МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра компьютерных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

АРЫКОВ НИКИТА ЕВГЕНЬЕВИЧ

Разработка white-box криптографической системы

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель

Кренделев С.Ф.

канд. физ.-мат. наук, доцент, НГУ

……..

(подпись, дата)

Автор

Арыков Н.Е.

ФИТ, 9205

..……..

(подпись, дата)

Новосибирск, 2013 г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра компьютерных систем

УТВЕРЖДАЮ

Зав. кафедрой Пищик Б.Н.

……
(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту *(ке)* Арыкову Никите Евгеньевичу

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема: Разработка white-box криптографической системы

Исходные данные (или цель работы): разработать и реализовать криптографическую систему с открытым ключом, зависящую от большого количества секретных параметров.

Структурные части работы:
1) Анализ наиболее распространённых существующих решений
2) Описание алгоритма шифрования
3) Анализ безопасности криптографической системы
4) Описание реализации разработанного алгоритма

Содержание

ВВЕДЕНИЕ

Классические способы защиты хранилищ конфиденциальных данных становятся с каждым годом более уязвимыми из-за увеличения мощностей электронных вычислительных устройств. Наиболее распространённые методы защиты данных, такие как, хеш-функции и криптографические протоколы с открытым ключом, например RSA и схема Эль-Гамаля, имею ряд недостатков. За 2012 год произошла утечка баз данных паролей огромных хранилищ данных, таких как LinkedIn, Yahoo и других. В базах данных находились хеш-значения паролей пользователей, для большей части хеш-значений удалось найти прообразы или коллизии.

Хеш-функции применяются во многих критичных для информационной безопасности местах от проверки целостности сообщений до генерации псевдослучайных чисел. Процент и скорость нахождения прообразов и коллизий хеш-значений паролей велик, что показывает необходимость в создание криптографической системы обладающей большей криптостойкостью и устойчивостью к атакам методами полного перебора, по сравнению с существующими методами.

Главной причиной успеха атак методами полного перебора является малое количество секретных параметров хеш-функций и криптографических протоколов с открытым ключом. Основная идея усиления криптостойкости алгоритма шифрования - увеличение количества секретных параметров, вследствие чего увеличивает сложность полного перебора и соответственно его скорость.

1 Теоретическая часть

1.1 Цель работы

Целью данной работы является разработка криптографического протокола с открытым ключом, зависящего от большого количества секретных параметров, устойчивого к атакам методами полного перебора и к атакам по подобранному и адаптивно подобранному шифротексту, для шифрования конфиденциальных данных. А также реализация приложений для шифрования данных, на примере паролей пользователей:

- White-box шифратор
- Плагин для системы управления баз данных MySQL и веб-интерфейса phpMyAdmin

1.2 Анализ существующих решений

1.2.1 Хеш-функции

Главная задача хеш-функций обеспечение целостности и конфиденциальности данных. Примером обеспечения конфиденциальности является хранение паролей пользователей. Например, в базах данных, файле /etc/shadow на UNIX-системах, а так же маршрутизаторах Cisco. В случае утечки базы данных хешей паролей, злоумышленник не должен иметь возможности восстановить оригинальные пароли. Для обеспечения целостности информации используются расширения хеш-функций при помощи секретного ключа, такие как HMAC, MAC.

Хеш-функция - это функция H(M), которая применяется к сообщению произвольной длины M и возвращает значение фиксированной длины h, m = H(M), где m имеет длину h. Хеш-функция является примером односторонней функции.

Криптографической хеш-функцией называется хеш-функция удовлетворяющая следующим условиям:

- Необратимость или стойкость к восстановлению прообраза: для заданного значения m хеш-функции должно быть невозможно за приемлемое время найти блок данных X, для которого H(X) = m.

- Стойкость к коллизиям первого рода: для заданного сообщения M должно быть невозможно за приемлемое время подобрать другое сообщение N, для которого H(N) = H(M).

- Стойкость к коллизиям второго рода: должно быть невозможно за приемлемое время подобрать пару сообщений (M, M'), имеющих одинаковое значение хеш-функции.

- Удовлетворять условию лавинного эффекта. Лавинный эффект – свойство для шифров, которое означает, что изменение малого количества битов во входном тексте ведет к "лавинному", то есть большому изменению значений выходных битов шифротекста.

1.2.1.1 Недостатки хеш-функций

Главной проблемой хеш-функций является использование двух секретных параметра – тип функции хеширования и соль. Список популярных хеш-функций не является достаточно большим: MD5, SHA-1, SHA-512, RIPEMD-256, BCrypt и другие. При этом количество криптографических хеш-функций на порядок меньше.

Определив тип используемой хеш-функции, атакующий может использовать радужные таблицы или метод грубой силы с применением эвристик, таких как, парадокс дней рождений, частотный анализ, метод ветвей и границ, перебор по словарю и другие для извлечения конфиденциальной информации и нахождения прообразов или коллизий хеш-значений.

Для увеличения количества параметров используется случайная соль. Соль – это строка случайных данных, которая объединяется с исходными данными и передается на вход хеш-функции. Для каждого пароля используется своя уникальная соль, которая хранится вместе с паролем в открытом виде, в результате получаются новые конструкции $H_1(pass, salt)$, $H_1(H_2(pass, salt))$ и тому подобные. Здесь H_1, H_2 – хеш-функции, $pass$ – пароль, $salt$ – соль.

Использование соли, оставляет уязвимость коллизий, которую невозможно избежать в случае отображения одного множества большей мощности в другое множество намного меньшей мощности, а так же не защищает одно конкретное хеш-значение от полного перебора.

1.2.1.2 Анализ массовых утечек баз данных паролей

За 2012 год было несколько инцидентов утечек баз данных паролей пользователей больших порталов. В интернет попал большой объем хеш-значений паролей, используя эвристические методы полного перебора, удалось восстановить большую часть прообразов хеш-значений, либо найти коллизию.

Таблица 1: Статистика по утечке баз данных паролей

Сервер/Владелец	Год	Средняя длина пароля	Количество паролей	Процент нахождения, %	Тип хеш-функции
Yahoo	2012	8	453000	-	Без шифрования
LinkedIn	2012	-	6500000	58	SHA-1 (без соли)
Last.fm	2012	-	17300000	95	MD5 (без соли)
Formspring	2012	-	420000	-	SHA-256 со случайной солью

Из таблицы 1 видно, что процент расшифровки достаточно велик. В первую очередь это связано с малой длиной пользовательских паролей. Во вторых, скорость расчета используемых хеш-функций, таких как MD5 и SHA-1 не устойчива к полному перебору на графический вычислительных устройствах (GPU). В открытом доступе находятся специальные программы, такие как hashcat, John The Ripper и другие, которые специально оптимизированы для конкретного типа хеш-функции и использующие эвристические правила построения последующих значений для перебора. Высокая скорость перебора так же связана с малым количество используемых параметров хеш-функций.

В таблице 2 приведена статистика по скорости вычисления значений хеш-функций [1]. Замеры производились на кластере, составленном из 25 графических вычислителей AMD Radeon.

Таблица 2: Скорость вычисления хеш-функций

Хеш-функция	Скорость вычисления (Гигабайт/сек)
LM	20

SHA-1	63
MD5	180
NTLM	348

После данных инцидентов для противостояния методу грубой силы начался переход к увеличению времени расчета хеш-функции. Одним из способов является применение функции хеширования несколько раз, при большом применение хеш-функции к самой себе $H(H(....H(m)...))$ происходит сужение области значений хеш-функции. Другим вариантом замедления вычисления является использование адаптивных хеш-функций типа BCrypt, SCrypt [2].

1.2.2 Определения

1.2.2.1 Односторонняя функция

Функция $f:\{0,1\}^* \to \{0,1\}^*$ называется односторонней, если:

1) Существует полиномиальный алгоритм, который для всякого входного значения x вычисляет $f(x)$.

2) Не существует полиномиального алгоритма, который верно вычисляет обратную функцию F^{-1} к $f(x)$.

Существование односторонних функций пока не доказано, они строятся из предположения, что класс сложности задач P не равен классу сложности NP. Самым известным примером является умножение и обратная к нему операция - факторизация. Функция $f(x,y)$ принимает на вход два простых числа p и q, возвращает произведение входных параметров за время $O(n^2)$, где n - общая длина (количество двоичных чисел) входных данных. Построение обратной функции требует нахождения множителей заданного целого числа N, и является NP полной задачей. Криптографические системы с открытым ключом строятся на односторонних функциях, таких как факторизация, задача о рюкзаке, задача дискретного логарифмирования.

1.2.2.2 Криптографический протокол с открытым ключом

Пусть K – пространство ключей, e – ключ шифрования, d – ключ расшифровки. E_e функция шифрования для произвольного ключа $e \in K$ такая что $E_e(m) = c$, здесь

$m \in M$ - шифруемое сообщение из пространства сообщений M, $c \in C$ - зашифрованное сообщение из пространства шифротекстов C. D_d - функция расшифровки, с помощью которой можно найти исходное сообщение m, зная шифротекст c по формуле $D_d(c) = m$.

$\{E_e, e \in K\}$ - пара для шифрования, которая объявляется публичным ключом. $\{D_d, d \in K\}$ - пара для расшифровки, которая объявляется приватным ключом. Каждая пара (E, D) имеет свойство: зная E_e невозможно решить уравнение $E_e(m) = c$, то есть для заданного произвольного зашифрованного сообщения $c \in C$ невозможно найти исходное сообщение $m \in M$. Это означает, что по заданному ключу e невозможно определить соответствующий ключ расшифровки d. E_e - является односторонней функцией.

Ниже показана схема передачи информации лицом А лицу В. А – Алиса, В – Боб.

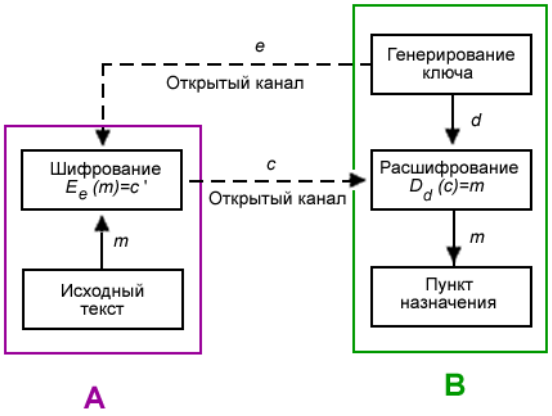

Рисунок 1: Схема работы криптосистемы с открытым ключом

1. Боб выбирает пару (e, d) и шлёт ключ шифрования e (открытый ключ) Алисе по открытому каналу, а ключ дешифрования d (закрытый ключ) защищён и секретен (он не должен передаваться по открытому каналу).

2. Чтобы послать сообщение m Бобу, Алиса применяет функцию шифрования, определённую открытым ключом e: $E_e(m) = c$, c — полученный шифротекст.

3. Боб расшифровывает шифротекст c, применяя обратное преобразование D_d, однозначно определённое значением d.

1.2.3 Анализ существующих криптографических систем с открытым ключом

Хеш-функции могут быть заменены ассиметричными алгоритмами шифрования, при условии, что из системы будет удален приватный ключ, используемый для расшифровки данных [3]. Самыми распространёнными криптосистемами с открытым ключом являются RSA и схема Эль-Гамаля, данные криптографические протоколы обладают свойством гомоморфности [4] относительно умножения, которое позволяет производить атаку по подобранному и подобранному шифротексту. В криптографических протоколах с открытым ключом исчезает проблема коллизий, но появляются новые уязвимости Количество секретных параметров так же не велико, как и у хеш-функций. В системе RSA их три – тип функции, модуль и секретная экспонента.

1.2.3.1 RSA

В качестве односторонней функции использует умножение простых чисел, основываясь на вычислительной сложности задачи факторизации. RSA обладает свойством гомоморфности относительно умножения. Пусть m_1, m_2 - различные открытые тексты и c_1, c_2 - соответствующие им шифротексты. Процесс шифрования сообщения $m_1 * m_2$ имеет следующий вид, $(m_1 * m_2)^e = m_1^e * m_2^e = c_1 * c_2 (\mathrm{mod}\, n)$, что эквивалентно $E(k, m_1 m_2) = E(k, m_1) * E(k, m_2)$. Из данного преобразования видно, что из двух открытых текстов, можно получить третий путем их объединения, шифротекстом для которого будет объединение шифротекстов c_1, c_2.

Другая распространённая уязвимость протокола RSA - Common Modulus Attack [5]. Пусть модуль $N = p * q$ и $< e_1, N >$, $< e_2, N >$ два публичных ключа, при этом e_1 и e_2 взаимно простые. Открытый текст m зашифрован обоими публичными ключами. Имеется два шифротекста $c_1 = m^{e_1} (\mathrm{mod}\, N)$ и $c_2 = m^{e_2} (\mathrm{mod}\, N)$, тогда публичный ключ можно вычислить за полиномиальное время $\log(N)$. Используя расширенный алгоритм Евклида, находятся a_1, a_2 такие, что $a_1 * e_1 + a_2 * e_2 = 1$. Тогда отрытый текст легко вычисляется $c_1^{e_1} * c_2^{e_2} = m^{a_1 e_1} m^{a_2 e_2} = m^{a_1 e_1 + a_2 e_2} = m (\mathrm{mod}\, N)$.

Так же распространены случаи неправильного выбора параметров алгоритма RSA, вследствие чего схема становится возможной для взлома. Например, для ускорения шиф-

рования публичную экспоненту делают маленькой, в результате шифротекст m^e может оказаться меньше модуля $N = p * q$.

1.2.3.2 Схема Эль-Гамаля

Данная криптосистема с открытым ключом основывается на сложности вычисления дискретного логарифма в конечном поле. Схема Эль-Гамаля, так же как RSA, обладает свойством гомоморфности относительно умножения. Пусть m_1, m_2 - различные открытые тексты, c_1, c_2 - соответствующие им шифротексты. Опишем процесс шифрования сообщения $m_1 * m_2$, $m_1 * m_2 = b_1 * a_1^{(p-1-k)} b_2 * a_2^{(p-1-k)} \bmod p = (b_1 * b_2) * (a_1 * a_2)^{(p-1-k)} \bmod p$, что эквивалентно

$E(y, m1) = (y^{r1} m_1, g^{r1}), E(y, m_2) = (y^{r2} m_2, g^{r2}) \Rightarrow E(y, m_1 m_2) = E(y^{r1} y^{r2} m_1 m_2, g^{r1} g^{r2})$.

1.3 Постановка задач

Главной задачей является разработка криптосистемы с открытым ключом, зависящей от большого числа параметров. Программной реализацией работы являются графические приложения, с помощью которых пользователь может задавать параметры криптосистемы, а так же зашифровывать данные.

Криптографическая система должна удовлетворять следующим требованиям:

1) Система должна зависеть от большого количества параметров
2) Потенциально бесконечное множество возможных ключей
3) Использование в качестве шифрования вычисление функции в точке
4) Функции должны вычисляться относительно долго, чтобы значительно замедлить полный перебор
5) Система не должна обладать свойством гомоморфности

1.4 Конструирование системы с открытым ключом

Пусть p простое число, тогда кольцо вычетов по модулю p - Z_p является полем. Любой автоморфизм поля Z_p описывается, как многочлен. Для шифрования необходимы только такие многочлены, которые являются взаимно-однозначными отображениями (пе-

рестановками) – перестановочные многочлены. Дадим определение перестановочного многочлена из [6].

Многочлен $f \in F_q[x]$ называется перестановочным многочленом поля F_q, если соответствующая ему полиномиальная функция $f : F_q \to F_q$, отображает элемент $c \in F_q$ в элемент $f(c) \in F_q$, является перестановкой элементов поля F_q.

Согласно теореме о перестановочных многочленах, S_p порождается полиномами cx, $x+1$, x^{p-2} [6]. Так же одночлен x^n является перестановочным многочленом поля F_q тогда и только тогда, когда $НОД(\mathrm{n}, \mathrm{q}-1) = 1$, то есть числа n и $q-1$ взаимно простые. Обратным отображением к отображению x^{k} будет отображение x^{d}, где d обратный элемент k в кольце $\mathrm{Z}_{\mathrm{p-1}}$. Также обратимым отображением будет аффинное отображение вида ax+b, где a не равно нулю. Не зная p, как и в RSA найти обратный элемент можно только с помощью метода полного перебора.

Следовательно, перестановочный многочлен над полем может быть описан как различная суперпозиция отображений таких, что: $P(x) = (ax+b)^e + d$, где $a \in Z_p$, $a \neq 0$, b, d – произвольные элементы из Z_p, e – обратимый элемент из $Z_{\varphi(p)} = Z_{p-1}$ ($\varphi(p)$ - функция Эйлера). Такие отображения $P(x)$ назовем элементарными. Число всех взаимно-однозначных отображений можно легко посчитать - оно равно порядку симметрической группы S_p, то есть группе всех перестановок на множестве из p элементов – $p!$.

В качестве примера можно рассмотреть поле Z_{17}. Для него существует 17! равное 355687428096000 перестановочных многочленов. Максимальная степень многочлена равна 16. Элементарные многочлены вида x^n являются взаимно-однозначными при $n = 1,3,5,7,9,11,13,15$ в силу того, что функция Эйлера $\phi(17) = 16$ и все нечетные числа взаимно просты с 16.

Учитывая данные теоритические основы, предлагается следующая схема шифрования:

- Выбираются простые числа $p_1 \ldots p_r$. Модулем будет являться их произведение $n = \prod_{i=1}^{r} p_i$.

- Для каждого простого числа p_i выбирается набор элементарных полиномов $f_1^{p_i}(x)...f_d^{p_i}(x)$ над Z_{p_i} и вычисляется полином $F_{p_i}(x) \equiv f_1^{p_i}(...(f_d^{p_i}(x)...)(\mathrm{mod}\, Z_{p_i})$. После раскрытия скобок в $F_{p_i}(x)$ получается взаимно-однозначный полином над полем Z_{p_i}.

- При помощи китайской теоремы об остатках и расширенного алгоритма Евклида (Приложение А, Листинг 1) вычисляется полином $F(x)$ над полем Z_n: $F(x) = F_{p_i}(x)(\mathrm{mod}\, p_i) i = 1...r$.

Открытый ключ состоит из взаимно-однозначного многочлена $F(x)$ и модуля n. Закрытый ключ состоит из простых чисел, элементарных полиномов и порядка раскрытия скобок. Длина публичного ключа в битах равна произведению длины модуля в битах, умноженному на количество ненулевых коэффициентов многочлена.

1.4.1 Шифрование

Пусть Боб обладает публичным ключом Алисы (F, n) и хочет передать ей секретное сообщение m по открытому каналу. Для шифрования Боб переводит сообщение m в числовое представление и вычисляет шифротекст $c = F(m)(\mathrm{mod}\, n)$ после чего передает его Алисе по открытому каналу.

1.4.2 Дешифрование

Алиса хочет восстановить сообщение m из шифротекста c, используя свой приватный ключ. Для дешифрования используется следующий алгоритм:

- Алиса вычисляет числа $C_i \equiv c(\mathrm{mod}\, p_i)$, $i = 1...r$.

- Затем Алиса решает систему уравнений $F_{p_i}(x) \equiv C_i(\mathrm{mod}\, p_i)$, $i = 1...r$ используя информацию о построение отображений $F_{p_1}...F_{p_s}$. В результате Алиса получает систему $x \equiv m_i(\mathrm{mod}\, p_i)$ и, используя расширенный алгоритм Евклида, получает m – исходное сообщение, посланное Бобом.

1.5 Пример работы алгоритма

Следующий пример иллюстрирует работу алгоритма на малых параметрах.

- Пусть $p_1 = 7$, $p_2 = 17$, $p_3 = 23$. Модулем в данном случае является число $n = 7*17*23 = 2737$.

- Фиксируется набор элементарных полиномов для каждого простого числа p_i.

  - Для простого числа p_1: $f_1^{p_1}(x) = x + 1, f_2^{p_1}(x) = 5x, f_3^{p_1}(x) = x + 6$.
    $F_{p_1}(x) = f_1^{p_1}(f_2^{p_1}(f_3^{p_1}(x))) = 5x + 31 = 5x + 3(\mathrm{mod}\,7)$

  - Для простого числа p_2: $f_1^{p_2}(x) = x + 10, f_2^{p_2}(x) = 2x^5, f_3^{p_2}(x) = 7x$.
    $F_{p_2}(x) = f_1^{p_2}(f_2^{p_2}(f_3^{p_2}(x))) = 5x^5 + 10(\mathrm{mod}\,17)$

  - Для простого числа p_3: $f_1^{p_3}(x) = 2x + 3, f_2^{p_3}(x) = 12x, f_3^{p_3}(x) = x^7$.
    $F_{p_3}(x) = f_1^{p_3}(f_3^{p_3}(f_2^{p_3}(x))) = 9x^7 + 3(\mathrm{mod}\,23)$

- Пусть полином имеет вид $F(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5 + a_6x^6 + a_7x^7$. Для получения коэффициентов a_0 необходимо решить систему уравнений $a_0 = 3(\mathrm{mod}\,7), a_0 = 10(\mathrm{mod}\,17), a_0 = 3(\mathrm{mod}\,23)$.

  Простые числа 7,17,23 – взаимно простые. Данная система решается с применением китайской теоремы об остатках. Коэффициент a_0 имеет вид $a_0 = M_1y_1 + M_2y_2 + M_3y_3(\mathrm{mod}\,n)$, где $M_1y_1 = 3(\mathrm{mod}\,7)$, $M_2y_2 = 10(\mathrm{mod}\,17)$, $M_3y_3 = 3(\mathrm{mod}\,23)$ и $M = \prod_{i=1}^{3} p_i = 2737$. Следовательно $M_1 = M/7 = 391$, $M_2 = M/17 = 161$, $M_3 = M/23 = 119$.

  Результатом решения системы при помощи расширенного алгоритма Евклида являются обратные к M_i элементы $y_1 = 4, y_2 = 14, y_3 = 18$. Следовательно, коэффициент $a_0 = 4M_1 + 14M_2 + 18M_3 = 1564 + 2254 + 2142 = 486(\mathrm{mod}\,2737)$.

  Данный алгоритм аналогичен для вычисления остальных коэффициентов. С учетом коэффициентов полином имеет вид $F(x) = 952x^7 + 11127x^5 + 782x + 486$. Взаимнооднозначный полином $F(x)$ и модуль $n = 2737$ являются публичным ключом.

1.6 Безопасность криптосистемы

Предлагаемая система не является гомоморфной, так как по построению взаимно-однозначного полинома мономы запрещены. Следовательно, система устойчива к атака по подобранному и адаптивно подобранному шифротексту. Наилучшее известное время для взлома данной криптосистемы – субэкспоненциальное [8], как black-box задачи над полем простых чисел. Криптосистема основывается не только на сложности факторизации целых чисел, как RSA, но и на сложности разложения многочленов. Наилучшие алгоритмы факторизации работают за субэкспоненциальное время. Главной отличительной особенностью алгоритма является то, что секретных параметров намного больше, чем в криптосистемах RSA и Эль-Гамаля.

В экспериментах, которые проводились в работе [9] доказано, что длина цикла значений многочлена $F(x)$ над кольцом Z_n больше, чем многочленов $f_1^{p_i}(x)...f_d^{p_i}(x)$ над полем Z_{p_i}, $i = 1...r$. Таким образом, данная система устойчива к атаке на циклы.

2 Практическая часть

2.1 Программная реализация

2.1.1 Источники энтропии

Результатом программной реализации работы являются приложения для шифрования. Параметров у функции много и их переменное количество, пользователь может задать параметру вручную, но так же может использовать генератор случайных чисел. Использование генераторов псевдослучайных чисел, при генерации ключей ведет к понижению уровню защищённости системы, т.к. научившись предугадывать порядок чисел, атакующий сможет предсказать новые и старые значения ключей, классическим примером является линейный конгруэнтный метод генерации случайных чисел. Поэтому необходим хороший источник энтропии. Для этого используются системные генераторы случайных чисел. На Linux, Unix, MacOS системах это устройство /dev/urandom, а на Windows системах CryptoAPI.

2.1.2 White-box шифратор

Результатом данной реализации является графическое приложение “Encryptor creator”. В данном приложении пользователь задает приватный ключ:

1) Произвольный набор простых чисел $p_1,..,p_n$. Длина произведения простых чисел, должна быть не меньше установленного в программе значения, 512 бит. Пользователь может использовать список первого миллиона простых чисел, встроенный в программу, так же есть возможность ввести числа вручную или использовать генератор случайных чисел.

2) Произвольный набор элементарных взаимно-однозначных полиномов $f_i^{p_j}$. Полином описывается набором степеней и коэффициентов. Имеется функциональность позволяющая задать полиномы автоматически, используя генератор случайных чисел.

3) Порядок раскрытия скобок. Может быть введен пользователем вручную, либо используя генератор случайных чисел.

Программа строит многочлены $F_{p_i}(x) \equiv f_1^{p_i}(...(f_d^{p_i}(x)...)(\mathrm{mod}\, Z_{p_i})$ для каждого простого числа, затем вычисляет итоговый взаимно-однозначный многочлен $F(x)$. Используя

генерацию кода программа “Encryptor creator” создает скомпилированный файл “Encryptor” на языке python, в который помещается публичный ключ (F, n). Данный исполняемый файл “Encryptor” имеет только одну функцию – зашифровать открытый текст публичным ключом. Данная реализация является white-box, потом что программу можно исследовать. Исследователь программы может увидеть публичный ключ и схему шифрования, используя декомпилирование байт-кода, но из данных заложенных в программу “Encryptor” не получиться восстановить приватный ключ, и соответственно восстановить исходный текст по зашифрованному тексту [10].

Владелец безопасного хранилища задает программе “Encryptor creator” приватный ключ и в качестве результата получает приложение для шифрования данных “Encryptor”.

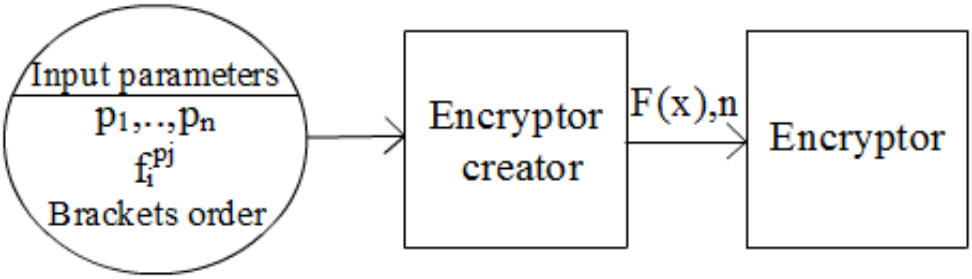

Рисунок 2: Схема работы программы

Для реализации данного приложения использовался язык программирования Python 2.7 и библиотека PyQt4, приложение кроссплатформенно и работает на операционных системах семейства Windows, UNIX, MacOS.

2.1.2.1 Структура программы

На рисунке 3 представлена структура проекта программы “Encryptor creator”.

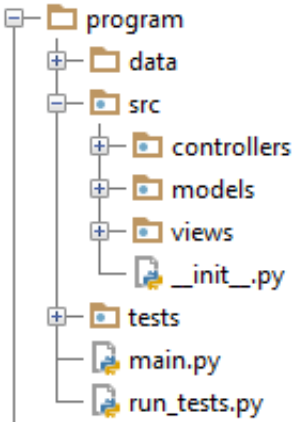

Рисунок 3: Структура исходного кода программы

Для того чтобы разделить данные и логику, представление и обработку событий пользовательского интерфейса используется шаблон проектирования Model-View-Controller. Пакет src содержит файлы исходного кода программы. Классы, находящиеся в пакете src.models, описывают данные и логику приложения. Классы, находящиеся в пакете src.views содержат элементы пользовательского интерфейса и используют библиотеку PyQt. Классы, находящиеся в пакете src.controllers обрабатывают события пользовательского интерфейса, такие как взаимодействие с кнопками, изменения значений текстовых полей и другие. В папке data находятся используемые программой данные - список простых чисел, шаблон файла для программы “Encryptor”.

Код пакета src.models покрыт unit-тестами на 83%. При написании тестов использовался встроенный в python фреймворк – unittest. Для оценки покрытия использовалась программа с открытым исходным кодом coverage.py. Все исходные коды unit-тестов находятся в пакете src.tests. Для профилирования и поиска ошибок в коде использовался бесплатный анализатор pylint. При помощи pylint был построен граф зависимостей между пакетами, данный граф представлен на рисунке 4. Для упорядочивания вершин использовалась топологическая сортировка.

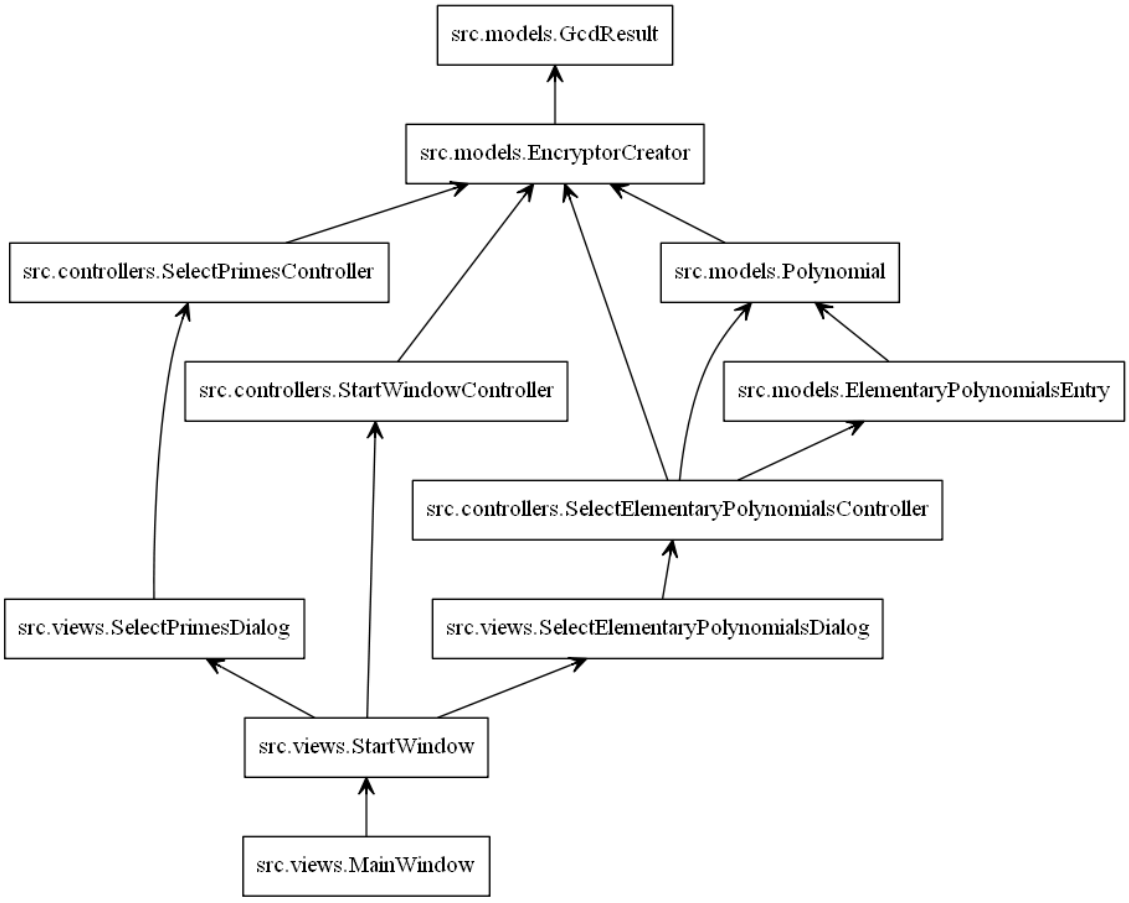

Рисунок 4: Граф зависимостей пакетов

Из графа видно, что компоненты View имеют связь с компонентами Model только через компоненты Controller. Дизайн связей пакетов в данной схеме удовлетворяет принципу “ацикличность зависимостей”, что позволяет переиспользовать пакеты.

2.1.3 Плагин для СУБД MySQL

Второй реализацией является плагин для системы управления баз данных MySQL и веб-интерфейса к ней phpMyAdmin. Владелец базы данных генерирует для каждого отношения публичный ключ. Шифрование можно использовать на уровне триггера на вставку и обновление кортежей в таблице. Имеется возможность зашифровать уже созданное отношение. Триггер исполняет роль программы “Encryptor” в реализации white-box шифратора.

2.1.4 Используемые алгоритмы

2.1.4.1 Быстрое возведение в степень

Основной операцией при вычислении значения многочлена в точке является возведение в степень. Для ускорения вычисления используется алгоритм быстрого возведения в степень по модулю. Показатель степени x в выражении $y = a^x \bmod p$ представляется в двоичной системе счисления $x = (x_t x_{t-1} ... x_1 x_0)_2$. Тогда число $y = a^x \bmod p$ может быть вычислено по формуле $y = \prod_{i=0}^{t} a^{x_i * 2i} \bmod p$. Количество умножений при вычислении по данной формуле не превосходит $2 * [\log_2 x]$ (Приложение А, Листинг 2).

2.1.4.2 Бином Ньютона

При композиции элементарных взаимно-однозначных отображений возникает необходимость возведения в степень суммы двух переменных. Для этого используется бином Ньютона $(a+b)^n = \binom{n}{0} a^n + \binom{n}{1} a^{n-1} b + ... + \binom{n}{k} a^{n-k} b^k + ... + \binom{n}{n} b^n$, где $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ - биномиальные коэффициенты. Самой трудоемкой операцией здесь является расчет факториала, поэтому используется отложенный расчет факториалов с запоминанием, и использование результатов для расчётов на последующих итерациях.

Заключение

В ходе работы были рассмотрены наиболее распространенные существующие решения в области криптографии для хранения конфиденциальных данных. Был разработан алгоритм шифрования с открытым ключом, зависящий от произвольного количества параметров. Для данного алгоритма в процессе работы не были выявлены уязвимости. Было реализовано два приложения для шифрования данных. Цель проекта была успешно достигнута.

В дальнейшем криптосистему можно сделать вероятностной, что позволит принципиально усилить шифрование, путем изменения кодирования битов входного текста. При вероятностном шифровании открытого текста m с помощью одного и того же открытого ключа можно получить разные зашифрованные тексты $C_1 = E_k(m), C_2 = E_k(m), ..., C_N = E_k(m)$, которые при расшифровке дают один и тот же открытый текст $m = D_k(C_1) = D_k(C_2) = ... = D_k(C_N)$.

На конкурсе студенческий докладов РусКрипто 2013, работа заняла 3 место и представлена к публикации в журнале “Системы высокой доступности”, издаваемом под редакцией ВАК.

Работа была удостоена диплома первой степени на 51-ой Международной научной конференции “Студент и научно-технический прогресс” и опубликована в сборнике материалов конференции [11].

Работа была представлена в финале секции Young School международного форума по практической безопасности Positive Hack Days, и представлена к публикации в журнале “Безопасность информационных технологий”, издаваемом под редакцией ВАК.

Литература

1. Gosney J.M. Password Cracking HPC // Passwords^12 Security Conference, 2012. [Электронный ресурс] — Режим доступа: http://passwords12.at.ifi.uio.no/Jeremi_Gosney_Password_Cracking_HPC_Passwords12.pdf, свободный.

2. Let’s talk about password storage [Электронный ресурс] — Режим доступа: http://blog.mozilla.org/webdev/2012/06/08/lets-talk-about-password-storage, свободный.

3. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. – М.: Триумф, 2012. – 816 с.

4. Варновский Н. П. Гомоморфное шифрование // Труды Института Системного Программирования. Том 12. М: ИСП РАН, 2007, с. 27-36.

5. Salah I. K., Radi A. D., Oqeili S. Mathematical Attacks on RSA Cryptosystem // Journal of Computer science, 2006. - Т. 2. - No. 8. С. 665 – 671.

6. Лидл Р. Конечные поля. – М.: Мир. 1988. – 808 с.

7. Галуев Г.А. Математические основы криптологии: Учебно-методическое пособие. – Таганрог: Изд-во ТРТУ. 2003. – 120 с.

8. Boneh D., Lipton J.R. Algorithms for black-box fields and their application to cryptography // 16th Annual International Cryptology Conference, 1996. – Т. 1109. С. 283 – 297.

9. Кренделев С.Ф., Спицына Е.О. Число 667. Проверка RSA на устойчивость. Вариант криптографии с открытым ключом // Системы высокой доступности, 2011. - Т. 7. – No. 2. - С. 34 – 38.

10. Wyseur B. White-Box Cryptography // Katholieke Universiteit Leuven. 2009. – 169 с.

11. Арыков Н.Е., Спицына Е.О. Разработка алгоритма шифрования с открытым ключом и его применение для построения безопасного хранилища данных // Материалы 51-й Международной научной студенческой конференции «Студент и научно-технический прогресс»: Информационные технологии. – Новосиб. гос. ун-т. Новосибирск, 2013. – С. 114.

Приложение А

(обязательное)

Листинг 1. Метод gcdGeneralized класса EncryptorCreator

```
@staticmethod
def gcdGeneralized(a, b):
    """
    a*x + b*y = gcd(a, b)
    Return x, y, gcd(a,b)
    """
    if a < b:
        a, b = b, a
    U = GcdResult(a, 1, 0)
    V = GcdResult(b, 0, 1)
    T = GcdResult(0, 0, 0)
    while V.getGcd() > 0:
        # q = U1 div V1
        q = int(U.getGcd() / V.getGcd())
        # U1 mod V1
        T.setGcd(U.getGcd() % V.getGcd())
        # U_x - q*V_x
        T.setX(U.getX() - (q * V.getX()))
        # U_y - q*V_y
        T.setY(U.getY() - (q * V.getY()))
```

```
        U.setGcd(V.getGcd())
        U.setX(V.getX())
        U.setY(V.getY())
        V.setGcd(T.getGcd())
        V.setX(T.getX())
        V.setY(T.getY())
    return U
```

Листинг 2. Метод quickStepByModule класса EncryptorCreator

```
@staticmethod
def quickStepByModule(a, x, p):
    """Return a ^ x mod p"""
    result = 1
    s = a
    bits = bin(x)[2:]
    bitLength = len(x)
    for i in reversed(range(0, bitLength)):
        if '1' == x[i]:
            result = (result * s) % p
        s = (s * s) % p
    return result
```