Министерство образования и науки Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)»
ИНСТИТУТ ЕСТЕСТВЕННЫХ И ТОЧНЫХ НАУК
Факультет математики, механики и компьютерных технологий
Кафедра прикладной математики и программирования
Направление подготовки Программная инженерия

РАБОТА ПРОВЕРЕНА
Рецензент,

_____
«____»_____20___г.

ДОПУСТИТЬ К ЗАЩИТЕ
Заведующий кафедрой, д.ф.-м.н., доцент
_____/А.А.Замышляева
«____»_____2017 г.

Обеспечение безопасного канала связи в социальных сетях

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ
ЮУрГУ–090404.2017.166.ПЗ ВКР

Руководитель работы, к.ф.-м.н., доцент кафедры ПМиП
_____/С.М. Елсаков
«____»_____2017 г.
Автор работы
студент группы ЕТ-223
_____/ А.С. Каныбеков
«____»_____2017 г.
Нормоконтролер, к.э.н., доцент
_____/ Д.А. Дрозин
«____»_____2017 г.

Челябинск, 2017

АННОТАЦИЯ

Каныбеков А.С. Обеспечение безопасного канала связи в социальных сетях.– Челябинск: ЮУрГУ, ЕТ-223, 52 с., 19 ил., 2 табл., библиогр. список – 29 наим., 2 прил.

В работе исследованы различные алгоритмы текстовой стеганографии. Рассмотрены виды атак на криптографические системы, проведен анализ стойкости алгоритмов. Приведен алгоритм кодирования текстовых сообщений использующий положительные особенности криптографических и стеганографических алгоритмов. Приведен анализ предложенного алгоритма и сравнение с существующими алгоритмами. Реализовано клиентское приложение для социальной сети ВКонтакте, использующее предложенный алгоритм.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

В процессе передачи информации участвуют источник информации и приемник информации. Между ними действует канал передачи информации - канал связи. Канал связи - совокупность логических (способы пред- и пост-обработки сообщений) и физических (тип носителя сигнала) средств, обеспечивающих этот процесс.

Под безопасным будем подразумевать канал, обладающий стойкостью к прослушиванию (атакующая сторона не может узнать текст сообщения) и внешнему воздействию (атакующая сторона не может так или иначе изменить текст сообщения, отправленного пользователем).

При достижении решения задачи обеспечения безопасного канала связи можно пойти несколькими путями.

1. Использовать непосредственно безопасный физический носитель сигнала и обеспечивая его целостность, помешать атакующей стороне получить сигнал. Подобного рода решения подходят для крупных компаний, которые базируются в одной локации: внутренняя информация не должна покидать неких границ. Минус - высокая стоимость, необходимость постоянной поддержки, сложность расширения, каждый клиент - потенциальная слабая точка.
2. Зашифровать сообщение, используя какую-либо криптосистему. Подходит для передачи в открытых каналах. Минус - передавать в открытом канале зашифрованное сообщение само по себе достаточно подозрительно.
3. Используя стеганографию, оставлять закодированное сообщение в открытом месте. При должной реализации, никто, кроме адресата не сможет осознать, что перед ним находится спрятанное сообщение.

Первый вариант представляется достаточно сомнительным для реализации из-за дороговизны и сложности поддержки. Далее, в третьем случае, тот факт, что атакующей стороне нужно обнаружить существование закодированного сообщения, усложняет ей задачу. Соответственно, было принято решение об использовании стеганографии в процессе разработки безопасного канала связи.

В социальных сетях сейчас находится огромное количество информации, и внедрить туда сообщение тем или иным способом не представляет особого труда. При этом, если замаскировать сообщение под типичный контент социальной сети, обнаружить факт сокрытия информации будет достаточно проблематично.

Из информации представленной выше, вытекает цель данной работы: реализация безопасного канала связи в социальной сети.

Для выполнения поставленной цели необходимо решить следующие задачи:

- Проанализировать существующие подходы к обеспечению безопасности каналов передачи информации.

- Проанализировать существующие решения подобного рода задач.
- Разработать алгоритм кодирования
- Программная реализация - реализация алгоритма кодирования, оболочки API ВКонтакте, клиента для социальной сети.
- Тестирование программного обеспечения.

ГЛАВА 1. МЕТОДЫ ТЕКСТОВОЙ СТЕГАНОГРАФИИ.

Стеганография (греч. στεγανός - скрытый + γράφω - пишу) - наука о передаче сокрытых данных, внутри других, открытых данных . В отличие от криптографии, где просто происходит конвертирование сообщение в некий нечитаемый массив данных, стеганография делает сообщение незаметным, пряча его в других данных тем или иным способом.

В стеганографии используется следующая терминология:

1. Стеганографическая система (стегосистема) — объединение методов и средств, используемых для создания скрытого канала для передачи информации.
2. Стеганографическое сообщение, встроенное сообщение (стегосообщение) — общее название передаваемой скрытой информации из набора допустимых для использования в качестве сообщения.
3. Стеганографический контейнер (стегоконтейнер) — любая информация, используемая для сокрытия тайного сообщения. Пустой контейнер – контейнер, не содержащий секретного послания. Заполненный контейнер – контейнер, содержащий секретное послание. Представим контейнер в виде конечного набора неких элементов из множества.

Процесс формирования стего можно выразить следующим образом:

4. Стеганографическая плотность — отношение количества битов/сэмплов или других элементов сигнала, которых пришлось изменить в ходе шифрования, к общему объему контейнера.
5. Стеганографический канал (стегоканал) — канал передачи стегоконтейнера.
6. Стеганографический ключ (стегоключ) — секретный ключ, необходимый для сокрытия стегоконтейнера. Можем также представить стегоключ в виде конечного набора из множества элементов, пригодных в качестве ключа.
7. Стеганоанализ — процесс выявления стеганографических сообщений.

Принято выделять три основных “цели” применения стеганографии.

- Цифровые отпечатки (Digital fingerprints)

При использовании стеганографии, как средства генерации цифровых отпечатков, для каждого экземпляра контейнера генерируется своя стеганографическая метка-сообщение. Яркий пример использования - защита исключительного права. В каждый экземпляр защищаемого продукта встраивается информация о покупателе (например, адрес электронной почты, инициалы, IP-адрес). Теперь, нарушитель не может незаконно распространять продукт, так как он сразу станет обнаружен, но до тех пор, он не научится подделывать цифровой отпечаток.

Цели атак на подобного рода системы делятся две группы:

1. Подмена одного цифрового отпечатка другим
2. Простое извлечение цифрового отпечатка.

- Стеганографические водяные знаки (Stego Watermarks)

В данном случае, для каждого экземпляра контейнера предполагается наличие одинаковых меток. Алгоритмы, работающие с водяными знаками, в числе прочих, можно применять для защиты авторского права. К примеру, можно внедрять в фотографии/аудио/видео информацию об авторе, времени записи и т.д.

- Скрытая передача данных.

Классическая цель стеганографии. Необходимо передать данные так, чтобы нарушитель не догадался о факте наличия сообщения. Алгоритмы из данной группы наиболее хорошо подходят для решения задачи обеспечения безопасного канала связи.

1.1. Характеристики методов

Метод, с помощью которого будет решаться поставленная задача будет выбираться на основе следующих характеристик:

1. Определим, что под пропускной способностью каналов передачи скрываемых сообщений или просто скрытой пропускной способностью (ПС) будем понимать максимальное количество информации, которое может быть вложено в один элемент контейнера. При этом, скрываемые сообщения должны быть безошибочно переданы получателю и защищены от атак нарушителя, таких как попытки обнаружения факта наличия канала скрытой связи, чтения скрываемых сообщений, преднамеренного ввода ложных сообщений или разрушения встроенной в контейнер информации. Канал скрытой связи образуется внутри канала открытой связи. Пропускная способность канала открытой связи определяется как количество информации, которое потенциально можно передать без ошибок за одно использование канала. При этом не предъявляется никаких требований к защищенности от атак организованного нарушителя. Поэтому логично предположить, что скрытая пропускная способность должна быть меньше пропускной способности канала открытой связи, в котором за одно использование канала передается один элемент контейнера, в который вложена скрываемая информация.
2. Качество сокрытия. Любое внедрение сообщения, заставляет так или иначе меняться контейнер. Качество сокрытия (или “прозрачность” или качество алгоритма) - качественная характеристика меры искажения контейнера. Его нельзя выразить численно, так что лучший способ измерить эту характеристику - представить нескольким наблюдателям контейнеры до и после внедрения. Если никто не может обнаружить разницы между контейнерами, стеганографический алгоритм оценивается хорошо. Качество сокрытия прямо зависит от восприятия человека или группы людей. Высокая пропускная способность обычно бесполезна, так как она ухудшает качество сокрытия.
3. Необнаружимость. Атакующая сторона может определить наличие сообщения в контейнере путем подсчета определенных статистических свойств файла и сравнения полученных результатов со значениями, которые ожидаются от таких типов файла. Для примера, ошибки в предсказании пикселя в цветном или

черно-белом сообщении соответствуют распределению Лапласа. Если конкретное изображение было рассмотрено и было обнаружено, что оно имеет иное распределение пикселей, атакующая сторона может заподозрить контейнер и провести дальнейшие проверки. Хороший стеганографический алгоритм не должен менять статистических свойств контейнера. Главное отличие необнаружимости от качества сокрытия — первое не зависит от человеческого восприятия.

4. Робастность. Мера способности алгоритма сохранять сообщение даже после того, как контейнер подвергался неким изменениям: линейная и нелинейная фильтрация, добавление случайного шума, сжатие с потерями/восстановление или некоторых видов обработки. В случае, если стегоконтейнер - изображение, возможные операции- это наложение фильтров, изменение резкости, вращение изображения, изменение дискретизации/резкости. Пользователь может сжимать и восстанавливать контейнер в попытках уничтожить сообщение. Робастность очень важна, когда скрытое сообщение представляет из себя копирайт или информацию о правообладателе (водяной знак).

5. Устойчивость к искажению. Атакующая сторона может попытаться изменить сообщение, вместо того, чтобы уничтожить его. Алгоритмы, устойчивые к искажению делают очень сложным возможность изменить спрятанное сообщение или заменить его другим. Особенно уязвимой является информация о копирайте. Подобного рода информация должна оставаться актуальной на протяжений десятилетий.

6. Соотношение "сигнал-шум". Эта величина является мерой качества сокрытия или необнаружимости. В основном, высокое значение соотношение идеально для систем коммуникации, а низкое- идеально для стеганографии, так как контейнер - это шум, а сообщение - сигнал.

Пропускную способность, необнаружимость и робастность выделяют в т.н. “Треугольник характеристик” [#] - улучшение одной характеристики алгоритма, вызывает ухудшение остальных, поэтому, ни один стеганографический алгоритм не может быть абсолютно неразличим, полностью робастным и иметь максимальную пропускную способность.

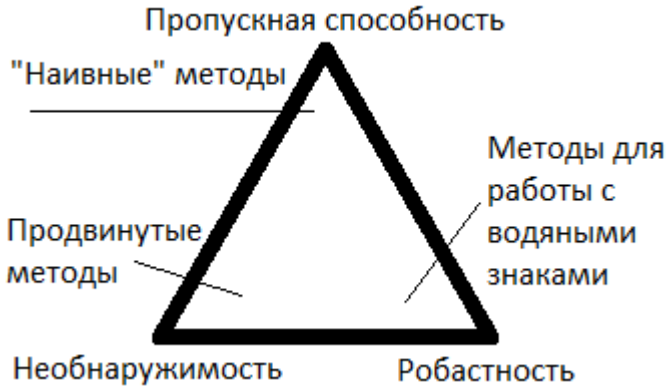

Рис. 1.1. Треугольник характеристик алгоритмов стеганографии.

7. Прочие свойства.

7.1. Так же, как и в криптографии, в стеганографии может иметься стегоключ для улучшения безопасности. Принцип Керкгоффса, применяемый к стеганографии, гласит, что безопасность сообщения должна зависеть от секретности стего-ключа и не должен зависеть от секретности алгоритма сокрытия данных.

7.2. Сложность алгоритма также является важным свойством. Метод требующий больших вычислений может привести к высоким качеству сокрытия и робастности, что также может повлечь увеличение времени, требующегося атакующей стороне на попытку взлома. Время выполнения кодирования в большинстве случаев менее приоритетно, по сравнению с остальными свойствами алгоритма.

7.3. Наличие асимметричного алгоритма кодирования. Медленный алгоритм кодирования может быть приемлем, если алгоритм декодирования очень простой и быстрый.

7.4. Использование кодов, корректирующих ошибки. Большое количество манипуляций с контейнером, может привести к его искажению. Идеальный стеганографический алгоритм способен определять и корректировать ошибки в передаваемом сообщении.

7.5. Определенные типы атак могут повреждать часть контейнера. В идеальном случае, алгоритм декодировать может извлекать сообщение из оставшейся части контейнера.

7.6. Стеганографический алгоритм, поддерживающий параллельные вычисления, может быть исполнен сразу на нескольких процессорах.

1.2. Классификация стеганографических алгоритмов

Методы современной стеганографии можно разделить на две основные группы - цифровую стеганографию и лингвистическую стеганографию. Методы первой группы основываются на внедрении дополнительной информации в цифровые объекты, во втором случае производится внедрение сообщения в контейнер с опорой на свойства языка и лингвистические ресурсы.

1.2.1. Цифровая стеганография

Контейнерами для методов цифровой стеганографии могут служить: изображения, видео, аудио, текстуры 3D - объектов. Простейшим примеров алгоритма цифровой стеганографии является алгоритм LSB (Least Significant Bit, наименьший значащий бит), сутью которого является замещение последних значащих битов в контейнере на биты скрываемого сообщения так, чтобы разница между заполненным и пустым контейнерами была неощутима для восприятия человека. Методы LSB являются неустойчивыми ко всем видам атак и могут быть использованы только при отсутствии шума в канале передачи данных.

Обнаружение LSB - кодированного стего осуществляется по аномальным характеристикам распределения значений диапазона младших битов отсчетов цифрового канала.

Эхо-методы применяются в цифровой аудиостеганографии и используют неравномерные промежутки между эхо-сигналами для кодирования последовательности значений. При наложении ряда ограничений соблюдается условие незаметности для человеческого восприятия. Эхо характеризуется тремя параметрами: начальной амплитудой, степенью затухания, задержкой. При достижении некоего порога между сигналом и эхом они смешиваются. В этой точке человеческое ухо уже не может различить эти два сигнала. Наличие этой точки сложно определить, и она зависит от качества исходной записи и качества слуха слушателя. Чаще всего используется значение задержки около 1/1000, что вполне приемлемо для большинства записей и слушателей. Для обозначения логического нуля и единицы используется две различных задержки. Они обе должны быть меньше, чем порог чувствительности уха слушателя к получаемому эху.

1.2.2. Лингвистическая стеганография

Под лингвистической стеганографией подразумевается набор алгоритмов и приемов, использующих текст в качестве контейнера, при этом используя некие лингвистические знания. Заполненный в результате использования алгоритма лингвистической стеганографии контейнер все еще должен содержать достаточно осмысленный текст с орфографией, лексикой и синтаксисом.

Приведем основные типы алгоритмов лингвистической стеганографии:

1. Семаграммы скрывают информацию используя символы или знаки. Визуальные семаграммы используют ничем не примечательные или обыденные физические объекты для передачи сообщения, к примеру, каракули или позиционирование элементов веб-сайта. Текстовые семаграммы скрывают сообщение, модифицируя контейнер, например, изменяя шрифт или его размер, добавляя лишние пробелы. Для шифрование рукописного текста возможно добавление различного рода “завитушек” в записи букв.
2. Открытое шифрование скрывает сообщение в “легальном” контейнере методами, неочевидными для наблюдателя.
3. Закрытое шифрование скрывает сообщение таким образом, чтобы оно могло быть восстановлено персоной, знающей, как сообщение было зашифровано.
4. Использование жаргонов. Жаргонизмы - некие слова/словосочетания, которые понимает определенная группа людей: для непосвященных они не имеют смысла. Подклассом такого рода алгоритмов является ключевой код, где конкретные, заранее подготовленные фразы кодируют сообщение. Ключевые коды используют наиболее краткий носитель сообщения, чтобы сигнализировать о существовании стеганографического сообщения, семантика которого была заранее оговорена.

5. Скрытые шифры. Покрытые или маскируемые шифры скрывают сообщение, открытое в носителе, так что оно может быть восстановлено кем угодно, кто знает алгоритм кодирования. Скрытые шифры можно разделить на решеточные и нулевые шифры.

1.3. Описание методов текстовой стеганографии

Рассмотренные методы текстовой стеганографии представлены ниже.

1. Метод изменения регистров символов.

Данный метод предполагает кодирование нулевого бита сообщения строчным символом контейнера, а единичного - прописным символом. Для кодирования можно использовать только буквенные символы контейнера.

Алгоритм сокрытия выглядит следующим образом:

Содержимое файла-контейнера считывается посимвольно, если очередной символ является буквой, происходит кодирование бита сообщения.

Если сделать текст сообщения достаточно легкомысленным, контейнер не будет привлекать внимания обычного пользователя. В лучшем случае, каждая буква может кодировать бит и пропускная способность может достигать 100%. Однако, этот алгоритм является одним из базовых алгоритмов текстовой стеганографии и стегоанализ текста на использование этого алгоритма проводится в первую очередь.

ПРимеР ПиСьМА ЗаборЧИКОм

Рис. 1.2. Пример возможного стегоконтейнера

Алгоритм, представленный ниже может применяться для обнаружения использования метода изменения регистра символа

```
пока (в файле есть слова){
     емкость+=емкость(текущего слова)
     если (слово не «нормальное»){
     емкость скрытого+=емкость(текущего слова)
}
если(емкость>ЗАДАННАЯ_ЕМКОСТЬ){
     если((ЕМКОСТЬ_СКРЫТОГО)/ЕМКОСТЬ>ЗАДАННОЕ_ОТНОШЕНИЕ){
          сделать вывод, что имеется скрытое сообщение
          завершение работы
     }иначе {
          обнулить емкость и емкость_скрытого
     }
}
сделать вывод что в файле нет скрытого сообщения
завершение работы
```

Ёмкость в таком случае - это число байт, которое можно скрыть в данном слове данным методом.

Слово считается нормальным, если оно состоит только из букв одного регистра. Как только отношение “подозрительных” слов превышает некоторое значение, устанавливаемое нами, можно считать что применялся этот метод.

2. Метод добавления хвостовых пробелов.

В процессе шифрования текстовый контейнер считывается построчно. Удаляются пробелы, находящиеся в конце строки, игнорируются символы ‘\r’ (символ возврата каретки) и ‘\t’ (табуляция). Соответственно, происходит дописывание нуля или одного в зависимости от текущего бита стегосообщения.

Пропускная способность данного алгоритма напрямую зависит от количества строк в контейнере. В среднем, для сетей Facebook и VKontakte, длина строки равна 80 символам, так что для первой кодировки пропускная способность будет равна 80/100*80 = 1/100 бит = 0,1% , для второй кодировки - 80/200*80 = 1/200 бит = 0.05%, что является достаточно низким результатом. Достаточно важным фактом является то, что при распечатке контейнера, сообщение будет потеряно.

Заполненный контейнер достаточно незаметен для невооруженного глаза, однако, этот алгоритм также является одним из базовых и в процессе стегоанализа проверяется одним из первых.

Стегоанализ происходит следующим образом:

```
пока (в файле есть строки){
     считать очередную строку
     удалить символы  \r’,’\t’ из конца строки
          если(в конце строки 2 пробела)
            число_единиц+=1
иначе если (в конце строки 1 пробел)
     число_нулей+=1
          если(min(число_нулей,число_единиц)>ЗАДАННАЯ_ВЕЛИЧИНА){
       сделать вывод, что имеется скрытое сообщение
}иначе {
  сделать вывод что в файле нет скрытого сообщения
     }
  }
  завершение работы
```

Контейнер считывается построчно и считается количество хвостовых пробелов. Как только это число превысит определенное значение, можно сделать вывод, что использовался данный алгоритм.

3. Модифицированный метод добавления хвостовых пробелов.

В конце каждой строки добавляется от нуля до пятнадцати пробелов, кодируя полубайт. Контейнер становится достаточно подозрительным, но при определенном форматировании текста можно добиться корректного визуального эффекта. Стегоанализ проводится аналогично предыдущему методу.

4. Знаки одинакового начертания.

Ряд символов русского и английского языка имеют одинаковое начертание в некоторых шрифтах, но разные ASCII-коды. Приведем таблицу (таблица 1.1.) встречаемости подобных символов в русском языке на основе статистики частотности букв [3]. Для сокрытия сообщения, контейнер считывается посимвольно. На основании текущего бита сообщения происходит замена символа контейнера на латинский символ из таблицы.

Таблица 1.1.

Таблица употребляемости символов одинакового начертания в русском языке.

Из таблицы следует, что вероятность встретить в русскоязычном тексте символ, подходящий для кодирования примерно равна 0.49283. Соответственно, максимальная пропускная способность может достигать примерно этой отметки.

5. Двойные пробелы между словами.

Один или два пробела кодируют бит информации. Невооруженным взглядом достаточно сложно различить количество пробелов, однако машинные методы стегоанализа относительно легко справляются с этим методом. Из [5] следует, что вероятность встречи пробела в русском языке равна 0,021.

6. Использование синонимов.

Используя большой словарь синонимов, можно заменять отдельные слова их синонимами, при этом кодируя от одного бита информации. Упрощенный вариант этого метода, рассмотренный в [3], гарантирует, что количество битов скрываемой информации равно двоичному логарифму от длины вектора синонимов. Там же, указывается, что словарь из 1000 слов обеспечивает покрытие более 50% любого осмысленного текста. Из этих данных можно получить количественную характеристику величины отношения «сигнал-шум». Однако данный алгоритм достаточно сложен для реализации в машинном виде, особенно для использования с русским языком из-за наличия видоизменения слов.

7. Опечатки в тексте.

Суть алгоритма заключается во внедрении в готовый текстовый контейнер опечаток. К примеру, происходит замена буквы на располагающуюся рядом на клавиатуре. В идеальном случае (в тексте без коротких слов), это позволяет в каждом слове шифровать один бит. В данном алгоритме характеристики «качество сокрытия» и отношение «сигнал-шум» обратно пропорциональны.

8. Эмотиконы.

Эмотико́н (англ. emoticon) — пиктограмма, изображающая эмоцию; чаще всего составляется из типографских знаков. Особое распространение получил в Интернете и SMS (и пр. текстовых сообщениях), однако в последнее время используется повсеместно. UTF-16 кодирует символы в виде последовательности 16-битных слов, это позволяет записывать символы Юникода в диапазонах от

U+0000 до U+D7FF и от U+E000 до U+10FFFF (общим количеством 1 112 064). Если требуется представить в UTF-16 символ с кодом больше U+FFFF, то используются два слова: первая часть суррогатной пары (в диапазоне от 0xD800 до 0xDBFF) и вторая (от 0xDC00 до 0xDFFF).

Рис. 1.3. Эмотикон - “Grinning face”

К примеру, рассмотрим эмотикон “улыбающееся лицо” с кодом 0x1F600. В c# переменная типа string, будет иметь длину 2 и два элемента типа char будет хранить в себе значения - 0x0001 и 0xF600. Соответственно значения суррогатных пар можно получить выполнив следующий код на языке C#:

```
short c1 = (short)text[0];
short c2 = (short)text[1];
```

Введем словарь, в котором числам от 0 до 9 и 32 буквам русского алфавита соответствуют разные эмотиконы. Предлагаемый алгоритм кодирования заключается в замене символов на эмотиконы с последующим внедрением в пустой контейнер. Данный алгоритм более подробно будет рассмотрен в следующей главе.

1.4. Оценка алгоритмов текстовой стеганографии

Приведем оценочную таблицу (таблица 1.2.) для каждого алгоритма. Алгоритмы будем оценивать по следующим параметрам:

1. Качество сокрытия. 0 баллов – стегоканал плохо сокрыт, заполненный контейнер очень просто обнаружить. 1 балл – стегоканал сокрыт относительно хорошо. 2 балла – стегоканал полностью сокрыт.
2. Необнаружимость. 0 баллов – стегоканал обнаружим за очень короткое время. 1 балл – стегоканал обнаружим за относительно небольшое время. 2 балла – стегоканал не обнаружим за разумное время.
3. Робастность 0 баллов – после изменения контейнера сообщение полностью теряется. 1 балл – после изменения контейнера сообщение можно прочесть, однако произошло искажение стегосообщения. 2 балла - после изменения контейнера стегосообщение полностью сохранилось.

4. Соотношение «сигнал – шум». 0 баллов – соотношение находится в пределах [1% - 33%]. 1 балл – соотношение находится в пределах [34% - 66%]. 2 балла – соотношение находится в пределах [67% - 100%].

Таблица 2.

Сравнительная таблица характеристик стеганографических алгоритмов.

Номер алгоритма	Соотношение «сигнал – шум»	Робастность	Необнаружимость	Качество сокрытия	Итоговая оценка
1	2	1	1	0	4
2	0	1	0	1	2
3	1	1	0	0	2
4	1	1	1	1	4
5	0	1	0	1	2
6	1	2	1	2	6
7	1	1	1	1	4
8	2	2	1	2	7

1.5. Классификация атак на стеганографические алгоритмы

Описанная Густавосом Симмонсом в 1983 году, т.н. “Проблема заключенных” часто используется для описания различных сценариев использования стеганографии, несмотря на то, что в оригинале она была использована для описания криптографических сценариев. Эта проблема описывает двух заключенных - Алису и Боба, которые заперты в разных камерах тюрьмы и хотят обсудить некий секретный план. Алиса и Боб могут так или иначе передавать сообщения друг другу, однако Уильям - охранник, может читать все сообщения. Заключенные знают, что Уильям запретит общение, если обнаружит секретный канал передачи информации.

Охранник может вести себя активно или пассивно. В пассивном режиме, он будет проверять каждое сообщение и решать передавать ли его дальше или нет, на основе своей наблюдательности. В активном режиме, Уильям может изменять сообщения так, как захочет. Подозрительный охранник может модифицировать сообщения, в попытке уничтожить скрытое сообщение, так что Алиса и Боб должны использовать достаточно робастный алгоритм. Сложность задачи Уильяма будет зависеть от сложности выбранной стегосистемы и его предыдущего подобного опыта.

Стегоанализ - определение наличия использования стеганографии третьей стороной. Можно выделить основные задачи стегоанализа.

1. Обнаружение факта присутствия скрытого сообщения.
2. Извлечение скрытого сообщения.
3. Модификация скрытого сообщения.
4. Запрет на выполнение любой пересылки информации, в т.ч. скрытой.

Первые две задачи относятся к пассивной модели анализа, вторые две - к активной.

Первым делом происходит первичный анализ контейнера, включающий в себя следующие действия:

1. Сортировка контейнеров по внешним признакам.
2. Выделение контейнеров с известным алгоритмом встраивания.
3. Определение использованных алгоритмов.
4. Проверка достаточности объема материала для анализа.
5. Аналитическая разработка стегоматериалов. Разработка методов вскрытия стегосистемы.
6. Выделение стегосообщений с известными алгоритмами встраивания, но неизвестными ключами и т.д.

Методы стегоанализа можно классифицировать схожим с криптоанализом образом, на основе количества известной атакующей стороне информации о стегосистеме:

1. Атака на основе известного заполненного контейнера:

В данном сценарии атакующая сторона располагает каким-то количеством заполненных стегоконтейнеров (в случае, если их несколько, предполагается, что применявшийся алгоритм встраивания был одним и тем же). Её задачей может являться в обнаружении факта наличия канала скрытой передачи данных, нахождения использовавшегося алгоритма встраивания, извлечении данных. Зная использованный алгоритм, атакующая сторона может извлечь остальные стегосообщения.

2. Атака на основе известного встроенного сообщения.

Данный сценарий наиболее характерен для задачи защиты интеллектуальной собственности, в случае, когда в качестве цифрового водяного знака используется некий известный элемент. Задачей стегоанализа в данном случае является нахождение водяного знака и его извлечение.

3. Адаптивная атака на основе выбранного сообщения.

Данный сценарий является частным случаем предыдущего. Атакующая сторона может навязывать жертве сообщение для дальнейшего кодирования (на основе предыдущих анализов) и наблюдать его результат .

4. Атака на основе выбранного заполненного контейнера.

У атакующей стороны есть детектор заполненных контейнеров - “черный ящик” и некоторое количество самих заполненных контейнеров. Нарушитель пытается определить алгоритм встраивания путем анализа найденных скрытых сообщений.

5. Атака на основе известного пустого контейнера.

Если атакующей стороне известен пустой контейнер, то сравнивая его с подозрительным контейнером, можно обнаружить факт передачи скрытого сообщения.

6. Атака на основе известной математической модели контейнера или его части.

Имея возможность получать структуру контейнера, можно сравнивать модель подозрительного контейнера и известную ему модель. К примеру, последние биты соседних пикселей изображения обычно отличаются друг от друга, так что в случае, если если был использован метод наименьшего значащего бита, атакующая сторона сразу заметит аномалию. Так что одним из требований к хорошему стеганографическому алгоритму является отсутствие нарушения различных статистических характеристик контейнера.

Стегоанализ следует путем, которым работает стеганографический алгоритм. Самый простой способ - визуальный осмотр стегоконтейнера. К примеру, большинство инструментов стеганографии работают непосредственно с битовым представлением изображения и выбирают биты сообщения из стегоконтейнера независимо от его содержимого. Несмотря на то, что легче прятать сообщения в области яркого цвета или громкого звука, программа может не проверять специально эти области. Поэтому, в некоторых случаях визуального осмотра может быть достаточно для определения наличия стеганографии.

Второй способ - поиск структурных аномалий контейнера, которые подразумевают наличие манипуляций. Использование метод наименьшего значащего бита в изображениях, основанных на палитре часто оставляет за собой большое количество повторяющихся цветов, в которых одинаковые (или почти одинаковые) цвета отличаются только последними битами. Стеганографические алгоритмы, которые манипулируют порядком цветов в палитре также вызывают структурные изменения в контейнере. Эти структурные изменения обычно уникальны для каждого алгоритма.

Статистический стегоанализ становится сложнее, так как некоторые алгоритмы стараются сохранить структуру контейнера, чтобы избежать обнаружения. Предварительное шифрование скрытого сообщения также делает обнаружение сложнее, так как зашифрованные данные обычно имеют большую степень случайности.

Восстановление скрытого сообщения добавляет еще один уровень сложности в сравнении с обычным обнаружением наличия скрытого сообщения. Этот процесс

подразумевает тот факт, что атакующая сторона знает или предполагает длину сообщения, ключ шифрования и знание примененного криптографического алгоритма.

Более сложные методы стегоанализа используют статистику более высоких порядков, линейный анализ, сети Маркова, вейвлет- статистику.

Сегодня, самые распространенные методы стегоанализа основаны на сигнатурах, что роднит их с антивирусами и системами обнаружения вторжений. Несмотря на то, что старые алгоритмы достаточно точны и обладают достаточным уровнем робастности, новые - все более гибкие и способные быстро меняться с появлением новых стеганографических алгоритмов.

Будем считать, что стегосистема взломана, если атакующей стороне, как минимум, удалось доказать наличие спрятанного сообщения в стегоконтейнере. Предполагается, что атакующая сторона может осуществлять любые типы атак и имеет неограниченные вычислительные возможности. Из этого предположения следует, что стегосистема устойчива, если невозможно за разумное время подтвердить гипотезу о существовании в контейнере скрытого сообщения.

Приведем оценочную таблицу (таблица 1.3.) для каждого алгоритма. Алгоритмы будем оценивать по следующим параметрам:

1. Качество сокрытия. 0 баллов – стегоканал плохо сокрыт. 1 балл – стегоканал сокрыт относительно хорошо. 2 балла – стегоканал полностью сокрыт.
2. Необнаружимость. 0 баллов – стегоканал обнаружим за очень короткое время. 1 балл – стегоканал обнаружим за относительно небольшое время. 2 балла – стегоканал не обнаружим за разумное время.
3. Робастность 0 баллов – после изменения контейнера сообщение полностью теряется. 1 балл – после изменения контейнера сообщение можно прочесть, однако произошло искажение стегосообщения. 2 балла - после изменения контейнера стегосообщение полностью сохранилось.
4. Соотношение «сигнал – шум». 0 баллов – соотношение находится в пределах [1% - 33%]. 1 балл – соотношение находится в пределах [34% - 66%]. 2 балла – соотношение находится в пределах [67% - 100%].

Таблица 1.3.
Таблица устойчивости стеганографических алгоритмов к различным криптоатакам.

#	Атака на основе выбранного пустого контейнера	Атака на основе выбранного заполненного контейнера	Атака на основе выбранного скрытого сообщения	Атака на основе известного заполненного контейнера.	Итоговая оценка
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	0
4	0	1	0	1	2
5	0	0	0	0	0
6	0	1	0	1	2
7	0	1	0	2	3
8	0	2	0	2	4

Из таблицы можно сделать вывод, что достаточно устойчивыми алгоритмами являются [4],[6],[7],[8].

1.6. Программное обеспечение, использующее методы текстовой стеганографии

Представим некоторые программные решения использующие алгоритмы текстовой стеганографии в с своей работе, для решения задач близких к обеспечению безопасного канала.

1.6.1. Stego! Text Steganography

Этот инструмент реализует очень простой стеганографический алгоритм: предварительно криптографически зашифрованный текст переводится в то, что на первый взгляд кажется английским текстом. По факту оно и является английским текстом, но полностью бессмысленным, состоящим из слов выбранных из словаря из 65536(2^{16}) слов, каждое из которых шифрует два байта сообщения, соответствующему положению слова в словаре.

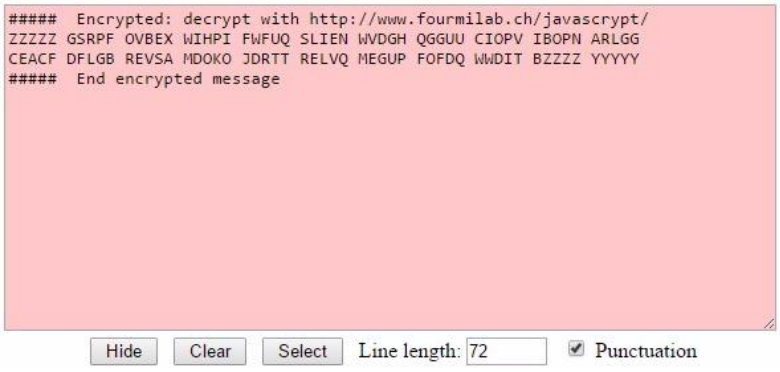

Рис. 1.4. Поле ввода сообщения приложения Stego! Text Steganography.

Добавление знаков пунктуации и разбиение текста на абзацы делает текст чуть более правдоподобным. На самом деле, любой, кто уделит тексту более пристальный взгляд, сразу найдет текст подозрительным. Как вариант, можно вставить полученный текст в большой документ, предварительно уведомив получателя, где конкретно искать текст для расшифрования.

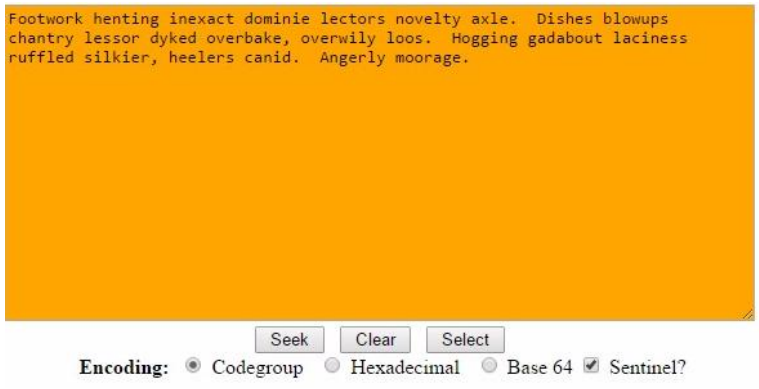

Рис. 1.5. Результат работы ПО.

1.6.2. Text2Text Steganography

Целью данного проекта было написание простого приложения, способного отправлять и принимать зашифрованные сообщения в форматах Rich Text Format: *.DOC, *.RTF и т.д. Пользователь может сам предоставлять пустой контейнер, программа делает вывод, может ли он использоваться.

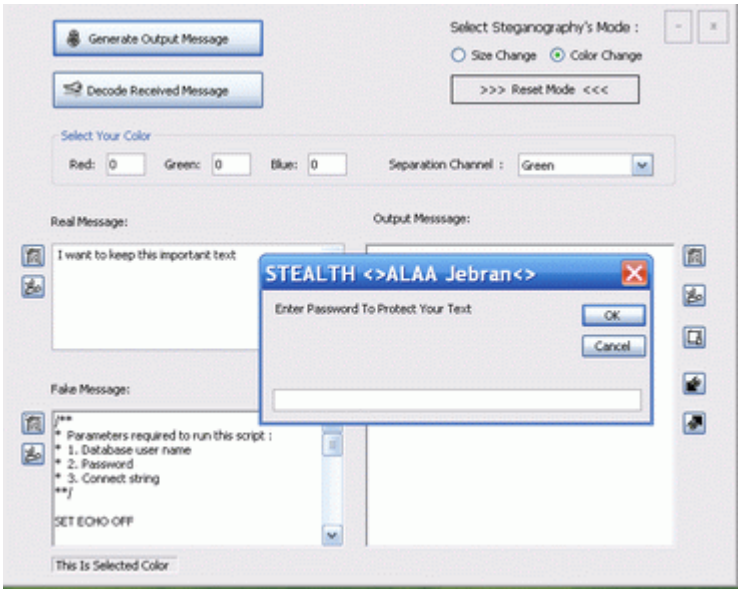

Рис. 1.6.Главное окно приложения Text2Text Steganography.

Программа меняет не сам текст пустого контейнера, а некие различные аттрибуты текста, в частности, размер и цвет шрифта.

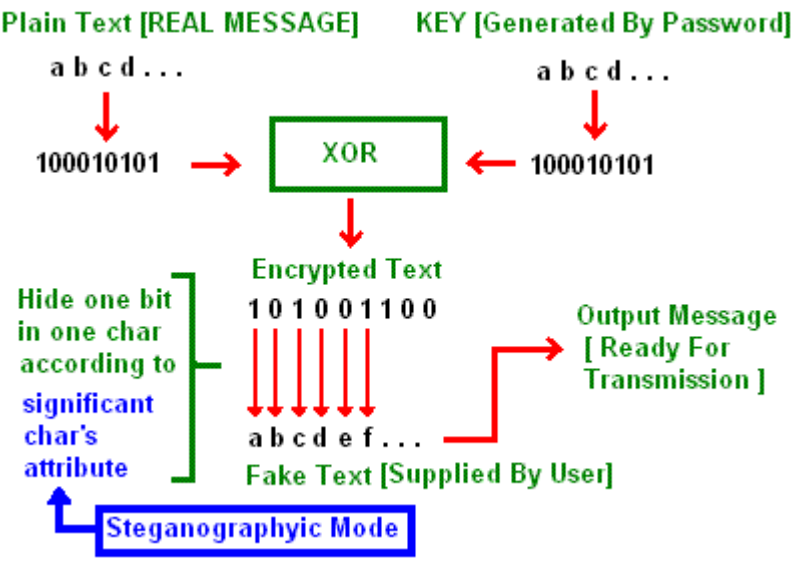

Рис. 1.7. Схема работы приложения Text2Text Steganography.

1.6.3. Spammimic

Данный инструмент является реализацией лингвистического алгоритма стеганографии использующего в качестве стегоконтейнера спам. Спам - массовая

рассылка коммерческой и иной рекламы или подобных коммерческих видов сообщений лицам, не выражавшим желания их получать. Принцип работы алгоритма достаточно прост. Каждая буква сообщения кодируется одним предложением, причем первые буквы слова кодируются предложениями, содержащими приветствие, а последние - прощание - для придания заполненному контейнеру более-менее осмысленного вида. В результате, заполненный контейнер просто напросто игнорируется обычным человеком из-за особенности содержания.

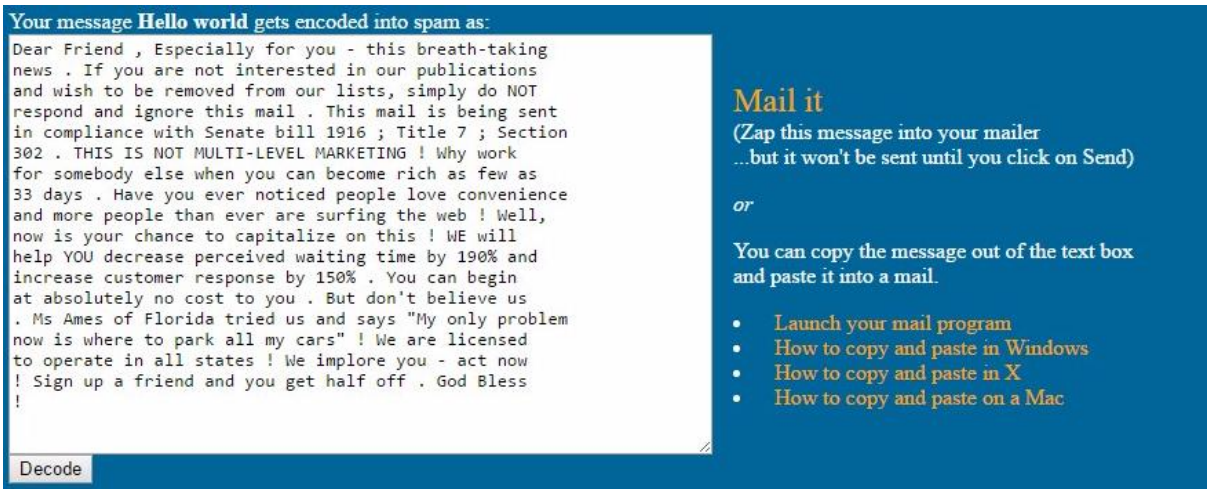

Рис. 1.8. Пример кодирования сообщения “Hello world”

Как можно заметить, основным недостатком данных решения является низкий уровень степени сокрытия и незаметности.

1.7. Выводы по разделу

В данной главе были представлены описания характеристик и характеристики основных алгоритмов текстовой стеганографии и основные типы атак на стегосистемы. Был сделан вывод о том, что почти все методы текстовой стеганографии страдают от недостатка степени сокрытия и незаметности. Также они достаточно взламываются различного рода статистическими атаками.

Для реализации был выбран алгоритм, использующий словарь - (буква, эмотикон),

В результате было принято решение использовать метод стеганографии, основанный на эмотиконах по следующим причинам:

1. Заполненный стегоконтейнер не вызывает подозрений.
2. Данный алгоритм не влияет на статистические характеристики стегоконтейнера.
3. Символ (эмотикон) кодирует не один бит, а целый символ.
4. Словарь “эмотикон-символ” можно менять в любое время.

В качестве дополнительной меры защиты, можно, например, сначала шифровать исходный текст каким-либо методом, а результат помещать в стегоконтейнер. Самый простой вариант решения этого вопроса: перемешать символы исходного текста перед помещением в стегоконтейнер.

ГЛАВА 2. МЕТОД КОДИРОВАНИЯ ТЕКСТА, ИСПОЛЬЗУЮЩИЙ ПРИНЦИПЫ СТЕГАНОГРАФИИ

Рассмотрим алгоритм, выбранный в главе 1, более подробно. Сутью данного алгоритма является замена буквы на эмотикон с последующим их внедрением в текст - пустой стегоконтейнер и отправкой в социальную сеть. За счет того, что использование эмотиконов в социальных сетях - явление обыденное, заполненный стегоконтейнер не будет вызывать подозрений.

Для успешного кодирования необходимы следующие элементы:

1. Пустые стегоконтейнеры - текст из нескольких предложений, используемый в дальнейшем при кодировании. Стегоконтейнеры собираются из публичных сообществ/стен и должны быть достаточного размера (не меньше определенного количества предложений)
2. Библиотека эмотиконов - 42 эмотикона взаимнооднозначно определяющие символы, участвующие в процессе кодирования (32 буквы + 10 цифр). Чтобы сделать заполненный контейнер более необнаружимым, на основе эмоциональной окраски пустого контейнера выбираются “грустные” или “веселые” эмотиконы. Процесс генерации такой библиотеки выглядит следующим образом: необходимо получить значение хэша от текущего времени в формате “dd-mm-yyyy-hh-MM”. На основе этого хеша генерируются 42 различных числа, которые однозначно определяют набор эмотиконов для использования в текущем сеансе кодирования.

2.1. Описание алгоритма кодирования

Приведем схему алгоритма кодирования

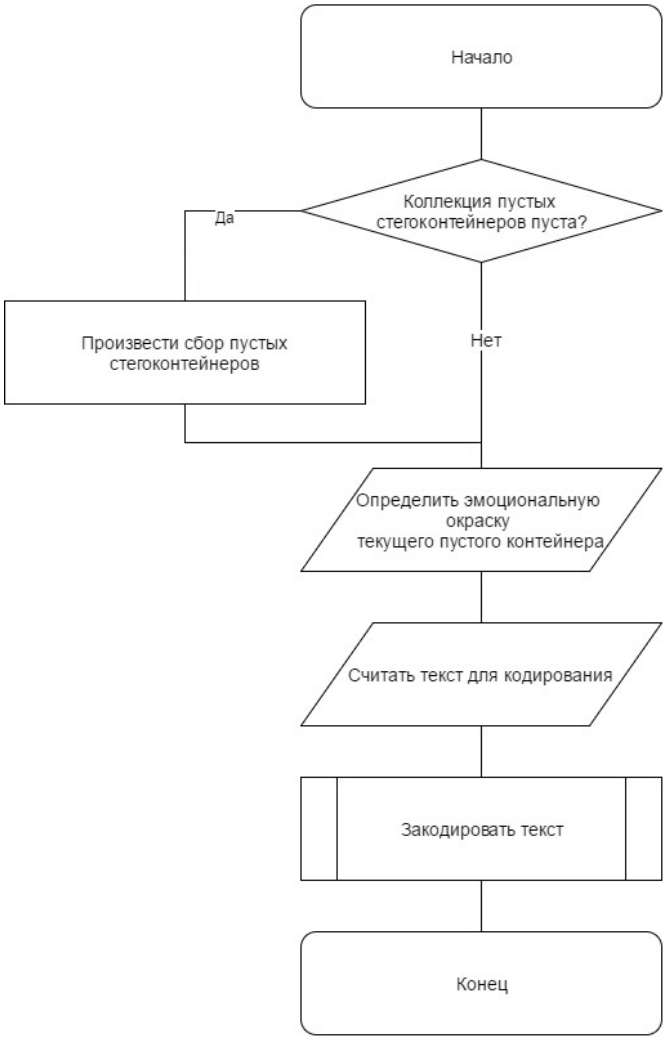

Рис. 2.1. Схема алгоритма процесса кодирования.

Алгоритм кодирования текста:

1. Имеется пустой контейнер и сообщение для кодирования. Необходимо получить хеш от строкового представления текущего времени и на основе его значения генерируется 42 различных числа с помощью метода Random.Next(int seed) с сидом - полученным хешом.
2. На основе этих чисел из общего набора выбираются 42 эмотикона, которые будут использоваться в текущей сессии кодирования.
3. Генерируется словарь “символ - эмотикон”; каждому символу сообщения ставится в соответствие свой эмотикон.
4. Для каждого символа сообщения получаем эмотикон из сгенерированной словаря.
5. Необязательный шаг. Полученные эмотиконы перемешиваются тем или иным способом.
6. Инжектируем полученные эмотиконы в пустой контейнер.
7. Получаем значение хеша времени и на его основе генерируем три эмотикона. Инжектируем их так, чтобы они были первыми тремя эмотиконами в тексте.

8. Получаем значение хеша сообщения и на его основе генерируем три эмотикона. Инжектируем их так, чтобы они были последними тремя эмотиконами в тексте.

Приведем схему алгоритма кодирования текста (рис 2.2.).

Рис. 2.2. Схема алгоритма кодирования текста.

2.2. Анализ разрабатываемого алгоритма

2.2.1. Анализ характеристик алгоритма

Проведем анализ данного алгоритма с точки зрения стойкости и устойчивости к внешнему воздействию.

Так как для кодирования используются распространенные в социальных сетях эмотиконы, догадаться о наличии скрытого сообщения достаточно сложно.

Предположим, атакующая сторона догадалась о наличии скрытого сообщения. Перед ней стоит задача расшифровать тексты.

Фактически, данный алгоритм является шифром простой замены, а это значит, что пространство ключей данного шифра составляет, как минимум, $42! \approx 2^{170}$ (количество перестановок используемого алфавита).

Особенности алгоритма кодирования, усиливающие стойкость шифра:

1. Ключ меняется при каждом кодировании. Из-за того, что библиотека эмотиконов меняется каждый раз, перед процессом кодирования, для каждого сообщения будет своя таблица “эмотикон-символ”.
2. Использование русского языка при кодировке. В английском языке существует огромное количество групп подряд идущих символов в словах (THE, ING, и т.д), что может упростить задачу атакующей стороне.
3. После конвертации символов в эмотикон, они инжектируются не в том же порядке, в каком они шли в исходном слове.
4. Шифротекст достаточно короток. Расстояние единственности (в криптологии) — число символов шифротекста, при которых условная информационная энтропия ключа (а, следовательно, и открытого текста) равна нулю, а сам ключ определяется однозначно. Расстояние единственности для русского текста, зашифрованного шифром простой замены равно = 35,1. То есть, если атакующая сторона получит более 35 символов шифротекста, то с большой вероятности он сможет восстановить исходное сообщение.

Так как заполненный контейнер хранит в себе хеш сообщения, невозможно просто изменить контейнер без его компрометации. Если атакующая сторона попытается изменить сообщение, ей придется узнать алгоритм хеширования, чтобы подделать и хеш.

Более подробно рассмотрим характеристики данного алгоритма:

Качество сокрытия - так как в социальной сети комбинация “текст+эмотиконы” используются огромным количеством людей, заполненный стегоконтейнер ничем не будет отличаться от обычных сообщений.

Необнаружимость - из-за причин описанных выше, стегоканал достаточно сложен для обнаружения.

Робастность - любые виды взаимодействия с контейнером, не включающие в себя удаление эмотиконов из текста, никак не влияют на зашифрованное сообщение. Если скопировать текст в другой мессенджер, изменится только внешний вид эмотикона, код останется неизменным.

Соотношение “сигнал-шум” - чтобы добиться максимальной необнаружимости, стоит добавлять не больше одного эмотикона на предложение контейнера (в лучшем случае - еще реже) . Средняя длина предложения в русском языке - 10,3 слова, средняя длина слова - 7 букв. Поэтому, чтобы закодировать 25 символов, контейнер должен содержать примерно 1800 символов. В итоге, соотношение “сигнал-шум” для данного алгоритма будет равняться примерно 1%. Однако, при желании, можно искусственно поднять значение этой характеристики вставляя эмотикон не в конце каждого предложения, а после любых слов. Если сделать это аккуратно, необнаружимость и качество сокрытия пострадают не слишком сильно.

2.2.2. Анализ устойчивости алгоритма к атакам

- Атака на основе известного заполненного контейнера.

Располагая некоторым количеством заполненных контейнеров, можно сделать вывод о наличии стегоканала и составить некоторое представление об алгоритме. Однако, как будет показано в следующей главе, достаточно сложно получить исходное сообщение или подменить его.

- Атака на основе известного встроенного сообщения.

Даже если у атакующей стороны имеется экземпляр сообщения, есть вероятность того, что она не сможет понять алгоритм кодирования, так как алгоритм обладает достаточно высоким показателем

- Атака на основе известного пустого контейнера

Данный алгоритм уязвим к подобного рода атаке, так как процесс кодирования подразумевает только добавление эмотиконов в контейнер. Атакующая сторона будет знать механизм кодирования, однако, перед ней все еще будет стоять задача расшифровывания кодированного текста. Вкупе с этим, так как для каждой сессии кодирования используются разные ключи, атакующей стороне придется каждый раз проводить заново процесс расшифровывания для каждого контейнера.

- Атака на основе известной математической модели контейнера или его части

Алгоритм в некотором роде уязвим в такого рода атакам, однако по причинам, описанным в предыдущем пункте, расшифрование контейнера - достаточно трудоемкий процесс.

2.3. Выводы по разделу

Был придуман алгоритм кодирования сообщений, использующий текстовую стеганографию.

Из положительных качеств алгоритма можно выделить следующие особенности:

- Высокий уровень качества сокрытия и необнаружимости, за счет использования популярных в социальных сетях эмотиконов в качестве кодирующих элементов.
- Устойчивость к большинству видов атак за счет постоянного обновления ключа кодирования, который зависит от времени, и использования хеша времени отправки в качестве подписи.

Из отрицательных качеств можно выделить достаточно низкий показатель уровня “сигнал-шум” и уязвимость к атаке по известному сообщению.

ГЛАВА 3. КЛИЕНТ КАНАЛА СВЯЗИ В КОНТЕКСТЕ АЛГОРИТМА КОДИРОВАНИЯ ТЕКСТА

Одной из задач данной работы является написание десктопного клиента для социальной сети “ВКонтакте”, работающего на операционной системе Windows и поддерживающего выбранный алгоритм сокрытия данных.

Для реализации программы была выбрана платформа .NET Framework, как одна из наиболее популярных платформ, рассчитанных на работу под Windows. Также реализация классов и методов на данной платформе допускает их совместимость с кодом, написанным на JScript.NET(поставляется вместе с Microsoft Visual Studio) и PHP (активно поддерживается Microsoft Visual Studio). Это дает возможность развить приложение под, например, расширение для какого-либо браузера, что, в принципе, облегчит работу пользователя.

Язык C# был выбран прежде всего из-за его востребованности на рынке труда, однако, не меньшую роль имеют следующие факторы:

- «улучшенная» объектная ориентированность

В C# для объектов, которые должны быть видны в пределах сборки введен отдельный модификатор internal, a protected сохраняет свой изначальный смысл, взятый из C++ — доступ только из классов-потомков.

В C# в дополнение к примитивным типам передаются по значению структуры (struct) (т. н. значимые типы), остальные типы передаются по ссылке (т. н. ссылочные типы).

В C# запрещается давать методам наименование, совпадающее с названием класса, что предотвращает ошибки (например, в Java программист может определить конструктор, который будет на самом деле являться методом).

- компонентно-ориентированное программирование;

Компонентно-ориентированное программирование (КОП = COP = component-oriented programming) возникло как своего рода дисциплина, т.е. набор определенных ограничений, налагаемых на механизм ООП, когда стало ясно, что бесконтрольное использование ООП приводит к проблемам с надежностью больших программных комплексов.

Это, так называемая, проблема хрупких базовых типов [fragile base class problem]; проблема может проявиться при попытке изменить реализацию типа-предка, когда может оказаться, что изменить реализацию типа-предка даже при неизменных интерфейсах его методов невозможно, не нарушив корректность функционирования типов-потомков.

•безопасный (по сравнению с языками C и C++) код;

Одна из самых интересных функций языка C# заключается в поддержке кода, не являющегося строго типизированным. Обычно среда CLR отслеживает выполнение кода MSIL и предотвращает все сомнительные операции. Однако в некоторых случаях может потребоваться прямой доступ к низкоуровневым функциям, например вызовам Win32 API, и программист может сделать это, взяв на себя ответственность за обеспечение правильной работы этого кода. Такой код необходимо помещать внутрь небезопасных блоков в исходном коде.

•унифицированная система типизации;

Common Type System (сокр. CTS, рус. Стандартная система типов) — формальная спецификация, определяющая, как какой-либо тип (класс, интерфейс, структура, встроенный тип данных) должен быть определён для его правильного выполнения средой .NET. Кроме того, данный стандарт определяет, как определения типов и специальные значения типов представлены в компьютерной памяти. Целью разработки CTS было обеспечение возможности программам, написанным на различных языках программирования, легко обмениваться информацией. Как это принято в языках программирования, тип может быть описан как определение набора допустимых значений (например, «все целые от 0 до 10») и допустимых операций над этими значениями (например, сложение и вычитание).

- поддержка событийно-ориентированного программирования;

Событийно - ориентированное программирование - парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета);

3.1. Требования к разрабатываемому клиенту

Конечным продуктом данной работы является программное обеспечение - десктопный клиент для социальной сети “ВКонтакте”.

Данный клиент обладает следующим функционалом:

1. Шифрование и расшифрование сообщений, введенных пользователем.
2. Отправка и получение сообщений на стену пользователя, группы.
3. Сбор сообщений пользователей, групп со стен для дальнейшего использования в качестве пустых стегоконтейнеров.
4. Изменение словаря “символ - эмотикон”.
5. Авторизация в ВК под разными пользователями.
6. Один символ может кодироваться разными эмотиконами - из разных эмоций.
7. Пользователь может выбирать эмоциональную окраску предложения, в зависимости от чего выбирается эмотикон для шифрования.

Принцип работы данного приложения заключается в замене букв сообщения на эмотиконы из словаря (стандартный набор эмотиконов) и дальнейшей их вставке в текст. Далее, текст с эмотиконами посылается на стену пользователя или группы ВКонтакте. Все пользователи, имеющие экземпляр данного приложения могут получить закодированное сообщение.

Алгоритм работы приложения:

1. Пользователь вводит сообщение для кодирования
2. Программа, используя словарь “эмотикон - буква” кодирует сообщение в список эмотиконов.
3. Программа размещает в пустом текстовом контейнере эмотиконы.
4. Заполненный контейнер отправляется на сервер.

Приведем схему алгоритма работы приложения (рис. 3.1.)

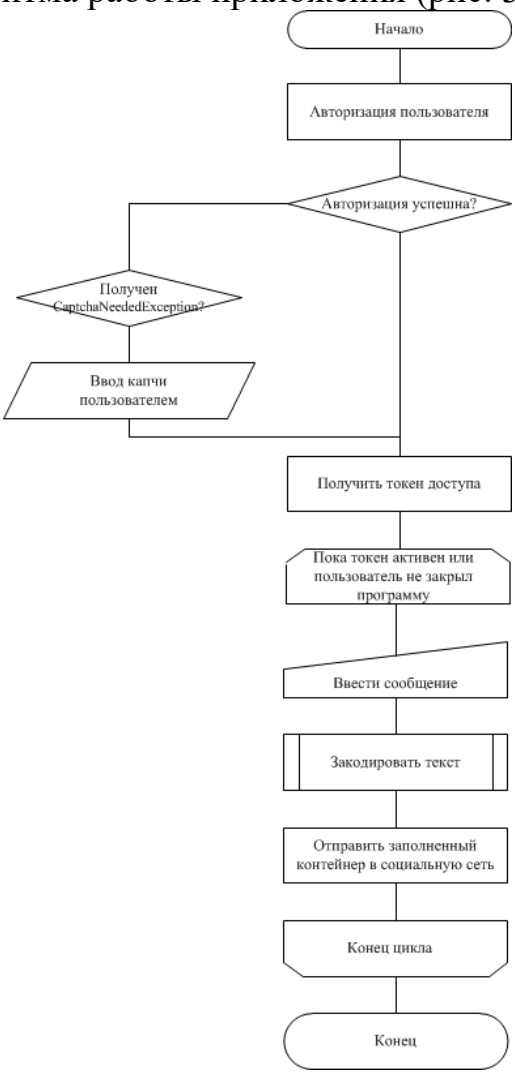

Рис. 3.1. Схема алгоритма программного обеспечения

3.2. Описание архитектуры программы

Таким образом, разработка всего проекта разбивается на две части - разработку непосредственно программного продукта - десктопного клиента под ОС Windows и разработку оболочки программного интерфейса социальной сети ВКонтакте на языке C#.

3.2.1. Описание C# VK API

Для реализации данного функционала было решено разработать собственную оболочку API ВКонтакте на языке C#.

Для минимальной работоспособности клиента, оболочка API ВКонтакте должна выполнять следующие задачи:

1. Получение токена доступа пользователя.

Токен доступа (авторизации) - строка, которую получает пользователь при авторизации и которую он должен передавать в теле запроса, требующего наличии авторизации. К примеру, получить записи со стены открытой группы можно и без авторизации в сети, однако, неавторизованный пользователь не может написать сообщение кому-либо.

2. Получение основной информации пользователя, группы.
3. Отправка сообщения на стену пользователя, группы.
4. Удаление сообщения со стены пользователя, группы.

Дополнительно можно реализовать следующие функции:

1. Прикрепление медиаконтента (аудио, видео, изображение) к сообщению.

Добавление различного рода приложений к тексту может сделать контейнер еще менее подозрительным.

2. Работа с комментариями к записям.

Если отправлять заполненный контейнер в комментарии к какой-либо записи (особенно в те, где уже есть комментарии), можно сделать контейнер еще менее заметным.

Порядок действий при работе с VK Api выглядит следующим образом:

1. Открытие диалога авторизации.

Браузер пользователя перенаправляется по адресу: https://oauth.vk.com/authorize вместе со следующими параметрами (рис 3.2.).

- client_id - Идентификатор приложения. Обязательный параметр.
- redirect_uri - Адрес, на который будет передан code (домен указанного адреса должен соответствовать основному домену в настройках приложения и перечисленным значениям в списке доверенных redirect uri — адреса сравниваются вплоть до path-части). Обязательный параметр.

- display - Указывает тип отображения страницы авторизации. Поддерживаются следующие варианты:
- scope - Битовая маска настроек доступа приложения, которые необходимо проверить при авторизации пользователя и запросить отсутствующие.
- response_type - Тип ответа, который нужно получить. В случае авторизации используется тип “code”.
- v - Версия API, которая используется в приложении.
- state - Произвольная строка, которая будет возвращена вместе с результатом авторизации.

```
https://oauth.vk.com/authorize?
client_id=1&display=page&redirect_uri=http://example.com/callback&scope=friends&
response_type=code&v=5.60
```

Рис. 3.2. Пример запроса авторизации

2. Разрешение прав доступа.

После успешного входа на сайт пользователю будет предложено авторизовать приложение, разрешив доступ к необходимым настройкам, запрошенным при помощи параметра scope.

3. Получение code.

После успешной авторизации приложения браузер пользователя будет перенаправлен по адресу redirect_uri, указанному при открытии диалога авторизации. При этом код для получения ключа доступа code будет передан в GET-параметре на указанный адрес:

REDIRECT_URI?code=7a6fa4dff77a228eeda56603b8f53806c883f011c40b72630bb50df056f6479e52a

Параметр code может быть использован в течение 1 часа для получения ключа доступа к API access_token с сервера.

В случае возникновения ошибки браузер пользователя будет перенаправлен с кодом и описанием ошибки:

REDIRECT_URI?error=invalid_request&error_description=Invalid+display+parameter

4. Получение access_token

Для получения access_token необходимо выполнить запрос (рис. 3.3.) с сервера на https://oauth.vk.com/access_token, передав следующие параметры:

client_id - Идентификатор приложения. Обязательный параметр.

client_secret - Защищенный ключ приложения (указан в настройках приложения). Обязательный параметр.

redirect_uri - URL, который использовался при получении code на первом этапе авторизации. Должен быть аналогичен переданному при авторизации. Обязательный параметр.

code - Временный код, полученный после прохождения авторизации. Обязательный параметр.

```
https://oauth.vk.com/access_token?
client_id=1&client_secret=H2Pk8htyFD8024mZaPHm&redirect_uri=http://mysite.ru&cod
e=7a6fa4dff77a228eeda56603b8f53806c883f011c40b72630bb50df056f6479e52a
```

Рис. 3.3. Пример запроса для получения access token.

В результате выполнения данного запроса сервер получит (рис. 3.4.) вновь созданный access_token. Вместе с access_token серверу возвращается время жизни ключа expires_in в секундах. Процедуру авторизации приложения необходимо повторять в случае истечения срока действия access_token, смены пользователем своего логина или пароля или удаления приложения из настроек.

```
{"access_token":"533bacf01e11f55b536a565b57531ac114461ae8736d6506a3", "expires_in":43200,
"user_id":66748}
```

Рис. 3.4. Пример ответа на запрос.

5. Работа с методами VK Api

После получения токена доступа, пользователь может выполнять любые операции, разрешенные настройками доступа приложения.

Однако для корректной работы программы достаточно реализовать следующие методы:

1. Wall.post

Позволяет создать запись на стене, предложить запись на стене публичной страницы, опубликовать существующую отложенную запись. После успешного выполнения возвращает идентификатор созданной записи (post_id).

Коды ошибок:

- 214 - Публикация запрещена. Превышен лимит на число публикаций в сутки, либо на указанное время уже запланирована другая запись, либо для текущего пользователя недоступно размещение записи на этой стене.

2. Wall.edit

Редактирует запись на стене. После успешного выполнения возвращает 1.

Параметры:

- owner_id - идентификатор пользователя или сообщества, на стене которого должна быть опубликована запись.
- friends_only - 1 — запись будет доступна только друзьям, 0 — всем пользователям. По умолчанию публикуемые записи доступны всем пользователям.
- from_group - данный параметр учитывается, если owner_id < 0 (запись публикуется на стене группы). 1 — запись будет опубликована от имени группы, 0 — запись будет опубликована от имени пользователя (по умолчанию).
- message - текст сообщения (является обязательным, если не задан параметр attachments) .
- attachments - список объектов, приложенных к записи и разделённых символом ",".

Коды ошибок:

- 220 - Слишком много получателей.
- 222 - Запрещено размещать ссылки

3. Wall.delete

Удаляет запись со стены. После успешного выполнения возвращает 1.

Параметры:

- owner_id - идентификатор пользователя или сообщества, на стене которого находится запись.
- post_id - идентификатор записи на стене.

Коды ошибок:

- 210 - Нет доступа к записи.

4. Wall.getComments

Возвращает список комментариев к записи на стене. После успешного выполнения возвращает объект, содержащий число результатов в поле count и массив объектов комментариев в поле items.

Параметры

- owner_id - идентификатор владельца страницы (пользователь или сообщество).
- post_id - идентификатор записи на стене.
- offset - сдвиг, необходимый для получения конкретной выборки результатов.
- count - число комментариев, которые необходимо получить. По умолчанию: 10, максимальное значение: 100.

Коды ошибок:

- 212 - Нет доступа к комментариям записи.

5. Wall.createComment

Добавляет комментарий к записи на стене. После успешного выполнения возвращает идентификатор добавленного комментария.

Параметры:

- owner_id - идентификатор пользователя или сообщества, на чьей стене находится запись, к которой необходимо добавить комментарий.
- post_id - идентификатор записи на стене.
- message - текст комментария. новый текст комментария. Обязательный параметр, если не передан параметр attachments.
- attachments - список объектов, приложенных к комментарию и разделённых символом ",".

Коды ошибок:

- 213 - Нет доступа к комментированию записи
- 223 - Превышен лимит комментариев на стене

3.3. Описание интерфейса программного обеспечения

Главное окно приложения должно содержать три вкладки: “Отправка”, “Чтение”, “Сбор”.

Вкладка “Отправка” (рис. 3.6.) содержит текстовое поле, в которое пользователь вводит сообщение, которое он хочет передать. Далее, он вводит ссылку на стену, где он хочет опубликовать свое сообщение и нажимает кнопку “Отправка”, после чего, сообщение кодируется и отправляется на сервер.

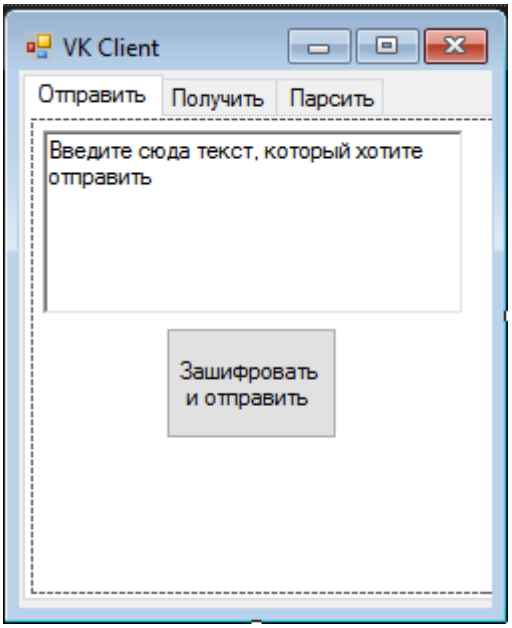

Рис.3.6. Вкладка “Отправка”

Закодированные сообщения (рис. 3.7.) видят все пользователи, которые имеют доступ к стене, на котором опубликовано данное сообщение.

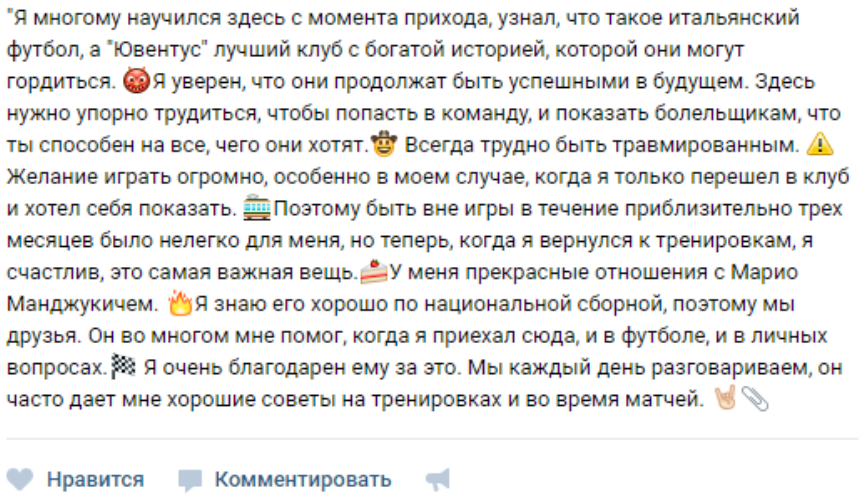

Рис. 3.7. Пример заполненных контейнеров на стене пользователя

Вкладка “Чтение” предназначена непосредственно для получения закодированных сообщений. Пользователь вводит ссылку на стену, нажимает кнопку “Получить”, в результате чего программа получает список сообщений со стены, декодирует нужные. Далее, в текстовом поле появляется список, содержащий все декодированные сообщения, которые удалось получить по данной ссылке. Вкладка “Чтение” выглядит следующим образом.

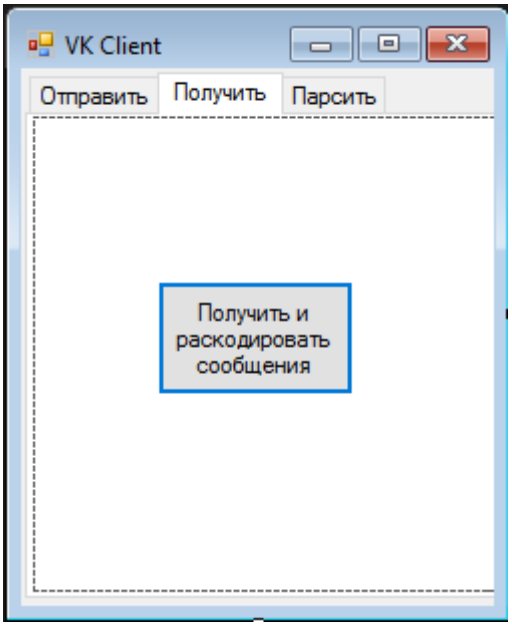

Рис. 3.7. Вкладка “Чтение”

Вкладка “Сбор” предназначена для работы над сбором пустых стегоконтейнеров со стен различных групп или пользователей. Пользователь вводит ссылку на стену, с которой будет производиться сбор пустых стегоконтейнеров - текста достаточной длины, длина контролируется пользователем. После удачного сбора появляется всплывающее окно с указанием количества собранных контейнеров.

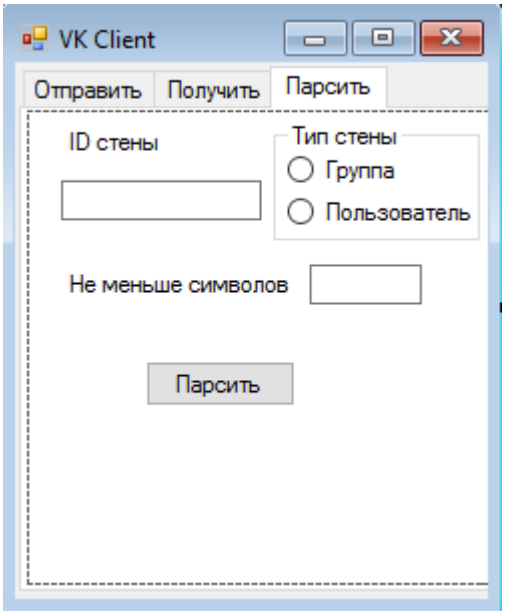

Рис. 3.8. Вкладка “Сбор контейнеров”

Так же в программном обеспечении предусмотрен случай, когда при авторизации необходимо ввести капчу.

3.4. Структура проекта

В процессе разработки использовалась методология Domain Driven Development. Решение было разбито на логические единицы (рис. 3.9.), представленные ниже.

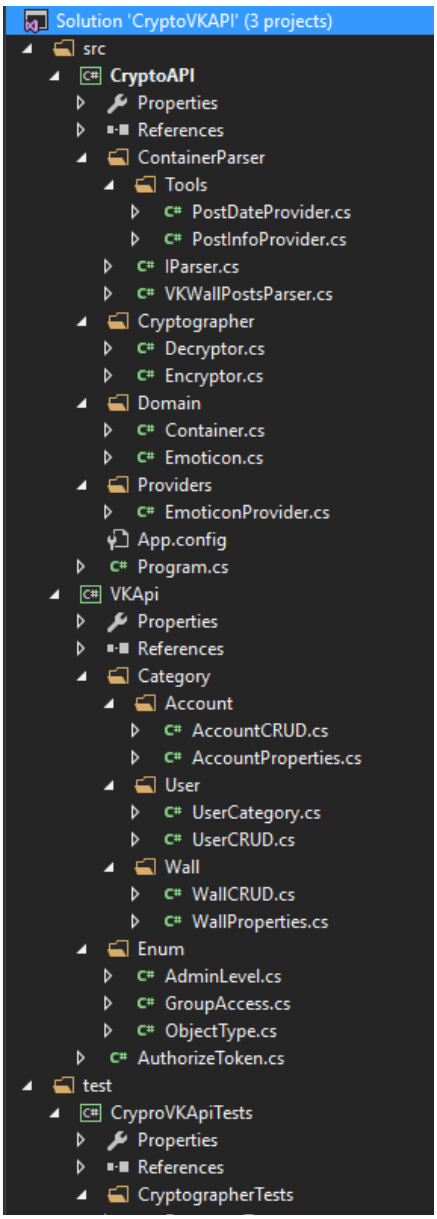

Рис. 3.9. Структура проекта клиентского приложения.

- Библиотека классов CryptoAPI.

Данная библиотека классов содержит необходимые инструменты для кодирования/декодирования сообщений и разработки новых алгоритмов. В процессе разработки эту библиотеку классов можно отдельно подключить к любым проектам.

В процессе реализации были выделены следующие классы:

- Cryptographer - класс непосредственно отвечающий за кодирование и декодирование контейнеров. Наследуется от интерфейсов IEncryptor и IDecryptor.

Реализация интерфейса IEncryptor - метод `string Encrypt(string textToEncrypt)`. Метод принимает строковую переменную - текст сообщения, который необходимо закодировать. Возвращаемое значение - заполненный контейнер, содержащий закодированное сообщение.

Реализация интерфейса IDecryptor - метод `string Decrypt(string textToDecrypt)`. Метод принимает строковую переменную - текст заполненного контейнера, который необходимо раскодировать. Возвращаемое значение - раскодированное сообщение.

- PostGrabber - класс, реализующий API парсинга пустых контейнеров нужного вида. Наследуется от интерфейса IGrabber.

Реализация интерфейса IGrabber - метод void GrabPosts(int wallID, int containerSize). Метод принимает идентификатор пользователя/группы со стены которой происходит сбор контейнеров и размер контейнера, в данном случае - необходимое количество предложений в контейнере.

- DataAccess - класс для доступа к сохраненным пустым контейнерам в формате json.

Члены этого класса:

public IEnumerable<string> Posts - содержит в себе список пустых контейнеров.

public IEnumerable<string> EmotCodes - содержит в себе список кодов эмотиконов.

3.5. Выводы по разделу

Было разработано программное обеспечение – клиент для социальной сети ВКонтакте, использующий разработанный алгоритм кодирования текстовых сообщений.

- Логирование.

Все ошибки, исключения, выбрасываемые программным кодом, логируются при помощи платформы логирования NLog в файл логов.

Файл логов - текстовый файл с названием в виже <Дата>-<ClientLogs>.txt. Он содержит в себе строки в формате <Дата и время> : <Сообщение исключения> - <Трассировка стека вызовов>

- Тестирование.

Разработка велась с использованием методологии Test Driven Development, были разработаны модульные тесты при помощи фреймворка NUnit. Исходный код тестов находится в Приложении 2.

ЗАКЛЮЧЕНИЕ

Обеспечение безопасного канала связи является одной из важнейших задач криптографии. Эта задача обычно решается либо с помощью дорогостоящих технических средств, либо при помощи сложных криптографических алгоритмов. Однако использование подобного рода решений является достаточно подозрительным для атакующей стороны, и обладая знанием о канале передачи данных, она может так или иначе добиться компрометации канала.

Было принято решение разработать алгоритм, который позволит скрывать сам факт существования передачи сообщений, в связи с чем, были выбраны алгоритмы текстовой стеганографии для исследования возможности их использования в процессе решения задачи обеспечения безопасного канала связи.

Рассмотрены алгоритмы текстовой стеганографии, их плюсы и минусы. Алгоритмы текстовой стеганографии обладают невысокой пропускной способностью, но нужно использовать достаточно нетривиальные методы, чтобы алгоритм обладал достаточным показателем незаметности.

Так же, были рассмотрены различные типы атак на стеганографические системы, почти все алгоритмы текстовой стеганографии уязвимы к атаке по известному шифротексту.

Рассмотрены существующие решения, основными проблемами которых является низкий уровень незаметности и отсутствие возможности использовать в публичных каналах связи.

В процессе выполнения работы был реализован безопасный канал связи, в социальной сети ВКонтакте. Был использован самостоятельно разработанный алгоритм текстовой стеганографии, подразумевающий замену символов сообщения на эмотиконы, причем набор эмотиконов зависит времени написания сообщения.

Преимуществом разработанного алгоритма является высокая степень незаметности, учитывая всеобщее использование пользователями эмотиконов в их сообщениях, возможность встраивания для использования почти в любой социальной сети и любом канале связи использующий текстовые контейнеры.

Недостатком алгоритма является отсутствие фильтрации эмотиконов в зависимости от эмоциональной окраски текстового контейнера, из-за чего, случается ситуация, когда текст не совсем сочетается с эмотиконами.

Было разработано программное обеспечение на языке C#, платформе .NET позволяющее пользоваться данным каналом и автоматизирующее процесс кодирования и декодирования заполненных контейнеров. Также, разработанное программное обеспечение позволяет автоматически собирать пустые контейнеры подходящего размера.

В процессе разработки были написаны юнит-тесты с использованием платформы NLog, разработка велась по принципам Test Driven Development, Domain Driven Development, SOLID. Также имеется функция логирования ошибок.

Для реализации данного программного обеспечения был разработан базовый функционал C# VK Api - обертки программного интерфейса социальной сети ВКонтакте, в частности методы авторизации, отправления/получения постов.

Направления развития проекта:

1. Реализация собственной хэш-функции
2. Автоматическое определение эмоциональной окраски текста, возможно с помощью нейронных сетей.
3. Использование медиафайлов в качестве стегоконтейнеров.

ЛИТЕРАТУРА

1. Cross, M. Scene of the Cybercrime / M. Cross, D.L. Shinder. — Massachusetts: Syngress Publishing Inc, 2008. — 704 p.
2. Last, M. Web Intelligence and Security / M. Last, A. Kandel. — Amsterdam: IOS Press, 2010. — 280 p.
3. Gupta, M. Handbook of Research on Social and Organizational Liabilities in Information Security / M. Gupta, R. Sharman. — Hershey: Information Science Reference, 2008. — 596 p.
4. Kizza, J. Ethical and Social Issues in the Information Age / J. M. Kizza. — New-York: Springer, 2013, — 372 p.
5. Alam, M. Recent Developments In Computing And Its Applications / M. A. Alam, T. Siddiqui, K.R. Seeja. — New Delhi: IK International Publishing House, 2009. — 616 p.
6. Martin, K. Everyday Cryptography: Fundamental Principles and Applications / K.M. Martin. — New Delhi: Oxford University Press, 2012. — 552 p.
7. Information Hiding: 6th International Workshop: revised selected papers / J. Fridrich. — New-York: Springer , 2004. — 381 p.
8. Goje, A. Proceedings Of The 2nd National Conference On Emerging Trends In Information Technology / A.C. Goje, S.S. Gornale, P.L. Yannawar. — New Delhi: IK International Publishing House, 2007. — 368 p.
9. Shih, F. Image Processing and Pattern Recognition: fundamentals and Techniques / F. Shih. — New-Jersey: John Wiley & Sons, 2010. — 552 p.
10. Zelkowitz, M, Security on the Web / M. Zelkowitz. — London: Academic Press, 2011. — 368 p.
11. Arnold, M. Techniques and Applications of Digital Watermarking and Content Protection / M.K. Arnold, M. Schmuker, S,D, Wolthusen. — Massachusetts: Artech House, 2003. — 296 p.
12. Chapman, M. Hiding the Hidden: A Software System for Concealing Ciphertext as Innocuous Text / M. T. Chapman. — Milwaukee: University of Wisconsin-Milwaukee, 1998. — 83 p.
13. Wayner, P. Mimic Functions / P. Wayner // Cryptologia XVI. — Abingdon: Taylor & Francis. — 1992. — V. 16, №3. — P. 193 — 214.
14. Wayner, P. Strong theoretical steganography / P. Wayner // Cryptologia XIX. — Abingdon: Taylor & Francis. — 1995. — V. 19, №3. — P. 285 — 299.
15. Wayner, P. Disappearing Cryptography: Information Hiding: Steganography & Watermarking / P. Wayner. — San-Francisco: Morgan Kaufmann, 2008. — 456 p.
16. Johnson, N. “Steganalysis” In Information Hiding: Techniques for Steganography and Digital Watermarking / N.F. Johnson. — Boston: Artech House, 2000. — 267 p.
17. Bennet K. Linguistic steganography: Survey, analysis, and robustness concerns for hiding information in text / K. Bennet. — Lafayette: Purdue University Press, 2004. — 30 p.

18. Bolshakov, I. A method of linguistic steganography based on collocationally-verified synonymy / I. A. Bolshakov // Information Hiding: 6th International Workshop. New-York: Springer. — 2004. — V. 4. — P. 180 — 191.
19. Singh H. Survey on Text Based Steganography / H. Singh, P.K. K. Singh, K.A. Saroha // Proceedings of the 3rd National Conference INDIACom-2009. New Delhi: Amity University Press. — 2009. — P. 3–9.
20. Kaleem, M. An Overview of Various Forms of Linguistic Steganography and Their Applications in Protecting Data / M.K. Kaleem // Journal of Global Research in Computer Science. New Delhi: JGRCS. — 2012. — V. 3. — №5. — P. 33 — 38.
21. Chan, W. Modified Linguistic Steganography Approach by Using Syntax Bank and Digital Signature / W.E. Chan, K.M. Aye // International Journal of Information and Education Technology. Singapore: IJIET. — 2011. — V.1 .— №5. — P.410 — 415.
22. Алиев А.Т. Лингвистическая стеганография на основе замены синонимов для текстов на русском языке / А.Т. Алиев // Известия ЮФУ. Технические науки . – 2010. –№11.
23. Грибунин В. Г. Цифровая стеганография / В. Г. Грибунин, И. Н. Оков, И. В. Туринцев. – М.: Солон-Пресс, 2009. – 265 с.
24. Ляшевская, О.Н. Частотный словарь современного русского языка (на материалах Национального корпуса русского языка) / О.Н. Ляшевская, С.А. Шаров. – М.: Азбуковник, 2009. – 1087 с.
25. Nagarhalli, T.P. A New Approach to SMS Text Steganography using Emoticons. – Mumbai: IJCA Proceedings on National Conference on Role of Engineers in National Building, 2014. – 3p.
26. An Overview of Steganography for the Computer Forensics Examiner . — Дата обновления: 17.02. 2015. URL: http://www.garykessler.net/library/fsc_stego.html (дата обращения: 07.07.2017).
27. An Overview of Cryptography . — Дата обновления: 27.04.2017. URL: http://www.garykessler.net/library/crypto.html (дата обращения: 07.07.2017).
28. Steganography: Hiding Data Within Data. — Дата обновления: 09.09.2001. URL: http://www.garykessler.net/library/steganography.html. Дата обращения (07.07.2017).
29. Стеганография в World Of Warcraft. - Дата обновления: 06.02.2013. URL: http://www.nestego.ru/2013/02/world-of-warcraft.html (дата обращения: 07.07.2017).
30. Delfs, H. Introduction to Cryptography: Principles and Applications / H. Delfs, H. Knebl. – New-York: Springer, 2015. – 508 p.
31. Freeman, A. Pro LINQ: Language Integrated Query in C# 2010 / A. Freeman, J.C. Ratts. – New-York: Apress, 2010. – 840 p.
32. Troelsen, A. Pro C# 5.0 and the .NET 4.5 Framework / A. Troelsen. – New-York: Apress, 2012. – 1560 p.
33. Albahari, J. C# 5.0 in a Nutshell: The Definitive Reference / J. Albahari, B. Albahari. – Sebastopol: O'Reilly Media, 2012. – 1064 p.

34. Greene, J. Head First C#: A Learner's Guide to Real-World Programming with C#, XAML and .NET / J. Greene, A. Stellman. – Sebastopol: O'Reilly Media, 2012. – 1100 p.
35. Skeet, J. C# in Depth / J. Skeet. – Greenwich: Manning Publications, 2013. – 616 p.
36. Griffiths, I. Programming C# 5.0: Building Windows 8, Web, and Desktop Applications for the .NET 4.5 Framework / I. Griffiths. – Sebastopol: O'Reilly Media, 2012. – 886 p.
37. McLean Hall, G. Adaptive Code via C#: Agile coding with design patterns and SOLID principles / G. McLean Hall. – Washington: Microsoft Press, 2014. – 448 p.
38. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. / Б. Шнайер. – М.: Триумф, 2002. – 816 с.
39. Смарт, Н. Криптография / Н. Смарт, под ред. С.К. Ландо. – М.: Техносфера, 2005. – 528 с.
40. Coutinho, S.C. The Mathematics of Ciphers: Number Theory and RSA Cryptography / S.C. Coutinho. – Boca Raton: A K Peters/CRC Press, 1999. – 198 p.
41. Hoffstein, J. An Introduction to Mathematical Cryptography / J. Hoffstein, J. Pipher, J.H. Silverman. – New-York: Springer, 2010. – 524 p.
42. Баричев, С.Г. Основы современной криптографии / С.Г. Баричев, Р.Е. Серов. – М.: «Горячая Линия - Телеком», 2011. – 176 с.
43. Грибунин В.Г., Оков И.Н., Туринцев И.В. Цифровая стеганография. – М.: Солон-Пресс, 2002. – 272 с.
44. Конахович Г.Ф., Пузыренко А.Ю. Компьютерная стеганография. Теория и практика. – Киев: МК-Пресс, 2006. – 288 с.
45. Avery Li-chun Wang. An Industrial-Strength Audio Search Algorithm. [Электронный ресурс] – Режим доступа: http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf (дата обращения 2.06.2017).
46. Bender W., Gruhl D., Morimoto N., Lu A. Techniques or Data Hiding // IBM Systems Journal. 1996. № 35.
47. J. Foote, J. Adco, and A. Girgensohn. Time base modulation: a new approach to watermarking audio. [Электронный ресурс] – Режим доступа: http://www.fxpal.com/publications/FXPAL-PR-03-212.pdf (дата обращения 2.06.2017).
48. Jaap Haitsma, Ton Kalker. A Highly Robust Audio Fingerprinting System. [Электронный ресурс] – Режим доступа: http://ismir2002.ismir.net/proceedings/02-FP042.pdf (дата обращения 2.06.2017).
49. Cormen T.H., Leiserson C.H., Rivest R.L., "Introduction To Algorithms", MIT Press, ISBN 0-262-53091-0, 1998
50. William H. Press, Saul A. Teukolsky, William T. Vetterling at all. Numerical recipes in C. The Art of Scientific Computing. Cambridge: Cambridge University Press, 2002. 2-nd edition.

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

П1.1. ФАЙЛ “ENCODER.CS”

```
namespace CryptoAPI.API.Encoding
{
   using System;
   using System.Linq;
   using System.Text;

   public class Encoder : IEncoder
   {
      public string Encode(string textToEncrypt)
      {
         var data = new DataAccess();

         var r = new Random();
         var q = r.Next();
         q %= data.Posts.Count;
         var container = new StringBuilder(data.Posts[q]);
         var dotCount = container.ToString().Count(f => f == '.');

         var emotPerDot = textToEncrypt.Length/dotCount;
         var lastEmot = textToEncrypt.Length - dotCount * emotPerDot;

         var coolTextToEncrypt = textToEncrypt.ToLower().Replace(" ", "");
         for (var index = 0; index < coolTextToEncrypt.Length; index++)
         {
            var insertPosition = 0;
            for (var i = 0; i < dotCount; i++)
            {
               insertPosition = container.ToString().IndexOf('.', insertPosition);
               if (i == dotCount - 1)
                  emotPerDot = coolTextToEncrypt.Length - emotPerDot*(dotCount - 1);
               for (var j = 0; j < emotPerDot; j++)
               {
                  var c = coolTextToEncrypt[index];
                  var qq = c - 1072;
                  container.Insert(insertPosition + 1, data.EmotCodes[qq]);
                  insertPosition += 9;
                  index++;
               }
            }
         }
         return container.ToString();
      }
   }
}
```

П.1.2. ФАЙЛ “DECODER.CS”

```
namespace CryptoAPI.API.Encoding
{
   using System.Text;
   using System.Text.RegularExpressions;

   public class Decoder : IDecoder
   {
      public string Decode(string textToDecrypt)
      {
         var data = new DataAccess();

         var sb = new StringBuilder();

         var regex = new Regex("&#\\d+;");
         var match = regex.Match(textToDecrypt);

         while (match.Success)
         {
            var matchEntrance = match.Groups[0].Value;

            var index = data.EmotCodes.FindIndex(a => a == matchEntrance);

            sb.Append('a' + index);

            match = match.NextMatch();
         }

         return sb.ToString();
      }
   }
}
```

П.1.3. ФАЙЛ “AUTHFORM.CS”

```
namespace YOLOMAG.Forms
{
   using System;
   using System.Windows.Forms;
   using VKApi;
   using VkApi.Exception;

   public partial class AuthForm : Form
   {
      private const ulong AppID = 4999838;
      private ApiAuthParams auth;

      public AuthForm()
      {
         InitializeComponent();
         auth = new ApiAuthParams();
      }

      private void button1_Click(object sender, EventArgs e)
```

```
        {
            var scope = Settings.All;
            var api = new VkApi();
            auth.Login = textBox1.Text;
            auth.Password = textBox2.Text;
            auth.ApplicationId = AppID;
            auth.Settings = scope;

            try
            {
                api.Authorize(auth);
                new Form1(api).Show();
            }
            catch (CaptchaNeededException ex)
            {
                var url = ex.Img;
                var ID = ex.Sid;
                using (var form = new CaptchaForm(url, ID, api))
                {
                    var res = form.ShowDialog();
                    if (res == DialogResult.OK)
                    {
                        var val = form.CaptchaReturnValue;
                        auth.CaptchaKey = val;
                        auth.CaptchaSid = ID;
                    }
                }
            }
            catch (Exception ex)
            {
                if (ex.GetType() == typeof(VkApiAuthorizationException))
                {
                    label1.Text = @"Неправильный логин или пароль";
                    label1.Visible = true;
                }
                else
                {
                    label1.Text = @"Неизвестная ошибка";
                    MessageBox.Show(ex.Message);
                    label1.Visible = true;
                }
            }
        }
    }
}
```

П1.4. ФАЙЛ “CAPTCHAFORM.CS”

```
namespace YOLOMAG.Forms
{
    using System;
    using System.Drawing;
    using System.IO;
    using System.Net;
    using System.Windows.Forms;
```

```
    using VkNet;

    public partial class CaptchaForm : Form
    {
        public string CaptchaReturnValue { get; set; }
        public CaptchaForm(Uri uri, long ID, VkApi vk)
        {
            InitializeComponent();
            var client = new WebClient();
            var image = Image.FromStream(new MemoryStream(client.DownloadData(uri)));
            captchaPictureBox.Image = image;
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.CaptchaReturnValue = textBox1.Text;
            this.DialogResult = DialogResult.OK;
            this.Close();
        }
    }
}
```

П1.5. ФАЙЛ “MAINFORM.CS”

```
namespace YOLOMAG.Forms
{
    using System;
    using System.Windows.Forms;
    using VKApi;
    using VkApi.RequestParameters;

    public partial class Form1 : Form
    {
        private readonly VkApi _api;

        public Form1(VkApi api)
        {
            _api = api;
            InitializeComponent();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            var groupID = Convert.ToInt64(textBox1.Text);
            if (radioButton1.Checked)
                groupID *= -1;
            var posts = _api.Wall.Get(new WallGetParams
            {
                OwnerId = groupID,
                Count = 100
            }, true);

            var saver = new PostGrabber();
            foreach (var wallPost in posts.WallPosts)
                if (wallPost.Text.Length > Convert.ToInt32(textBox2.Text))
                    saver.SaveToFile(wallPost.Text);
```

```
            MessageBox.Show(@"Done!");
        }
        private void button1_Click(object sender, EventArgs e)
        {
            var cryptographer = new Encoder();
            var qq = cryptographer.Encode(richTextBox1.Text);
            try
            {
                _api.Wall.Post(new WallPostParams
                {
                    OwnerId = -107415817,
                    Message = qq
                });
                MessageBox.Show(@"We gucchi");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        private void button3_Click(object sender, EventArgs e)
        {
            try
            {
                var post = _api.Wall.Get(new WallGetParams
                {
                    Count = 2,
                    OwnerId = -107415817
                });
                var res = new Decoder().Decode(post.WallPosts[1].Text);
                MessageBox.Show(res);
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

ПРИЛОЖЕНИЕ 2. ОПИСАНИЕ ПРОГРАММЫ

П2.1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

Название программного продукта – “Клиент канала связи социальной сети ВКонтакте”.

Программа написана на языке C# под платформу .NET. Для использования программы необходима операционная система Microsoft Windows XP/Vista/7/8/8.1/10.

П2.2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

Программа предназначена работы с безопасным каналом связи в социальной сети “ВКонтакте”.

- Программа обладает следующими возможностями.
- Сбор пустых текстовых контейнеров определенного размера.
- Кодирование/декодирование сообщений.

Отправка/получение постов со стен пользователей/групп социальной сети “ВКонтакте”.

П2.3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

Работа приложения происходит в соответствии с алгоритмом, представленным на рис. П2.1.

В файле “Encoder.cs” и “Decoder.cs” описаны разработанные алгоритмы кодирования и декодирования сообщений.

В файле “VKPostsGrabber.cs” описаны методы получение пустых текстовых контейнеров.

В файле «WallCategory» описана структура сущности “Стена” из VK API .

В файле «AccountCategory» описана структура сущности “Аккаунт” из VK API.

В файле «UserCategory» описана структура сущности “Пользователь” из VK API .

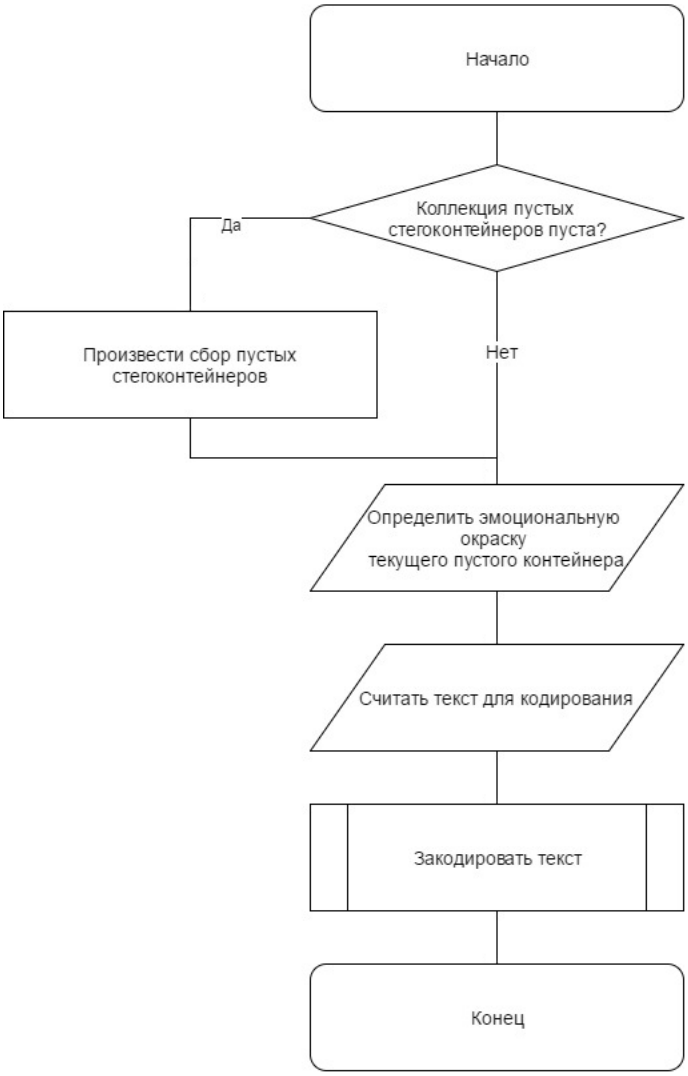

Рис. П2.1. Схема основного алгоритма программы

Схема алгоритма кодирования представлена на рис. П2.2.

Рис. 3.2. Схема алгоритма кодирования текста.

П1.4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Для работы программы пользователем рекомендуется персональный компьютер со следующей конфигурацией:

- процессор с тактовой частотой 2 ГГц;

- объем оперативной памяти 1 Гб;

На компьютере должна быть установлена операционная система не младше Windows XP.

П1.5. ВЫЗОВ И ЗАГРУЗКА

Для запуска программы необходимо открыть приложение “VKClient.exe”

П1.6. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ

Входными данными для аутентификации являются логин и пароль пользователя, а на этапе шифрования - сообщение.

Выходными данными первого этапа аутентификации является корректность логина и пароля. Для второго этапа выходными данными является заполненный текстовый контейнер.