Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Вычислительная техника
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой

_____ _____
подпись инициалы, фамилия
« ___» _____ 20 __ г.

БАКАЛАВРСКАЯ РАБОТА

09.03.01 Информатика и вычислительная техника
код и наименование направления

Разработка формата файлового контейнера для безопасного хранения данных с применением криптографической защиты на основе алгоритмов с открытым кодом
тема

Руководитель	_____	доцент, к.т.н.	Д.А. Швец
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____	КИ12-10Б	М.С. Милютин
	подпись, дата	номер группы	инициалы, фамилия
Нормоконтролер	_____	доцент, к.т.н.	В.И. Иванов
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2016

СОДЕРЖАНИЕ

Введение 5
1 Анализ существующих решений 7
1.1 Архиваторы 7
1.1.1 Формат tar 7
1.1.2 Формат ZIP 9
1.1.3 Архиватор WinZip 10
1.1.4 Архиватор 7-Zip 11
1.1.5 Архиватор WinRAR 11
1.2 Шифрование диска и виртуальные зашифрованные диски 11
1.2.1 Шифрование диска Windows Bitlocker 12
1.2.2 Шифрование диска OSX FileVault 12
1.2.3 Шифрование диска VeraCrypt 13
1.3 Медиаконтейнеры 13
1.3.1 Формат WAV 14
1.3.2 Формат MPEG-4 14
1.3.3 Формат Matroska 14
1.3.4 Формат TIFF 15
1.4 Другие контейнеры 15
1.4.1 Формат PE (EXE файлы) 15
1.4.2 Формат ISO-образ 16
1.4.3 Формат IMG 16
1.4.4 Базы данных 16
2 Шифрование 18
2.1 Симметричные алгоритмы шифрования 19
2.1.1 Алгоритм DES 20
2.1.2 Алгоритм ГОСТ 28147-89 22
2.1.3 Алгоритм AES 24
2.2 Режимы шифрования симметричного блочного шифрования . . . 25
2.2.1 Режим шифрования ECB 27
2.2.2 Режим шифрования CBC 28
2.2.3 Режим шифрования CFB 29

2.2.4 Режим шифрования OFB 30
2.3 Асимметричное шифрование 31
2.3.1 Алгоритм RSA 33
3 Описание разработанного файлового формата контейнера 34
3.1 Общее строение контейнера 34
3.1.1 Хранение файлов 34
3.1.2 Директория 35
3.1.3 Подпись 35
3.2 Выбор алгоритма шифрования 36
3.3 Выбор библиотеки для шифрования 36
3.3.1 Библиотека OpenSSL 37
3.3.2 Библиотека Crypto++ 38
3.3.3 Библиотека Botan 39
3.4 Выбор дополнительных библиотек 39
3.5 Библиотека CryptoPP 41
3.5.1 Заголовочный файл cryptopp/aes.h 41
3.5.2 Заголовочный файл cryptopp/files.h 41
3.5.3 Заголовочный файл cryptopp/filters.h 42
3.5.4 Заголовочный файл cryptopp/modes.h 42
3.5.5 Заголовочный файл cryptopp/osrng.h 43
3.5.6 Заголовочный файл cryptopp/rsa.h 43
3.6 Библиотека Boost 43
3.6.1 Заголовочный файл boost/filesystem.hpp 43
3.6.2 Заголовочный файл boost/serialization.hpp 44
4 Описание разработанной библиотеки 45
4.1 Заголовочный файл crypt.h 46
4.2 Заголовочный файл rsa.h 48
4.3 Заголовочный файл container.h 49
4.4 Сборка библиотеки с помощью CMake 51
4.5 Сборка библиотеки под операционную систему Android 51
4.5.1 Установка NDK 52
4.5.2 Компиляция разработанной библиотеки CryptoContainer . . 53
4.5.3 Компиляция библиотеки CryptoPP 54

4.5.4 Компиляция библиотек из набора Boost 55
4.5.5 Подключение библиотек к проекту 56
Заключение 59
Список использованных источников 60
ПРИЛОЖЕНИЕ А Исходный код программы 64

ВВЕДЕНИЕ

Человечество располагает огромным объемом информации, хранимым в цифровом виде, иногда эту информацию необходимо защитить от доступа посторонних лиц. Как раз в этом и помогает шифрование данных [1].

Криптография применяется как для защиты информации, обрабатываемой в ЭВМ или хранящейся в ней, так и для закрытия информации передаваемой по различным каналам связи. Криптографическое преобразование как метод защиты от несанкционированного доступа к данным имеет долгую историю. В данный момент разработано огромное количество различным методов шифрования, созданы теоретические и практические основы их применения. Большее число этих методов может быть успешно использовано и для шифрования данных.

Использование шифрования стало в настоящий момент особо актуально. Расширилось использование компьютерных сетей по которым передаются огромные объемы приватных данных, требующие надежной защиты. Также необходимо защитить уже хранимую информацию от несанкционированного доступа.

В данный момент существует множество программ, обеспечивающих шифрование данных, как проприетарных так и с открытым исходным кодом.

Проприетарное программное обеспечение является закрытым для пользователя, он не знает как оно работает, если там какие либо лазейки для третьих лиц или эта программа вообще не отправляет его данные третьим лицам. Ну и конечно же в большинстве своем проприетарное программное обеспечение является платным. Открытое программное обеспечение является бесплатным, но доступные решения являются громоздкими или не отвечают современным требованиям к защите информации.

Целью выпускной квалификационной работы создание формата файлового контейнера для безопасного хранения данных с применением криптографической защиты на основе алгоритмов с открытым кодом. Требовалось было разработать структуру контейнера для максимальной защиты данных и удобства работы с ним, отвечающий современным стандартам шифрования и являющийся полностью открытым. Так же необходимо разработать библио-

теку с простым API для работы с этим форматом данных.

Для достижения поставленной цели в работе ставились следующие задачи:

– анализ существующего программного обеспечения, выбор средств реализации криптографического файлового контейнера;

– разработка спецификации формата контейнера для безопасного хранения данных;

– создание API для использования возможностей разработанного файлового контейнера;

– реализация кроссплатформенной библиотеки для работы с криптографическим файловым контейнером;

– проверка работоспособности библиотеки.

Результатом выпускной квалификационной работы является кроссплатформенная программная библиотека с открытым исходным кодом для работы с криптографическим файловым контейнером. Разработанная программная библиотека обеспечивает следующие возможности:

– создание, открытие и редактирование файлового контейнера;

– выбор алгоритма и размер ключа для симметричного и ассиметричного шифрования.

1 Анализ существующих решений

Цель работы разработать защищенный формат контейнера, необходимо рассмотреть различные варианты как зашифрованных контейнеров или архивов, так и других различных видов контейнеров.

Каждый из них имеет свои плюсы и минусы, которые будут рассмотрены далее.

1.1 Архиваторы

Архиватор — это программа, осуществляющая упаковку одного и более файлов в архив или серию архивов для удобства переноса или хранения данных, а также для распаковки архивов сделанных этой программой.

Большинство современных архиваторов также реализуют сжатие, упакованных в архив данных и шифрование данных [2].

В большинстве случаев архиватор записывает файлы последовательно в один файл, при этом храня различные метаданные, такие как размер файла оригинального файла, оригинальное имя исходного файла. Более продвинутые архиваторы так-же записывают различные дополнительные системные метаданные (например время создания файла, атрибуты файла или список контроля доступа).

1.1.1 Формат tar

tar — формат архивации файлов и одноименная программная утилита для сборки различных файлов в один, для различных целей, таких как архивация и рассылка [3].

Первоначально была разработана для записи данных на устройства последовательной записи на которых не было файловых систем. Так же хранит в себе различную метаинформацию: название файла, временные штампы, контроль доступа, организацию папок.

Сама по себе программа не предоставляет возможность шифрования, но пользователь может зашифровать весь архив целиком или файлы по отдельности отдельными утилитами, что требует дополнительных затрат и знаний. При шифровании архива целиком отпадает возможность расшифровать от-

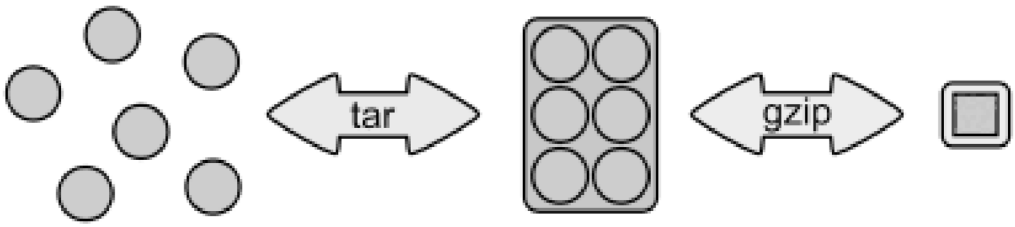

Рисунок 1 – Графическое представление сжатия файлов с один архив с помощью tar и архиватора gzip

дельные файлы, а если шифровать файлы по отдельности, то будет видна структура контейнера изнутри.

tar и gzip — широко принятая комбинация использования программного обеспечения для упаковки и сжатия файлов [4]. В соответствии с традициями UNIX-программирования, gzip выполняет только две функции: сжатие и распаковку одного файла; упаковка нескольких файлов в один архив невозможна [5], и именно из-за этого и используют формат tar. Утилита gzip для сжатия использует алгоритм Deflate [4], это алгоритм сжатия без потерь, использующий комбинацию алгоритмов LZ77 и Хаффмана.

tar и bzip2 — так-же популярный метод компрессии нескольких файлов в один контейнер [6]. bzip2 — бесплатная свободная утилита командной строки с открытым исходным кодом для сжатия данных, реализация алгоритма Барроуза-Уилера [7].

Стабильность и популярность компрессора росли в течение многих лет, и версия 1.0 была опубликована в конце 2000 года. bzip2 сжимает большинство файлов эффективнее, но медленнее, чем более традиционные утилиты gzip или zip. В этом отношении он похож на другие современные алгоритмы сжатия [6].

bzip2 выполняет сжатие данных с существенной нагрузкой на CPU (что обусловлено его математическим аппаратом). bzip2 применяют, если нет ограничений на время сжатия и на нагрузку на CPU, например, для разовой упаковки большого объёма данных.

1.1.2 Формат ZIP

ZIP — популярный формат архивации файлов который поддерживает сжатие данных без потерь [8]. Файл .ZIP может содержать от одного и более файлов или директорий, которые могут быть сжаты с помощью различных алгоритмов сжатия. Формат был разработан в 1989 году Филипом Кацем.

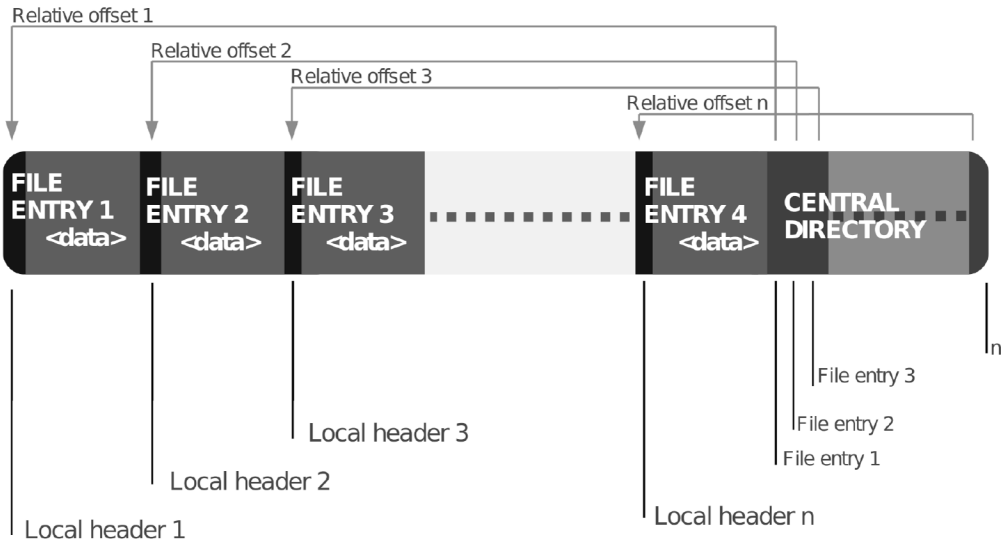

Рисунок 2 – Графическое представление структуры формата ZIP

В данный момент поддерживается различными операционными системами такими как Windows и Mac OS X. Изначально формат был проприетарным, но в данный момент он приобрел статус общественного достояния [9], то-есть позволяет свободно его использовать в коммерческом и свободном программном обеспечении.

Стоить заметить что не все виды архиваторов ZIP полностью придерживаются его спецификации и следовательно не всегда поддерживает работу с ним в полной мере.

Структура формата ZIP (рисунок 2) состоит из:

– Центрального каталога — в нем хранится расположение всех файлов относительно друг друга;

– Локальный заголовок — перед каждым файлов находится его описание с метаинформацией и различными данными описывающие этот файл;

– Сам файл — сам файл хранящийся в бинарном формате, не обяза-

тельно сжатый.

Изначально стандарт формата ZIP не описывал возможное шифрование, но позже появилось описание проприетарного алгоритма шифрования PKWARE, так-же позже была описано шифрование DES, Triple DES, RC2, RC4, в данный момент эти форматы уже считаются уязвимыми или устаревшими.

В последних версиях спецификации был описан способ шифрования каталога, который хранит расположение файлов в архиве и метаинформацию о нем, но заголовок файла и тип сжатия и шифрования является открытым и не замаскирован.

Так-же в новых версия спецификации описан способ защиты архива с использованием синхронного шифрования AES, но как писалось выше - не все реализации поддерживают его.

1.1.3 Архиватор WinZip

WinZip — условно-бесплатный файловый архиватор работающий на основе спецификации ZIP и собственного формата ZIPX [10]. В данный момент правами владеет компания Corel [11].

Поддерживает 128- и 256- битную длину ключей при шифровании с помощью синхронного шифрования AES. Полностью поддерживает юникод в названий файлов.

Программа была разработана как графический интерфейс к программе PKZIP (изначальная программа для архиваций и разархивации файлов ZIP), но в дальнейшем в нее начали добавляться новые оригинальные решения, которые невозможно было реализовать в оригинальном формате ZIP, поэтому они начали использовать свой формат ZIPX который немного расширял стандартную спецификацию.

Главными недостатками являются:

– не полная кроссплатформенность — нет реализации под *nix системы;

– проприетарность — данный программный продукт является платным и его исходный код закрыт и мы не знаем как он на самом деле шифрует данные и работает.

1.1.4 Архиватор 7-Zip

7-Zip — полностью бесплатная программа с открытым исходным кодом распространяющееся под лицензией GNU LGPLv2.1 [12]. В заявленных поддерживаемых операционных системах указаны Windows, Linux, OS X.

Использует свою реализация формата ZIP - 7z, но может читать и записывать другие форматы, в том числе ZIP. Имеет версию для командной строки под названием p7zip и так-же версию с графическим интерфейсом.

1.1.5 Архиватор WinRAR

WinRAR — проприетарное программное обеспечение, позволяющее создавать архивы в формате RAR и ZIP [13]. Официально поддерживается графический интерфейс только на операционной системе Windows, но командный интерфейс доступен в Linux, FreeBSD и OS X. Так-же есть свободная реализация UNRAR - библиотека позволяющая только разархивировать архивы формата RAR [14]. Очень популярна в странах СНГ.

Архивы RAR поддерживают шифрование 128- и 256-битными ключами с помощью шифрования AES. Главный его недостаток это проприетарность, алгоритм шифрования полностью закрыт и не известно как он шифрует данные и есть ли там какие либо лазейки для обхода защиты.

1.2 Шифрование диска и виртуальные зашифрованные диски

Шифрование диска — технология защиты информации, переводящая данные на диске в нечитаемую последовательность данных, которую нелегальный пользователь не сможет легко расшифровать. Для шифрования диска используется специальное программное или аппаратное обеспечение, которое шифрует каждый бит хранилища [15]. Такие решения обычно уже встроены в многие популярные операционные системы, например в Windows и OS X.

Основной проблемой использования встроенного в ОС шифрования это не прозрачность его перед пользователя, он не знает как происходит процесс шифрования, хранения данных и расшифровки. И самым главным минусом являются постоянно всплывающие уязвимости позволяющие обойти подоб-

ного вида защиту за считанные минуты [16].

Виртуальные зашифрованные диски — позволяет монтировать контейнер как логический диск с своей файловой системой и шифрует данные на лету. Они сложны в реализации, так-как необходимо написать или использовать готовый драйвер, например fuse, для доступа к виртуальному логическому диску [17].

Из самых больших плюсов стоит отметить полное встраивание в текущую операционную систему и пользователь работает с контейнером как с физическим диском. Такие решения так-же встроены в некоторые операционные системы, например Windows и OS X.

1.2.1 Шифрование диска Windows Bitlocker

Windows Bitlocker — проприетарная технология, являющаяся частью ОС Windows, позволяющая защитить данные путем полного шифрования диска. Поддерживает AES шифрование с 128-ти и 256-ти битными ключами.

Не рекомендуется к использованию. Постоянно появляющиеся новости о новых выявленных уязвимостях ставят под вопрос использование этой технологии [16], так-же данная разработка работает только под операционной системой Windows и является проприетарной.

1.2.2 Шифрование диска OSX FileVault

OSX FileVault — проприетарная система шифрования встроенная в операционную систему Mac OS X [18].

Полностью шифрует логический диск при помощи режима AES-XTS шифрования AES с длиной ключа 256 бит. Ключ генерируется из пароля пользователя операционной системы. Пользователь так-же может создавать шифрованные виртуальные диски с помощью встроенного программного обеспечения.

Главным недостатком является опять же то, что технология полностью закрыта и проприетарна, работает только на операционной системе OSX и ставит под сомнение свое использование.

1.2.3 Шифрование диска VeraCrypt

VeraCrypt — бесплатное программное обеспечение для шифрования данных на лету, подключая виртуальный зашифрованный раздел диска к системе. VeraCrypt является бесплатным и открытым проектом, который начал свою разработку как ответвление от закрытого проекта TrueCrypt.

VeraCrypt может использовать следующие алгоритмы шифрования: AES, Serpent и Twofish. Дополнительно доступны 5 комбинаций этих алгоритмов: AES-Twofish, AES-Twofish-Serpent, Serpent-AES, Serpent-Twofish-AES и Twofish-Serpent [19]. Программа включает в себя криптографические хэш-функции: RIPEMD-160, SHA-256, SHA-512 и Whirlpool [20]. Работает с большинством операционных систем, таких как Linux, Windows и OSX.

Из плюсов данного программного обеспечения стоит отнести его бесплатность и открытость, из минусов - его громоздкость и сложность, т.к. он работает с виртуальными дисковыми разделами.

1.3 Медиаконтейнеры

Медиаконтейнер —- используется для хранения и использования различных медийных типов данных. Простые форматы контейнеров могут содержать различные типы аудио форматов, в то время как более продвинутые форматы медиаконтейнеров могут поддерживать несколько аудио и видео дорожек, субтитры, информацию о главах, другие мета-данные , теги, информация о синхронизации дорожек, необходимая для воспроизведения различных потоков синхронно. В большинстве случаев, заголовок файла, метаданные и способ синхронизации определяются типом контейнера. Существуют форматы контейнеров оптимизированные для низкого качества используемые передачи потокового видео через интернет.

Рассмотрение таких видов контейнеров необходимо, для понимаю общих методов создания любых контейнеров и их внутренней структуры. Мы рассмотрим только популярные виды таких контейнеров.

1.3.1 Формат WAV

WAV — это аудиофайл, файл-контейнер для хранения записи оцифрованного аудио, подвид RIFF [21]. Формат был разработан Microsoft и IBM для хранения аудио-битового потока на ПК.

Этот контейнер, обычно, используется для хранения несжатого звука в импульсно-кодовой модуляции. Однако контейнер разрешает использование какого либо кодирования.

1.3.2 Формат MPEG-4

MPEG-4 [22] — стандарт описывающий формат который представляет собой мультимедийный контейнер использующийся для хранения видео и аудио дынных, но так-же может использоваться для хранения субтитров и статичных изображений. Как и большинство современных форматов контейнеров он хорошо подходит для потоковой передачи видео по интернету. Единственным официальным расширением файла является .mp4.

1.3.3 Формат Matroska

Matroska [23] (Матрёшка) — медиаконтейнер представляет собой открытый стандарт описывающий собой контейнер, формат файла, который может содержать неограниченное количество видео, аудио, изображений и дорожен субтитров в одном файле [24]. Он призван служить в качестве универсального формата для хранения файлов мультимедийного характера, например фильмы или ТВ-шоу. Matroska аналогична концепции других контейнеров, таких как AVI, MP4, или AFS, но спецификация формата полностью открыта. Так-же есть множество реализации с открытым исходным кодом.

Расширениями файлов Matroska являются:

- MKV - для видео (с субтитрами и аудио дорожками);
- MK3D - для стереоскопического видео;
- MKA - только для аудио файлов;
- MKS - только для субтитров.

Название формата происходит от русской деревянной игрушки — матрешки, которая полая внутри и может содержать еще одну куклу матрешку,

а та еще одну и так далее.

Разработка проекта началась 6 декабря 2002 года в качестве развития мультимедийного формата MCF. После разногласий главных разработчиков формата MCF насчет выбора хранения данных в самом контейнере, произошло разделение на два разных проекта, одним из которых и стал Matreska.

1.3.4 Формат TIFF

TIFF — компьютерный формат файла для хранения растровой графики [25], широко используется графическими художниками [26], издательской индустрией и фотографами. Формат TIFF широко поддерживается различными приложениями для фоторедактирования, издательской деятельности, верстки, сканирования, факса, обработки текста, распознавания символов и других.

Формат был разработан корпорацией Aldus для использования издательскими домами. Последняя версия была выпушена в 1992 году, после чего права на этот формат купила корпорация Adobe.

Изначально формат поддерживал сжатие без потерь, впоследствии формат был дополнен для поддержки сжатия с потерями (например JPEG).

Структура контейнера представляет собой последовательный контейнер, в начале файла расположен заголовок в котором указано расположение первого блока данных. После каждого блока хранится адрес следующего блока, и так далее.

Формат поддерживает различные виды сжатия, например такие как JPEG и ZIP.

1.4 Другие контейнеры

1.4.1 Формат PE (EXE файлы)

PE (EXE файлы) — формат для исполняемых файлов, объектного кода и динамических библиотек используемых в 32- и 64-битных версиях операционной системы Windows [27]. Формат PE представляет собой описание контейнера который содержит в себе всю информацию необходимую для операционной системы Windows для загрузки исполняемого файла и использу-

емых им данных в память. Исполняемый код включает в себя ссылки на связывания динамически загружаемых библиотек, таблиц экспорта и импорта API функции. В операционных системах NT, формат PE используется для EXE, DLL, SYS и других типов файлов.

1.4.2 Формат ISO-образ

ISO-образ — это архивный файл содержащихся данных оптического диска, тип образа диска который содержит в себе все данные из каждого записанного сектора оптического диска, включая его файловую систему. Название формата пришло от названия файловой системы используемое на CD дисках - ISO 9660, обычно используемой на DVD и Blu-ray дисках [28].

ISO-образы могут создаваться с помощью специального программного обеспечения, как копия существующего диска, как коллекция файлов или как копия другого формата образа дисков.

В данный момент почти во всех современных операционных системах встроена поддержка работы с ISO образами, например Windows, OS X и Linux.

1.4.3 Формат IMG

IMG — файловый формат который позволяет хранить копии CD дисков, магнитных дисков и жестких дисков в бинарном виде, записываемый сектор к сектору. Главное отличие от ISO файлов состоит в том что IMG образ хранит в себе полную копию диска, включая различные заголовки и дорожки которые ISO образ может пропустить.

1.4.4 Базы данных

Базы данных — базы данных в каком-то виде являются контейнером, т.к. они хранят в себе различную текстовую информацию и файлы в отдельном файле.

SQLite — компактная встраиваемая реляционная база данных написанная на языке программирования C [29]. В отличий от других баз данных SQLite не является клиент-серверной, а встраивается в конечное приложе-

ние. Исходный код библиотеки передан в общественное достояние.

SQLite популярное решение для встраивания базы данных в конечное приложение, например в веб-браузер. В данный момент она является самое используемой базой данных, т.к. она встроена в различное программное обеспечение, такое как браузеры, операционные системы, мобильные телефоны и другие [30].

Полное состояние базы SQLite обычно представляет собой один файл на диске называемый "основной файл базы данных". Во время транзакции создается еще один файл в котором хранятся все изменения которые произошли во время ее выполнения, если транзакция прошла неудачно, то все изменения отменяются. Файл базы данных разделен на страницы, размер страниц во всей базе данных одинаковый. В каждой странице находится заголовок, дерево данных, дополнительная информация и номер страницы. В начале файла хранится заголовок с адресами страниц [31].

2 Шифрование

Шифрование — обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения конфиденциальности передаваемой информации. Важной особенностью любого алгоритма шифрования является использование ключа, который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма [1, 32].

Пользователи являются авторизованными, если они обладают определенным аутентичным ключом. Вся сложность и, собственно, задача шифрования состоит в том, как именно реализован этот процесс [1].

Предположим, что отправитель хочет послать сообщение получателю, но вместе с этим он хочет послать свое сообщение безопасно, он хочет быть уверен что его сообщение не смогут прочитать.

Само сообщение называют *открытым текстом*, изменение вида сообщения так, чтобы спрятать его суть называют шифрованием. Шифрованное сообщение называют *шифротекстом*. Процесс преобразования шифротекста в открытый текст называется *дешифрованием* [1]. Эта последовательность продемонстрирована на рисунке 5.

Рисунок 3 – Шифрование и дешифрование

Наука которая занимается шифрование называется криптографией, а людей которые этим занимаются называют криптографами. Криптоаналитиками называют тех, кто постоянно использует криптоанализ для взлома шифротекста, тое есть раскрытия оригинального сообщения. Область математики, которая включает в себя криптографию и криптоанализ называют криптологией, а людей которые ей занимаются - криптологами.

Кроме обеспечение конфиденциальности криптография часто используется для других функции [1, 32]:

– Проверка подлинности. Получатель сообщения может проверить его источник, а злоумышленник не сможет выдать себя за отправителя.

– Целостность. Получатель сообщения может проверить, не было ли изменено сообщение в процессе доставки. Злоумышленник не сможет подменить правильное сообщение своим.

– Неотрицание авторства. Отправитель не сможет отрицать отправку сообщения, т.к. он его подписал.

Криптографический алгоритм, также называемый шифром, представляет собой математическую функцию используемую для шифрования и дешифрования. Если безопасность алгоритма основана на сохранений его в тайне, то это ограниченный алгоритм, его не стоит использовать в настоящее время.

Симметричное шифрование — это такая криптосистема в которой для шифрования и расшифрование используется один и тот же ключ. Использование алгоритмов с симметричным шифрованием требует, чтобы отправитель и получатель согласовали используемый ключ перед началом передачи сообщений. Безопасность симметричного ключа определяется ключом, раскрытие ключа означает, что кто угодно сможет шифровать и дешифровать сообщения.

Симметричные алгоритмы делятся на две категории:

– Потоковые — обрабатывает сообщение побитно или побайтно, используется для потоковой передачи данных.

– Блочные — обрабатывают сообщение блоками данных одинакового размера.

Шифрование с открытым ключом (асимметричное шифрование) — шифрование разработанное таким образом, что ключ, используемый дли шифрования, отличается от ключа дешифрования. Ключ дешифрования не может быть рассчитан по ключу шифрования за разумный интервал времени.

2.1 Симметричные алгоритмы шифрования

Рассмотрим только алгоритмы которые в данный момент являются общепринятыми к использованию стандартами или являются предшественниками текущих и имеют только историческую ценность.

2.1.1 Алгоритм DES

DES — алгоритм блочного симметричного шифрования [33]. Был разработан в 1975 году компанией IBM, и был принят как государственный стандарт шифрования США в 1977 году. В данный момент был заменен алгоритмом AES [34].

DES когда-то был общепринятым стандартом для симметричного шифрования цифровых данных. Он очень повлиял на развитие современной криптографии в академическом мире. Разработан в начале 1970-х годов в компании IBM на основе более ранней работы Хорста Фейстеля, алгоритм был представлен в Национальное бюро стандартов США в соответствии с конкурсом агентства по разработке системы для защиты чувствительных и секретных государственных данных. В 1976 году, после консультации с Агентством национальной безопасности (АНБ), НБС в конечном итоге выбрал слегка измененный вариант (усиленный против дифференциального криптоанализа, но ослабевший против грубого перебора), был опубликован в качестве стандарта для Соединенных Штатах в 1977 году. Публикация АНБ утвержденного стандарта шифрования, привело к его быстрому принятию в международном использований и широкого академического изучения. Разногласия в академических кругах возникли из-за некоторых элементов дизайна алгоритма, относительно короткого ключа, конструкции блочных шифров, а также участия АНБ в разработке алгоритма, которую подозревали в использований возможных закладок в алгоритме. Интенсивное изучение алгоритма в течение времени привело к современному развитию блочного шифрования и их криптоанализа.

DES сейчас считается небезопасным для использования. Главным образом из-за размера ключа — 56 бит, он слишком слишком короток. В январе 1999 года, сайт distributed.net и Фонд Electronic Frontier сотрудничая смогли взломать шифрование DES за 22 часа и 15 минут [35].

Есть также некоторые исследования, которые демонстрируют теоретические слабости в шифре, хотя они являются неосуществимыми на практике. Алгоритм, как считается, практически безопасен в виде тройного DES (3DES), хотя существуют теоретически разработанные атаки на данный вид

реализации. В последние годы данный алгоритм был заменен на Advanced Encryption Standard (AES). Кроме того, DES был убран из стандарта в США.

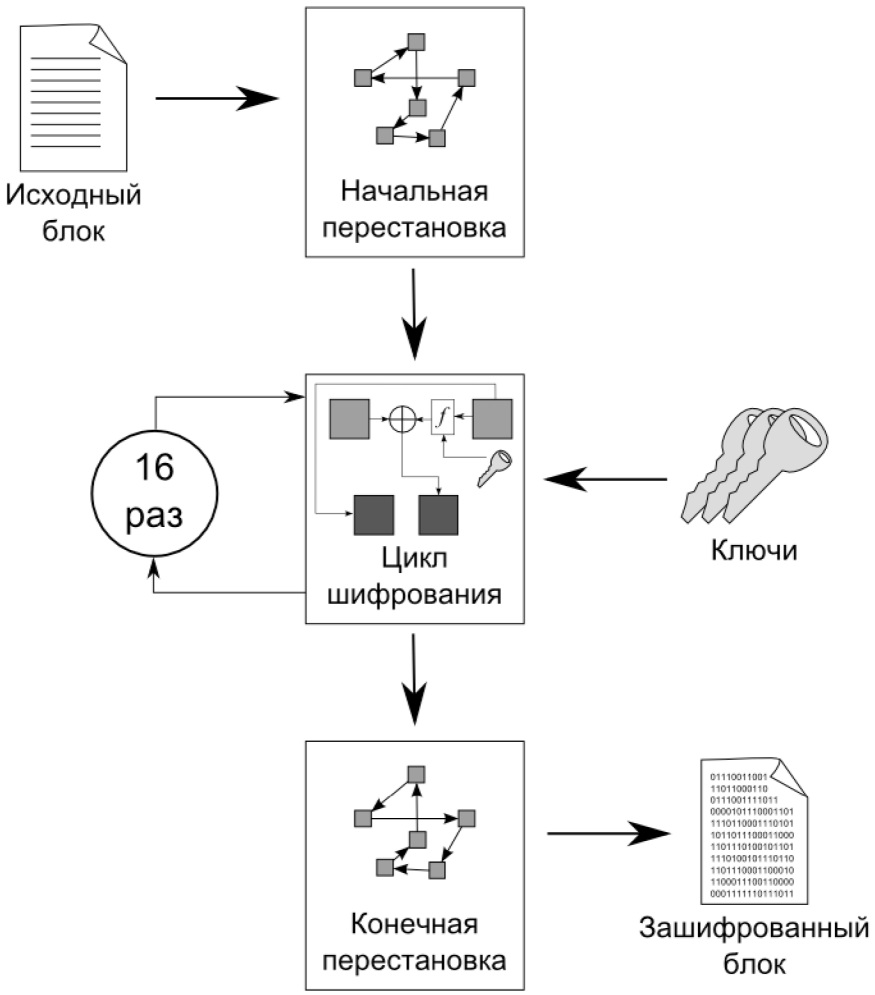

Рисунок 4 – Схема шифрования алгоритма DES

DES является архитипическим блочным алгоритмом шифрования, который принимает битовую строку открытого текста фиксированной длины и преобразует его через ряд сложных операций в другую битовую строку зашифрованного текста такой же длины. В случае DES — размер блока составляет 64 бита. DES также использует ключ для шифровки, так что расшифровка может выполняться только теми, кто знает оригинальный ключ, используемый для шифрования. Ключ обычно состоит из 64 бит; Однако, только 56 из них на самом деле используются алгоритмом. Восемь битов ис-

пользуются исключительно для проверки четности, и затем отбрасываются. Таким образом, эффективная длина ключа составляет 56 бит.

Ключ номинально хранится или передается в виде 8 байт, каждый из которых имеет нечетности.

Один бит в каждом 8-битном байте ключа может быть использован для обнаружения ошибок в генерации ключа, распределения и хранения. Биты 8, 16, ..., 64 предназначены для использования в обеспечении четности.

Как и другие блочные шифры, DES сам по себе не является безопасным способом шифрования, но должен вместо этого использоваться в различных режимах работы, которые осуществляют безопасность. FIPS-81 определяет несколько режимов работы с DES. Дополнительные комментарии по использованию DES содержатся в FIPS-74.

Дешифрирование использует тот же ключ, как и в шифрование, однако ключом подается в обратном порядке.

3DES (Triple DES) — является модификацией алгоритма DES, главное отличие является в том что длина ключа увеличена до 112 бит, благодаря тому что алгоритм проходит один и тот-же блок 3 раза. 3DES является очень защищенным, но главным его недостатком по сравнению с другими современными алгоритмами является его низкая скорость работы.

2.1.2 Алгоритм ГОСТ 28147-89

ГОСТ 28147-89 (Магма) — межгосударственный стандарт симметричного шифрования, введенный в 1990 году [36]. Полное название — "ГОСТ 28147-89 Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования". Блочный шифроалгоритм. При использовании метода шифрования с гаммированием может выполнять функции поточного шифроалгоритма. Кроме того, на основе этого блочного шифра существует хэш-функция ГОСТ.

Разработан в 1970-е годы, стандарт был отмечен "Совершенно секретно а затем понижен до уровня "секретно"в 1990 году. Вскоре после распада СССР, он был рассекречен и был выпущен для публики в 1994 году. ГОСТ 28147 был советской альтернативой стандартному алгоритму США — DES, так-же они очень похожи по структуре.

ГОСТ — блочный шифр и имеет размер блока 64-бита и размер ключа 256-бит и 32 циклами преобразования. В основе алгоритма — сеть Фейстеля [1].

Выделяют четыре режима алгоритма:

– простой замены или режим электронной кодовой книги;

– гаммирование;

– гаммирование с обратной связью;

– режим выработки имитовставки — используется как хэш функция.

Достоинства алгоритма ГОСТ [1]:

– нет смысла атаки с полным перебором;

– высокое быстродействие на современных компьютерах;

– наличие защиты от навязывания ложных данных.

ГОСТ устойчив к широким видам атак, такими как линейный и дифференциальный криптоанализ [37]. Однако было теоретически доказано что данный алгоритм имеет некоторые слабости в работе с ключами, например что разряженные ключи, с значительным преобладанием 0 или 1, являются слабыми для применения [38].

Главные проблемы алгоритма связаны с неполнотой стандарта, например плохо описаны генерация ключей и таблицы замен. Как писалось выше, у алгоритма существуют слабые и ключи и таблицы замен, но в стандарте нет рекомендации по выбору и отсева слабых. ГОСТ не был принят в международный блочный стандарт шифрования ISO/IEC 18033-3 [39], и в нем небыли опубликованы стандартные и рекомендованные таблицы замены. Это поднимает ряд проблем:

– нельзя определить криптостойкость алгоритма, не зная таблицы замен;

– реализация алгоритма в двух системах могут быть одинаковые, но совместимость между ними может отсутствовать, из-за разных таблиц замены;

– возможно что вам предоставят заранее слабые таблицы замен.

2.1.3 Алгоритм AES

AES — также известный как Rijndael (его первоначальное название), является стандартом для шифрования электронных данных, установленный американским Национальным институтом стандартов и технологий (NIST) в 2001 году.

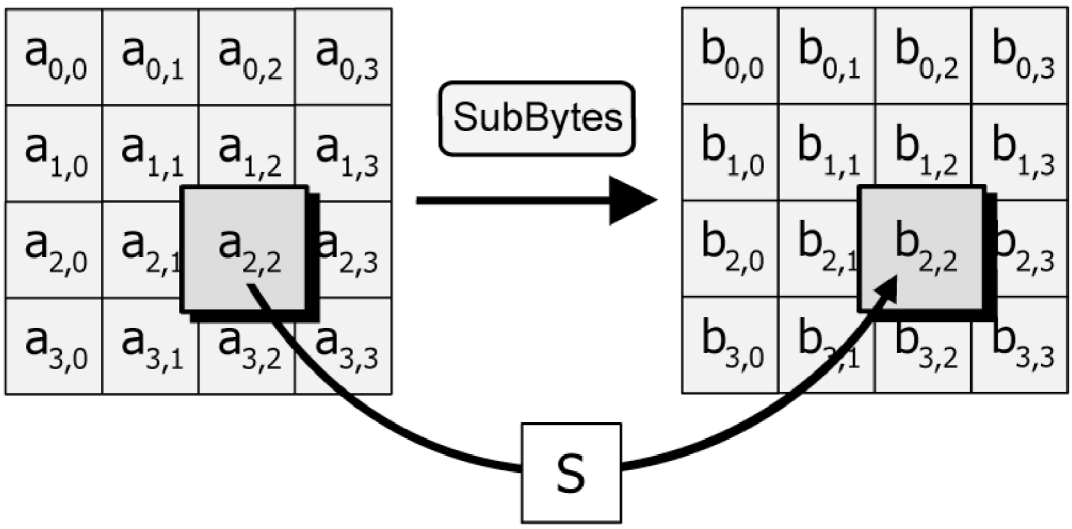

Рисунок 5 – Побайтовая операция замены алгоритма AES

AES разработан на основе шифра Rijndael, разработанный двумя бельгийскими криптографами, Daemen и Rijmen, которые представили его NIST в процессе выбора нового стандарта шифрования. Rijndael представляет собой семейство шифров с различными размерами ключей и блока.

Для AES, NIST выбраны три вида алгоритма шифрования Rijndael, с размером блока 128 бит, но с различной длинной ключа: 128, 192 и 256 бит.

AES был принят правительством США и в настоящее время используется во всем мире. Он заменяет стандарт шифрования данных (DES), который был опубликован в 1977 году. Алгоритм AES описывается как симметричный, то есть тот же ключ используется для шифрования и дешифрования данных.

В США, AES. был принят как стандарт в ноябре 2001 года. Он был выбран после тщательного отбора в течений 5 лет из 15 других предложенных решений.

AES вступил в силу в качестве федерального стандарта правительства США 26 мая 2002 года после утверждения министром торговли. AES входит в стандарте ISO / IEC 18033-3. AES доступен во многих различных программных пакетах шифрования, и является первым (и единственным) общедоступным алгоритмом одобренным Агентством национальной безопасности (NSA) для совершенно секретной информации при использовании в АНБ.

Имя Rijndael является игрой слов от имени двух изобретателей (Daemen и Rijmen).

В данный момент опубликовано несколько возможных вариантов атак на этот алгоритм, но все они не имеют практического использования [40, 41].

Высокая скорость и низкое потребление оперативной памяти является огромным плюсом в выборе шифрования AES. Применение AES возможно на огромном выборе устройств, от 8-битных чипов до высокопроизводительных компьютеров [42].

2.2 Режимы шифрования симметричного блочного шифрования

Режим шифрования — представляет собой алгоритм, который использует блок шифрования для шифрования сообщений произвольной длины таким образом, что-бы обеспечивать конфиденциальность и аутентичность шифруемых данных. Блочный шифр сам по себе подходит только для защищенного криптографического преобразования (шифрования или дешифрования) одной строки битов, одинаковой длины, называемой блоком. Режим работы описывает, как повторно применить операцию блочного шифра, чтобы надежно трансформировать объемы данных больше, чем блок.

Большинство режимов требует уникальную двоичную последовательность, которую часто называют вектор инициализации (IV), для каждой операции шифрования. IV должен быть неповторяющимся, для некоторых режимов, а также случайным для других. Вектор инициализации используется для обеспечения создания различного шифртекста, если тот же самый открытый текст шифруется несколько раз подряд с тем же самым ключом. Блочные шифры поддерживают один или несколько размеров блока, но во время преобразования размер блока всегда фиксирован.

Режимы блочного шифрования работают на целых блоках и требуют,

чтобы последняя часть данных была дополнена до полного блока, если он меньше, чем текущий размер блока. Существуют, однако, способы, которые не требуют заполнения пустых частей блока, потому что они эффективно используют блочный шифр в качестве потокового шифра, такие шифры может кодировать сколь угодно длинные последовательности байтов или битов.

Исторически сложилось так, что режимы шифрования были тщательно изучены в отношении их свойств распространения ошибок при различных сценариях изменения данных. В дальнейшем развитие защиты целостности рассматривается как совершенно отдельная криптографическая тема. Некоторые современные режимы работы сочетают конфиденциальность и целостность эффективным способом.

Вектор инициализации (IV), или стартовая переменная (SV) представляет собой блок битов, который используется несколькими режимами для рандомизации шифрования, следовательно, для получения различных шифротекстов даже если тот же самый открытый текст шифруется несколько раз.

Вектор инициализации имеет другие требования к безопасности, чем ключ, таким образом, IV, как правило, не должен быть секретным. Тем не менее, в большинстве случаев, важно, чтобы вектор инициализации никогда не использовать повторно под тем же ключом. Для CBC и CFB, повторное использование создает утечку некоторой информации о первом блоке открытого текста, а также о любом общем текстом между двумя сообщениями. Для OFB и CTR, повторное использование IV полностью разрушает безопасность. Это можно проверить, потому что оба режима эффективно создают битовый поток, который подвергается операции XOR с открытым текстом, и этот поток битов зависит от пароля и только IV. Повторное использование битового потока разрушает безопасность. В режиме CBC, IV должен, кроме того, быть непредсказуемым во время шифрования, в частности, раньше было обычной практикой повторно использовать последний блок зашифрованного текста сообщения, в качестве IV для следующего сообщения, но это действие небезопасно (например, этот метод был использовался в SSL 2.0).

Блочный шифр работает на блоках фиксированного размера (называемый как размер блока), но сообщения приходят в различных длинах. Таким

образом, некоторые режимы (а именно ECB и CBC) требуют, чтобы конечный блок был дополнен перед шифрованием. Существует несколько схем дополнения. Проще всего добавить нулевые байты к открытому тексту, чтобы довести его длину до кратной размеру блока, но необходимо позаботиться о том, что исходная длина открытого текста могла быть восстановлена, это тривиально, например, если открытый текст представляет собой строку в стиле языка программирования C, который не содержит нулевых байт за исключением конца. Немного более сложным является оригинальный метод DES, который добавляет один бит в конец, а затем дополняется нулями чтобы заполнить блок. Если сообщение заканчивается на границе блока, будет добавлен целый блок заполнения.

Режимы CFB, OFB и CTR не требует каких-либо специальных мер для обработки сообщений, длины которых не кратны размеру блока, так как данные режимы работы с работают с помощью применения функции XOR с открытым текстом.

2.2.1 Режим шифрования ECB

Electronic code book (ECB) — режим электронной кодовой книги, этот режим еще называют "режимом простой замены". В данном режиме каждый блок шифруется отдельно, и никак не влияет на последующие шифрования (Схема работы представлена на рисунке 6).

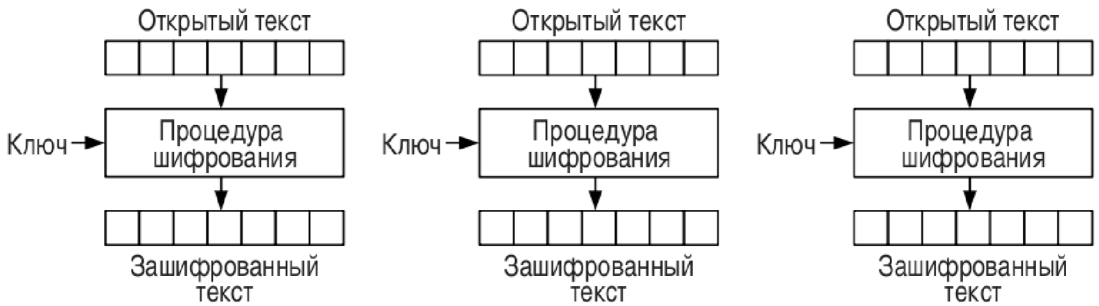

Рисунок 6 – Шифрование в режиме ECB (режиме электронной кодовой книги)

Главным недостатком является то, что во время шифрования, результат сохраняет статические особенности открыто текста, поскольку одинаковым

блокам шифротекста соответствуют одинаковые блоки открытого текста.

Этот режим называется режимом электронной кодовой книги, так как существует возможность создать книгу, в которой каждому блоку открытого текста будет сопоставлен блок зашифрованного текста. Однако создать книгу — нетривиальная задача. Если размер блока равен x бит, то в книге будет содержаться в два раза больше записей, и каждая книга будет соответствовать одному ключу.

2.2.2 Режим шифрования CBC

Cipher block chaining (CBC) — каждый блок открытого текста обрабатывается функцией XOR с предыдущим блоком зашифрованного текста перед шифрованием. Таким образом, каждый блок зашифрованного текста зависит от всех блоков открытого текста обработано до этого момента. Для того, чтобы каждое сообщение было уникальным, вектор инициализации должен использоваться в первом блоке.

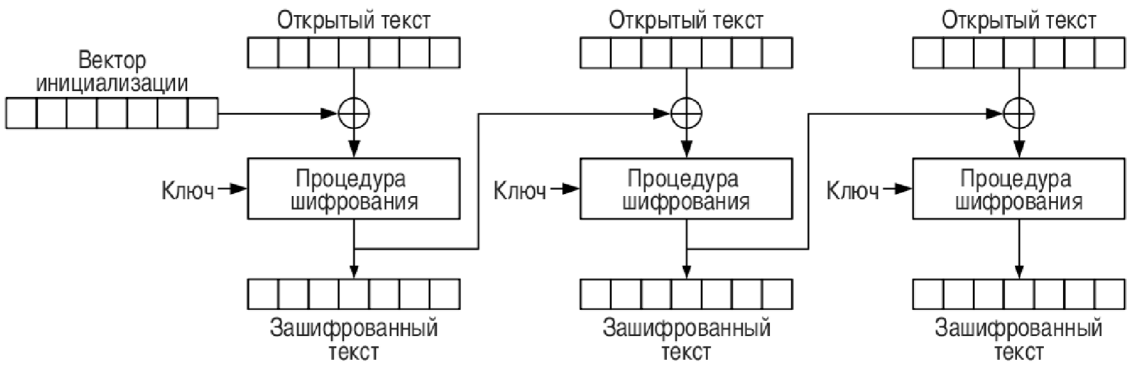

Рисунок 7 – Шифрование в режиме CBC

CBC был наиболее часто используемый режим работы. Основной недостатком является, то что шифрование является последовательным (т.е. оно не может быть распределено), и сообщение должно быть расширено до кратности размера блока. Обратите внимание, что изменение одного бита в текстовом формате или IV влияет на все следующие блоки шифротекста.

Дешифровка с неправильным IV делает первый блок открытого текста нечитаемым. Из-за того что шифрование основано на XOR с уже зашиф-

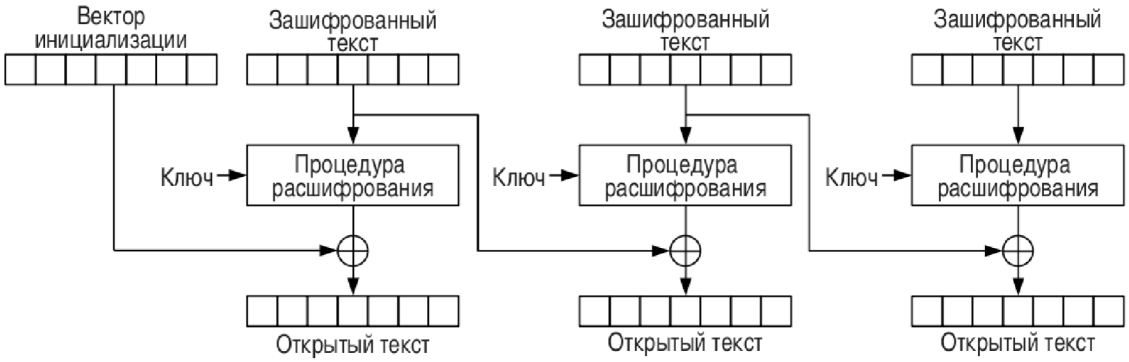

Рисунок 8 – Расшифрование в режиме CBC

рованным блоком, это дает возможность злоумышленнику распараллелить операцию дешифровки, и это дает некоторое преимущество злоумышленнику при попытке взлома шифра.

У этого режима есть некоторые особенности:

– Наличие механизма распространения ошибки: если при передаче произойдёт изменение одного бита шифротекста, данная ошибка распространится и на следующий блок. Однако на последующие блоки (через один) ошибка не распространится, поэтому режим CBC также называют самовосстанавливающимся [43];

– Неустойчив к ошибкам, связанным с потерей или вставкой битов, если не используется дополнительный механизм выравнивания блоков;

– Злоумышленник имеет возможность добавить блоки к концу зашифрованного сообщения, дополняя тем самым получаемыйм открытый текст (однако без ключа получается мусор);

– Для очень крупных сообщений (32 Гбайта при длине блока 64 бита) всё-таки возможно применение атак, основанных на структурных особенностях открытого текста (следствие парадокса дней рождений).

2.2.3 Режим шифрования CFB

Cipher Feedback (CFB) — близкий родственник CBC, преобразует блочный шифр в самосинхронизирующийся поточный шифр. Режим работы очень похож; в частности, CFB дешифрование практически идентичен CBC, но

шифрование выполняется в обратном порядке.

По определению самосинхронизирующийся шифр, если часть шифротекста теряется (например, из-за ошибок при передаче), то приемник потеряет лишь часть исходного сообщения (искажает контент), и алгоритм в состоянии продолжать правильную расшифровку после обработки некоторого количество входных данных.

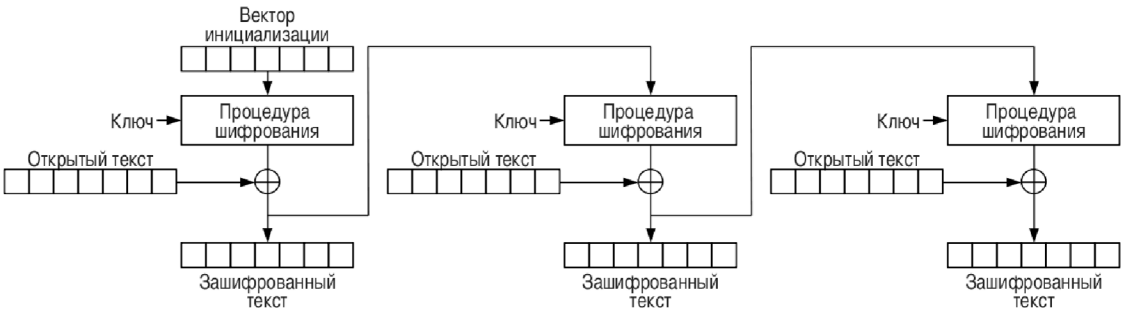

Рисунок 9 – Шифрование в режиме обратной связи по шифротексту

Криптостойкость CFB задается криптостойкостью используемого шифра. Блоки открытого текста «смешиваются» («маскируются») с блоками шифротекста. Если в режиме CFB с полноблочной обратной связью имеется два одинаковых блока шифротекста, результат, например, шифрования алгоритмом DES на следующем шаге будет тем же. Скорость шифрования режима CFB с полноблочной обратной связью такая же, как и у блочного шифра, причём возможности распараллеливания процедуры шифрования ограничены.

2.2.4 Режим шифрования OFB

Output Feedback (OFB) — представляет блочный шифр в виде синхронного шифра потока. Он генерирует блоки потока ключей, которые затем проходят через с функцию XOR с блоками открытого текста, чтобы получить зашифрованный текст. Так же, как и с другими потоковые шифры, если перевернуть бит в шифротексте, то мы перевернем бит в том-же месте в открытом тексте. Это свойство позволяет исправлять множество ошибок, даже если код сильно испорчен.

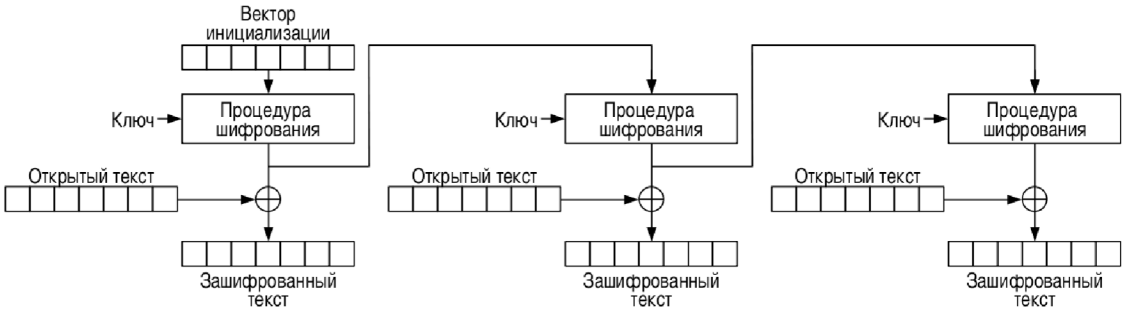

Рисунок 10 – Шифрование в режиме OFB

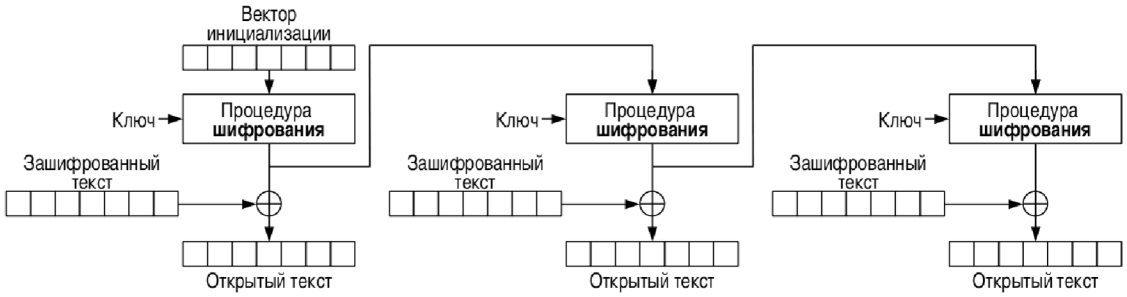

Рисунок 11 – Расшифрование в режиме OFB

Из-за симметрии операции исключающее ИЛИ, шифрование и дешифрование одинаковые.

Каждый вывод операции блочного шифра зависит от предыдущего блока, и поэтому он не может быть выполнены параллельно.

2.3 Асимметричное шифрование

Шифрование с открытым ключом, или асимметричная криптография, это любая криптографическая система, которая использует пары ключей: открытый ключ, который может быть известен всем, в паре с закрытым ключом, который известен только владельцу. Есть две функции которые может выполнять этот вид шифрования: подпись сообщения, чтобы знать что вам отправили оригинал и шифрование с помощью открытого ключи и дешифровка с помощью закрытого.

В системе шифрования с открытым ключом, любой человек может зашифровать сообщение с помощью публичного ключа, но сообщение может

быть расшифровано только закрытым ключом. Сила системы криптографии с открытым ключом зависит от степени сложности (вычислительной непрактичности) секретного ключа, чтобы было сложно определить из открытого ключа - закрытый. Безопасность зависит только от хранения приватного ключа, и публичный ключ может быть спокойно опубликован без опасности для пользователя.

Асимметричное шифрование часто построено на криптографических алгоритмах, которые основаны на математических проблемам, которые в настоящее время не имеют эффективного решения, например дискретного логарифмирования и отношения эллиптических кривых. Алгоритмы с открытым ключом, в отличие от симметричных алгоритмов, не требуют безопасного канала для первоначального обмена одного (или более) секретных ключей между сторонами.

Из-за вычислительной сложности асимметричного шифрования, как правило, из-за этого используется только для небольших блоков данных, обычно для передачи ключа симметричного шифрования (например, ключ сеанса). Этот симметричный ключ затем используется для шифрования остальной части потенциально длинной последовательности сообщений. Симметричного шифрования / дешифрования основано на более простых алгоритмов и намного быстрее.

Алгоритмы с открытым ключом являются основным ингредиентом безопасности в криптосистемах, приложениях и протоколах. Они лежат в основе различных стандартов работающих в Интернете, таких как Transport Layer Security (TLS), S / MIME, PGP и GPG. Некоторые общественные открытые алгоритмы обеспечивают распределение ключей и секретности (например, обмен ключами Деффи-Хеллмана), некоторые предоставляют цифровые подписи (например, алгоритм цифровой подписи), а некоторые предоставляют оба (например, RSA).

Рассмотрим только те виды ассиметричного шифрования в котором есть режим шифрования, так-как только он нужен в разработке программного продукта.

2.3.1 Алгоритм RSA

RSA – это первая практическая реализация систем криптографии с публичным ключом. В основе алгоритма лежит задача факторизации произведения двух простых больших чисел. Для шифрования используется операция возведения в степень по модулю N. Для расшифрования необходимо вычислить функцию Эйлера от числа N, для этого нужно знать разложения числа на простые множители.

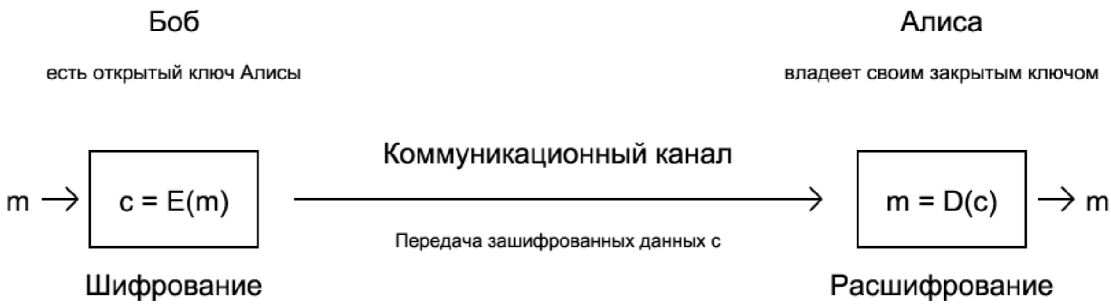

Рисунок 12 – Принцип работы алгоритма RSA

Открытый и закрытый ключ состоит из пары больших целых чисел. Закрытый ключ хранится в секрете, а публичный - общедоступен для использования.

Скорость работы RSA очень низкая и поэтому он не подходит для шифрования длинных сообщений, но его можно использовать в гибридных системах, где например сообщение кодируется с помощью синхронного алгоритма, далее его ключ шифруется с помощью уже асинхронного алгоритма.

В данный момент RSA не рекомендуется использовать с размером ключа меньше 2048 бит, т.к. недавние исследования показали что уже сейчас возможно взломать ключ длиной 768 бит, и что успешно выполнила группа ученых и Швейцарии, Японии, Франции, Германии и США.

3 Описание разработанного файлового формата контейнера

3.1 Общее строение контейнера

Целью работы является разработка файлового контейнера для безопасного хранения данных, было исследовано множество различных контейнеров и были выбраны лучшие стороны каждого из них. В итоге был выбран данный вид структуры файла, почему это было сделано, будет описано далее.

Рисунок 13 – Строение контейнера

3.1.1 Хранение файлов

Каждый файл зашифрован с помощью алгоритма AES с режимом работы CBC, размер ключа выбран максимально большой — 256 бит, размер блока 128 бита, вектор инициализаций совпадает с размером блока. Размер ключа выбран максимально большим, это замедляет работу шифрование примерно на 40%, но при этом отпадает возможность полного перебора в целом.

Предоставить возможность менять размер ключа, если это необходимо.

Зашифрованный файл хранится в контейнере в бинарном виде, без преобразование в другие системы такие как HEX (16-ричная система) или BASE64.

При записи файла, генерируется новый секретный ключ и вектор инициализации. Далее он записывается в "Директорию".

3.1.2 Директория

В директории хранятся расположение файлов в контейнере, ключи и вектор инициализации (IV) к ним, оригинальное название файла и путь в котором он находился и другая различная метаинформация.

Запись в директории представляет из себя:

- Название файла;
- Key (ключ шифрования);
- IV (вектор инициализации);
- Начальная позиция в файле;
- Размер файла в байтах;
- Другая метаинформация.

Директория тоже шифруется, так-же с помощью алгоритма AES в режиме работы CBC, размер ключа – 256 бит, размер блока 128 бит, вектор инициализаций совпадает с размером блока.

Необходимо разрешить в дальнейшем свободно добавлять необходимые поля, без потери обратной совместимости.

3.1.3 Подпись

Подпись представляет собой зашифрованную структуру данных, записанную бинарно в конец файла.

Подпись состоит из:

- Ключа от директории - 32 байта
- Вектор инициализации - 16 байт
- Адрес нахождения начала директории - 19 байт.

Поле "Адрес нахождения начала директории"представляет собой 64-рех разрядное поле для записи целого числа, пустая часть забивается нулями.

3.2 Выбор алгоритма шифрования

Для шифрование файлов и директории (каталога) используется AES.

Выбор в пользу AES состоялся из-за того что он является текущим международным стандартом, так-же существует множество готовых открытых библиотек полностью реализующие его.

Так-же AES имеет множество преимуществ:

– AES очень безопасен. В данный момент есть только теоретические атаки на данный алгоритм.

– Поддержка длинных ключей. Другие алгоритмы, например алгоритм 3DES поддерживает только 112 или 168 бит, а AES поддерживает до 256 бит.

– AES быстрее всех остальных алгоритмов, и реализован как и на аппаратном так и программном уровне.

– Размер блока 128-бит делает атаку через "проблему дня рождения"более трудно реализуемой.

Для шифрования подписи выбран алгоритм RSA. Так-же как и AES он выбран из-за того что это общедоступный стандарт, его реализация доступна в множестве открытых библиотеках. Его преимущества:

– Очень быстрое шифрование по сравнению с другими алгоритмами с открытым ключом;

– Более легкая имплементация по сравнению с другими алгоритмами;

– Более легок в понимании;

– Подпись и расшифровка - похожи; шифрование и проверка подписи - так-же похожи;

– Широко распространен, более лучшая поддержка различными средствами.

3.3 Выбор библиотеки для шифрования

Требование к библиотеки шифрования:

– Библиотека должна быть полностью бесплатной и с открытым кодом.

– Удобство пользования. Нужна хорошо проработанная библиотека имеющая правильную и продуманная структуру;

– Библиотека должная осуществлять как шифрование AES, так и шифрование с открытым ключом RSA.

– Библиотека должна поддерживать работу с потоками, опять таки для удобства использования.

3.3.1 Библиотека OpenSSL

OpenSSL — криптографический пакет с открытым исходным кодом для работы с SSL/TLS. OpenSSL представляет собой библиотеку программного обеспечения для использования в приложениях, которые требуют защиту связи от перехвата информации. Библиотека нашла широкое применение в интернет веб-серверах, обслуживающих большое кол-во веб сайтов.

OpenSSL содержит реализацию с открытым исходным кодом SSL и TLS протоколов. Ядро библиотеки, написанной на языке программирования C, реализует основные криптографические функции и предоставляет различные вспомогательные функции. Так-же существуют обертки которые позволяют использовать библиотеку OpenSSL в других различных компьютерных языках программирования.

Версии библиотеки доступны для большинства Unix-подобных операционных систем (в том числе Solaris, Linux, Mac OS X и BSD), OpenVMS и Microsoft Windows.

Проект OpenSSL был основан в 1998 году, чтобы создать бесплатный набор инструментов для шифрования кода, используемого в Интернете. Он основан на развилке SSLeay, развитие которого неофициально, закончилось 17 декабря 1998 года, когда основатели Янг и Хадсон и начал работать в компании RSA Security.

Команда управления проектом OpenSSL состоит из четырех европейцев. Вся группа развития состоит из 11 членов, из которых 10 являются добровольцами. Есть только один штатный сотрудник, Стивен Хенсон, ведущий разработчик.

Проект имеет бюджет в размере менее 1 миллиона долларов в год и частично полагается на пожертвования. Стив маркиз, бывший консультант ЦРУ в штате Мэриленд начал основные пожертвования и предлагал контракты на консультационные услуги и получил спонсорство от Соединенных

Штатов Министерства внутренней безопасности и Министерство обороны США.

В 2013 году WikiLeaks опубликовал документы, полученные Сноуденом, который показал, что с 2010 года, АНБ было эффективно сломаны SSL / TLS протоколы, возможно, путем использования уязвимостей, таких как HeartBleed.

По состоянию на 2014 г. две трети всех веб-серверов используют библиотеку OpenSSL [44].

В данной библиотеки есть реализации как и AES с всеми режимами работы, так и RSA, но главным недостатком является то, что библиотека написана на языке C, и трудна для понимания без знаний о работе с памятью. Так-же у нее есть функции высокого уровня для шифрования, которые в целом удобны для использования.

3.3.2 Библиотека Crypto++

Crypto++ является свободной с открытым исходным кодом C++ библиотекой классов криптографических алгоритмов и схем, написанная Рэй Дай (один из главных разработчиков криптовалюты Bitcoin). Crypto++ широко используется в научных кругах, студенческих проектах, с открытым исходным кодом и некоммерческих проектах, а также предприятий. Выпущенный в 1995 году, библиотека полностью поддерживает 32-разрядные и 64-разрядные архитектуры для многих основных операционных систем и платформ, включая Android (с использованием STLPort), Apple (Mac OS X и IOS), BSD, Cygwin, IBM AIX и S / 390, Linux, MinGW, Solaris, Windows, Windows Phone и Windows RT. Проект также поддерживает компиляцию под C++ 03 и C++ 11, различные компиляторы и среды разработки, включая Borland Turbo C++, Borland C++ Builder, Clang, CodeWarrior Pro, GCC (включая GCC от Apple), Intel C++ Compiler (ICC) , Microsoft Visual C / C++ и Sun Studio.

Crypto++ предоставляет полный набор криптографических реализаций, и так-же включает в себя менее популярные, менее часто используемые схемы. Например, Camellia является ISO/NESSIE/IETF-одобренным блоковым алгоритмом шифрования эквивалент алгоритма AES, и Whirlpool является

ISO/NESSIE/IETF-одобренной хэш функцией эквивалент SHA, оба алгоритма включены в библиотеку.

Эта высокоуровневая библиотека имеет быструю реализацию алгоритмов RSA и AES. Так же библиотека может работать с потоками.

3.3.3 Библиотека Botan

Botan криптографическая библиотека с открытым исходным кодом, распространяется с лицензией BSD, написанная на C++. Она обеспечивает широкий спектр криптографических алгоритмов, форматов и протоколов, например, SSL и TLS.

Проект был первоначально назывался OpenCL, но данное имя в настоящее время используется Apple Inc. и Khronos Group для гетерогенного платформы системного программирования. Библиотека была переименован в 2002 году в Botan.

Данная библиотека поддерживает алгоритмы AES и RSA, но данная библиотека имеет слабую документацию по сравнению с другими библиотеками и так-же имеет малое сообщество.

Окончательным решением в выборе библиотеки было отдано в пользу CryptoPP, так-как она предоставляет множество преимуществ, таких как:

– Библиотека высокоуровневая. Почти для каждой потребности разработчика уже разработана функция, например запись в поток (файл), дополнение блоков, генерация псевдослучайных чисел, генерация ключей и тд;

– У библиотеки есть полная документация, хоть код легок в понимании;

– Не было обнародовано не одной уязвимости, в отличие от библиотеки OpenSSL.

3.4 Выбор дополнительных библиотек

Для решения различных задач необходимы функции и классы не реализованные в стандартной библиотеки C++, например для работы с:

– файловой системой — необходимо создавать папки, обрабатывать пути, редактировать размер файлов. Для всего этого нужна мультиплатформен-

ная библиотека, так-как системные вызовы для всех этих действий отличаются в операционных системах;

– командной строкой — так-как программа будет без графического интерфейса, необходимо предоставить интерфейс с помощью командной строки, и для большего удобства стоит использовать уже готовую библиотеку с распознаванием команд;

– серриализация — для хранения некоторых структур (например каталога файлов) необходимо его как-то сериализовать, использование записи в виде строки потребует дополнительных усилий хоть и небольших, но поддержка версионности уже потребует больших усилий, нужна библиотека в которой уже все это реализованно.

Есть множество различных библиотек для каждого из этих пунктов, но в итоге выбор был остановлен на наборе библиотек Boost, так-как он включает в себя все эти пункты.

Boost — набор библиотек для языка программирования C++ который предоставляет решение задач для таких вещей как линейная алгебра, генерация случайных чисел, мультипоточность, обработка изображений, регулярные выражения, тестирование и др. В данный момент набор библиотек состоит из более 80 индивидуальных библиотек.

Большая часть библиотек из набора библиотек Boost находится под лицензией Boost Software Licencse, которая позволяет использовать библиотеки Boost как в свободном так и в проприетарном программном обеспечении.

Многие разработчики библиотек Boost, находятся в комитете стандартизации языка C++, и множество библиотек из набора Boost в итоге были включены в STL (набор стандартных шаблонных библиотек языка C++).

Библиотеки нацелены на широкий круг пользователей C++ в разных областях. Они варьируются от библиотек общего пользования например, от библиотеки "умных"указателей, до библиотеки для работы с файловой системой операционной систем, до узконаправленных библиотек для опытных пользователей например шаблоны метапрограммирование (MPL) и предметно-ориентированный язык (DSL).

В целях обеспечения эффективности и гибкости, Boost преимущественно использует шаблоны. Boost был источником большой работы над шабло-

нами и изучения обобщенного программирования и метапрограммирования в C++.

Большинство библиотек описаны в заголовочных файлах, состоящих из inline функций и шаблонов, из-за этого их не нужно компилировать перед использованием. Многие библиотеки из набора Boost представляют собой независимые библиотеки.

3.5 Библиотека CryptoPP

Библиотека шифрования CryptoPP представляет собой исходные файлы на языке C++. Библиотека использует средства объектно-ориентированного программирования и реализована в виде множества заголовочных файлов описание только используемых из них приведено ниже.

3.5.1 Заголовочный файл cryptopp/aes.h

Заголовочный файл который содержит используемые функции для работы с алгоритмом шифрования AES. В нем есть функции как для шифрования, так и для дешифрования. Используемые методы и статические переменные указаны ниже.

CryptoPP::AES — основной класс выполняющий шифрование и дешифрование. Сам по себе отдельно не используется, но используется как параметр шаблона для фильтров CryptoPP.

CryptoPP::AES::BLOCKSIZE — статическая переменная указывающая на размер блока, у AES размер блока всегда равен 16 байтам.

CryptoPP::AES::MAX_KEYLENGTH — статическая переменная указывающая на максимальный размер ключа, но библиотека поддерживает размеры ключей 16, 24 и 32 байт (128, 192 и 256 бит соответственно).

3.5.2 Заголовочный файл cryptopp/files.h

Заголовочный файл содержит в себе классы для работы с файлами, например для чтения из файла или запись в него. Данные классы представляют из себя классы для создания конвейера, у нас есть источник, из источника мы обрабатываем данные как захотим, а далее мы передаем полученный ре-

зультат в конечный пункт.

CryptoPP::FileSource — класс позволяющий считывать данные из файла частями. В конструкторе мы передаем название файла и куда хотим передавать данные (например в другой файл или строку), и далее мы можем вручную считывать с файла частями или считать сразу же весь файл.

CryptoPP::FileSink — класс позволяющий записывать данные в файл, в конструкторе мы передаем название файла в который будет происходить запись, и далее мы можем отправлять в него сообщения которые будут записаны в файл.

3.5.3 Заголовочный файл cryptopp/filters.h

Данный заголовочный файл используется в большинстве внутри библиотеки, он используется для создания фильтров. Фильтр это абстрактный базовый класс который позволяет создавать конвейеры, это паттерн программирования который позволяет оборачивать одну функцию в другую, и они будут обрабатывается в цепочке. Ниже приведен пример исходно кода, который будет считывать данные из одного файла, создавать хеш с помощью хеш функций, после запишет полученный хэш в строку.

```
string str;
FileSource file(filename, true,
    new HexEncoder(new StringSink(str)));
```

CryptoPP::StringSource — класс позволяющий считывать данные из строки, и передавать их дальше по конвейеру.

CryptoPP::StringSink — класс позволяющий записывать данные в строку, в конструкторе мы передаем переменную куда будут записываться исходящий данные в конце конвейера.

3.5.4 Заголовочный файл cryptopp/modes.h

В данном заголовочном файле описываются шаблонные классы для шифрования и дешифрования в различных режимах работы, которые принимают в качестве шаблонного параметра алгоритм шифрования.

CryptoPP::CBC_Mode — класс позволяющий шифровать и дешифровать

данные с помощью заданного в шаблоне алгоритма в режиме работы CBC.

3.5.5 Заголовочный файл cryptopp/osrng.h

В данном заголовочном файле описываются классы для доступа к генератором случайных чисел операционной системы.

CryptoPP::AutoSeededRandomPool — класс который генерирует случайные числа без задавания источника энтропии, он сам выберет лучший смотря в какой операционной вы находитесь.

3.5.6 Заголовочный файл cryptopp/rsa.h

В данном заголовочном файле описываются классы и структуры необходимые для работы с шифрование с помощью алгоритма RSA.

CryptoPP::InvertibleRSAFunction — класс в котором создается и хранится приватный ключ из заданных входных параметров.

CryptoPP::RSAFunction — класс в котором создается и хранится публичный ключ из заданных входных параметров (приватного ключа).

3.6 Библиотека Boost

Набор библиотек для различных нужд Boost представляет собой исходные файлы на языке C++. Библиотека использует средства объектно-ориентированного программирования и реализована в виде множества заголовочных файлов описание только используемых из них приведено ниже.

3.6.1 Заголовочный файл boost/filesystem.hpp

Набор кроссплатформенных библиотек для работы с файловой системой, при компиляции подставляются команды для POSIX систем и для Win API автоматически.

boost::filesystem::exists — функция которая принимает путь в качестве входного параметра, возвращяет булево значение, оно указывает существует по данному пути файл или директория или нет.

boost::filesystem::is_directory — функция которая принимает путь в качестве входного параметра, возвращяет булево значение, оно указывает явля-

ется ли директорией указанный путь.

boost::filesystem::path — класс который хранит в себе путь, и позволяет производить различные манипуляции над ним, например объединение нескольких путей и вычитание одного пути из другого (создание относительного пути).

boost::filesystem::relative — функция которая принимает два пути, возращяет относительный путь от первого пути к второму.

boost::filesystem::is_regular_file — функция которая принимает путь в качестве входного параметра, возращяет булево значение, оно указывает является ли указанный путь файлом.

boost::filesystem::create_directories — функция которая принимает путь в качестве входного параметра, создает полностью папки необходимые для полного построения входного пути.

boost::filesystem::file_size — функция которая принимает путь в качестве входного параметра и его возращяет размер файла, если это файл.

3.6.2 Заголовочный файл boost/serialization.hpp

boost::serialization::access — класс, которые необходимо сделать дружественным к классу который необходимо сериализовать. Позволяет автоматически сериализовать экземпляр класса.

boost::serialization::split_member — позволяет разделить логики сериализации и десериализации экземпляра класса, необходимо это при сохранений сложных структур данных.

boost::archive::binary_oarchive — класс который реализует сохранение экземпляров класса в текст с помощью сериализации.

boost::archive::binary_iarchive — класс который реализует сохранение экземпляров класса из текст с помощью десериализации.

boost::archive::archive_flags::no_header — флаг, запрещает сохранения заголовка в начале строки, это необходимо в целях безопасности при шифровании.

4 Описание разработанной библиотеки

В результате выполнения выпускной квалификационной работы была разработана переносимая библиотека, которая позволяет создавать, открывать и редактировать контейнеры разработанного формата.

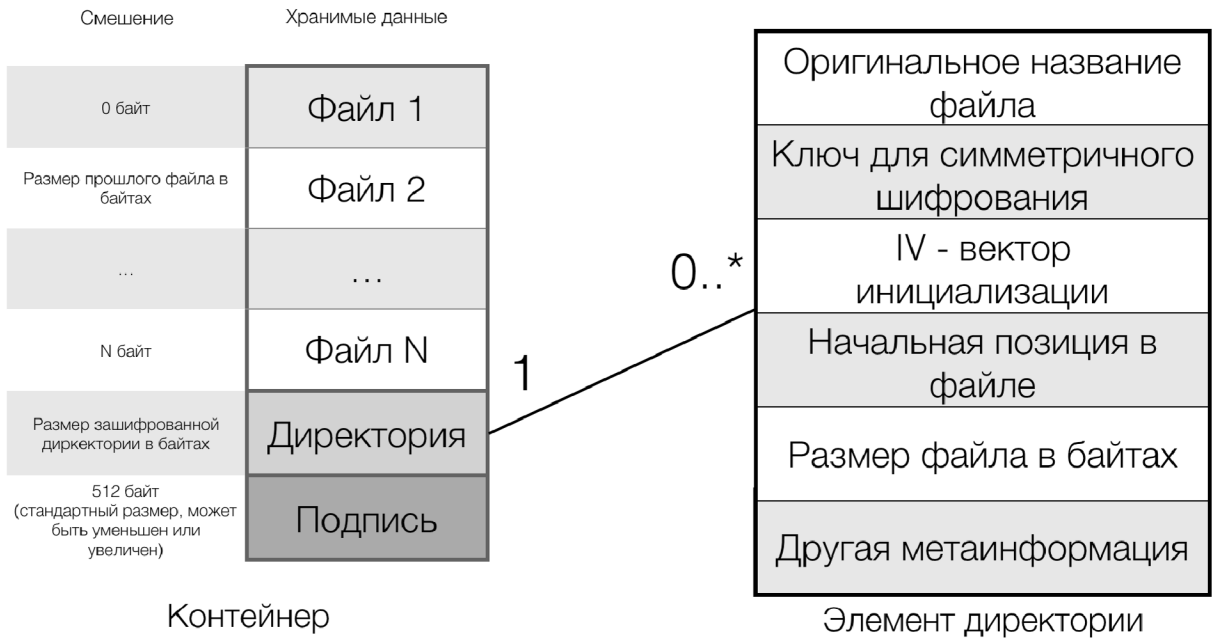

Рисунок 14 – Структура контейнера

Предполагается что пользователь библиотеки будет использовать только шаблонный класс Container. Он позволяет создавать контейнеры или открывать существующие. В качестве параметра шаблона он принимает структуру описанную в файле Utils, которая описывает алгоритм шифрования и размер ключа. Если подать не правильное сочетания размера ключа и алгоритма, не поддерживающий размер заданного ключа, библиотека не скомпилируется, так-как все входные параметры заданы с помощью специализаций шаблонов.

Публичным API для работы с контейнером является шаблонный класс Container, который будет описан ниже, вместе в другими классами и функциями используемые в библиотеки.

На рисунке 14 изображена структура контейнера. Первоначально создается пустой контейнер, при инициализации создается пустой массив эле-

ментов директории. Далее разработчик использующий данную библиотеку может добавить какую либо информацию в контейнер. При записи генерируется ключ для симметричного шифрования и вектор инициализации, название файла (записи) и размер файла.

4.1 Заголовочный файл crypt.h

В данном заголовочном файле описаны шаблонные классы и функции для работы с блочным шифрованием, диаграмма классов изображена на рисунке 15.

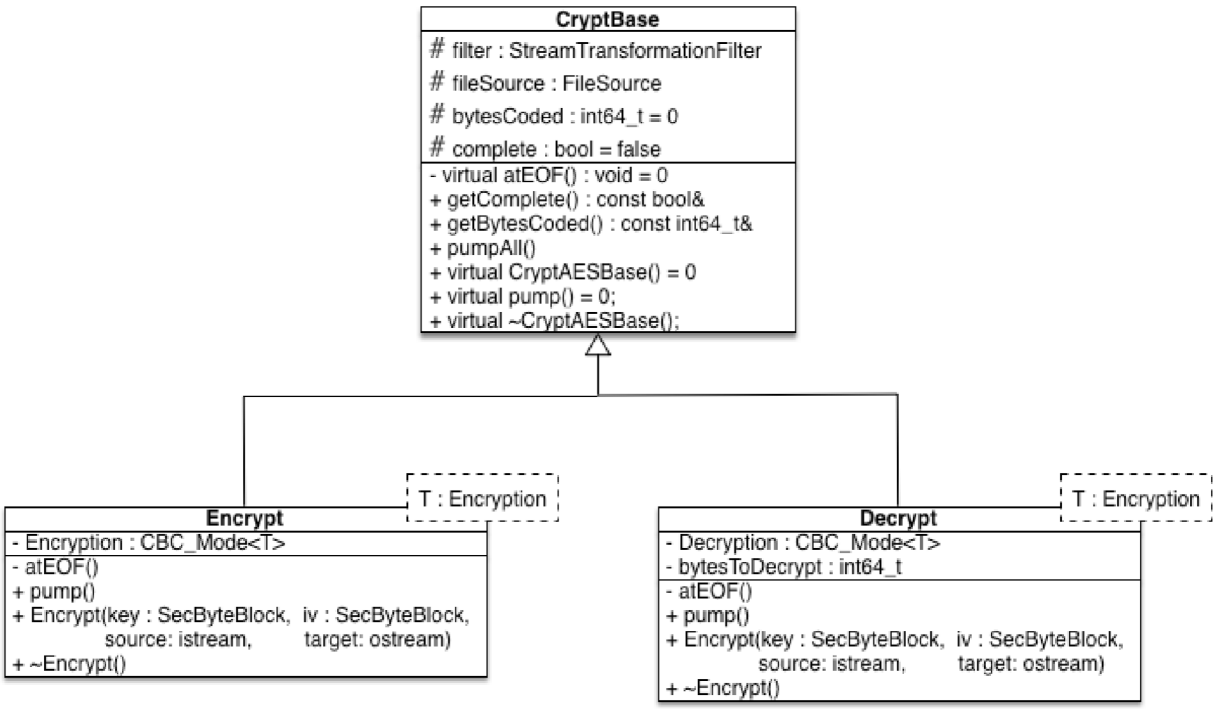

Рисунок 15 – UML диаграмма классов заголовочного файла crypt.h

Функция generateRandomKey(int keySize) — создание случайного блока данных с заданным размером. Данная функция используется для генераций ключей и векторов инициализаций.

Класс CryptBase — шаблонный абстрактный класс используется для создания классов шифрования и дешифрования различными алгоритмами блоками. Вид шифрования принимается в виде параметра шаблона. Не имеет конструктора.

Класс Encrypt — шаблонный класс который наследуется от класса CryptBase, используется для шифрования данных из входного потока в исходящий поток. Вид шифрования принимается в виде параметра шаблона.

Конструктор принимает в качестве параметров:

– Ключ — используемый для шифрования;

– Вектор инициализации — используемый для шифрования, т.к. шифрование работает только в режиме CBC;

– Входной поток — поток из которого происходит чтение данных и их шифрование, при ошибке чтение происходит вызов исключения;

– Исходящий поток — поток в который происходит запись уже зашифрованных данных, при ошибки записи происходит вызов исключения.

Класс Decrypt — шаблонный класс который наследуется от класса CryptBase, используется для расшифрования данных из входного потока в исходящий в поток. Вид шифрования принимается в виде параметра шаблона.

Конструктор принимает в качестве параметров:

– Ключ — используемый для шифрования;

– Вектор инициализации — используемый для шифрования, т.к. шифрование работает только в режиме CBC;

– Входной поток — поток из которого происходит чтение данных и их расшифрование, при ошибке чтение происходит вызов исключения;

– Исходящий поток — поток в который происходит запись уже расшифрование данных, при ошибки записи происходит вызов исключения;

– Количество байт для дешифрования — указывается сколько байт нужно расшифровать.

Ниже представлен исходный код для шифрования строки с помощью алгоритма AES и записи зашифрованных данных в строку.

```
std::string test = "Hello world!";
std::stringbuf inSB(test);
std::unique_ptr<std::istream> in =
    std::make_unique<std::istream>(&inSB);

std::stringbuf outSB;
std::unique_ptr<std::ostream> out =
```

```
    std::make_unique<std::ostream>(&outSB);

cc::Encrypt<cc::AES_256> encrypter(
    key, iv, in.get(), out.get());
encrypter.pumpAll();

std::string testCrypted = outSB.str();
```

4.2 Заголовочный файл rsa.h

В данном заголовочном файле описаны шаблонные классы и функции для работы с ассиметричным алгоритмом шифрования RSA.

Шаблонная функция RSAKeyToString — преобразует ключ формата RSA в строку. Функция может преобразовать как и публичный ключ, так и приватный. Тип функций принимается параметром шаблона.

Шаблонная функция RSAKeyFromString — создает ключ RSA из строки. Тип получаемого ключа, публичный или приватный, задается параметром шаблона.

Шаблонная функция loadKeyFromFile — данная функция загружает ключ из файла, и возращяет публичный или приватный ключ, в зависимости от указанного параметра в шаблоне.

Шаблонная функция saveKeyToFile — данная функция сохраняет ключ в файл, тип ключа указывается с помощью параметра шаблона.

Функция generateRSAKeys — генерирует два ключа приватный и публичный, заданного размера.

Функция encryptStringRSA — шифрует входящую строку с помощью алгоритма RSA с заданным публичным ключом и возвращает зашифрованную строку.

Функция decryptStringRSA — расшифровывает входящую строку с помощью алгоритма RSA с заданным приватным ключом и возвращает расшифрованную строку.

Далее представлен пример исходного кода который загружает публичный ключ RSA из файла и кодирует с помощью его строку.

```
auto publicKey =
    cc::loadKeyFromFile<CryptoPP::RSA::PublicKey>("key.pub");
```

```
std::string cryptedSign = cc::encryptStringRSA(
    *publicKey, "Hello world!");
```

4.3 Заголовочный файл container.h

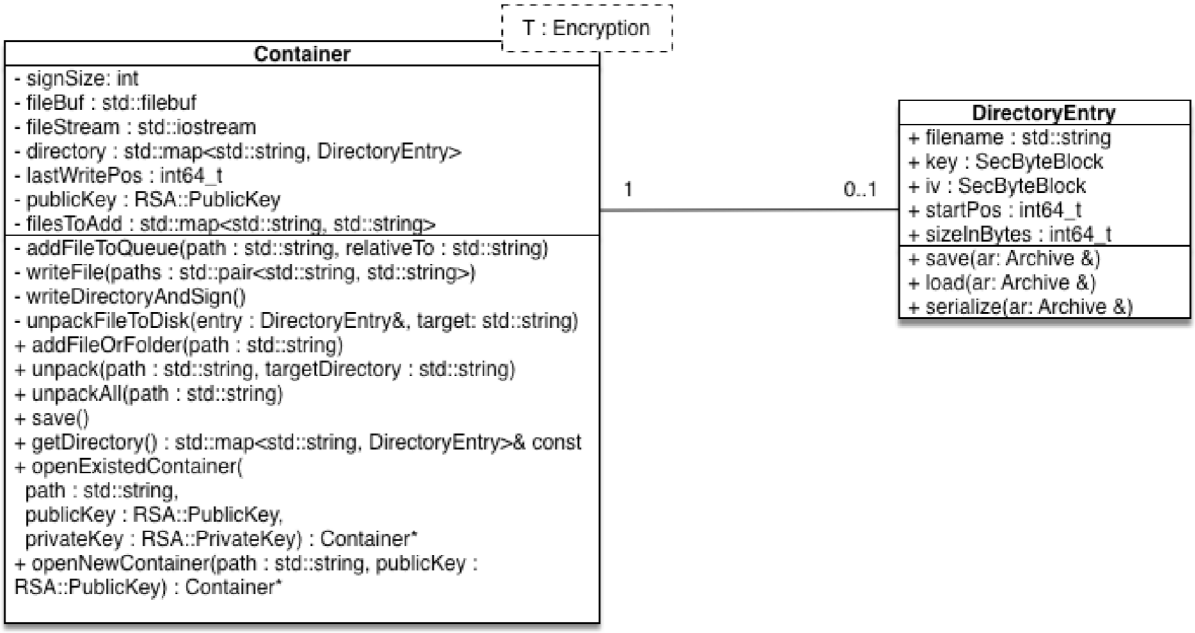

Рисунок 16 – UML диаграмма классов заголовочного файла container.h

В данном заголовочном файле описаны шаблонные классы и структуры для работы с разработанным форматом контейнера, UML диаграмма которого изображена на рисунке 16.

Структура DirectoryEntry — данная структура используется для хранения записей в директории, хранимой в конце разработанного файлового контейнера. В нем указываются необходимые данные для дальнейшей работы с контейнером:

- filename — имя файла;
- key — ключ используемый для шифрования;
- iv — вектор инициализации используемый для шифрования;
- startPos — позиция откуда началась запись файла в контейнер;
- sizeInBytes — размер итогового зашифрованного файла в байтах;

Так-же в данной структуре описаны вспомогательные функции save и load, необходимые для сериализации и десериализации.

Шаблонный класс Container — класс который отвечает за создание или открытие контейнера. В параметре шаблона передается тип использованного алгоритма для блочного шифрования и размер его ключа, при передаче не поддерживаемого размера ключа алгоритмом — компиляция остановится.

У данного шаблонного класса нет публичного конструктора, но есть два статических метода которые создают экземпляр данного класса (шаблонные фабричные методы):

– openExistedContainer — статический метод который принимает путь до уже существующего контейнера, публичный и приватный ключи. Возращяет уникальный указатель на экземпляр класса Container, который уже загрузил в себя директорию существующего контейнера и позволяет манипулировать с ним (добавлять и распаковывать данные).

– openNewContainer — статический метод который принимает путь к новому контейнеру и публичный ключ для ассиметричного шифрования. Возращяет уникальный указатель на экземпляр класса Container с пустой директорией.

Так-же доступны публичные методы:

– addFileOrFolder — метод который позволяет добавлять папку или файл в контейнер;

– unpack — распаковывает указанный файл или папку в директорию;

– unpackAll — распаковывает все данные из контейнера в нужную директорию директорию;

– save — сохраняет изменения в контейнере.

Ниже представлен пример исходного в котором генерируются новый случайный ключ RSA, создается новый контейнер, добавляется папка и происходит сохранение.

```
auto RSAKeys = cc::generateRSAKeys();
auto container =
    cc::Container::openNewContainer("new", RSAKeys.publicKey);
container->addFileOrFolder("test");
container->save();
```

4.4 Сборка библиотеки с помощью CMake

Для библиотеки был написан модуль для системы автоматизации сборки программного обеспечения из исходного кода CMake, он автоматически создает make файл для конкретной операционной системы, прописывая необходимые пути до используемых библиотек и подключаемых заголовочных файлов.

В файле CMakeLists.txt были указаны:

– название проекта;

– пути до исходных файлов и заголовков библиотеки;

– подключаемые библиотеки;

– тип используемого компилятора и других подключаемых библиотек для подключения к Android NDK.

Для сборки, например для linux, необходимо выполнить всего две команды, которые представлены ниже.

```
cmake -G "Unix Makefiles"
make
```

После компиляции библиотеки, будет создана статическая библиотека в корневом каталоге проекта под названием libcryptocontainer.a, которую можно использовать в своем проекте.

4.5 Сборка библиотеки под операционную систему Android

Библиотеки написанные на C, C++ и других языках могут быть скомпилированы под архитектуры ARM, MIPS или x86 в машинный код и подключены с помощью набора Android Native Development (NDK) к приложению. Нативный код вызванный из Java кода работает под виртуальной машиной Dalvik VM с помощью вызова функции System.loadLibrary, которая является частью стандартных классов Java в Android.

Приложение с нативным кодом может быть собрано и установлено с использованием традиционных инструментов разработки, например Android Studio. Тем не менее, в соответствии с документацией Android, NDK не должен использоваться полностью для разработки приложений. Использование NDK увеличивает сложность разработки, в то время как большинству при-

ложений не пойдет на пользу его использование.

Машинный код для работы в NDK может быть скомпилирован с помощью компиляторов GCC или Intel C++ на стандартном ПК. Графическая библиотека, которая использует Android для арбитража и управления доступом к данному устройству называется Skia Graphics Library (SGL). Skia имеет может работать как для Win32 и Unix, позволяя разрабатывать кроссплатформенных приложений. Этот графический движок используется в основе веб-браузера Google Chrome.

В отличие от разработки приложений Java, основанной на IDE, таких как Eclipse, NDK базируется на средствах командной строки и требует использования их вручную для создания, развертывания и отладки приложений. Существует несколько инструментов сторонних производителей позволяют интегрировать NDK в Eclipse и Visual Studio.

Android NDK в данный момент плохо поддерживается, документация на официальном сайте сильно устарела и не подходит для разработки в Android Studio.

В NDK включены компиляторы под различные платформы, т.к. операционная система Android работает на различных платформах, например 32- и 64-бит ARM, x86, MIPS и MIPS64.

4.5.1 Установка NDK

После создания нового проекта, необходимо необходимо подключить NDK указав путь к его местоположению (рисунок reff:23). Если путь к NDK не будет указан или вы не скачали дистрибутив NDK самостоятельно, то Android Studio предложит скачать этот набор при открытии настроек проекта.

После установки необходимо указать в файле build.gradle экспериментальную версию сборщика gradle с поддержкой NDK.

```
dependencies {
    classpath 'com.android.tools.build:
        gradle-experimental:0.7.0'
}
```

Для дальнейшего удобства работы с командной строкой при сборке биб-

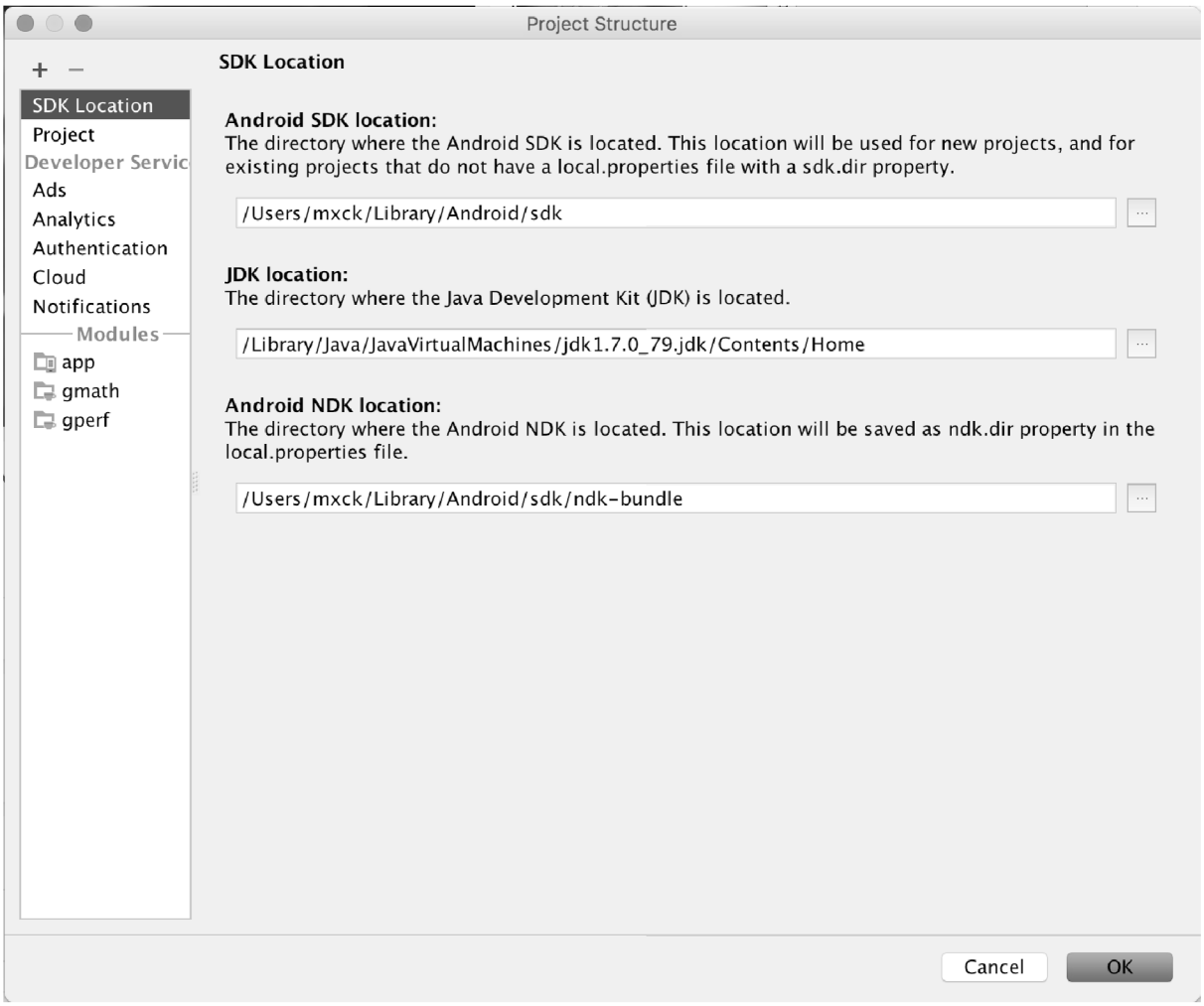

Рисунок 17 – Параметры проекта в Android Studio

лиотек необходимо прописать переменную в среду окружения указывающую на путь к NDK:

```
export ANDROID_NDK_ROOT=/path/to/NDK/
```

4.5.2 Компиляция разработанной библиотеки CryptoContainer

Разработанная библиотека использует систему сборки CMake, и для указания типа платформы и подключаемых библиотек были подготовлены две переменные CMAKE_CXX_COMPILER и STD_LIB. Изменив значение данных переменных, можно скомпилировать библиотеку готовую к подключению к NDK для выбранной платформы.

После изменения переменных, для сборки достаточно выполнить следующую команду.

```
cmake -G "Unix Makefiles" || make
```

4.5.3 Компиляция библиотеки CryptoPP

Библиотека CryptoPP имеет встроенные средства для сборки под операционную систему Android, но как показала практика они либо устарели, либо не работают должным образом.

В корне каталога библиотеки CryptoPP расположен испольняемый файл setenv-android.sh, он осуществляет настройку среды окружения, исходя от указанной платформы компилирования и использованной стандартной библиотеки, gnustd или stdlib. При указания платформы и типа стандартной библиотеки, происходит установка путей относительно переменной из среды ANDROID_NDK_ROOT, которую мы установили выше:

– платформа под которую будет компилироваться библиотека;

– версия Android SDK API;

– путь до компилятора выбранной платформы;

– флаги необходимые для компиляции под выбранную платформу (например флаг включения исключений);

– пути до заголовочных файлов выбранной стандартной библиотеки;

– путь до скомпилированной версий стандартной библиотеки;

Как оказалось для компиляции необходимы дополнительные заголовочные файлы, которые отсутствуют в стандартной библиотеке встроенной в NDK. Для их подключения необходимо исправить файл setenv-android.sh перед использованием добавив в конце к флагам компиляции путь к заголовочной файлам стандартной библиотеки встроенной в NDK.

```
-
I/\$ANDROID\_NDK\_ROOT/sources/cxx-stl/gnu-libstdc++/4.9/li
bs/armeabi-v7a/include
```

Далее необходимо выполнить команду которая очистит папку от уже скомпилированных файлов и других временных файлов.

```
make -f GNUmakefile-cross distclean
```

После очистки каталога можно приступать к компилированию статической библиотеки.

```
make -f GNUmakefile-cross static
```

На выходе мы получаем файл libcryptopp.a, представляющий собой статическую библиотеку готовую к подключению в Android Studio для выбранной вами платформы.

4.5.4 Компиляция библиотек из набора Boost

Стандартная библиотека filesystem из набора boost изначально не работает под системой Android, т.к. использует вызовы функции std::codecvt из стандартной библиотеки, но эта функция не реализована в gnustd, т.к. встроенная библиотека gnustd отстает от текущих стандартов C++.

Чтобы это исправить необходимо самостоятельно заменить вызов этой функции на другой, воспользоваться общедоступными патчами из набора Booost-for-Android [45] или воспользоваться сторонней версией NDK с поддержкой данных вызовов — CrystalX [46]. В наборе инструментов CrystalX уже встроена последняя библиотека Boost и используется более современная версия gnustd.

В наборе библиотек Boost имеется своя система сборки называемая b2. Для сборки и первоначальной настройки библиотеки необходимо выполнить команду следующую команду.

```
./bootstrap.sh--
  with-libraries=serialization,filesystem,program_options,
     system
```

После выполнения этой команды будет создан исполняемый файл b2 который и занимается сборкой библиотеки под различные платформы.

Далее необходимо создать файл user-config.jam в корне пользовательского каталога, в нем хранятся пользовательский настройки, например пути до библиотек и используемых компиляторов.

```
import option ;
local NDK = [ os.environ ANDROID_NDK_ROOT ] ;

using gcc : arm : $(NDK)/toolchains/arm-linux-androideabi-4.9
    /prebuilt/darwin-x86_64/bin/arm-linux-androideabi-g++ ;

option.set keep-going : false ;
```

Был подгружен модуль для опций сборки, указан путь до NDK, полученный из переменной пользовательского окружения, указан путь до компилятора платформы ARM, указываем что при ошибке компиляции — происходит её остановка.

После сохранения файла настроек, можно приступать к компиляции библиотеки. В качестве параметров к следующей команде мы указываем: целевую платформу, путь к скомпилированной стандартной библиотеки, путь к заголовочным файлам стандартной библиотеке и путь куда будет сохранена скомпилированная библиотека.

```
./b2 --reconfigure toolset=gcc-arm \
    include=$ANDROID_NDK_ROOT/sources/cxx-stl/
            gnu-libstdc++/4.9/include
    include=$ANDROID_NDK_ROOT/sources/cxx-stl/
            gnu-libstdc++/4.9/libs/armeabi/include
    include=$ANDROID_NDK_ROOT/platforms/android-21/
            arch-arm/usr/include
    install --libdir=stage/lib/arm
```

После выполнения команды в указанной папке будут находится скомпилированные файлы указанных библиотек из набора Boost, готовые для подключения к проекту в Android Studio.

4.5.5 Подключение библиотек к проекту

Создадим в корне проекта папку libs, в ней создадим подкаталоги для библиотек, в каждом из них создадим каталог lib для хранения скомпилированной статической библиотеки и include для хранения заголовочных файлов библиотеки. После этого скопируем скомпилированные библиотеки и заголовочные файлы по нужным каталогам.

Для подключения готовых статических библиотек, необходимо явно указать их нахождения системе сборки Gradle. Добавим в файл отвечающий за сборку проекта build.gradle данные о расположении наших библиотек.

```
repositories {
    libs(PrebuiltLibraries) {
        gmath {
            headers.srcDir "libs/cryptopp/include"
```

```
            binaries.withType(StaticLibraryBinary) {
                staticLibraryFile = file("libs/cryptopp/
                    lib/${targetPlatform.getName()}/cryptopp.a
                        ")
            }
        }
    }
```

После этого необходимо указать тип подключаемых библиотек для системы сборки, динамическая или статическая, после чего он автоматически подгрузит данную библиотеки и она будет доступна для использования в нативном исходном коде.

```
sources {
    main {
        jni {
            dependencies {
                library 'cryptopp' linkage 'static'
                library 'boost' linkage 'static'
                library 'cryptocontainer' linkage 'static'
            }
        }
    }
}
```

После обновления файла build.gradle, Android Studio автоматически подгрузит все заголовочные файлы и библиотеки. Теперь можно использовать подключенные нами библиотеки в коде Java.

Создадим метод у главного класса, который будет кодировать строку с помощью RSA.

```
public class HelloJni extends Activity {
...
    public native String encodeStringFromJNI();
...
}
```

Ключевое слово native указывает, что метод необходимо подгрузить из исходных файлов написанных на C/C++. После создания метода, Android Studio предложит имплементировать данный метод, после этого будет создан файл hello-jni.cpp.

```
#include <string.h>
#include <jni.h>
#include <CryptoContainer/rsa.h>

#define LOGI(...) \
  ((void)__android_log_print(ANDROID_LOG_INFO, "hello-jni::",
    __VA_ARGS__))

extern "C" JNIEXPORT jstring JNICALL
Java_com_hellojni_MainActivity_encodeStringFromJNI(
    JNIEnv *env, jobject thiz) {

    auto publicKey =
    cc::loadKeyFromFile<CryptoPP::RSA::PublicKey>("key.pub");

    std::string cryptedSign = cc::encryptStringRSA(
        *publicKey, "Hello world!");

    LOGI("calculation time: %" PRIu64, cryptedSign.c_str());

    return env->NewStringUTF(report.str().c_str());
}
```

После компиляции и выполнения программы в терминале разработчика будет выведено зашифрованное сообщение "Hello World!".

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы создано программное обеспечение позволяющее создавать зашифрованные файловые контейнеры, сочетающие в себе безопасность и простоту использования.

Для достижения поставленной цели в ходе выполнения работы были решены следующие задачи:

– произведен анализ существующего программного обеспечения, выбор средств реализации криптографического файлового контейнера;

– разработана спецификация формата контейнера для безопасного хранения данных;

– разработан API для использования возможностей созданного файлового контейнера

– реализована кроссплатформенная библиотека для работы с криптографическим файловым контейнером;

– изучены особенности библиотеки CryptoPP, набора библиотек Boost и стандартной шаблонной библиотеки языка C++ (C++14);

– проведена проверка работоспособности библиотеки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер. — Москва : Триумф, 2002. — 816 с.

2. Ватолин, Д. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк. — Москва : ДИАЛОГ-МИФИ, 2002. — 384 с.

3. GNU tar [Электронный ресурс] : GNU tar - Summary. — Режим доступа: http://savannah.gnu.org/projects/tar.

4. DEFLATE [Электронный ресурс] : DEFLATE Compressed Data Format Specification version 1.3. — Режим доступа: https://tools.ietf.org/html/rfc1951.

5. gzip [Электронный ресурс] : The gzip home page. — Режим доступа: http://www.gzip.org.

6. bzip2 [Электронный ресурс] : bzip2 and libbzip2. — Режим доступа: http://bzip.org.

7. Burrows, M. A block sorting lossless data compression algorithm / M. Burrows, D. Wheeler. — Digital Equipment Corporation, 1994. — 124 с.

8. ZIP [Электронный ресурс] : ZIP File Format Specification. — Режим доступа: https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT.

9. PKZip [Электронный ресурс] : PKZip Must Open Up. — Режим доступа: http://brianlivingston.com/eweek/article2/0,4149,1257562,00.html.

10. WinZip [Электронный ресурс] : WinZip for Windows. — Режим доступа: http://winzip.com/.

11. Corel [Электронный ресурс] : Corel home page. — Режим доступа: http://www.corel.com/.

12. 7-Zip [Электронный ресурс] : 7-Zip is a file archiver with a high compression ratio. — Режим доступа: http://7-zip.org.

13. WinRAR [Электронный ресурс] : WinRAR is a powerful archive manager utils. — Режим доступа: http://rarlab.com.

14. WinRAR описание формата [Электронный ресурс] : WinRAR is a powerful archive manager utils. — Режим доступа: http://www.rarlab.com/rar_add.htm.

15. Scarfone, K. Guide to Storage Encryption Technologies for End User Devices / K. Scarfone, M. Sexton. — National Institute of Standards and Technology, 2007. — 40 с.

16. Microsoft Security [Электронный ресурс] : Microsoft Security Bulletin MS15-122 - Important. — Режим доступа: https://technet.microsoft.com/library/security/MS15-122.

17. FUSE [Электронный ресурс] : The reference implementation of the Linux FUSE (Filesystem in Userspace) interface — Режим доступа: https://github.com/libfuse/libfuse.

18. FireVault [Электронный ресурс] : Использование FileVault для шифрования загрузочного диска на компьютере Mac. — Режим доступа: https://support.apple.com/kb/HT4790?viewlocale=ru_RU.

19. VeraCrypt [Электронный ресурс] : Encryption Algorithms. — Режим доступа: https://veracrypt.codeplex.com/wikipage?title=Encryption%20Algorithms.

20. VeraCrypt hash [Электронный ресурс] : the website title. — Режим доступа: https://veracrypt.codeplex.com/wikipage?title=Hash%20Algorithms.

21. WAV [Электронный ресурс] : Hash Algorithms. — Режим доступа: http://www-mmsp.ece.mcgill.ca/documents/AudioFormats/WAVE/Docs/riffmci.pdf.

22. MPEG [Электронный ресурс] : 3GPP2 File Formats for Multimedia Services. — Режим доступа: http://www.3gpp2.org/Public_html/specs/C.S0050-B_v1.0_070521.pdf.

23. MKV [Электронный ресурс] : Matroska Media Container - Homepage. — Режим доступа: https://www.matroska.org.

24. MKV spec [Электронный ресурс] : Matroska Media Container - Specifications. — Режим доступа: https://www.matroska.org/technical/specs/index.html#track.

25. TIFF [Электронный ресурс] : Tagged Image File Format. — Режим доступа: http://partners.adobe.com/public/developer/tiff/index.html.

26. William, V. Encyclopedia of Graphics File Formats / V. William. — O'Reilly, 1996. — 384 с.

27. EXE [Электронный ресурс] : Microsoft PE and COFF Specification. — Режим доступа: https://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx.

28. ISO [Электронный ресурс] : Standard ECMA-119 Volume and File Structure of CDROM for Information Interchange. — Режим доступа: http://www.ecma-international.org/publications/standards/Ecma-119.htm.

29. SQlite [Электронный ресурс] : SQLite Homepage. — Режим доступа: http://www.sqlite.org.

30. SQlite deps [Электронный ресурс] : SQLite Homepage - specs.— Режим доступа: https://sqlite.org/mostdeployed.html.

31. SQlite format [Электронный ресурс] : Most Widely Deployed and Used Database Engine.— Режим доступа: https://www.sqlite.org/fileformat2.html.

32. Мейволд, Э. Безопасность сетей / Э. Мейволд. — Москва : Эком, 2011. — 528 с.

33. DES [Электронный ресурс] : DES specs.— Режим доступа: http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf.

34. AES [Электронный ресурс] : fips-197 specifictaions.— Режим доступа: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

35. Kumar, S. How to Break DES for Euro 8,980 / S. Kumar. — SHARCS, 2006. — 96 с.

36. GOST [Электронный ресурс] : Межгосударственный совет по стандартизации, метрологии и сертификации (МГС) Содружества Независимых Государств (СНГ).— Режим доступа: http://www.easc.org.by/russian/mgs_org.php.

37. Shorin, V. Linear and Differential Cryptanalysis of Russian GOST / V. Shorin, V. Jelezniakov. — Electronic Notes in Discrete Mathematics, 2001. — 547 с.

38. Ростовцев, А.Г. О стойкости ГОСТ 28147–89. Проблемы информационной безопасности / А.Г. Ростовцев, Е.Б Маховенко. — Москва : Компьютерные системы, 2003. — 84 с.

39. ГОСТ стандарт шифрования [Электронный ресурс] : Технический комитет по стандартизации (ТК 26) "Криптографическая защита информации"О деятельности по международной стандартизации алгоритма шифрования ГОСТ 28147-89.— Режим доступа: https://www.tc26.ru/.

40. Nikolić, I. Distinguisher and Related-Key Attack on the Full AES-256 / I, Nikolić. — Advances in Cryptology, 2009. — 349 с.

41. AES hack [Электронный ресурс] : Related-key Cryptanalysis of the Full AES-192 and AES-256.— Режим доступа: https://eprint.iacr.org/2009/317.pdf.

42. Performance Comparisons of the AES submissions / B. Schneier [и др.]. — Digital Equipment Corporation, 1999. — 20 с.

43. Modes [Электронный ресурс] : Режимы шифрования.— Режим доступа: http://citforum.ru/security/cryptography/rejim_shifrov/.

44. OpenSSL spread [Электронный ресурс] : Critical crypto bug in OpenSSL opens two-thirds of the Web to eavesdropping.— Режим доступа: http://arstechnica.com/security/2014/04/critical-crypto-bug-in-openssl-opens-two-thirds-of-the-web-to-eavesdropping/.

45. Boost for Android [Электронный ресурс] : Boost for Android repository.— Режим доступа: https://github.com/moritz-wundke/Boost-for-Android/.

46. CrystaX [Электронный ресурс] : Набор инструментов для C/C++ разработки под Android.— Режим доступа: https://www.crystax.net.

ПРИЛОЖЕНИЕ А

Исходный код программы

Заголовочный файл container.hpp

```
#ifndef INCLUDE_CRYPTOCONTAINER_CONTAINER_HPP_
#define INCLUDE_CRYPTOCONTAINER_CONTAINER_HPP_

#include <cryptopp/hex.h>
#include <cryptopp/rsa.h>

#include <map>
#include <set>
#include <string>
#include <list>
#include <utility>

#include <CryptoContainer/aes.hpp>
#include <CryptoContainer/utils.hpp>
#include <boost/archive/text_oarchive.hpp>
#include <boost/archive/text_iarchive.hpp>

namespace cc {
struct DirectoryEntry {
    std::string filename;
    CryptoPP::SecByteBlock key;
    CryptoPP::SecByteBlock iv;
    int64_t startPos;
    int64_t sizeInBytes;

    friend class boost::serialization::access;

    template<class Archive>
    void save(Archive & ar, const unsigned int version) const{
        // NOLINT
        std::string aKey = cc::SecByteBlockToString(key);
        std::string aIV = cc::SecByteBlockToString(iv);

        ar << filename;
```

```
        ar << aKey;
        ar << aIV;
        ar << startPos;
        ar << sizeInBytes;
    }

    template<class Archive>
    void load(Archive & ar, const unsigned int version) { //
       NOLINT
        std::string aKey;
        std::string aIV;
        ar >> filename;
        ar >> aKey;
        ar >> aIV;
        ar >> startPos;
        ar >> sizeInBytes;

        key = cc::SecByteBlockFromString(aKey);
        iv = cc::SecByteBlockFromString(aIV);
    }

    template<class Archive>
    void serialize(Archive & ar, const unsigned int version) {
        // NOLINT
        boost::serialization::split_member(ar, *this, version)
           ;
    }
};

class Container {
 private:
    static int signSize;

    Container();

    std::filebuf fileBuf;
    std::unique_ptr<std::iostream> fileStream;  // @@ Remove
       this
```

```
    std::map<std::string, cc::DirectoryEntry> directory;

    // This need to set after each file save or read
    int64_t lastWritePos;

    std::unique_ptr<CryptoPP::RSA::PublicKey> publicKey;

    // First - original path, second - stored path
    std::map<std::string, std::string> filesToAdd;
    void addFileToQueue(std::string path, std::string
       relativeTo = "");

    void writeFile(std::pair<std::string, std::string> paths);
    void writeDirectoryAndSign();
    void unpackFileToDisk(const DirectoryEntry& entry, std::
       string targetPath);

 public:
    void addFileOrFolder(std::string path);
    void unpack(std::string path, std::string targetDirectory
       = "");
    void unpackAll(std::string path = "");

    // Save changes
    void save();

    const std::map<std::string, cc::DirectoryEntry>&
       getDirectory() const;

    static std::unique_ptr<Container> openExistedContainer(
        std::string path,
        CryptoPP::RSA::PublicKey publicKey,
        CryptoPP::RSA::PrivateKey privateKey);

    static std::unique_ptr<Container> openNewContainer(
        std::string path, CryptoPP::RSA::PublicKey publicKey);
};
}  // namespace cc

#endif  // INCLUDE_CRYPTOCONTAINER_CONTAINER_HPP_
```

Заголовочный файл rsa.hpp

```
#ifndef INCLUDE_CRYPTOCONTAINER_RSA_HPP_
#define INCLUDE_CRYPTOCONTAINER_RSA_HPP_

#include <cryptopp/rsa.h>
#include <cryptopp/osrng.h>
#include <cryptopp/files.h>
#include <sstream>
#include <string>
#include <utility>

namespace cc {
template <typename T>
struct isRSAKey {
};

template <>
struct isRSAKey<CryptoPP::RSA::PublicKey> {
    typedef CryptoPP::RSA::PublicKey type;
};

template <>
struct isRSAKey<CryptoPP::RSA::PrivateKey> {
    typedef CryptoPP::RSA::PrivateKey type;
};

template <typename T>
inline std::string RSAKeyToString(const typename isRSAKey<T>::
   type& key) {
    CryptoPP::ByteQueue queue;
    key.Save(queue);

    std::string result;
    CryptoPP::StringSink sink(result);
    queue.CopyTo(sink);
    return result;
}

template <typename T>
inline typename isRSAKey<T>::type RSAKeyFromString(const std::
```

```
   string& str) {
    CryptoPP::StringSource stringSource(str, true);
    CryptoPP::ByteQueue queue;
    stringSource.TransferTo(queue);

    typename isRSAKey<T>::type key;
    key.Load(queue);
    return key;
}

template<typename T>
inline typename isRSAKey<T>::type loadKeyFromFile(std::string
   filename) {
    std::ifstream in(filename);
    std::stringstream buffer;
    buffer << in.rdbuf();
    return RSAKeyFromString<typename isRSAKey<T>::type>(buffer
       .str());
}

template<typename T>
inline void saveKeyToFile(std::string filename,
                          typename isRSAKey<T>::type key) {
    std::filebuf buf;
    buf.open(filename,
        std::ios::trunc | std::ios::out | std::ios::binary);
    std::ostream os(&buf);
    os << RSAKeyToString<typename isRSAKey<T>::type>(key);
}

std::pair<CryptoPP::RSA::PrivateKey, CryptoPP::RSA::PublicKey>
    generateRSAKeys();

std::string encryptStringRSA(CryptoPP::RSA::PublicKey
   publicKey,
                             std::string input);

std::string decryptStringRSA(CryptoPP::RSA::PrivateKey
   privateKey,
                             std::string input);
```

```
}  // namespace cc

#endif  // INCLUDE_CRYPTOCONTAINER_RSA_HPP_
```

Заголовочный файл utils.hpp

```
#ifndef INCLUDE_CRYPTOCONTAINER_UTILS_HPP_
#define INCLUDE_CRYPTOCONTAINER_UTILS_HPP_

#include <cryptopp/rsa.h>
#include <string>

namespace cc {
inline std::string SecByteBlockToString(const CryptoPP::
   SecByteBlock& str) {
    return std::string(reinterpret_cast<const char*>(str.data
       ()),
                       str.size());
}

inline CryptoPP::SecByteBlock SecByteBlockFromString(const std
   ::string& str) {
    return CryptoPP::SecByteBlock(reinterpret_cast<const byte
       *>(str.data()),
                       str.size());
}
}  // namespace cc

#endif  // INCLUDE_CRYPTOCONTAINER_UTILS_HPP_
```

Заголовочный файл crypt.hpp

```
#ifndef INCLUDE_CRYPTOCONTAINER_AES_HPP_
#define INCLUDE_CRYPTOCONTAINER_AES_HPP_

#include <cryptopp/aes.h>
#include <cryptopp/files.h>
#include <cryptopp/filters.h>
#include <cryptopp/modes.h>
#include <cryptopp/osrng.h>  // Random
```

```
#include <stdint.h>

#include <memory>
#include <string>
#include <utility>

namespace cc {
// Generate a random key using maximum length
CryptoPP::SecByteBlock generateRandomAESKey();

// Generate a random initialization vector
CryptoPP::SecByteBlock generateRandomAES_IV();

/*
  Classes used to write from stream to stream.
*/
class CryptBase {
 protected:
    CryptoPP::StreamTransformationFilter* filter;
    std::unique_ptr<CryptoPP::FileSource> fileSource;
    int64_t bytesCoded = 0;
    bool complete = false;
    virtual void atEOF() = 0;
 public:
    const bool& getComplete() const;
    const int64_t& getBytesCoded() const;
    void pumpAll();

    CryptBase() {}
    virtual void pump() = 0;
    virtual ~CryptBase();
};

class Encrypt : public CryptBase {
 private:
    typedef CryptoPP::CBC_Mode<CryptoPP::AES>::Encryption
       Encryption;
    Encryption encryption;
    void atEOF();
 public:
```

```
    virtual void pump();
    Encrypt(CryptoPP::SecByteBlock key,
                CryptoPP::SecByteBlock iv,
                std::istream* source,
                std::ostream* target);
    ~Encrypt();
};

class Decrypt : public CryptBase {
 private:
    typedef CryptoPP::CBC_Mode<CryptoPP::AES>::Decryption
       Decryption;
    Decryption decryption;
    void atEOF();
    int64_t bytesToDecrypt;
 public:
    virtual void pump();
    Decrypt(CryptoPP::SecByteBlock key,
                CryptoPP::SecByteBlock iv,
                std::istream* source,
                std::ostream* target,
                int64_t bytesNeedToDecrypt);
    ~Decrypt();
};
}  // namespace cc

#endif  // INCLUDE_CRYPTOCONTAINER_AES_HPP_
```