ВВЕДЕНИЕ

На текущий момент многие обеспокоены защитой передаваемой ими информации по открытым каналам связи в повседневной жизни. Это может быть как личная переписка, так и какая-либо конфиденциальная информация.

Существует множество средств шифрования данных, но для их работы необходимо участие пользователя. При использовании таких средств шифрование происходит явным образом. Многие из реализованных программ предназначены для шифрования электронных писем. Самыми популярными являются Messenger 400 и ProtonMail. Средства шифрования требуют установки нескольких программ и точной настройки используемых средств. Самым оптимальным решением для узкоспециализированных задач является написание своего инструмента, который будет удовлетворять требуемым условиям и режимам работы.

Так же о подобной проблеме задумаются и разработчики современных мессенджеров. В таких приложениях реализовано «прозрачное» шифрование. Пользователь не принимает никакого участия в процессе шифрования информации.

Но многие продолжаются использовать старые приложения, в которых не реализована защита информации и работы по улучшению больше не ведутся. В таких случаях требуется дополнительная программа, которая обеспечит шифрование данных перед передачей по каналу связи.

В соответствии с вышеизложенными причинами были сформулированы и решены следующие задачи:

* Анализ существующих решений по выборочному шифрованию;
* Проектирование дизайна и архитектуры программы;
* Создание дизайна и архитектуры программы;
* Обработка полученных данных;
* Разработка подсистемы шифрования и работы с ключами шифрования.

# 1 ПРОГРАММНЫЕ СРЕДСТВА ШИФРОВАНИЯ ДАННЫХ

В современном мире для передачи данных в зашифрованном виде могут использоваться как готовые программные решения, так и разработанные пользователем для себя с помощью одного из языков программирования. Достоинства уже реализованных продуктов в том, что над их качеством трудилась команда разработчиков, но так же там могут быть реализованы те функции, которые не нужны конкретному пользователю. В таком случае можно написать своё приложение, которое будет выполнять узкий круг требований или же реализовывать специфичный алгоритм обмена данными, но за качество и стойкость алгоритмов шифрования уже отвечает пользователь.

## 1.1 Готовые программные решения

Многие из реализованных программ предназначены для шифрования электронных писем. Самыми популярными являются Messenger 400 и ProtonMail.

### 1.1.1 Messenger 400

Защищенная электронная почта Х.400 на базе электронного почтамта Messenger 400 (М400) фирмы Infonet Software Solutions (ISS) предназначена для предоставления абонентам почтовых услуг по обмену защищенными (зашифрованными и подписанными) сообщениями с использованием механизма двусторонней аутентификации абонента на почтамте и почтамта на абонентском пункте, реализованного с помощью ЭЦП.

Из возможностей стоит отметить, что защищенная электронная почта Х.400 способна устойчиво функционировать в различных сетях, в том числе на недорогих низкоскоростных линиях, и использовать различные протоколы связи, включая Х.25, X.28, TCP/IP, IPX/SPX и другие. Система хранения и передачи сообщений поддерживает развитые средства маршрутизации, обеспечивающие возможность оптимальной производительности и настройки с целью уменьшения стоимости коммуникационных услуг. Использование сервисов, предусмотренных стандартом Х.400, а именно квитанций о доставке и прочтении, гарантированной доставки и маршрутизации, является большим преимуществом данной почтовой системы по сравнению с другими системами.

В состав защищенной электронной почты X.400 входят программные и аппаратно-программные средства:

1) Защищенный почтовый сервер;

2) Защищенные СКЗИ абонентские пункты (АП);

3) Центр управления ключевой системой.

АП предназначены для обмена зашифрованными и подписанными со-общениями и поддержки механизма двухсторонней аутентификации при об-мене через электронный почтамт. АП позволяют в соответствии с рекомендациями X.400 обмениваться зашифрованными и подписанными почтовыми сообщениями между пользователями, зарегистрированными в системе защищенной электронной почты X.400 с использованием механизма двухсторонней аутентификации. АП позволяют оператору создавать, рассылать и обрабатывать электронные сообщения в ручном и автоматическом режимах, редактировать конверты, работать с папками, работать по заданному сценарию без участия пользователя, сканировать сетевой разделяемый диск и обрабатывать сценарии. АП также позволяют выполнять ряд специфических задач, связанных с обеспечением безопасности и конфиденциальности данных. На АП может быть зарегистрировано неограниченное число пользователей. Разграничение доступа к ресурсам АП обеспечивается применением паролей. Часть пользователей может быть зарегистрирована в качестве администраторов и иметь доступ ко всем ресурсам АП. Для АП разработан специализированный помехоустойчивый протокол LAPS, позволяющий работать на каналах связи плохого качества и обеспечивающий механизм двухсторонней аутентификации, а также протокол ELINK, обеспечивающий работу АП в локальных сетях.

### 1.1.2 ProtonMail

Особенностями безопасности ProtonMail являются:

1) Сквозное шифрование

Смысл сквозного шифрования состоит в том, что прочитать сообщение не сможет никто, кроме нужного получателя Сообщения хранятся на серверах ProtonMail в зашифрованном виде. Также в зашифрованном виде они передаются между серверами и пользовательскими устройствами. Сообщения между пользователями ProtonMail также передаются в зашифрованном виде внутри защищённой сети серверов компании. Proton Technologies. И поскольку данные в любом случае зашифрованы, риск перехвата сообщений в значительной мере устранён.

2) Нулевой доступ к данным пользователя

Архитектура нулевого доступа ProtonMail означает, что данные зашифрованы таким способом, который делает их недоступными для владельцев сервера, на котором хранится информация. Данные шифруются на стороне клиента с применением ключа, доступа к которому ни у кого нет. Это значит, что владелец сервера не имеем технической возможности расшифровать сообщения, и как результат, не можем передать ваши данные третьим лицам. С ProtonMail конфиденциальность гарантирована математически.

3) Шифрование, основанное на открытом исходном коде

ProtonMail применяет только безопасные реализации AES, RSA, а также OpenPGP. Кроме того, все применяемые шифровальные библиотеки являются открытым исходным кодом. Применяя библиотеки с открытым исходным кодом, гарантируется, что в используемых алгоритмах шифрования нет тайно встроенных "закладок". ПО с открытым исходным кодом ProtonMail было тщательно проверено экспертами в области безопасности со всего мира, чтобы гарантировать наивысшие степени защиты.

### 1.1.2 Delta chat

Delta Chat - децентрализованный мессенджер для Android, iOS, Linux, Mac, функционирующий поверх стека протоколов E-mail.

Ключевые особенности DeltaChat:

* приложение подключается к любому серверу электронной почты, выбранному вами;
* распределённая связь осуществляется через федерации почтовых серверов;
* адресатом может быть любой владелец электронной почты, даже если он не установил себе Delta Chat;
* при наличии возможности для сквозного шифрования переписки применяется механизм Autocrypt;
* отображение отметок о прочтении и статусе доставки и быстрые уведомления функционируют по протоколу Push-IMAP;
* полностью открытый исходный код приложения под лицензией GPLv3 и протоколы, основанные на стандартах.

## 1.2 Реализация собственного инструмента

### 1.2.1 VPN

Самым простым способом является использование VPN-туннеля между двумя пользователями. Никто, кроме участников сети, не сможет прочитать информацию, передаваемую по VPN. Одним из примеров является OpenVPN, он имеет возможность создания виртуального сетевого TAP-интерфейса способного инкапсулировать Ethernet-фреймы, а не только более высокоуровневые IP-пакеты.

### 1.2.2 Разработка средства шифрования с помощью языка программирования

Так же возможна реализация шифрования трафика канального уровня модели OSI с помощью языка программирования. Преобладающая часть программно-аппаратных средств по шифрованию трафика работает на третьем и более высоких сетевых уровнях. Логические структуры и организация сетей третьего уровня, зачастую, отдалённо коррелирует со структурой низлежащей сети второго. Из-за этого, создание виртуальных частных криптографически защищённых сетей является не тривиальной задачей, затрагивающей всю систему маршрутизации.

Прозрачное шифрование трафика именно на втором уровне избавит от изменений в маршрутизации, уменьшит накладные расходы от туннелированния сетевых пакетов. Проще организовать обмен криптографическими ключами между подразделениями одной организации и сократить сложные протоколы и процедуры установления общего ключа. Задержки связи из-за использования симметричной криптографии со статическими ключами отсутствуют. Кроме того, объёмы трафика останутся неизменны.

## 1.3 Выводы и результаты

Средства шифрования требуют установки нескольких программ и точной настройки используемых средств. Самым оптимальным решением для узкоспециализированных задач является написание своего инструмента, который будет удовлетворять требуемым условиям и режимам работы.

# 2 РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

Для реализации шифрования передаваемых данных нам необходимо средство перехвата этих самых данных на уровне операционной системы. А так же возможность получать зашифрованные данные от других участников общения и производить их дешифровку.

## 2.1 Выбор средств разработки и системных программных средств

Так как Windows является одной из популярных используемых операционных систем, то программа должна поддерживать данную операционную систему. Для реализации перехвата отправляемых и полученных данных будем использовать библиотеку WinAPI и соответственно язык программирования C для работы с ней. Разработка программы будет происходить в среде разработки Visual Studio 2019 community edition, как в удобной и современной IDE для работы с языком программирования C.

## 2.2 Требования к разрабатываемой программе

Программа должна реализовывать все требования к системе «прозрачного» шифрования и не требовать вмешательства и настройки от конечного пользователя. Шифроваться должны все данные отправляемые через программы, установленные на рабочей машине конечного пользователя.

## 2.3 Структура программы

Программа должна состоять из 3 частей:

1. Шифратор/дешифратор и средство по работе с ключами шифрования;
2. Средство для преобразования данных;
3. Средство по работе с операционной системой.

Шифратор/дешифратор отвечает за корректное шифрование отправляемых и дешифрование полученных пользовательских данных.

Средство по работе с ключами шифрования отвечает за генерацию корректных ключей шифрования и поддержания актуального их состояния между двумя конкретными пользователями.

Средство для преобразования данных необходимо для обработки бинарных данных для корректной работы с операционной системой и отображению их пользователю.

Средство по работе с операционной системой необходимо, чтобы получать данные введенные пользователем для дальнейшей их пересылки, а так же отображения полученных от других пользователей сообщений.

### 2.3.1 Основные модули

Программа состоит из следующих модулей:

1) Шифратор – предназначен для шифрования данных заданным алгоритмом шифрования с использованием сгенерированного ключа шифрования;

2) Дешифратор – предназначен для дешифрования данных исходя из заданного алгоритма шифрования с использованием сгенерированного ключа дешифрования;

3) Средство захвата отправляемых данных – необходимо для захвата введённых пользовательских данных, обработки их шифратором и возвращения программе для дальнейшей его работы по их пересылке;

4) Средство отображения полученных данных – необходимо для перехвата полученных данных, их дешифрования и дальнейшего их отображения пользователю;

5) Средство генерации ключей шифрования/дешифрования – необходимо для генерации корректных ключей, с помощью которых возможно произвести процедуру шифрования или дешифрования;

6) Средство синхронизации ключей шифрования/дешифрования – необходимо для правильной работы программы и возвращения корректных результатов процедур шифрования/дешифрования даже после появления и устранения неисправностей в работе приложения или в режиме работы, когда свои данные шифрует только один из собеседников.

## 2.4 Описание интерфейса пользователя

Программа должна работать в скрытом режиме и не выдавать своего присутствия в системе. Графический интерфейс минимален.

## 2.5 Выводы и результаты

Программа реализована в стиле системы «прозрачного» шифрования и обработки пользовательских данных на уровнях получения сообщения из поля ввода и отображения полученного сообщения в специальном окне или передачи текста на отправку по каналу связи и приёма из канала связи ответа собеседника, в зависимости от реализации сетевого взаимодействия конкретной программы.

Для наименьшего вмешательства пользователя в работу программы было выбрано взаимодействие через горячие клавиши.

Алгоритм работы программы при шифровании:

* Обработка нажатой горячей клавиши;
* Сохранение состояния буфера обмена;
* Имитация горячей клавиши «Вырезать»;
* Чтение текста из буфера обмена;
* Шифрование данных;
* Запись данных в буфер;
* Имитация горячей клавиши «Вставить»;
* Восстановление состояния буфера обмена.

Алгоритм работы программы при шифровании:

* Обработка нажатой горячей клавиши;
* Сохранение состояния буфера обмена;
* Имитация горячей клавиши «Копировать»;
* Чтение текста из буфера обмена;
* Дешифрование данных;
* Отображение окна с дешифрованным сообщением;
* Восстановление состояния буфера обмена.

Работа с сессионными ключами шифрования будет происходить следующим образом:

* Получаем текст шифруемого сообщения;
* Генерируем из мастер-ключа и «одноразового» случайного числа значения на вход Argon2;
* Получаем из Argon2 сессионный ключ;
* Шифруем полученным сессионным ключом сообщение;
* Отправляем пользователю строку, состоящую из «одноразового» случайного числа и шифротекста.

Работа с сессионными ключами дешифрования будет происходить следующим образом:

* Получаем текст зашифрованного сообщения;
* Получаем из строки «одноразовое» случайное число;
* Генерируем из мастер-ключа и «одноразового» случайного числа значения на вход Argon2;
* Получаем из Argon2 сессионный ключ;
* Дешифруем полученным сессионным ключом сообщение;
* Отображаем пользователю полученное сообщение.

Для гарантии того, что зашифрованные данные отобразятся у любого пользователя, после шифрования и перед дешифрованием происходит кодирование/декодирование Base64.

# 3 ИСПОЛЬЗУЕМЫЕ АЛГОРИТМЫ И ФУНКЦИИ

## 3.1 AES-128

Rijndael – симметричный блочный шифр, который может обрабатывать данные блоками по 128 бит, используя ключи шифрования длиной 128, 192 и 256 бит. Rijndael спроектирован так, что позволяет использовать и другие длины блоков и ключей.

Входом и выходом алгоритма AES являются последовательности из 128 бит. Эти последовательности рассматриваются как блоки. Количество бит в блоке будет называться длиной блока. Ключ шифрования для алгоритма AES – это последовательность из 128, 192 или 256 бит.

Базовым элементом, которым оперирует алгоритм AES, является байт – последовательность из восьми бит, обрабатываемых как единое целое. Описанные ранее последовательности бит, обрабатываются как массивы байт.

Внутри алгоритма AES выполняются операции над двумерным массивом байт, называемым матрицей состояния. Матрица состояния образована четырьмя строками, каждая из которых содержит Nb байт, где Nb – длина блока, делённая на 32. В матрице состояния обозначается символом s.

В начале процедур шифрования и дешифрования, вход копируется в матрицу состояния. Затем процедуры шифрования и дешифрования управляют матрицей состояния, после чего её финальное значение копируется в выход.

Четыре байта в каждом столбце матрицы состояния образуют 32-битные слова. Номер строки является для этих четырёх байт индексом в пределах каждого слова. Таким образом, матрица состояния может быть интерпретирована как одномерный массив 32-битных слов (столбцов).

В алгоритме AES все байты рассматриваются в качестве элементов конечного поля. Элементы конечного поля можно складывать и умножать. Но эти операции отличаются от принятых для обычных чисел.

### 3.1.1 Сложение

Сложение двух элементов в конечном поле производится путём «сложения» коэффициентов при соответствующих степенях многочленов, представляющих эти два элемента. Сложение выполняется с помощью операции XOR, то есть по модулю 2. Следовательно, вычитание многочленов равнозначно их сложению. Альтернативно сложение элементов конечного поля может быть описано, как сложение по модулю 2 соответствующих бит в байте.

### Умножение

При представлении байт в виде многочленов операция умножения в поле соответствует умножению многочленов по модулю, представляющему собой неприводимый многочлен степени 8. Многочлен является неприводимым, если его делителями являются только единица и он сам. Для алгоритма AES этим неприводимым многочленом является:

Приведение к модулю m(x) гарантирует, что результат будет двоичным многочленом со степенью меньше 8 и, следовательно, может быть представлен в виде байта. В отличие от сложения, на уровне байт нет простой операции, которая соответствовала бы умножению.

Произведение двоичного многочлена на многочлен х вычисляется путём приведения полученного простым произведением многочлена к модулю m(x), определённому выражением (3.1). Умножение на x может быть выполнено на уровне байт как сдвиг влево и последующий условный побитовый XOR с числом 00011011. Эта операция над байтами обозначается xtime(). Умножение на более высокие степени х может быть выполнено путём повторения операции xtime(). Умножение на любую константу может быть выполнено путём сложения промежуточных результатов.

### 3.1.3 Описание алгоритма

В алгоритме AES входной блок, выходной блок и матрица состояния содержат 128 бит. Это показано равенством Nb = 4, где Nb отражает количество 32-битных слов в матрице состояния. В алгоритме AES длина ключа шифрования K равна 128, 192 или 256 бит. Длина ключа показана переменной Nk, равной 4, 6 или 8 и отражающей количество 32-битных слов в ключе шифрования. В алгоритме AES количество раундов, выполняющихся в процессе работы алгоритма, зависит от длины ключа. Количество раундов показано переменной Nr, где Nr = 10, когда Nk = 4; Nr = 12, когда Nk = 6; и Nr = 14, когда Nk = 8.

В алгоритме AES в процедуре шифрования и процедуре дешифрования используется функция раунда, состоящая из четырёх различных байт-ориентированных преобразований. Этими преобразованиями являются:

1. Замена байт с помощью таблицы (S-блока);
2. Сдвиг строк матрицы состояния на различную величину;
3. Перемешивание данных в пределах каждого столбца матрицы состояния;
4. Сложение ключа раунда с матрицей состояния.

В начале процедуры шифрования вход копируется в матрицу состояния. Затем производится первое сложение матрицы состояния и ключа раунда. После этого матрица состояния преобразуется с помощью функции раунда 10, 12 или 14 раз в зависимости от длины ключа. Причём последний раунд немного отличается от предыдущих Nr - 1 раундов. Конечное значение матрицы состояния копируется в выход.

Функция раунда использует в качестве параметра массив подключей, который представляет собой одномерный массив 4-байтных слов. Эти слова вычисляются с помощью процедуры расширения ключа. Отдельные преобразования – SubBytes(), ShiftRows(), MixColumns() и AddRoundKey() – обрабатывают матрицу состояния.

Преобразование SubBytes() выполняет нелинейную замену байт матрицы состояния с помощью таблицы (S-блока). При этом каждый байт обрабатывается независимо от других.

Преобразование ShiftRows() выполняет циклический сдвиг байт в трёх последних строках матрицы состояния на различное число байт (смещений). Первая строка не сдвигается. В итоге байты сдвигаются на «младшие» позиции в строке, в то время как «самые младшие» байты перемещаются вокруг строки в её «вершину».

Преобразование MixColumns() обрабатывает матрицу состояния столбец за столбцом. Каждый столбец используется в качестве четырёхчленного многочлена. Столбцы рассматриваются как многочлены в поле и умножаются по модулю на фиксированный многочлен а(х):

Преобразование AddRoundKey() выполняет сложение ключа раунда и матрицы состояния с помощью простой побитовой операции XOR. Каждый ключ раунда состоит из Nb слов массива подключей. Каждое из этих Nb слов складывается со столбцами матрицы состояния.

Процедура расширения ключа используется в алгоритме AES для создания массива подключей из ключа шифрования K. Всего генерируется Nb(Nr + 1) слов. В начале алгоритма требуется Nb слов и затем в каждом из Nr раундов требуется Nb слов ключевых данных.

Функция SubWord() принимает на вход 4-байтное слово. Выходное слово формируется путём замены каждого из этих четырёх байт с помощью S-блока.

Функция RotWord() принимает на вход слово и выполняет циклическую перестановку.

Первые Nk слов расширенного ключа заполняются ключом шифрования. Каждое последующее слово, вычисляется путём выполнения операции XOR между предыдущим словом, и словом, находящимся на Nk позиций раньше.

При вычислении слов, находящихся на позициях кратных Nk, над предыдущим словом производятся дополнительные операции. Сначала в слове производится циклический сдвиг байт функцией RotWord(). Затем функция SubWord() изменяет все четыре байта слова с помощью таблицы. После чего выполняется операция XOR между выходом функции SubWord() и словом из массива констант.

Важно отметить, что процедура расширения ключа для 256-битных ключей шифрования (Nk = 8) немного отличается от процедуры для 128- и 192-битных ключей. Если Nk = 8 и i – 4 кратно Nk, то перед операцией XOR предыдущее слово обрабатывается функцией SubWord().

Преобразования, составляющие процедуру шифрования, могут быть инвертированы и применены в обратном порядке для получения прямой процедуры дешифрования алгоритма AES. Отдельные преобразования, использующиеся в процедуре дешифрования – InvShiftRows(), InvSubBytes(), InvMixColumns() и AddRoundKey() – обрабатывают матрицу состояния.

Преобразование InvShiftRows() является обратным к ShiftRows(). Байты в трёх последних строках матрицы состояния циклически сдвигаются на различное число байт (смещений). Первая строка (с номером r = 0) не сдвигается. Остальные строки сдвигаются аналогично ShiftRows().

Преобразование InvSubBytes() является обратным к преобразованию замены байт. Преобразование InvSubBytes() изменяет каждый байт матрицы состояния с помощью инвертированного S-блока. Для этого выполняется преобразование, обратное аффинному преобразованию, и затем находится обратный по умножению элемент в поле .

Преобразование InvMixColumns() является обратным к MixColumns(). Преобразование InvMixColumns() обрабатывает матрицу состояния столбец за столбцом. Каждый столбец используется в качестве четырёхчленного многочлена. Столбцы рассматриваются как многочлены в поле и умножаются по модулю x на фиксированный многочлен :

Преобразование AddRoundKey(), описанное ранее, является обратным само к себе, так как включает в себя только операцию XOR.

В прямой процедуре дешифрования порядок следования преобразований отличается от порядка, применённого в процедуре шифрования. При этом для шифрования и дешифрования используется одинаковый массив подключей. Однако некоторые свойства алгоритма AES позволяют использовать эквивалентную процедуру дешифрования, в которой последовательность преобразований совпадает с последовательностью в процедуре шифрования (при этом преобразования заменяются на обратные). Это достигается путём изменения массива подключей.

Двумя свойствами, благодаря которым возможна эквивалентная процедура дешифрования, являются:

1. Преобразования SubBytes() и ShiftRows() коммутативны. То есть преобразование SubBytes(), следующее сразу после преобразования ShiftRows(), будет эквивалентно преобразованию ShiftRows(), следующему сразу после преобразования SubBytes(). То же верно и для обратных к ним преобразований – InvSubBytes() и InvShiftRows();
2. Операции перемешивания столбцов – MixColumns() и InvMixColumns() – являются линейными по отношению ко входному столбцу.

Эти свойства позволяют поменять местами преобразования InvSubBytes() и InvShiftRows(). Преобразования AddRoundKey() и InvMixColumns() тоже можно поменять местами, если при этом столбцы (слова) массива подключей дешифрования будут изменены преобразованием InvMixColumns().

Эквивалентная процедура дешифрования образуется путём обмена местами преобразований InvSubBytes() и InvShiftRows(), а также обмена местами в “цикле раунда” преобразований AddRoundKey() и InvMixColumns(). При этом подключи дешифрования, используемые с 1 по Nr -1 раунд, должны быть изменены преобразованием InvMixColumns(). Но первые и последние Nb слов массива подключей дешифрования не должны быть им изменены.

С учётом этих изменений структура эквивалентной процедуры дешифрования является более эффективной.

## 3.2 Argon2

В современном мире остро стоит вопрос замедления хэширования. Это было еще до введения моды на быстрые алгоритмы по нахождению исходного значения для конкретного хэша. Существуют разные способы замедления, такие как применение хэширования несколько раз подряд или использование соли, но GPU и специальные устройства ускоряют перебор, с которым сложно бороться, даже с помощью таких вещей, как bcrypt.

Для решения возникшей проблемы было решено провести соревнование, результатом которого должен был стать алгоритм, который сложно ускорить на специальных устройствах и GPU, при этом он должен быть настраиваемым в зависимости от пожеланий разработчика. Таким алгоритмом стал Argon2.

Argon2 позволяет настраивать следующие параметры хэширования:

* Количество итераций;
* Желаемый объем занимаемой памяти;
* Степень параллелизма;
* Размер результата, в байтах;
* Секретный ключ;
* Дополнительные данные.

Так же, Argon2 реализован в двух вариантах: Argon2i и Argon2d. Argon2i делает больше проходов по памяти и более медленный, Argon2d быстрее, но у него отсутствует защита от timing атак, а так же его сложнее подбирать на GPU. Чаще всего выбирают Argon2i.

Argon2i рекомендуется для хэширования паролей, Argon2d — для криптовалют, там timing атаки не страшны.

Алгоритм оптимизирован именно для архитектуры x86/x64, поэтому его крайне сложно ускорять на ASIC/GPU и прочих устройствах. Используется многократный проход по памяти, внутри формируется матрица хэшей большого объёма, которые зависят друг от друга и сложным образом обрабатываются.

## 3.3 Base64

Позволяет кодировать информацию, представленную набором байтов, используя всего 64 символа: A-Z, a-z, 0-9, /, +. В конце кодированной последовательности может содержаться несколько спецсимволов (обычно “=”).

Преимущества:

* Позволяет представить последовательность любых байтов в печатных символах;
* В сравнении с другими Base-кодировками дает результат, который составляет только 133.(3)% от длины исходных данных.

Недостатки:

* Регистрозависимая кодировка.

# ЗАКЛЮЧЕНИЕ

После анализа существующих решений были выяснены основные достоинства и недостатки таких систем, а также требуемые режимы работы и уровень вмешательства конечного пользователя.

Данная программа направлена на работу со средствами обмена сообщениями, которые имею программы-клиенты для связи по сети Internet. За счёт перехвата сообщений во время отправки и получения между программой-клиентом и каналом связи, а также отсутствия действий и вмешательств в работу конечного пользователя, разрабатываемое приложение удовлетворяет требованиям системы «прозрачного» шифрования.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Страница, посвящённая алгоритму AES: [Электронный ресурс] // NIST. URL: http://www.nist.gov/CryptoToolkit4. (Дата обращения 16.03.2019)
2. Реестр объектов компьютерной безопасности (CSOR): [Электронный ресурс] // Computer Security Resource Center. URL: http://csrc.nist.gov/csor/. (Дата обращения 16.03.2019)
3. J. Daemen and V. Rijmen, «AES Proposal: Rijndael», AES Algorithm Submission, 3 September 1999 г.
4. J. Daemen and V. Rijmen, «The block cipher Rijndael, Smart Card research and Applications», LNCS 1820, Springer-Verlag, pp. 288-296
5. Домашняя страница B. Gladman’а, посвящённая алгоритму AES: [Электронный ресурс] // Brian Gladman's Home Page. URL: http://fp.gladman.plus.com/cryptography\_technology/. (Дата обращения 16.03.2019)
6. Lee, NIST Special Publication 800-21, Guideline for Implementing Cryptography in the Federal Government, National Institute of Standards and Technology, November 1999
7. Menezes, P. van Oorschot and S. Vanstone, «Handbook of Applied Cryptography», CRC Press, New York, 1997, pp. 81-83
8. J. Nechvatal, «Report on the Development of the Advanced Encryption Standard (AES)», National Institute of Standards and Technology, 2 October 2000.
9. Argon2: the memory-hard function for password hashing and other applications [Электронный ресурс] // GitHub. URL: https://github.com/P-H-C/phc-winner-argon2/blob/master/argon2-specs.pdf. (Дата обращения 04.05.2019)