаних повноважень, а у випадках виконання надсекретних завдань — дезінформація. Сукупність виконавців та техніки захисту інформації, яку вони використовують, а також об“єкти захисту, що організовані та функціонують згідно правил, що відповідають юридичним, правовим та нормативним документам, складають систему захисту інформації.

Система обробки інформації є складною системою, що складається з великої кількості компонентів, які мають різну степень автономності, що обмінюються між собою даними й можуть піддітися негативному зовнішньому впливу, та пов“язані зв“язками, які мають різну роль відносно архітектури системи в цілому. Компоненти системи обробкі інформації можна розділити на декілька груп: апаратні засоби — комп“ютери та їх складові(процесори, термінали, периферійні пристрої) тощо; програмні засоби — програми,

розполільні бібліотеки, об“єктні модулі, ОС, компілятори, компанувальникі, лінкувальники, відладчики, користувацьке ПЗ, тощо; дані — інформація, що зберігається тривало та недовготривало, архіви, системні журнали, користувацькі компоненти файлової системи, тощо; персонал — обслуговуючий персонал(адміністратори, ІБ-спеціалісти) та користувачі. Забезпечення інформаційної безпеки маніпулює з деякими абстрактними поняттями, які втілюються в фізичне уявлення: фізичним втіленням інформації є різноманітні носії інформації; фізичним втіленням об“єкту системи є пассивні компоненти системи, які зберігають, приймають або передають інформацію(файл, порт, пристрій вводу/виведення); фізичним втіленням суб“єктів системи є активні компоненти системи, що можуть оперувати с об“єктами або змінювати стан системи(користувачі та їхні програми, що виконуються(процеси) , службові процеси або демони). Крім забезпечення цілісності, конфеденційності та доступності даних, що обробляються, інформаційна безпека повинна гарантувати цілісність та доступність компонентів та ресурсів системи.

Задля кваліфікації доступу, як санкціонованого або несанкціонованого(з порушення встановлених правил розмежування доступу) система повинна впроваджувати наступні службові процедури: ідентифікацію, аутентифікацію, авторизацію та аудит. Процедура ідентифікації покликана забезпечувати ідентифікацію суб“єкта, через впровадження якоїсь персоніфікованої інформації(стрічка, число), що буде виступати в ролі ідентифікатора суб“єкта і однозначно ідентифікує окремий суб“єкт. Процедура аутентифікації уможливлює перевірку справжності суб“єкта відповідному йому ідентифікатору, через впровадження факторів аутентифікації(те що відомо суб“єкту, те чим володіє суб“єкт або те що є частиною суб“єкту), кількість та порядок яких може різнитися в окремих реалізаціях системи забезпечення захисту інформації. Процедура авторизації передбачає надання суб“єкту, що пройшов ідентифікацію та аутентифікацію, відповідні йому повноваження та доступ до ресурсів системи. Процедура аудиту(звітності) покликані

відстежувати дії суб“єктів системи, які стосуються внесення змін до системи, зберігаючи при цьому інформацію(журналювання), якої достатньо щоб відтворити деталі події, яка мала місце.

Конкретна процедура аутентифікації може передбачати багатофакторну аутентифікацію. Фактор аутентифікації передбачає додаткові кроки, через які користувач повинен пройти щоб довести свою справжність. Це можуть бути паролі, парольні фрази, PIN-коди, біометричні характеристикі суб“єкта системи, наявність аутентифікаційних карток, тощо. Сучасні ОС впроваджують механізми, якими може скористатися розробник ПЗ, щоб користувачі пройшли аутентифікацію за необхідними для конкретного випадка правилами. Зокрема, Unix системи впроваджуються механізм PAM(Pluggable Authentification Modules), що архітектурно складається з модулів, які динамічно підвантажуючись, впроваджуються логіку щодо обробки фактору аутентифікації. Вбудований фактор аутентифікації в ОС, базується на валідації набору імені користувача та паролю, який ОС зберігає в файлах, доступ до яких має тільки адміністратор. Логіка PAM модулів може передбачати захист від перебору пароля грубою силою(«bruteforce») або перебору пароля через словник(«dictionary attack»). Для запобігання можливості підбору пароля через rainbow-table, якщо файл с захешованими паролями виявився у зловмисника, кожен пароль користувача перед хешуванням «солиться» якимось випадковим набором байтів, що отриманий з пристрою псевдовипадкових послідомностей. В якості суворих запобіжних заходів може використовуватися хешування з розтягуванням, коли детермінований «підсолений» результат хеш функції проходить кількість раундів, що вказаний в файлі з паролями(наприклад якщо функція хешування SHA512 та розтягнення 3, то результатом буде SHA512(SHA512(SHA512(пароль + сіль))).

Несанкціонований доступ до інформації є набільш розповсюдженим видом комп“ютерних порушень, тому розмежування доступу має особливу роль в системах захисту інформації, та вимагає зваженого та релевантного для

окремого випадку застосування засобів розмежування доступу. Насамперед, застосування та впровадження адміністративних засобів розмежування доступу: політик; практик найму; перевірки біографії; класифікації даних; навчання з безпеки співробітників; ревізій та перевірок активностей співробітників. Політика щодо обов“язкового розмежування доступу(Mandatory Access Control(MAC)) впроваджує обмеження дій які може виконувати суб“єкт системи з об“єктом. В MAC у кожного об“єкта є мітка, і у кожного суб“єкту є допуск, що формується від мітки користувача, завдяки чому з“являється можливість обмеження доступу відповідності до рівня секретності об“єкту. Маркування об“єктів, мітки суб“єктів та правила формує тільки особа з адміністративними повноваженнями. В системах на ядрі Linux реалізацією MAC є підсистема Selinux. Політика щодо дескреційного розмежування доступу(Discretionary Access Control(DAC)) уможливлює застосовання обмеження доступу до об“єктів на основі рішення власника(суб“єкту) цього об“єкта. Дискреційність(вібірковість) проявляється в тому що власник об“єкта може надати будь-які права доступу на об“єкт іншому суб“єкту. В будь-яку Unix систему вбодований DAC та перевірка прав відбувається на рівні ядра ОС, а списки доступу (Posix ACL — access control lists) є частиною об“єктів файлової системи, та складаються з дозволів на читання/запис/виконання для власника, групи та інших. В маршрутизаторах реалізація DAC проявляється у використанні ACL, що застосовуються для мережевого інтерфейсу та визначають дозволений вхідний та вихідний трафік мережі. Політика щодо контролю доступу на основі ролей(Role-Based Access Control(RBAC)) визначає доступність зважаючи на роль суб“єкта. Ролі вимагають певних дозволів для виконання певних операцій, та отримуються суб“єктами на основі своїх повноважень або інших аспектів, що притаманні до окремого випадку. В Unix ОС вбудовано реалізацію RBAC та засоби декларації та приєднання суб“єктів до груп суб“єктів. Політика щодо розмежування доступу на основі правил використовує списки контроля доступу(ACL), що налаштовуються особою з

адміністративними повноваженнями та допомагають визначити чи надавати доступ суб“єкту. Для реалізації ACL необхідні наступні додаткові параметри: классифікація суб“єкта; час доби; членство в групах; властивості об“єкта. Наприклад, можна обмежити доступ співробітників до ресурсів після закінчення робочого часу. В unix-системах реалізація ACL, RBAC вбудована в підсистему selinux та apparmor.

Для процедури аудиту важливим є відображення інформації щодо подій, з службовими даними достатніми щоб відтворити їх хронологію і зміст, та збереження цілісності журналу. Вбудоване збереження цілісності файлів журналу не передбачене, хоча зловмисник скомпроментувавши системи неодмінно зажадає скористатися логклинером(logcleaner) щоб замести свої сліди в системі. В Unix системах, в якості системи аудиту, використовується демон syslogd, що описаний в RFC3164 кожна подія якого описується об“єктом(facility), що є відправником повідомлення та суворістю(severity), що описує рівень повідомлення. Конкретна реалізація syslog може передбачати налаштування щодо захищеного мережевого транспорту, ретрансляції логування на інший сервер, зберігання логування в БД, ротацію логування, сортування по severity або facility тощо.

Доступність ресурсів інформаційної системи може порушитись внаслідок атак на доступність DOS(Denial of Service), що унеможливлюють використання ресурсу системи навіть для санкціонованих користувачів, в деяких(Distributed DOS) випадках використовуючи ботнет(мережа скординованих джерел запитів) та командний центр для управління ботнетом. Першим вектором DOS є перевантаження великою кількістю трафіку, що технологічно може бути нівельований засобами захисту L3-L7: використанням файрволу та правил, що ідентифікують IP-адреси нападників та їхної належності до AS(autonomic system) та блокують їхній трафік; кластерною інфраструктурою сервісу з розподілом навантаження запитів на деяку кількість IP-адрес(DNS round-robin); використання проксі серверів, правила якіх можуть

валідувати санкціонований трафік. Другим вектором DOS є перевантаження неправильними пакетами, що виникає внаслідок ситуації коли порушнику створити запит просто(наприклад O(1)), а обробити сервісу цей запит потребує значно більше зусиль (наприклад O(n^2)) - внаслідок значної різниці в виконувальних операціях кліента та сервісу, останній, при чисельних запросах з неправильними пакетами, перестає відповідати на запити кліентів або значно уповільнюється при обробці цих запитів. В цьому випадку необхідно прибрати вразливість алгоритму обробки сервісу щодо продуктивності через модифікацію початкового коду ПЗ, якщо використовується продукт власного виробництва або сповістити розробника ПЗ щодо знайденої вразливості, якщо використовується сторонній продукт.

Засоби захисту інформації в сучасних комп“ютерах починають спрацювати вже на початку завантаження системи. Система UEFI що прийшла на зміну BIOS, має криптографічний протокол(так званий «Secure Boot»), що є частиною її специфікації, та покликаний перевіряти цифрові підписи виконуваних EFI-додатків(в ролі яких можуть виступати завантажувальні образи ОС), використовуючи асиметричну криптографію відносно ключів, що зберігаються в ключовому сховищі системи. Таким чином розробник ПЗ, підписавши виконувальний файл ядра ОС своїм приватним ключем та загрузивши відповідний ланцюг сертифікатів в сховище ключів системи UEFI, може аппаратно гарантувати користувачу завантаження виключно підписаного образа ОС, таким чином унеможлививши приховування логіки роботи типових руткітів, щодо модифікації критичних компонентів ОС. Початкові засоби захисту інформації в ядрі ОС впроваджують можливість додавання до свого складу публічної частини, якою ядро буде перевіряти цифровий підпис своїх компонентів, які є її динамічно підвантажувальними модулями, що розширюють функціонал основного ядра (різноманітних драйверів пристроїв, програмних компонентів).

Для захисту носіїв інформації в сучасних ОС існує багато програмних інструментів. Насамперед розділ носія інформації може бути зашифрований на рівні блочного пристрою, реалізацією цієї схеми в ОС Linux є технологія LUKS (The Linux Unified Key Setup), що використовує симетричну криптографію для шифрування, формуючи мастер ключ та додаткові 8 слотів для користувача, до кожного з яких користувач може приєднати парольну фразу або ключ. Іншою схемою захисту носіїв інформації є шифрування на рівні файлової системи(окремих файлів або каталогів), реалізацією якої є файлова система ext4(Fourth Extended File System), яка з додатковою опцією encrypt впроваджує шифрування на рівні файлів. Перед шифруванням файлів необхідно побудувати ключ шифрування, який додається в Linux Kernel Keyring(сховище ключів шифрування) для користувача, який його створив. Після додавання ідентифікатора ключа до розширених аттрибутів файла або директорії, його зміст буде шифруватися зазначеним ключем. Після аннуляції ключа зміст стане не доступним, доки правильний ключ знову не з“явиться в keyring, хоча аттрибути файла(його розмір, DAC, значення іноду) залишаються відкритими.

Віруси є шкідливим програмним кодом, що прикріпляється до виконувальних файлів. Як правило, функціональність віруса потребує виконання зараженого файла, та передбачає зміну оригінальної точки входу програми в точку входу на функцію, що активує функціональність віруса, згодом запускаючи код з оригінальної точки входу виконувального файлу, таким чином у користувача не виникає підозр щодо якоїсь зловмисної активності. Іноді віруси вражаються завантажувальний сектор або ФС флеш-накопичувача. Для вірусів існує база сигнатур, яка ідентифікує характерні елементи зловмисного коду, за якими можна розпізнати вірус. Троянській кінь — зловмисний код, який здійснює свої дії під виглядом бажаної операції, відрізняючись від віруса ти, що прив“язується до невиконувальних файлів(наприклад файлів зображень). Черв“які — зловмисний код, логіка якого передбачає самостійне виконання певної зловмисної дії та механізм поширення

через певну вразливість, зазвичай суттєво уповільнюючи роботу мережі. Backdoor — програмний код, який додається злочинцем в скомпроментовану систему, щоб потім минаючи фазу стандартної аутентифікації потрапити до неї в майбутньому. Rootkit — зловмисний програмний код, метою якого є вкоренитися у скомпроментованій системі, виконуючи модифікацію ОС та її системних файлів, через ескалацію прав доступу. Логіка зловмисного ПЗ може передбачати: активацію дій прийнятним для зловмисника(тригеритися після якоїсь системної події, активовуватися в певний час, проводити опитування ресурсу командного центру); виконання прослуховування натискань клавіш користувачем (keylogging); від імені користувача проводити протиправні дії; вимагати від користувача кошти(наприклад шифрування даних ОС та вимагання коштів за ключ, що може розшифрувати їх). Зловмисне ПЗ може опинитися в системі багатьма способами(через електрону пошту, завантаження з інтернету, через змінні носії, через спам, тощо), тому користувачу необхідно проявляти уважність щодо контенту від недовірених джерел та регулярно оновлювати ПЗ, яке він використовує. Експлуатації ПЗ уможливлюється внаслідок помилок, які в ньому присутні та притаманні МП на якій це ПЗ написано. Наприклад для С програм, найпопулярнішим методом експлуатації є техніка переповнення буфера(переповнення статичного буфера, помилки в форматуванні рядків, переповнення динамічної пам“яті), для Web-програм(python, ruby, php) - SQL-инекція(через помилки можливість виконання запитів до БД), для Java — вразливості, якими можна обійти обмеження sandbox(пісочниці) для застосунків.

Панацеєю від вірусів є встановлення ПЗ з довірених джерел використовуючи системні пакетні менеджери та їхній криптографічний функціонал щодо верифікації виконувальних файлів, уважність користувача щодо запуску виконувальних файлів, формування виконувальних файлів з оптимізаційними флагами компілятора задля ускладнення міграції зловмисного коду в секції виконувального файлу. Захистом від троянських конів та черв“яків

можна вважати уважність щодо контенту з невідомих джерел та своєчасне оновлення встановленого ПЗ для усунення відомих вразливостей, якими може скористатися зловмисне ПЗ. Протекцією від backdoor є моніторинг підозрілої мережевої активності процесів системи, зокрема відкриття та прослуховування мережевих портів. Протекція від rootkit є використання UEFI Secure boot, моніторинг підозрілих змін в системних файлах.

Рис 3. Структура організації захисту інформації

Забезпечення захисту даних під час передачі є однією з найскладніших задач інформаційного захисту. Вектор мережевих атак направлений на: пассивне прослуховування(sniffing) трафіка; прослуховування та модифікацію (hijacking) трафіка; видавання зловмисника за когось іншого (spoofing); маскування зловмисника під санкціонований ресурс(fishing); зловмисного використання бездротових технології; міжсайтовий скриптінг(XSS).

Прослуховування трафіку стає можливим завдяки MITM-атаці і spoofing(наприклад, коли зловмисник наприклад видає свій хост за маршрутизатор за замовчуванням для мережі) та сучасному ПЗ є аналізатором трафіку(наприклад Wireshark). Spoofing стає можливим завдяки вразливостям існуючих мережевих протоколів, в яких немає засобів перевірки достовірності — ARP(MAC), DNS, IP. Найпопулярнішим інструментом для spoofing є ПЗ Ettercap. Прослуховування та модифікація трафіка передбачає виконання сниффінгу та модифікацію трафіку задля входження в мережеві сессії жертв. Інструментом для цього може слугувати ПЗ mitmproxy. Fishing є типом шахрайства, що маскується під організацію з гарною репутацією, задля збору реєстраційних даних облікових записів жертв, шпигунства та інших зловмисних дій. Фішингове ПЗ може використовувати різні методи доставки(смс, голосове, електронна пошта, програми обміну миттєвими повідомленнями, SEO-poising) посилань для жертв , щоб змусити їх перейти за посиланням до підконтрольного зловмисникам сайту або запиту з XSS-кодом, для викрадення конфеденційної інформації. Найпопулярнішим інструментом для реалізації фішингу та векторів соціальної інженерії є ПЗ Social Enginering Toolkit(set). Досягнення зловмисних цілей з використанням бездротових технологій можуть передбачати: підняття неавторизованих точок доступу; глушіння частот; атаки на Bluetooth; злам Wifi-мереж(WPA, WEP, WPA2). Для деяких функцій з вищезазначеного списку є ПЗ aircrack-ng. Міжсайтовий скриптінг(Cross-Site Scripting) XSS — вразливість у WEB-застосунках, внаслідок експлуатації якої зловмисник може вбудовувати зловмисні скрипти опосередковано, а браузер жертви буде виконувати ці скрипти. Шкідливий скрипт при цьому може мати доступ до конфеденційної інформації — файлів cookie, ідентифікаторів сеансу, прямого доступу до пам“яті браузера, тощо.

Вразливість канального рівня протоколу IPv4 уможливлює реалізацію атаки MITM без надзусиль, тому ускладненням такого типу атаки може слугувати перехід на протокол IPv6, з його надійнішим канальним рівнем хоча

виникає інша проблематика - зниження конфеденційнійності трафіку кліентів, оскільки автоматичне конфігурування DHCPv6 на основі MAC-адреси мережевого адаптера буде утворювати однакові молодші біти IPv6-адреси в будь-якій IPv6 мережі, що призводить до ідеальних умов щодо відслідковування користувачів в мережах. Допомогою в протистоянні фішингу та ефективних тактик соціальної інженерії може бути тільки уважність та технічна обізнаність користувачів. Протидією щодо шахрайства в області бездротових технологій може стати: впровадження криптографічного інструментарію в будь-які бездротові технології(наприклад HMAC що використовується в APDU NFC); обізнаність користувачів щодо неавторизованих точок доступу та базових технологічних особливостей бездротових технологій. Проблематика навколо захисту міжсайтового скриптінгу є складною, оскільки викликана внаслідок наявності помилок розробників у WEB-застосунках, користувацька провина може полягати лише в відкриванні посилань з неперевірених джерел. В деяких випадках протидії атакам на інформацію під час її передачі є використання захищеного транспорту на криптографічній основі TLS(Transport Layer Security), SSH(Secure Shell), VPN(Vritual Private Network), IPsec(IP Security).

Спам є небажаним контентом в переважно в email-повідомленнях, що негативно впливає на пропускну спроможність мережі та є основним джерелом поширення вірусів та рекламного контенту, забираючи багато часу у адміністраторів поштових серверів. Серед новітніх технологій, що протистоять спаму можна виділити: використання «сірих списків»; технологія SPF(Sender Policy Framework); технологія DKIM(Domain Key Identified Mail). Технологія сірих списків автоматично блокує спам, зважаючи на поведінку спамерського ПЗ, яке не відповідаючи вимогам SMTP(Simple Mail Transport Protocol), надсилає своє повідомлення одноразово та отримуючи помилку про тимчасову недоступніть не буде намагатися повторити свою спробу, як це роблять легітимні SMTP-сервера. Технологія SPF вимагає від адміністратора поштового домену вказати в TXT-записі список серверів, які мають право відправляти

email-повідомлення зі зворотніми адресами в цьому домені, таким чином приймаюча сторона може перевірити легітимність зважаючи на відправника та DNS-запит по цьому поштовому домену. Технологія DKIM впроваджує криптографічні засоби захисту, потребуючи від адміністратора домену додавання в TXT-записі публічної частини ассиметричного ключа, якою можна перевірити підпис для сервера-приймача.Таким чином сервер-відправник маючи приватну частину створює на контент повідомлення підпис, який може перевірити сервер-приймач використовуючи публічний ключ ассиметричної пари с DNS-запиту щодо відповідного домену.

Зломисникам значно додає можливостей сучасний стан розвитку ПЗ, що покликане проводити тести на проникнення(penentation testing), але в їхніх руках використовується з інакшими цілями. Наприклад, ПЗ metasploit значно спрощує написання експлойтів, оскільки в своєму складі має узагальнені набори з енкодерами, заповнювачами нуп(nop), корисними навантаженнями, й містить весь необхідний супроводжувальний інструментарій для будь-якої фази хакингу. Стадія footprinting передбачає процес сбору якомога більшої інформації про цільову систему, серед якої: IP-адреси, записи DNS, відомості whois, версії ПЗ, електронна пошта, тощо. Стадія сканування(scanning) передбачає набір процедур для ідентифікації мережевих об“єктів, відкритих портів, версій та архітектур їхнього ПЗ(використовуючи мережеві сканери), виявлення загроз та вразливостей(використання сканерів вразливостей) задля формування профілю цільової організації. Стадія перерахування (enumeration) передбачає процес активного сбору більш детальної інформації про ціль: виявлення імен користувачів; мережевих ресурсів та їхніх налаштувань; таблиць маршрутизації; тощо. Стадія непосередкованого хакінгу передбачає наступні кроки: отримання доступу(gaining access); ескалацію привілеїв(priviledge escalation); виконання ПЗ(executing applications); приховування файлів(hiding files); приховування доказів порушення та перебування хакера в системі(covering tracks).

1.3 Засоби захисту ОС Android

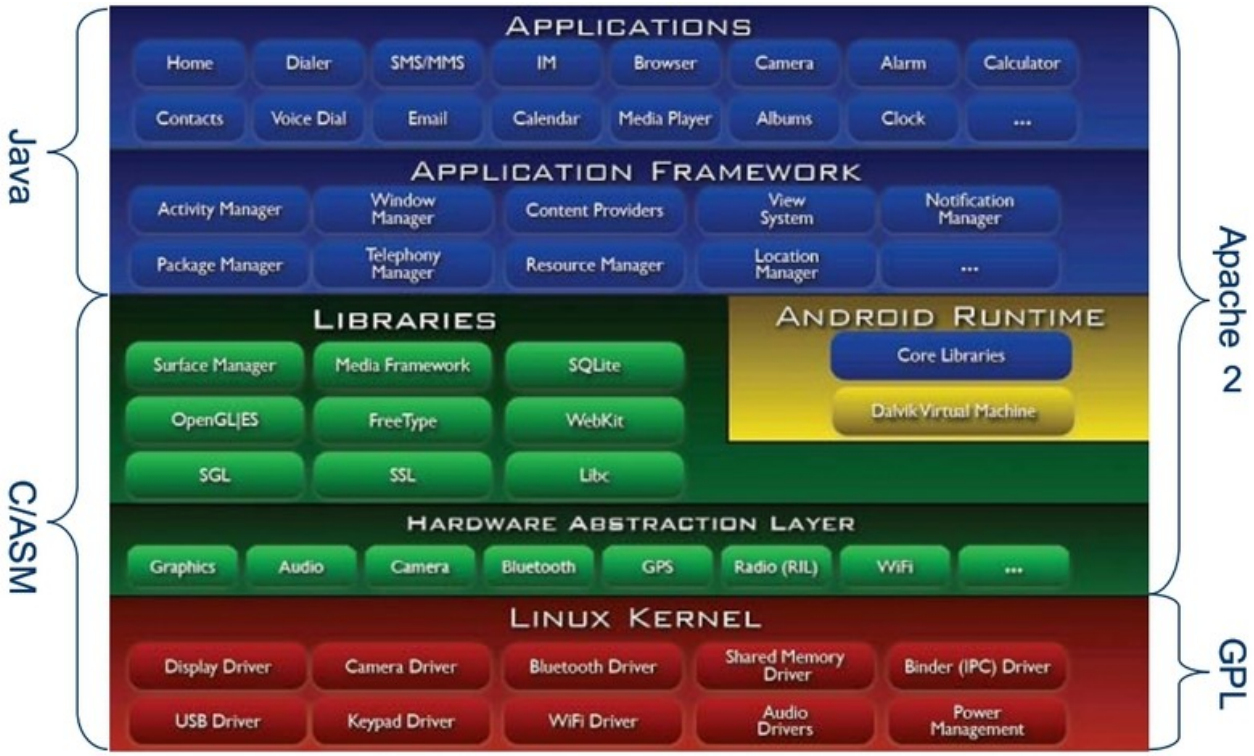


Рис 4. Структурна схема ОС Android

Мобільні пристрої за призначенням значно відрізняються від звичайних комп“ютерів. Комп“ютери володіючи стаціонарними клавіатурами та великими екранами, можуть використовуватися багатьма користувачами, мати загальні з іншими комп“ютерами ресурси(локальні та мережеві), бути у складі домена мережі або бездисковими станціями, в яких завантажувальний образ ОС зберігається в мережі. Мобільні пристрої є більш персоніфікованими однокористувацькими пристроями, що зберігають значно більше локалізованої та конфеденційної інформації користувача. Крім того мобільні пристрої підтверджуючи своє реноме щодо мобільності мають високий параметр доступності, що спонукає користувачів використовувати мобільний пристрій для простих повсякденних дій — що в свою чергу додає локалізованої інформації на пристрій. Сучасні мобільні пристрої мають широкий спектр обладнення — мікрофон, GPS, постійно ввімкнений радіозв“язок, акселерометр, гіроскоп, NFC, тощо. Отже широкий спектр обладнення

мобільного пристрою і набір локалізованої та персоніфікованої інформації, що поповнюється відносно його користування - роблять його ідеальним засобом для шпіонажу за користувачем. Тому функціональність ОС мобільного пристрою змушена мати розвинуті засоби безпеки.

ОС Android побудована поверх ядра Linux, що впроваджує численні драйвера для взаємодії з аппаратними, програмними(блоковими та символьними) та мережевими пристроями, файловим системами та іншою периферією. В будь-якій Unix-системі ядро забезпечує низку службових функцій: доступ до системних викликів з деякої множини(таблиця системних викликів); менеджмент процесів та потоків, що виконуються; менеджмент пам“яті (відображення віртуальних сторінок пам“яті на фізичні адреси) для кожного окремого процесу; обробку програмних та аппаратних переривань; мережеву інфраструктуру; контроль пристроїв; тощо. Через таблицю системних викликів кожен процес або поток може звертатися до ОС задля: файлового вводу/виведення; мережевої взаємодії; зміни своїх аттрибутів; взаємодії з іншими процесами системи через IPC або через сигнали; менеджмент криптографічних ключів; створення іншого потоку або процесу; тощо. До основного ядра Android впроваджує додатковий необхідний йому функціонал: low memory killer; wakelock; будильники;анонимна спільна пам“ять(ashmem) та фізична пам“ять(pmem); параноідальна мережева взаємодія; IPC Binder; відмінна від традиційної система логування; збереження логування ядра в RAM. Low memory killer — підсистема, яка при пороговому значенні використання оперативної пам“яті(що може змінюватися через конфігурацію) задля її звільнення починає радикально завершувати процеси(застосування) в залежності від їхнього статусу(пусті, контент провайдери, приховані, вторинні сервіси, видимі, на передньому плані). ОС Android прагне залишити якомога більше енергії тому постійно буде намагатися перевести процесор та компоненти в енергозберігаючий режим, наприклад після вимкнення екрану може бути вимкнутий процессор повністю, а напруга залишиться лише на

ОП(suspend режим). Механізм wakelock покликаний для забезпечення функціональності застосувань навіть після вимкнення екрану, а механізм будильників — для виходу з режиму suspend у визначенні інтервали часу, якщо цього потребують застосунки. Анонімна віртуальна пам“ять використовується для міжпроцессної взаємодії, що існує поки вони функціонують, pmem — регіони фізичної ОП, які використовуються коли потрібні значна за об“ємом неподільна пам“ять(між пам“ятю застосунка та драйверів ядра). Система логування Android на аллоцюється в пам“яті ядра ОС, маючи інтерфейс с узерпростору, в звичайній системі журналювання транслюється в файли службовим демоном syslog. Збереження логування ядра в RAM, забезпечує запобіжні заходи коли трапляється паніка ядра та логування ядра буде доступне при наступному завантаженні через інтерфейс /proc/last_kmsg. Параноїдальна мережева взаємодія реалізується через додавання в ядро можливості щодо обмеження доступу до мережевих ресурсів процесам(застосуванням), що не є членами групи inet.

Основним засобом міжпроцесної взаємодії в ОС Android є Binder, що є ядерним драйвером та доступний через інтерфейс /dev/binder. Сама взаємодія реалізується через окремий ioctl до binder драйвера, який контролює деяку частину адресного простору процеса, що для процеса відкрита тільки на читання, запис же виконується ядерним драйвером. Коли один процес надсилає повідомлення іншому процесу, ядро аллоцює деякий постір в процесі-приймачі та копіює повідомлення безпосередньо в пам“ять приймача. Коли процес завершується з повідомленням, він рапортує binder-драйверу для позначення пам“яті як звільненої. На високому рівні доступ до кожного об“єкту може здійснюватися через binder интерфейс, виклики до якого виконуються в binder транзакціях, що в своєму складі мають: посилання на цільовий об“єкт; идентифікатор методу для виконання; буффер з інформацією для метода. Binder драйвер до binder-транзакції додає ідентифікатор процесу(PID) та еффективний ідентифікатор користувача(EUID) процесу, що її створює.

Ключовим моментом є заповнення драйверним рівнем PID та EUID, яке унеможливлює додавання фейкових ідентифікаційних даних задля ескалації прав доступу. Кожен Binder об“єкт, що може використовуватися в транзакціях, має унікальні ідентифікаційні дані(токен), що формуються на рівні ядра. Ядро слідкує за відображенням живих binder-об“єктів та їхніх посилань в інших процесах та за їх станами, таким чином формуючи у кожного binder-об“єкта унікальність, незабутність та коммунікабульність, крім того забезпечується унеможливлення створення binder-об“єкту дубліката. Binder-об“єкти можуть використовуватися як токени, які інтегрують в себе сам об“єкт та набір прав доступу, впроваджуючи таким чином capability модель прав доступу для звичайної Linux системи.

Наступним слоєм в архітектурі Android є впровадження JVM(Java Virtual Machine) через середовище виконання Android Runtime, що прийшовши на заміну Dalvik, компілює застосування при їх встановленні, не використовуючи JIT компіляцію, за рахунок чого досягається збільшення швидкості роботи програм, та збільшення часу роботи від батареї. Застосування не інтерпретують Java bytecode(файли з розширенням .class), а використовують свій формат Dalvik Executable(DEX)(файли з розширенням .dex), опткоди якого більш оптимізовані під мобільні платформи. Dex-файли пакуються в Java Libraries(JAR), йормуючи файли формату Android Applications(APK).

Набір бібліотек Java Runtime необхідний для забезпечення стандартної бібліотеки МП Java, як правило це файли з пакетів java.* та javax.*. Крім базових функцій МП Java, він впроваджує JNI(Java Native Interface), що потрібний для виклику нативного коду(написаного на МП C, C++) з МП Java. Переважна більшість системних сервісів, що впроваджені окремою версією ОС Android(перебувають в точці монтування /system), які є стандартними компонентами для ОС Android написані з застосуванням IPC Binder, через який вони впроваджують власний RPC(Remote Procedure Call), який може потребувати логіка функціонування користувацьких застосувань(застосувань з

точки монтування /data). Користувацькі застосування, що включаються в PlayMarket обов“язково повинні бути підписані приватним ключем розробника.

Кожний процес є суб“єктивною складовою багатокористувацької ОС, що обов“язково має: ідентифікатор користувача(UID); ідентифікатор группи(GID); ідентифікатор процесу(PID); ідентифікатор предка процесу(PPID); відкриті ресурси(файли, сокети, тощо); список наявних потоків; тощо. ОС Linux впроваджує додаткові засоби для гнучкого менеджменту процесів: контрольні групи(ccgroups) ; простір імен(namespace); можливості(capabilities). Cgroups дає можливість ізоляції(призупинення; перевантаження) і встановлення обмеження для процесу(або групи процесів) на обчислювальні ресурси та аудит використання ресурсів. Android декларуючи декілька ccgroups, для кожної встановлює деякі параметри, що впливають та пріоритет виконання процесу коли він належить окремій групі(наприклад застосування на передньому плані попадає в найпріорітетнішу ccgroups таким чином отримуючи більше процесорного часу для виконання, інші процеси попадають у менш пріоритетні групи і відповідно мають менше процесорного часу). Технологія namespace дозволяє ізолювати процес(группу процесів) через створення просторів імен(процесів, файлових систем, користувачів, мережних пристроїв, IPC, тощо) для нього, в результаті чого він не буде мати доступу до хостової машини, та опиниться в пісочниці(sandbox). Capabilities дозволяют процесам мати обмежені адміністративні повноваження(можливості користувача root) з деякої множини(наприклад cap_net_bind дозволяє непривілігіованому процесу відкрити порти, які менше 1024). Отже, базові засоби захисту Linux для Android забезпечують: обов“язкове встановлення UID користувача, що його запускає(не враховуючи suid та sgid біти); для кожного процесу власний простір віртуальної пам“яті; перевірку прав доступу(DAC); менеджмент та встановлення обмежень для процесу; можливість формування пісочниці для процесу використовуючи окремий namespace та cgroups.

Для кожного нового застосування ОС Android при його встановлені формує новий користувацький UID(починаючи від 10000) , що буде унікальним в рамках даної системи, та приватну директорію для застосування, таким чином кожний процес цього застосування буде мати унікальний UID, власну приватну директорію(правами на яку може редагувати застосування) та пісочницю, що побудована засобами Linux. ОС Android впроваджує специфічні для нього типи об“єктів, що може використовувати застосування: Activity - екран застосування; Service - впровадження виконання RPC-процедур; ContentProvider - впровадження інтерфейсу до якоїсь БД; BroadcastReceiver — отримувач деяких подій в системі. Оскільки за замовчуванням окреме застосування має доступ тільки до своїх файлів, необхідне впровадження для нього дозволів, яке воно декларує в файлі AndroidManifest.xml. Прості дозволи застосування може отримати після процедури встановлення, для захищених дозволів вмикається механізм запросу дозволів під час виконання застосування, ескалаційна зміна дозволів якого призводить до перестворення процесу застосування. Для пришвидшення завантаження застосувань створення процесу для застосування відбувається через системний процес zygote, в якому відкриті всі необхідні застосуванню ресурси, а процес застосування стає його нащадком.

До додаткових засобів захисту ОС Linux, що використовуються в Android можна віднести secomp(secure computing mode), MAC, що базується на підсистемі SELinux, та ASLR. Secomp — механізм ядра, який дозволяє обмежувати набір доступних системних викликів для застосувань, та за допомогою BPF(Berkley Packet Filter) фільтра виконувати складну фільтрацію викликів та їх аргументів. Використання MAC дозволяє ОС гарантувати виконання дій процесами(суб“єктами) з об“єктами згідно встановленої політики безпеки. MAC функціонує з контекстом безпеки - множиною атрибутів об“єктів(що повинні бути поміченими через розширені атрибути файлових систем) та множиною атрибутів суб“єктів(процесів). Політика безпеки повинна декларувати набір правил щодо регулювання взаємодії

суб“єктів(аттрибутів і дій) та об“єктів. ASLR(Address Space Layout Randomization) — це технологія сучасних ОС, внаслідок якої в адресному просторі процесу випадково змінюються адреси важливих структур даних — аргументів програми, динамічних бібліотек, стека, хіпу(heap) тощо. ASLR значно ускладнює експлуатацію вразливостей коли атакуючому необхідно встановити розташування стеку або інших важливих структур щоб розташувати там свій шелл-код. З мінусів ASLR можна відмітити збільшення розміру виконувальних файлу та збільшення часу завантаження програм. Для вмикання ASLR необхідно під час компіляції програми використовувати спеціальні флаги компілятора(зокрема для gcc -pie).

Для кожного APK-файлу с застосуванням необхідний підпис ключем розробника, для системних застосувань Android необхідний підпис платформеним ключем, що має виробник окремого пристрою. Оновлення застосувань найчастіше відбувається через PlayMarket, системні оновлення можуть автоматично генеруватися виробником пристрою, вручну через recovery mode встановленням підписаного виробником пристрою архіву з новою системою. В деяких випадках може застосовуватися розблокування завантажника, яке потребує повного збросу контенту файлової системи, щоб гарантувати неможливість доступу к наявними даним, якщо це зловмисні дії. Для системних блокових пристроїв, де зберігається ядро та інші компоненти, обов“язково перевіряється цілісність компонентів, в додачу реалізований механізм, що не дає змоги відкатити систему до функціонування з попередніми версіями, оскільки вони можуть мати експлойти та відомі вразливості.

Засоби захисту ОС Android дають можливість шифрування файлової системи на рівні файлів та на рівні блокового пристрою з використанням симетричних ключів шифрування з можливою протекцією з паролем. Засоби регістрації формують деяку ключову випадкову інформацію(64-бітний ідентифікатор), що слугує ідентифікатором для користувача та маркером прив“язки до криптографічного матеріалу користувача. Засоби аутентифікації

ОС Android впроваджують сучасні фактори, якими користувач може скористатися: PIN-код; шаблон; пароль; відбиток пальця. Наявність безпечного функціонування через SoC(System On Crystal) дає можливість для експорту аппаратних служб, таких як сховище ключів, що забезпечує генерацію та імпорт пар ассиметричних ключів, формування підпису та його верифікацію.

1.4 Висновки до розділу 1

Заходи щодо поліпшення методик захисту інформації повинні постійно вдосконалюватися, оскільки вдосконалюється ПЗ, яке функціонує з інформацією в різних її станах: збереження, обробка, вимірювання та передача. Протягом кожного стану інформації захисні функції делегуються на різних спеціалістів ІТ-сфери. Так, група безпеки мережі відповідає за дані під час передачі, програмісти відповідають за дані під час її обробки, а фахівці з аппаратної частини та серверної підтримки відповідають за збережені дані. Крім того, існує постійна ймовірність знайдення вразливості(0-day) в ПЗ, що використовується окремою системою.

Практично будь-яке сучасне комп“ютерне ПЗ в своєму складі має функціональність щодо захисту інформації. Для ІБ-спеціаліста важливим є: моніторинг появи вразливостей; співпраця з розробниками ПЗ; розробка оптимальних технік щодо захисту інформації; впровадження тестів на проникнення. Для розробника ПЗ важливо: робити якомога менше критичних помилок, які можуть стати вразливостями; не ускладнювати компоненти без необхідності; використовувати найновіші версії бібліотек та оптимальні захисні флаги при компіляції ПЗ; писати різноманітні функціональні тести, залучаючи тестувальників проводити регресійні тести щодо ПЗ. Для адміністратора важливим є: чітке розуміння технічної документації; активація захисних функцій ПЗ; співробітництво з розробниками ПЗ щодо знайдених вразливостей; технічне консультування користувачів. Для користувачів важливим є здобуття базової технічної грамотності щодо захисту інформації та дотримання її постулатів, тим самим створюючи опір щодо технік соціальній інженерії.

РОЗДІЛ 2. КРИПТОГРАФІЧНІ МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ

2.1 Сучасні методи криптографічного захисту

Криптографія протягом тисячоліть була мистецтвом, за допомогою якого важлива інформація державних утворень ставала секретною всупереч тому що передача її здійснювалася по незахищеним каналам, а криптоаналіз був пов“заний з опозиційною до криптографії діяльністю, що передбачала розкриття секретної інформації, перетвореної за допомогою криптографії. Тому криптологія, що інтегрує в себе множину засад криптографії, криптоаналізу та їхної взаємодії, значно відрізняючись від інших академічних наук, історично знаходилася у розпоряджені військових, дипломатичних та політичних відомств, що впливало на її розвиток. З часом політично вмотивована завіса над криптологією почала розкриватися пропорційно з розсекречуванням наукової літератури, що містила здобутки цієї науки. Принципово новий, математично обґрунтований, підхід до конструювання надійних криптосистем заклав Л.Хілл у 1929 році. Протягом 70-80 рр. 20 ст. почалися наукові відкриття, які значно вплинули на розвиток криптографічної складової криптології, зокрема: вагомий внесок Х.Фейстеля у вивчення симетричної криптографії(мережі Фейстеля); відкриттям криптографії з відкритим ключем завдяки У.Діффі, М.Хеллману та Р.Мерклу; відкриття ассиметричного алгоритма RSA завдяки Р.Рівесту, А.Шаміру та Л.Адельману. Одним з ключових здобутків новітньої історії науки стало використання в ассиметричних криптосистемах еліптичних кривих над кінечними полями, що було запропоновано Н.Кобліцем в 1985 році. В сьогоденні наукові досягнення науки стабілізуються завдяки роботі інституцій, що проводять науково-дослідну роботу щодо властивостей криптостійкості існуючих криптографічних систем та їх параметрів(зокрема NIST(The National Institute of Standarts and Technology)), результати та авторитет яких впливають на формування національних стандартів держав, міжнародних законодавчих норм та іншої юриспонденції щодо захисту інформації криптографічними засобами. Інтенсивне використання криптографічних засобів захисту

інформації зачно посилило якість теоретичних основ математикі, особливо розділів, які використовуються в криптографії. Спостерігається розвиток нових напрямків в криптографії, наприклад квантовий її вектор, де окрім математичних засобів, використовується методики квантової фізики.

В наш час інтернет-технології тотально увійшли в побут людства, та продовжують свою інтеграцію у всі важливі сфери діяльності через впровадження електронного документообігу серед яких: банківська сфера; фінансово-інвестиційна сфера; ринок цінних паперів; торгово-промислові компанії, тощо. Бурхливий розвиток сфери електронних послуг та коммерції значнішою мірою актуалізували проблему забезпечення інформаційної безпеки мереж, оскільки вони технологічно є відкритими каналами зв“язку. Роль криптографічних засобів захисту інформації на узагальнену галузь забезпечення інформаційної безпеки є опорною та найвідповідальнішою, а їх дієвість вбачається незаперечною. У більшості випадків математичних доказів щодо що підтвердили б неможливість розкриття математичних девіацій, на яких побудоване сучасне вістря криптографії — немає, однак є експлуатаційна складова, що формує репутацію криптографічного алгоритма. Крім того, існує проблематика навколо інженерної складової щодо проектування криптографічних систем, коли, наприклад, всупереч зачисту та ефективності функціонування віддають перевагу зручності користування, або впроваджують без необхідності надмірну складність негативно впливаючи на захист, на додачу на ІБ-спеціалістів психологічно негативно може впливати постійна атмосфера протистояння з зловмисниками. Більш якісного проектування криптосистем можна досягти керуючись догмами правила слабкої ланки(захищеність системи пропорційна безпеці її найслабшої ланки) та слідування дещо упередженому вектору проектування(параноїдальні твердження).

Криптографія, є обширною наукою, яка історично починалася як мистецтво та наука шифрування, в сьогоденні може робити суміжними дискусії

навколо комп“ютерної безпеки, вищої математикі, квантової фізики, права, юриспонденції, проектування мікросхем, програмування, політикі, тощо та має більш обширні засоби впливу на інформацію для різного функціонального застосування: шифрування, хешування, аутентифікація, цифровий підпис, інфраструктура відкритих ключів(PKI). Шифрування було початковим призначенням криптографії, та являє собою зворотнє перетворення інформації, переведення інформації з одної множини в іншу з біективними властивостями, задля її приховування для неавторизованих суб“єктів, в одночас впровадження доступу до неї для авторизованих суб“єктів, таким чином слугуючи засобом забезпечення конфеденційності інформації. Шифрування є функцією результатом якої буде зашифрований текст, а вхідними параметрами- ключова інформація та відкритий текст($c=E(K_E,m)$). Якісна функція шифрування унеможливлює визначення відкритого тексту m , по шифрованому тексту c , без ключа K_E . Для дешифрування повідомлення($m=D(K_E,c)$) необхідно знати алгоритм шифрування D та ключ дешифрування K_E . Згідно закону О.Керкгоффса надійність схеми шифрування повинна залежати виключно від секретності ключа K_E , та не повинна залежати від секретності алгоритмів шифрування та дешифрування. Практичне значення щодо відповідності схем шифрування цим вимогам полягає в складності зміни алгоритмів - вони вбудовуються в програмне та аппаратне забезпечення та оновлюються з застосуванням значних зусиль. Використання закритих алгоритмів потенційно може збільшити рівень безпеки, але залишається велика ймовірність зменшення рівня безпеки оскільки властивості щодо криптостійкості закритих алгоритмів не є наявними і можуть мати серйозні недоліки, якими скористаються зловмисники. Шифрування з відкритим ключем використовує схеми шифрування в яких процеси шифрування $c=E(P_E,m)$ та дешифрування $m=D(S_E,c)$ значно різняться з математичної точки зору

та вимагають різної ключової інформації(відкритого/закритого ключа), отже на заміну спільного ключа K_E , приходить пара ключів (P_E, S_E) , а алгоритми в функціях (D, E) принципово різняться. За шифруванням з відкритим ключем завжди стоїть серьйозна математична закономірність, що впроваджує суттєву різницю в зусиллях необхідних для здійснення деяких математичних перетворень, і пов“язує ключову пару математично унемождивлюючи отримання приватного ключа з публічного. Шифрування з відкритим ключем нівелює проблему необхідності обміну спільним ключем K_E , що є з криптографічного боку великою проблематикою, та впроваджує необхідність генерації пари ключів (P_E, S_E) , публічна частина якої P_E може бути опублікована, завдяки чому будь-який суб“єкт зможе скористатися нею щоб захищено скомунікуватися з власником приватного ключа. Таким чином шифрування впроваджує для інформації властивість секретності. Різні комбінації типів шифрування використовуються в багатьох сучасних протоколах та технологіях, зокрема в TLS, SSH, IPsec, LUKS, WPA,тощо.

Хешування є службовою процедурою, що внаслідок деяких перетворень вхідних даних довільної довжини, формує деяку послідомність фіксованої довжини(хеш, відбиток, digest). Хешування може використовуватися в багатьох випадках: в програмуванні для побудови ассоціаціативних массивів(тип хеш-таблиця з операціями доступу до даних з O(1)); для формування контрольних сум об“єктів для інспекції зміни їхнього стану контенту; для збереженя інформації у вигляді хеш-кодів, тощо. Хеш-функція $H_m = hash(m)$ повинна мати наступні властивості: незворотність — тобто не було можливості з H_m отримати m ; довжина H_m повинна бути фіксованою і залежати від hash(); криптостійкість — можливість чинити опір відомим технікам криптоаналізу; мінімальна кількість коллізій, тобто H_m з мінімальною вірогідністю повинно відповідати контенту відмінному від m ; прийнятна алгоритмічна

складність(O). Існує багато різних хеш-функцій, з різними характеристиками, зокрема: MD5, SHA1, SHA256, SHA512, SHA3(Keccak), RIPEMD-160, WHIRPOOL, BLAKE2. В більшості випадків хешування допомагає сприяти вирішенню задачі цілісності, де необхідне отримання стану контенту інформації(відбиток), який згодом наприклад можна підписати формуючи таким чином ЕЦП. Для криптовалют, зокрема Bitcoin, хешування має надважливу роль в функціонуванні її блокчейна (blockchain). По-перше внаслідок хешування створюються символьні імена користувачів(21-байтова Bitcoin адреса), власників електроних гаманців наприклад 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa, використовуючи спеціальний енкодер Base58(уникає дублювання подібних символів, наприклад 0 та O, 1 та l) і логіку формування з $subname = ripemd160(sha256(publickey))$ з використанням публічного ключа, з вставкою контрольних послідомностей через два раунди хешування $sha256(sha256(subname))$.

По-друге кожна складова блокчейну - окремий блок та окрема транзакція, мають у своєму складі поле зі своїм хешем, крім того блок обов“язково включає список своїх транзакцій, які зберігаються в дереві хешей(merkle tree), що є відбитком всіх транзакцій в блоці(наприклад $hsh_{00} = hsh(L_0)$; $hsh_{01} = hsh(L_1)$; $hsh_0 = hsh(hsh_{00} + hsh_{01})$; $hsh = sha256\, sha256(x)$), завдяки чому стає можливим спрощена перевірка статусу окремої транзакції, кліентська сторона завантажує заголовок блоку та аутентифікаційний шлях окремої транзакції, дерево, в свою чергу, в деякій мірі знижує рекурентність обчислення. По-третє, блокчейн технології користуючись властивістю незворотності хешування, в процесі функціонування ставлять завдання майнерам(додавачам блоків), що базується на складності підбору хеш значення, яке менше за деяке поточне, тобто майнерам доводиться підбирати значення хешу лінійно, змінюючи окреме поле в блоці, що є кандидатом на включення в блокчейн. Крім того блокчейн технології ввели в побут одиниці обчислювальної потужності(хешрейту або кількості хешів) для обладнення та комп“ютерів відносно часу - хеш/сек.

Системи контроля версій ПЗ, зокрема GIT, використовують хешування як основний інструмент для отримання стану об“єкту, хеш якого вставляється в історію системи контроля версій ПЗ. Важливість забезпечення неколізійності та криптостійкості алгоритмом хешування є ключовою, оскільки значна частина криптографічних технологій(зокрема ЕЦП) покладається на ці властивості використовуючи хеш-функцію. Якщо ж вони порушені можуть виникати серйозні наслідки — наприклад, вразливість MD5(вразливість до коллізійної атаки с заданим префіксом) призвели до створення підробленого сертифікату за хешем однакового з легітимним сертифікатом Microsoft, що дало можливість черв“яку Frame підписувати оновлення для Windows XP, 7, Vista, тому нерекомендується використовувати алгоритми хешування MD5 та SHA1 через знайдену в них численну кількість коллізій.

В більшості практичних випадків властивості секретності щодо інформації недостатньо, оскільки відкриті канали зв“язку можуть контролюватися зловмисником, який може активно впливати на: факт та порядок отримання інформації сторонами; контент інформації. Проблема факту та порядку отримання може нівелюватися через додаванням збиткових даних в протокол між сторонами. Модифікація контенту інформації може призвести до спотворених даних при їх розшифруванні, які правильно спроектована система обробки інформації на приймачі не буде сприймати, але якщо інтерпретація все таки відбувається то забезпечення секретності інформації не означає забезпечення справжності інформації. Крім того, архітектурно правильніше, коли сам приймач валідує авторство та факт справжності отриманної інформації. Засоби криптографічної аутентифікації допомагають вирішити задачу аутентифікації через забезпечення наступних властивостей: справжність інформації — гарантія, що інформація в процесі передачі не піддалася несанкціонованій модифікації; авторство інформації — гарантія для приймача інформації, що дані створені очікуваним джерелом. Спроможність системи забезпечити надійну аутентифікацію даних називається аутентичністю, в ряді

випадків реалізовується імітозахистом - встановленням MAC(Message Authentication Code) або MDC(Manipulation Detect Code). Імітозахист даних є протекцією від нав“язування неправдивих даних, що реалізується за допомогою додавання до повідомлення додаткового коду. При використанні MDC застосовується тільки хеш-функція без додаткової ключової інформації, тому широкого затосування ця методика не має. При використанні MAC, контрольна комбінація обчислюється за допомогою секретного ключа з використанням деякого блокового шифру або хеш-функції. Прикладом MAC є режим імітовставки блокового шифру ГОСТ 28147-89, та HMAC(Hash MAC), що обчислює відбиток(хеш-суму) повідомлення фіксованої довжини. HMAC використовує спеціальний алгоритм, що поєднує додатковий секретний ключ, що відомий відправнику та приймачу інформації, та хеш функцію, результат цієї функції у підсумку залежить від вхідних даних та секретного ключа, що вирішує задачу аутентифікації джерела та цілісності даних. Завдяки швидкості калькуляції HMAC в порівнянні з ЕЦП, він має широке застосування в VPN-мережах побудованих за допомогою IPsec, openVPN, SSH2 та в службових пакетах між маршрутизаторами Cisco, забезпечуючи можливість перевірки цілісності та аутентифікації джерела для кожного пакета з інформацією.

Електронно-цифрові підписи можуть забезпечувати еквівалентні рукописним підписам та печаткам функціональні можливості, однак в більш поглибленому своєму використанні значно розширюють їх. Законодавства багатьох країн декларують юридичну тотожність щодо підписаних вручну паперових документів та електроних документів з цифровими підписами, впроваджуючи журнали подій для відстеження історії змін електроних документів задля їхнього юридичного захисту та регулювання. Цифровий підпис є додатковими даними, що обчислюються деякою математичною функцією, внаслідок застосування якої з“являється можливість перевірки авторства, справжньості, та цілісності для повідомлення, цифрового документу або ПЗ. В основі цифрових підписів є асиметрична криптографія,

тобто використовується пара ключів, з якої приватний ключ слугує для підпису, а публічний — для верифікації підпису. Цифровий підпис є асиметричною альтернативою HMAC, що не потребує обміну секретним ключем між сторонами, оскільки для верифікації достатньо публічної частини асиметричної пари окремого суб“єкта(яка може бути завантажена з сервера ключів в локальне сховище ключів як у схемі роботи сервіса gnuPG). Преференцією ЕЦП, перед HMAC, є також розподіл ключів, що для HMAC буде потребувати секретного ключа для кожної пари віддалених сторін, а для ЕЦП — достатньо одного публічного ключа для окремого суб“єкту. Крім того ЕЦП, однозначно зв“язує публічний та приватний ключі з окремим суб“єктом, в свою чергу секретний ключ HMAC може бути відомий іншим суб“єктам тому в його схемі однозначне трактування щодо дій окремого суб“єкту ускладнюється, отже юридично є менш вагомим доказом, аніж трактування в схемі ЕЦП. Необхідність підпису хеш-суми повідомлення, а не повного його змісту, з“являється зважаючи на ресурсоемність операцій асиметричної криптографії, та покликане підвищити ефективність і спростити задачі перевірки цілісності та забезпечення сумісності. Крім того, в деяких випадках, зокрема в RSA, прямий підпис байтів та доступ до публічної частини асиметричної пари може допомогти зловмиснику при розшифруванні контенту повідомлення, що був підписаний раніше (вразливість RSA щодо підпису всліпу). ЕЦП володіє властивістю автентичності, яка проявляється в складності його підробки при відсутності приватної частини ассиметричної пари і якщо процедура підписання здійснювалася за результатом неколлізійної хеш-функції. Автентичність може слугувати доказом факту підписання окремим суб“єктом документу, та інспекції стану самого документу, редагування якого після підписання призведе до неуспішної верифікації підпису. ЕЦП, на відміну від паперового підпису, не може бути перенесений в інший документ оскільки залежить від поточного стану контенту документу при його підписанні. Крім того ЕЦП гарантує факт підписування документу, що унеможливлює для

суб“єкта відмови від авторства підпису або підпису саме під таким контентом, тим самим перетвоюючись на дієвий юридично важливий доказовий засіб. Крім відсутності математичної доказовості щодо неможливості розв“язання математичних задач навколо асиметричної криптографії та коллізій хеш-функцій, практична складова вносить в ЕЦП чергову проблематику, сутність якої полягає в тому що ЕЦП створюється не самим суб“єктом, а комп“ютерною системою за інтенцією суб“єкта, функціонування якої може бути порушено зловмисним ПЗ, тому і факт підписання в цих випадках може бути спростований. Використання ЕЦП в електроному документообігу є ключовим оскільки там важливим є факт підпису деякого стану контенту об“єкта, в додаток, третя сторона, - TSP(Time Stamp Protocol) сервер, володіючи точним часом може на деякий хеш контенту поставити штамп часу, який може слугувати доказом факту існування електроного документу в окремий момент часу. Верифікація ЕЦП використовується в TLS(Transport Layer Security), коли захищене з“єднання валідує ланцюг сертифікатів(в складі якого є сертифікат віддаленої сторони), в кожному, (крім коріневого самопідписаного), з яких ЕЦП від сертифікату, що його видав. ЕЦП(ECDSA) використовується в блокчейн технологіях, зокрема в Bitcoin вона є може виступати операндом для команди вбудованої макромови(Bitcoin Script), що в простому варіанті є гарантією трати коштів їх власником, включаючись в кожну транзакцію блоку, в складному варіанті — використовується так званий «мультіпідпис», що для здійснення транзакції взалежності від контенту Bitcoin Script потребує декількох підписів власників публічних частин для арбітражу, обопільної згоди, замороження коштів на деякий термін тощо. ЕЦП(ECDSA) в криптовалюті ethereum використовується для підписування транзакцій (грошових переводів та створення смартконтрактів), також може використовуватися в коді смартконтрактів, які виконуються в віртуальній машини Ethrereum(EVM), та в транзакціях так званих «оракулів», які є носіями реальних фактів в блокчейн, генеруючи спеціальні події на які смартконтракти можуть реагувати.

Застосування асиметричної криптографії значно спростило управління ключами, однак публічні ключів суб“єктів повинні бути опублікованими, для завантаження іншими суб“єктами, до того ж повинна бути гарантія, що деякий ключ належить окремому суб“єкту, а не є згенерованим зловмисником. Для забезпечення гарантії належності публічної частини окремому суб“єкту та суміжних заходів, що роблять управління ключами більш гнучким, та деяких інших службових задач використовується інфраструктура відкритих ключів(PKI). Сама по собі асиметрична пара ключів є математичною пов“язанною ключовою інформацією, що може використовуватися в процедурах підпису та шифрування(у випадку з RSA) однак не несе ніякої службової інформації, яка потрібна для: ідентифікації суб“єкту якому вона належить; початку та сроку її дії; в якій області вона може використовуватися. Щоб ця інформації стала доступною для асиметричної пари використовується контейнер формату X.509(або цифровий сертифікат), стандарт якого передбачає низку службових аттрибутів, головним з яких є підпис, що накладається третьою стороною суб“єктом з роллю PKI. Суть методики гарантії належності ключа полягає у використанні суб“єктами функціональності третьої сторони, яка виступає в ролі PKI, що може мати наступну функціональність: збереження сертифікатів та надання публічного доступу до них; генерація сертифікатів з заявок кліентів, що містять саму публічну частину асиметричної пари та службову інформації щодо суб“єкта отримувача сертифікату, та їх підписування своїм приватним ключем з кінцевим сроком дії; ведення архіву сертифікатів; ведення списку відізваних сертифікатів(CRL), що не повинні вважатися дієвими внаслідок якихось обставини, та надання публічного доступу до нього; наддання послуги OCSP, яка дозволяє перевірити статус сертифікату в онлайн, без необхідності завантаження великого за об“ємом CRL листа та його парсингу; надання послуги TSP, що є доказом факта існування електроного документу на окремий час; дотримання правил щодо регенерації коріневого сертифікату та його

ассиметричної пари; гарантування цілісності своїх компонентів; тощо. Використання PKI відбувається під час будь-якої сессії TLS, коли браузер користувача починає валідувати ланцюг сертифікатів до яких належить віддалена сторона, починаючи від кореневого, щоб впевнитися в невідкликанні будь-кого з них, в підсумку формуючи симетричний унікальний ключ сеансу. Деякі сертифікати є коріневими, тобто самопідписаними, і лежать в локальному сховищі сертифікатів в кожній ОС, кожному з яких довіряє будь-яке ПЗ, що встановлює TLS-сессію. Отже користувачам треба буди впевненими, що в локальному сховищі знаходяться діючі сертифікати, оскільки будь-яка невідповідність може призвести до атаки SSL-mitm, внаслідок якої відбувається підміна ланцюгу сертифікатів, і зловмисний сертифікат, що є в локальному сховищі ОС жертви стає коріневим для сертифікату будь-якого сайту, і як результат зашифровані дані TLS-сессії зможе розшифрувати зловмисна сторона, яка контролює канал передачі та володіє приватним ключем від публічного ключа, що є в складі зловмисного сертифікату. Крім того, коріневим сертифікатам необхідно дотримуватися політики перегенерації своїх сертифікатів, оскільки приватна частина коріневого сертифікату може за час існування сертифікату скомпроментована, що в практичній реалізації може бути непросто оскільки впливає на дочірні сертифікати цього коріневого сертифікату. Крім того ,тяжко, з юридичного боку, трактується відповідальность в разі видачі сертифікатів скомпроментованим PKI, тому між окремим суб“єктом та PKI необхідна конвенція, що підкріплена юридичним договором.

2.2 Порівняльна характеристика сучасних методів криптографічного захисту

2.2.1 Симетричні методи шифрування

Симетрична криптографія впроваджує способи шифрування, для яких в процедурах шифрування та дешифрування інформації використовується одна ключова інформація, тобто виконується $c = E_k(m) \&\& m = D_k(c)$ та приймач

і відправник повинні мати спільний секретний ключ. Симетрична криптографія є фундаментальною складовою криптографічної науки, засоби якої є широко вживаними на практиці та мають довгий експлуатаційний термін, як наслідок вони ретельно вивчені за роки експлуатації. Вістря методик симетричної криптографії почало формуватися ще в докомпьютерну епоху. Одним з перших шифрів був шифр зсуву, який використовував Цезарь, де кожному символу відкритого тексті відповідав інший елемент з деяким параметром зсуву(якщо $k=3$, то $E_k(A)=D$). Через замалу кількість ключів в шифрі зсуву, був винайдений шифр заміни(моноалфавітний шифр), за яким кожному символу відповідав символ відповідної позиції з ключа, що збільшило простір ключів до 26!. Наступною градацією в методиці побудови шифрів був шифр Віженера(поліалфавітний шифр заміни), який замість стандартного алфавіту використовував декілька наборів символів в деякій послідомності, що ускладнило застосування частотного аналізу при криптоаналізі, оскільки шифротекст в значно меншій мірі зберігав схожість з відкритим текстом.Ідеї з історичних шифрів заміни(що в сучасній науці називаються Sbox), та шифрів перестановки(Pbox) є сучасними складовими алгоритмів симетричної криптографії. В роки Першої Світової війни криптологія почала використовувати одноразові шифр-блокноти(шифри Вернама), що передбачає довжину ключа пропорційну довжині повідомлення, що передається, та двійникову операцію XOR над відкритими даними та відповідним ключем, внаслідок правильного використання цього шифру може бути побудована абсолютно стійка система за законом Шеннона, тобто система яку не можна зламати навіть за допомогою безкінечного числа операцій та зміст її шифрованого тексту не дає ніякої інформації про відкритий текст. Умова одноразового використання ключа породжує проблему розподілу ключів, оскільки скрита передача ключа рекурентно потребує іншого ключа. Задля нівелювання проблеми розподілу ключів абсолютно стійкі алгоритми змінюються на обчислювально захищені, в яких: один ключ може

використовуватися декілька раз; невеликим за довжиною ключем шифруються довгі повідомлення. Згодом історично знадобилася механізація процесу шифрування, що призвело до створення роторних машин, найбільш відомимою з яких є «Енігма». Внаслідок розвитку цифрової техніки, з“явилася чисельна кількість блокових шифрів, що були більш криптостійкими ніж роторні машини, що в свою чергу призвело до повного виведення з побуту останніх. Історично найвагомішим для сучасної симетричної криптографії стало створення мережі Фейстеля та SP (підстановочно-перестановочна) мережі, співробітником компанії IBM Хорстом Фейстелем. Мережа Фестеля описує ітераційну структуру з однорідних уніфікованих об“єктів, в яких є змінні абстрактні компоненти(зокрема логіка використання підключів та функція впливу для раунду), які може змінювати конкретний алгоритм. SP-мережа є комбінацією з слоїв підстановки та перестановки, що використовуються багатократно.

Множину існуючих симетричних шифрів можна розділити на дві групи: потокові та блочні. Потокові шифри за один раз обробляють один оговорений елемент даних, а блочні — оперують з групою даних фіксованої величини. Зазвичай, потокові шифри слугують для шифрування великих за об“ємом даних(наприклад відео даних в реальному часі), тому важливим є алгоритмічне пристосування потокового шифру для аппаратної та програмної імплементації, при цьому можуть не накопичуювати помилки, тобто спотворений за якихось причин шматок даних не відображається на інших. Для більшості потокових шифрів характерним є застосування операції XOR до потоку відкритих даних з генератора ключового потоку ($C_i = m_i \oplus k_i$), алгоритм якого є детермінованим. Для додавання необхідної стійкості генератор ключового потоку повинен мати: великий період, зважаючи на детермінований характер свого функціонування; властивості щодо формування псевдо-випадкової послідомності, яка здається випадковою; високу лінійну складність, що є мірою якості послідомності біт. РЗЛЗЗ(регістр зсуву з лінійним зворотнім

зв“язком) є стандартною базовою широковживаною мікросхемою, що впроваджує функціонал потокового шифру. На кожному кроці свого функціонування РЗЛЗЗ, значення окремих бітів регістру пропускає через деяку лінійну функцію зворотнього зв“язку значення якої записується в крайню ліву складову регістру, в результаті чого утворюється зсув на один біт вправо, крайне праве значення регістру є результуючим. Властивості послідомності з РЗЛЗЗ напряму пов“язані з властивостями поліному, що з ним асоційований(на практиці це примітивні поліноми) . Зважаючи на лінійність функції зворотнього зв“язку РЗЛЗЗ швидко генерує потік бітів та ефективно працює в аппаратних реалізаціях, однак послідомність має слабку криптостійкість - знаючи довжину l регістру та 2l послідомних бітів з РЗЛЗЗ можна обчислити весь поліном а з ним і весь потік, що генерується, також лінійна складність РЗЛЗЗ не є високою (тобто РЗЛЗЗ генерує біти, які не є випадковими). Тому виникає необхідність знайти нелінійний спосіб експлуатації РЗЛЗЗ, комбінуючи схему задля генерації послідомності, яка має високу лінійну складність, чим користуються сучасні аппаратно-орієнтовані потокові шифри, зокрема Grain. Для історії програмно-орієнтованих потокових шифрів значимим є створення RC-4(Rivest Chipher), що приймає на вхід массив заповнений цілими числами від 0 до 255, які переставлені залежним від ключа чином внаслідок роботи з“являється потік ключів. RC-4 виявився вразливим щодо свого ключового розкладу, перші байти якого є невипадковими серед множини варіантів, завдяки чому може бути розкритий ключ. Внаслідок вразливостей RC-4 подальша експлуатація стандарту безпеки бездротових мереж WEP стала неможливою до того ж TLS, що, в деяких випадках, використовував RC-4 став також вразливим. Безпечне використання потокового шифру RC-4 передбачає якомога частішу зміну ключових даних, використовуючи конкатенацію ключової інформації з одноразовими числами(nonce) та невикористовувати початкові байти ключового потоку. Сучасними напрямами потокових шифрів є фіналісти конкурсу eSTREAM зокрема HC-128 та Grain. Також популярною є схема

потокового шифру SNOW2.0, що складається з 16 РЗЛЗЗ з 32-бітними словами (має 512-бітний внутрішній стан) та скінчений автомат з 2 32-бітними регістрами пам“яті, що слугує для породження нелінійності, реалізуючи нелінійну бієкцію між своїми регістрами. Прикладом SNOW2.0 може бути потоковий симетричний шифр «Струмок».

Блокові шифри є фундаментальною складовою криптографічних систем, та оперують блоками фіксованої, в залежность від окремого алгоритму, довжини, в результаті їхньої роботи з“являється оборотний зашифрований блок довжина якого еквівалентна вхідному. Якщо довжина інформації не кратна розміру блоку постає необхідність використання заповнення(padding) або спеціальних режимів окремого блокового шифру. Оборотність блокового шифру стає можливою внаслідок однозначної відповідності відкритого тексту та шифрованому тексту в множині деякого ключа. Практично всі сучасні блокові шифри є послідомним застосуванням слабкого блочного шифру, що називається раундом. Послідомність раундів якісного блокового шифру має забезпечити лавинний ефект що є наслідком якісної дифузії та конфузії за Шенноном. Дифузія є розподілом збитковості в статистиці вхідних даних по всій структурі вихідних даних та забезпечується Р-блоком, конфузія є ускладнення залежності ключа від вихідних даних — реалізується через застосування S-блоків.

Існує чисельна кількість блокових шифрів, які в своїй основі використовують мережу Фейстеля. Преференціями при використанні цієї структури є: простота аппаратної та програмної реалізації; вивченість алгоритмів на її основі; операції шифрування та дешифрування можливо зробити однаковими відрізняється тільки порядок підключів; гарантія переміщування правої та лівої частини тексту; абстрактна функція F не обов“язково має бути зворотньою. Алгоритм DES(Data Encryption Standart) був розроблений Х.Фейстелем, та має більш ніж 40-літню історію експлуатації та використовує процедуру з 16-ти раундів та 16 48-бітових підключа, що отримуються з основного(в наявності стандарту є перелік слабких ключів). На

кожному раунді відбувається XOR даних з підключем, що гарантує переміщування ключа та даних, а S-матриці забезпечують нелінійність, унеможливлюючи використання методів лінійної алгебри при зламі — в сумі отримується прийнятний рівень дифузії(при зміні одного біту у вхідному значенні функції F відбувається зміна декількох біт в вихідному). Обмеження щодо простору ключів в 2^{56} , розмір блоку в 64-біта роблять DES не придатним до сучасних умов експлуатації. 3DES збільшує простір ключів до 2^{168} через виконання $DES(k_3; DES(k_2; DES(k_3; m)))$ однак це значно уповільнює роботу і без того повільного, за сучасними мірами, DES, крім того не вирішує проблему замалого розміру блоку. ГОСТ 28147-89 є ще одним представником створеними за ітераційною мережою Фейстеля, який використовує простір ключів 2^{256} , 32 раундову процедуру(в режимі імітовставки 16), розмір блоку 64-біта та декларує декілька S-матриць, що можуть слугувати як додаткова ключова інформація. На кожному раунді крім операцій що асоційовані з мережою Фейстеля, відбувається: операція XOR з ключем; заміна відносно обраного Sbox; зсув на 11-біт вліво.

Ще однією базовою схемою побудови є SP-мережа, яку використовує переможець конкурсу AES(Advanced Encryption Standart) шифр Rijndael. AES є блоковим шифром зі змінною довжиною блоку та змінною довжиною ключа(обопільно незалежно 128-, 192-, 256-біт), що використовує в своїх елементарних операціях поле Галуа $GF(2^8)$, що побудоване як розширення поля $GF(2)$ за корнями поліному $m(x)=x^8+x^4+x^3+x+1$. AES є послідомністю ітерацій SP-мережі — раундів, кількість яких залежить від довжини ключа(10, 12 або 14), що виконуються над проміжною структурою(або станом термінологією теорії кінцевих автоматів), яка є матрицею розміром 4x4, 4x6, 4x8 байт в залежності від блоку. На вхід AES подається блок відкритого тексту, який складається з підключем раунду, потім кожний байт з результуючих подається на вхід однакових S-матриць, які, в свою чергу, нелінійно відображає

байтові вхідні значення в байтові вихідні. Вихідні байти переставляються в деякому порядку, і кожна група з деяких байт перемішується, зворотньо зсуваючись залежно від своєї позиції. Прийнятний рівень дифузії та конфузії в досягається вже в 7 раунді ітерацій, а відсутність слабких ключів та неможливості застосування лінійного та дифференціального криптоаналізу будує для AES гарну криптологічну репутацію. Використання SP-мережі унеможливлює уніфікацію процедури шифрування/дешифрування, однак распаралелювання процедури раундів є більш здійсненим ніж при використанні класичної мережі Фейстеля. Існує набір розширених аппаратних команд(AES-NI) до процессорів компанії Intel, однак враховуючи відомості Е.Сноудена, про наявність «чорних ходів» для Агенства національної безпеки США, у мотивації щодо використання таких прискорювачів з“являється деякий скептицизм. До того ж рекомендований розмір ключів для AES є 256-біт оскільки існує квантовий алгоритм Говера(Gover search algorithm), який має складність $O(\sqrt{n})$ (де n — множина ключів) , повний перебір яким для AES-128 буде означати замість 2^{128} , 2^{64} операції. Вітчизняний стандарт ДСТУ 7624:2014(«Калина»), що планується на заміну ГОСТ 28147-89, також побудований за допомогою SP-мережі, який підтримує боки та ключі розміром 128-, 256- та 512-біт. Однак його сильні криптографічні властивості на даний час під питанням.

Режими блочних шифрів насамперед покликані для унеможливлення зчитування потока повідомлень для зловмисника, хоча і є можливість забезпечення цілісності та в деякій мірі конфеденційності в деяких режимах. Більшість режимів блокових шифрів потребують довжини повідомлення, що є кратною розміру блоку конкретного шифра, тому виникає необхідність додавання заповнювання(padding). Існує чисельна кількість стандартів щодо формування заповнювання(бітові і байтові), найпопулярнішим є PKCS#7 padding. За деякого обраного стандарту заповнення, обидві сторони не враховують заповнення як корисні дані та прибирають його при знаходженні.

Низка режимів потребує вектора ініціалізації(initialization vector IV) - деякого початкового значення, рекомендації щодо формування якого наступні: фіксований VI — якщо повідомлення будуть починатися з однакової послідомності, то шифротекст також в перших блоках також буде мати ці послідомності; випадковий IV — з точки зору безпеки є прийнятним однак з“являється проблема передачі IV між сторонами; застосування оказії(nonce) — унікального числа, яке в рамках використання окремого ключа повинно використовуватися один раз. Для деяких симетричних шифрів є спеціальні режими, зокрема режим вироблення імітовставки в ГОСТ 28147-89 та ДСТУ 7624:2014(«Калина») CMAC, що покликані забезпечити цілісність не використовуючи при цьому початковий вектор иниціалізації, інші, зокрема CBC-MAC, GCM-GMAC, впроваджують подібний функціонал з використанням початкового вектору ініціалізації — в результаті в повідомленнях з“являються коди аутентичності.

Самим простим режимом шифрування є ECB(electronic codebook) або електроної шифровальної книги(в деяких випадках режим простої заміни), для якого є характерним: $c_i = E(K, m_i), i = 1...k$. ECB шифруючи кожен блок окремо, дозволяє розпаралелити процес шифрування однак не рекомендований оскільки однакові блоки відкритого повідомлення будуть однаковими і в шифрограмі, що в практичному застосуванні є неприйнятним. Режим CBC(cipher-block-chaining) зчеплення шифрованих блоків, що характерізується формулою $c_0 = IV; c_i = E(K, m_i \oplus c_{i-1})$, складаючи кожен блок відкритого тексту з попереднім блоком шифрованого тексту. CBC нівелює недоліки ECB при правильному формуванні IV, накопичує помилки в попередніх блоках на всі наступні, однак унеможливлює распаралелювання процесу шифрування. Режим зворотнього зв“язку по виведенню OFB(output feedback) не використовує в якості вхідних даних блокового шифру повідомлення, а є генератором псевдовипадкового потоку байтів(ключового потоку), який за допомогою XOR складається з відкритим текстом для отримання шифрованого тексту — відповідно є

потоковим шифром, що транслює одну бітову помилку в блоці шифротексті в однобітову помилку в відкритому тексті. OFB характерізується формулою $k_0 = IV; k_i = E(K, k_{i-1}); c_i = m_i \otimes k_i$, використовує функцію шифрування як в процесі шифрування так і в процесі дешифрування та, зважаючи на свою поточність, не потребує використання заповнення. Режим зворотнього зв“язку по шифротексту CFB(cipher feedback) є схожим на OFB та характерізується $c_0 = IV; c_i = m_i \otimes E(k, c_{i-1})$, відрізняючись тим, що помилка в криптограмі впливає як на блок в якому вона допущена, так і на наступний. Режим лічільника CTR(counter) також базується на застосуванні поточного шифра(як CFB та OFB) та характерізується $k_i = E(k, nonce \| i); c_i = m_i \otimes k_i$, будуючи оказію шляхом конкатенації значення оказії nonce зі значенням лічільника, відповідно необхідно подбати про корректний розмір оказії та місце для лічільника і, та гарантію щодо одноразового використання пари «ключ-оказія».

2.2.2 Асиметричні методи шифрування

Історія асиметричної криптографії є відносно новітньою зважаючи на історичний аспект симетричної. Офіційно вона почалась з роботи У.Діффі та М.Хелмана(та опосередковано Р.Меркла) «Нові напрями в криптографії» («New directions in cryptography») в 1976 році, що призвела до збільшення вченого електорату та інтересу до криптографічної науки в цілому, відкривши нову область в криптографії — криптографію з відкритим ключем. Вістря та вплив на подальший розвиток в криптографії цієї роботи були високо оцінені світовою комп“ютерною спільнотою в результаті чого її автори отримали премію Тьюринга в 2015 році. Хоча зважаючи на політичну зацікавленність відносно будь-якого періоду розвитку криптографії, можна менш скептично ставитися до неофіційної версії, за якою відкриття та перше використання однонаправлених функцій в криптографічному аспекті належить британським спецслужбам. Згодом, в 1976 році, з“являється перша асиметрична криптосистема система RSA(Rivest Shamir Adleman), за яку розробники отримали премію Тьюрінга в 2002 році, та дефакто є найвживанішою

асиметричною криптосистемою сучасності. В 1985 році Т. Ель-Гамалем була запропонована схема асиметричного шифрування та перевірки підпису, яка вдосконалювала схему Діффі-Хелмана. В тому ж році Н.Кобліцем було пропоновано застосування еліптичних кривих. В 1989 році Ш.Гольдвассер, С.Мікалі, Ч.Рекофор розробили новаторську концепцію інтерактивних систем з нульовим розголошенням в статті «Знання та складність інтерактивної системи з доказом», розробивши згодом першу систему ймовірнісного шифрування, яка була високо оцінена комп“ютерною спільнотою, а її автори отримали премію Тьюринга в 2012 році. В 1994 році П.Шор розробив поліноміальний алгоритм розкладу великих чисел на множникі для квантового комп“ютера. Наприклад для зламу RSA(знаходження приватного ключа по публічному) довжиною 2048 біт необхідно підтримувати 4096 зв“язаних кубіт. Внаслідок чого в 2016 році NIST запустив конкурс на стандартизацію алгоритмів постквантової асиметричної криптографії. Подальші кроки в побудові квантових комп“ютерів великими корпораціями, зокрема IBM(зокрема реліз першого квантового комп“ютера для домашнього використання в 2019 році потужністю 20 кубіт), поставили під загрозу використання всіх широковживаних алгоритмів асиметричної криптографії.

Головним лейтмотивом асиметричної криптографії є використання різних ключових даних в задачах дешифрування/шифрування та формування /перевірки підпису — приватного та публічного ключа, що математично пов“язані одностороннім зв“язком. Публічна частина є відкритою, та може бути опублікована в загальнодоступному місці, що використовується для шифрування або перевірки підпису, також з неї неможливо отримати приватну частину - що проявляє властивості однонаправленої математичної функції. Приватна частина є секретною інформацією, що використовується при дешифруванні та формуванні підпису. Однонаправленість математичного перетворення є ключовою властивістю асиметричної криптографії є ключовим її аспектом, та полягає в складності її звертання без знання ключової

інформації(приватної частини). Найбільш широковживаними в поточних криптографічних схемах та всебічно дослідженими однонаправленими функціями є: задача розкладу цілих чисел на множникі(difficulty of integer factorization); задача дискретного логарифму в кінечних полях(DLP); задача дискретного логарифму в групі точок еліптичної кривої(ECDLP — eliptic curve discrete logariphm problem). Окрема схема асиметричної криптографії може вирішувати задачі щодо: шифрування повідомлень; формування підпису(простого та сліпого); обміну спільними ключами; доказу з нульовим розголошенням, тощо.

Протокол обміну ключами Діффі-Хелмана(Diffie-Helman Key exchange protocol) є вирішує задачу обміну спільним секретним ключем між учасниками, які використовують незахищені канали зв“язку. В класичній реалізації алгоритм використовує DLP проблематику, та передбачає публічними параметрами мультіплікативну групу чисел кінцевого поля p(число від 1024 біт) та примітивний корінь модуля(генератор) p — g(надійне просте число). Складність DH в будується навколо двох задач(G — кінцева абелева група): задача дискретного логарифмування — дано $A,B\in G$ знайти x такій що $B=A^x$; дано $A\in G, B=A^x, C=A^y$ необхідно знайти $D=A^{xy}$. Внаслідок чого складність DH можна свести до DLP. Кожен користувач генерує секретний ключ a, та відправляє $g^a \bmod p$ до іншого учасника, який в свою чергу генерує секретний ключ b та відправляє $g^b \bmod p$ внаслідок чого спільним секретним ключем буде $s=g^{ab} \bmod p$. Еліптичний варіант протоколу(ECDH) використовуючи ECDLP, передбачає публічно доступні доменні параметри еліптичної кривої, серед яких є генератор підгрупи групи точок еліптичної кривої(base point) G. Аналогічно кожен користувач генерує $[a]\cdot G$ та $[b]\cdot G$, внаслідок чого спільним секретним ключем буде $s=[a]\cdot([b]\cdot G)$. Основна проблематика навколо протоколу DH існує внаслідок його вразливості до атаки MITM, оскільки жодна зі сторін не знає з ким конкретно вона взаємодіє - з зловмисником або легітимним користувачем. Тому бажаним є використання

ресурсів третьої сторони, що може підтвердити персону кожної зі сторін(використання PKI), які взаємодіють між собою. Реалізація DH повинна передбачати перевірку значеннь $g!=1, g^a!=1$ оскільки зловмисник може змінити це значення викривши загальний секретний ключ, та правильний порядок підгрупи g^a оскільки зловмисник може навмисно його понизити. Протокол DH використовує чисельна кількість сучасних мережевих протоколів, зокрема IKE, SSH, TLS, тому для уніфікації існує документ RFC5114, що містить рекомендовані параметри для протокола DH.

Алгоритм RSA є найшироковживанішим алгоритмом для схем асиметричної криптографії сьогодення. RSA є універсальним засобом, оскільки впроваджує формування цифрових підписів та шифрування. Його вістря побудоване на проблематиці факторизації чисел, дослідження якої почалося ще за часів Евкліда, тому для сучасності є ретельно дослідженим питанням. Складність RSA будується навколо двох задач: задача факторизації — знати p та q по відомому N($N=p\cdot q, n=?, p=?$); дані числа c та e, де $НОД(e,(p-1)(q-1))=1$, знайти m таке що $m^e=C(mod\,N)$. Тому вважається що складність RSA зводиться до факторизації чисел, хоча вірогідно 2 задача може вирішуватися іншим чином. Функціонування RSA зводиться до: генерації двох взаємнопростих чисел приблизно однакових за довжиною $N=p\cdot q$ зважаючи на шифруючу константу(E = 65537) та співвідношення $НОД(E,(p-1)\cdot(q-1))=1$; пара (N; E) публічний ключ; приватний ключ d з“являється в результаті застосування розширеного алгоритму Евкліда таке що $E\cdot d=1(mod(p-1)(q-1))$; (d;p;q) приватний ключ. Операції що впроваджує алгоритм RSA наступні: шифрування $C=m^E(mod\,N)$; дешифрування $m=C^d(mod\,N)=(m^E)^d(mod\,N)$; формування підпису $s=m^d(mod\,N)$ - пара (m; s); верифікація підпису зводиться до перевірки співвідношення $s^E=m(mod\,N)$. Параметр E значно впливає на складність обчислення операції шифрування та верифікації підпису тому раціональним є формування множників за умови прийнятного значення E=65537, оскільки великі значення будуть впливати на тривалість операції,

замалі — допоможуть зловмиснику розшифрувати повідомлення якщо не використовується хешування та воно значно менше n(операція взяття за модулем не задіяна). Розмір n, враховуючи сучасні умови експлуатації, повинен бути більшим 1024 біт. Експлуатація RSA маючи чітку математичну структуру не повинна передбачати використання прямих підписів(без хешування) оскільки внаслідок мультіплікативних властивостей стає можливим розшифровування повідомлення(m_p - підписане раніше зашифроване повідомлення, r — множник взаємнопростий з N): зловмисник формує $m_{cur}=m_p\cdot r^E(mod\,N)$; підписувач $s_{cur}=m_{cur}^d(mod\,N)$; зловмисник $m=s_{cur}\cdot r^{-1}(mod\,N)$. До того ж зловмисник маючи підписи на повідомлення m_1,m_2 зможе зформувати підпис для $m_3=m_1\cdot m_2(mod\,N)$. Зважаючи на математичну сутність RSA та складність щодо обчислення арифметикі залишків та зведення в шифруючу та дешифруючу експоненти повідомлень для комп“ютера, швидкість цього алгоритму є невисокою, тому шифрування без використання комбінованих схем не буде прийнятним з точки зору швидкості обрахунків. Підпис RSA є одним з найбільших за об“ємом з алгоритмів цифрового підпису, крім того алгоритм не передбачає ефемерний ключ, внаслідок чого деякій інформації буде однозначно відповідати деякий підпис. Ключі RSA не можуть бути подані в компактному(зтисненому) вигляді на відміну наприклад від ECDSA. Крім того генерація ключів потребує генерації двох псевдовипадкових великих числа та їх перевірки на взаємну простоту, що як правило вимагає від реалізації декількох поліноміально-складних перетворень. Напродуктивнішим методом факторизації цілих чисел є загальний метод решета числового поля, що має субекпотенціальну складність, тому RSA-4096 має достатній рівень криптостійкості, якщо не брати до уваги квантовий алгоритм Шора.

В 1991 році NIST запропонував DSA(Digital Signature Algorithm) як DSS (Digital Signature Standart). DSA на відміну RSA впроваджує роботу тільки з цифровими підписами, не забезпечуючи шифрування. Класична версія DSA

створює менший за об“ємом підпис ніж RSA та потребує менше зусиль при генерації ключів, криптостійкість якої базується на DLP в кінечних полях. DSA є алгоритмом підпису з доповненням, що складається з двох 160-бітних значень(R та S), одне з яких(R) є фукцією 160-бітного випадкового числа k(ефемерного ключа), що змінюється з кожним новим повідомленням, інше(S) — є функцією від повідомлення секретного ключа x, що належить підписувачу, числа R та ефемерного ключа. DSA передбачає наявність публічних параметрів(параметрів домену) - простого числа Q, великого числа P((P-1)/Q), та великого числа G, такого що $G^Q=1(mod\,P)$. Користувач генерує приватний ключ $x(0<x<Q)$, публічний відповідно буде $Y=G^x(mod\,P)$. Для підпису користувач виконує наступні дії щоб сформувати (R;S): обчислює хеш H=h(m) повідомлення та обирає ефемерний ключ $k(0<k<Q)$; обчислює $R=(G^k(mod\,P))(mod\,Q)$ та $S=(\frac{H+xR}{k})(mod\,Q)$. Для перевірки підпису передбачається наступна процедура: H=h(m); $A=H/S(mod\,Q), B=R/S(mod\,Q)$; $V=(G^AY^B(mod\,P))(mod\,Q)$; підпис корректний якщо $V=R$. Швидкість функціонування DSA в рамках криптостійкості ($P>2^{1024}, Q>2^{160}$) значно зменшується оскільки використовує більш складну математичну процедуру ніж RSA (потребує два возведення до степені 1024-бітного числа), однак таким чином з“являється криптостійкість щодо атак решетом в числовому полі.

Варіант DSA, що побудований на еліптичній кривій(ECDSA) в криптостійкості базується на ECDLP, яка вважається складніше за DLP, оскільки не існує субекспотенціальних алгоритмів для її вирішення, отже єдиний метод дискретного логарифмування в загальних еліптичних кривих — паралельна версія ро-методу Полларда що має складність $O(\sqrt{q})$. Ця обставина дозволяє в еліптичних кривих використовувати значно менші за об“ємом ключі $q\approx2^{160}$, що значно збільшує швидкість роботи алгоритму ECDSA. Алгоритм працює з циклічними абелевими групами $\approx2^{160}$, що породжені генератором G порядку n над окремим рівнянням кривої(Curve) та

нейтральним елементом O, що відповідно є публічними параметрами(доменами еліптичної кривої). Приватним ключем є випадкове число $d_A \in [1; n-1]$, публічним - $Q_A = d_A \cdot G$. Генерація підпису передбачає: генерацію ефемерного ключа $k \in [1; n-1]$ та точки $(x_1; y_1) = k \cdot G$; генерацію пари $(r; s), r = x_1 (mod\, n), s = k^{-1}(H(m) + rd_A) mod\, n$. Верифікація передбачає наступне: перевірка $r \in [1; n-1], s \in [1; n-1]$; обчислення $u_1 = H(m) s^{-1} mod\, n, u_2 = r s^{-1} mod\, n$ та точки $(x_1; y_1) Q_A = u_1 \cdot G + u_2 \cdot Q_A$; перевірка $r \equiv x_1 mod\, n$. Алгоритм ECDSA завдяки меншим розмірам ключа працює швидше ніж RSA та DSA, маючи при цьому аналогічні криптографічні властивості. Для ECDSA може використовуватися упакований формат ключів — пакується тільки x та його парність, що широко використовується в криптовалютах, крім того підпис має невеликий постійний, взалежності від окремої кривої, розмір(в загальному випадку 65 байт). Для алгоритмів DSA та ECDSA критичним є секретність, унікальність та випадковість ефемерного ключа k для окремого хешу повідомлення, в іншому випадку зловмисник може обчислити приватний ключ d_A вирішивши нескладну систему рівнянь. Розкриття ключа мало місце в 2010 році в приставках Sony Play Station оскільки розробники ефемерний ключ k зробили статичним, та в деяких імплементаціях Java в класі SecureRandom, де k мав коллізії. Алгоритм ECDSA поступово переважає RSA в популярності в послугах щодо підпису, і є найраціональнішим доквантовим алгоритмом сучасності.

Крокування в побудові квантових комп“ютерів створила проблему для методів асиметричної криптографії сьогодення, внаслідок чого NIST запустив конкурс для постквантових алгоритмів, що мають стати стандартами. Відбувся другий тур конкурсу, за результатами якого залишається 26 конкурсантів. Більшість кандидатів є криптографічними примітивами, що використовують решітки. Другою за популярністю серед кандидатів є проблематика з декодування повних лінійних кодів. Третьою є — схеми, що побудовані на

рішеннях рівнянь на багатомірних поліномах над кінечним полем. Крім того є кандидат SIKE, що впроваджує аналог DH, що базується на блуканні в суперсингулярному ізогенному графі, що побудований двома еліптичними кривими.

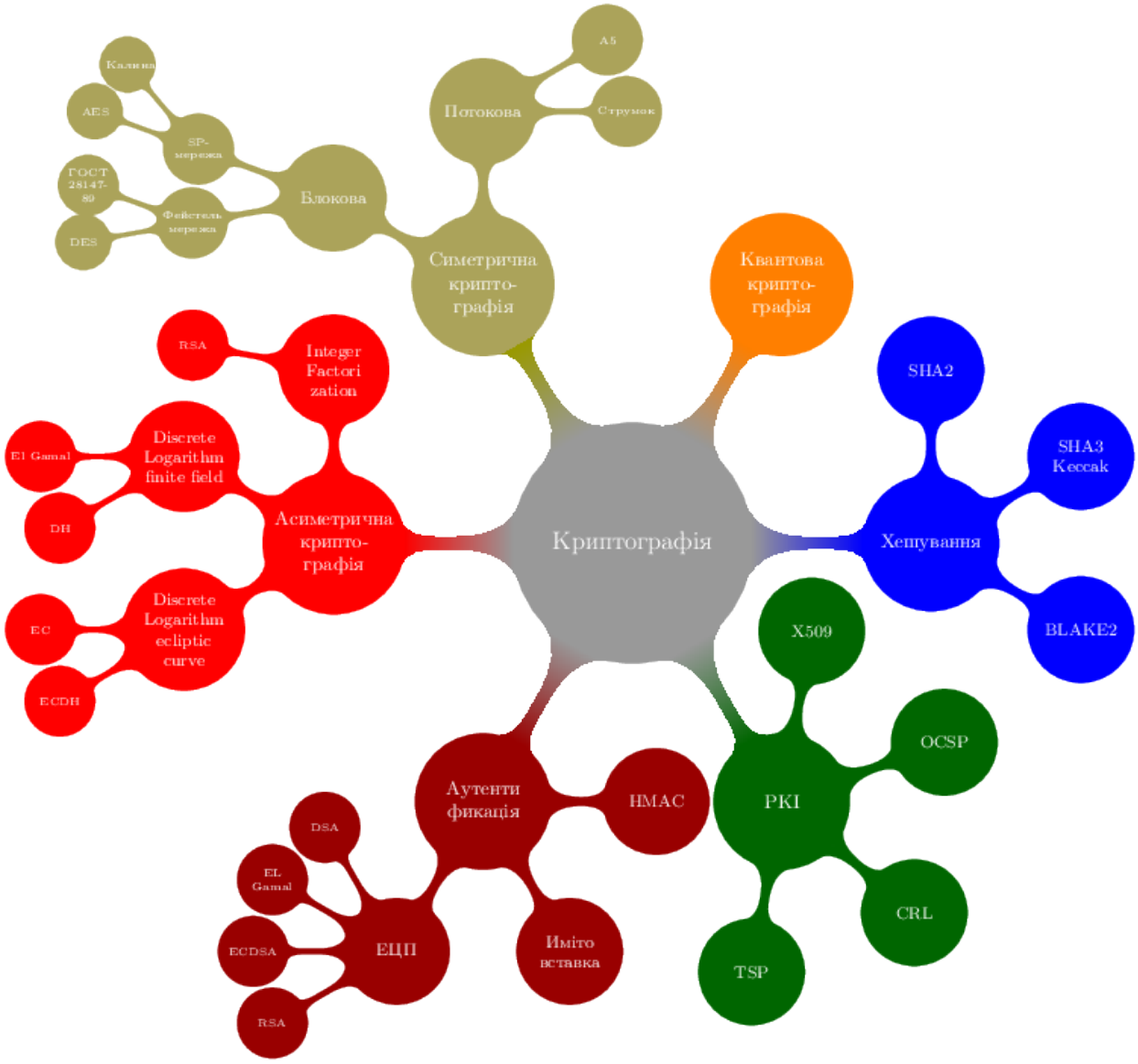


Рис 5. Напрями криптографії

2.2.3 Критеріальне порівняння методів шифрування

За критерієм криптостійкості сучасні методи шифрування як симетричного так і асиметричного можна ототожнювати, однак ця тотожність потребує різних довжин ключів. Для симетричних шифрів як правило рівень криптостійкості декларується простором значень ключа, наприклад AES-256

2^{256} , та означає найнеприємнішу з точки зору зловмисника операцію повного перебору, якщо не існує методик щодо зниження кількості операцій. Рівень криптостійкості асиметричної системи залежить від складності(O()) ефективних методик, якими можна атакувати односторонню функцію асиметричної криптосистеми деякого типу, та збільшується використанням більших за довжиною ключів. Ототожнення рівня криптостійкості для методів шифрування означає тотожність зусиль щодо їхнього зламу, та формується авторитетними організаціями, зокрема NIST, як рекомендації. Так, за NIST, AES-256 бітами відповідає: 15360-бітна RSA; 15360-бітний DLP; 512-бітний ECDLP. Найпродуктивнішою доквантовою методикою для RSA є субекспоненціальний алгоритм решета, до якого може бути зведена і задача DLP тому вони мають однакове значення. Найпродуктивнішою доквантовою методикою для ECDLP є алгоритм ро-Полларда, що є експоненціальним тому ключ збільшується несуттєво. Задачі факторизації чисел, DLP, ECDLP можуть оптимально вирішуватися через поліноміальний квантовий алгоритм Шора, для AES ефективним є квантовий алгоритм Гровера зі складністю $O(\sqrt{N})$.

За критерієм швидкості функціонування значна перевага на боці симетричних шифрів оскільки в їхній математичній складовій присутні лише прості математичні операції, які впроваджує будь-яка архітектура сучасності, причому ці команди можуть бути виконані за такт роботи процесора, та не потребують виділення великої за об“ємом пам“яті. Особливо простими з точки зору реалізації є шифри на основі мережі Фейстеля, оскільки як правило мають уніфіковану операцію шифрування/дешифрування. З точки зору продуктивності переважають шифри на основі SP-мережі оскільки мають більш високу здатність до розпаралелювання. Використання асиметричних шифів впроваджує в кінцеву архітектуру поняття «великих цілих», що потребують додатково виділеної пам“яті, операції з ними потребують численну кількість тактів процесора. Математичні структури асиметричних шифрів мають складні операції, що з боку комп“ютера є дуже дорогими(зокрема введення великих

чисел в експоненти), а для мікроархітектур, в деяких випадках, взагалі нездійсненими.

За критерієм розподілу ключів асиметричні шифри значно переважають симетричні шифри, хоча і з деякими нюансами. Функціонування симетричних шифрів передбачає знання секретної інформації сторонами, через що породжує проблематику щодо передачі ключової інформації між ними через незахищені канали зв“язку. Крім того в симетричних криптосистемах з великою кількістю користувачів кількість секретних ключів значно зростає оскільки кожен сеанс між різними учасниками потребує генерації ключа. В асиметричних криптосистемах відкриті ключі є публічною інформацією, яка може бути відома всім что бажає захищеного обміну інформацією з власником, тому кількість ключів не зростає. В асиметричних криптосистемах виникає проблема гарантування відповідності публічного ключа легітимному користувачу(уникання підміни), яка вирішується засобами PKI.

Широковживаною чеснотою асиметричної криптографії є сервіс аутентифікації сторони, що реалізується використанням ЕЦП. Симетрична криптографія також впроваджує аутентифікацію сторін(наприклад HMAC) однак секретна інформація завдяки, якій вона уможливлюється не є персональною, ЕЦП ж однозначно персоніфікує сторону.

Деяким компромісним варіантом є використання комбінованих криптосистем в складі яких є симетричні та асиметричні засоби криптографії. В комбінованих криптосистемах велика за об“ємом інформація шифрується сессійним симетричним ключем, що передається між сторонами в зашифрованому вигляді внаслідок застосування асиметричного алгоритму. Таким чином нівелюється низька швидкодія асиметричних засобів та проблематика розподілу секретного ключа між сторонами, при цьому залишаються чесноти асиметричної криптографії.

Назва	Кількість раундів	Довжина ключа, біт	Розмір блоку, біт/розмір стану(для поточних шифрів)	Схема (F -мережа Фейстеля, SP — SP-мережа)	Середній час шифрування ~300MB файла з мінімальною довжиною ключа (java bouncy castle framework), сек
AES	10,12,14	128,192, 256	128,192,256	SP	11.5
Гост 28147-89	16, 32	256	64	F	28.0
blowfish	16	32-448	64	F	12.5
camellia	18, 24	128,192, 256	128	F	13.8
cast5(128)	12,16	40-128	64	F	13.5
cast6(256)	48	128, 160, 192, 224, 256	128	F	14.6
tripledes	48	56, 112,168	64	F	31.2
des	16	56	64	F	14.8
idea	8.5	128	64	F	18.3
noekeon	16	128	128	Самообернені перетворення	16.3
rc2	16+2	8-1024	64	Незбалансований F	20.3
rc4	1	40–2048	2064	-	8.8
rc5	1-255	1 - 2040	32,64,128	F	9.5
rc6	20	128,192,256	128	F	11.4
seed	16	128	128	F	14.6
serpent	32	128,192,256	128	SP	23.2
skipjack	32	80	64	Незбалансований F	32.2
tea	32(варіативний)	128	64	F	15.6
twofish	16	128,192,256	128	F	14.4
xtea	32(варіативний)	128	64	F	15.6
ДСТУ-7624:2014 Калина	10,14,18	128,256,512	128,256,512	SP	35.0 (тест на С)
HC-128	1	128	(1024*32)*2	-	9.3
HC-256	1	256	(1024*32)*2	-	9.9

Salsa-20	20	128,256	512	-	11.0
ДСТУ-8845 струмок	1	256,512	1152	-	8.9(тест на С)

Таб.1 Порівняльна характеристика симетричних шифрів

Назва	Пробламатика однонаправленої функції	Послуга	Розмір ключа, біт	Криптостійкі рекомендації NIST	Час формування ЕЦП/шифрування/генерації спільного ключа, сек	Час верифікації ЕЦП / дешифрування, сек (openssl)
RSA	Integer factorization	Шифрування/ ЕЦП	512 1024 2048 3072 4096 7680 15360	Використання ключів не менше 3072 біт	0.000032 0.000068 0.000471 0.001460 0.003228 0.029703 0.148971	0.000002 0.000004 0.000014 0.000030 0.000050 0.000167 0.000639
DSA	DLP	ЕЦП	512 1024 2048	Не розглядається як кандидат використання	0.000046 0.000079 0.000199	0.000031 0.000062 0.000179
ECDSA	ECDLP	ЕЦП	224(nistp224) 256(nistp256) 384(nistp384) 521(nistp521)	Рекомендоване вживання nistp384	0.0009 0.0001 0.0025 0.0058	0.0008 0.0002 0.0018 0.0041
ECDH	ECDLP	Обмін ключами	224(nistp224) 256(nistp256) 384(nistp384) 521(nistp521) 253(X25519) 448(X448)		0.0003 0.00031 0.0008 0.0018 0.00033 0.0004	- - - - - -
DH	DLP	Обмін ключами	1024-160 2048-224 2048-256 ffdhe2048 ffdhe3072 ffdhe4096 ffdhe6144 ffdhe8192	Використання ключів не менше 3072 біт	0.0000 0.0000 0.0001 0.0001 0.0001 0.0002 0.0004 0.0006	- - - - - - - -

Таб.2 Порівняльна характеристика асиметричних шифрів

2.3 Обгрунтування вибору криптографічних методів для реалізації захисту повідомлень у середовищі ОС Android

Функціонування додатку, наслідуючи класичну комбіновану схему, яка пришвидшує операції шифрування та дешифрування, потребує від множини криптографічних методів трьох сервісів(послуг) — симетричне шифрування, асиметричне шифрування та ЕЦП, задля вирішення задач аутентифікації джерела, цілісності та конфеденційності інформації щодо повідомлень. В деяких задачах(формування AES-ключа з паролю) використовується криптографічне хешування з підсоленням. Додаток підтримує два профіля шодо реалізації вирішення цих задач — ключі RSA(проблематика факторизації чисел) та ключі EC(проблематика ECDLP). Для гарантування прийнятного рівня криптостійкості ключі RSA мають розмір 2048 біт, EC-ключі використовують стандартну еліптичну криву secp256k1, що рекомендована NIST та широко поширена в сучасних схемах ЕЦП(зокрема криптовалюті Bitcoin).

Задача аутентифікації джерела та цілісності інформації реалізується через використання ЕЦП по алгоритмам RSA або ECDSA, які підтримує стандарт PKCS7 контейнеру CMSSignedData. Контейнер CMSSignedData, що формується додатком, використовує хешування SHA256, тому повною схемою підпису буде для ECDSA — ecdsawithsha256, для RSA — rsawithsha256. Ці ж схеми ЕЦП використовують контейнери X509, що інкапсулюють токен та публічний ключ асиметричної пари(RSA та EC). Опціонально до контейнеру CMSSignedData додаток може вкладати додатковий атрибут та необхідні сертифікати, що отримані з Timestamp сервера на SHA256-дайджест підпису підписувача повідомлення, для доказу деякого стану контенту в часовий період формування підпису.

Симетричне шифрування використовується додатком в задачах: 1) шифрування контенту повідомлення; 2) шифрування приватних частин асиметричних пар, що зберігаються в БД пристрою. Зважаючи на сучасні методи симетричного шифрування, найраціональнішими є схеми SP-мережі,

тому для задачі 1 було обрано AES-128(для пришвидшення операції шифрування та розшифровки(10 раундів)), а для 2 — AES-256(для більшого рівня захищеності(14 раундів)). В обох випадках використовується режим шифрування зчеплення блоків шифротексту(CBC).

Асиметричне шифрування використовується додатком в задачі формування шифрованого симетричного сесійного ключа, яким шифрується контент повідомлення. В схемі RSA сессійний випадковий ключ шифрується введенням в степень шифруючої експоненти(публічного ключа) на кінечному полі, внаслідок чого додаток створює стандартний PKCS7-контейнер EnvelopedData, який описує для отримувача службову інформацію щодо використаних алгоритмів шифрування — параметрів комбінованої схеми шифрування. В схемі EC використовується схема ECIES(ecliptic curve integrated encryption scheme), що широко застосована в сучасних схемах шифрування(зокрема в платіжній NFC-системі GooglePay), оскільки стандартною послугою з якою використовуються EC-ключі є тільки формування ЕЦП(ECDSA) та протокол обміну ключем(ECDH). Функціонування шифрування в ECIES полягає в наступному(домені параметри EC — (n, G) ,публічні ключі $K_a = k_a \cdot G, K_b = k_b \cdot G$, опціональна інформація S_1, S_2): генерація $r \in [1; n-1]$ та обчислення $R = r \cdot G$; формування загального секрету $S = (P_x), P = (P_x, P_y) = r \cdot K_b$; використання KDF-функції для обчислення симетричного ключа та MAC-ключа - $k_e \| k_m = KDF(S \| S_1)$; шифрування повідомлення $c = E(k_e, m)$; обчислення MAC $d = MAC(k_m; c \| S_2)$; вихід функціонування схеми $(R; c; d)$. Для дешифрування в ECIES необхідні наступні кроки: обчислення загального секрету $S = (P_x), P = (P_x, P_y) = r \cdot k_b (P = k_b \cdot R = r \cdot k_b \cdot G)$; обчислення ключа шифрування та MAC-ключа $k_e \| k_m = KDF(S \| S_1)$; виконання умови $d = MAC(k_m, c \| S_2)$; дешифрування повідомлення $m = D(k_e, c)$. Функція KDF(Key Derivation Function) формує пару секретних ключів на основі секретного значення S ,

опціонального S_1 , а функція MAC формує збиткову інформацію, яка слугує для перевірки цілісності переданих даних використовуючи k_m та опціональне S_2 . Таким чином в схемі ECIES забезпечується шифрування та перевірка цілісності даних.

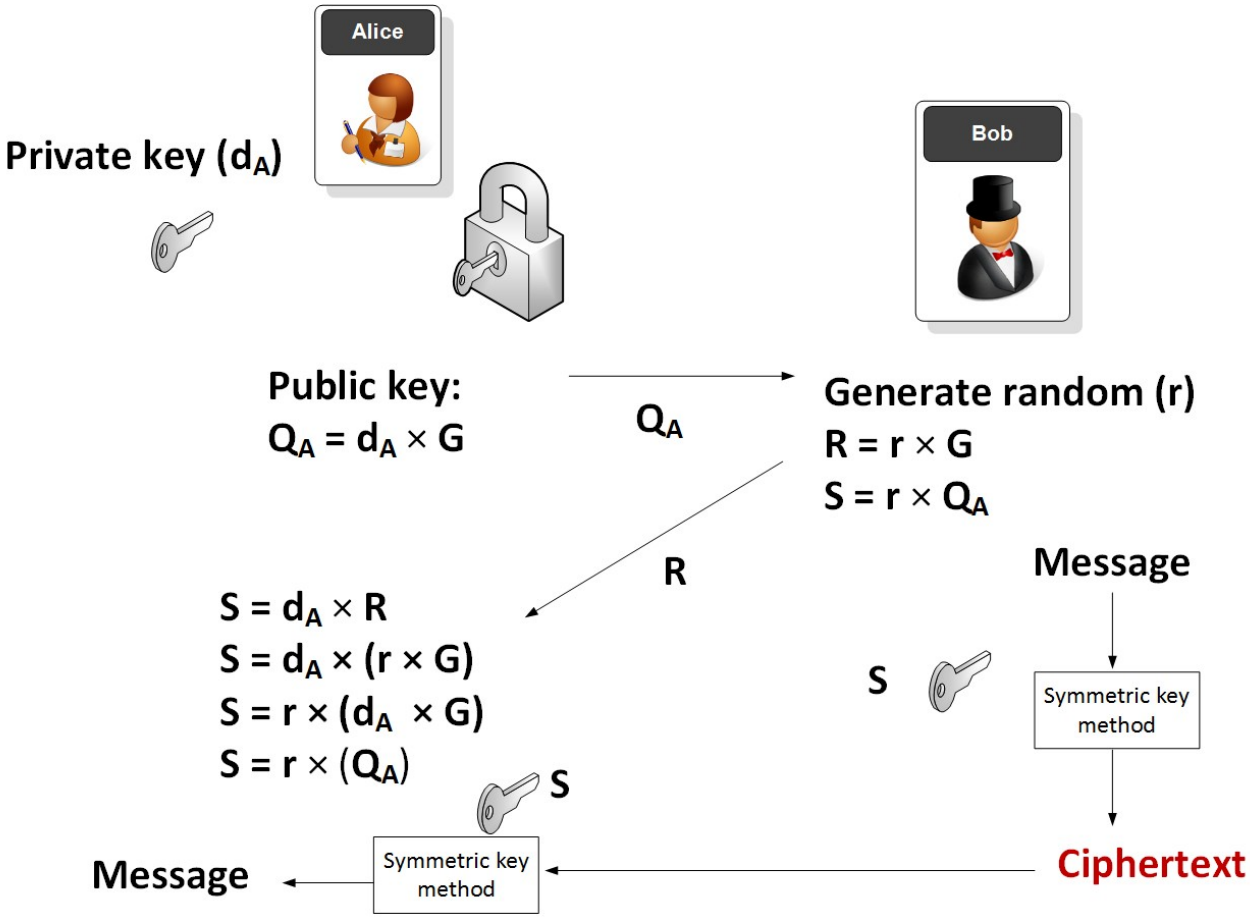


Рис 6. Функціонування шифрування за схемою ECIES

2.4 Висновки до розділу 2

Криптографічні засоби захисту інформації є найбільш продуктивними засобами захисту та грають ключову роль в схемах захисту інформації сьогодення. Криптографічні засоби захисту вирішують численну множину задач, які не можуть бути віришені іншими методами. Кожна захищена мережева транзакція інкапсулює якусь криптографічну методику захисту, тому розвиток криптографічних методів та дослідження їх криптостійкості є важливими задачами сьогодення, що гарантують уможливлення побудови

систем з прийнятним степенем захищеності зважаючи на можливості сучасних комп“ютерів та методики криптоаналізу.

Криптографія сьогодення являє собою набір суміжних складових, що покликані вирішувати окрему підзадачу. Раціональність використання тих чи інших методів насамперед залежить від дослідження криптостійкості компонентів та від контенту поставленої задачі. Симетричні шифри є одними з найкриптостійкішими компонентами криптографії, які до того ж є найбільш дослідженими, маючи свою нішу задач. Найвживанішим блоковим шифром є AES, що побудований на SP-мережі, і є стандартом NIST. Для потокових шифрів перспективним є схеми з декількома РЗЛЗЗ, що забезпечують нелінійність перетворення. Асиметричні шифри мають свої преференції, однак базуються на математичних одностороніх функціях, без математичних гарантій. Алгоритм ECDSA в останні роки стає більш широковживаним, що дещо знижує монополію алгоритму RSA, оскільки забезпечує суміжний рівень криптостійкості за використання менших за об“ємом ключових даних.

Просування фізиків в побудові квантових комп“ютерів та закон Шора, посприяли осучасненю асиметричних шифрів, про що свідчить конкурс NIST на прийняття стандартів щодо постквантових алгоритмів, що почався в 2016 році. Основними напрямами серед конкурсантів, що пройшли в другий раунд, є проблематика решіток, кодові криптосистеми(в тому числі криптосистема Мак-Еліса, що була створена в 1978 році), криптосистеми на основі хеш-функцій, багатомірна криптографія, криптосистеми на суперсингулярних ізогених графах(ізогенії) еліптичної криптографії. Пропорційно конкурсу openSSL, як основний програмний криптографічний фреймворк, створив гілку в своєму репозитарії, для імплементації постквантових алгоритмів, яка згодом буде змерджена в основну гілку. Також йдуть експеременти з TLSv3 для впровадження на стадії хендшейку постквантового алгоритму.

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ КРИПТОГРАФІЧНОГО МОДУЛЯ ЗАХИСТУ ПОВІДОМЛЕНЬ У СЕРЕДОВИЩІ ОС ANDROID

3.1 Опис середовища реалізації

ПЗ було реалізовано на МП Java, в стандартному середовищі програмування для ОС Android Android Studio 3.5, що впроваджує наступний функціонал: навігацію по коду; інтеграцію з системами контроля доступу(CVS); редагування збірочних файлів; SDK-менеджер; проектувальник View рівня ПЗ; відладчик функцій ПЗ(з точками останову); профіліровщик; відладочна консоль додатку; приєднання до лог-файлів на пристрої; відладчик по пам“яті; менеджер віртуальних пристроїв; навігатор файлів пристрою; формувальник та завантажник APK-файлів. В якості збиральника використовується стандартна система збирання gradle версії 3.4.1, що дозволяє додавати залежності через редагування. Для криптографічних функцій ПЗ використовує Java-бібліотеку Bouncy Castle, яка інкапсулює в собі як стандартні високорівневі криптографічні API та низькорівневі пропріетарні API для ефективнішого доступу до функціоналу, та випускається під ліцензією MIT. Для Android Bouncy Castle має форк що називається com.madgag.spongycastle версії 1.54.0. Для досупу до хмарної БД Firebase використовується пакет com.google.firebase(16.0.7), що підтримується корпорацією Google, та впроваджує повний функціонал необхідний для коммунікації з хмарною БД: отримання персонального токену пристрою; можливість відправки повідомлень в NoSQL БД в JSON форматі, схема якого потребує розробки; подійне прийняття повідомлень через наслідування методу зі спеціального інтерфейса обробника подій. Для зменшення об“єму коду щодо рутинних задач View-рівня (пошук графічних компонентів по идентифікатору, наслідування від базових компонентів, тощо) , які потребує ОС Android використовується процесор аннотацій, що доступний в пакеті com.androidannotations версії 4.6. Для операцій щодо парсингу об“єктів в форматі JSON використовується бібліотека корпорації Google Gson(com.google.code.gson) версії 2.8.2. Для мережевих

коммунікацій(формування POST-запитів, встановлення HTTP-хедерів та суміжних задач) використовується бібліотека okHttp(com.squareup.okhttp3) версії 3.10.0. Для відображення статистикі швидкості операцій щодо алгоритмів шифрування та підпису використовується компонент graphview(пакет com.jjoe64.graphview) версії 4.2.2.

3.2 Структура та опис програмної реалізації

Головний інтерфейс додатку має в своєму складі дві вкладки та п“ять пунктів меню. На першій вкладці знаходяться співрозмовникі користувача, з якими він мав діалог або просто експортував до себе їхні X509-сертифікати(з ключами RSA або EC) . На іншій вкладці самі X509-сертифікати користувача та інших його співрозмовників. Сертифікатів та співрозмовників може біти довільна кількість. За подією свайп на кожній з вкладок значення у вкладках буде відображений поточний стан БД, а саме таблиці, що зберігають значення моделей користувачів та ключів.

При першому заході або при відстутності імені і пароля користувача додаток формує алерт-діалог з двома полями — ім’я користувача та пароль, що необхідний для симетричного шифрування полів БД з приватними ключами користувача. До пароля користувача додається випадкова «сіль», формується результат хешування, частина якого(перші 16 байт) стає AES-ключем, що використовується для шифрування полів БД з приватними ключами користувача, та зберігається в приватному просторі додатку. Це запобіжні заходи, оскільки телефон може мати рут-доступ і можливо буде склонити БД, клонування ж приватного простору є більш складнішою задачею, в нормальному режимі до БД додатку(яка реалізована через контент провайдер що не експортиться) окремого затосунку немає доступу для інших застосунків, якщо в конфігурації(маніфесті) не вказано інакше. Досуп до сертиікатів з“являється для інших застосунків лише в окремій частині файлової системи приватного простору додатка.

По натисканню на лівий кут статус-бару с зображенням трьох точок з’являється меню додатка. Поточна версія додатка має п“ять пунктів меню - «New Cert», «Import User», «Statistics», «TSP» та «RSA».

При натисканні на «New Cert» генерується асиметрична пара RSA-ключів або EC-ключів, яка належить поточному користувачу, публічний ключ з цієї пари запаковується в X509-контейнер, що містить: дві дати строку дії(один рік з дати генерації), поле суб’єкт сертифіката(поле «Common Name» містить UTF-8 строку з ім“ям користувача, яке він вказав в початковому алерт діалозі та «SerialNumber» який містить номер, що інкрементується при генерації кожної нової пари ключів поточного користувача), поле «Signature Algorithm»(має ASN-значення «SHA256WithRSA» або «SHA256WithECDSA», що говорить про тип публічного ключа та алгоритм хешування), поле великого цілого «SerialNumber» (встановлюється з поточного часу оскільки централізована видача сертифікатів не передбачена дизайном програми), опціональне поле «SubjectAltName»(в яке вкладається під ASN-тегом «universalPrincipalName» UTF-8 строка, що є поточним ідентифікаційним токеном окремого пристрою, що потрібний для комунікації з хмарною БД Firebase). Після генерації сертифікат та приватний ключ(зашифрований AES-ключем) зберігаються в SQLite БД поточного пристрою через контент-провайдер в табліці для ключів.

Пункт меню «Import User» пропонує користувачу вибрати файл сертифікату іншого користувача з яким він хоче захищено обмінюватися текстовими повідомленнями. Після успішного обрання необхідного файлу сертифікат-кандидат валідується додатком та зберігається в таблицю ключів БД, також в таблицю користувачів БД додається користувач та його ідентифікаційний Firebase-токен. В подальшому ці дві таблиці використовуються для формування ари користувач та відповідне йому значення сертифікату.

При натисканні на пункт меню «Statistics» користувач опиняється в новому екрані(новій активності), яка відображає статистичні дані шодо

швидкості роботи алгоритмів (RSA та ECDSA). Дані збираються при виконанні додатком відповідних операцій застосовуючи один з алгоритмів, та збираються в окрему таблицю в БД додатку. Активність зі статистикою відображає збережені дані застосовуючи графічний канвас формуючи шкалу де абсцисса є експериментом(інкрементується з кожним застосуванням, а ордината є фактичним часом, що був втрачений додатком при виконанні відповідної операції(з відповідними помітками на графіку(Legend)). Підрахунок ведеться по двом алгоритмам таким чином формуючи зручну наявну графічну різницю між ними. Екран зі статистикою має п“ять пунктів меню - «Generate Key», «Encryption», «Signing», «Decryption» та «Verification». При натисканні на окремий пункт меню з“являється статистика стосовно відповідної операції.

Включення пункту меню «TSP» активує в застосуванні використання стороннього сервера Timestamp(одного з п“яти що статично защиті в коді та обираються випадково, які безкоштовно надають цю послугу) для встановлення мітки часу на ЕЦП, щоб третя сторона підтверджувала в окремий проміжок часу існування саме такого стану контенту(результату його хешу), а отримувач зміг це перевірити побачивши цей атрибут в ЕЦП, що дистибутиться в PKCS7-контейнері формату SignedData.

При натисканні на елемент з вкладки з сертифікатами, додаток формує діалог з, зручним локалізованим під конкрентий користувацький профіль, вибором засобу для експорту сертифіката іншій стороні. Список програм через які можна експортувати сертифікат залежить від вподобань користувача пристрою(електронна пошта, bluetooth, Wifi-direct тощо) та встановлених затосувань, що обробляють інтенти щодо поширення інформації. При цьому зберігаючи X509-сертифікат в файл приватного простору додатку(в каталог шляху, що контролюється файл-провайдером) та поширюючи шлях файлу через файл-провайдер, що задекларований в налаштуваннях (маніфесті) застосування.

Задля успішного захищеного спілкування криптографічна система додатку повинна мати публічний сертифікат віддаленої сторони, та свою пару(сертифікат та приватний ключ). Оскільки додатком передбачено для поточного користувача довільна кількість асиметричних пар, тому при натисканні перший раз на елемент з вкладки з користувачами додаток формує алерт-діалог зі списком сертифікатів, що належать поточному користувачу. При натисканні на елемент цього списку вибраний сертифікат та відповідний йому приватний ключ стають параметрами для шифрування повідомлень до віддаленої сторони. При натисканні наступні рази відкривається активність з повідомленнями, що належить співрозмовнику і поточному користувачу. Знаходячись на екрані з користувачами та сертифікатами користувач може отримати повідомлення від невідомого користувача, внаслідок перевірки правильності сигнатури з сертифікату, що вкладений в PKCS7-контейнер, для користувача формується алерт-діалог з ім“ям непідтвердженого користувача, від якого прийшло повідомлення, внаслідок вибору користувача та натискання на кнопку «ОК», новий користувач з“являється у вкладці з користувачами. Таким чином додаток може зкомунікувати користувачів при наявності сертифікату з піблічним ключем, що є в наявності тільки у одного користувача в додатку, не потребуючи обопільного обміну сертифікатами.

Активність з чатом має чітку ідентифікацію повідомлень від поточного користувача та повідомлень до нього з віддаленої сторони. Кожне повідомлення має поле з датою, відсутність якої сигналізує про неуспішну доставку повідомлення(наприклад через проблему з мережевим зв’язком) до хмарної бази даних Firebase, тобто встановлення дати доставки до хмарної БД Firebase не свідчить про успішну доставку до застосування іншого користувача. Кожне окреме повідомлення має поле з власним контентом. Активність в статус-барі встановлює ім’я користувача з віддаленої сторони, що отримана з його сертифікату. Активність має поле для вводу контенту повідомлення, при натисканні на яке з’являється клавіатура, для його набирання. При появі

клавіатури список з повідомленнями переміщується на останне актуальне з списку. Активність також має кнопку для відправлення повідомлення, яка запускає процедуру шифрування, підписування, зберігання в БД з повідомленнями поточного пристрою та відправки відповідного повідомлення в хмарну базу даних Firebase. Задля забезпечення більш високої швидкості роботи додаток зберігає в незашифрованому вигляді повідомлення, які він отримав та для яких пройшла успішно процедура верифікації та дешифрування, в локальну БД пристрою.

Інша сторона знаючи Firebase-токен віддаленої сторони(яка отримана з X509 контейнеру) та свій власний може отримати призначене їй повідомлення з хмарної БД, що зберігає свої дані в NoSQL-базі в форматі JSON. Гілка для прослуховування додатком формується внаслідок хешування за допомогою SHA256 його токену. Таким чином щоб іншому додатку написати повідомлення достатньо знати токен віддаленої сторони, який вона отримує від X509-сертифікату з заповненим полем «SubjectAltName», провівши хешування додаток формує шлях в хмарній БД, що відповідає цьому хешу та власному хешу і записує защифрований та підписаний контент повідомлення в гілку отримувача. Після видалення з хмарної БД відповідних повідомленнь, перевірки його підпису публічним ключем віддаленої сторони, та вдалої спроби дешифрування своїм приватним ключем контент повідомлення зберігається в БД, а саме в таблиці з повідомленнями, а активність з повідомленнями відображає їх користувачу. Таким чином відбувається захищений обмін повідомленнями між двома пристроями в реалізації з поточного додатку.

Додаток розробляється використовуючи мінімальний рівень Android API 19, а орієнується на 28, тобто додаток може використовуватися на платформах Android починаючи з 4.4(KitKat) до 9.0(Pie). Розробка створюється під роздільну здатність екрану 1920x1080 пікселів, можливістю адаптації дизайну на інші роздільні здатності, однак зважаючи широкий спектр технічних характеристик екранів пристроїв та ОС Android, графічна складова додатку

може різнитися. Системний інтерфейс має бути на англійській мові. Необхідне використання узагальненої базової теми для UI -інтерфейса на всіх екранах. Для обміну X509-сертифікатами достатньо експорту сертифікату до однієї з сторін, інша сторона може отримати X509-сертифікат неопосередковано з PKCS7-контейнеру формату SignedData. В додатку має бути передбачена тількі портретна орієнтація активностей, оскільки в альбомній орієнтації немає необхідності. В додатку має бути передбачений контент провайдер однак без експорту до цільової системи. В додатку має бути передбачений файл провайдер оскільки функціонал передбачає експорт X509-сертифікатів з бази даних додатку до зовнішнього додатку, що призначений для обміну. Приватні ключи для сертифікатів користувача зберігаються в БД обов’язково в зашифрованому вигляді. Пароль користувача повинен обов’язково хешуватися з додаванням «солі», а результат - зберігатися в приватному середовищі додатка. Користувач повинен мати можливість генерувати будь-яку кількість пар ключів, публічні ключі мають бути обернуті в X509-сертифікати. Повідомлення користувачів в хмарній базі даних Firebase повинні зберігатися тільки в зашифрованому вигляді. Кожне повідомлення повинно бути зашифрованим та підписаним для можливості перевірки сторони, що передає тому вкладання сертифікату X509 є необхідністю, а перевірка ЕЦП є обов“язковою. Опціонально в PKCS7-контейнер формату SignedData вбудовується мітка часу, що отримана від TSP-сервера та відповідний йому ланцюг сертифікатів. Якщо додаток бачить мітку TSP в контейнері, то перевірка мітки часу з врахуванням ланцюгу серифікатів TSP-сервера обов“язкова. Для асиметричної складової шифрування повинні використовуватися RSA ключі довжини не нижче 2048 біта, для EC — окремий, статично задекларований тип еліптичної кривої(secp256k1), що має розмір 256 біт, що приблизно відповідають криптостійкості AES-128. Термін дії сгенерованих X509-сертифікатів повинен складати не менше одного року. Оскільки ПЗ імплементує дві різні схеми шифрування необхідно впроваджувати для користувача можливість

візуалізованого порівняння швидкості роботи тієї чи іншої схеми під час основних операцій(шифрування, дешифрування, формування ЕЦП, перевірка ЕЦП, генерація ключів). Додатково ПЗ повинно обробляти подію лівого свайпу на елементах, що відображають активних співрозмовників та активних сертифікатів, за конвенцією UI-рівня ОС Android це означає виделення елементу. Після лівого свайпу на елементі сертифікатів, ПЗ повинно видавати користувачу підтвердження, якщо сертифікат задіяний після підтвердження видаляється сертифікат та відповідний йому користувач і повідомлення, що належать йому. Після лівого свайпу на елементі, що представляє співрозмовника, ПЗ повинно видавати користувачу підтвердження, після якого каскадом видаляються ті ж самі елементи, що і на елементі сертифікатів.

3.3 Проектування та реалізація програмного забезпечення

Повний цикл розробки ПЗ включаючи процеси проектування, реалізації, та тестування, включає наступні етапи:

- розробка концепції програми, інформаційне проектування, оформлення технічного завдання;
- розробка ескізів базового дизайну додатку;
- проектування структури бази даних SQLite на кліентській стороні;
- проектування NoSQL бази на стороні хмарної БД Firebase;
- побудова UI дизайну (xml файли для рівня відображення ОС Android);
- програмування та підключення рівня логіки до рівня відображення та зв’язок його з моделями з БД;
- підключення контент провайдеру до додатку;
- проектування та реалізація криптографічного функціоналу;
- підключення Firebase до додатку та встановлення зв’язку з хмарною базою даних;
- реалізація функціоналу додатку в alpha версії;
- підключення іншого пристрою та перевірка зв’язку;
- базове тестування функціоналу;

- поверхневе негативне тестування;
- доробки та глобальний рефакторинг коду.

Як правило проектування додатків починається з структури БД, що зберігає моделі, які потрібні для імплементації логікі ПЗ. Основними сутностями якими оперує додаток є ключі, користувачі та повідомлення, а додатковими є статистика щодо швидкості роботи алгоритмів. В термінах реляційної моделі це чотири окремі таблиці, кожна з яких повинна бути проіндексована обмеженням первинного ключа.

Таблиця ключів повинна зберігати два різних класи моделей: ассиметричні пари, що належать поточному користувачу додатку(з зашифрованою симетричним алгоритмом приватною частиною) та сертифікати інших користувачів, які він експортував для обміну повідомленнями. Основними даними для цієї таблиці є: приватні(зашифрований симетрично) та публічні ключі, відповідний X509-сертифікат, короткий опис для користувача та тип та дата генерації або експорту відповідного ключа. Ті елементи, що мають приватний ключ не рівним NULL, належать поточному користувачу додатка.

Таблиця з користувачами, яких імпортує користувач повинна мати два додаткових обмеження на первинні ключі в таблиці ключів: зовнішній ключ, що вказує на елемент, який зберігає сертифікат яким володіє віддалений користувач(для побудови зв’язку користувач-сертифікат), що використовується для перевірки підпису та шифрування повідомлення до віддаленої сторони, та зовнішній ключ, що вказує на елемент, що зберігає приватний ключ(для побудови зв’язку користувач-приватний ключ), який використовується для розшифровки повідомлень від відповідного користувача. Крім вище зазначених полів таблиця зберігає: ідентифікатор (ім’я) користувача, що отримано з поля «Common Name» сертифіката, токен для хмарної БД, що також отриманий з сертифікату — з поля «SubjectAltName», дата імпорту відповідного користувача користувачем додатку.

Таблиця з повідомленнями зберігає інформацію про окреме повідомлення. Вона повинна мати обмеження зовнішнього ключа, що вказує на відповідного даному повідомленню користувача з таблиці користувачів. Також вона повинна мати поля для ідентифікації в якій ролі воно виступає для поточного користувача(приймач або відправник). Крім вище зазначеної інформації таблиця з повідомленнями зберігає: дату отримки або відправлення, оригінальні отримані (зашифровані і підписані) дані, розшифрований контент повідомлення, статус повідомлення(успішна відправка або прийом).

Таблиця зі статистикою зберігає дані, які отримуються внаслідок виконання окремим алгоритмом(RSA, ECDSA, ECIES) своєї послуги. Множина послуг обмежується наступними операціями: генерація ключа, шифрування, дешифрування, генерація ЕЦП та перевірка ЕЦП. Кожна окрема операція фіксується в БД з наступними полями: унікальний первинний ключ, що ідентифікує окремий запис в таблиці зі статистиками; алгоритм(RSA, ECDSA), який провів операцію; операція з множини можливих(цілочислене число); час виконання операції; довжина даних з якими маніпулює операція(про генерації ключів не використовується); фактивний час в який відбулася операція.

Хмарна БД Firebase зберігає свої дані в форматі JSON, відноситься до NoSQL БД, що своєю структурою значно відрізняється від традиційної реляційної моделі БД. Ця обставина впливає на процес проектування такої бази. Кожне повідомлення має два поля: ідентифікатор повідомлення в рамках однієї транзакції передачи та контент повідомлення в форматі ASN1, що енкодований стандартним base64 енкодером(це допомагає мати платформонезалежний порядок байт(без впливу big- та little-ending)).

Додатково до моделей які зберігаються в реляційній БД, частина з яких(повідомлення) береться з хмарної БД, додаток потребує також зберігання наступної інформації: ім’я користувача додатку; кількість його згенерованих асиметричних пар ключів; токен хмарної БД, що належить поточному

пристрою; випадкова сіль, що додається до паролю щоб згенерувати хеш а з нього AES-ключ; AES-ключ, що використовується для шифрування полів з приватними ключами користувача. Ця інформація може зберігатися відокремлено від інших даних, оскільки має більший рівень конфеденційності для додатку, а ніж інші моделі. Її відгалудження від інших моделей може принести дивіденди, коли стане потреба імплементації більш високого рівня захищеності(наприклад впровадження захищеного носія).

Найбільш доцільними до потреб даної задачи(шифрування повідомлення та підписування результату цього шифрування) є об’єкти стандарту PKCS7 та більш його нової реалізації CMS, що описана в RFC5652. Цей стандарт декларує великий об’єм функціоналу, який може бути використаним(серед якого digest-data, enrypted-data, authenticated-data). Крім цього декларується можливість мульти-підпису повідомлення(тобто повідомлення, що підписане декількома об’єктами) та мульти-шифрування, що можуть знадобитися в наступних версіях додатку для реалізації чату з декількома користувачами. Для поточних потреб використовується два CMS типи контенту: enveloped-data та signed-data. Перший тип представляє упакований та зашифрований контент, інший — підписаний контент.

Зовнішньою структурою ASN1 є signed-data, яка складається з: версії CMS, ідентифікатора дайджеста алгоритму, внутрішнього контенту(в нашому випадку enveloped-data), опціонального списку сертифікатів(що є підписантами) та CRL, та signedInfo(інформації, що стосується цифрового підпису(тип, алгоритм підпису хеш внутрішнього контенту, час підписання тощо). В кожну signed-data додаток вкладає свій сертифікат, що використовується для діалогу з іншою стороною. Таким чином будь-яка зацікавлена сторона може перевірити цифровий підпис, використовуючи цей прикріплений сертифікат.

Внутрішнім контентом signed-data для алгоритму RSA є enveloped-data, що в свою чергу є набором з зашифрованого контенту та інформації про ключ,

яким цей контент зашифрований. Вона складається з: версії CMS, опціонального originatorInfo(набору сертифікатів та CRL), recipientsInfo та encryptedContentInfo(тип контенту, ідентифікатор алгоритму шифрування та сам контент). ReceipientsInfo пов’язує публічний ключ, яким шифрується випадковий симетричний ключ, інформація про який включається в encryptedContnentInfo, і який використовується для розшифровки контенту.

Схема ECIES стандартом контейнеру Enveloped Data не передбачена, тому внутрішній контент для ECIES передставляє собою тип Octet String формату ASN1, що складається з триплету(R|c|d): R інтегрує в собі секретне випадкове число, з якого утворюється загальний секрет, та яке може стати відоме тільки власнику приватного ключа(віддалений співрозмовник); c — зашифроване повідомлення загальним секретом(точка на еліптичній кривій, що може бути отримана з R і передставляє собою кординату x); d — слугує для перевірки цілісності отриманого криптоповідомлення(є контрольними бітами, що отримуються внаслідок використання MAC). Контейнер CMS Signed Data є обопільним для двох схем, оскільки стандартом передбачено формування підпису як з алгоритмом RSA так і ECDSA.

Таким чином формування контейнеру з повідомленням використовує таку процедуру: повідомлення шифрується випадковим симетричним ключем (в даному випадку - AES, 128 біт в режимі CBC), який в свою чергу шифрується публічним ключем віддаленої сторони або є кординатою x загальний секрету; формується enveloped-data з інформацією з попереднього шагу або Octet String триплет(R|c|d); enveloped-data(або триплет(R|c|d)) вкладається в signed-data та підписується приватним ключем відправника; вкладається сертифікат відправника. Як результат, випадковий симетричний ключ знає віддалена сторона, а перевірити підпис може будь-хто оскільки CMS має вкладений сертифікат. Опціонально після активації флагу TSP користувачем з головного меню додатку, додатково формується запит на TSP-сервер(один з п“яти, що хардово прописані в коді, які безкоштовно надають цю послугу) з SHA256-

дайджестом на сигнатуру повідомлення. Внаслідок мережевої операції з POST-запитом протоколу HTTP зі спеціальним хедером (Content type: application/timestamp-query) з“являється об“єкт timeStampToken який вставляється в CMS контейнер формату SignedData з всіми необхідними сертифікатами, що перевірити додатковий атрибут. ASN1 структури повідомлення та сертифікатів повідомлення наведені в додатках А, Б. В, Г, а в додатку Д — з додатковим timestamptoken, що отриманий з TSP-сервера.

View-рівень додатку повинен забезпечити зручну візуалізацію вище зазначених моделей для користувача. Моделі користувачів та сертифікатів, є схожими та ієрархічно лежать на одному рівні, тому, з точки зору агрономіки, можуть бути реалізованими через один екран додатку. Модель повідомлень відноситься до конкретного віддаленого користувача тому для зручності має бути окремим екраном. Модель статистикі також вимагає окремого екрану з меню яке буде слугувати перемикачем активного елементу з множини операцій формуючи необхідний для відображення компонент статистикі. Моделі, що лежать в БД, з точки зору логіки можуть мати довільну кількість тому раціональним буде використовувати адаптери, що беруть свої дані з БД додатку та дозволяють візуалізувати довільну кількість компонентів. Для узальнення коду необхідний узагальнюючий(узагальненням буде динамічний View-компонент) абстрактний клас, який вбере в себе всю низькорівневу логіку по роботі з реляційною БД. Таким чином додаток має три основних екрани — екран з користувачами та сертифікатами, екран з повідомленнями окремого користувача користувачу додатку та екран з статистикою. На першому з них зображені динамічні за кількістю компоненти сертифікатів та користувачів(з іхніми графічними реалізаціями). На другому відображається повідомлення двох типів(відправлені від поточного користувача та отримані від віддаленої сторони), кнопка для відправки та поле в яке користувач може записувати контент повідомлення. На третьому, відображається графічне відображення статистикі відносно двох схем шифрування, також йому належить меню з

п“яти компонентів, що перемикає активний елемент формуючи релевантний запит до таблиці статистикі з БД локального пристрою.

Додаток використовує алерт-діалоги для запиту: імені користувача та пароля, вибору окремого сертифікату для комунікації з віддаленою стороною, запиту на додавання віддаленої сторони, що надіслала повідомлення та запиту на видалення користувача або сертифікату. Першим з них є алерт-діалог з полем імені та пароля користувача, дані з нього потрібні для формування AES-ключа та X509-сертифікатів, що належать користувачу. Другий відповідно використовується коли потрібно вказати якою парою асиметричних ключів слід користуватися при комунікації з окремою віддаленою стороною. Третій використовується коли прймаюча сторона не має інформації про віддалену сторону, яка написала їй повідомлення, тому потрібна згода користувача щодо додавання цієї віддаленої сторони до списку користувачів. Четвертий з“являється коли користувач генерує подію лівого свайпу на елементі сертифікату, або елементі, що представляє співрозмовника, що означає операцію рекурсивного(каскадного) видалення моделей, що пов“язані з цим користувачем — сертифікат в таблиці ключів, модель в таблиці користувачів та всі повідомлення в ких він був задіяний, з таблиці повідомлень.

Додаток також використовує п“ять пунктів меню на основному екрані: для процедури генерації нової пари асиметричних ключів; приєднання користувача по його сертифікату, що знаходиться в ФС пристрою; виклику екрану зі статистикою; використання альтернативної схеми RSA, оскільки основною в додатку вважається ECIES; встановлення або зняття флагу TSP, для активації або деактивації користування зовнішнім TSP-сервером відповідно.

Таким чином візуальна складова додатку складається з: екрану сертифікатів та користувачів, екрану повідомлень до конкретного користувача, екрану статистикі, чотирьох вищезазначених алерт-діалогів та п“яти пунктів меню, що належать екрану сертифікатів та користувачів та п“яти пунктів меню, що належать екрану зі статистикою.

Додаток виконаний за допомогою шаблону проектування MVP(Model-View-Presenter). Моделями є сутності в БД присторою, View-рівнем є графічна визуалізація, Presenter є шар логіки, що відноситься до конкретного компоненту. View-рівень повинен чітко розмежовуватися від рівня логіки. Для цього рівню логіки потрібно мати інтерфейс через який він спілкується з view-рівнем, задля виконання view-рівнем його потреб. В свою чергу view-рівню потрібно мати інтерфейс через який він реагуючи на події від користувача надсилає їх рівню логіки. Там чином за допомогою рівня логіки можна закодувати поведінку додатку, яка не буде кардинально пов’язана з рівнем відображення. Нативними потоками ОС Android є об“єкти Java Thread, які в свою чергу на Unix-системах транслюються в Posix-потоки. Типовий додаток ОС Android має головний потік, що є відповідає за UI-інтерфейс, тому його блокування, тобто виконання в ньому довгих операцій(таких як мережеві операції, шифрування, підпис тощо), може унеможливлювати взаємодію користувача з додатком, тому вкрай важливим є виконання таких операцій в інших потоках. Обмеженнями ОС Android є неможливість прямого доступу до UI-компонентів(об“єктів UI-потоку) з інших потоків додатку, тому необхідно впроваджувати подійну модель, реєструючи UI-поток для прийому подій, та обробляти їх в ньому змінюючи UI-інтерфейс. Кожний потік має пріоритет(nice-number), який буде впливати на пріоритет вибору окремого потоку з множини існуючих в системі з боку планувальника, який приймає рішення щодо надання ресурсів окремому процесу або потоку. Стандартний планувальник Linux зберігає процеси, що є кандидатами на виконання, в червоно-чорному дереві, узлами якого є посилання на процеси з додатковими параметрами(такими як максимальний час виконання). Дерево сортується відносно поточного часу, що був витрачений процессором на виконання процесу або потоку(що є в складі процесса однак планується окремо), в обробнику переривання, яке генерує системний таймер. Таким чином планувальник намагається справедливо розподіляти ресурси між всіма процессами(потоками) системи. ОС Android для нівелювання такої

поведінки планувальника використовує декілька налаштованих Linux cgroups, які є агрегаторами деякого набору обмежень. Таким чином додаток на передньому плані(його UI-потік) завжди попадає в пріоритетну групу, для якої налаштовані мінімальні обмеження, в той час інші процеси знаходяться в менш пріоритетних групах, тим самим отримуючи набагато менше ресурсів. Для побудови додаткових потоків ОС Android в складі свого фреймворка має чисельну кількість засобів: Thread, AsyncTask, IntentService, HandleThread тощо. Thread є простим контексном для виконання інтерфейсу Runnable. AsyncTask є шаблонізованим класом для одноразового виконання фонової задачи з вбудованим методом, через який, після закінчення задачі, можна оновлювати графічні компоненти в складі UI. IntentService уможливлює виконання задачи в контексті іншого потоку по замовленню, зформувавши Intent з ідентифікатором цього сервісу, ОС Android автоматично виділить новий потік та виконає завдання в цьому потоці. HandleThread представляє фоновий потік, який запускаючись прослуховує низькорівневий об“єкт Looper, що відповідає за передачу та отримання низькорівневих повідомлень class Message, що можуть надходити з різних компонентів додатку. Для організації додаткових потоків, використовується HandleThread, який виконується наряду з основним екраном(активністю), що потребують наступні ресурсозатратні задачі: мережева комунікація з TSP-сервером, шифрування, дешифрування, формування ЕЦП, верифікації ЕЦП та генерації ключів. Хмарна БД Firebase використовує свої додаткові потоки, повністю інкапсулюючи всю низькорівневу логіку у складі свого фреймворку.

Мережева комунікація додатку складається з подієорієнтованого підходу, що використовується в хмарінй БД Firebase. Додаток підписується на подію якихось змін в хмарній БД відносно окремого шляху в ній. Цей шлях складається з хешу поточного токену пристрою, в шістнадцятирічній системі. Тобто у кожного додатку з окремого пристрою свій шлях в цій хмарній БД. Коли інший пристрій хоче написати повідомлення він, знаючи значення віддаленого

токена, додає до шляху отримувача свій хеш токену та свої повідомлення. До приймаючої сторони, якщо вона виконується, тим часом приходить подія, що в її шлях хтось написав. Після валідації додаток забирає свої дані з хмарної БД, видаляючи їх, та розкладає інформації в свою локальну БД. Якщо додаток бачить токен, інформація про який відсутня в нього він видає алерт-діалог для користувача щодо підтверження невідомого користувача. Якщо користувач згоджується в БД додається новий користувач та його сертифікат, що береться з CMS та записується повідомлення від цього користувача. Оскільки додаток має два основних екрани, тобто може перебувати в двох незалежних станах, кожен рівень логіки цих станів повинен мати змогу обробити подію щодо нових повідомлень. Тобто повинна використовуватися IPC, що реалізовує зв’язок один до багатьох. Крім того рівень логіки на екрані повідомлень повинен обробляти подію успішної доставки(запису в хмарну БД) повідомлення, яке написав користувач віддаленій стороні.

Процедура генерації асиметричної пари може бути викликана користувачем з відповідного пункту меню. Вона генерує нову пару асиметричних ключів, пакує публічний ключ в X509-контейнер, додає до нього свій токен хмарної БД, своє ім’я та інше, шифрує локальним AES-ключем новий приватний ключ, та додає іх в локальну БД, після цього користувач може експортувати цей сертифікат для віддаленої сторони.

Процедура імпорту користувача приймаючи ASN1 структуру X509 сертифікату валідує його структуру, знаходить віддалений токен, ім’я користувача та інші параметри, та додає нового користувача та його сертифікат в локальну БД пристрою.

Додаток реалізований на мові програмування Java, основній мові для програмування на платформі Android, яка використовує об’єктно-оріентований підхід. Нюансом, що притаманний платформі Android, є подієоріентована парадигма програмування та проектування. Шаблоном проектування додатку є

MVP, за яким код повинен бути розділеним на моделі, візуальні компоненти та рівень логіки.

Клас, що описує додаток називається CryptoChatApp, який крім опису додатку реалізує інтерфейс ValueEventListener, що дає змогу відслідковувати стан окремої змінної в хмарній БД Firebase, додаючи можливість реагувати на події в БД. Таким чином логіка щодо прийому та відправки повідомлень зоріентована в одному місці. Також в цьому класі приватною властивістю описаний екземпляр класу PrefsProvider. Цей клас реалізує функціонал по роботі з постійною пам’яттю, що відкривається в приватному режимі для додатку. В цій пам’яті зберігається поточне ім’я користувача, кількість асиметричних ключових пар, токен для хмарної БД, AES ключ для розшифровки приватних ключів в БД, випадкову сіль, що додається до паролю користувача щоб згенерувати AES ключ. Клас додатку використовує локальний LocalBroadcastManager для організації зв’язку один до багатьох, в рамках домену додатку, та розсилає два типи повідомлень: про успішну доставку та про успішний прийом повідомлень. Інша сторона(одна з активностей додатку) знаходиться в контакті з користувачем та реагує на ці події асинхронно підписавшись на них та реалізувавши інтерфейс BroadcastReceiver.

Окремий екран, що має незалежний життєвий цикл, в Android описується класом Activity та похідними від нього. Фрагментована візуальна сутність, що може бути використана повторно та використовується для відображення наборів динамічних компонентів, описується класом Fragment та похідними від нього.

Додаток має дві активності: UserChoiceActivity та MainActivity. Перша з них описує екран користувачів та сертифікатів, інша - екран повідомлень. В свою чергу активність UserChoiceActivity має два фрагменти: KeyHolderFragment та UserHolderFragment. Перший відповідає за відображення сертифікатів що належать користувачу та його контактам, інший — за відображення контактів поточного користувача пристрою. До складу цих фрагментів та активності повідомлень входить компонент RecyclerView, що

дозволяє відображати довільну кількість компонентів та відслідковувати подію свайп, відображаючи стандартний графічний компонент.

Для кожного компонента, що наслідує клас GroupView(напиклад RecyclerView) додатково потрібен клас Adapter, який надає порядок, кількість та самі дані. В додатку використовуються три типи адаптерів: адаптери, що беруть дані з БД через абстрактний клас RecyclerAdapterCursorableBase(що агрегує у себе низькорівневу логіку для відображення даних з курсорів) — це CertificateAdapter, UserAdapter, MessengerAdapter; адаптери що динамічно формуються з ArrayList(використовуються при обиранні користувачем сертифікату що буде використовуватися для зв’язку з контактом та при появі повідомлення від невідомого користувача щоб надати користувачу змогу підтвердити його відправника) — це анонімні інстанси що наслідують стандартний клас ArrayAdapter; адаптер, що використовується для групування фрагментів в активності UserChoiceActivity(BasicFragmentAdapter). MessengerAdapter має додатковий функціонал що відображає два типи візуальних компонентів — повідомлення, для яких відправником є поточний користувач, та повідомлення для яких відправником є віддалена сторона діалогу. Більшість адаптерів описана в пакеті Adapters.

Згідно шаблону MVP активності, фрагменти та динамічні компоненти з адаптерів відносяться до рівня відображення, тому для максимального абстрагування цього рівня від рівня логіки, кожен елемент має наслідувати власний інтерфейс, через який з ним буде працювати логічний рівень додатку. В додатку це інтерфейси з пакету MVPView. Наступні інтерфейси використовуються для спілкування рівня логіки з візуальним компонентом. CertificateItem — описує інтерфейс з динамічним компонентом, що відображає сертифікат. MessageItemView — описує інтерфейс з динамічним компонентом, що відображає повідомлення. UserItem - описує інтерфейс з динамічним компонентом, що відображає контакти користувача. MessengerView — описує інтерфейс з активністю MainActivity. UserChoiceMVPView — описує інтерфейс

з активністю UserChoiceActivity. Крім того є базовий інтерфейс для інтерфейсів зі складними компонентами MVPViewBase. Основна візуальна складова додатку проіллюстрована в додатку Г.

Рівень логіки також повинен мати інтерфейси до кожного з візуальних компонентів, щоб мати можливість реагувати на події які можуть відбутися внаслідок дій користувачата нюансів самого компонента. Ці інтерфейси знаходяться в пакеті Communicators. CertificateFrCommunicator — описує інтерфейс з фрагментом, що відображає сертифікати. UserFrCommunicator - описує інтерфейс з фрагментом, що відображає контакти. UserChoiceCommunicator, MessengerCommunicator — описують інтерфейси з активностями UserChoiceActivity та MessengerActivity. Крім того для компонентів з подовженим життевим циклом є базовий інтерфейс CommunicatorBase.

Агреговані компоненти логіки, що мають життевий цікл відповідно базовому візуальному компонету(наприклад Activity) називаються презентерами. Презентери имплементують всі інтерфейси в складі однієї логічної складової додатку роблячи поведінку, відображення та реагування на події від візуальних компонентів одним цілим. Вони знаходяться в пакеті Presenters. MessengerPresenter — це рівень логіки додатку коли він знаходиться на екрані з повідомленнями. UserChoicePresenter — це рівень логіки додатку коли він знаходиться на екрані контактів та сертифікатів. Крім того у них є базовий узагальнений абстрактний клас PresenterBase. Кожен з презентерів підписується на події зміни хмарної БД. А MessengerPresenter додатково на подію щодо успішної доставки повідомлень.

БД в додатку реалізована через контентпровайдер, що є стандартом для системи Android. Серед численних його можливостей найважливішими є можливість синхронізації з віддаленою інфраструктурою та можливість передачи даних які він зберігає іншим додаткам. Крім того запроси до нього мають більш людиночитабельний формат — наприклад доступ до всіх

сертифікатів в додатку виглядає як «content:://com.example.j41ss.cryptochat/keys». Клас контент провайдеру, що називається CCContentProvider, використовує БД SQLite через посередника SqliteBackendProvider, що створює за необхідністю саму БД та оновлює її при переході на нову версію. SQL схема знаходиться в інтерфейсі DBCSheme. Кожна активність наслідує узагальнений інтерфейс LoaderManager.LoaderCallbacks<Cursor>, який дозволяє в іншому потоці завантажувати дані з БД, щоб оновлювати іх відповідно реальним даним в БД не зупиняючи при цьому головний потік додатку. Додаток використовує допоміжний клас, що складається зі статичних методів для операцій з БД що знаходиться в пакеті CryptoGraphy. В ньому зосереджені операції з БД, які потрібні додатку: збереження сертифікатів; знаходження віддаленого користувача за його ідентифікатором; знаходження сертифікату за ідентифікатором(за текстовим ім’ям); знаходження зашифрованих приватних ключів користувача; збереження користувача; збереження шляху сертифікату; знайдення всіх сертифікатів що належать користувачу; зміна сертифікату, що використовується для окремого користувача; перевірка чи має контакт сертифікат поточного користувача; знайдення максимального значення ідентифікатору для повідомлення; збереження повідомлення; зміну статусу доставки для повідомлення; знайдення всіх токенів, що знаходяться в БД; знайдення користувача по його токену тощо. Ці дані кваліфікуються згідно шаблону MVP як моделі.

Доступ до даних, що знаходяться в приватному середовищі додатку, відбувається через клас посередник PrefsProvider. Він дозволяє зберегти або отримати доступ до наступних даних: ім’я користувача; кількість його асиметричних пар; токен для доступу до хмарної БД; випадкова сіль; AES-ключ, що формується з хешу операції конкатенації паролю користувача та випадкової солі.

Криптографічні функціонал додатку використовує java бібліотеку bouncyCastle, яка впроваджує функціонал аналогічний openSSL — високорівневий API для низькорівневих математичних та криптографічних об’єктів та операцій з ними, і ASN1 структур. Криптографія зосереджена в пакеті Cryptography. Клас CryptoUtils є допоміжним класом що складається зі статичних функцій. Функція generateKeyPair генерує асиметричну пару ключів. Функція generateX509Cert генерує сертифікат використовуючи ключову асиметричну пару, вказаний subjectAltName та subject distinguished name. Функція SHA256 розруховує хеш відповідного типу. Функція getRandomString генерує випадкову послідомність с безпечного рандомізатора. Функція getAesCipher повертає низькорівневий об’єкт відповідного шифру. Функції aesEncrypt та aesDecrypt — виконують відповідно AES шифрацію та дешифрацію контента. Функція readCertificateFromStream повертає X509-сертифікат з stream об’єкта. Функція parseDnSubj повертає відображення subject distinguished name. Функція parseX509Certificate проводить розбір сертифікату та його валідацію. Функція readSubjectAltNameChunk читає частини сирого ASN1, що знаходиться в полі SubjectAltName сертифікату. Функція encryptCMSData виконує шифрування контенту використовуючи об’єкт сертифікату та повертає сирий зашифрований ASN1 об’єкт CMS типу enveloped-data. Функція decryptCMSData виконує розшифровку ASN1 об’єкту CMS типу enveloped-data, використовуючи сертифікат та приватний ключ. Функція signData виконує підпис контенту використовуючи сертифікат та приватний ключ та повертає сирий ASN1 об’єкт CMS типу signed-data з вказаним контентом та сертифікатом всередині. Функція verifySignedObject — предикат, який перевіряє підпис для сирого ASN1 об’єкту типу signed-data, використовуючи вкладені в нього сертифікати. Функція getCMSData повертає тільки контент з повного ASN1 об’єкту CMS типу signed-data. Функція getSignerCertFromCMS повертає X509-сертифікат, яким підписувалася CMS

типу signed-data з сирого об’єкту цього типу. Іллюстрація деяких криптографічних функцій наведена в додатку Д.

Клас Utils з пакету CryptoGraphy складається з допоміжних статичних методів, які виконують рутинні службові операції, що потрібні додатку. Серед яких: читання об’єкту типу stream; повернення поточної дати; переведення символьних строк в 16-тирічні строки; узагальнене безпечне перетворення одного об’єкта в інший; процедура перевірки вхідного повідомлення та його дешифрування; отримання імен користувачів з черги непідтвержених повідомлень; збереження черги повідомлень в БД; повернення всіх сертифікатів, які належать користувачу; процедура додавання користувача по його сертифікату.

3.5 Тестування програмного забезпечення

Тестування базового функціоналу проводилося на 3 пристроях з різними версіями ОС Android — 7, 8 та 6. Мінімальний рівень Android API, що підтримує додаток — 19, що відповідає версії 4.4. Обов’язковою вимогою до кінцевого пристрою є працездатні сервіси Google(такі як PlayMarket), оскільки додаток використовує хмарну БД цієї компанії, замість власної централізованої інфраструктури сервера посередника між користувачами додатку. В інших випадках воно працювати не буде.

Тестування локальних функцій додатку є найлегшим вектором тестування для додатку оскільки візуальні частини відображають стани даних які наочно можна побачити та потребують одного пристрою. Тому функціонал по додаванню користувача по сертифікату та генерація асиметричних пар ключів та вкладення їх в сертифікат є візуально наочним та взаємозалежним з кількістю елементів, що відображаються в фрагменті з сертифікатами та фрагменті з контактами. Крім того впевнитися в коректно згенерованому сертифікаті можна натиснуваши на нього — чим викликати його експорт, після чого подивитися на сиру ASN1 структуру та її зміст. Після цього тестування можна зробити деякі висновки: локальна БД має коректну структуру; сутності

користувачів та сертифікатів корректно додаються до локальної БД; криптографічна складова щодо генерації асиметричних пар ключів, створення сертифікату і його експорту та симетричного шифрування полів з приватними ключами працює корректно; вікно вибору способу обміну сертифікатом також працює очікувано.

Тестування відправки повідомлень може відбуватися наступним чином: після процедури додавання користувача та його візуальній появі на вкладці контактів з’являється можливість натиснути на нього тим самим викликається перехід на екран з повідомленнями що стосуються цього контакта та користувача додатку. При першому натисканні на контакт має бути вибраний сертифікат користувача(а з ним і приватний ключ) зі списку його сертифікатів, який буде використовуватися для комунікації з цим контактом. Після цього вибору всі повідомлення мають підписуватися приватним ключем, що відповідає публічному ключу сертифікату. Для віддаленого перегляду хмарної БД та контенту, що змінився в ній існує Firebase Console. При правильній роботі додатку після шифрування та підписування повідомлення та зберігання в локальній БД, модель повідомлення повинена з’явитися в хмарній БД по окремому шляху, що призначений для віддаленого пристрою. Після появи на екрані повідомлення та в консолі хмарної БД, яке було щойно написане локальним користувачем можна зробити наступні висновки: таблиця БД з повідомленнями працює правильно оскільки повідомлення додаються; кнопка відправки та поле контенту також працюють очікувано; відправка повідомлень в хмарну БД працює корректно; хмарна БД має заплановану структуру, яка відповідає структурі з етапу проектування; криптографічна складова по генерації CMS типу enveloped-data та signed-data, тобто шифрування та підписування повідомлень працює очікувано. Крім того з консолі хмарної БД можна побачити сиру ASN1 структуру повідомлення та її складові.

Тестування на отримання повідомлень від контактів відбувається за таким же сценаріем, як і тестування відправки від поточного користувача. Однак до

тестування додається ще один пристрій, що є відправником. Корректна робота додатку щодо прийому адресованих йому повідомлень передбачає прийом цих повідомлень, якщо додаток запущений та пристрій знаходиться в мережі. Незалежно від екрану на якому знаходиться користувач(екран контактів та сертифікатів або екран повідомлень) додаток повинен корректно обробляти прийом повідомлень. Крім того в будь-який момент експлуатації додатку можуть прийти повідомлення від невідомого контакту, про що повинен бути проінформований користувач через обов’язковий діалог з підтвердженням невідомого користувача. Також передбачена ситуація коли пристрій знаходиться поза межами інтернету, тоді пристрій відправник пише в хмарну БД свої повідомлення, що адресовані віддаленій стороні, які потрапляють до шляху в хмарній БД, що стосується виключно віддаленої сторони. Після того як додаток з віддаленої сторони буде запущений на пристрої він забере з хмарної БД повідомлення що адресовані йому. Вищезгаданий функціонал після декількох циклів тестування працює очікувано. Парадигмою, що була обрана в якості негативного тестування додатку є фаззинг, тобто вставка завідомо неправильних або неочікуваних даних до криптографічної підсистеми додатку. Серед таких даних: CMS без підпису або з неправильним підписом, CMS(enveloped-data та signed-data) з неправильною ASN1 структурою, CMS з відсутніми обов’язковими елементами структури ASN1, що передбачені стандартом. В таких ситуаціях додаток просто ігнорує такі дані, що і є очікуваною поведінкою.

Оскільки додаток виконує функції месенжера та його основний набір функцій працюють корректно(прийом та відправлення повідомлень, з іншими допоміжними етапами, серед яких: шифрування, збереження в локальній БД, взаємодія з хмарною БД та інші), що було доведено на етапі тестування, можна констатувати факт того що додаток відповідає тематиці проекту та цілком відповідає вимогам що були описані в ТЗ.

3.6 Висновки по розділу 3

ПЗ, яке виконує функції месенжера є вкрай складним для реалізації оскільки потребує впровадження складної логіки як на кліентському ПЗ так і на серверній стороні ПЗ. Складними деталями з поточної імплементації логіки мессенджера є: повністю подійна модель взаємодії як між внутрішніми компонентами додатку так і з компонентами системи; слідування шаблону проектування MVP, тобто логічно розподілений код на рівні — моделей, відображення та логіки; узгодження життєвого циклу основних елементів(3 класа Активностей) та потоків, що виконують тривалі ресурсоемні операції, для уникнення блокування потоку, який відповідає за відтворення графічного інтерфейсу(UI); формування складних зв“язків реляційної БД через контент провайдер; формування криптографічних контейнерів та сертифікатів, з редагуванням об“єктів формату ASN.1; нетривіальна графічна складова ПЗ.

Однією з чеснот додатку з цього проекту є децентралізованість, що реалізована через можливості хмарної БД Firebase. Чим впроваджується доступність достатнього рівня, яка залежить від пропускної здатності до хмарних сервісів Google відносно поточної локації та нівелює недоступність сервісу через можливість відмови однієї точки. Конфеденційність користувача додатку забезпечується через: симетричне шифрування його приватних ключів в локальній БД з використанням солі; симетричне внутрішне шифрування контенту повідомлення, що пов’язане з поточною(віддаленою) асиметричною парою користувача; підписування повідомлень; використання токену хмарної БД, що є персональним для пристрою. Забезпечення цілісності контенту повідомлень забезпечується через хешування та підпис цього хешу публічним ключем користувача. Зважаючи на всі ці обставини додаток в достатній мірі забезпечує цілісніть, конфеденційність та доступність.

ВИСНОВКИ

Параметр якості в захисті ОС мобільних пристроїв є вкрай важливим, оскільки пристрої зберігають численну кількість персональної та локалізованої інформації, доступ до якої повинен мати тільки власник цього пристрою. Android - найпопулярніша мобільна ОС, вібрала в себе всі найкращі техніки та практики експлуатації Unix-систем, зокрема ОС Linux. Засоби захисту ОС Android одні з найпродуктивніших, що доведено практикою використання тривалістю протягом 11 років та загальною експлуатацією кількісно близько 2.5 млрд користувачів. Основна проблематика - це своєчасне оновлення системи, яке може не доходити до користувачів через політики щодо підтримки пристроїв самим виробником. Процент використання месенжерів на мобільних платформах користувачами є вкрай великим. Це пов’язане з комфортом при використанні їхніх функцій та потребою в комунікації між собою користувачів. Однак комерціалізоване ПЗ, фінансово зацікавлено у розширених даних(місцеположення, вподобання тощо) про користувача, внаслідок чого конфеденційність користувача забезпечена на слабкому рівні. ПЗ з цього проекту прибирає цю прогалину. Тому можна сказати що, актуальність тематикі проекту прямопропорційна популярності месенжерів на мобільних платформах.

В цій роботі було розглянуто сучасні методи криптографічного захисту інформації та їхню роль в захисті інформації вцілому в умовах сучасного насиченого інформаційного простору, та впровадження ними послуг щодо захисту інформації, зокрема її цілісності та конфеденційності. Ефективне їх впровадження, як правило, потребує від розробника розуміння математичних складових криптографічних алгоритмів, задля індексації їхньої криптостійкості та досягнення найоптимальніших показників в кінечній реалізації, та деяких зусиль під час їхньої імплементації. В процесі тестування ПЗ показало свою корректну роботу, щодо заявленого функціоналу та може використовуватися на практиці впроваджуючи властивості конфеденційності та цілісності повідомлень, що підлягають обміну між користувачами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аргановский А.В. Практическая криптография: алгоритми и их програмирование / Аргановский А.В, Хади Р.А — М.: СОЛОС-Пресс, 2009. — 256 с.
2. Бакланов В.В. Введение в информационную безопасность. Направления информационной защити: уч. Пособие /Бакланов В.В,2012. — 235 с.
3. Беломойцев Д.Е. Основние методи криптографической обработки даних: уч.пособие / Беломойцев Д.Е. — М.: Изд-во МГТУ им Н.Е.Баумана, 2014. — 76 с.
4. Бессалов А.В. Елиптические кривие в форме Едвардса и криптография: монография /Бессалов А.В. — Киев: ІВЦ «Видавництво «Політехніка»»,2017. — 272 с.
5. Брассар Ж. Современная криптология.Руководство /Брассар Ж. — Москва, издательско-полиграфическая фирма ПОЛИМЕД, 1999. — 178 с.
6. Bitcoin White Paper [Електронний ресурс]. Режим доступу World Wide Web. — URL: https://bitcoin.org/bitcoin.pdf
7. Cisco Network Academy: CCNA Security Course Booklet Version 1, 2010. — 428 с.
8. Венбо М. Современная криптография: теория и практика./ пер. с англ. М.: Издательский дом «Вильямс», 2005. — 768 с.
9. Вигерс К. Разработка требований к программному обеспечению/Пер с англ. - М.: «Русская Редакция», 2004. - 576 с.
10. Документація по LATEX [Електронний ресурс]. Режим доступу World Wide Web.— URL: https://www.latex-project.org/help/documentation
11. Документація по ОС Android з рівня МП Java [Електронний ресурс]. Режим доступу World Wide Web. — URL: https://developer.android.com/guide

12. Документація по БД SQLite [Електронний ресурс]. Режим доступу World Wide Web.— URL: https://www.sqlite.org/docs.html
13. Дрейк Д. Android Hacker Handbook/ Дрейк Д., 2014. — 544 с.
14. Established Mandatory Access Control on Android OS [Електронний ресурс]. Режим доступу World Wide Web. — URL: доступ (*https://publikationen.sulb.uni-saarland.de/bitstream/thesis_bugiel*)
15. Еленков Николай: Android Security Internals. An In-Depth Guide to Android“s Security Architecture, 2015. — 406 с.
16. Еранссон А. Ефективное использование потоков в операционной системе Android / пер. с англ. А.Снастина — М.: ДМК Пресс, 2018.— 312 с.
17. Кобліц Н. Курс теорії чисел та Криптографії/ Кобліц Н. — Москва, наук. видавництво ТВП, 2001. — 254 с.
18. Молдовян С.А. Введение в криптосистеми с открытим ключом/Молдовян С.А, Молдовян А.А - Спб.: БХВ-Петербург, 2005. — 288 с.
19. Панасенко С.П. Алгоритми шифрування. Спеціальний довідник /Панасенко С.П — СПб.: БХВ-Петербург, 2009. — 576 с.
20. Результати 2 раунду конкурсу щодо стандартизації постквантових алгоритмів NIST [Електронний ресурс]. Режим доступу World Wide Web. — URL: https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
21. RFC-5652 [Електронний ресурс]. Режим доступу World Wide Web. — URL: https://tools.ietf.org/html/rfc5652
22. RFC-5114 [Електронний ресурс]. Режим доступу World Wide Web. — URL: https://tools.ietf.org/html/rfc5114
23. RFC-7919 [Електронний ресурс]. Режим доступу World Wide Web. — URL: https://tools.ietf.org/html/rfc7919

24. Смарт Н. Криптография/пер. С англ. С.А.Кулешова — Москва.: Техносфера, 2005. — 528 с.

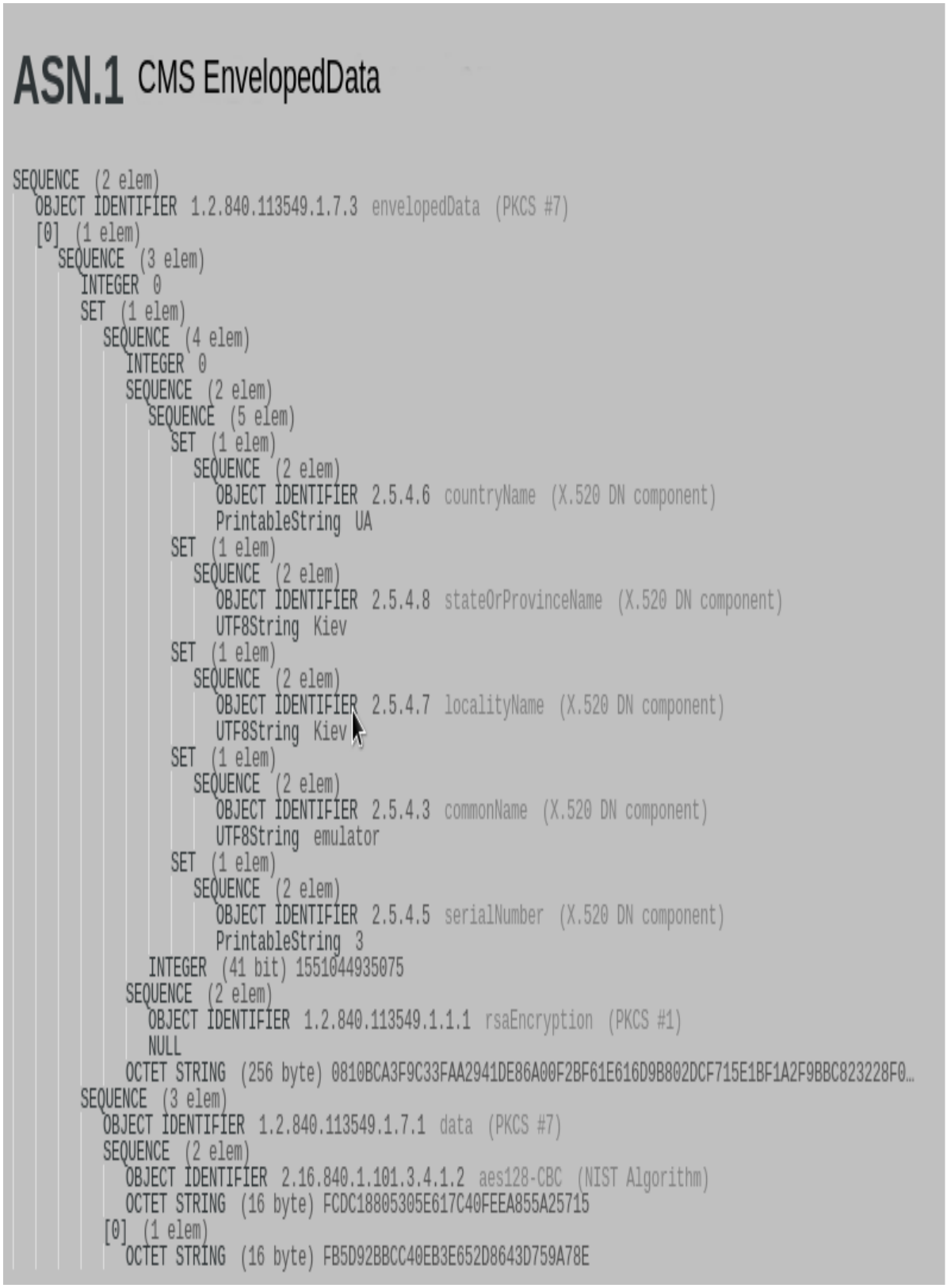
25. Фергюсон Н. Практическая криптография/ Фергюсон Н., Шнайер Б.; пер. с англ. — М.: Издательский дом «Вильямс», 2005. — 424 с.

26. Шаньгин В.Ф. Защита информации в компьютерних системах и сетях / Шаньгин В.Ф. — М.: ДМК Пресс, 2012. — 592 с.

27. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы и исходные тексты на языке Си / Шнайер Б.— М.: Издательство Триумф, 2003.— 816 с.

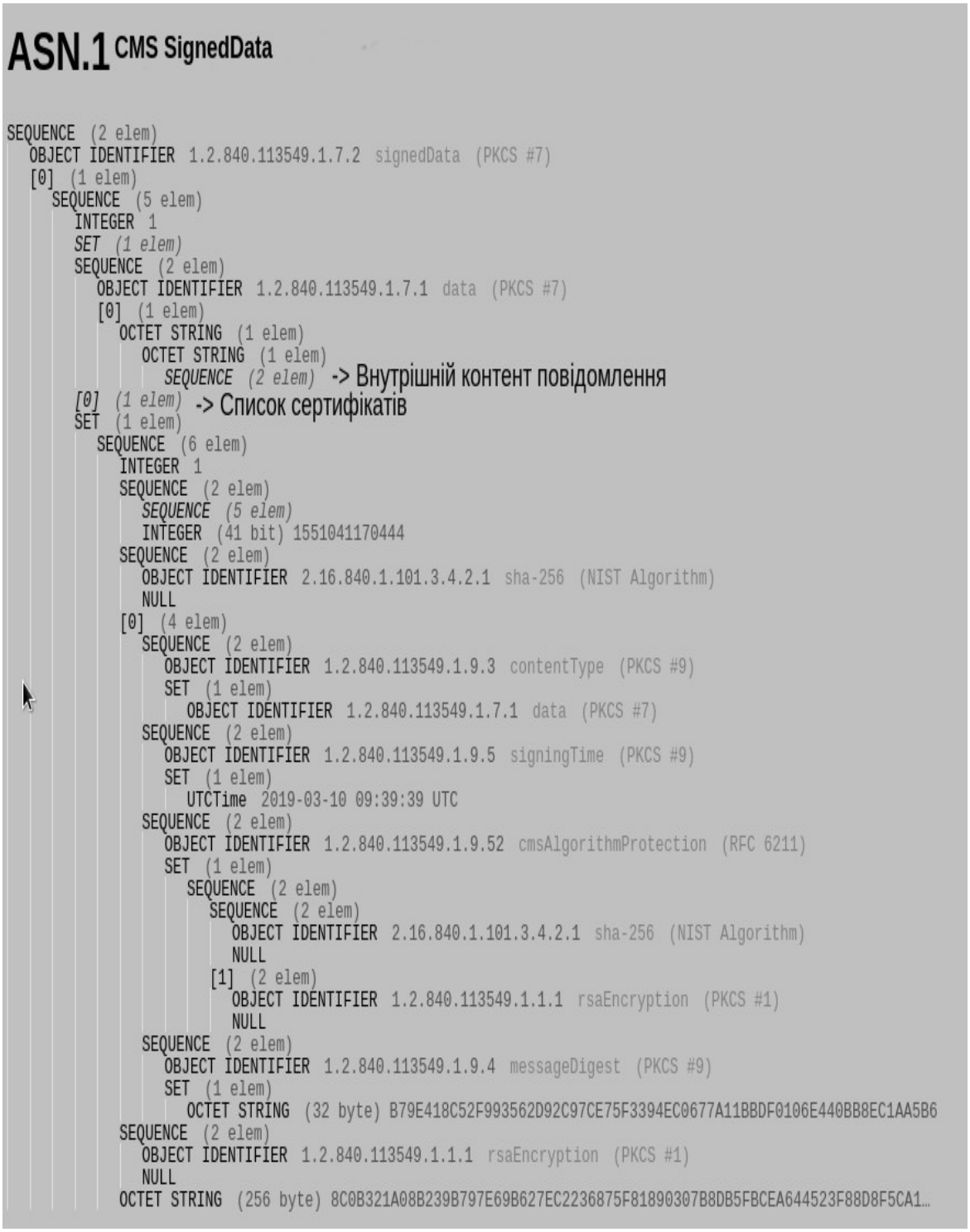
Додаток А

ASN1 структура внутрішнього контенту повідомлення



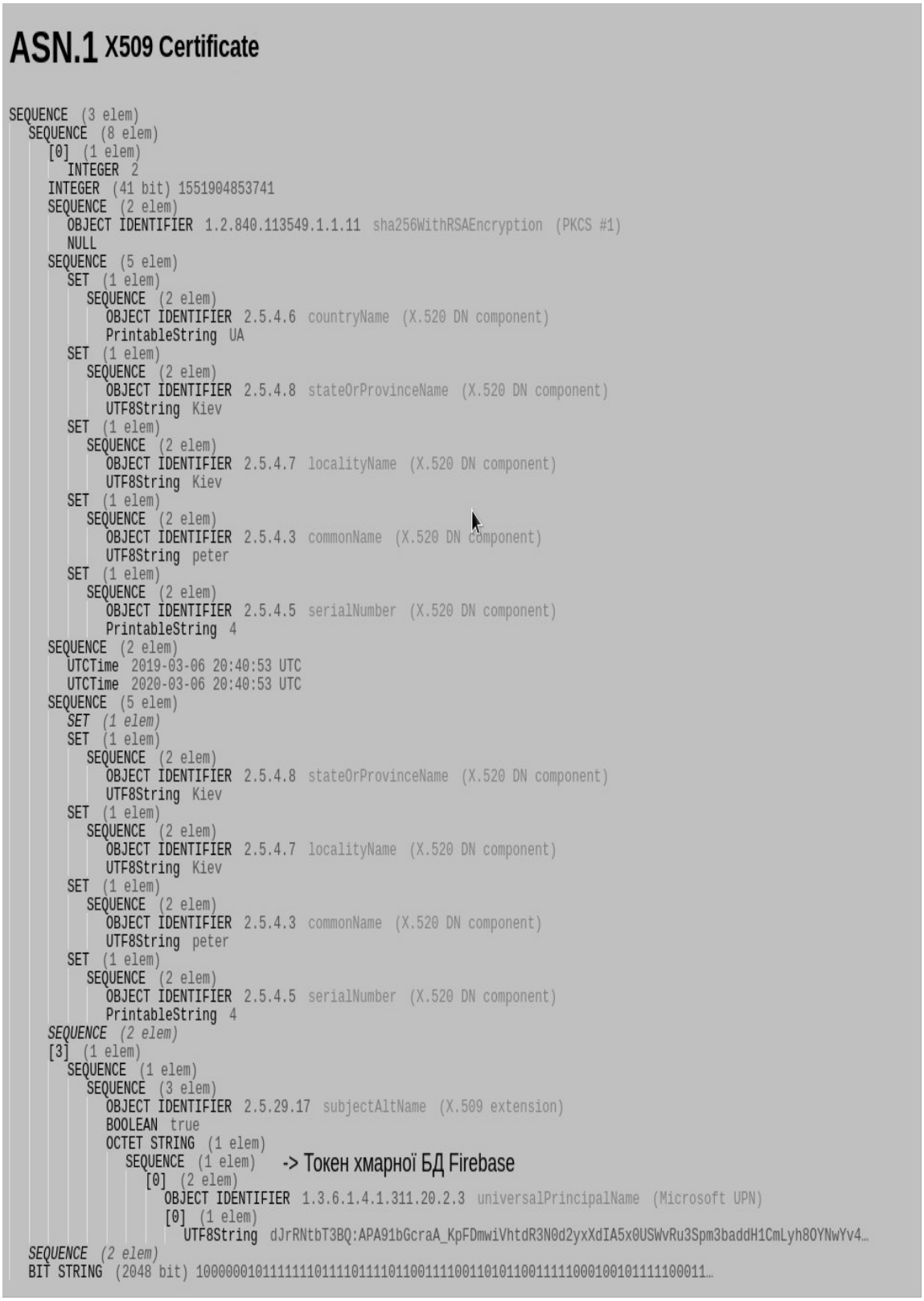
Додаток Б

ASN1 структура зовнішнього контенту повідомлення



Додаток В

ASN1 структура X509 контейнеру



Додаток Г

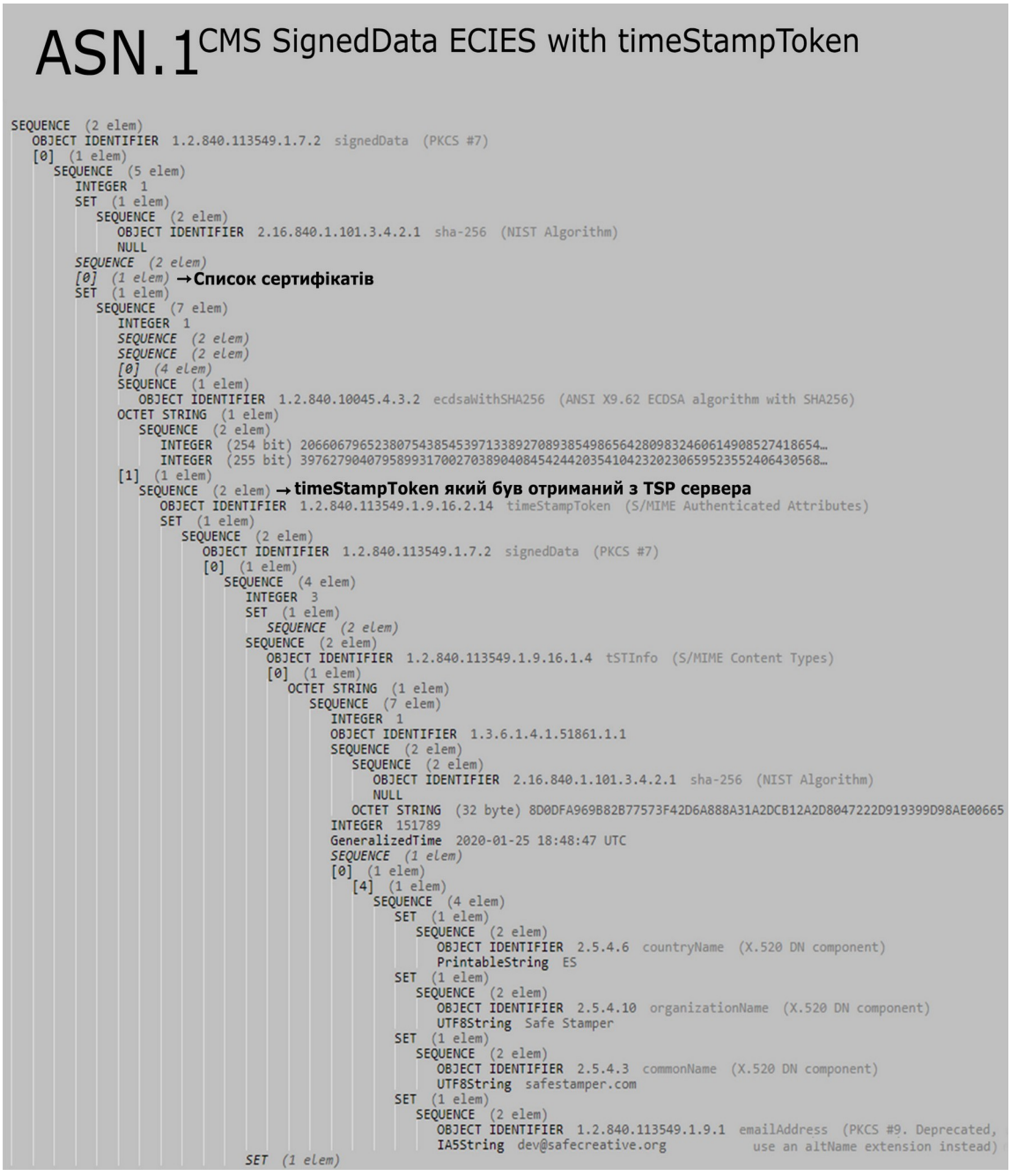
ASN1 структура зовнішнього контенту повідомлення в схемі ECIES

ASN.1 CMS SignedData following ECIES

```
SEQUENCE  (2 elem)
   OBJECT IDENTIFIER  1.2.840.113549.1.7.2  signedData  (PKCS #7)
   [0]  (1 elem)
      SEQUENCE  (5 elem)
         INTEGER  1
         SET  (1 elem)
            SEQUENCE  (2 elem)
               OBJECT IDENTIFIER  2.16.840.1.101.3.4.2.1  sha-256  (NIST Algorithm)
               NULL
         SEQUENCE  (2 elem)
            OBJECT IDENTIFIER  1.2.840.113549.1.7.1  data  (PKCS #7)
            [0]  (1 elem)                                          Внутрішній контент повідомлення (R|c|d)
               OCTET STRING  (1 elem)                                          ↙
                  OCTET STRING  (177 byte) 04B615B4FA27AB8F5588097BF8B6B6B2D347B8F6D1433F96307AC954DFA8898C830DB…
         [0]  (1 elem)→Список сертифікатів
         SET  (1 elem)
            SEQUENCE  (6 elem)
               INTEGER  1
               SEQUENCE  (2 elem)
                  SEQUENCE  (5 elem)
                     SET  (1 elem)
                        SEQUENCE  (2 elem)
                           OBJECT IDENTIFIER  2.5.4.6  countryName  (X.520 DN component)
                           PrintableString  UA
                     SET  (1 elem)
                        SEQUENCE  (2 elem)
                           OBJECT IDENTIFIER  2.5.4.8  stateOrProvinceName  (X.520 DN component)
                           UTF8String  Kiev
                     SET  (1 elem)
                        SEQUENCE  (2 elem)
                           OBJECT IDENTIFIER  2.5.4.7  localityName  (X.520 DN component)
                           UTF8String  Kiev
                     SET  (1 elem)
                        SEQUENCE  (2 elem)
                           OBJECT IDENTIFIER  2.5.4.3  commonName  (X.520 DN component)
                           UTF8String  петро
                     SET  (1 elem)
                        SEQUENCE  (2 elem)
                           OBJECT IDENTIFIER  2.5.4.5  serialNumber  (X.520 DN component)
                           PrintableString  1
                  INTEGER  (41 bit) 1579562789273
               SEQUENCE  (2 elem)
                  OBJECT IDENTIFIER  2.16.840.1.101.3.4.2.1  sha-256  (NIST Algorithm)
                  NULL
               [0]  (4 elem)
                  SEQUENCE  (2 elem)
                     OBJECT IDENTIFIER  1.2.840.113549.1.9.3  contentType  (PKCS #9)
                     SET  (1 elem)
                        OBJECT IDENTIFIER  1.2.840.113549.1.7.1  data  (PKCS #7)
                  SEQUENCE  (2 elem)
                     OBJECT IDENTIFIER  1.2.840.113549.1.9.5  signingTime  (PKCS #9)
                     SET  (1 elem)
                        UTCTime  2020-01-25 17:15:19 UTC
                  SEQUENCE  (2 elem)
                     OBJECT IDENTIFIER  1.2.840.113549.1.9.52  cmsAlgorithmProtection  (RFC 6211)
                     SET  (1 elem)
                        SEQUENCE  (2 elem)
                           SEQUENCE  (2 elem)
                              OBJECT IDENTIFIER  2.16.840.1.101.3.4.2.1  sha-256  (NIST Algorithm)
                              NULL
                           [1]  (1 elem)
                              OBJECT IDENTIFIER  1.2.840.10045.4.3.2  ecdsaWithSHA256  (ANSI X9.62 ECDSA algorithm with SHA256)
                  SEQUENCE  (2 elem)
                     OBJECT IDENTIFIER  1.2.840.113549.1.9.4  messageDigest  (PKCS #9)
                     SET  (1 elem)
                        OCTET STRING  (32 byte) B6E95FBAC4209C3679DDEC1B070BFD1DECCF8202CDE6B086DEFC9972A8F30C0D
               SEQUENCE  (1 elem)
                  OBJECT IDENTIFIER  1.2.840.10045.4.3.2  ecdsaWithSHA256  (ANSI X9.62 ECDSA algorithm with SHA256)
               OCTET STRING  (1 elem)
                  SEQUENCE  (2 elem)
                     INTEGER  (254 bit) 266524722959779119874508605315884805030796841578054204811102071060284 8…
                     INTEGER  (256 bit) 797117372295353943285186301751966696149553365056448377152741994958585 5…
```

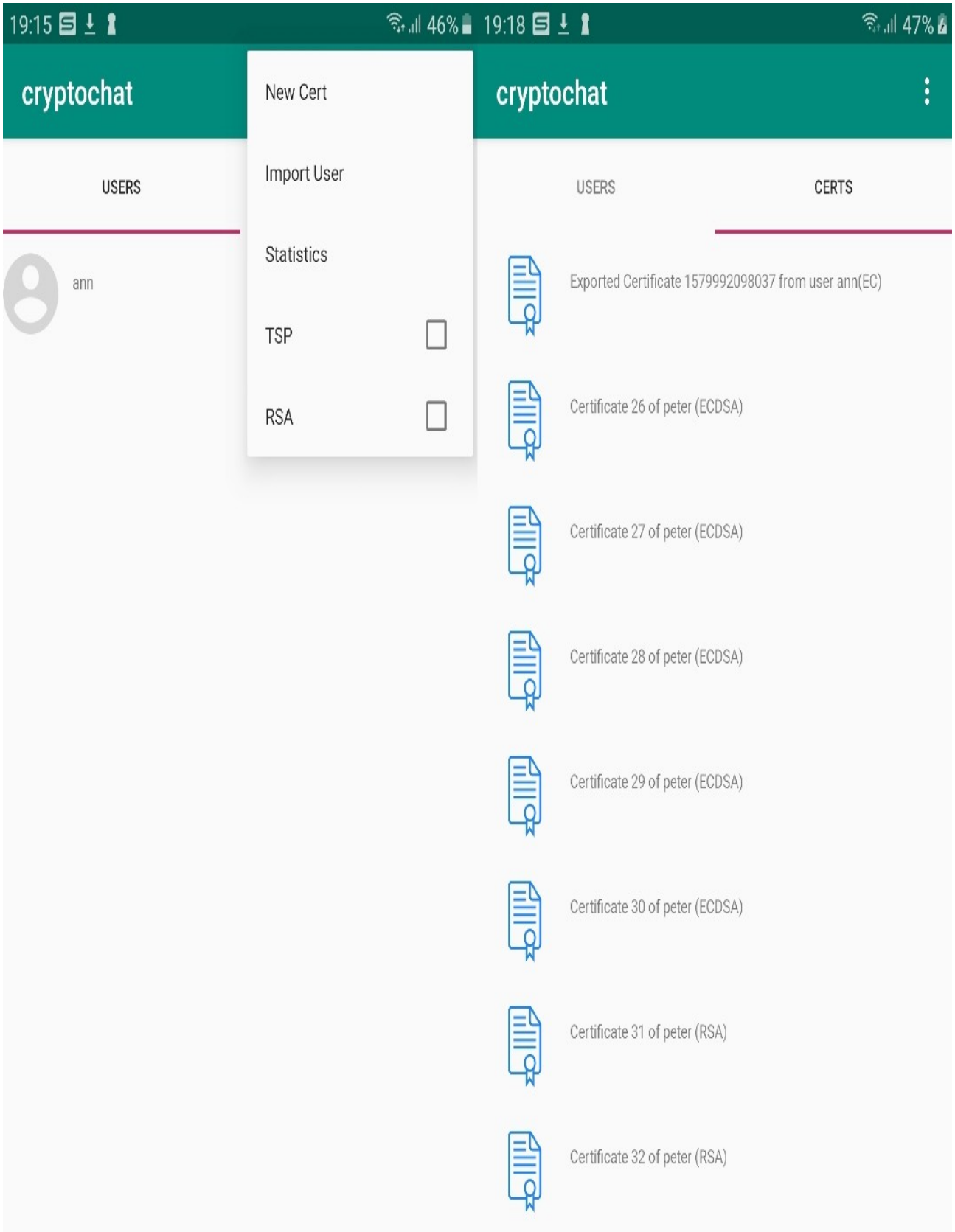
Додаток Д

ASN1 структура зовнішнього контенту повідомлення з TimeStampToken аттрибутом

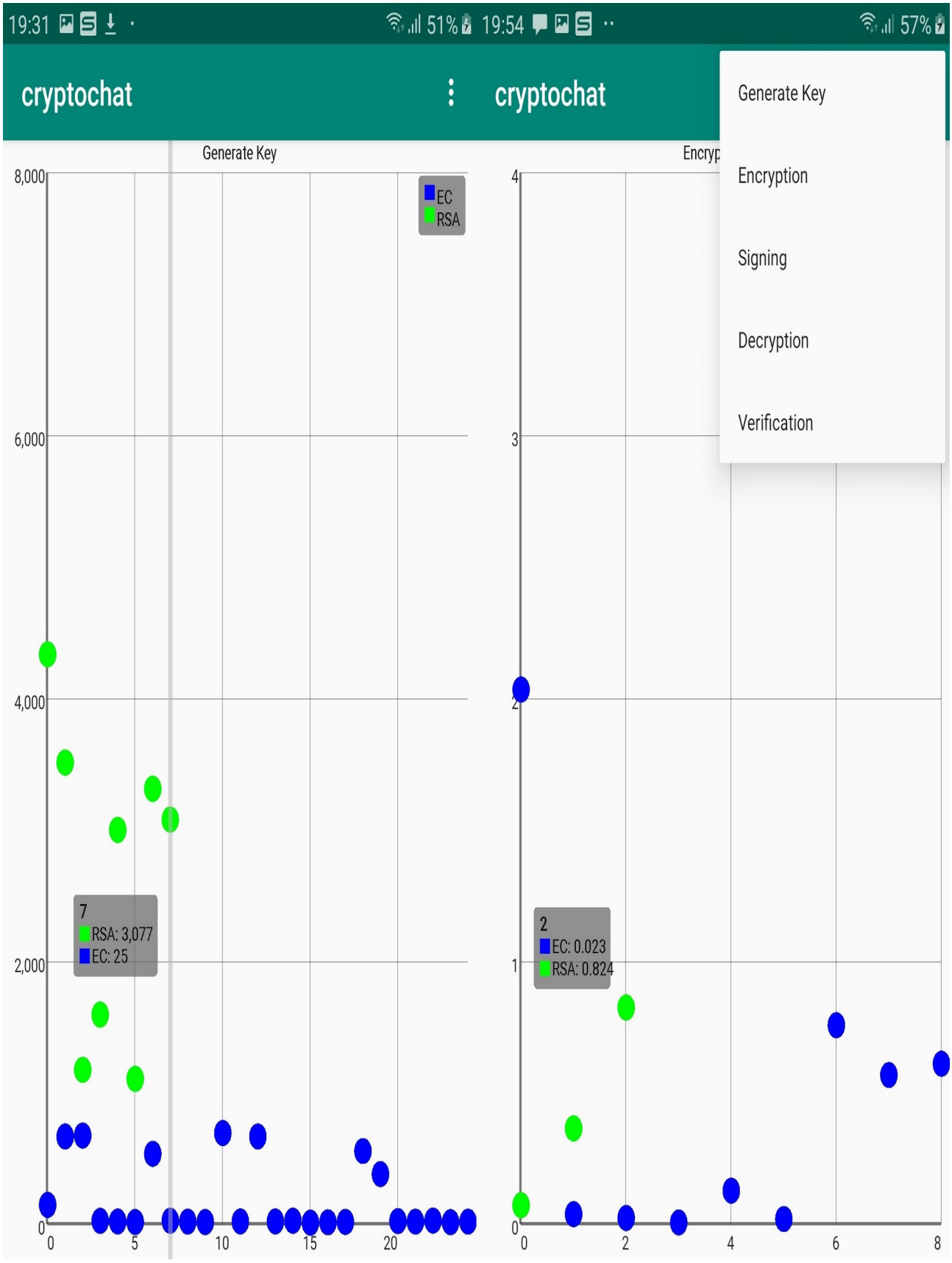


Додаток Е

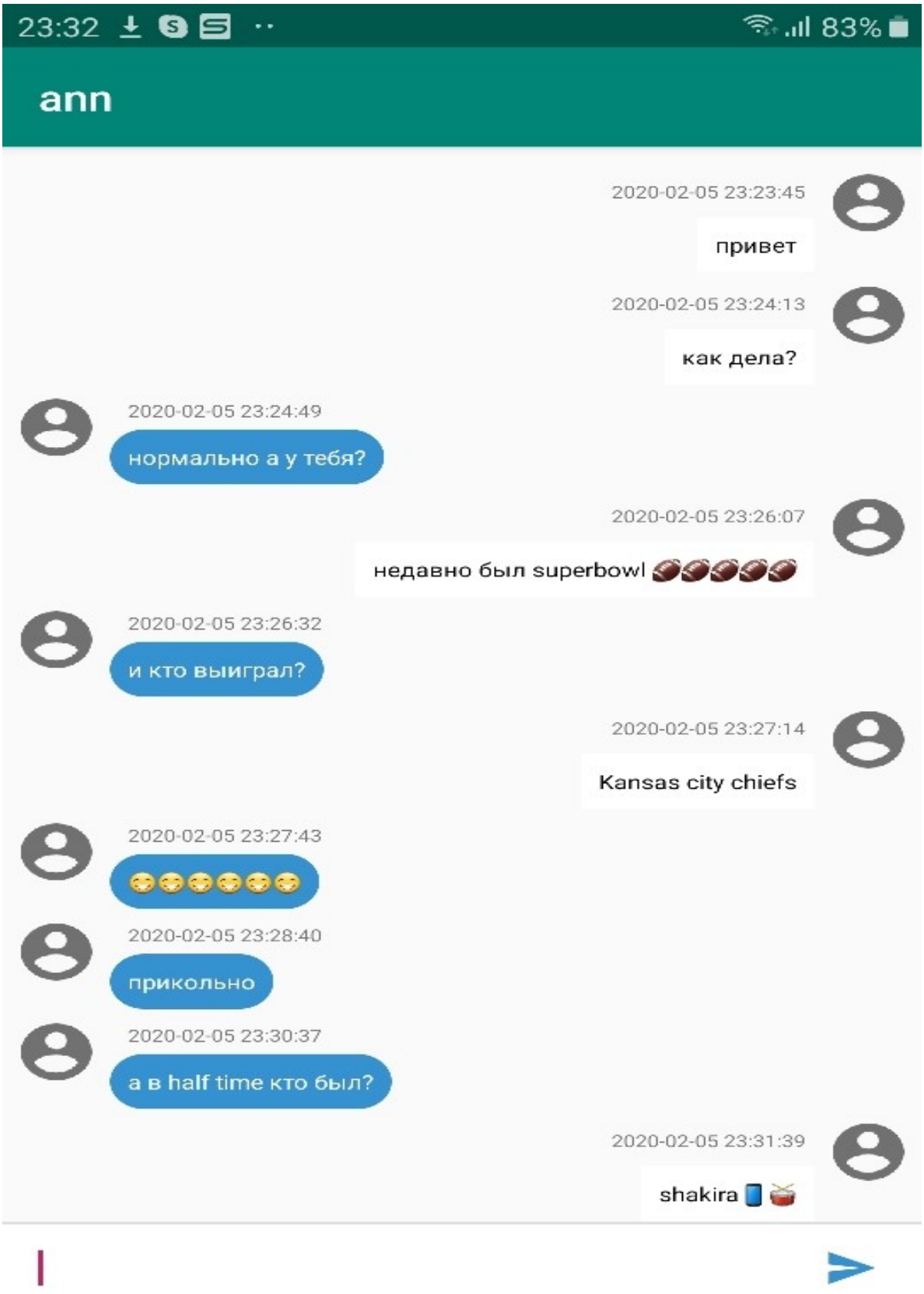
Візуальна складова додатку



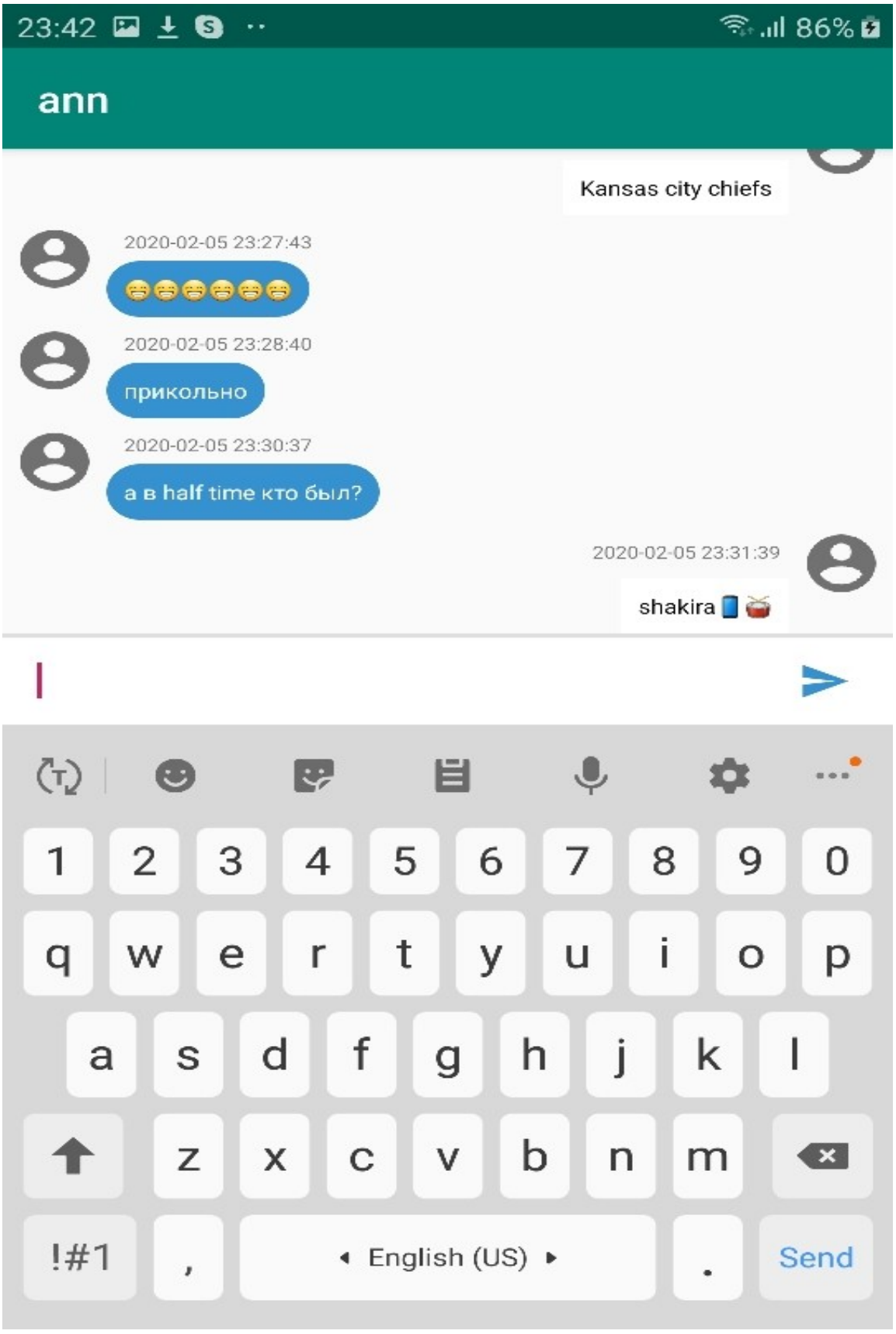
Продовження додатку Е



Продовження додатку Е



Продовження додатку Е



Додаток Ж

Основні криптографічні функції додатку

```
public class CryptoUtils {
    private static Provider bcProvider;
    private static SecureRandom random;
    private static final String TAG="CryptoProvider";
    static {
        bcProvider = new org.spongycastle.jce.provider.BouncyCastleProvider();
        Security.addProvider( bcProvider );
        random = new SecureRandom();
    }
    public static KeyPair generateKeyPair(int keySize, String type)  throws
NoSuchAlgorithmException, InvalidAlgorithmParameterException {
        KeyPairGenerator keypairGen = KeyPairGenerator.getInstance(type);
        if (type.equals("RSA")) {
            keypairGen.initialize(keySize, random);
        } else {
            ECNamedCurveParameterSpec curve =
ECNamedCurveTable.getParameterSpec("secp256k1");
            keypairGen.initialize(curve,random);
        }
        return keypairGen.generateKeyPair();
    }
    public static Certificate generateX509Cert(KeyPair keyPair, String subjectDN, String
altName)
            throws OperatorCreationException, CertificateException, IOException {
        long now = System.currentTimeMillis();
        Date startDate = new Date(now);

        X500Name dnName = new X500Name(subjectDN);
        BigInteger certSerialNumber = new BigInteger(Long.toString(now));
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(startDate);
        calendar.add(Calendar.YEAR, 1); //  validity period

        Date endDate = calendar.getTime();
        String signatureAlgorithm =
                keyPair.getPublic().getAlgorithm().equals("RSA") ? "SHA256WithRSA":
                        "SHA256withECDSA";

        ContentSigner contentSigner = new JcaContentSignerBuilder(signatureAlgorithm)
```

```
                .setSecureRandom(random)
                .setProvider(BouncyCastleProvider.PROVIDER_NAME)
                .build(keyPair.getPrivate());

        JcaX509v3CertificateBuilder certBuilder = new JcaX509v3CertificateBuilder(dnName,
certSerialNumber, startDate, endDate, dnName, keyPair.getPublic());
        if( altName != null) {
            try {
                ASN1EncodableVector otherName = new ASN1EncodableVector();
                otherName.add(new DERObjectIdentifier("1.3.6.1.4.1.311.20.2.3"));
                otherName.add(new DERTaggedObject(true, 0, new DERUTF8String( altName )));
                ASN1Object genName = new DERTaggedObject(false, 0, new
DERSequence(otherName));
                ASN1EncodableVector genNames = new ASN1EncodableVector();
                genNames.add(genName);
                certBuilder.addExtension(Extension.subjectAlternativeName, true, new
DERSequence(genNames));
            }catch (Exception ignore){}
        }
        return new
JcaX509CertificateConverter().setProvider(bcProvider).getCertificate(certBuilder.build(conte
ntSigner));
    }
    public static byte[] SHA256(final String cont) {
        MessageDigest md = null;
        try {
            md = MessageDigest.getInstance("SHA-256");
            md.update(cont.getBytes());
        }catch (Exception ignore){}
        if( md != null)
           return md.digest();
        return null;
    }
    public static String getRandomString(int len) {
        byte[] bytes = random.generateSeed(len);
        return new String(bytes);
    }
    public static byte[] aesEncrypt(byte[] key, byte[] data, boolean action) {
        Cipher cipher = getAesCipher();
        if( cipher == null)
            return null;
        SecretKey aesKey = new SecretKeySpec(key, "AES");
        try {
            cipher.init( action ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, aesKey);
```

```
            return  cipher.doFinal(data);
        }catch (Exception ignore){
            return null;}
    }
    public static X509Certificate readCertificateFromStream(final Context context, final Uri
uri) {
        String cont=null;
        X509Certificate cert = null;
        Pattern parse = Pattern.compile("(?m)(?s)^---*BEGIN.*---*$(.*)^---*END.*---*$.*");
        try (InputStream is = context.getContentResolver().openInputStream(uri) ) {
            byte[] bcont = Utils.readStream(is);
            cont = new String(bcont);
        } catch (Exception ignore) {}
        if( cont != null) {
            String encoded = parse.matcher(cont).replaceFirst("$1");
            try {
                byte[] raw_cert = android.util.Base64.decode(encoded,
android.util.Base64.DEFAULT);
                CertificateFactory cf = CertificateFactory.getInstance("X.509");
                cert = (X509Certificate) cf.generateCertificate(new
ByteArrayInputStream(raw_cert));

            } catch (Exception ignore){}
        }
        return cert;
    }
    public static Map<String,String> parseDnSubj(final String dn) {
        Map<String,String> res = new HashMap<>();
        String[] parts = dn.split(",");
        for( final String part: parts) {
            String[] dnChunks = part.split("=");
            if( dnChunks.length == 2) {
                res.put(dnChunks[0], dnChunks[1]);
            }
        }
        return res;
    }
    private static Pair<Boolean,String> makeResult( boolean state, String val) {
        return Pair.create(state, val);
    }
    public static Pair<Boolean,String> parseX509Certificate(final X509Certificate cert,
Map<String, String> values ) {

        Principal principal = cert.getSubjectDN();
```

```
        if( principal == null) {
            return makeResult(false, "Certificate not provide Directory Name");
        }
        Map<String,String> dn = parseDnSubj(principal.getName());
        values.putAll(dn);
        if( !values.containsKey("CN")) {
            return makeResult(false, "Certificate not contain  CommonName");
        }
        values.put("certSerialNumber", cert.getSerialNumber().toString());
        try {
            Collection<List<?>> altNames =  cert.getSubjectAlternativeNames();
            if( altNames == null)
                  throw new Exception();
            String fbToken = null;
            for( final List<?> altName: altNames) {
                   fbToken = readSubjectAltNameChunk((Integer)altName.get(0),
(byte[])altName.get(1));
                   if( fbToken != null) {
                       break;
                   }
            }
            values.put("fbToken", fbToken);
        }catch (Exception ignore){
            Log.e(TAG, ignore.toString());
            return makeResult(false, "Subject alt name not valid"); }
        return Pair.create(true, null);
    }
    private static String readSubjectAltNameChunk( Integer type, byte[] rawAsn1 ) throws
Exception{
        String result = null;
        if( type != 0)
            return result;
        ASN1InputStream input = new ASN1InputStream(rawAsn1);
        ASN1Primitive p = input.readObject();
        ASN1TaggedObject tObj = DERTaggedObject.getInstance(p);
        ASN1Sequence seq = DLSequence.getInstance(tObj.getObject());
        if( seq.size() < 2 )
            return result;
        ASN1ObjectIdentifier ids = ASN1ObjectIdentifier.getInstance(seq.getObjectAt(0));
        if( ids == null)
            return result;
        if( ids.getId().equals("1.3.6.1.4.1.311.20.2.3")) {
            ASN1TaggedObject objs = DERTaggedObject.getInstance(seq.getObjectAt(1));
            DERUTF8String octstr = DERUTF8String.getInstance(objs.getObject());
```

```
        result = octstr.toString();
    }
    return result;
}
public static String getCertFirebaseToken(final Map<String,String> params) {
    return params.get("fbToken");
}
public static String getCertSerialNumber( final  Map<String,String> params) {
    return params.get("certSerialNumber");
}
public static byte[] encryptCMSData(byte[] data,
                                X509Certificate encryptionCertificate ) {

    byte[] encryptedData = null;
    if (null != data && null != encryptionCertificate) {
        CMSEnvelopedDataGenerator cmsEnvelopedDataGenerator
                = new CMSEnvelopedDataGenerator();
        try {
            JceKeyTransRecipientInfoGenerator jceKey
                    = new JceKeyTransRecipientInfoGenerator(encryptionCertificate);
            CMSTypedData msg = new CMSProcessableByteArray(data);
            String algo = encryptionCertificate.getPublicKey().getAlgorithm();
            if (algo.equals("RSA")) {
                OutputEncryptor encryptor
                        = new JceCMSContentEncryptorBuilder(CMSAlgorithm.AES128_CBC)
                        .setProvider(BouncyCastleProvider.PROVIDER_NAME).build();
                cmsEnvelopedDataGenerator.addRecipientInfoGenerator(jceKey);
                CMSEnvelopedData cmsEnvelopedData = cmsEnvelopedDataGenerator
                        .generate(msg, encryptor);
                encryptedData = cmsEnvelopedData.getEncoded();
            } else {
                IESEngine engine = new IESEngine(new ECDHBasicAgreement(),
                        new KDF2BytesGenerator(new SHA256Digest()),
                        new HMac(new SHA256Digest()),
                        new PaddedBufferedBlockCipher(

                            new CBCBlockCipher(new AESEngine())));
                IESCipher cipher = new IESCipher(engine);
                IESParameterSpec parameterSpec = new IESParameterSpec(null, null,
                        256, 256, null, false);
                cipher.engineInit(Cipher.ENCRYPT_MODE,

                                encryptionCertificate.getPublicKey()
                        , parameterSpec, new SecureRandom());
                encryptedData = cipher.engineDoFinal(data, 0, data.length);
            }
```

```
            }catch (Exception ignore){ Log.w(TAG, ignore.toString());}
        }
        return encryptedData;
    }
    public static byte[] decryptCMSData(byte[] data,X509Certificate certificate, PrivateKey
privateKey) {
        byte[] decryptData = null;
        if (null != data && null != privateKey) {
           try {
               if (certificate.getPublicKey().getAlgorithm().equals("RSA")) {
                   CMSEnvelopedData envData = new CMSEnvelopedData(data);
                   RecipientInformationStore recipients = envData.getRecipientInfos();
                   Collection c = recipients.getRecipients(

                               new   JceKeyTransRecipientId(certificate));
                   Iterator it = c.iterator();
                   if (it.hasNext()) {
                       RecipientInformation recipient = (RecipientInformation) it.next();
                       return recipient.getContent(new   JceKeyTransEnvelopedRecipient(

                           privateKey).setProvider(BouncyCastleProvider.PROVIDER_NAME));
                   }
                   throw new IllegalArgumentException(

                          "recipient for certificate not found");
               } else {
                   IESEngine engine = new IESEngine(new ECDHBasicAgreement(),
                           new KDF2BytesGenerator(new SHA256Digest()),
                           new HMac(new SHA256Digest()),
                           new PaddedBufferedBlockCipher(new CBCBlockCipher(

                                  new AESEngine())));
                   IESCipher cipher = new IESCipher(engine);
                   IESParameterSpec parameterSpec = new IESParameterSpec(null, null,
                           256, 256, null, false);
                   cipher.engineInit(Cipher.DECRYPT_MODE, privateKey,
                           parameterSpec, new SecureRandom());
                   decryptData = cipher.engineDoFinal(data, 0, data.length);

               }
           }catch (Exception ignore){Log.e(TAG, ignore.toString());}
        }
        return decryptData;

    }
    public static byte[] signData(
```

```
            byte[] data,
            X509Certificate signingCertificate,
            PrivateKey signingKey) {

        byte[] signedMessage = null;
        List<X509Certificate> certList = new ArrayList<X509Certificate>();
        CMSTypedData cmsData= new CMSProcessableByteArray(data);
        certList.add(signingCertificate);
        try {
            Store certs = new JcaCertStore(certList);
            CMSSignedDataGenerator cmsGenerator = new CMSSignedDataGenerator();
            ContentSigner contentSigner
                    = new JcaContentSignerBuilder(
                            signingCertificate.getPublicKey().getAlgorithm().equals("RSA") ?
                    "SHA256withRSA": "SHA256withECDSA")
                    .setProvider(BouncyCastleProvider.PROVIDER_NAME)
                    .build(signingKey);
            cmsGenerator.addSignerInfoGenerator(new JcaSignerInfoGeneratorBuilder(
                    new JcaDigestCalculatorProviderBuilder()
                            .build()).build(contentSigner, signingCertificate));
            cmsGenerator.addCertificates(certs);

            CMSSignedData cms = cmsGenerator.generate(cmsData, true);
            signedMessage = cms.getEncoded();
        } catch (Exception ignore) {}
        return signedMessage;
    }
    public static boolean verifySignedObject(byte[] cmsSignedData )
    {
        boolean state = false;
        try {
            CMSSignedData signedData = new CMSSignedData(cmsSignedData);
            Store certStore = signedData.getCertificates();
            SignerInformationStore signers = signedData.getSignerInfos();
            Collection c = signers.getSigners();
            Iterator it = c.iterator();

            while (it.hasNext()) {
                SignerInformation
                        signer = (SignerInformation) it.next();
                Collection certCollection = certStore.getMatches(signer.getSID());
                Iterator certIt = certCollection.iterator();
                X509CertificateHolder cert = (X509CertificateHolder) certIt.next();
                AttributeTable      attrs = signer.getUnsignedAttributes();
```

```
            if (!signer.verify(new JcaSimpleSignerInfoVerifierBuilder().setProvider(
                    BouncyCastleProvider.PROVIDER_NAME).build(cert))) {
                state = false;
                break;
            } else
                state = true;
            if (state  && attrs != null) {
                Attribute           att = attrs.get(
                   PKCSObjectIdentifiers.id_aa_signatureTimeStampToken);
                ASN1Encodable dob = att.getAttrValues().getObjectAt(0);
                try {
                    byte[] encodedTsp = dob.toASN1Primitive().getEncoded();
                    CMSSignedData cms = new CMSSignedData(encodedTsp);
                    TimeStampToken result = new TimeStampToken(cms);
                    Store storeTt = result.getCertificates();
                    Collection collTt = storeTt.getMatches(result.getSID());
                    Iterator certIt2 = collTt.iterator();
                    X509CertificateHolder cert2 = (X509CertificateHolder)certIt2.next();
                    result.validate(new
                            JcaSimpleSignerInfoVerifierBuilder().setProvider(
                                    BouncyCastleProvider.PROVIDER_NAME).build(cert2));
                }catch( Exception ignore) {
                    state=false;break;}
            }
        }
    } catch (Exception ignore){Log.w(TAG, ignore.toString());}
    return state;
}
public static byte[] getCMSData( byte[] fullCMS) {
    byte[] data = null;
    try {
        CMSSignedData signedData = new CMSSignedData(fullCMS);
        CMSProcessable sc = signedData.getSignedContent();
         data = (byte[]) sc.getContent();
    } catch (Exception ignore){}
    return data;
}
public static X509CertificateHolder getSignerCertFromCMS( byte[] cmsSignedData ) {
    X509CertificateHolder cert = null;
    try {
        CMSSignedData signedData = new CMSSignedData(cmsSignedData);
        Store certStore = signedData.getCertificates();
        SignerInformationStore signers = signedData.getSignerInfos();
        Collection c = signers.getSigners();
```

```
            Iterator it = c.iterator();
            while (it.hasNext()) {
                SignerInformation
                        signer = (SignerInformation) it.next();
                Collection
                        certCollection = certStore.getMatches(signer.getSID());
                Iterator
                        certIt = certCollection.iterator();
                cert = (X509CertificateHolder) certIt.next();
            }
        } catch (Exception ignore){Log.w(TAG, ignore.toString());}
        return cert;
    }
    private static RecipientInformation getSingleRecipient(CMSEnvelopedDataParser parser) {
        Collection recInfos = parser.getRecipientInfos().getRecipients();
        Iterator recipientIterator = recInfos.iterator();
        if (!recipientIterator.hasNext()) {
            throw new RuntimeException("Could not find recipient");
        }
        return (RecipientInformation) recipientIterator.next();
    }
    public static X509Certificate certFromHolder( X509CertificateHolder certificateHolder )
{
        X509Certificate certificate = null;
        try {
            certificate = new JcaX509CertificateConverter()
                    .setProvider(BouncyCastleProvider.PROVIDER_NAME)
                    .getCertificate(certificateHolder);
        } catch (Exception ignore){}
        return certificate;
    }
    public static Attribute createTSToken(byte[] data) {
        Attribute attr = null;
        try {
            MessageDigest digest = MessageDigest.getInstance("SHA256",
                    BouncyCastleProvider.PROVIDER_NAME);
            TimeStampResponse response = new TimeStampResponse(data);
            TimeStampToken timestampToken = response.getTimeStampToken();
            attr = new Attribute(PKCSObjectIdentifiers.id_aa_signatureTimeStampToken, new
DERSet(ASN1Primitive.fromByteArray(timestampToken.getEncoded())));
        }catch (Exception ignore){}
        return attr;
    }
    public static Collection<SignerInformation>  getSignerFromSignature(byte[] signedData )
```

```
    {
        CMSSignedData data;
        try {
            data = new CMSSignedData(signedData);
        }catch (Exception ignore) { return null;}
        return data.getSignerInfos().getSigners();

    }
    public static byte[] replaceSignSectionCMS(byte[] tsResponse, byte[] sign ) {
        byte[] newSign = null;
        try {
            CMSSignedData data = new CMSSignedData(sign);
            Collection<SignerInformation> signers = data.getSignerInfos().getSigners();
            SignerInformation signer = signers.iterator().next();
            ASN1EncodableVector timestampVector = new ASN1EncodableVector();
            Attribute token = CryptoUtils.createTSToken(tsResponse);
            timestampVector.add(token);
            AttributeTable at = new AttributeTable(timestampVector);
            signer = SignerInformation.replaceUnsignedAttributes(signer, at);
            signers.clear();
            signers.add(signer);
            CMSSignedData signedData = CMSSignedData.replaceSigners(data,
                    new SignerInformationStore(signers));
            newSign = signedData.getEncoded();
        }catch (Exception ignore) {}
        return newSign;
    }
    public static TimeStampRequest  tspRequest(byte[] contsha256 ) {
        TimeStampRequest resp = null;
        TimeStampRequestGenerator reqGen = new TimeStampRequestGenerator();
        reqGen.setCertReq(true);
        try {
            resp = reqGen.generate(TSPAlgorithms.SHA256, contsha256);
        }catch (Exception ignore) {}
        return resp;
    }
}
```