

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

|  |  |  |
| --- | --- | --- |
| Зав. кафедрой | | «КБИС» |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | О.А. Сафарьян |
| подпись | |  |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_ 2026 г. | |

**КУРСОВАЯ РАБОТА**

Тема: «РЕАЛИЗАЦИЯ ПО НА БАЗЕ ANDROID ДЛЯ ЗАЩИТЫ ЦИФРОВЫХ ЛОКАЛЬНЫХ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ ОТ КОМПРОМЕТАЦИИ»

Дисциплина: «Методы и средства криптографической защиты информации»

Специальность: 10.05.01 Компьютерная безопасность

Специализация: Математические методы защиты информации

Обозначение курсовой работы МСКЗИ.990000.000 Группа ВКБ43

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д. П. Ковалев

подпись, дата

Курсовая работа защищена с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_

Руководитель работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент, О. А. Сафарьян

подпись, дата

Ростов-на-Дону

2026



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ДГТУ)**

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

|  |  |  |
| --- | --- | --- |
|  | | |
| Зав. кафедрой | | «КБИС» |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | О.А. Сафарьян |
| подпись | |  |
| «\_\_\_» | \_\_\_\_\_\_\_\_\_\_\_\_\_ 2026 г. | |

**ЗАДАНИЕ**

на выполнение курсовой работы

Тема «РЕАЛИЗАЦИЯ ПО НА БАЗЕ ANDROID ДЛЯ ЗАЩИТЫ ЦИФРОВЫХ ЛОКАЛЬНЫХ ПОЛЬЗОВАТЕЛЬСКИХ ДАННЫХ ОТ КОМПРОМЕТАЦИИ»

Дисциплина: Методы и средства криптографической защиты информации

Обучающийся: Ковалев Данил Петрович

Обозначение курсовой работы МСКЗИ.990000.000 Группа: ВКБ43

Срок представления работы к защите «\_\_\_» \_\_\_\_\_\_\_\_\_ 2026 г.

Исходные данные для курсовой работы:

1. Задание на выполнение курсовой работы;

2. Ильин М.Е., Калинина Т.И., Пржегорлинский В.Н. Криптографическая защита информации в объектах информационной инфраструктуры.-Москва, 2019.-138 с.;

3. Федеральный закон № 149-ФЗ «Об информации, информационных технологиях и о защите информации»;

|  |
| --- |
| **Содержание пояснительной записки** |
| Введение:  В данном разделе будет определена актуальность работы, её цель и задачи. |
| Разделы основной части: |
| 1. В разделе "Теоретический основы криптографии" дается полное описание основ криптографии, симметричного и асимметричного шифрования. 2. В разделе "Анализ симметричных криптографических алгоритмов" дается полное описание выбранных алгоритмов для ПО, их история и сравнение между собой. 3. В разделе “Разработка программного средства” дается обоснование выбора языка программирования и среды разработки, описывается архитектура программного средства, используемые паттерны проектирования, а также показывается то, как выглядит программное средство. 4. В разделе "Демонстрация программного средства" демонстрируется работоспособность программного средства, входные и выходные данные. |
| Заключение:  В рамках данной курсовой работы было разработано программное средство, реализующее алгоритмы 3DES, ChaCha20 Poly1305, AES-GCM для шифрования и расшифрования данных.   |  |  |  |  | | --- | --- | --- | --- | | Руководитель работы | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | О. А. Сафарьян | |  |  |  | | Задание принял к исполнению | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  подпись, дата | Д. П. Ковалев | |

**Содержание**

[Введение 5](#_Toc195634689)

[1 Анализ алгоритмов сжатия 6](#_Toc195634690)

[1.1 Утилита gzip 6](#_Toc195634691)

[1.2 Алгоритм FastLZ 8](#_Toc195634693)

[1.2.1 Описание работы FastLZ Level 1 8](#_Toc195634699)

[1.2.2 Описание работы FastLZ Level 2 10](#_Toc195634704)

[1.3 Утилита Pigz 12](#_Toc195634705)

[2 Программная реализация архиватора данных 16](#_Toc195634706)

[2.1 Выбор языка программирования, инструментов и архитектуры 16](#_Toc195634707)

[2.2 Основные этапы работы программы 17](#_Toc195634708)

[2.3 Реализация алгоритмов компрессии и декомпрессии 18](#_Toc195634709)

[3 Работоспособность программного средства 24](#_Toc195634710)

[4 Сравнительный анализ алгоритмов 25](#_Toc195634711)

[Заключение 26](#_Toc195634712)

[Перечень используемых информационных ресурсов 27](#_Toc195634713)

[Приложение А Листинг кода 28](#_Toc195634714)

Введение

Благодаря бурному развитию информационных технологий, современный мир невозможно представить с их отсутствием. Цифровизация стала неотъемлемой частью повседневной жизни, начиная от бизнеса и науки, заканчивая государственным управлением. Но не стоит забывать о том, что стремительное развитие Интернет-сервисов сопровождается увеличением киберугроз, таких как: атаки на конфиденциальность, целостность и доступность информации. Безопасность информации, её защищенная передача – одна из ключевых проблем современного мира. Для защиты данных разработано огромное множество различных механизмов, но основным рычагом остается криптография.

Криптография, зародившаяся еще в древности как искусство тайнописи – шифр Цезаря, скитала спратанцев – сегодня превратились в фундаментальную науку цифровой безопасности [1]. Если раньше она защищала лишь секретные послания полководцев и дипломатов, то в современную эпоху криптографические методы обеспечивают безопасность и конфиденциальность электронного взаимодействия, а также подлинность документов. От простых шифров прошлого до сложных алгоритмов сегодняшнего дня – криптография остается незаменимым инструментом защиты информации, сохраняя свою актуальность.

В зависимости от решаемых задач современная криптография использует симметричные и ассиметричные методы шифрования. Для защиты локально хранимых данных оптимальным решением являются симметричные алгоритмы. Данный выбор обоснован рядом факторов: во-первых, симметричное шифрование обеспечивает высокую скорость обработки данных, что критично для приложений реального времени. Во-вторых, оно требует меньше вычислительных ресурсов, что важно для мобильных устройств с ограниченной производительностью. В-третьих, при локальном хранении отсутствует необходимость в безопасном обмене ключами через небезопасные каналы связи, что является основной сложностью ассиметричной криптографии. Таким образом, симметричные алгоритмы (AES, ChaCha20 Poly 1305, 3DES) представляют собой оптимальный баланс между уровнем безопасности, производительностью и эффективностью использования ресурсов для задач локального шифрования.

Объектом данной курсовой работы являются процессы при разработке симметричных алгоритмов AES-GCM, DES-EDE3, ChaCha20 Poly1305.

Таким образом целью курсовой работы является исследование алгоритмов, анализ их криптографических свойств и практическая их реализация на примере Android приложения.

Цель определила следующие задачи:

1. Анализ предметной области.
2. Классификация симметричных алгоритмов и сравнительный анализ AES-GCM, DES-EDE3, ChaCha20 Poly1305.
3. Проведение обзора математических и криптографических принципов, лежащих в основе алгоритмов AES-GCM, DES-EDE3, ChaCha20 Poly1305.
4. Реализация программного средства.
5. Проведение проверки работоспособности программного средства.
6. Теоретические основы криптографии
   1. Основные понятия криптографии

Криптография является той наукой, которая посредством преобразования данных обеспечивает защиту информации на предмет ее конфиденциальности, целостности, подлинности и доступности. Криптография играет ключевую роль в обеспечении безопасности в современном цифровом мире и имеет тесную связь с математическими алгоритмами, которые помогают обеспечивать защиту от несанкционированного доступа.

К основным понятиям, используемых в криптографии, относятся:

Шифрование – процесс преобразования исходной (открытой) информации и в зашифрованную форму (шифртекст), недоступную для понимания без знания специальной информации (ключа) [2];

Дешифрование – процесс восстановления исходной информации из шифртекста с использованием соответствующего ключа [2];

Ключ – секретная или публичная (в зависимости от алгоритма), необходимая для выполнения операции шифрования, либо же дешифрования [2];

Открытый ключ – ключ, предназначенный для свободного распространения, объявлении на распределенном ресурсе и используемый для шифрования данных или проверки подписи [2];

Закрытый ключ – секретный ключ, информация, предназначенная для дешифрования данных или создания подписи [2].

Криптография основывается на алгоритмах, важными характеристиками которых должны являться стойкость к атакам криптоаналитиков, эффективность и корректность. Криптографические алгоритмы можно разделить на два раздела: симметричные и ассиметричные. При переводе открытого текста в шифртекст следует использовать либо один секретный ключ, либо пару – закрытый и открытый соответственно. Решение использования как раз и будет определять вид алгоритма – симметричный\ассиметричный.

* 1. Симметричные алгоритмы шифрования

Симметричное шифрования базируется на использовании одного и того же ключа шифрования. С его помощью информация не только зашифровывается, но и расшифровывается. В такой системе каждая из переписывавшихся сторон должна иметь копию общего секретного ключа, что создает сложнейшую проблему управления ключами [3]. Для успешного использования симметричных криптосистем партнеры должны обдумать и договориться о том, каким образом последует передача ключа. Это может быть как физическое распределение (личная встреча), так и распределение с помощью протоколов.

Протоколы с секретным ключом отличаются тем, что секретные ключи распределены между пользователем и неким центром, которым обычно называют центром доверия и используют для генерирования ключей и обмена между любыми пользователями всякий раз, когда в этом возникает необходимость. При протоколах с открытым ключом пользователи же предварительно могут договориться об общем секретном ключе [3]. Помимо этого пользователь, который будет зашифровывать информацию, может поместить ключ на распределенном ресурсе, доступ к которому будет у ограниченного круга лиц. В свою очередь партнер, расшифровывающий сообщение, будет иметь доступ к ограниченному ресурсу для взимания секретного ключа и выполнение действия алгоритма расшифрования.

Симметричные алгоритмы шифрования играют важную роль в обеспечении безопасности данных. Выбор конкретного алгоритма зависит от требований к безопасности, производительности и специфики применения. Сейчас же алгоритмы, оперирующие симметричным ключом, относятся к блочным или поточным шифрам. Яркими представителями и наиболее значимыми алгоритмами симметричного шифрования являются такие алгоритмы, как: DES-EDE3, AES-GCM и ChaCha 20 Poly 1305.

DES – был одним из первых широко используемых блочных симметричных стандартов шифрования, разработанных в 1970-х годах. В обработке информации он использует 56-битный ключ и обрабатывает блоки длиной 64 бита. Несмотря на свою историческую значимость, DES считается устаревшим и небезопасным из-за короткой длины ключа, что делает его уязвимым для атак методом перебора [4]. В ответ на ограничения DES был разработан алгоритм Triple DES (3DES), использующий схему DES-EDE3 (Encrypt-Decrypt-Encrypt с тремя независимыми ключами), где данные последовательно шифруются первичным ключом, дешифруются вторым и шифруется третьим. Это позволяет значительно увеличить криптостойкость по сравнению с оригинальным DES, обеспечивая эффективную длину ключа до 168 бит (при использовании трех независимых ключей) или 112 бит (при использовании двух ключей, где первый применяется дважды).

AES – современный симметричный стандарт, один из самых распространенных. Он был принят в качестве стандарта шифрования США в 2001 году. Алгоритм работает с ключами 128, 192 и 256 бит, оперирует 128-битной последовательностью. AES обеспечивает высокий уровень безопасности и используется в различных приложениях, включая защиту данных в облачных сервисах и шифрование файлов [4].

ChaCha20-Poly1305 представляет собой схему аутентифицированного шифрования (AEAD – Authencated Encryption With Associated Data), разработанную Дэниелом Дж. Бернстайном и стандартизированную в RFC 8439. Алгоритм объединяет потоковый шифр ChaCha20 для шифрования данных и аутентификационный код Poly 1305 для обеспечения целостности и подлинности информации. ChaCha20 использует 256-битный ключ и 96-битный одноразовый номер (nonce), обеспечивая высокую криптостойкость. Преимуществом данного алгоритма является высокая производительность программных реализаций, особенно на устройствах без аппаратной поддержки AES, что делает его эффективным решением для мобильных приложений. Poly 1305 генерирует 128-битный тег аутентификации, который позволяет обнаруживать несанкционированные изменения данных, гарантируя свойство целостности зашифрованной информации.

По сравнению с ассиметричными алгоритмами, симметричные работают быстрее, что и является их преимуществом. Такие алгоритмы просты в реализации и требуют меньшей вычислительной мощности – высоко эффективны при обработке текстов переменной длины. Но остается главный недостаток – безопасная передача ключа между сторонами взаимодействия. Если ключ будет скомпрометирован, то под угрозой окажется безопасность общения. Развивая ключевую отрасль, также не стоит забывать о безопасном хранении ключей при работе с системами большого количества пользователей. Таким образом симметричные методы шифрования чувствительны к атакам – они уязвимы к атакам криптоаналитиков, которые основываются на анализе частоты символов текста. Понимание этих аспектов является ключевым для выбора подходящего метода шифрования в зависимости от конкретных требований к безопасности.

* 1. Ассиметричные алгоритмы шифрования

Как было рассмотрено ранее, в симметричной криптографии копия секретного ключа должна храниться у обоих собеседников – данный факт создает сложнейшую проблему в использовании симметричных алгоритмов, а также в вопросах аутентификации и распределения ключей. В 1976 году случился прорыв в криптографии, когда появилась революционная статья У. Диффи и М. Хеллмана, в которой вводились основные понятия криптографии с открытым ключом. У. Диффи и М. Хеллман предложили два фундаментальных изменения: персонализация и разделение ключей. Такая криптография стала называться ассиметричной. Ассиметричные алгоритмы используют пару ключей: открытый и закрытый, и именно это решило проблему симметричной криптографии. [5]

Ключевая особенность ассиметричного шифрования заключается в том, что в процессе взаимодействия партнеры не только могут не встречаться лично, они могут даже не подозревать о существовании друг друга до первого цифрового разговора. Как было сказано ранее, операции асимметричного подхода к шифрованию и дешифрованию оперируют ключевой парой: закрытым и открытыми ключами. При этом открытый ключ может быть размещен на распределенном ресурсе, доступ к которому есть у желающего наладить контакт – с целью воспользоваться этим ключом для зашифрования информации и отправки её владельцу ключа.

Но секретный ключ должен храниться владельцем в секрете, именно только он имеет возможность расшифровать принятое сообщение пользователя.

Безопасность ассиметричных алгоритмов зависит от секретности закрытых ключей. Каждый может послать владельцу секретную информацию, воспользовавшись открытым ключом, но лишь сам владелец в состоянии расшифровать сообщение. Такая ситуация происходит по причине того, что лишь у него есть соответствующий секретный ключ. Между ключами существует математическая связь, что и является причиной работоспособности таких криптосистем. Данная связь гарантирует, что взятый открытый ключ из распределенного ресурса никак не приведет к рассекречиванию и восстановлению закрытого ключа. Но при этом владение секретным ключом обеспечивает возможность расшифровать сообщение, зашифрованное открытым. [3]

Особенность использования открытых ключей ассиметричных систем позволяет решать фундаментальную проблему криптографии – безопасный обмен ключами в условиях незащищенных каналов связи. В отличие от симметричных систем, они исключают необходимость предварительной передачи секретного ключа. Что делает их незаменимыми для систем с повышенными требованиями к информационной безопасности. Среди множества ассиметричных криптографических алгоритмов в настоящее время наиболее актуальными и используемыми в обществе являются RSA, ECDSA и алгоритм Эль-Гамаля.

Алгоритм RSA является одним из первых ассиметричных криптосистем, основанный на сложности факторизации больших чисел. Данный алгоритм был разработан в 1977 году в Массачусетском технологическом институте США [4]. Его надежность и относительная простота реализации сделали его стандартом для защищенного обмена данными, цифровых подписей и шифрования ключей. В основе алгоритма RSA лежит вычислительная проблема разложения больших чисел на простые множители, т. е. факторизации простых больших чисел. Однако с ростом вычислительных мощностей требуемая длина ключей RSA постоянно увеличивается для поддержания высокого уровня безопасности, что снижает его эффективность в некоторых сценариях.

В 1992 году канадский криптограф Скотт Ванстоун предложил концепцию ECDSA – алгоритма с открытым ключом, который использовал математику эллиптических кривых [8]. Данный алгоритм предлагал сопоставимую с алгоритмом RSA стойкость, но преимущество заключалось в использовании значительно меньших размеров ключей, что являлось огромным плюсом в вопросе обработки ключей.

Предложенный Тахером Эль-Гамалем в 1985 году ассиметричный алгоритм использовался как для шифрования, так и для обеспечения аутентификации (цифровой подписи). В отличие от RSA алгоритм Эль-Гамаля, основанный на проблеме дискретного логарифмирования алгоритм, не был запатентован и, поэтому, стал более дешевой альтернативой RSA [6]. Система является вероятностной, что увеличивает стойкость алгоритма в случаях криптоанализа, и обеспечивает конфиденциальность с блокировкой посторонних лиц.

Ассиметричные криптосистемы находят широкое применение в современных информационных технологиях, обеспечивает защиту данных и безопасность коммуникаций. Они являются основой для безопасного обмена ключами в криптографических протоколах, используются в системах электронной цифровой подписи, защищенной электронной почте, аутентификации в банковских системах и защищенных мессенджерах. Как повествовалось выше, ассиметричное шифрование обеспечивает безопасный обмен данными без предварительной передачи секретного ключа, что делает такой вид шифрования незаменимым инструментом в современных распределенных системах и использовании открытых сетей.

4. Анализ симметричных криптографических алгоритмов
   1. Алгоритм Triple DES

### **История создания алгоритма Triple DES**

Несмотря на то, что алгоритм DES долгое время являлся стандартом шифрования и широко использовался в различных приложениях, к концу 1990-х годов его криптостойкость стала вызывать серьезные опасения. Уязвимость DES была обусловлена относительно короткой длиной ключа – всего 56 бит, что делало алгоритм уязвимым к атакам методом полного перебора. Уже в 1998 году специалистам из Electronic Frontier Foundation удалось взломать DES за 56 часов, используя специально разработанную аппаратуру [7].

Перед криптографическим сообществом встала задача создания более стойкого алгоритма, который бы сохранял совместимость с существующим оборудованием и программным обеспечением, использующим DES. Решение было найдено в применении принципа многократного шифрования – идеи, восходящей к работам Клода Шеннона о композиции шифров [8]. Вместо разработки принципиально нового алгоритма было предложено последовательно применять DES трижды с различными ключами, что позволяло значительно увеличить эффективную длину ключа и сохранять совместимость с существующей инфраструктурой.

В 1999 году Национальный институт стандартов и технологий США (NIST) официально стандартизировал алгоритм Triple DES в документе FIPS 46–3, определив его как временную замену DES до появления нового стандарта шифрования [9].

### **Описание криптосистемы Triple DES и сравнение с DES.**

Triple DES относится к классу симметричных блочных алгоритмов шифрования и представляет собой композицию трех операций DES, применяемых последовательно к блоку данных. В отличие от простого DES, который использует один 56-битный ключ, Triple DES может использовать два или три независимых ключа, что существенно увеличивает криптостойкость алгоритма.

Одной из самых популярных разновидностей Triple DES является DES-EDE3. Данная модификации обеспечивает обратную совместимость с DES при определенных условиях. Если же все три ключа одинаковы , то операция Triple DES эквивалента однократному DES, так как операции шифрования и расшифрования взаимно обратны. Это свойство позволяет использовать существующее оборудование DES для реализации Triple DES без существенных модификаций. Криптосистемы Triple DES и DES используют идентичные математические принципы и структуру раундов Фейстеля, но принципиально отличаются в организации применения ключей и количестве раундов шифрования. Прежде всего, функции шифрования в алгоритмах различны: в DES используется один ключ K длиной 56 бит, а шифрование блока данных m размером 64 бита осуществляется по формуле:

Рассмотрим формулу 1. – блок открытого текста (plaintext) размером 64 бита, – ключ шифрования 56 бит, – блок шифр-текста (ciphertext) размером 64 бита, – функция шифрования алгоритма DES. В отличие от этого, в схеме Triple DES используется последовательное применение трех операций DES с различными ключами. Существуют две основные разновидности алгоритмов: 3DES-112 (DES-EDE2), 3DES-168 (DES-EDE3).

3DES-112 использует два независимых ключа и , при этом первый ключ применяется дважды, можно представить в виде формулы:

В формуле 2 и – два независимых 56-битных ключа, – функция расшифрования алгоритма DES.

3DES-168 использует в свою очередь уже 3 независимых ключа , , – три независимых 56-битных ключа.

Важным преимуществом 3DES является его стойкость к атакам дифференциального и линейного криптоанализа, которые были разработаны специально для анализа DES. Тройное применение алгоритма значительно усложняет проведение атак, так как требует анализа композиции трех функций, что вычислительно намного сложнее, чем анализ одной функции. [10]

### **Описание алгоритма Triple DES**

Криптосистема Triple DES разделяется на три этапа: этап генерации ключей, этап шифрования и этап расшифрования.

При создании ключей для Triple DES необходимо сгенерировать два или три независимых 56-битных ключа в зависимости от выбранного варианта алгоритма. Для 3DES-112 используются ключи и , при этом общая эффективная длина ключа составляет 112 бит. Для 3DES-168 используется три независимых ключа , и , обеспечивая эффективную длину ключа 168 бит. Каждый ключ должен быть сгенерирован с использованием криптографически стойкого генератора случайных чисел для обеспечения безопасности алгоритма.

Следующим этапом является процесс шифрования переданного сообщения. Пусть следует зашифровать блок данных размером 64 бита. Операция шифрования для варианта DES-EDE3 (3DES-168) выполняется в три этапа. Первый этап – первичное шифрование блока данных с использованием ключа , формула представлена ниже:

Рассмотрим математические обозначения в формуле 3. – промежуточный результат шифрования после первого этапа, – функция шифрования алгоритма DES, – блок открытого текста, - ключ для первого этапа.

Второй этап заключается в расшифровке промежуточного результата с использованием ключа , формула представлена ниже:

Рассмотрим математические обозначения в формуле 4. – промежуточный результат шифрования после второго этапа, – функция дешифрования алгоритма DES, – ключ для второго этапа.

Теперь финальное шифрование происходит с использованием ключа , как представлено на формуле ниже:

Рассмотрим математические обозначения в формуле 5. – итоговый блок шифр-текста, – промежуточный результат шифрования после второго этапа, – ключ для третьего этапа.

Рассмотрим теперь процесс расшифрования. При получении зашифрованного сообщения с целью восстановления данных необходимо выполнить обратные операции в обратном порядке. Для DES-EDE3 расшифрование производится по следующим этапам.

Первый этап – расшифрование с использованием ключа , формула представлена ниже:

Рассмотрим математические обозначения в формуле 6. – промежуточный результат расшифрования после первого этапа, – функция дешифрования алгоритма DES, – шифр-текст, - ключ для первого раунда дешифрования.

Второй этап – расшифрование с использованием ключа , формула представлена ниже:

В формуле 7 – промежуточный результат расшифрования после первого этапа, - промежуточный результат после второго этапа, – функция шифрования.

Финальное расшифрование с использованием ключа представлено ниже:

В формуле 8 – итоговый восстановленный блок открытого текста. Для DES-EDE2 процесс аналогичен, но на первом этапе используется ключ , формула представлена ниже:

В практических приложениях Triple DES редко используется в режиме электронной книги (ECB), так как этот режим не обеспечивает достаточную криптостойкость для длинных сообщений. Вместо этого применяется режим сцепления блоков шифр-текста (CBC – Cipher Block Chaining), который обеспечивает зависимость каждого блока шифр-текста от всех предыдущих блоков открытого текста.

В режиме CBC перед шифрованием первого блока данных выполняется операция исключающего ИЛИ (XOR) с вектором инициализации (IV) – случайным 64-битным значением. Каждый последующий блок открытого текста перед шифрованием комбинируется с предыдущим блоком шифр-текста:

В формуле 10 – первый блок открытого текста, – i-й блок текста (i = 1, 2, 3, …), – первый блок шифр-текста, – i-й блок шифр-текста, – предыдущий (i-1)-й блок шифр-текста, – вектор инициализации (Initialization Vector) размером 64 бита, - операция побитового исключающего ИЛИ (XOR), – функция шифрования Triple DES в варианте DES-EDE3.

Процесс расшифрования происходит в обратном порядке, ниже представлена формула:

Режим СBC обеспечивает распространение ошибок: изменение одного бита в шифр-тексте приводит к искажению соответствующего и всех последующих блоков открытого текста при расшифровании, что позволяет обнаруживать попытки несанкционированного изменения данных.

Таким образом, Triple DES представляет собой надежное решение для обеспечения конфиденциальности данных, сочетающее проверенное временем структуру DES с увеличенной длиной ключа и стойкостью к современным криптоаналитическим атакам. Несмотря на появление более современных алгоритмов, таких как AES, Triple DES продолжает использоваться в различных приложениях, требующих высокой степени совместимости с существующими системами. [11]

* 1. Описание алгоритма AES

### **История создание криптосистемы AES**

Несмотря на то, что алгоритм Triple DES обеспечивал достаточную криптостойкость для большинства приложений, к концу 1990-х годов стало очевидно, что необходима разработка нового стандарта шифрования, который бы превосходил предшественников по производительности, безопасности и гибкости. Triple DES, будучи композицией трех операций DES, имел существенный недостаток – низкую скорость работы, что ограничивало его применение в современных высокопроизводительных системах. [11]

В 1997 году Национальный Институт стандартов и технологий США (NIST) объявил открытый конкурс на разработку Advanced Encryption Standard (AES) – нового симметричного блочного алгоритма шифрования, который должен был заменить устаревшие DES и Triple DES. Категории конкурса включали требования к безопасности, эффективности реализации на различных платформах, гибкости и простоте архитектуры. На конкурс было подано 15 кандидатов, среди которых были алгоритмы от ведущих криптографов и исследовательских групп со всего мира. [12]

После трехлетнего процесса анализа и тестирования, в октябре 2000 года NIST объявил победителем конкурса Rijndael, разработанный бельгийским криптографом Винсейтом Рейменом (Vincent Rijmen) и Йоаном Дейменом (Joan Daemen) из Католического университета Лёвена. Название алгоритма представляет собой комбинацию фамилий его создателей. В декабре 2001 года NIST официально стандартизировал AES в документе FIPS 197, определив его как федеральный стандарт шифрования США [13]. С тех пор AES стал наиболее широко используемым симметричным алгоритмом шифрования в мире, применяясь в различных областях от защиты данных в интернете до шифрования файловых систем.

### **Описание криптосистемы AES и сравнение его с предшественниками**

Advanced Encryption Standard (AES) относится к классу симметричных блочных алгоритмов шифрования и представляет собой итеративный алгоритм на основе структуры сети подстановок и перестановок (Substitution-Permutation Network, SPN). В отличие от DES и Triple DES, которые используют структуру сети Фейстеля, AES применяет более современную архитектуру, обеспечивающую лучшую параллелизуемость операций и повышенную стойкость к крипто аналитическим атакам.

Наиболее важной особенностью AES является его гибкость в выборе длины ключа: алгоритм поддерживает три варианта – AES-128, AES-192, AES-256, использующие ключи длиной 128, 192 и 256 бит соответственно. Количество раундов шифрования зависит от длины ключа: для AES-128 используется 10 раундов, для AES-192 – 12 раундов, а для AES-256 – 14 раундов. Такая структура обеспечивает оптимальный баланс между безопасностью и производительностью для различных применений. Криптосистемы AES и его предшественники (DES, Triple DES) принципиально отличаются как в архитектуре, так и в математических основах. Прежде всего, функции шифрования в алгоритмах различны: в DES используется один ключ K длиной 56 бит, а шифрование блока данных размером m 64 бита осуществляется по формуле 1. В отличие от этого, в схеме AES используется ключ K длиной 128, 192 и 256 бит, а шифрование блока данных m размером 128 бит осуществляется по формуле:

Рассмотрим математические обозначения, которые представлены в формуле 12. – блок открытого текста размером 128 бит, – ключ шифрования длиной 128, 192 или 256 бит, – блок шифр-текста (ciphertext) размером 128 бит, – количество раундов шифрования (10, 12 или 14 в зависимости от длины ключа), – функция шифрования алгоритма AES. Такая схема обеспечивает значительно большую криптостойкость по сравнению с DES и Triple DES. Эффективная длина ключа AES-128 составляет 128 бит, что эквивалентно 2^128 возможных ключей, что делает атаки методом полного перебора вычислительно cложными для текущих компьютеров. Для сравнения, DES с 56-битным ключом (2^56 возможных ключей) был взломан в 1998 году, а Triple DES с эффективностью ключа 112 бит (2^112 возможных ключей) хотя и остается стойким, но значительно уступает AES по производительности.

Важным преимуществом AES является его стойкость к известным крипто аналитическим атакам, включая дифференциальный и линейный криптоанализ, которые были эффективны против DES. Архитектура SPN с операциями SubBytes, ShiftRows, MixColumns и AddRoundKey обеспечивает высокую степень диффузии и перемешивания данных, что делает проведение таких атак вычислительно неосуществимым [14]. Кроме того, AES демонстрирует отличную производительность как на программных, так и на аппаратных реализациях, превосходя Triple DES по скорости в 5–10 раз.

### **Описание алгоритма AES**

Криптосистема AES разделяется на три этапа: этап генерации ключей и расширении ключа, этап шифрования и этап расшифрования.

При создании ключей для AES необходимо сгенерировать ключ длиной 128, 192 или 256 бит в зависимости от выбранного варианта алгоритма. Ключ должен быть сгенерирован с использованием криптографически стойкого генератора случайных чисел для обеспечения безопасности алгоритма. Особенностью AES является процедура расширения ключа (Key Expansion), в ходе которой исходный ключ K преобразуется в массив раундовых ключей (round keys) до , где (количество 32-битных слов в блоке), а – количество раундов. Каждый раундовый ключ используется в операции AddRoundKey соответствующего раунда. Процедура расширения ключа включает применение функции SubWord (применение S-box к каждому байту слова), функцию RotWord (циклический сдвиг байтов слова) и операцию XOR с константами раунда . [13]

Следующим этапом является процесс шифрования переданного сообщения. Пусть следует зашифрованный блок данных m размером 128 бит. Операция шифрования для AES выполняется в несколько этапов.

Начальный раунд: перед первым раундом выполняется операция AddRoundKey с использованием начального раундового ключа, как представлено в формуле ниже:

Рассмотрим математические обозначения в формуле 13. – текущее состояние блока данных (представленное как матрица 4x4 байтов), m – блок открытого текста, – начальный раундовый ключ, - операция побитового исключающего или.

Основные раунды: каждый из основных раундов (для AES-128 это раунды 1–9) включает четыре операции.

Этап 1: SubBytes – замена каждого байта состояния с использованием S-box (таблицы подстановки), формула представлена ниже:

В формуле 14 – это функция нелинейной подстановки, обеспечивающая стойкость к линейному и дифференциальному криптоанализу.

Этап 2: – циклический сдвиг строк матрицы состояния, формула представлена ниже:

В формуле 15 – функция перестановки, обеспечивающая диффузию данных по строкам матрицы.

Этап 3: – линейное преобразование каждого столбца матрицы состояния:

В формуле 16 – функция умножения каждого столбца на фиксированную матрицу в поле GF(2^8), обеспечивающая диффузию данных по столбцам матрицы.

Этап 4: – побитовое сложение состояния с раундовым ключом:

В формуле 17 – раундовый ключ для раунда , где принимает значения от 1 до .

Финальный раунд: последний раунд () – включает операции , и , но исключает , то есть можно представить в виде формулы:

В формуле 18 – финальный раундовый ключ. Результат выполнения финального раунда является шифр-текстом и далее может быть использован для безопасной передачи в различных каналах связи.

Процесс расшифрования заключается в том, что требуется проделать обратные операции в обратном порядке. Для AES расшифрование производится по следующим этапам.

Начальный раунд начинается с операции с финальным раундовым ключом, который использовался при шифровании, как представлено по формуле ниже:

В формуле 19 – это блок шифр-текста, – финальный раундовый ключ. Далее каждый из основных обратных раундов включает четыре операции в обратном порядке.

Этап 1: - обратный циклический сдвиг строк матрицы состояния, формула представлена ниже:

Этап 2: – обратная замена каждого байта состояния с использованием обратного , формула представлена ниже:

Этап 3: – побитовое сложение состояния с раундовым ключом, формула представлена ниже:

В формуле 22 – раундовый ключ для обратного раунда.

Этап 4: – обратное линейное преобразование каждого столбца матрицы состояния:

Конечный раунд завершается с начальным раундовым ключом:

В формуле 24 – итоговый восстановленный блок открытого текста. В практических приложениях AES редко используется в режиме электронной кодовой книги (ECB), так как этот режим не обеспечивает достаточную криптостойкость для длинных сообщений. Вместо этого широко применяется режим Galois/Counter Mode (GCM), который обеспечивает не только конфиденциальность данных, но и аутентификация, гарантируя целостность и подлинность зашифрованных данных.

Режим GCM представляет собой режим аутентифицированного шифрования с ассоциированным данными (AEAD – Authenticated Encryption with Associated Data). Он сочетает счетчик (Counter mode) для шифрования с аутентификацией на основе умножения в поле Галуа GF(2^128). В режиме GCM перед шифрованием генерируется вектор инициализации (IV) длиной 96 бит, который используется для генерации счетчика. [15]

Процесс шифрования в режиме GCM для блока данных выполняется по формуле:

В данной формуле – значение счетчика для -го блока, вычисляемое как (конкатенация и инкрементированного счетчика), – ключ шифрования, - -й блок шифрования, - -й блок открытого текста.

Параллельно с шифрованием вычисляется тег аутентификации (authencation tag) длиной 128 бит, который обеспечивает целостность данных:

В формуле 26 – хеш-подключ, полученный как (шифрование нулевого блока), – ассоциированные (незашифрованные) данные, – шифр-текст, – функция аутентификации на основе умножения в поле Галуа.

При расшифровании процесс происходит в обратном порядке, как представлено по формуле ниже:

Перед использованием расшифрованных данных проверяется тег аутентификации: если вычисленный тег не совпадает с переданным, данные считаются скомпрометированными и отклоняются. Это свойство делает GCM особенно ценным для приложений, требующих гарантий целостности данных, таких как защищенные протоколы связи и шифрование файловых систем.

Режим GCM обеспечивает высокую производительность благодаря возможности параллельной обработки блоков данных и эффективной реализации операций в поле Галуа на современных процессорах. Кроме того, GCM не требует дополнительного механизма заполнения (padding), так как счетчик позволяет обрабатывать данные произвольной длины. [15]

Таким образом, AES представляет собой современное и надежное решение для обеспечения конфиденциальности и целостности данных, сочетающее проверенную криптостойкость с высокой производительностью и гибкостью применения. Благодаря своей архитектуре и широкой поддержке в различных платформах протоколах, AES стал де-факто стандартом шифрования в современной криптографии.

* 1. Криптосистема ChaCha20-Poly1305

### **История создания криптосистемы ChaCha20-Poly1305**

Несмотря на то, что алгоритм AES стал доминирующим стандартом симметричного шифрования в современной криптографии, к началу 2000-х годов возникла необходимость в разработке альтернативных алгоритмов, которые бы обеспечивали высокую производительность на платформах с ограниченными вычислительными ресурсами, а также повышенную стойкость к атакам по побочным каналам и возможность реализации с использованием исключительно базовых арифметических операций. [16]

В 2008 году американский криптограф Дэниел Бернштейн (Daniel J. Bernstein) разработал алгоритм алгоритм ChaCha – улучшенную версию своего алгоритма Salsa20, созданного в 2005 году. Название “ChaCha” отражает улучшенную диффузию данных благодаря модифицированной функции раунда. ChaCha20, использующий 20 раундов (вместо 8 или 12 в вариантах ChaCha8 и ChaCha12), стал наиболее широко применяемый версией алгоритма. [17]

Параллельно с обработкой данных потоковых шифров возникла потребность в безопасных режимах аутентифицированного шифрования. В 2005 году Бернштейн также разработал алгоритм Poly1305 – высокопроизводительный код аутентификации сообщений (MAC), основанный на универсальном хешировании в поле Галуа GF(2^130-5). Poly1305 демонстрирует отличные характеристики производительности благодаря использованию простых арифметических операций, не требующих таблиц перестановок или сложных математических преобразований.

Объединение ChaCha20 и Poly1305 в единую схему аутентифицированного шифрования было формализовано в документе RFC8439, опубликованном в 2018 году. Схема ChaCha20-Poly1305 представляет собой AEAD (Authenticated Encryption with Associated Data) конструкцию, которая обеспечивает как конфиденциальность, так и целостность данных единого криптографического примитива. Благодаря своим характеристикам безопасности и производительности, ChaCha20-Poly1305 получил широкое признание и был интегрирован в такие протоколы, как TLS 1.3, QUIC и WireGuard VPN. [18]

### **Описание криптосистемы ChaCha20-Poly1305 и сравнение с блочными шифрами**

ChaCha20-Poly1305 относится к классу симметричных алгоритмов аутентифицированного шифрования и представляет собой комбинацию потокового шифра ChaCha20 и функции аутентификации Poly1305. В отличие от блочных шифров, таких как AES и Triple DES, которые обрабатывают данные фиксированными блоками, ChaCha20 является потоковым шифром, генерирующим поток ключевого материала произвольный длины, который затем используется для шифрования данных побитовым сложением по модулю 2 (XOR).

Наиболее важной особенностью ChaCha20-Poly1305 является его архитектурная простота и высокая производительность на различных платформах, особенно в устройствах с ограниченными ресурсами. В отличие от AES, который требует таблиц подстановок (S-box) и сложных операций в поле Галуа для MixColumns, ChaCha20 использует только простые арифметические операции: сложение по модулю 2^32, XOR и циклические сдвиги. Это делает алгоритм устойчивым к атакам по времени выполнения (timing attacks) и позволяет создавать эффективные программные реализации без аппаратной поддержки.

Криптосистемы ChaCha20-Poly1305 и AES принципиально отличаются в архитектуре, так и в математических основах. Прежде всего, функции шифрования в алгоритмах различны: в AES используется блочный шифр с размером блоков 128 бит, а шифрование блока данных m размером 128 бит осуществляется по формуле 12. В отличие от этого, в схеме ChaCha20-Poly1305 используется потоковый шифр, который генерирует ключевой поток произвольной длины. Шифрование данных m произвольной длины производится по формулам, которые представлены ниже:

В данном случае – блок открытого текста, – ключ шифрования длиной 256 бит, – одноразовое число длиной 96 бит, – счетчик блоков (32 бита), – ключевой поток, генерируемый ChaCha20, – шифр-текст, – ассоциированные (незашифрованные) данные, – одноразовый ключ для Poly1305, – тег аутентификации длиной 128 бит. Такая схема обеспечивает несколько преимуществ по сравнению с блочными шифрами. Во-первых, потоковые шифры не требуют механизма заполнения (padding), так как могут обрабатывать данные произвольной длины без необходимости выравнивания по границам блоков. Во-вторых, генерация ключевого потока может выполняться параллельно с шифрованием, что повышает производительность на современных многоядерных процессорах. В-третьих, архитектура ChaCha20 обеспечивает высокую степень диффузии за счет 20 раундов преобразований, включающих чередование колоночных и диагональных раундов. Важным преимуществом ChaCha20-Poly1305 является его стойкость к атакам по времени выполнения и другим атакам по побочным каналам. Поскольку алгоритм использует только операции сложения, XOR и сдвига, время выполнения не зависит от значений от значений обрабатываемых данных, что затрудняет извлечение информации о ключе через анализ времени выполнения. Кроме того, Poly1305 обеспечивает криптографически стойкую аутентификацию с вероятностью подделки тега не более 2^-106 для сообщений длиной до 2^64 байт. [18]

По сравнению с AES-GCM, ChaCha20-Poly1305 демонстрирует сопоставимую или превосходящую производительность на платформах без аппаратной поддержки AES, что делает его предпочтительным выбором для мобильных устройств, встраиваемых систем и облачных приложений, требующих высокой пропускной способности. [18]

### **Описание алгоритма ChaCha20-Poly1305**

Криптосистема ChaCha20-Poly1305 разделяется на три основных компонента: генерацию ключевого потока ChaCha20, шифрование данных и аутентификацию с использованием Poly1305.

При создании ключей для ChaCha20-Poly1305 необходимо сгенерировать ключ длиной K длиной 256 бит (32 байта). Ключ должен быть сгенерирован с использованием криптографически стойкого генератора случайных чисел для обеспечения безопасности алгоритма. Для каждого сеанса шифрования также требуется одноразовое число (nonce) длиной 96 бит (12 байт), которое должно быть уникальным для каждого сообщения, зашифрованного одним и тем же ключом.

ChaCha20 использует внутреннее состояние (state) размером 512 бит (64 байта), представленное как массив из 16 слов по 32 бит каждое. Состояние инициализируется следующим образом:

Здесь – константа из четырех слов, определенная в специфике RFC8439, – ключ шифрования, – счетчик блоков (начинается с 0 или 1 в зависимости от использования), – одноразовое число.

Процесс генерации ключа потока в ChaCha20 основан на генерации функции раунда, которая применяется 20 раз (10 колоночных, чередующихся с 10 диагональными раундами). Основной операцией функции раунда является четвертый раунд (quarter round), который преобразует четыре 32-битных слова состояния.

Четверной раунд (quarter round) для слов a, b, c, d определяется следующими операциями:

Колоночный раунд применяет четверной раунд к столбцам матрицы состояния 4x4. Можно представить следующим образом в виде формул:

Диагональный раунд применяет четверной раунд к диагоналям матрицы состояния:

После 20 раундов (10 колоночных, 10 диагональных) к рабочему состоянию добавляется исходное состояние, как представлено на формуле ниже:

В формуле 55 – это выходной блок ключевого потока длиной 512 бит (64 байта), – состояние после 20 раундов, – исходное состояние.

Следующим этапом является процессом шифрования переданного сообщения. Пусть следует зашифровать данные произвольной длины с использованием ключа и .

Первый этап: генерация одноразового ключа для Poly1305, формула представлена ниже:

В формуле 56 – ключ для Poly1305 длиной 256 бит (32 байта), ChaCha20\_Block – функция генерации блока ChaCha20, [0..31] – первые 32 байта выходного блока.

Второй этап: генерация ключевого потока и шифрование данных, можно представить в виде формулы:

В случае формулы 58 keyStream – это ключевой поток, генерируемый ChaCha20, начиная со счетчика 1, – это шифр-текст.

Третий этап: генерация тега аутентификации. Можно выразить с помощью формул, как представлено ниже:

В данном случае – ассоциированные (незашифрованные) данные, и – длины и шифр-текста, представленные как 64-битные числа в порядке little-endian, - операция конкатенации, – тег аутентификации длиной 128 бит (16 байт). Результатом шифрования является пара (, ), которая далее может быть использована для безопасной передачи в различных каналах связи.

При расшифровании в начале производится проверка тега аутентификации.

Этап 1: генерация одноразового ключа для Poly1305, как и для шифрования по формуле 56.

Этап 2: проверка тега аутентификации, используя формулы 59, 60. Если теги при шифровании и расшифровании не совпадают, то процесс расшифрования прерывается, и данные считаются скомпрометированными. Это свойство обеспечивает защиту от несанкционированного изменения данных и гарантирует целостность сообщения.

Этап 3: расшифрование данных, данный этап выполняется только при успешной проверке тега. Формула для вычисления представлена под номером 57. Для восстановления текста используется следующая формула, которая представлена ниже:

В данном случае – итоговый восстановленный открытый текст.

Рассмотрим более подробно, как работает алгоритм аутентификации Poly1305. Poly1305 представляет собой универсальную функцию хеширования, которая генерирует 128-битный тег аутентификации для сообщения произвольной длины. Алгоритм работает в поле Галуа GF(2^130-5) и использует одноразовый ключ длиной 256 бит.

В начале происходит инициализация, ключ разделяется на две части:

В данном случае используется для хеширования, а – для финального маскирования.

При процессе хеширования сообщение разбивается на блоки по 16 байт. Для каждого выполняется:

В данном случае – это аккумулятор, инициализированный нулем, – текущий блок сообщения (с добавлением байта 0x01 в конце последнего блока), - умножение в поле .

После обработки всех блоков выполняется финальное преобразование:

В данном случае – итоговый 128-битный тег аутентификации. Простота операций Poly1305 (сложение, умножение по модулю и XOR) обеспечивает высокую производительность, и устойчивость к атакам по времени выполнения [18].

Режим аутентифицированного шифрования с ассоциированными данными (AEAD) ChaCha20-Poly1305 обеспечивает одновременную защиту конфиденциальности и целостности данных. Ключевой особенностью является то, что ассоциированные данные (AAD) остаются в открытом виде, но участвуют в вычислении тега аутентификации, что позволяет получателю проверить подлинность как зашифрованных, так и открытых частей сообщения. Это свойство особенно ценно в сетевых протоколах, где заголовки пакетов должны оставаться открытыми для маршрутизации, но и их подлинность должна быть проверена вместе с зашифрованным содержимым. Примером такого применения является протокол TLS 1.3, где ChaCha20-Poly1305 используется в качестве одного из рекомендуемых AEAD алгоритмов. [18]

Режим AEAD обеспечивает защиту от атак повторного использования (replay attacks), так как nonce должен быть уникальным для каждого сообщения. При правильном использовании счетчика или генерации случайных nonce для каждого сообщения, вероятность коллизии после остается пренебрежимо малой. Таким образом, ChaCha20-Poly1305 представляет собой современное и эффективное решение для аутентифицированного шифрования, сочетающее высокую производительность с криптографической стойкостью и устойчивостью к атакам по побочным каналам. Благодаря своей простоте реализации и отличным характеристикам безопасности, ChaCha20-Poly1305 стал одним из наиболее широко используемых AEAD алгоритмов в современных криптографических протоколах. [18]

1. Разработка программного средства

## Обоснование выбора языка и среды разработки

Для разработки программного средства был выбран высокоуровневый язык программирования Kotlin. На данный момент это является официальным языком для разработки под платформу Android. Написание программного кода на данным языке программирования осуществляется на бесплатной основе, любой желающий может начать разрабатывать программу. Язык включает в себя большой набор библиотек, классический Си-подобный синтаксис.

Преимуществом языка заключается в его характеристике: Kotlin является статически типизированным языком, компилируемым в байт-код Java Virtual Machine (JVM) или в нативный код для Android. Условие компиляции в JVM байт-код позволяет коду работать на любой платформе, где установлена виртуальная машина Java, - данная особенность является очень полезной для обеспечения кроссплатформенности приложений. Кроме того, Kotlin обеспечивает полную совместимость с Java, что позволяет использовать существующие Java-библиотеки и постепенно мигрировать существующий код.

Kotlin идеально подходит для поставленной цели задачи работы – создания графического Android-приложения для шифрования и расшифрования текстовых данных с использованием криптографических алгоритмов.

Для реализации программного средства была выбрана интегрированная среда разработки Android Studio. Android Studio является официальной средой разработки для создания Android-приложений, основанной на Intellij IDEA от Jetbrains. На этой среде присутствует все необходимое для написания, тестирования и отладки кода Android-приложений: Android-SDK, эмуляторы устройств, профилировщики производительности, инструменты для анализа памяти и сети.

Одной из преимущественных черт выбора Android Studio является наличие возможности автодополнения кода на основе анализа контекста и доступных API. Автодополнение является очень удобной функцией, которая помогает не только сокращать время написание программного продукта, но и изучать новые методы и классы Android SDK, которые можно использовать по отношению к существующим в листинге структурам. Пронумерованные строки, возможность сворачивания написанных блоков и условных конструкций помогает программисту разделять данные и легче читать написанное. Среда разработки совместима с Git и имеет все нужные инструменты для работы с системами контроля версий, что упрощает работу в команде и управление историей изменений проекта. Также стоит упомянуть кроссплатформенность среды – поддерживается работа как на Windows, так и на Linux и macOS, что обеспечивает единообразие процесса разработки независимо от используемой операционной системы.

Данная среда идеально подходит для написания данного проекта благодаря своей специализации на Android-разработке, удобному отладчику с возможностью подключения к реальным устройствам и эмуляторам, встроенному терминалу для работы с командной строкой, а также инструментам для анализа безопасности приложения, что критично для криптографических приложений. Android Studio идеально показывает себя в работе с выбранным языком программирования Kotlin, увеличивая производительность разработки и улучшая качество кода, обеспечивая при этом все необходимые инструменты для создания безопасных и функциональных мобильных приложений.

## Архитектура программного средства

Архитектура программного средства построена на принципах многослойной архитектуры с четким разделением ответственности между уровнями. В основе архитектуры лежит комбинация подходов Domain-Driven Design (DDD), Clean Architecture и паттерна Model-View-ViewModel (MVVM), что обеспечивает высокую модульность, тестируемость и поддерживаемость кода. Применение принципов Clean Architecture обеспечивает разделение программного средства на независимые слои с зависимостями, направленными от внешних слоев к внутренним. Внутренний слой представляет собой доменную модель (Domain Layer), содержащую бизнес-логику приложения, сущности и правила работы с криптографическими алгоритмами. Этот слой не зависит от внешних фреймворков и библиотек, что делает его полностью переносимым и легко тестируемым. Следующий слой – уровень приложения (Application Layer), который координирует работу доменных сервисов и содержит команды (Commands) и запросы (Queries) в соответствии с паттерном CQRS (Command Query Responsibility Segregation). Внешние слои – уровень представления (Presentation Layer) и уровень инфраструктуры (Infrastructure Layer) – зависят от внутренних слоев, но не наоборот, что обеспечивает инверсию зависимостей и возможность замены внешних компонентов без изменения бизнес-логики.

Использование подхода Domain-Driven Design позволяет моделировать предметную область криптографии через доменные сущности (EncryptionKey, EncryptedText, Text), доменные сервисы (AesEncryptionService, ChaCha20EncryptionService, TripleDesEncryptionService) и value objects (EncryptionAlgorithm, TextEncoding). Доменные сервисы инкапсулируют сложную логику работы с криптографическими алгоритмами, обеспечивая правильность выполнения операций шифрования и расшифрования независимо от способа их использования в приложении.

Паттерн Model-View-ViewModel (MVVM) реализован на уровне представления для обеспечения разделения логики отображения данных и бизнес-логики. ViewModel управляет состоянием пользовательского интерфейса и координирует работу с презентерами (Presenter), которые выполняют преобразование данных между уровнями приложения и представления. Композируемый пользовательский интерфейс (Jetpack Compose) взаимодействует с ViewModel через наблюдаемые потоки данных (StateFlow), что обеспечивает реактивное обновление интерфейса при изменении состояния.

Инфраструктурный слой реализует порты, определенные на уровне приложения, и включает в себя персистентное хранилище данных на основе Room Database для безопасного хранения ключей шифрования, использование Android Keystore для защиты критически важных данных, а также адаптеры для преобразования данных между доменными сущностями и моделями базы данных.

Такая архитектура обеспечивает высокую степень модульности: каждый слой имеет четко определенные обязанности и интерфейсы взаимодействия с другими слоями. Изменения в одном слое не требуют модификации других слоев при соблюдении контрактов взаимодействия. Кроме того, данная архитектура упрощает тестирование – доменная логика может быть протестирована независимо от Android-специфичных компонентов, а слои могут быть протестированы изолированно с использованием mock-объектов.

На рисунке 3.1 представлена блок-схема процесса генерации ключа для DES.

Изображение выглядит как текст, диаграмма, Параллельный, План

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.1 – Блок-схема генерации ключей Triple DES CBC

На рисунке 3.2 представлена блок-схема генерации ключей для алгоритма AES GCM.

Изображение выглядит как текст, диаграмма, снимок экрана, План

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.2 – Блок-схема генерации ключей AES GCM

На рисунке 3.3 представлена блок-схема процесса генерации ключей для алгоритма ChaCha20-Poly1305 соответственно.

Изображение выглядит как текст, диаграмма, зарисовка, рисунок

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.3 – Блок-схема генерации ключей ChaCha20-Poly1305

На рисунке 3.4 представлена блок-схема процесса шифрования AES GCM.

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.4 – Блок-схема шифрования AES

На рисунке 3.5 представлена блок-схема шифрования алгоритма 3DES CBC.

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.5 – Блок-схема шифрования алгоритма 3DES-CBC

На рисунке 3.6 представлена блок-схема шифрования для алгоритма ChaCha20-Poly1305.

Изображение выглядит как текст, чек, диаграмма, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.6 – Блок-схема шифрования ChaCha-Poly1305

На рисунке 3.7 представлена схема расшифрования Triple DES CBC.

Изображение выглядит как текст, диаграмма, План, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.7 – Блок-схема расшифрования Triple DES CBC

На рисунке 3.8 представлена блок-схема расшифрования алгоритма AES GCM.

Изображение выглядит как текст, диаграмма, План, Технический чертеж

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.8 – Блок-схема расшифрования AES GCM

На рисунке 3.9 представлена блок-схема расшифрования алгоритма ChaCha20-Poly1305.

Изображение выглядит как текст, диаграмма, Параллельный, чек

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 3.9 – Блок-схема расшифрования ChaCha20-Poly1305

1. Демонстрация программного средства

При скачивании apk файла пользователю предлагается его установить. На рисунке 4.1 представлен интерфейс установки apk файла на примере смартфона Vivo x300 pro на Android 16.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.1 – Установка apk файла

На рисунке 4.2 представлен пример экрана с установкой пароля при первой установке, если используется белая тема в Android системе.

Изображение выглядит как текст, снимок экрана, Мобильный телефон, мультимедиа

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.2 – Установка пароля при первом использовании приложения

При удачном сохранении пароля пользователя встречает стартовый экран с генерацией ключей. По умолчанию в приложении используется английский язык, интерфейс представлен на рисунке 4.3.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.3 – Интерфейс для генерации ключей с белой темой на английском

На рисунке 4.4 представлено меню, пользователь может выбрать криптографические алгоритмы для шифрования и расшифрования, а также настроить приложение по его вкусу. Например, выбрать темную тему или сменить язык.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.4 – Интерфейс меню в приложении “Криптограф”

На рисунке 4.5 представлен интерфейс для шифрования и расшифрования текста с использованием алгоритмов AES-128 GCM, AES-192 GCM, AES-256 GCM.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.5 – Интерфейс для выбора ключа шифрования и расшифрования

На рисунке 4.6 представлен интерфейс для шифрования и расшифрования файла при использовании белой темы и английского языка.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.6 – Интерфейс для шифрования и расшифрования файла с использованием AES GCM

Приложение также поддерживает русский язык, пример экрана представлен на рисунке 4.7.

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.7 – пример интерфейса на русском языке

На рисунке 4.8 представлен пример интерфейса с использованием темной темы и русского языка.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.8 – пример интерфейса с использованием темной темы

На рисунке 4.9 можно наблюдать процесс создания криптографических ключей. Пользователь должен выбрать алгоритм для шифрования. Снизу появится окошко с описанием созданного ключа, а именно ID, ключ для какого криптографического алгоритма, значение данного ключа. Также появится кнопка для удаления данного ключа.

Изображение выглядит как текст, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.9 – генерация ключа для AES 256 GCM

После генерации ключа пользователь может им воспользоваться для шифрования данных. На рисунке 4.10 представлен пример шифрования текстового сообщения “Hello world!”.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.10 – пример шифрования текстового сообщения

На рисунке 4.11 представлен пример расшифрования текстового сообщения с использованием алгоритма AES 256 GCM.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.11 – пример успешного расшифрования

Рассмотрим теперь пример шифрования музыкального файла – звук уведомления “Попкорн” с телефона Google Pixel 4xl. С помощью алгоритма AES 256 и ключа, который был представлен на рисунке 4.9, будет данный файл зашифрован. Результат представлен на рисунке 4.10.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.10 – пример шифрования файла

Проведем теперь попытку расшифровки нашего шифрованного файла. Интерфейс представлен на рисунке 4.11. Для остальных алгоритмов интерфейс не меняется.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дизайн

Содержимое, созданное искусственным интеллектом, может быть неверным.

Рисунок 4.11 – пример интерфейса при расшифровании файла

При изменении расширения файла в проводнике он будет корректно открываться любыми редакторами.

Заключение

В рамках данной работы было разработано программное средство – криптографическое приложение, реализующее функции шифрования и дешифрование данных с использованием современных алгоритмов, таких как AES, Triple DES, ChaCha20-Poly1305 и подходов Domain-Driven Design (DDD), Clean Architecture и SOLID принципов.

В ходе работы были решены следующие задачи:

1. Проведен анализ криптографических алгоритмов AES (режим GCM), Triple DES (режим CBC) и ChaCha20-Poly1305, включая математические принципы, этапы обработки данных и особенности реализации в контексте Android приложений.
2. Разработано программное средство для платформы Android с использованием данных криптографических алгоритмов.
3. Выявлены преимущества и недостатки каждого из алгоритмов с целью определения их пригодности для конкретных задач в различных областях применения.

Итогом данной курсовой работы стало создание рабочего и эффективного программного средства, способного удовлетворять современные требования к шифрованию и расшифрованию. Проведенное исследование и сравнительный анализ алгоритмов позволили не только выбрать оптимальные решения для конкретных задач, но и заложить основу для дальнейшего развития проекта и его интеграции с другими алгоритмы. Достигнутые результаты свидетельствуют о практической значимости выполненной работы и перспективности выбранного направления исследований.

Перечень использованных информационных ресурсов

1. Защита данных сквозь тысячелетия: что древние шифры могут рассказать о цифровой безопасности [Электронный ресурс]. – URL: https://www.computerra.ru/314068/zashhita-dannyh-skvoz-tysyacheletia-chto-drevnie-shifry-mogut-rasskazat-o-tsifrovoj-bezopasnosti/ (дата обращения 16.01.2026). – Текст : электронный.
2. Основные понятия криптографии [Электронный ресурс]. – URL: https://www.cryptocom.ru/articles/crypto.html (дата обращения 16.01.2026). - Текст : электронный.
3. Смарт Н. Криптография / Н. Смарт ; пер. с англ. С. А. Кулешова – Москва : Техносфера, 2005. – 528 с. – ISBN 5-94836-043-1. – Текст : непосредственный.
4. Ильин М. Е., Калинкина Т. И., Пржегорлинский В. Н. Криптографическая защита информации в объектах информационной инфраструктуры : учеб. для студ. учреждений сред. проф. образования / М. Е. Ильин, Т. И. Калинкина, В. Н. Пржегорлинский. – Москва : Издательский центр «Академия», 2019. – 138 с. – ISBN 978-5-4468-8717-0. – Текст : непосредственный.
5. Набиев, С. Р. Ассиметричные методы шифрования: RSA, ELGAMAL, ECC / С. Р. Набиев. – Казань : Научный центр “Олимп” 2018. – с. 474–475 с. – ISSN 2415–8402. – Текст : непосредственный.
6. Бадасян, Т. С. Схема Эль-Гамаля в криптографии / Т. С. Бадасян, С. К. Авагян. – Ереван : Олимп, 2020 – с. 21–24 – ISSN: 2312–8089. – Текст : непосредственный.
7. Electronic Frontier Foundation. Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design. – Sebastopol : O’Reilly Media, 1998. – 278 с. – ISBN 1-56592-520-3. – Текст : непосредственный.
8. Communication Theory of Secrecy Systems [Электронный ресурс]. – URL: https://www.cs.miami.edu/home/burt/learning/csc685.211/bstj28-4-656.pdf (дата обращения 16.01.2026). – Текст : электронный.
9. Data Encryption Standard [Электронный ресурс]. – URL: https://www.cs.miami.edu/home/burt/learning/csc685.211/bstj28-4-656.pdf (дата обращения 16.01.2026). – Текст : электронный.
10. Recommendation for the Triple Data Encryption Algorithm (TDEA) Block Cipher [Электронный ресурс]. – URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf (дата обращения 16.01.2026). – Текст : электронный.
11. The ESP Triple DES Transform [Электронный ресурс]. – URL: https://www.rfc-editor.org/rfc/rfc1851.html (дата обращения 16.01.2026). – Текст : электронный.
12. Advanced Encryption Standard (AES) [Электронный ресурс]. – URL: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf (дата обращения 17.01.2026)
13. Joan Daemen, Vincent Rijmen The Design of Rijndael – Berlin : Springer, 2020. - 29c. – ISBN 978-3-662-60769-5. – Текст : непосредственный.
14. Recommendation for Block Cipher Modes of Operation [Электронный ресурс]. – URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38a.pdf (дата обращения 17.01.2026).
15. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC [Электронный ресурс]. – URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-38d.pdf (дата обращения 17.01.2026).
16. The Use of Galois/Counter Mode (GCM) in IPSec ESP. RFC 4106 [Электронный ресурс]. – URL: https://www.rfc-editor.org/rfc/rfc4106.txt (дата обращения 20.01.2026)
17. ChaCha, a variant of Salsa 20 [Электронный ресурс]. – URL: https://cr.yp.to/chacha/chacha-20080120.pdf (дата обращения 20.01.2026)
18. The Transport Layer Security (TLS) Protocol Version 1.3 [Электронный ресурс]. – URL: https://datatracker.ietf.org/doc/html/rfc8446 (дата обращения 20.01.2026)

Приложение А Листинг кода

class AesEncryptionService : DomainService() {companion object {  
 private const val GCM\_TAG\_LENGTH = 16  
 private const val GCM\_IV\_LENGTH = 12  
 private const val BITS\_IN\_BYTE = 8  
  
 // AES key sizes in bytes  
 private const val AES\_128\_KEY\_SIZE\_BYTES = 16  
 private const val AES\_192\_KEY\_SIZE\_BYTES = 24  
 private const val AES\_256\_KEY\_SIZE\_BYTES = 32  
 }  
  
 private data class KeyContext(  
 val roundKeys: AesRoundKeys,  
 val numRounds: AesNumRounds,  
 )  
  
 private data class CiphertextAndTag(  
 val ciphertext: ByteArray,  
 val tag: ByteArray,  
 )   
  
 private val emptyAad = ByteArray(0)  
  
 fun encrypt(data: ByteArray, key: EncryptionKey): Result<EncryptedText> {  
 return try {  
 logger.info **{** "Starting AES encryption: algorithm=${key.algorithm}, dataSize=${data.size} bytes" **}** // Validate key  
 validateKey(key)// Create value objects for validation and type safety  
 val keySize = AesKeySize.create(key.algorithm).getOrElse **{** throw UnsupportedAlgorithmError(key.algorithm, "AesEncryptionService")  
 **}** val numRounds = AesNumRounds.create(key.algorithm).getOrElse **{** throw UnsupportedAlgorithmError(key.algorithm, "AesEncryptionService")  
 **}**  
 keySize.validateKeyBytes(key.value).getOrThrow()  
 val roundKeys = AesKeyExpansion.expandKey(key.value, numRounds)  
 val iv = ByteArray(GCM\_IV\_LENGTH)  
 SecureRandom().nextBytes(iv)val (ciphertext, tag) = AesGcmMode.encrypt(  
 AesGcmMode.EncryptParams(  
 plaintext = data,  
 iv = iv,  
 aad = emptyAad,  
 roundKeys = roundKeys,  
 numRounds = numRounds,  
 ),  
 )  
   
 val encryptedData = ByteArray(ciphertext.size + tag.size)  
 System.arraycopy(ciphertext, 0, encryptedData, 0, ciphertext.size)  
 System.arraycopy(tag, 0, encryptedData, ciphertext.size, tag.size)Result.success(  
 EncryptedText(  
 encryptedData = encryptedData,  
 algorithm = key.algorithm,  
 initializationVector = iv,  
 ),  
 )  
 } catch (e: UnsupportedAlgorithmError) {Result.failure(e)  
 } catch (e: DomainError) {Result.failure(e)  
 }  
 }  
  
   
 fun decrypt(encryptedText: EncryptedText, key: EncryptionKey): Result<ByteArray> {  
 return try {val iv = requireInitializationVector(encryptedText)  
 val encryptedData = encryptedText.encryptedData  
 requireEncryptedDataLength(encryptedData)  
  
 val keyContext = buildKeyContext(key)  
 val (ciphertext, tag) = splitCiphertextAndTag(encryptedData)  
 val decryptedData = decryptAndVerify(ciphertext, tag, iv, emptyAad, keyContext)Result.success(decryptedData)  
 } catch (e: UnsupportedAlgorithmError) {Result.failure(e)  
 } catch (e: DomainError) {Result.failure(e)  
 }  
 }  
 fun generateKey(algorithm: EncryptionAlgorithm): Result<EncryptionKey> {  
 return try {  
 logger.info **{** "Starting AES key generation: algorithm=$algorithm" **}** val keySizeBytes = when (algorithm) {  
 EncryptionAlgorithm.AES\_128 -> AES\_128\_KEY\_SIZE\_BYTES  
 EncryptionAlgorithm.AES\_192 -> AES\_192\_KEY\_SIZE\_BYTES  
 EncryptionAlgorithm.AES\_256 -> AES\_256\_KEY\_SIZE\_BYTES  
 else -> throw UnsupportedAlgorithmError(  
 algorithm,  
 "AesEncryptionService",  
 )  
 }val keyBytes = ByteArray(keySizeBytes)  
 SecureRandom().nextBytes(keyBytes)  
 logger.debug **{** "Random key bytes generated successfully" **}** logger.info **{** "AES key generation completed successfully: algorithm=$algorithm, " +  
 "keySize=${keyBytes.size} bytes (${keyBytes.size \* BITS\_IN\_BYTE} bits)"  
 **}** Result.success(  
 EncryptionKey(  
 value = keyBytes,  
 algorithm = algorithm,  
 ),  
 )  
 } catch (e: UnsupportedAlgorithmError) {Result.failure(e)  
 } catch (e: DomainError) {Result.failure(e)  
 }  
 }