МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Ярославский государственный университет им. П.Г. Демидова»

Кафедра компьютерной безопасности и математических методов обработки информации

Зав. кафедрой,
д.ф.-м.н., профессор
_____ В.Г. Дурнев
«__» _____ 2013 г.

Курсовая работа
«Реализация фреймворка для анализа криптограмм»

Научный руководитель
к.э.н, доцент
_____ Белова Л.Ю.
«__» _____ 2013 г.

Студент группы КБ-51
_____ Бедняков А.И.
«__» _____ 2013 г.

Ярославль 2013

Аннотация

Изучение методов защиты неразрывно связано с изучением возможных атак на алгоритмы и на их реализации. Работы по анализу таких шифров, как DES, ГОСТ 28147-89, Blowfish требуют большого ресурса и являются чрезвычайно сложными. В то же время на примерах классических шифров можно проиллюстрировать некоторые важные приемы и методы криптоанализа. После анализа классических шифров возможно изучение современных блочных алгоритмов шифрования, становятся доступными идеи линейного и дифференциального криптоанализа.

В этой работе предпринята попытка написания фреймворка для криптоанализа классических шифров и оценки стойкости современных шифров.

Содержание

1. Введение

Определение 1. *Криптограмма* (шифротекст) — результат операции шифрования.

Объектом исследования является возможность чтения шифротекста в условиях минимальной информированности о способе зашифрования. В часности, исследованы случаи неизвестного ключа при известном алгоритме шифрования и неизвестного алгоритма шифрования.

Определение 2. *Криптология* — наука, занимающаяся методами шифрования и дешифрования.

Криптология состоит из двух частей — криптографии и криптоанализа. Криптография занимается разработкой методов шифрования данных, в то время как криптоанализ занимается оценкой сильных и слабых сторон методов шифрования, а также разработкой методов, позволяющих взламывать криптосистемы. Слово «криптология» (англ. cryptology) встречается в английском языке с XVII века, и изначально означало «скрытность в речи»; в современном значении было введено американским учёным Уильямом Фридманом и популяризовано писателем Дэвидом Каном [5].

Определение 3. *Криптоанализ* — наука о методах расшифровки зашифрованной информации без предназначенного для такой расшифровки ключа.

В большинстве случаев под криптоанализом понимается выяснение ключа; криптоанализ включает также методы выявления уязвимости криптографических алгоритмов или протоколов. Первоначально методы криптоанализа основывались на лингвистических закономерностях естественного текста и реализовывались с использованием только карандаша и бумаги. Со временем в криптоанализе нарастает роль чисто математических методов, для реализации которых используются специализированное криптоаналитическое программное обеспечение.

Для исследования сферы программных методов криптоанализа реализован фреймворк, в удобной форме объединяющий все необходимое для автоматизированного анализа классических алгоритмов шифрования.

Определение 4. *Фреймворк* (англ. framework — каркас, структура) — структура программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Употребляется также слово «каркас», а некоторые авторы используют его в качестве основного, в том числе не базируясь вообще на англоязычном аналоге. Можно также говорить о каркасном подходе как о подходе к построению программ, где любая конфигурация программы строится из двух частей: первая, постоянная часть — каркас, не меняющийся от конфигурации к конфигурации и несущий в себе гнезда , в которых размещается вторая, переменная часть — сменные модули (или точки расширения) — интерфейс пользователя, программный продукт.

Для программирования выбран язык Python 3. Такой выбор обоснован ориентированностью Python на быстное портотипирование. В программировании это значит возможность быстро написать структуру приложения и затем в плотную заниматься функциональностью при необходимости переписывая узкие места алгоритма на компилируемых языках (рефакторинг).

Фреймворк использует только средства, идущие в стандартной поставке Python. Это позволяет использовать его сразу после установки интерпретатора. Основная функциональность фреймворка не завязана ни на архитектуру системы, ни на установленную операционную систему, что позволяет говорить о кроссплатформенности фреймворка. То есть, все работает в UNIX-подобных системах и запускается в Windows с незначительными потерями работоспособности.

2. Лингвистический и статистический анализ

Решение проблемы поиска открытого текста по шифровке всегда возможно свести к некоторой математической задаче. В данной работе изучается только текстовые формы шифров, поэтому мостом для перевода изначальной проблемы в математическую задачу будет служить лингвистика.

Определение 5. *Лингвистика* — наука, это наука о всех языках мира как индивидуальных его представителях. В широком смысле слова, лингвистика подразделяется на научную и практическую.

Первоначально все методы криптоанализа основывались на лингвистических закономерностях естественного текста и реализовывались с использованием только карандаша и бумаги. Со временем в криптоанализе нарастает роль чисто математических методов, и такие методы уже сформировали свой раздел в лингвистике.

Определение 6. *Компьютерная лингвистика* — научное направление в области математического и компьютерного моделирования интеллектуальных процессов у человека и животных при создании систем искусственного интеллекта, которое ставит своей целью использование математических моделей для описания естественных языков.

Современная лингвистика обладает мощными методами анализа языковых структур, в том числе методы синтеза и анализа. В этой работе внимание уделено только последним.

Определение 7. *Анализ текста* — процесс получения информации из текста на естественном языке. Как правило, для этого применяется статистическое обучение на основе шаблонов: входной текст разделяется с помощью шаблонов, затем производится обработка полученных данных.

Возможен анализ документа, написанного на неизвестном языке и/или неизвестной системой письма, но это так-же выходит за рамки данной работы.

Во время анализа шифротекста бывает полезно попробовать расшифровать текст на каком-то подмножестве ключей и посмотреть результаты. На основе того, что какой-то текст выглядит более или менее похоже на русский (или любой другой рассматриваемый язык), мы можем заключить что ключ более или менее хорош. Итак, можно вывести две интересующие нас проблемы:

1) возможность определения языка текста по шифротексту;

2) возможность определения корректности текста по самому тексту и языку его написания (метрика корректности).

Проблема 1) выглядит неразрешимо, она должна решаться для каждого шифротекста отдельно — тогда возможно стороить гипотезы на основе характеристик оппонента. В данной работе выполнен только простейший анализ шифротекста на наличие лигатур и диактрических знаков характерных для языка.

Определение 8. *Лигатура* — знак любой системы письма или фонетической транскрипции, образованный путем соединения двух и более знаков.

Определение 9. *Диактрические знаки* — различные надстрочные, подстрочные, реже внутристрочные знаки, применяемые в буквенных (в том числе консонантных) и слоговых системах письма не как самостоятельные обозначения звуков, а для изменения или уточнения значения других знаков.

Каждый язык для фреймворка выглядит как словарь (*dict* в нотации Python). Для примера, французский язык:

```
'fr' :
    {
        'alphabet': (
            ('A', 8.11), ('B', 0.91), ('C', 3.49),
            ('D', 4.27), ('E',17.22), ('F', 1.14),
            ('G', 1.09), ('H', 0.77), ('I', 7.44),
            ('J', 0.34), ('K', 0.09), ('L', 5.53),
            ('M', 2.89), ('N', 7.46), ('O', 5.38),
            ('P', 3.02), ('Q', 0.99), ('R', 7.05),
            ('S', 8.04), ('T', 6.99), ('U', 5.65),
            ('V', 1.30), ('W', 0.04), ('X', 0.44),
            ('Y', 0.27), ('Z', 0.09)
        ),
        'ligatures': (
             'À', 'Â', 'Æ', 'Ç', 'É',
             'È', 'Ê', 'Î', 'Ï', 'Ô',
             'Œ', 'Ù', 'Û', 'Ü', 'Ÿ'
        ),
        'kappa':0.0746
    },
```

Первый ключ всегда является кодом языка по стандарту ISO 639-1:2002. Далее идет перечень всех букв с частотой их встречаемости и некоторые характеристики языка. Причем некоторые буквы имеют несколько вариантов написания (например U с четырю вариантами). Процент встречаемости встречаемости таких вариантов и служит меткой языка.

Проблема 2) более прозаична так как имеет несколько методов решения, необходимо только выбрать наиболее подходящий. В целях данного исследования реализованы простейшие методы подобного анализа и проведено сравнение их корректности и скорости работы.

Тестирование каждого метода - запуск с романом «Война и мир» Льва Николаевича Толстого в качестве входных данных. Такой выбор текста обусловлен его легендарной длинной, что даст корректное представление о эффективности метода по памяти и по времени, и вкраплением в текст иностранных слов и терминов, что покажет общую корректность метода.

1. Словарный перебор

Наиболее простой метод — сравнение всех слов текста со словарем корректных слов нужного языка. Полученное количество совпавших слов делится на количество слов исследуемого текста. Результирующая величина может рассматриваться как вероятность того, что данный текст принадлежит рассматриваемому языку.

Подобная оценка подходит целям данной работы — критерий по этой 'вероятности' позволит отделить зерна от плевел и выделить наиболее пригодный вариант — ложное срабатывание в общем случае маловероятно из-за специфики процесса. При негативном результате мы видим несвязный набор символов.

```
1   In [1]: import linguistics
2   In [2]: a = open('../sample/warandpeace', 'r').readlines()
3   In [3]: linguistics.istext_dict(a)
4   ('ru', 0,864536523576)
```

Здесь в первой строке подключается лингвистический модуль, во второй строке тестируемый текст записывается в переменную a, и наконец в третьей вызывается метод определения языка по словарю. Метод возвращает предполагаемый язык и величину соответствия $V = \frac{n}{N}$, где n — количество совпавших слов, а N — количество слов в тексте.

Практика демонстрирует жизнеспособность такого метода. Во-первых, составление словаря для известного языка в необходимой стилистике не представляет трудности при условии доступности интернета. Во-вторых, современные компьютерные мощности позволяют сравнительно быстрое выполнение подобного анализа:

```
1   # /usr/bin/time python ./cipher/linguistics.py -d ./sample/warandpeace
2   python  710.10s user 780.55s system 0% cpu 20.002 total
```

Здесь мы вызываем метод $istext\_dict$ через внешний интерфейс и отслеживаем время выполнения команды с помощью стандартной утилиты UNIX time.

Возможно совершествование данного метода путем написания более эффективных структур для хранения словаря, грамотной сериализации и использования оптимизированных алгоримов поиска по сортированному массиву. Но, как будет показано, в этом нет необходимости. Во-первых мы не перешагнем известное ограничение сложности в $O(log(n))$ для алгоритмов поиска. Во-вторых, отсуствует необходимость в строгом соответствии текста языку определенному в словаре. В-третьих, текст шифрограммы сравнительно редко содержит пробелы, либо им нельзя доверять.

Но, эти пробелы возможно восстановить зная статистическое распределение слов языка. То есть необходимо на огромном объеме текста расчитать встречаемость каждого слова и каждого словосочетания. Такие данные сложно расчитать, но они уже были получены в исследовании [8].

В 14 главе книги [10] разобран рекурсивный алгоритм, решающий поставленную задачу. Он был использован в фреймворке.

Для примера, мы получили фразу `HELLOHOWAREYOUTODAY` и пытаемся определить наиболее вероятное разделение. Возможные разделения включают в себя:

```
HELLOHOWA REYOUTODAY
HELLO HOW ARE YOU TODAY
HE LL OH OW AR EY OU TO DA Y
H ELLOH OW AREYOU TOD AY
HELL OHO WARE YOU TODAY
```

и многие другие.

```
1   In [1]: import cipher.linguistics as linguistics
2   In [2]: linguistics.restore_words('HELLOHOWAREYOUTODAY')
3   Out[2]: ['HELLO', 'HOW', 'ARE', 'YOU', 'TODAY']
```

2. Использование n-грамм

Второй метод анализа так же основан на работе со словарем, но обладает рядом преимуществ.

Определение 10. *n-грамма* — последовательность из n элементов. С семантической точки зрения, это может быть последовательность звуков, слогов, слов или букв. На практике чаще встречается n-грамма как ряд слов. Последовательность из двух последовательных элементов часто называют биграммы, последовательность из трех элементов называется триграмма. Не менее четырех и выше элементов обозначаются как n-грамма, n заменяется на количество последовательных элементов.

В области обработки естественного языка, n-граммы используется в основном для предугадывания на основе вероятностных моделей. n-граммная модель рассчитывает вероятность последнего слова n-граммы, если известны все предыдущие. При использовании этого подхода для моделирования языка предполагается, что появление каждого слова зависит только от предыдущих слов.

Определение 11. *Инвертированный индекс* — структура данных, в которой для каждого слова коллекции документов в соответствующем списке перечислены все места в коллекции, в которых оно встретилось. Инвертированный индекс используется для поиска по текстам.

Опишем как решается задача нахождения документов в которых встречаются все слова из поискового запроса. При обработке однословного поискового запроса, ответ уже есть в инвертированном индексе — достаточно взять список соответствующий слову из запроса. При обработке многословного запроса берутся списки, соответствующие каждому из слов запроса и пересекаются.

Пусть у нас есть корпус из трех текстов T_0='it is what it is', T_1='what is it' и T_2='it is a banana', тогда инвертированный индекс будет выглядеть следующим образом:

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

Здесь цифры обозначают номера текстов, в которых встретилось соответствующее слово. Тогда отработка поискового 'what is it ' запроса даст следующий результат $\{0,1\} \cap \{0,1,2\} \cap \{0,1,2\} = \{0,1\}$.

```
1   In [1]: import linguistics
2   In [2]: a = open('../sample/warandpeace', 'r').readlines()
3   In [3]: linguistics.istext\_wgramms(a)
4   0,892340923846
```

Время исполнения теста так-же в разы превышает результат перебора по словарю:

```
1   # /usr/bin/time python ./cipher/linguistics.py -w ./sample/warandpeace
2   python  210.10s user 280.55s system 0% cpu 20.002 total
```

Такой метод, как и словарный перебор не справляется с текстом, в котором отсутствуют пробелы. Для решения этой проблемы необходимо сделать атомарным элементом не слова в тексте, а букву.

В фреймворке реализован метод поочередного теста текста с тетраграммами, триграммами и биграммами. Как только какой-либо из тестов получает ответ с величиной V выше некоторого порога (например в 0,8), этот результат возвращается в качестве ответа. Такой порядок тестов обусловлен характеристиками реализованного лексера. Вызов выглядит так-же как и в прошлом примере:

```
1   In [1]: import linguistics
2   In [2]: a = open('../sample/warandpeace', 'r').readlines()
3   In [3]: linguistics.istext_lgramms(a)
4   0,802304958733
```

Время исполнения теста:

```
1   # /usr/bin/time python ./cipher/linguistics.py -l ./sample/warandpeace
2   python  170.10s user 170.55s system 0% cpu 20.002 total
```

Итак, метод буквенных n-грамм является наиболее эффективным из рассмотренных и будет использоваться в фреймворке.

3. Индекс совпадений

Рассмотрим текст, написанный на некотором языке. Алфавит данного языка будем полагать состоящим из m букв. Рассмотрим достаточно длинную строку $\vec{x}$ из n букв. Если f_i задаёт количество i-той буквы алфавита в строке $\vec{x}$, то можно определить индекс совпадений как вероятность совпадения двух произвольных букв в строке:

$$I(\vec{x}) = \sum_i f_i \frac{f_i - 1}{n(n-1)}$$

откуда при достаточно больших n и определении p_i как $p_i = f_i/n$ получаем приближённую формулу:

$$I(\vec{x}) = \sum_i p_i^2$$

4. Критерий χ^2 (хи-квадрат)

Определение 12. *Критерий χ^2 (хи-квадрат)*, или критерий Пирсона — наиболее часто употребляемый критерий для проверки гипотезы о законе распределения.

Для проверки критерия вводится статистика:

$$\chi^2 = N \sum \frac{\left(P_i^{\text{emp}} - P_i^{\text{H}_0}\right)^2}{P_i^{\text{H}_0}},$$

где $P_i^{\text{H}_0} = F(x_i) - F(x_{i-1})$ — предполагаемая вероятность попадания в i-й интервал, $P_i^{\text{emp}} = \frac{n_i}{N}$ — соответствующее эмпирическое значение, n_i — число элементов выборки из i-го интервала, N — полный объём выборки. Также используется расчет критерия по частоте, тогда:

$$\chi^2 = \sum \frac{\left(V_i - NP_i^{\text{H}_0}\right)^2}{NP_i^{\text{H}_0}},$$

где V_i — частота попадания значений в интервал. Эта величина, в свою очередь, является случайной (в силу случайности χ) и должна подчиняться распределению χ^2.

5. Реализация критериев

Написан подключаемый модуль `linguistics`, выполняющий все операции, связанные с естественными языками. Статистический анализ содержится в базовом классе и будет описан далее. Модуль состоит из 6 функций:

`get_alphabet(lang)` — возвращает алфавит с частотой встречамости букв и некоторыми дополнительными характеристиками языка.

`define_language(text)` — определяет язык текста.

`istext_dictionary(text)` — словарный анализ.

`istext_lgramms(text)` — анализ словарных n-грамм.

`istext_wgramms(text)` — анализ буквенных n-грамм.

3. Классические шифры

В работе [1] Клод Шеннон обобщил накопленный до него опыт разработки шифров. Оказалось, что даже в сложных шифрах в качестве типичных компонентов можно выделить шифры замены, шифры перестановки или их сочетания.

Определение 13. *Шифры перестановки* — такие шифры, преобразования из которых приводят к изменению только порядка следования символов исходного сообщения.

Обычно открытый текст разбивается на отрезки равной длины и каждый отрезок шифруется независимо. Пусть, например, длина отрезков равна n и σ — взаимнооднозначное отображение множества $1, 2, \ldots, n$ в себя. Тогда шифр перестановки действует так: отрезок открытого текста $x_1, \ldots, x_n$ преобразуется в отрезок шифрованного текста $x\sigma(1) \ldots x\sigma(n)$.

Простейший шифр перестановки — шифр Скитала.

Определение 14. *Шифры замены* — такие шифры, преобразования из которых приводят к замене каждого символа открытого сообщения на другие символы - шифробозначения , причем порядок следования шифробозначений совпадает с порядком следования соответствующих им символов открытого сообщения.

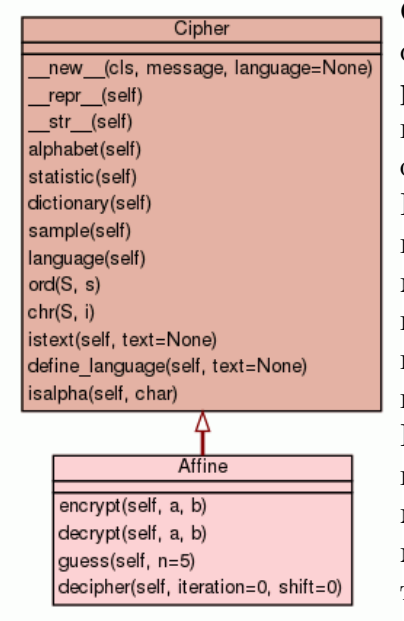
Дадим математическое описание шифра замены. Пусть X и Y — два алфавита открытого и соответственно шифрованного текстов, состоящие из одинакового числа символов. Пусть также $g : X \to Y$ - взаимнооднозначное отображение X в Y. Это значит, что каждой букве x алфавита X соответствует однозначно определенная буква y алфавита Y, которую мы обозначаем символом $g(x)$, причем разным буквам соответствуют разные. Тогда шифр замены действует так: открытый текст $x_1, x_2, \ldots, x_n$ преобразуется в шифрованный текст $g(x1), g(x2), \ldots g(x_n)$.

Простейший шифр замены — шифр Цезаря.

1. Структура фреймворка

Для понимания способа изложения методов в дальнейшем, следует обсудить разработанную структуру фреймворка.

Для этого рассмотрим подробнее программную реализацию на примере афинного шифра, не затрагивая саму реализацию криптоаналитических функций, которая будет обсуждаться в соответствующем параграфе.



Слева представлена UML диаграмма базового класса `Cipher` и класса для исследования афинных шифров `Affine`. Python, выбранный в качестве языка программирования, поддерживает парадигму объектно-ориентированного программирования.

Базовый класс наследован от класса строки, что позволяет использовать все методы для работы со строками внутри объекта (например, объект шифра можно использовать в итераторе). Соответственно и методы базового класса, необходимые для анализа всех изучаемых шифров будут описаны только в одном месте.

Первые три метода, название которых начинается и заканчивается двумя подчеркиваниями `__` («магические методы» в терминологии Python), не вызываются напрямую а исполняются при попадании в какой-либо контекст.

Методы `__init__`, `__repr__` и `__str__` управляют созданием, представлением в виде объекта и представлением в виде строки соответственно.

Оставшиеся методы работают согласно своему названию:

`alphabet` возвращает алфавит языка;

`statistic` возвращает статистику встречаемости символов в сообщении;

`dictionary` возвращает словарь языка;

`sample` возвращает алфавитные символы в сообщении;

`language` возвращает заданный или определенный язык;

`ord(c)` возвращает позицию символа в алфавите языка;

`chr(n)` возвращает символ на позиции `n`, то есть `chr(ord(c)) == c`;

`istext(m)` проверяет является ли m осмысленным текстом в языке;

`define_language(m)` определяет язык сообщения `m`;

`isalpha(c)` проверяет, есть ли символ `c` в алфавите.

Класс `Affine` является наследником класса `Cipher`. Он реализует три метода, заложенных в родительском классе и добавляет свой:

`encrypt` возвращает зашифрованное сообщение;

`decrypt` возвращает расшифрованное сообщение;

`decipher` возвращает результат криптоанализа (эти три метода технически являются перегруженными методами класса `Cipher`);

`guess` сужает множество ключей расшифровки.

Python не реализует кеширующих свойств для объектов, то есть таких свойств, которые рассчитывались бы при первом обращениии и возвращали бы этот результат при дальнейших обращениях. Это довольно неудобно в моем случае, так как во время исследования некоторых шифрограмм необходимы большие объемы данных (например, тетраграммы для выбранного языка). Возможно расчитывать и подгружать их во время инициализации класса с помощью уже обсуждавшегося метода `__init__`, но это будет чувствительно занижать скорость работы в случаях где в таких данных нет необходимости. Такая проблема возникает, например, при анализе n-грамм: триграммы могут дать достаточно хорошую оценку для текста и не будет необходимости загружать статистику по тетраграммам.

Но такая проблема разрешима с помощью другого механизма Python — механизма декораторов.

Определение 15. *Декоратор* — это функция, ожидающая другую функцию в качестве параметра.

Такой подход часто используется в функциональных языках и был перенесен в Python. Консистентность такой парадигмы достигается благодаря тому, что все функции в Python являются объектами. В фреймворке используется класс кешиирующих свойств, его структура представлена на диаграмме 1.

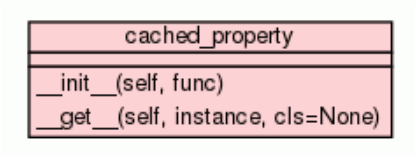


Рис. 1. UML диаграмма класса для кеширования свойств

.

Использование декоратора:

```
1   @cached_property
2   def tetragrams(self):
3       '''
4       S.tetragrams -> list
5       List of tetragrams for selected language.
```

```
6       '''
7       tetragrams = linguistics.get_ngramms(4, self.language)
8       return tetragrams
```

После первого вызова тетраграммы будут содержаться в памяти до выхода из программы.

В итоге, из-за такого структурирования модулей каждый из них можно использовать как подключаемую библиотеку и как самодостаточный скрипт.

Пример работы продемонстрирован в IDLE — это интегрированная среда разработки на языке Python, которая позволяет увидеть результат исполнения сразу после ввода текста. Работа с классом выглядит так:

```
1   # ipython3
2   In [1]: import cipher.classic.affine as a
3   In [2]: secret = a.Affine('EJJEKI EJ NESR')
4   In [3]: secret
5   Out[3]: EJJEK IEJNE SR
6   In [4]: secret.statistic()
7   Out[4]: (('E', 4), ('J', 3), ('R', 1), ('S', 1), ('K', 1), ('I', 1), ('N', 1))
8   In [5]: secret.decipher()
9   Out[5]: ('ATTACK AT DAWN', 3, 4)
```

В первой строке вызывается реализация IDLE, ipython. В строках со второй по седьмую импортируется модуль аффинного шифра, создается объект `secret` с анализируемым шифротекстом `EJJEKI EJ NESR`, проверяется работа вывода и подсчета статистики. В восьмой строке вызван метод криптоанализа, который возвращает открытый текст и параметры функции дешифрования в девятой строке.

Как уже было сказано, каждый модуль является исполняемым файлом, для которого реализован консольный интерфейс по следующей схеме:

```
1   # ./cipher/classic/affine.py 'EJJEKI EJ NESR'
2   Analysis...
3   Defined language is en.
4   ATTACK AT DAWN
5   Decryption parametres is a=3 and b=4.
6
7   # ./cipher/classic/affine.py -e 9x23 sample/en.opentext > sample/en.affine
8
9   # ./cipher/classic/affine.py sample/en.affine
10  Analysis...
```

```
11  Defined language is en.
12  ['TO BE, OR NOT TO BE: THAT IS THE QUESTION:', ...
13  Decryption parametres is a=9 and b=23.
```

В первой строке модуль вызван с помощью ввода параметров из командной строки с уже продемонстрированными параметрами. В строке 7 с параметрами a=9 и b=23 зашифрован отрывок соннета Шекспира из файла `sample/en.opentext`, результат чего записан в файл `sample/en.affine`. В девятой строке как аргумент передается только что полученный шифротекст и модуль успешно справляется с восстановлением открытого текста.

Тепрерь перейем к шифрам и методам их криптоанализа.

2. Шифр Цезаря

Шифр Цезаря — это вид шифра подстановки, в котором каждый символ в открытом тексте заменяется буквой находящейся на некоторое постоянное число позиций левее или правее него в алфавите.

Шифр назван в честь римского императора Гая Юлия Цезаря, использовавшего его для секретной переписки со своими генералами.

1. Описание шифра Цезаря

Если сопоставить каждому символу алфавита его порядковый номер (нумеруя с 0), то шифрование и дешифрование можно выразить формулами модульной арифметики:

$$y = (x + k) \mod n$$
$$x = (y - k + n) \mod n,$$

где x — символ открытого текста, y — символ шифрованного текста, n — мощность алфавита, а k — ключ.

2. Криптоанализ, реализация простого перебора и архитектура базового класса

Шифр Цезаря может быть легко взломан даже в случае, когда взломщик знает только зашифрованный текст. Можно рассмотреть две ситуации:

1) Известно что использовался простой шифр подстановки, но не известно, что это — схема Цезаря;

2) Известно что использовался шифр Цезаря, но не известно значение сдвига.

В первом случае шифр может быть взломан, используя те же самые методы что и для простого шифра подстановки — частотный анализ с перебором по описанным ранее лингвистическим метрикам. Таким образом, взломщик, вероятно, быстро заметит регулярность в решении и поймёт, что используемый шифр — это шифр Цезаря.

Во втором случае, взлом шифра является даже более простым. Существует не так много вариантов значений сдвига (26 для английского языка), все они могут быть проверены методом перебора.

Для обычного текста на естественном языке, скорее всего, будет только один вариант декодирования. Но, если использовать очень короткие сообщения, то возможны случаи, когда возможны несколько вариантов расшифровки с различными сдвигами. Например зашифрованный текст MPQY может быть расшифрован как «aden» так и как «know» (предполагая, что открытый текст написан на английском языке). Точно также «ALIIP» можно расшифровать как «dolls» или как «wheel »; «AFCCP» как «jolly» или как «cheer».

Благодаря малому количеству ключей криптоанализ сводится к применению функции расшифрования ко всем текстам и поиск в результатах текста с максимальной метрикой:

```
1   import cipher.cesar
2   def dechiper(ctext):
3       c = cipher.cesar(ctext)
4       scores = [(fitness.score(c.decrypt(i)), i) for i in range(26)]
5       return max(scores)
```

Относительно шифра Цезаря осталось добавить, что многократное шифрование никак не улучшает стойкость, так как применение шифров со сдвигом a и b эквивалентно применению шифра со сдвигом $a + b$. В математических терминах шифрование с различными ключами образует группу.

3. Аффинный шифр

Аффинный шифр — это частный случай более общего моноалфавитного шифра подстановки. К шифрам подстановки относятся также шифр Цезаря, ROT13 и Атбаш. Поскольку аффинный шифр легко дешифровать, он обладает слабыми криптографическими свойствами.

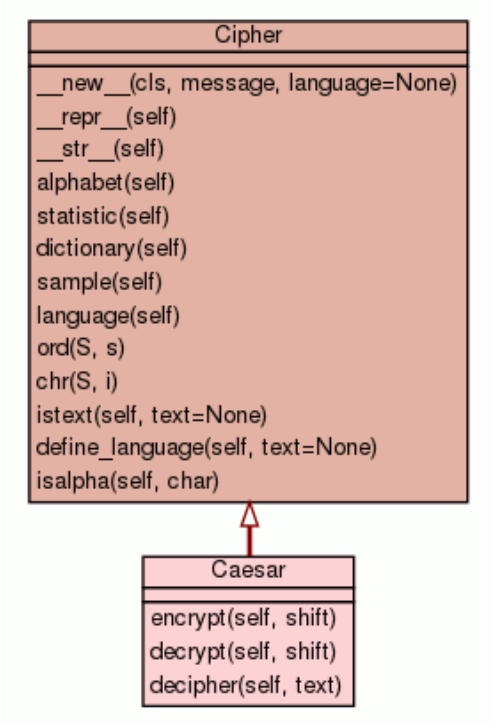


Рис. 2. UML диаграмма класса анализа шифра Цезаря

Рис. 3. Иерархическая диаграмма класса анализа шифра Цезаря

1. Описание аффинного шифра

В аффинном шифре каждой букве алфавита размера m ставится в соответствие число из диапазона $[0, ..., m-1]$. Затем при помощи модульной арифметики для каждого числа, соответствующего букве исходного алфавита, вычисляется новое число, которое заменит старое в шифротексте. Функция шифрования для каждой буквы

$$\mathrm{E}(x) = (ax + b) \mod m,$$

где модуль m — размер алфавита, а пара a и b — ключ шифра. Значение a должно быть выбрано таким, что a и m — взаимно простые числа. Функция расшифрования

$$\mathrm{D}(x) = a^{-1} \times (x - b) \mod m,$$

где a^{-1} — обратное к a число по модулю m. То есть оно удовлетворяет уравнению

$$1 \equiv a \times a^{-1} \mod m.$$

Обратное к a число существует только в том случае, когда a и m — взаимно простые. Значит, при отсутствии ограничений на выбор числа a расшифрование может оказаться невозможным. Покажем, что функция расшифрования является обратной к функции шифрования:

$$\begin{aligned} \mathrm{D}(\mathrm{E}(x)) &= a^{-1} \times (\mathrm{E}(x) - b) \mod m \\ &= a^{-1} \times ((ax + b) - b) \mod m \\ &= a^{-1} \times (ax + b - b) \mod m \\ &= a^{-1} \times ax \mod m \\ &= x \mod m. \end{aligned}$$

Количество возможных ключей для аффинного шифра можно записать с помощью функции Эйлера как $\varphi(m) \times m$.

2. Криптоанализ, минимизация вариантов перебора

Так как аффинный шифр является по сути моноалфавитным шифром замены , то он обладает всеми уязвимостями этого класса шифров. Шифр Цезаря — это аффинный шифр с $a = 1$, что сводит функцию шифрования к простому линейному сдвигу.

В случае шифрования сообщений на русском языке (т. е. с помощью $m = 33$) существует 627 нетривиальных аффинных шифров, не учитывая 33 тривиальных шифра Цезаря. Это число легко посчитать, зная, что существует всего 20 чисел взаимно простых с 33 и меньших 33 (а это и есть возможные значения a). Каждому значению a могут соответствовать 33 разных дополнительных сдвига (значение b); то есть всего существует 2033 или 660 возможных ключей. Аналогично, для сообщений на английском языке (т.е. $m = 26$) всего существует 1226 или 312 возможных ключей. Такое ограниченное количество ключей приводит к тому, что система крайне не криптостойка с точки зрения принципа Керкгоффса.

Основная уязвимость шифра заключается в том, что криптоаналитик может выяснить (путем частотного анализа, полного перебора, угадывания или каким-либо другим способом) соответствие между двумя любыми буквами исходного текста и шифротекста. Тогда ключ может быть найдет путем решения системы уравнений. Кроме того, так мы знаем, что a и m — взаимно простые, это позволяет уменьшить количество проверяемых ключей для полного перебора:

```
1   def decipher(self, iteration = 0, shift = 0):
2       base = range(1, len(self.alphabet) + 1)
3       for a in base:
```

```
4           coprimes = [b for b in base if cipher.routine.gcd(a, b) == 1]
5           for b in coprimes:
6               ot = self.decrypt(a, b)
7               if self.istext(ot) > 0.7:
8                   return (ot, a, b)
```

Преобразование, подобное аффинному шифру, используется в линейном конгруэнтном методе (разновидности генератора псевдослучайных чисел). Этот метод не является криптостойким по той же причине, что и аффинный шифр.

Диаграмма показывает, что афинный шифр успено вписывается в разработанную архитектуру.

4. Шифр Виженера

Шифр Виженера — метод полиалфавитного шифрования буквенного текста с использованием ключевого слова.

Этот метод является простой формой многоалфавитной замены. Шифр Виженера изобретался многократно. Впервые этот метод описал Giovan Battista Bellaso) в 1553 году, однако в XIX веке получил имя Блеза Виженера, французского дипломата. Метод прост для понимания и реализации, он является недоступным для простых методов криптоанализа.

Первое точное документированное описание многоалфавитного шифра было сформулированно Леоном Баттиста Альберти в 1467 году, для переключения между алфавитами использовался металлический шифровальный диск. Система Альберти переключает алфавиты после нескольких зашифрованных слов. Позднее, в 1518 году, Иоганн Трисемус в своей работе «Полиграфия» изобрел tabula recta — центральный компонент шифра Виженера.

Блез Виженер представил свое описание простого, но стойкого шифра перед комиссией Генриха III во Франции в 1586 году, и позднее изобретение шифра было присвоено именно ему. Давид Кан в своей книге «Взломщики кодов» отозвался об этом осуждающе, написав, что история «проигнорировала важный факт и назвала шифр именем Виженера, несмотря на то, что он ничего не сделал для его создания».

Шифр Виженера имел репутацию исключительно стойкого к «ручному» взлому. Известный писатель и математик Чарльз Лютвидж Доджсон (Льюис Кэрролл) назвал шифр Виженера невзламываемым в своей статье «Алфавитный шифр», опубликованной в детском журнале в 1868 году. В 1917 году Scientific American также отозвался о шифре Виженера, как о неподдающемся взлому. Это представление было опровергнуто после того, как Касиски полностью взломал шифр в XIX веке, хотя известны случаи взлома этого шифра некоторыми опытными криптоаналитиками еще в XVI веке.

1. Описание шифра Виженера

В шифре Цезаря каждая буква алфавита сдвигается на несколько строк; например в шифре Цезаря при сдвиге +3, A стало бы D, B стало бы E и так далее. Шифр Виженера состоит из последовательности нескольких шифров Цезаря с различными значениями сдвига. Для зашифровывания может использоваться таблица алфавитов, называемая tabula recta или квадрат (таблица) Виженера. Применительно к латинскому алфавиту таблица Виженера составляется из строк по 26 символов, причём каждая следующая строка сдвигается на несколько позиций. Таким образом, в таблице получается 26 различных шифров Цезаря. На разных этапах кодировки шифр Виженера использует различные алфавиты из этой таблицы. На каждом этапе шифрования используются различные алфавиты, выбираемые в зависимости от символа ключевого слова. Например, предположим, что исходный текст имеет вид:

```
ATTACKATDAWN
```

Человек, посылающий сообщение, записывает ключевое слово («LEMON») циклически до тех пор, пока его длина не будет соответствовать длине исходного текста:

```
LEMONLEMONLE
```

Первый символ исходного текста A зашифрован последовательностью L, которая является первым символом ключа. Первый символ L шифрованного текста находится на пересечении строки L и столбца A в таблице Виженера. Точно так же для второго символа исходного текста используется второй символ ключа; то есть второй символ шифрованного текста X получается на пересечении строки E и столбца T. Остальная часть исходного текста шифруется подобным способом.

```
ATTACKATDAWN
LEMONLEMONLE
LXFOPVEFRNHR
```

Расшифровывание производится следующим образом: находим в таблице Виженера строку, соответствующую первому символу ключевого слова; в данной строке находим первый символ зашифрованного текста. Столбец, в котором находится данный символ, соответствует первому символу исходного текста. Следующие символы зашифрованного текста расшифровываются подобным образом.

Если буквы A-Z соответствуют числам 0-25, то шифрование и расшифрование Виженера можно записать в виде формул:

$$C_i \equiv (P_i + K_i) \mod 26$$
$$P_i \equiv (C_i - K_i + 26) \mod 26$$

2. Криптоанализ, работа с полиалфавитными шифрами

Шифр Виженера «размывает» характеристики частот появления символов в тексте, но некоторые особенности появления символов в тексте остаются. Главный недостаток шифра Виженера состоит в том, что его ключ повторяется. Поэтому простой криптоанализ шифра может быть построен в два этапа:

Поиск длины ключа. Можно анализировать распределение частот в зашифрованном тексте с различным прореживанием. То есть брать текст, включающий каждую 2-ю букву зашифрованного текста, потом каждую 3-ю и т. д. Как только распределение частот букв будет сильно отличаться от равномерного (например, по энтропии), то можно говорить о найденной длине ключа.

Метод Касиски В 1863 году Фридрих Касиски был первым, кто опубликовал успешный алгоритм атаки на шифр Виженера, хотя Чарльз Беббидж разработал этот алгоритм уже в 1854 году. В то время когда Беббидж занимался взломом шифра Виженера, John Hall Brock Thwaites представил новый шифр в «Journal of the Society of the Arts»; когда Беббидж показал, что шифр Thwaites'a является лишь частным случаем шифра Виженера, Thwaites предложил ему его взломать. Беббидж расшифровал текст, который оказался поэмой «The Vision of Sin» Альфреда Теннисона, зашифрованной ключевым словом Emily — именем жены поэта.

Тест Касиски опирается на то, что некоторые слова, такие как «the» могут быть зашифрованы одинаковыми символами, что приводит к повторению групп символов в зашифрованном тексте. Например: сообщение, зашифрованное ключом ABCDEF , не всегда одинаково зашифрует слово «crypto».

Зашифрованный текст в данном случае не будет повторять последовательности символов, которые соответствуют повторным последовательностям исходного текста. В данном шифрованном тексте есть несколько повторяющихся сегментов, которые позволяют криптоаналитику найти длину ключа.

Более длинные сообщения делают тест более точным, так как они включают в себя больше повторяющихся сегментов зашифрованного текста. В данном шифрованном тексте есть несколько повторяющихся сегментов, которые позволяют криптоаналитику найти длину ключа.

Расстояние между повторяющимися DYDUXRMH равно 18, это позволяет сделать вывод, что длина ключа равна одному из значений: 18, 9, 6, 3 или 2. Расстояние между повторяющимися NQD равно 20. Из этого следует, что длина ключа равна 20, 10, 5, 4 или 2. Сравнивая возможные длины ключей, можно сделать вывод, что длина ключа (почти наверняка) равна 2.

Тест Фридмана (иногда называемый каппа-тест) был изобретен Вильямом Фридманом в 1920 году. Фридман использовал индекс совпадения, который измеряет ча-

стоты повторения символов, чтобы взломать шифр. Зная вероятность $\varkappa_p$ того, что два случайно выбранных символа текста совпадают (примерно 0,067 для англ. языка) и вероятность совпадения двух случайно выбранных символов алфавита $\varkappa_r$ (примерно 1 / 26 = 0,0385 для англ. языка), можно оценить длину ключа как:

$$\frac{\varkappa_p - \varkappa_r}{\varkappa_o - \varkappa_r}$$

Из наблюдения за частотой совпадения следует:

$$\varkappa_o = \frac{\sum_{i=1}^{c} n_i(n_i - 1)}{N(N-1)}$$

Где — размер алфавита (26 символов для англ. языка), N — длина текста, и n_1 до n_c — наблюдаемые частоты повторения символов зашифрованного текста. Однако, это только приблизительное значение, точность которого увеличивается при большем размере текста. На практике это было бы необходимо для перебора различных ключей приближаясь к исходному.

Частотный анализ Как только длина ключа становится известной, зашифрованный текст можно записать во множество столбцов, каждый из которых соответствует одному символу ключа. Каждый столбец состоит из исходного текста , который зашифрован шифром Цезаря; ключ к шифру Цезаря является всего-навсего одним символом ключа для шифра Виженера, который используется в этом столбце. Используя методы, подобные методам взлома шифра Цезаря, можно расшифровать зашифрованный текст. Усовершенствование теста Касиски, известное как метод Кирхгофа , заключается в сравнении частоты появления символов в столбцах с частотой появления символов в исходном тексте для нахождения ключевого символа для этого столбца. Когда все символы ключа известны, криптоаналитик может легко расшифровать шифрованный текст, получив исходный текст. Метод Кирхгофа не применим, когда таблица Виженера скремблирована, вместо использования обычной алфавитной последовательности, хотя тест Касиски и тесты совпадения всё ещё могут использоваться для определения длины ключа для этого случая.

5. Энигма

Энигма — портативная шифровальная машина, использовавшаяся для шифрования и дешифрования секретных сообщений. Более точно, Энигма — целое семейство электромеханических роторных машин, применявшихся с 20-х годов XX века.

Энигма использовалась в коммерческих целях, а также в военных и государственных службах во многих странах мира, но наибольшее распространение получила в

нацистской Германии во время Второй мировой войны — именно Энигма вермахта, германская военная модель — чаще всего является предметом дискуссий.

1. Описание шифра Энигма

Как и другие роторные машины, Энигма состояла из комбинации механических и электрических подсистем. Механическая часть включала в себя клавиатуру , набор вращающихся дисков — роторов, — которые были расположены вдоль вала и прилегали к нему, и ступенчатого механизма, двигающего один или несколько роторов при каждом нажатии на клавишу.

Конкретный механизм работы мог быть разным, но общий принцип был таков: при каждом нажатии на клавишу самый правый ротор сдвигается на одну позицию, а при определённых условиях сдвигаются и другие роторы. Движение роторов приводит к различным криптографическим преобразованиям при каждом следующем нажатии на клавишу на клавиатуре.

Механические части двигались, замыкая контакты и образуя меняющийся электрический контур (то есть, фактически, сам процесс шифрования букв реализовывался электрически). При нажатии на клавишу клавиатуры контур замыкался, ток проходил через различные цепи и в результате включал одну из набора лампочек, и отображавшую искомую букву кода. (Например: при шифровке сообщения, начинающегося с ANX. . . , оператор вначале нажимал на клавишу A — загоралась лампочка Z — то есть Z и становилась первой буквой криптограммы. Далее оператор нажимал N и продолжал шифрование таким же образом далее).

Таким образом, постоянное изменение электрической цепи, через которую шёл ток, вследствие вращения роторов позволяло реализовать многоалфавитный шифр подстановки, что давало высокую, для того времени, устойчивость шифра.

Роторы — сердце Энигмы. Каждый ротор представлял собой диск примерно 10 см в диаметре, сделанный из эбонита или бакелита, с пружинными штыревыми контактами на одной стороне ротора, расположенными по окружности . На другой стороне находилось соответствующее количество плоских электрических контактов. Штыревые и плоские контакты соответствовали буквам в алфавите (обычно это были 26 букв от A до Z). При соприкосновении контакты соседних роторов замыкали электрическую цепь. Внутри ротора каждый штыревой контакт был соединён с одним из плоских. Порядок соединения мог быть различным.

Сам по себе ротор производил очень простой тип шифрования: элементарный шифр замены. Например, контакт, отвечающий за букву E, мог быть соединён с контактом буквы T на другой стороне ротора. Но при использовании нескольких роторов в связке (обычно трёх или четырёх) за счёт их постоянного движения получается более надёжный шифр.

Преобразование Энигмы для каждой буквы может быть определено математически

как результат перестановок. Рассмотрим трёхроторную армейскую модель . Положим, что P обозначает коммутационную панель, U обозначает отражатель , а L, M, R обозначают действия левых, средних и правых роторов соответственно . Тогда шифрование E может быть выражено как:

$$E = PRMLUL^{-1}M^{-1}R^{-1}P^{-1}$$

После каждого нажатия клавиш ротор движется, изменяя трансформацию . Например, если правый ротор R проворачивается на i позиций, происходит трансформация $\rho^i R\rho^{-i}$, где ρ — циклическая перестановка , проходящая от A к B, от B к C, и так далее. Таким же образом, средний и левый ротор могут быть обозначены как j и k вращений M и L. Функция шифрования в этом случае может быть отображена следующим образом:

$$E = P(\rho^i R\rho^{-i})(\rho^j M\rho^{-j})(\rho^k L\rho^{-k})U(\rho^k L^{-1}\rho^{-k})(\rho^j M^{-1}\rho^{-j})(\rho^i R^{-1}\rho^{-i})P^{-1}$$

2. Криптоанализ, подбор ключа сложной структуры

Попытки «взломать» Энигму не предавались гласности до конца 1970-х. После этого интерес к Энигме значительно возрос, и множество шифровальных машин представлено к публичному обозрению в музеях США и Европы.

Как было указано, Энигма это цлое семейство машин а не один алгоритм. Мы изучим одну из последних модификаций. Она появилась летом 1939 года когда немцы усложнили процедуру шифрования, добавив в набор два ротора к имеющимся трем, увеличив количество возможных комбинаций установок роторов с $3! = 6$ до $A_5^3 = \frac{5!}{(5-3)!} = 60$. После изучения польских материалов Алан Тьюринг приш‘л к выводу, что использовать подход с полным перебором сообщений уже не получится. Во-первых, это потребует создания более 30 экземпляров «Бомбы», что во много раз превышало годовой бюджет «Station X». Во-вторых, Германия должна была в скором времени догадаться и исправить конструктивный недостаток, на котором основывался польский метод. Поэтому он разработал собственный метод, основанный на переборе последовательностей символов исходного текста. Однако, появившаяся в «Энигме» коммутационная доска, простейший с точки зрения схемотехники элемент, добавила проблем исследователям. С ней «боролся» Гордон Велчман, который изобр ‘л метод «диагональной доски». На основе своих методов, в августе 1940 года, с помощью компании British Tabulating Machines и ее конструктора Гарольда Кина была построена первая британская криптоаналитическая машина, которая была названа Bombe, в знак уважения к польским криптографам. Впоследствии за время войны было выпущено 210 устройств, позволивших расшифровывать до двух-трех тысяч сообщений в день.

С современной точки зрения шифр «Энигмы» был не очень надежным, но только сочетание этого фактора с наличием множества перехваченных сообщений, кодовых книг, донесений разведки, результатов усилий военных и даже террористических атак позволило «вскрыть» шифр.

4. Бинарные шифры

Данный раздел выделен в качестве переходной ступени от шифров, оперирующих алфавитом к шифрам бинарного уровня. Такая смена базиса мало влияет на логику анализа, но но изменяет программные алгоритмы.

1. XOR

XOR — это побитовое сложение по модулю (с инвертированием при переполнении), например, $1 + 1 = 0$ т.к. 1 - максимальное значение. Все варианты:

$$0 \oplus 0 = 0$$

$$0 \oplus 1 = 1$$

$$1 \oplus 1 = 0$$

1. Описание шифра XOR

То есть, операция $z = x \oplus y$ по сути поразрядная (побитовая — результат не зависит от соседних битов). Если только один из соответствующих битов равен 1, то результат 1. А если оба 0 или оба 1, то результат 0. Если внимательно посмотреть на результат применения XOR к двум двоичным числам, то можно заметить, что мы можем восстановить одно из слагаемых при помощи второго: $x = z \oplus y$ или $y = z \oplus x$.

2. Криптоанализ, интерпретация бинарных данных

Отсюда можно сделать следующие выводы: зная число y и применяя XOR к x, мы получим z. Затем, мы, опять же используя y, получим из z обратно число x. Таким образом мы можем преобразовать последовательность чисел $(x)_i$ в последовательность $(z)_i$. Теперь мы можем назвать число y кодирующим (или шифрующим) ключом. Если человек не знает ключа, то он не сможет восстановить исходную последовательность чисел $(x)_i$.

Поскольку каждая буква будет представлена в шифротексте одним и тем же кодом z, то пользуясь частотным словарем взломщик сможет вычислить шифрующий ключ y, если у него будет в распряженииии достаточно длинный шифротекст.

В случае длинного ключа применяются уже разобранные методы анализа из шифра Виженера. Структура класса для анализа шифра Xor представлена на диаграмме 5.

2. Blowfish

Blowfish — это алгоритм, разработанный Брюсом Шнайером специально для реализации на больших микропроцессорах. Алгоритм Blowfish не запатентован.

Алгоритм Blowfish оптимизирован для применения в системах, не практикующих частой смены ключей, например, в линиях связи и программах автоматического шифрования файлов. При реализации на 32-битовых микропроцессорах с большим размером кэша данных , например, процессорах Pentium и PowerPC, алгоритм Blowfish заметно быстрее DES.

1. Описание шифра Blowfish

Blowfish представляет собой 64-битовый блочный алгоритм шифрования с ключом переменной длины. Алгоритм состоит из двух частей: расширения ключа и шифрования данных. Расширение ключа преобразует ключ длиной до 448 битов в несколько массивов подключей общим размером 4168 байт.

Шифрование данных заключается в последовательном исполнении простой функции 16 раз. На каждом раунде выполняются зависимая от ключа перестановка и зависимая от ключа и данных подстановка. Используются только операции сложения и XOR над 32-битовыми словами. Единственные дополнительные операции каждого раунда - четыре взятия данных из индексированного массива. То есть, алгоритм Blowfish представляет собой сеть Фейстеля, состоящей из 16 раундов. На вход подается 64-битовый элемент данных .

В алгоритме Blowfish используется множество подключей. Эти подключи должны быть вычислены до начала зашифрования или расшифрования данных.

Подключи рассчитываются с помощью самого алгоритма Blowfish. Вот какова точная последовательность действий:

1) Сначала P-массив, а затем четыре S-блока по порядку инициализируются фиксированной строкой.

2) Выполняется операция XOR над P_1 с первыми 32 битами ключа, XOR над P_2 со вторыми 32 битами ключа, и т.д. для всех битов ключа (вплоть до P_{18}). Операция XOR выполняется циклически над битами ключа до тех пор, пока весь P-массив не будет инициализирован.

3) Используя подключи, полученные на этапах 1 и 2, алгоритм Blowfish шифрует строку из одних нулей.

4) P_1 и P_2 заменяются результатом этапа 3.

5) Результат этапа 3 шифруется с помощью алгоритма Blowfish и модифицированных подключей.

6) P_3 и P_4 заменяются результатом этапа 5.

7) Далее по ходу процесса все элементы P-массива, а затем все четыре S-блока по порядку заменяются выходом постоянно меняющегося алгоритма Blowfish.

Всего для генерации всех необходимых подключей требуется 521 итерация. Приложения могут сохранять подключи - нет необходимости выполнять процесс их получения многократно. В реализациях Blowfish, в которых требуется очень высокая скорость, цикл должен быть развернут, а все ключи храниться в кэше.

2. Криптоанализ, дифференциальный криптоанализ

В курсовой работе описан криптоанализ Blowfish с уменьшенным количеством раундов t (в оригинальном алгоритме 16 раундов) и без итоговой перестановки регистров в сети Фейстеля. Практически подтверждены границы стойкости данные в работе [4], а именно возможности с помощью дифференциального криптоанализа раскрыть P-массив с помощью 28^{t+1} выбранных открытых текстов.

Слабым является ключ, для которого два элемента данного S-блока идентичны. Для некоторых найденных слабых ключей, которые генерируют плохие S-блоки (вероятность выбора такого ключа составляет 1 к 214), показано что это же вскрытие раскрывает P-массив с помощью всего 24^{t+1}. При неизвестных S-блоках это вскрытие может обнаружить использование слабого ключа, но не может определит сам ключ (ни S-блоки, ни P-массив).

Случаи успешного криптоанализа Blowfish не известны.

5. Определение алгоритма шифрования по шифротексту

Исследуемый сценарий — известна криптограмма и неизвестен ни алгоритм шифрования, ни алгоритм шифрования, необходимо получить исходный открытый текст. В этой части описан примерный алгоритм идентификации шифра.

Для него понадобится следующая классификация шифров:

1) **Перестановочные шифры** — шифр с фиксированным периодом n подразумевает разбиение исходного текста на блоки по n символов и использование для каждого такого блока некоторой перестановки E. Ключом такого шифра является используемая при шифровании перестановочная матрица P или вектор t, указывающий правило перестановки. Таким образом, общее число возможных ключей определяется длиной блока n и равно $n!$. При дешифрации используется матрица обратной перестановки D, являющаяся обратной к матрице P по умножению, то есть $D \times P = I$, где I — единичная матрица;

2) **Моноалфавитные шифры** — шифры, в которых каждой букве кодируемого текста ставится в соответствие однозначно какая-то шифрованная буква;

3) **Полиалфавитные шифры** — шифры, в которых каждой букве Суть полиалфавитного шифра заключается в циклическом применении нескольких моноалфавитных шифров к определенному числу букв шифруемого текста. Например, пусть у нас имеется некоторое сообщение $x_1, x_2, x_3, \ldots, x_n, \ldots, x_{2n}, \ldots$ которое надо зашифровать. При использовании полиалфавитного шифра имеется несколько моноалфавитных шифров (например, n штук). Самым важным эффектом, достигаемым при использовании полиалфавитного шифра, является маскировка частот появления тех или иных букв в тексте, на основании которой обычно очень легко вскрываются моноалфавитные шифры;

4) **Смешанные шифры** — шифры, в которых сочетаются свойства перечисленных выше шифров.

Каждая из этих категорий оставляет после шифрования статистический отпечаток, который возможно заметить с помощью описываемых далее методов. Для их работы необходимо 1000 или более знаков шифротекста.

Возможно выяснить границу между шифрами перестановки и всеми остальными, сравнив встечаемость символов в шифротексте и в натуральных языках. Если шифр перестановочный, его частотная характеристика совпадет с характеристикой какого-либо из языков.

Следующий шаг — определение моноалфавитного шифра. Для этого можно посчитать индекс совпадений. Если он будет в районе 0.06, то можно заключить, что это шифр моноалфавитной подстановки. Если он меньше, то это либо полиалфавитная подстановка, либо смешанный шифр.

Дальнейший анализ построен на основе следующих величин:

1) **IC** — индекс совпадения умноженный на 1000;
2) **MIC** — максимальный индекс совпадения для периодов с 1 по 15 умноженный на 1000;
3) **DIC** — индекс совпадения биграмм умноженный на 10000;
4) **EDI** — индекс совпадения биграмм для четных по порядку пар умноженный на 10000;
5) **LR** — квадратный корень от процента от 3 повторений символа умноженный на 1000;
6) **ROD** — отношение повторений на нечетном интервале ко всем отношениям в тексте.
7) **DIC** — индекс совпадения биграмм умноженный на 10000;
8) **SDD** — средняя величина дисперсии букв в биграммах.

С их помощью можно составить таблицы, характеризующие шифры сложнее моноалфавитных подстановок. Такие таблицы были постоены и опубликованы Американским Сообществом Криптоаналитиков. Выписки из таблиц приведены в рисунках 6 и 7.

Величины для изучаемых шифров использованы в фреймворке, при достаточном объеме шифротекста они позоляют определить использованный шифр.

6. Заключение

Во время написания курсовой работы мною были изучены классические и блочные симметричные шифры на примере алгоритмов шифр Цезаря, Афинный шифр, шифр Виженера, Xor, Enigma и Blowfish.

Передо мной стояла задача программной реализации программной реализации фреймворка для криптоанализа, который имел бы удобный интерфейс пользователя и разработчика.

Эта задача была успешно выполнена. Фреймворк реализован, шифрует и расшифровывает файлы в соответствии с описаниями изученных алгоритмов. В процессе написания мною были изучены некоторые тонкости программирования, связанные с шифрованием, а также с написанием программных интерфейсов.

Исходный код Python разработанного фреймворка и исходный код TEX курсовой работы доступны в Интернете в виде git репозитория по ссылке https://github.com/ch3sh1r/cryptanalysis.

Список литературы

[1] Шеннон К., «Работы по теории информации и кибернетике» (перевод Писаренко), 1963

[2] Фомичев В.М., «Дискретная математика и криптология», 2003

[3] Ященко В.В., «Введение в криптографию», 1988

[4] Vaudenay S., «On the weak keys of Blowfish», 1996

[5] Khan D., The Codebreakers — The Story of Secret Writing Revised edition (ISBN 978-0-684-83130-9) (1996)

[6] Gillogly J., «Ciphertext only Cryptanalysis of the Enigma», 1995

[7] Erskine D., «Letter originally appeared in Cryptologia», 1996, http://web.archive.org/web/20060720035430/http://members.fortunecity.com/jpeschel/erskin.htm

[8] Franz A. and Brants T., «All Our N-gram are Belong to You», 2006, http://googleresearch.blogspot.com.au/2006/08/all-our-n-gram-are-belong-to-you.html

[9] Williams H., «Applying Statistical Language Recognition Techniques in the Ciphertext only Cryptanalysis of Enigma», 2005

[10] Hammerbacher K. and Segaran M., «Beautiful Data», 2009

[11] Стандарт представления наименований языков ISO 639-1:2002, http://www.infoterm.info/standardization/iso_639_1_2002.php

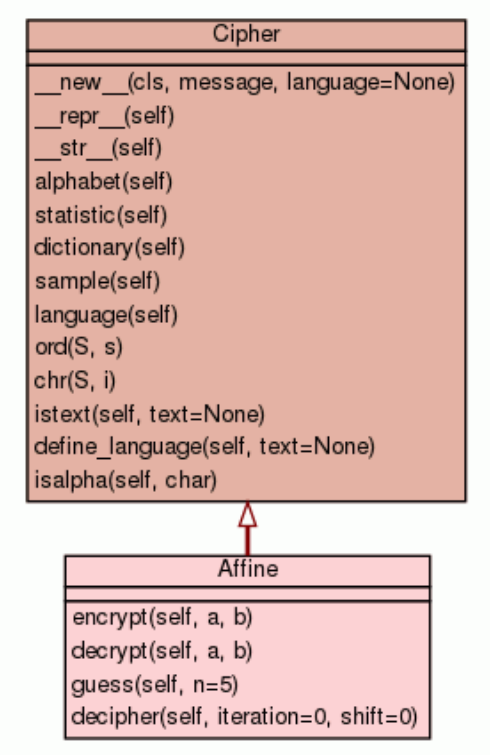


Рис. 4. UML диаграмма класса анализа афинного шифра

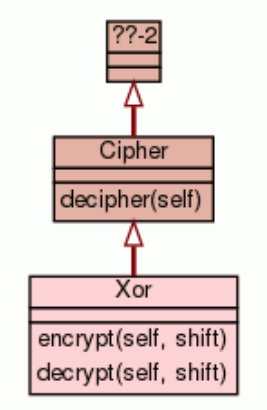


Рис. 5. UML диаграмма класса анализа шифра XOR

Type	IC	MIC	MKA	DIC	EDI	LR	ROD	LDI	SDD
Plaintext	63 ±5	73 ±11	95 ±19	72 ±18	73 ±24	22 ±5	50 ±6	756 ±13	303 ±23
Rand Digit	100 ±2	108 ±8	132 ±16	100 ±8	98 ±15	21 ±3	50 ±3	0 ±0	0 ±0
Rand Text	38 ±1	44 ±5	60 ±12	14 ±2	14 ±5	5 ±3	50 ±10	428 ±23	109 ±14
6x6 Bifid7	35 ±4	47 ±9	62 ±16	14 ±5	14 ±8	4 ±3	49 ±12	298 ±53	71 ±16
6x6Playfair	42 ±4	51 ±9	67 ±15	32 ±9	72 ±24	11 ±5	25 ±9	243 ±57	63 ±19
Amsco	63 ±5	72 ±10	94 ±19	44 ±10	43 ±13	11 ±4	50 ±8	688 ±15	188 ±17
Bazeries	64 ±4	74 ±13	94 ±20	60 ±15	61 ±20	17 ±5	49 ±5	477 ±44	112 ±21
Beaufort7	42 ±3	67 ±9	78 ±17	23 ±5	23 ±9	9 ±4	50 ±10	443 ±32	113 ±15
Bifid6	47 ±4	58 ±10	75 ±15	24 ±6	24 ±8	7 ±4	48 ±10	510 ±36	119 ±16
Bifid7	47 ±4	58 ±9	77 ±17	24 ±6	23 ±8	7 ±4	49 ±9	517 ±37	118 ±17
Cadenus	63 ±5	74 ±11	95 ±17	40 ±9	41 ±13	10 ±4	49 ±9	657 ±17	134 ±18
Cmbifid7	47 ±4	57 ±9	75 ±15	23 ±5	23 ±9	6 ±4	50 ±10	493 ±31	114 ±16
Columnar	63 ±5	73 ±11	96 ±18	41 ±8	41 ±12	11 ±4	50 ±7	653 ±16	128 ±15
Digrafid5	41 ±3	53 ±7	67 ±13	17 ±4	20 ±7	5 ±3	43 ±11	469 ±33	112 ±15
Dbl Ck Bd	90 ±13	133 ±18	149 ±23	110 ±30	207 ±58	25 ±5	13 ±7	609 ±44	133 ±19
4 Square	48 ±3	58 ±9	76 ±15	36 ±8	72 ±24	11 ±4	28 ±8	507 ±33	114 ±16
FracMorse	47 ±2	53 ±8	70 ±15	42 ±9	43 ±13	16 ±3	50 ±7	444 ±32	107 ±17
Grandpre	128 ±3	136 ±7	158 ±15	179 ±15	227 ±39	33 ±3	43 ±3	0 ±0	0 ±0
Grille	63 ±5	74 ±12	91 ±16	42 ±9	43 ±14	10 ±4	49 ±7	679 ±16	173 ±17
Gromark	39 ±1	46 ±7	63 ±13	15 ±3	15 ±6	4 ±3	50 ±12	431 ±26	109 ±15
Gronsfeld	40 ±2	66 ±8	76 ±19	21 ±5	25 ±11	9 ±4	42 ±14	444 ±27	111 ±15
Homoph	101 ±1	108 ±6	127 ±13	116 ±7	160 ±15	24 ±2	42 ±2	0 ±0	0 ±0
Mon Din	124 ±7	134 ±11	169 ±19	249 ±36	252 ±43	45 ±5	49 ±2	0 ±0	0 ±0
Morbit	122 ±4	129 ±7	156 ±16	193 ±15	194 ±25	38 ±2	49 ±2	0 ±0	0 ±0
Myszk	63 ±5	72 ±10	95 ±18	41 ±8	41 ±13	11 ±4	49 ±7	657 ±18	135 ±18
Nicodemus	42 ±3	50 ±7	73 ±14	18 ±4	18 ±7	5 ±3	50 ±10	442 ±35	112 ±15
Nihil Sub	144 ±11	201 ±23	195 ±30	218 ±33	266 ±42	38 ±4	40 ±6	0 ±0	0 ±0

Рис. 6. Таблица ACA характеристик полиалфавитных шифров

Type	IC	MIC	MKA	DIC	EDI	LR	ROD	LDI	SDD
Nihil Transp	63 ±5	73 ±12	97 ±18	41 ±9	40 ±13	10 ±4	50 ±9	654 ±17	129 ±17
Patristo-	63 ±5	73 ±11	95 ±19	72 ±18	73 ±24	22 ±5	50 ±6	414 ±57	106 ±23
Phillips	49 ±3	58 ±8	74 ±16	32 ±7	32 ±10	11 ±4	49 ±9	424 ±37	106 ±17
Per Grmk	38 ±1	45 ±7	63 ±14	14 ±3	15 ±6	4 ±3	48 ±11	428 ±26	108 ±16
Playfair	50 ±4	60 ±9	79 ±18	38 ±9	72 ±24	12 ±4	32 ±8	491 ±42	118 ±19
Pollux	100 ±0	103 ±2	121 ±9	105 ±2	105 ±4	23 ±1	50 ±1	0 ±0	0 ±0
Porta	41 ±2	66 ±9	74 ±16	22 ±6	25 ±11	9 ±4	42 ±13	432 ±35	111 ±16
Portax	42 ±2	51 ±7	66 ±14	18 ±3	19 ±8	6 ±3	48 ±12	442 ±24	113 ±13
Prog Key	38 ±1	45 ±6	63 ±13	14 ±3	13 ±5	4 ±3	49 ±14	428 ±24	109 ±15
Pg Key Be	38 ±1	45 ±6	63 ±14	14 ±3	14 ±6	4 ±3	49 ±12	429 ±26	109 ±14
Quagmire2	41 ±2	65 ±8	75 ±15	21 ±5	25 ±10	8 ±4	42 ±14	431 ±32	109 ±16
Quagmire3	42 ±2	66 ±9	76 ±18	22 ±5	24 ±10	8 ±4	43 ±12	444 ±36	110 ±17
Quagmire4	41 ±2	65 ±8	75 ±18	21 ±5	23 ±10	8 ±4	44 ±13	440 ±33	111 ±17
Ragbaby	41 ±1	49 ±8	71 ±14	18 ±4	18 ±6	6 ±4	49 ±11	473 ±23	112 ±15
Redefence	63 ±5	72 ±10	94 ±16	41 ±10	43 ±16	10 ±4	49 ±7	653 ±18	128 ±15
Run Key	39 ±4	56 ±18	74 ±22	16 ±8	16 ±15	4 ±5	49 ±19	445 ±35	107 ±23
Ser Pfair	48 ±3	56 ±9	75 ±19	25 ±6	25 ±9	7 ±4	49 ±8	484 ±38	115 ±17
Swagman5	62 ±5	72 ±11	90 ±17	39 ±7	39 ±12	10 ±4	50 ±6	650 ±18	135 ±16
Tridigital	122 ±8	134 ±15	161 ±22	195 ±29	197 ±37	38 ±4	49 ±3	0 ±0	0 ±0
Trifid7	42 ±3	53 ±8	68 ±14	18 ±5	18 ±8	6 ±3	51 ±12	462 ±37	112 ±15
Trisquare	43 ±2	51 ±5	64 ±11	21 ±3	21 ±6	7 ±2	49 ±6	503 ±23	119 ±14
Trisqu HR	43 ±1	52 ±5	65 ±11	21 ±3	21 ±5	7 ±3	50 ±7	512 ±23	120 ±13
2 Square	49 ±3	60 ±8	77 ±16	36 ±9	72 ±24	11 ±4	28 ±8	542 ±33	121 ±18
2 Sq spi	47 ±3	59 ±8	76 ±15	34 ±7	72 ±24	11 ±4	25 ±9	501 ±36	119 ±17
Vigautokey	39 ±1	45 ±6	62 ±12	15 ±3	14 ±5	4 ±3	50 ±12	434 ±23	111 ±16
Vigenere	42 ±2	65 ±8	74 ±15	22 ±6	26 ±11	8 ±4	42 ±13	438 ±33	106 ±16
Vigenere7	42 ±3	67 ±9	78 ±17	23 ±5	23 ±8	9 ±4	50 ±10	437 ±34	108 ±17
V Slidefair7	40 ±2	63 ±9	72 ±16	18 ±4	25 ±9	6 ±3	40 ±11	436 ±34	112 ±15

Рис. 7. Таблица ACA характеристик полиалфавитных шифров