

Реферат

Тема дипломной работы: «Разработка лабораторной работы на тему: "Побочные атаки на криптосистему RSA"»

Пояснительная записка содержит:

- страниц: 100 (из них 12 – приложения)
- рисунков: 41
- таблиц: 2
- приложений: 2
- используемых источников: 15

Ключевые слова: криптосистема RSA, побочные атаки, модульная арифметика, алгоритм нахождения корня, непрерывные дроби, факторизация, криптоанализ, разработка лабораторной работы, графический интерфейс, длинная арифметика.

Цель работы – разработка программного комплекса для постановки лабораторной работы на тему «Побочные атаки на криптосистему RSA», которая входит в состав дисциплины «Основы криптографии с открытым ключом». Под побочными атаками в рамках данной работы подразумеваются следующие шесть методов криптоанализа RSA: атака на малую открытую экспоненту, атака при малом числе возможных сообщений, атака на малую секретную экспоненту, атака, использующая мультипликативное свойство шифра RSA, циклическая атака и атака на общие модули.

В результате были исследованы и успешно реализованы на практике алгоритмы вышеперечисленных побочных атак, разработана кроссплатформенная программа с «дружественным» графическим интерфейсом, демонстрирующая работу этих алгоритмов и методические указания по выполнению лабораторной работы.

Содержание

Введение.....	7
Глава 1. Криптографические системы с открытым ключом	9
1.1 Классификация криптосистем	9
1.1.1 Симметричные криптосистемы	9
1.1.2 Несимметричные криптосистемы	10
1.2 Метод распределения ключей Диффи-Хеллмана	12
1.3 Криптосистема RSA (Райвеста-Шамира-Адлемана)	13
1.3.1 RSA алгоритм.....	14
1.3.1 Математическое обоснование алгоритма RSA.....	16
1.3.2 Стойкость криптосистемы RSA	20
Глава 2. Описание побочных атак на RSA	22
2.1 Атака на малую открытую экспоненту	22
2.2 Атака при малом объеме возможных сообщений	28
2.3 Атака на малую секретную экспоненту.....	29
2.3.1 Необходимые сведения о непрерывных дробях	29
2.3.3 Применение «алгоритма непрерывных дробей» для нахождения малой секретной экспоненты в RSA.....	35
2.3.4 Пример выполнения атаки на малую секретную экспоненту	38
2.3.5 Способы борьбы с атакой на малую секретную экспоненту	41
2.4 Атака с использованием мультипликативного свойства шифра RSA .	42
2.5 Циклическая атака.....	43
2.6 Атака на общие модули	44
Глава 3. Разработка лабораторной работы «Побочные атаки на криптосистему RSA»	49
3.1 Задачи, решаемые при разработке программного обеспечения	49
3.2 Инструменты для разработки программного обеспечения	49
3.3 Моделирование криптосистемы RSA	53
3.4 Разработка графического интерфейса.....	55

3.4.1 Окно демонстрации атаки на малую экспоненту шифрования.....	58
3.4.2 Окно демонстрации атаки при малом числе возможных сообщений	61
3.4.3 Окно демонстрации атаки на малую дешифрующую экспоненту .	64
3.4.4 Окно демонстрации атаки, связанной с мультипликативным свойством шифра RSA.	67
3.4.5 Окно демонстрации циклической атаки.....	70
3.4.6 Окно демонстрации атаки на общие модули	72
3.5 Локализация программы	74
Глава 4. Методические указания по проведению лабораторной работы.....	78
4.1 Рекомендации по подготовке к лабораторной работе.....	78
4.2 Методические указания к лабораторной работе.....	79
Заключение	85
Список литературы:	87
Приложение 1. Классы криптосистем.....	89
Приложение 2. Реализация алгоритмов побочных атак.....	94

Введение

Необходимость коммуникаций между двумя сторонами без риска прослушивания сообщений третьей стороной существовала всегда. С изобретением новых методов коммуникаций и увеличением расстояний между корреспондентами возникла потребность в системах, позволяющих отправлять сообщения между двумя сторонами так, чтобы злоумышленник не мог извлечь никакой информации из передаваемых сообщений. Разработкой и улучшением таких методов занимается наука «криптография», что в переводе с древнегреческого означает «тайное письмо».

Криптография - это наука о методах обеспечения конфиденциальности и аутентичности информации. Конфиденциальность подразумевает невозможность прочтения информации третьей стороной, а аутентичность – целостность информации и подлинность авторства. В век информационных технологий информация представляет огромную ценность. Интернет, сотовая связь, финансовые операции и многое другое нуждается в надежной защите, поэтому криптографические методы и протоколы очень востребованы.

Традиционная криптография образует раздел симметричных криптосистем, в которых шифрование и дешифрование производится с использованием одного и того же секретного ключа. Помимо этого раздела современная криптография включает в себя изучение асимметричных криптосистем, систем электронной цифровой подписи (ЭЦП), хеш-функций и криптографических протоколов. В данной работе рассматривается асимметричная криптосистема RSA и методы ее криптоанализа при различных параметрах криптосистемы.

Не смотря на то, что RSA является первой реализацией идеи криптосистемы с открытым ключом и является довольно простой в изучении принципов асимметричной криптографии, она широко применяется на практике в настоящее время, как для шифрования, так и для цифровой

подписи. Именно поэтому изучение данной криптосистемы и методов ее криптоанализа представляет интерес.

Побочные атаки на криптосистему RSA представляют угрозу информационной безопасности, так как они не связаны с решением каких-либо неполиномиально сложных задач и позволяют дешифровать сообщения в полиномиальное время, выполнив некоторые вычисления. Необходимо знать причины возникновения возможности использования побочных атак и способы борьбы с ними. Поскольку изучение данных этих атак не требует специальных знаний по криптоанализу, они могут быть изучены студентами в рамках курса «Основы криптографии». Не смотря на то, что в лекциях дается необходимая для этого база, для закрепления такой важной темы была бы весьма кстати отдельная лабораторная работа. Именно поэтому тема настоящей дипломной работы является весьма актуальной.

В качестве цели выступает разработка программного комплекса для постановки лабораторной работы по курсу «Криптография с открытым ключом». Необходимо создать модули для моделирования криптосистемы RSA и наглядной демонстрации побочных методов криптоанализа, угроза которых возникает вследствие выбора неправильных параметров криптосистемы.

В первой главе представлен краткий обзор криптосистем, используемых в современной криптографии, подробно рассмотрена криптосистема RSA, процедура генерирования ключей, алгоритмы шифрования дешифрования.

Вторая глава полностью посвящена побочным методам криптоанализа системы RSA, приведены доказательства и примеры выполнения атак.

Третья глава описывает процесс создания лабораторного комплекса. Объясняются основные этапы разработки, методы решения поставленных задач и используемые для разработки программы инструменты.

В четвертой главе представлены методические указания для выполнения лабораторной работы. Приводятся рекомендации по подготовке к работе, ее выполнению и составлению отчета.

Глава 1. Криптографические системы с открытым ключом

1.1 Классификация криптосистем

Криптографические алгоритмы используют ключи для выполнения операций шифрования и дешифрования. Согласно принципу Керхгофа [1], предполагается, что любому пользователю известно о криптосистеме все, кроме ключа для выполнения операции дешифрования. В свою очередь по типу ключа криптосистемы делятся на два класса:

1. Симметричные криптосистемы
2. Асимметричные криптосистемы

У каждого из этих двух типов криптосистем есть свои достоинства и недостатки, определяющие область их применения.

1.1.1 Симметричные криптосистемы

Симметричными называются криптосистемы, в которых один и тот же ключ используется как для шифрования, так и для дешифрования сообщений. В таких криптосистемах перед началом закрытого сеанса связи взаимодействующие стороны договариваются о том, какой ключ будет использован, и сохраняют его в секрете.

Симметричные шифры делятся на блочные и потоковые. Блочные шифры оперируют блоками сообщений определённой длины, шифруя их по одному и тому же правилу. В свою очередь потоковые шифры преобразуют каждый символ сообщения в символ криптограммы в зависимости от ключа и положения символа в потоке открытого текста.

К достоинствам симметричных криптосистем по отношению к асимметричным относятся:

- высокая скорость работы
- простота реализации (благодаря более простым операциям)
- меньшие длины ключей

В свою очередь к явным недостаткам симметричных шифров относятся:

- большое количество ключей для возможности связи каждого пользователя с любым другим в большой сети
- проблема надежной передачи и сохранения в тайне ключей

За счет своей высокой скорости работы симметричные шифры более эффективны при передаче большого количества данных. В частности, потоковые шифры наилучшим образом подходят в ситуациях, когда по сетям передается трафик реального времени, например, в городских телефонных сетях, сотовых сетях, VoIP трафик и т.д. Блочные шифры также хорошо подходят для локального шифрования таких конфиденциальных данных, передавать которые не предполагается, поскольку в таком случае нет проблемы передачи секретного ключа по небезопасным каналам.

1.1.2 Несимметричные криптосистемы

Несимметричные криптосистемы (или криптосистемы с открытым ключом) отличаются от симметричных наличием двух ключей: открытого для шифрования и закрытого для дешифрования. Преобразования, используемые для выполнения операций шифрования и дешифрования являются односторонними преобразованиями с «подсказкой», где подсказкой является соответствующий ключ. При наличии подсказки данные преобразования легко выполняются. В криптосистемах с открытым ключом любому пользователю известны алгоритм шифрования и дешифрования, а также открытый ключ, в то время как закрытый ключ известен только его владельцу. Таким образом, для асимметричных криптосистем выполнение дешифрования без секретного ключа-подсказки или выполнение обратного преобразования не является возможным в обозримое время.

В асимметричных криптосистемах нахождение ключа дешифрования при известном ключе шифрования является вычислительно сложным. По этой причине ключ шифрования может быть сделан общедоступным, и процедура распределения ключей в криптосистемах с открытым ключом упрощается в сравнении с симметричными криптосистемами. С другой стороны, перед

использованием открытого ключа для шифрования нужно быть уверенным в том, что предлагаемый открытый ключ в действительности является открытым ключом получателя сообщения. Иными словами, возникает проблема аутентификации открытых ключей пользователей. На практике эта проблема решается за счет использования специализированных центров распределения ключей (ЦРК). Доверие к открытому ключу определяется степенью доверия к ЦРК.

В отличие от симметричных криптосистем, стойкость которых доказывается относительно определенных видов известных атак, стойкость криптосистем с открытым ключом напрямую зависит от сложности решения строго определенной математической задачи. В этом плане асимметричные криптосистемы позволяют доказать их стойкость проще, чем в традиционных симметричных криптосистемах.

В итоге к основным преимуществам асимметричных криптосистем по сравнению симметричными относятся:

- отсутствие необходимости предварительного согласования ключа перед сеансом связи
- отсутствие необходимости менять ключ после каждого сеанса связи
- меньшее количество ключей в системе
- возможность выполнения других криптографических функций

Еще одной характерной особенностью криптосистем с открытым ключом является то обстоятельство, что алгоритм шифрования и дешифрования в них обычно выполняются как действия по модулю некоторого целого числа, в то время как для традиционных симметричных криптосистем более типичными являются булевы операции над двоичными числами.

К недостаткам асимметричных криптосистем по сравнению с симметричными можно отнести следующие факторы:

- низкая скорость работы
- сложность реализации

- высокие требования к вычислительным ресурсам

Таким образом, асимметричные криптосистемы упрощают процедуру распределения ключей, но значительно медленнее симметричных. Это особенно существенно при шифровании больших объемов информации, поэтому асимметричные криптосистемы не всегда целесообразно использовать, как самостоятельные средства шифрования данных. Для объединения положительных свойств симметричных и несимметричных криптосистем на практике часто используют гибридные криптосистемы, где несимметричные алгоритмы выполняют функцию распределения ключей, а непосредственно шифрование реализуют симметричные криптосистемы. Поскольку обновление ключей в асимметричных криптосистемах требуется относительно редко, гибридная система обеспечивает практически такую же скорость шифрования-дешифрования, как и симметричная.

1.2 Метод распределения ключей Диффи-Хеллмана

Для решения задачи только по распределению ключей к симметричным системам можно применить асимметричный алгоритм распределения ключей Диффи-Хеллмана. Данный метод позволяет абсолютно любым двум случайным сторонам согласовать по небезопасному каналу секретный ключ, но не может быть использован непосредственно для шифрования.

Перед выполнением алгоритма стороны A и B согласуют два параметра: простое число p и любое целое число $\alpha \in Z_p$. Затем выполняется протокол, показанный на рис. 1.1.

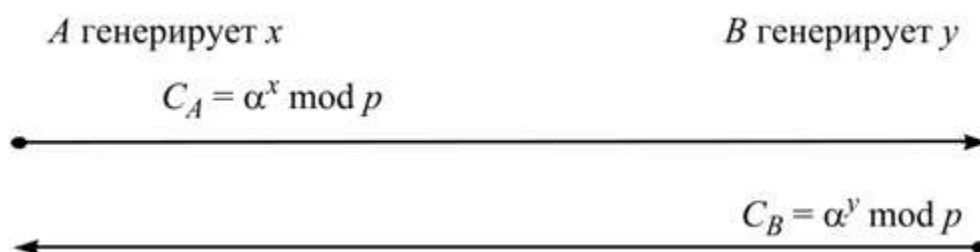


Рис. 1.1 Распределение ключей по методу Диффи-Хеллмана

Сторона A выбирает случайное большое целое число x и посылает стороне B число $C_A = \alpha^x \bmod p$. Аналогично сторона B выбирает число y и посылает стороне A число $C_B = \alpha^y \bmod p$.

После получения соответственно C_A и C_B стороны A и B вычисляют общий ключ K следующим образом:

$$\begin{cases} K_A = C_B^x \bmod p = (\alpha^y)^x \bmod p = \alpha^{xy} \bmod p = K_A \\ K_B = C_A^y \bmod p = (\alpha^x)^y \bmod p = \alpha^{xy} \bmod p = K_B \end{cases}, \quad K_A = K_B = K$$

При таком подходе никто из подслушивающих канал связи между двумя сторонами не сможет узнать согласованный секретный ключ K , поскольку для вычисления секретного ключа подслушивающий должен также иметь x или y . Однако получение x или y , которые не передаются в чистом виде по каналу связи при выполнении данного протокола, требует вычисления дискретного логарифма $\log_{\alpha} C \bmod p$, что является невыполнимой задачей при больших p .

Стойкость метода распределения ключей Диффи-Хеллмана обусловлена так называемой проблемой Диффи-Хеллмана, которая формулируется следующим образом: зная p , α , $\alpha^x \bmod p$, $\alpha^y \bmod p$, найти ключ $\alpha^{xy} \bmod p$ [1].

Однако, данный метод может быть полностью скомпрометирован, если активный злоумышленник выдает себя за одну из сторон. Чтобы этого избежать на практике, необходимо решить проблему аутентификации сторон, как и при использовании любого другого асимметричного алгоритма.

1.3 Криптосистема РША (Райвеста-Шамира-Адлемана)

В 1976 году Уитфилд Диффи и Мартин Хеллман предложили концепцию криптосистемы с открытым ключом в своей работе «Новые направления в современной криптографии» [3]. Затем уже в 1977 году учеными Рональдом Ривестом, Ади Шамиром и Леонардом Адлеманом был предложен первый пригодный для шифрования и цифровой подписи асимметричный алгоритм – криптосистема РША (RSA) [4].

1.3.1 РША алгоритм

Ключом шифрования в РША является пара положительных целых чисел (e, n) . Аналогичным образом ключом дешифрования является пара положительных целых чисел (d, n) . Каждый пользователь криптосистемы делает свой ключ шифрования общедоступным и хранит в секрете соответствующий закрытый ключ.

Теперь рассмотрим процесс генерирования ключей. Сначала вычисляется модуль n , как произведение двух простых целых чисел p и q :

$$n = p \cdot q$$

Целые числа обладают большой битовой длиной и являются случайными. Длиной ключа в РША является битовая длина модуля криптосистемы n . Так, если p и q имеют длину порядка 512 бит, то их произведение, т.е. модуль криптосистемы, будет порядка 1024 бит. Не смотря на то, что число n является общедоступным, его сомножители p и q остаются неизвестными, поскольку задача факторизации, т.е. разложения на множители, числа n является неполиномиально сложной [1]. По этой же причине секретная экспонента дешифрования d не может быть получена при известной открытой экспоненте e .

Затем случайным образом выбирается большое число d взаимно простое с $\varphi(n) = (p-1) \cdot (q-1)$, где $\varphi(n)$ - функция Эйлера, равная количеству целых неотрицательных чисел меньших n и взаимно простых с n . Иными словами, d удовлетворяет равенству:

$$\gcd((p-1) \cdot (q-1)) = 1,$$

где "gcd" означает "наибольший общий делитель".

Целое число e вычисляется с использованием $\varphi(n)$ и d , как обратная величина от d по модулю $\varphi(n)$. Таким образом, уравнение для вычисления открытой экспоненты e имеет вид:

$$e = d^{-1} \bmod \varphi(n) \quad (1)$$

Заметим, что существует отличие в методе вычисления обратной экспоненты при реализации криптосистемы РША на практике. Исходный документ, описывающий криптосистему [4], а также большинство учебных пособий предполагают вычисление обратной экспоненты по модулю $\varphi(n) = (p-1) \cdot (q-1)$.

Однако, современные стандарты (PKCS#1 [5] и RFC 3447 [6]) предписывает вычисление обратного элемента по формуле:

$$e \cdot d^{-1} \bmod \lambda(n),$$

где $\lambda(n)$ - функция Кармайкла, вычисляемая как:

$$\lambda(n) = \lambda(p \cdot q) = \text{lcm}(p-1, q-1), \quad (2)$$

где lcm означает наименьшее общее кратное [7].

Использование функции Кармайкла в настоящее время является распространенной практикой при реализации криптосистемы [5, 6]. Такой метод вычисления обратной экспоненты является немного более эффективным, поскольку значение функции Кармайкла почти всегда меньше, чем значение функции Эйлера.

Для шифрования в криптосистеме РША сообщение M представляется в виде целого числа от 0 до $n-1$ (длинные сообщения разбиваются на последовательности блоков, представленные в виде целых чисел).

Затем шифрование производится посредством возведения сообщения M в степень секретной экспоненты e по модулю n . Иными словами, результат шифрования (криптограмма C) – это остаток от деления M^e на число n .

В свою очередь для дешифрования криптограмма C возводится в степень секретной экспоненты d по модулю n . В итоге операции шифрования E и дешифрования D выглядят следующим образом:

$$C \equiv E(M) \equiv M^e \pmod{n}$$

$$M \equiv D(C) \equiv C^d \pmod{n}$$

Заметим, что шифрование в РША не увеличивает размеров сообщения, сообщение и криптограмма лежат в диапазоне чисел от 0 до $n-1$.

1.3.1 Математическое обоснование алгоритма RSA

Для доказательства справедливости алгоритма RSA и его пригодности для реализации шифрования и цифровой подписи необходимо доказать, что последовательное применение к сообщению M операции шифрования E и операции дешифрования D и наоборот приводит к исходному сообщению M :

$$D(E(M)) = (M^e \bmod n)^d \bmod n = M \quad (3)$$

$$E(D(M)) = (M^d \bmod n)^e \bmod n = M \quad (4)$$

Можно утверждать, что выражения (3) и (4) идентичны, поскольку

$$(M^e \bmod n)^d \bmod n = (M^d \bmod n)^e \bmod n = M^{ed} \bmod n$$

Таким образом, задача доказательства справедливости алгоритма RSA сводится к доказательству того, что:

$$M^{ed} \bmod n = M .$$

Из уравнения (1) видим, что открытая экспонента e и секретная экспонента d связаны следующим соотношением:

$$ed - 1 \equiv 0 \pmod{\varphi(n)}$$

Это означает, что $\varphi(n)$ делит $ed - 1$.

Поскольку модуль криптосистемы n является произведением простых целых чисел p и q , которые в свою очередь также являются взаимно простыми относительно друг друга, для вычисления функции Эйлера от n можно применить свойство мультипликативности функции Эйлера [1]. Таким образом, функция Эйлера от n является произведением значений функции Эйлера от p и q соответственно:

$$\varphi(n) = \varphi(pq) = \varphi(p) \cdot \varphi(q)$$

Значит оба числа $\varphi(p)$ и $\varphi(q)$ делят $ed - 1$:

$$\varphi(p) \mid ed - 1 \quad (5)$$

$$\varphi(q) \mid ed - 1 \quad (6)$$

Выражение (5) означает, что существует некое целое число k такое, что:

$$ed - 1 = k \cdot \varphi(p) , \quad (7)$$

а поскольку p является целым числом, функция Эйлера вычисляется, как

$$\varphi(p) = p - 1,$$

и выражение (7) можно переписать, как:

$$ed - 1 = k \cdot (p - 1) \quad (8)$$

Теперь рассмотрим случай возведения любого целого числа M в степень ed по модулю простого числа p :

$$M^{ed} \equiv M^{ed-1+1} = (M^{ed-1}) \cdot M \pmod{p}$$

Воспользовавшись заменой $ed - 1$ из (8), получим:

$$M^{ed} \equiv (M^{k(p-1)}) \cdot M \pmod{p} \quad (9)$$

Поскольку p является простым целым числом, любое целое число M будет либо взаимно простым с p (иными словами $\gcd(M, p) = 1$), либо будет являться кратным числу p (то есть $\gcd(M, p) = p$).

В случае, когда M является взаимно простым с p , из малой теоремы Ферма следует, что:

$$M^{p-1} \equiv 1 \pmod{p},$$

а это значит, что при любом положительном целом числе k справедливо:

$$M^{k(p-1)} \equiv 1^k \equiv 1 \pmod{p} \quad (10)$$

Учитывая (10), выражение (9) может быть переписано, как:

$$M^{ed} \equiv 1 \cdot M \equiv M \pmod{p} \quad (11)$$

Если же число M является кратным числу p , то число M , возведенное в степень любого положительного целого числа, будет также кратным числу p , а это значит, что:

$$M^{ed} \equiv 0 \equiv M \pmod{p} \quad (12)$$

Учитывая результаты анализа при $\gcd(M, p) = 1$ (11) и $\gcd(M, p) = p$ (12), можно утверждать, что для любых положительных целых чисел M справедливо:

$$M^{ed} \equiv M \pmod{p} \quad (13)$$

Делая аналогичные выводы для (6) получаем, что:

$$M^{ed} \equiv M \pmod{q} \quad (14)$$

Теперь необходимо доказать, что справедливость выражений (13) и (14) приводит к справедливости:

$$M^{ed} \equiv M \pmod{n} \quad (15)$$

Нам известно, что p и q являются простыми числами, а следовательно они взаимно просты относительно друг друга. Из выражения (13) видим, что p делит $M^{ed} - M$, а значит $M^{ed} - M$ можно представить в виде:

$$M^{ed} - M = p \cdot k, \quad (16)$$

где k – некое положительное целое число.

С другой стороны из выражения (14) известно, что q также делит $M^{ed} - M$, а значит, как видно из выражения (16), q также делит $p \cdot k$. Поскольку p и q являются взаимно простыми числами, q не может делить p , а значит q делит число k из (16). Тогда $M^{ed} - M$ может быть представлено в виде:

$$M^{ed} - M = p \cdot q \cdot l, \quad (17)$$

где l – некое положительное целое число.

Из выражения (17) видно, что $n = p \cdot q$ делит $M^{ed} - M$, а значит выражение (15) является справедливым.

Поскольку сообщение M лежит в диапазоне $0 \leq M < n$, каждому сообщению соответствует уникальная криптограмма, а значит из криптограммы может быть однозначно восстановлено исходное сообщение.

Теперь докажем, что при реализации криптосистемы РША с использованием функции Кармайкла (2) для вычисления обратной экспоненты алгоритм РША остается справедливым. Значение функции Кармайкла для заданного положительного числа n – это такая минимально возможная экспонента m , при которой для любого числа k взаимно простого с n выполняется условие:

$$k^m \equiv 1 \pmod n$$

Заметим, что аналогичное условие выполняется и для значения функции Эйлера, но при этом значение функции Эйлера не является наименьшим из возможных.

Для нечетных простых чисел и их степеней, а также для чисел 2 и 4 значение функции Кармайкла совпадает со значениями функции Эйлера. Для любых степеней двойки (за исключением чисел 2 и 4) значение функции Кармайкла равно половине значения функции Эйлера. Во всех остальных случаях для нахождения значения функции Кармайкла число n представляется в виде произведения степеней простых чисел (основная теорема арифметики), наименьшее общее кратное которых и будет являться значением функции Кармайкла [7].

Таким образом, общее правило для нахождения значения функции Кармайкла может быть записано следующим образом:

$$\begin{cases} \lambda(p^\alpha) = \varphi(p^\alpha) & \text{при } p = 2 \text{ и } \alpha \leq 2, \text{ или } p \geq 3 \\ \lambda(2^\alpha) = \frac{1}{2} \varphi(2^\alpha) & \text{при } \alpha \geq 3 \\ \lambda(n) = \text{lcm}(\lambda(p_i^{\alpha_i})) & \text{если } n = \prod_i p_i^{\alpha_i} \end{cases}$$

Так как модуль криптосистемы РША является произведением двух нечетных простых чисел p и q , значение функции Кармайкла вычисляется, как:

$$\lambda(n) = \text{lcm}[\lambda(p), \lambda(q)] = \text{lcm}[\varphi(p), \varphi(q)] = \text{lcm}[p-1, q-1],$$

где задача вычисления наименьшего общего кратного может быть сведена к вычислению наибольшего общего делителя [7]:

$$\text{lcm}[p-1, q-1] = \frac{(p-1) \cdot (q-1)}{\text{gcd}(p-1, q-1)}, \quad (18)$$

Видно, что значение функции Эйлера $\varphi(n)$ от модуля криптосистемы РША кратно значению функции Кармайкла $\lambda(n)$, а поскольку утверждение (15) уже было доказано ранее для случая $ed = 1 + \varphi(n)$, то оно также останется верным для $ed = 1 + \lambda(n)$. Из этого следует, что при реализации

криптосистемы РША, где обратная экспонента вычисляется по модулю $\lambda(n)$ (2), алгоритмы шифрования и дешифрования также являются справедливыми.

1.3.2 Стойкость криптосистемы РША

Нетрудно заметить, что криптосистема РША может быть вскрыта, если удастся найти простые числа p и q , т.е. факторизовать n .

Исходя из этого факта, p и q должны выбираться такой большой разрядности, чтобы факторизация числа n потребовала необозримо большого времени даже при использовании всех доступных и современных средств вычислительной техники. В настоящее время задача факторизации чисел не имеет полиномиального решения. Разработаны лишь некоторые алгоритмы, упрощающие факторизацию, но их выполнение при факторизации чисел большой разрядности все равно требует необозримо большого времени [1].

Другой естественной атакой на криптосистему РША является вычисление секретной экспоненты $d = \log_c M \bmod n$ при известных сообщении M и соответствующей ему криптограмме C . Данная задача называется задачей дискретного логарифмирования по модулю многоразрядных чисел и также относится к трудным в математике, имея почти такую же сложность, как и задача факторизации [1].

Помимо двух основных атак на криптосистему РША (факторизации и логарифмирования) существует ряд побочных атак, связанных с неправильным выбором параметров криптосистемы при ее разработке и эксплуатации. Побочные атаки на криптосистему РША являются предметом исследования настоящей дипломной работы, а их описание приводится в следующей главе.

Выводы по главе

Важность криптосистем с открытым ключом с точки зрения их практического применения особенно возросла в последние годы в связи с широким использованием электронных подписей и криптографических

протоколов для замены бумажного документооборота электронным. Криптосистема RSA довольно проста в изучении и вместе с тем находит активное применение на практике, поэтому ее изучение является важным этапом в освоении раздела криптосистем с открытым ключом. Изучение побочных атак на RSA позволит лучше понять зависимость стойкости криптосистемы от ее параметров, а также избежать типичных ошибок при ее проектировании.

Глава 2. Описание побочных атак на RSA

2.1 Атака на малую открытую экспоненту

Первая побочная атака, называемая также атакой Хастада [8], связана с выбором малой величины экспоненты шифрования e в целях ускорения операции шифрования. Например, $e=3$ часто используется на практике, поскольку шифрование в таком случае требует выполнения одной операции возведения в квадрат и одной операции умножения.

Малая открытая экспонента может быть использована злоумышленником, когда одно и то же сообщение транслируется несколькими участниками. Допустим, легитимный пользователь хочет отправить k получателям одинаковое сообщение M , при этом для увеличения скорости шифрования криптосистема спроектирована таким образом, что используется одинаковая малая шифрующая экспонента $e \leq k$, но разные модули n_i , а шифруемое сообщение M меньше любого из модулей n_i . Тогда, имея любые e криптограмм из C_1, \dots, C_k и e соответствующих наборов открытых ключей из $(e, n_1), \dots, (e, n_k)$, можно восстановить исходное сообщение M в полиномиальное время. Совокупность криптограмм и открытых ключей можно представить в виде системы уравнений:

$$\begin{cases} C_1 = M^e \bmod n_1 \\ C_2 = M^e \bmod n_2 \\ \vdots \\ C_k = M^e \bmod n_k \end{cases} \quad (19)$$

При этом, как уже отмечалось, для выполнения атаки злоумышленнику достаточно иметь систему из любых e уравнений системы (19). Весьма вероятно, что будет также выполнено условие:

$$\gcd(n_i, n_j) = 1 \text{ при } i \neq j.$$

Тогда, согласно китайской теореме об остатках [1], существует общее решение системы (19):

$$C' \equiv M^e \bmod \prod_{i=1}^e n_i ,$$

которое может быть найдено по следующей формуле:

$$C' = \sum_{i=1}^e C_i \cdot N_i \cdot N_i^{-1} \bmod N , \quad (20)$$

где $N = \prod_{i=1}^e n_i$, $N_i = N/n_i$ и $N_i^{-1} = \frac{1}{N_i} \bmod n_i$.

Известно, что сообщение $M < n_i$ для $i = 1, 2, \dots, k$, поэтому $M^e < \prod_{i=1}^e n_i$, а это значит, что исходное сообщение M может быть вычислено любым злоумышленником, который знает только криптограммы и не знает факторизацию модуля n или секретную экспоненту d , путем нахождения корня степени e по решению (20) системы (19):

$$M = \sqrt[e]{C'} .$$

Реализуя на практике данную атаку, для нахождения корня степени e можно использовать итерационный метод Ньютона [9], представляющий из себя последовательность приближений и обладающий квадратичной скоростью сходимости [10]. Приближение корня на каждом следующем шаге a_{i+1} вычисляется с использованием приближения на предыдущем шаге a_i по формуле:

$$a_{i+1} = \frac{1}{e} \left(\frac{C'}{a_i^{e-1}} + (e-1)a_i \right), \quad (21)$$

где e – показатель корня, C' – число, из которого извлекается корень [9].

Как сказано в [11], с целью уменьшения числа итераций при вычислении корней с использованием метода Ньютона в качестве начальной оценки корня удобно выбрать:

$$a_1 = 2^{\text{BitLength}[C']/e}, \quad (22)$$

где $\text{BitLength}[]$ – битовая длина числа.

Поскольку при выполнении атаки на малую открытую экспоненту заранее известно, что корень, который необходимо найти, является целым числом, результат вычисления на каждой итерации (21) округляется до ближайшего целого. В итоге, так как метод является сходящимся, на определенном этапе итерации полученное значение a_{i+1} совпадет со значением a_i , полученным на предыдущем этапе, что и будет являться целочисленным корнем степени e из C' .

Метод Ньютона

Метод Ньютона – это итерационный численный метод для нахождения корней уравнения, основанный на принципах метода простой итерации [10].

В свою очередь идея метода простой итерации состоит в том, чтобы уравнение вида:

$$f(x) = 0$$

привести к эквивалентному уравнению

$$x = \varphi(x)$$

так, чтобы отображение $\varphi(x)$ было сжимающим (сжимающее отображение - это отображение метрического пространства в себя, уменьшающее расстояние между любыми двумя точками не менее чем в 1 раз). Если это удастся, то последовательность итераций $x_{i+1} = \varphi(x_i)$ сходится.

Такое преобразование можно делать разными способами. В частности, сохраняет корни уравнение вида:

$$x = x - \lambda(x)f(x)$$

если $\lambda(x) \neq 0$ на исследуемом отрезке. В качестве $\lambda(x)$ можно выбрать:

$$\lambda(x) = \frac{1}{f'(x)},$$

что превращает метод простых итераций в «метод Ньютона». С учетом этого функция $\varphi(x)$ определяется как:

$$\varphi(x) = x - \frac{f(x)}{f'(x)}$$

При условии, что функция $f(x)$ дважды непрерывно-дифференцируемая в своей области определения и ее производная нигде не обращается в нуль, функция $\varphi(x)$ в окрестности корня осуществляет сжимающее отображение.

Арифметическим корнем n -ой степени $\sqrt[n]{A}$ положительного действительного числа A называется положительное действительное решение уравнения $x^n = A$. Тогда задача нахождения корня n -ной степени может быть рассмотрена, как задача нахождения решения уравнения:

$$x^n - A = 0 \quad (23)$$

Производная этой функции равна $f'(x) = nx^{n-1}$. Тогда итерационное правило будет иметь вид:

$$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)} = x_k - \frac{x_k^n - A}{nx_k^{n-1}} = x_k + \frac{1}{n} \left[\frac{A}{x_k^{n-1}} - x_k \right] = \\ &= \boxed{\frac{1}{n} \left[(n-1)x_k + \frac{A}{x_k^{n-1}} \right]}, \end{aligned}$$

что совпадает с (21).

Геометрически итерационный процесс метода Ньютона означает замену k -той итерации графика функции $y=f(x)$ на касательную к этой функции в точке $(x_k, f(x_k))$ (рис. 2.1). В связи с этим метод иногда называют методом касательных. Уравнение касательной имеет вид

$$y = f'(x_k)(x - x_k) + f(x_k)$$

Найдем точку пересечения с осью Ox этой касательной, что соответствует нахождению решения линейного уравнения:

$$f'(x_k)(x - x_k) + f(x_k) = 0$$

вместо нелинейного уравнения (23).

Выражая x , получаем: $x = x_k - f(x_k)/f'(x_k) \equiv x_{k+1}$

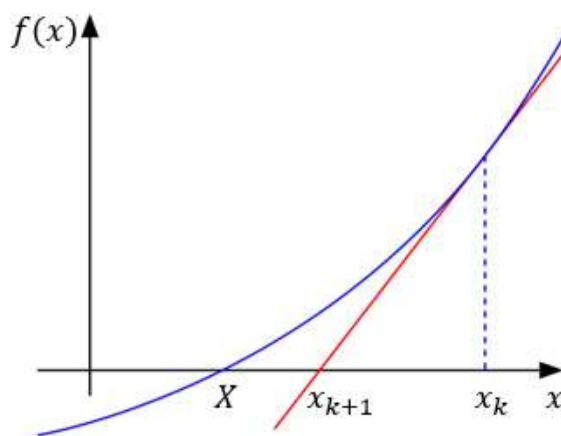


Рис. 2.1 Иллюстрация метода Ньютона

Теперь покажем, как итерационная формула может быть получена аналитически. Рассмотрим уравнение $f(x) = 0$, где X – его корень, x_k – k -ое приближение к корню. Тогда по теореме Лагранжа о средних значениях имеем:

$$0 = f(X) = f(x_k) + (X - x_k)f'(c_k),$$

где $c_k \in (X, x_k)$. Заменяя $f'(c_k)$ на значение $f'(x_k)$ (то есть используя предыдущее приближение к корню), приходим к приближенному равенству

$$0 \approx f(x_k) + (X - x_k)f'(x_k).$$

Откуда получаем $X \approx x_k - f(x_k)/f'(x_k) \equiv x_{k+1}$.

Численный пример выполнения атаки

Рассмотрим простой пример. Допустим, выбрана одинаковая малая шифрующая экспонента $e_1 = e_2 = e_3 = 3$.

Модули криптосистемы равны:

$$\begin{cases} n_1 = 3 \cdot 29 = 87 \\ n_2 = 5 \cdot 23 = 115 \\ n_3 = 11 \cdot 17 = 187 \end{cases}$$

Допустим, легитимный пользователь зашифровывает одинаковое сообщение $M=83$ и отправляет три различные криптограммы трем пользователям:

$$\begin{cases} C_1 = 83^3 \bmod(87) = 23 \\ C_2 = 83^3 \bmod(115) = 7 \\ C_3 = 83^3 \bmod(187) = 128 \end{cases}$$

Если злоумышленник перехватил эти криптограммы, то для вычисления исходного сообщения ему необходимо решить систему уравнений, используя китайскую теорему об остатках (20), а затем вычислить кубический корень из решения этой системы.

Используя перехваченные криптограммы и пару чисел (e, n) открытого ключа, злоумышленник составляет систему уравнений:

$$\begin{cases} 23 = x^3 \bmod(87) \\ 7 = x^3 \bmod(115) \\ 128 = x^3 \bmod(187) \end{cases} \quad (24)$$

Затем вычисляет $N = \prod n_i = 1870935$ и $N_i = \frac{N}{n_i}$:

$$\begin{cases} N_1 = 115 \cdot 187 = 21505 \\ N_2 = 87 \cdot 187 = 16269 \\ N_3 = 87 \cdot 115 = 10005 \end{cases}$$

и соответствующие обратные элементы:

$$\begin{cases} N_1^{-1} = 21505^{-1} \bmod 87 = 49 \\ N_2^{-1} = 16269^{-1} \bmod 115 = 49 \\ N_3^{-1} = 10005^{-1} \bmod 187 = 2 \end{cases}$$

Решение системы (24) вычисляется, как:

$$C' = (23 \cdot 49 \cdot 21505 + 7 \cdot 49 \cdot 16269 + 128 \cdot 2 \cdot 10005) \bmod(1870935) = 571787$$

Теперь найдем корень третьей степени из 571787, используя метод Ньютона. В качестве начального приближения воспользуемся формулой (22). Вычислим начальное приближение по формуле (22): $a_1 = 2^{20/3} \approx 102$, где 20 – битовая длина числа 571787, а 3 – показатель корня.

$$\begin{cases} a_2 = \frac{1}{3} \left(\frac{571787}{102^2} + 2 \cdot 102 \right) \approx 86 \\ a_3 = \frac{1}{3} \left(\frac{571787}{86^2} + 2 \cdot 86 \right) \approx 83 \\ a_4 = \frac{1}{3} \left(\frac{571787}{83^2} + 2 \cdot 83 \right) = 83 \end{cases} \quad (25)$$

Таким образом из (25) видно, что на четвертом шаге итерации полученное значение совпадает с округленным значением на третьем шаге, а значит число 83 действительно является целочисленным кубическим корнем из числа 571787 и соответствует исходному сообщению $M=83$.

Выбор малой экспоненты шифрования e вообще говоря, допустим, но для защиты от соответствующей атаки необходимо «подсаливать» сообщения [1]. Эффективным также является использование экспоненты $e=2^{16}+1=65537$, что требует 16 возведений в квадрат и одного умножения. Данный метод даже без «подсаливания» сообщений имеет преимущество перед выбором экспоненты $e=3$, поскольку используемая для малых экспонент атака будет эффективной, если только одно и то же сообщение шифруется и посылается одновременно 65537 пользователям, что маловероятно на практике.

2.2 Атака при малом объеме возможных сообщений

Атака при переборном числе сообщений довольно тривиальна и является возможной, если передаваемые сообщения не «подсаливаются».

Допустим, имеется набор сообщений M_1, M_2, \dots, M_r известных злоумышленнику при обозримом числе сообщений r (Это могут быть, например, различные команды – вперед, назад, влево, вправо и т.п.). Тогда перехваченная криптограмма может быть легко расшифрована.

Действительно, пусть злоумышленник перехватил криптограмму C . Для принятия решения о том, какое сообщение было передано, злоумышленник последовательно зашифровывает все варианты сообщений до появления совпадения с перехваченной криптограммой.

$$\left. \begin{array}{l} C_1 = M_1^e \bmod n \\ C_2 = M_2^e \bmod n \\ \vdots \\ C_r = M_r^e \bmod n \end{array} \right\} \stackrel{?}{=} C$$

Способ борьбы с такой атакой – это «подсаливание» сообщений, т.е. присоединение к ним небольших псевдослучайных цепочек бит [1].

2.3 Атака на малую секретную экспоненту

Атака на криптосистему RSA с малой величиной секретной экспоненты, называемая также атакой Винера [12], демонстрирует опасность, которая может возникнуть при проектировании системы с малой экспонентой дешифрования. Такой ситуацией, при которой использование малых экспонент является выгодным, может являться взаимодействие двух устройств с разницей в их вычислительной мощности. Примером является криптосистема RSA, используемая во взаимодействии между смарт-картой и компьютером. В этом случае было бы предпочтительным использование малой секретной экспоненты для смарт-карты и малой открытой экспоненты для компьютера, чтобы уменьшить время вычислений, затрачиваемое смарт-картой.

В качестве математического базиса атака Винера использует элементы теории непрерывных дробей для нахождения в полиномиальное время некой дроби, достаточно близкая оценка которой известна злоумышленнику [12]. Атака Винера демонстрирует, как открытая экспонента e и модуль криптосистемы $n=pq$ могут быть использованы для создания оценки некой дроби, которая содержит в себе секретную экспоненту d . Затем алгоритм, основанный на теории непрерывных дробей, использует известную оценку, составленную из открытого ключа, для нахождения секретной экспоненты d .

2.3.1 Необходимые сведения о непрерывных дробях

Непрерывной дробью является выражение вида:

$$\frac{\cfrac{a_1}{q_1 + \cfrac{a_2}{q_2 + \cfrac{a_3}{\dots + \cfrac{a_m}{q_{m-1} + \cfrac{a_m}{q_m}}}}}}{a_1 / (q_1 + a_2 / (q_2 + a_3 / (\dots / (q_{m-1} + a_m / q_m) \dots)))}$$

В конкретной ситуации нас интересуют непрерывные дроби, которые имеют все $a_i = 1$, где $i = 1, 2, \dots, m$. Для удобства будем обозначать:

$$\langle q_0, q_1, \dots, q_m \rangle = q_0 + 1/(q_1 + 1/(q_2 + 1/(\dots / (q_{m-1} + 1/q_m) \dots))) \quad (26)$$

Например, $\langle 0, 2, 1, 3 \rangle = 0 + 1/(2 + 1/(1 + 1/3)) = \frac{4}{11}$ означает, что $\langle 0, 2, 1, 3 \rangle$

является разложением в непрерывную дробь для рационального числа $4/11$.

i -ой подходящей дробью для непрерывной дроби $\langle q_0, q_1, \dots, q_m \rangle$ называется конечная непрерывная дробь $\langle q_0, q_1, \dots, q_i \rangle$, значение которой равно некоторому рациональному числу $\frac{n_i}{d_i}$, где $i = 0, 1, \dots, m$.

Разложение в непрерывную дробь для положительного рационального числа f выполняется путем вычитания целой части числа f , затем переворачивания оставшейся дробной части и вновь вычитания целой части до тех пор, пока дробная часть не будет равна 0 [12]. Обозначим q_i - целое частное и r_i - дробная часть на шаге i , и обозначим через m последний шаг, на котором дробная часть оказалась равна нулю. Тогда:

$$\begin{aligned} q_0 &= \lfloor f \rfloor, & r_0 &= f - q_0, \\ q_i &= \left\lfloor \frac{1}{r_{i-1}} \right\rfloor, & r_i &= \frac{1}{r_{i-1}} - q_i \quad \text{для } i = 1, 2, \dots \end{aligned} \quad (27)$$

Поскольку рациональные числа всегда могут быть представлены в виде конечной непрерывной дроби, то на определенном шаге r_m обязательно будет равно нулю, а разложение в непрерывную дробь рационального числа f будет выглядеть, как $f = \langle q_0, q_1, \dots, q_m \rangle$.

На данном этапе могут быть сделаны два наблюдения, которые будут полезны в будущем. Первое – $q_m \geq 2$ (это действительно так, поскольку $q_m = 1$ означало бы, что $r_{m-1} = 1$, а это невозможно). Второе – для любого $x > 0$ справедливо:

$$\begin{aligned} \langle q_0, q_1, \dots, q_m \rangle &< \langle q_0, q_1, \dots, q_{m-1}, q_m + x \rangle \text{ если } m \text{ четное} \\ \langle q_0, q_1, \dots, q_m \rangle &> \langle q_0, q_1, \dots, q_{m-1}, q_m + x \rangle \text{ если } m \text{ нечетное} \end{aligned} \quad (28)$$

В этом можно убедиться, глядя на число уровней дроби, вложенных в (26).

Теперь рассмотрим, как можно восстановить рациональное число f из разложения в непрерывную дробь. Используя формулу (26), можно восстановить число f , начав с q_m , суммируя и переворачивая полученную дробь на каждом шаге вплоть до q_0 . Однако, может оказаться более выгодным восстановление исходной дроби в прямом порядке, начиная с q_0 . Такой подход будет более удобным, к примеру, для последовательного восстановления рациональных чисел из соответствующих подходящих дробей, начиная с q_0 . Обозначим числитель и знаменатель рационального числа, как n_i и d_i соответственно. Тогда:

$$\frac{n_i}{d_i} = \langle q_0, q_1, \dots, q_i \rangle \text{ и } \gcd(n_i, d_i) = 1 \text{ для } i = 0, 1, \dots, m \quad [12] \quad (29)$$

Числитель и знаменатель рационального числа вычисляются по следующим формулам [12]:

$$\begin{aligned} n_0 &= q_0, & d_0 &= 1, \\ n_1 &= q_0 q_1 + 1, & d_1 &= q_1, \\ n_i &= q_i n_{i-1} + n_{i-2}, & d_i &= q_i d_{i-1} + d_{i-2} \text{ для } i = 2, 3, \dots \end{aligned} \quad (30)$$

В таком случае рациональное число f может быть восстановлено в прямом порядке, начиная с q_0 , как $f = \frac{n_m}{d_m}$.

Существует взаимосвязь между числителем и знаменателем, которая окажется полезной в дальнейшем [12]:

$$n_i d_{i-1} - n_{i-1} d_i = -(-1)^i \text{ для } i = 2, 3, \dots \quad (31)$$

Алгоритм нахождения числителя и знаменателя некой дроби, для которой известно достаточно хорошее приближенное значение, будем называть «Алгоритмом непрерывных дробей».

2.3.2 «Алгоритм непрерывных дробей»

Обозначим через f' «недооценку» рационального числа f :

$$f' = f(1 - \delta) \quad \text{для некоторого } \delta \geq 0, \delta \ll 1 \quad (32)$$

Обозначим через q_i , r_i и q'_i , r'_i i -ые частные и дробные части чисел f и f' соответственно. Если δ достаточно мало, то числитель и знаменатель числа f может быть найден с использованием следующего алгоритма (повторяется до тех пор, пока f не найдено) [12]:

- Вычислить следующее целое частное (q'_i) для разложения в непрерывную дробь числа f'
- Использовать формулы (30) для восстановления i -ого рационального числа из подходящей дроби:
 $\langle q'_0, q'_1, \dots, q'_{i-1}, q'_i + 1 \rangle$ если i четное,
 $\langle q'_0, q'_1, \dots, q'_{i-1}, q'_i \rangle$ если i нечетное.
- Проверить, не является ли восстановленное на предыдущем шаге рациональное число интересующим нас числом f .

Причиной добавления единицы к последнему q'_i на четном шаге является тот факт, что строящееся предположение f должно быть больше, чем f' , так как по условию f' является «недооценкой» числа f , однако из (28) видно, что на четном шаге $\langle q'_0, q'_1, \dots, q'_{i-1}, q'_i \rangle$ меньше, чем $f' = \langle q'_0, q'_1, \dots, q'_{i-1}, q'_i + r'_i \rangle$.

Заметим, что для выполнения алгоритма должен существовать тест для проверки, является ли предположение о числе f верным.

Алгоритм непрерывных дробей приводит к решению [12], если:

$$\begin{aligned} \langle q_0, q_1, \dots, q_{m-1}, q_m - 1 \rangle < f' \leq \langle q_0, q_1, \dots, q_m \rangle & \text{ при четных } m \\ \langle q_0, q_1, \dots, q_{m-1}, q_m + 1 \rangle < f' \leq \langle q_0, q_1, \dots, q_m \rangle & \text{ при нечетных } m \end{aligned} \quad (33)$$

Теперь рассмотрим влияние условия (33) на максимально возможную величину δ . Решение уравнения (32) относительно δ дает:

$$\delta = 1 - \frac{f'}{f} \quad (34)$$

Ниже будет выполнен отдельный анализ для следующих случаев: $m=0$, $m=1$, четное $m \geq 2$ и нечетное $m \geq 3$.

Случай 1: $m=0$

Использование (33) для замены f' в (34) дает:

$$\delta < 1 - \langle q_0 - 1 \rangle / \langle q_0 \rangle$$

С учетом (26) предыдущее неравенство упрощается до $\delta < 1/q_0$, что в свою очередь может быть переписано следующим образом (вспомним, что $n_0 = q_0$ и $d_0 = 1$):

$$\delta < \frac{1}{n_0 d_0}$$

Случай 2: $m=1$

Использование (33) для замены f' в (34) дает:

$$\delta < 1 - \langle q_0, q_1 + 1 \rangle / \langle q_0, q_1 \rangle$$

С учетом (26) предыдущее неравенство упрощается до:

$$\delta < \frac{1}{(q_0 q_1 + 1)(q_1 + 1)} \quad (35)$$

Ранее было замечено, что $q_m \geq 2$. Для данного случая это означает, что $(3/2)q_1 \geq q_1 + 1$. Объединяя это с (35) и выражениями для n_1 и d_1 в (30) получим:

$$\delta < \frac{1}{\frac{3}{2}n_1 d_1}$$

что является достаточным условием для гарантированной работы алгоритма непрерывных дробей.

Случай 3: четное $m \geq 2$

Использование (33) для замены f' в (34) дает:

$$\delta < 1 - \langle q_0, q_1, \dots, q_{m-1}, q_m - 1 \rangle / \langle q_0, q_1, \dots, q_m \rangle \quad (36)$$

С учетом (30) имеем:

$$\langle q_0, q_1, \dots, q_{m-1}, q_m - 1 \rangle = \frac{(q_m - 1)n_{m-1} + n_{m-2}}{(q_m - 1)d_{m-1} + d_{m-2}} \text{ и}$$

$$\langle q_0, q_1, \dots, q_m \rangle = \frac{q_m n_{m-1} + n_{m-2}}{q_m d_{m-1} + d_{m-2}}$$

Подстановка этих выражений в (36) дает:

$$\delta < \frac{n_{m-1}d_{m-2} - n_{m-2}d_{m-1}}{(q_m n_{m-1} + n_{m-2})(q_m d_{m-1} + d_{m-2} - d_{m-1})}$$

С учетом (31) и выражениями для n_m и d_m в (30) получаем неравенство:

$$\delta < \frac{1}{n_m(d_m - d_{m-1})}$$

Таким образом:

$$\delta < \frac{1}{n_m d_m}$$

является достаточным для выполнения алгоритма непрерывных дробей.

Случай 4: нечетное $m \geq 3$

Выполняя похожий анализ, что и в случае 3, получаем:

$$\delta < \frac{1}{n_m(d_m + d_{m-1})}$$

Поскольку $d_m = q_m d_{m-1} + d_{m-2}$ и $q_m \geq 2$, имеем $d_m + d_{m-1} \leq (3/2)d_m$. Таким образом,

$$\delta < \frac{1}{\frac{3}{2}n_m d_m}$$

является достаточным условием для выполнения алгоритма непрерывных дробей.

Принимая во внимание результаты анализа для всех четырех случаев, получаем неравенство:

$$\delta < \frac{1}{\frac{3}{2}n_md_m} \quad (37)$$

что является достаточным условием для выполнения алгоритма непрерывных дробей. Напомним, что n_m и d_m являются числителем и знаменателем рационального числа f , вычисляемые по формулам (30).

2.3.3 Применение «алгоритма непрерывных дробей» для нахождения малой секретной экспоненты в РША.

Теперь рассмотрим, как полученные ранее сведения о непрерывных дробях и рассмотренный алгоритм непрерывных дробей может быть применен для криптоанализа РША. Из описания работы криптосистемы РША известно, что между открытым и секретным ключом существует следующее соотношение:

$$ed \equiv 1 \pmod{\text{lcm}(p-1, q-1)}$$

Из этого следует, что существует такое K , что:

$$ed = K \cdot \text{lcm}(p-1, q-1) + 1 \quad (38)$$

Если обозначить $G = \text{gcd}(p-1, q-1)$, то по формуле (18):

$$\text{lcm}(p-1, q-1) = (p-1)(q-1)/G,$$

и тогда:

$$ed = \frac{K}{G}(p-1)(q-1) + 1$$

Возможно также, что K и G имеют общие множители. Тогда обозначим $k = K / \text{gcd}(K, G)$ и $g = G / \text{gcd}(K, G)$, а значит $k/g = K/G$ и при этом $\text{gcd}(k, g) = 1$. Теперь имеем:

$$ed = \frac{k}{g}(p-1)(q-1) + 1 \quad (39)$$

Деление правой и левой части равенства (39) на $d pq$ дает:

$$\frac{e}{pq} = \frac{k}{dg} (1 - \delta), \text{ где } \delta = \frac{p + q - 1 - \frac{g}{k}}{pq} \quad (40)$$

Заметим, что соотношение e/pq состоит полностью из известной информации об открытом ключе и является близкой оценкой соотношения k/dg при выполнении условия (37) для δ . Перед тем, как применить алгоритм непрерывных дробей, нужно помнить, что он находит несократимую дробь, то есть числитель и знаменатель полученной дроби не будет иметь общих множителей, что показано в (29). Из (38) мы видим, что $\gcd(K, d) = 1$. Поскольку k делит K , справедливо утверждение $\gcd(k, d) = 1$. Также известно, что $\gcd(k, g) = 1$ по определению. Таким образом выходит, что $\gcd(k, dg) = 1$ и алгоритм непрерывных дробей может быть успешно использован для нахождения несократимой дроби $\frac{k}{dg}$ при условии, что δ достаточно мало (37).

Используя выражение для δ из (40) и ограничение, наложенное на δ в (37), можно показать, что:

$$kdg < \frac{pq}{\frac{3}{2}(p+q)} \quad (41)$$

является достаточным для выполнения алгоритма непрерывных дробей. Заметим, что $(-1 - g/k)$ в выражении (41) было упущено, поскольку является малым в сравнении с $(p+q)$. Тем не менее это не влияет на верность утверждения (41), так как $(-1 - g/k)$ способствует лишь уменьшению δ .

Напомним, что для выполнения атаки необходимо реализовать алгоритм проверки. Рассмотрим, как можно проверить, действительно ли полученное на определенном шаге рациональное число является дробью $\frac{k}{dg}$. В целях упрощения проверки будем считать, что $ed > pq$. Это предположение не является значительно ограничивающим возможность проверки, потому что

когда либо e , либо d заранее задано, ожидаемое значение обратной экспоненты приблизительно pq/G [12] (напомним, что $G = \gcd(p-1, q-1)$), и до тех пор, пока G выбрано большим, весьма вероятно, что $ed > pq$. Из выражения (39) следствием того, что $ed > pq$, является $k > g$. Переписав выражение (39), как:

$$edg = k(p-1)(q-1) + g \quad (42)$$

нетрудно увидеть, что деление edg на k дает целое частное

$$(p-1)(q-1) = \lfloor edg / k \rfloor \quad (43)$$

и остаток

$$g = edg \bmod k, \quad (44)$$

так как $k > g$. Таким образом мы можем найти предположения о $(p-1)(q-1)$ и о g . Если вычисленное на определенном шаге предположительное $(p-1)(q-1)$ равно нулю, то предположительные k и dg неверны. Данный случай должен быть отброшен, поскольку это предполагает, что сомножителями модуля криптосистемы являются тривиально $p \cdot q$ и 1, чего быть не может. Предположение о $(p-1)(q-1)$ может быть использовано для вычисления предположительного $\frac{p+q}{2}$, используя следующее равенство [12]:

$$\frac{pq - (p-1)(q-1) + 1}{2} = \frac{p+q}{2} \quad (45)$$

Если предположение о $(p+q)/2$ не является целым числом, то предположительные k и dg не верны. Предположение о $\frac{p+q}{2}$ может быть использовано далее для вычисления предположительного $\left(\frac{p-q}{2}\right)^2$ используя следующее равенство [12]:

$$\left(\frac{p+q}{2}\right)^2 - pq = \left(\frac{p-q}{2}\right)^2 \quad (46)$$

Если предположительное $((p-q)/2)^2$ является квадратом целого числа, то предположительные k и dg верны. Тогда секретная экспонента d вычисляется делением dg на g (напомним, что из (42) следует, что g - остаток от деления edg на k).

Обозначим получаемые предположения из (45) и (46), как:

$$\begin{cases} \frac{p+q}{2} = a \\ \frac{p-q}{2} = b \end{cases} \quad (47)$$

Тогда при условии, что предположительное отношение $\frac{k}{dg}$ на текущем шаге является верным, можно получить факторизацию модуля криптосистемы, решая систему с двумя неизвестными (47):

$$\begin{cases} p = a + b \\ q = a - b \end{cases}$$

Если не предпринимать меры против данной атаки на РША, то ожидаемо, что g мало и $k < dg$. Из (41) видно, что при выполнении этих условий, секретная экспонента d битовой длины приблизительно до четверти битовой длины модуля может быть найдена в полиномиальное время [12].

Данная атака не может быть обобщена на обычный случай, когда секретная экспонента имеет приблизительно ту же битовую длину, что и модуль криптосистемы. Это происходит потому, что данная атака опирается на открытую экспоненту, содержащую в себе информацию, которая помогает факторизовать модуль [12]. В обычном же случае открытая экспонента может быть выбрана практически независимо от модуля.

2.3.4 Пример выполнения атаки на малую секретную экспоненту

Перед тем, как непосредственно привести пример, запишем последовательность необходимых действий:

Шаг 1. Записать открытый ключ в виде отношения $\frac{e}{n}$ и обозначить $i=0$.

Шаг 2. Вычислить q'_i и r'_i для рационального числа $\frac{e}{n}$, используя формулы (27).

Шаг 3. Вычислить предположение о $\frac{k}{dg}$ по формулам (30) из разложения:

$$\langle q'_0, q'_1, \dots, q'_{i-1}, q'_i + 1 \rangle \text{ если } i \text{ четное,}$$

$$\langle q'_0, q'_1, \dots, q'_{i-1}, q'_i \rangle \text{ если } i \text{ нечетное.}$$

Шаг 4. Вычислить предположение о edg , умножив известную открытую экспоненту e на предположения о dg из шага 3.

Шаг 5. Вычислить предположение о $(p-1)(q-1)$, как целую часть от деления edg на k (43), и предположение о g , как остаток от деления edg на k (44). Если предположение $(p-1)(q-1) = 0$, вернуться на шаг 2 и произвести вычисления для следующего i .

Шаг 6. Вычислить предположение о $\frac{p+q}{2}$, воспользовавшись формулой (45) и произвести проверку: является ли $\frac{p+q}{2}$ целым числом. Если нет, то предположение о $\frac{k}{dg}$ не является верным, следует вернуться на шаг 2 и произвести вычисления для следующего i .

Шаг 7. Вычислить предположение о $\left(\frac{p-q}{2}\right)^2$, воспользовавшись формулой (40) и произвести проверку: является ли $\left(\frac{p-q}{2}\right)^2$ квадратом целого числа. Если нет, то предположение о $\frac{k}{dg}$ не является верным, следует вернуться на шаг 2 и произвести вычисления для следующего i .

Шаг 8. Вычислить d путем деления знаменателя рационального числа $\frac{k}{dg}$ на g . Также, если необходимо, получаем факторизацию модуля криптосистемы, решая систему уравнения с двумя неизвестными (47).

Заметим также, что выполнение атаки прекращается на шаге 2 при некоем i , если $r'_{i-1} = 0$. Это означает, что данная атака не может быть применена при заданных условиях.

Приведем численный пример. Допустим, имеем секретную экспоненту $e=2621$ и модуль криптосистемы $n=8927$. Теперь покажем, как найти секретную экспоненту d , применяя вышеописанные шаги к рациональному числу $\frac{e}{n} = \frac{2621}{8927}$. Результаты представим в виде таблицы (табл. 2.1).

Таблица 2.1

Вычисляемая величина	Этап выполнения	$i = 0$	$i = 1$	$i = 2$
q'_i	Шаг 2	0	3	2
r'_i	Шаг 2	$\frac{2621}{8927}$	$\frac{1064}{2621}$	$\frac{493}{1064}$
предположение о $\frac{k}{dg}$	Шаг 3	$\frac{1}{1}$	$\frac{1}{3}$	$\frac{3}{10}$
предположение о edg	Шаг 4	2621	7863	26210
предположение о $(p-1)(q-1)$	Шаг 5	2621	7863	8736
предположение о g	Шаг 5	0	0	
предположение о $\frac{p+q}{2}$	Шаг 6	3153.5 (нечетное, возврат к шагу 2)	532.5 (нечетное, возврат к шагу 2)	96 (четное)
предположение о $\left(\frac{p-q}{2}\right)^2$	Шаг 7			289=17 ² (289 является квадратом целого числа)
секретная экспонента d	Шаг 8			5

Таким образом, при $i=3$ вычисленное предположение о $\frac{k}{dg}$ оказалось верным, а значит все последующие предположения тоже являются верными. Из знаменателя рационального числа $\frac{k}{dg}$ на шаге 8 была получена секретная экспонента $d=5$. Используя $\frac{p+q}{2}=96$ и $\frac{p-q}{2}=17$ дополнительно получаем простые сомножители модуля криптосистемы $p=96+17=113$ $q=96-17=79$.

2.3.5 Способы борьбы с атакой на малую секретную экспоненту

Существует 2 способа уменьшения максимального размера секретной экспоненты, которая может быть гарантированно найдена при помощи алгоритма непрерывных дробей. Из выражения (41) можно видеть, что противостоять атаке можно, увеличивая k и g .

Для увеличения k необходимо увеличивать открытую экспоненту e (см. равенство (39)). Этого можно достичь прибавлением $\text{lcm}(p-1, q-1)$ к экспоненте e [12].

Допустим, $e > (pq)^{1.5}$. Это влечет за собой $k/dg > (pq)^{0.5}$ (см. равенство (40)). Замена $k = dg(pq)^{0.5}$ в (41) приводит к $d < 1$. Таким образом, если $e > (pq)^{1.5}$, успешное выполнение алгоритма непрерывных дробей при любой длине секретной экспоненты d не гарантировано. Увеличение длины открытой экспоненты e имеет свой недостаток - это увеличение времени выполнения операции шифрования на открытом ключе, хотя такая ситуация может быть приемлема при проектировании некоторых систем [12].

В свою очередь, как следует из определения $g = \frac{\text{gcd}(p-1, q-1)}{\text{gcd}(K, G)}$, а значит для увеличения значения числа g сомножители p и q должны быть выбраны такими, чтобы $\text{gcd}(p-1, q-1)$ было большим.

2.4 Атака с использованием мультипликативного свойства шифра РША

Атака с использованием мультипликативного свойства шифра РША позволяет расшифровать криптограмму при условии, что легитимный пользователь криптосистемы согласен расшифровать любую другую криптограмму кроме той, что интересует злоумышленника.

Для любых сообщений M_1 и M_2 справедливо следующее [1]:

$$(M_1 \cdot M_2)^e = M_1^e \cdot M_2^e = C_1 \cdot C_2 \bmod n,$$

где $C_1 = M_1^e \bmod n$, $C_2 = M_2^e \bmod n$. Это свойство может использовать злоумышленник.

Предположим, что злоумышленник хочет дешифровать криптограмму C , предназначенную для легитимного пользователя, и предположим, что этот пользователь согласен дешифровать любую другую криптограмму для злоумышленника, кроме криптограммы C . Тогда для расшифровки криптограммы C злоумышленник выбирает случайное число x такое, что $\gcd(x, n) = 1$. Затем вычисляет

$$C' = C \cdot x^e \bmod n \quad (48)$$

и просит легитимного пользователя расшифровать данную криптограмму. В результате расшифровки получается бессмысленное на первый взгляд сообщение, но содержащее в себе интересующее злоумышленника исходное сообщение M :

$$M' = C'^d \bmod n = C^d \cdot (x^e)^d \bmod n = M \cdot x \bmod n \quad (49)$$

Теперь, зная x и M' (49) для получения M достаточно вычислить:

$$M = M' \cdot x^{-1} \bmod n \quad (50)$$

Рассмотрим простой пример. Допустим, злоумышленник хочет расшифровать криптограмму $C = 354$. Открытый ключ, с помощью которого была вычислена данная криптограмма, $(e, n) = (293, 361)$. Тогда злоумышленник выбирает любое целое число x (например, $x = 283$), которое

является взаимно простым с модулем криптосистемы n , а значит существует обратный элемент от x по модулю n :

$$x^{-1} \bmod n = 283^{-1} \bmod 361 = 199.$$

Далее злоумышленник вычисляет C' по формуле (48):

$$C' = 354 \cdot 283^{293} \bmod 361 = 53$$

и просит легитимного пользователя криптосистемы расшифровать $C' = 53$. При условии, что легитимный пользователь согласился расшифровать C' , используя свой закрытый ключ $(d, n) = (209, 361)$, злоумышленник получает некое сообщение $M' = 307$ и вычисляет M по формуле (50):

$$M = 307 * 199 \bmod 361 = 84$$

Используя открытый ключ, нетрудно убедиться в том, что сообщению $M=84$ действительно соответствует криптограмма $C=354$.

Абсолютно аналогичная атака находит применение при использовании цифровой подписи на основе криптосистемы RSA, когда злоумышленник просит легитимного пользователя подписать на первый взгляд бессмысленное сообщение, а в результате получает подпись на необходимое сообщение, воспользовавшись мультипликативным свойством алгоритма RSA [8].

2.5 Циклическая атака

Предположим, что злоумышленнику известна криптограмма C . Циклическая атака [1] на RSA представляет собой повторное выполнение операции шифрования над криптограммой:

$$C_1 = C^{e^1} \bmod n; C_2 = C^{e^2} \bmod n; C_3 = C^{e^3} \bmod n$$

Повторное шифрование выполняется до тех пор, пока результат не совпадет с исходной криптограммой: $C^{e^k} = C$. Тогда $C^{e^{k-1}}$ и будет являться исходным сообщением. Данное событие рано или поздно произойдет на каком-то шаге k , поскольку шифрование – это по существу перестановка чисел $\{0, 1, 2, \dots, n-1\}$. Обобщенная циклическая атака приводит также к факторизации модуля n , поэтому при больших n данный подход не лучше прямого метода факторизации модуля криптосистемы RSA [1].

2.6 Атака на общие модули

Возможна ситуация, при которой несколько или все пользователи сети имеют в составе своего ключа одинаковый общий модуль n , но различные экспоненты e и d . Такое встречается при генерировании пар ключей пользователей некоторым общим «центром распределения ключей» [1]. Однако, существует вероятностный алгоритм, воспользовавшись которым любой пользователь i , имеющий в составе своих ключей экспоненты e_i и d_i , способен факторизовать модуль n_i [13]. Тогда, если пользователь j имеет такой же модуль, то пользователь i с легкостью вычислит секретную экспоненту d_j , используя соотношение (1).

Покажем теперь, как знание обеих экспонент позволяет выполнить факторизацию модуля криптосистемы РША. Рассмотрим уравнение:

$$x^2 \equiv y^2 \pmod{n}, \quad (51)$$

где $n = pq$. Так как число n является составным, уравнение (51) имеет четыре решения, два из которых могут быть использованы для нахождения сомножителей числа n [7]. Двумя тривиальными решениями называются $x \equiv \pm y \pmod{n}$, а двумя нетривиальными $x \not\equiv \pm y \pmod{n}$. Покажем, почему это так. Рассмотрим равенство:

$$x^2 \equiv y^2 \pmod{p}, \quad (52)$$

где p – простое число. Данное уравнение имеет всего 2 решения $x \equiv \pm y \pmod{p}$.

Аналогичным образом уравнение:

$$x^2 \equiv y^2 \pmod{q}, \quad (53)$$

где q – простое число, имеет также два решения $x \equiv \pm y \pmod{q}$. С учетом того, что $n = pq$, по китайской теореме об остатках соотношение (51) будет верным только тогда, когда верны (52) и (53). Таким образом, уравнение (51) будет иметь 4 решения, которые могут быть найдены путем комбинирования решений равенств (52) и (53) четырьмя различными способами:

$$\begin{cases} x \equiv y \pmod{p} \\ x \equiv y \pmod{q} \end{cases} \quad (54)$$

$$\begin{cases} x \equiv -y \pmod{p} \\ x \equiv -y \pmod{q} \end{cases} \quad (55)$$

$$\begin{cases} x \equiv y \pmod{p} \\ x \equiv -y \pmod{q} \end{cases} \quad (56)$$

$$\begin{cases} x \equiv -y \pmod{p} \\ x \equiv y \pmod{q} \end{cases} \quad (57)$$

Комбинации решений (54) и (55) приводят к тривиальному решению уравнения (51) $x \equiv \pm y \pmod{n}$. Две последние комбинации (56) и (57) приводят к нетривиальному решению уравнения (51) $x \not\equiv \pm y \pmod{n}$. Из (51) видно, что $pq \mid (x+y)(x-y)$, но если был найден нетривиальный корень, то из (56) и (57) видно, что $(x+y)$ или $(x-y)$ не могут одновременно делиться на p и q . Тогда одно из чисел $(x+y)$ или $(x-y)$ делиться на p , а другое на q и нетривиальные сомножители числа n могут быть найдены, как $\gcd(x \pm y, n)$:

$$n = \gcd(x+y, n) \cdot \gcd(x-y, n) \quad (58)$$

Идея нахождения таких x и y , что $x^2 \equiv y^2 \pmod{n}$, но $x \not\equiv \pm y \pmod{n}$, используется многими известными методами факторизации, и в большинстве случаев разница заключается лишь в способах нахождения x и y [7]. Теперь рассмотрим свойства РША, которые позволяют найти такие x и y , зная обе экспоненты e и d .

Предположим, что мы имеем $x^2 \equiv 1 \pmod{n}$, где n – модуль криптосистемы. Тогда, если $x \not\equiv \pm 1 \pmod{n}$, то простые сомножители находятся аналогично (58) и в данном случае будут равны $\gcd(x \pm 1, n)$. Покажем, как можно найти такое число x . Из теоремы Эйлера известно, что для любого числа a взаимно простого с n справедливо:

$$a^{\varphi(n)} \equiv 1 \pmod{n} \quad (59)$$

С другой стороны, из (1) известно, что открытая и закрытая экспоненты в РША связаны следующим соотношением:

$$ed \equiv 1 \pmod{\varphi(n)}$$

Отсюда следует, что $ed - 1$ кратно φ . Значит можно заменить $\varphi(n)$ на $ed - 1$ в (59):

$$a^{ed-1} \equiv 1 \pmod{n} \quad (60)$$

Также для удобства выполнения факторизации запишем $ed - 1$ в виде:

$$ed - 1 = 2^k \cdot t \quad (61)$$

где k и t являются положительными целыми числами, причем t - нечетное число.

Теперь, если для случайно выбранного числа a взаимно простого с модулем n , существует $0 < k' < k$ такое, что:

$$x^2 = a^{2^{k'} \cdot t} \equiv 1 \pmod{n} \text{ и } x = a^{2^{k'-1} \cdot t} \not\equiv \pm 1 \pmod{n} \quad (62)$$

то x является нетривиальным квадратным корнем из единицы и делители числа n находятся, как $\gcd(x \pm 1, n)$. Данный алгоритм факторизации является вероятностным, потому что приблизительно для половины выбранных a будет найдено такое k' , что выполняется условие (62) [13].

Итак, запишем по порядку все шаги, необходимые для выполнения данного алгоритма:

Шаг 1. Привести $ed - 1$ к виду, показанному в (61).

Шаг 2. Выбрать случайное положительное целое число $a > 1$. Если случайное a окажется не взаимно простым с n , то делитель n вычисляется, как $\gcd(a, n)$ и работа алгоритма завершается.

Шаг 3. Вычислить $a^t \pmod{n}$. Если результат вычисления равен 1 или $n - 1$, вернуться к шагу 2 и выбрать другое число a .

Шаг 4. Последовательно вычислять $a^{2^i \cdot t} \pmod{n}$, где $i=1, 2, 3 \dots k$ до тех пор, пока $a^{2^i \cdot t} \pmod{n} \neq 1$. Если на каком-то шаге $a^{2^i \cdot t} \pmod{n} = n - 1$, вернуться к шагу 2 и выбрать другое a .

Шаг 5. Как только $a^{2^i \cdot t} \bmod n = 1$ на определенном шаге i , вычислить делители модуля n , как $\gcd(a^{2^{i-1} \cdot t}, n)$.

В [13] доказывається, что данный алгоритм приводит к успеху как минимум в половине случаев при случайно выбранном числе a .

Для наглядности приведем небольшой пример. Допустим, имеется модуль криптосистемы $n=2773$, открытая экспонента $e=17$ и секретная экспонента $d=157$.

На первом шаге представим $ed - 1$ в виде произведения нечетного числа и степени двойки, как показано в (61):

$$ed - 1 = 2668 = 2^2 \cdot 667,$$

где нечетное $t=667$ и $k=2$.

Перейдем к шагу 2 и выберем в качестве a число $a=7$. На шаге 3 вычислим:

$$a^t \bmod n = 7^{667} \bmod 2668 = 1$$

Поскольку на данном шаге результат вычисления $a^t \bmod n$ равен единице, необходимо вернуться к шагу 2 для выбора другого a . Возьмем тогда $a=8$. Вычислим:

$$a^t \bmod n = 8^{667} \bmod 2668 = 471$$

Результат не равен 1 или $n-1$, тогда перейдем к шагу 4 и вычислим:

$$a^{2^i \cdot t} \bmod n = 8^{667 \cdot 2} \bmod 2668 = 1,$$

где $i=1$. Тот факт, что $a^{2^i \cdot t} = 1$ и $a^t = 471$, означает, что 471 является нетривиальным квадратным корнем из единицы по модулю составного числа n . Тогда перейдем к шагу 5 и вычислим делители числа n :

$$\gcd(471 - 1, 2773) = 47$$

$$\gcd(471 + 1, 2773) = 59$$

Итак, использование общих модулей в криптосистеме RSA приводит к возможности полного взлома криптосистемы любым из ее участников. Таким образом, использование общих модулей недопустимо.

Выводы по главе

В данной главе были приведены подробные доказательства возможности выполнения следующих побочных атаки на криптосистему РША:

- Атака на малую открытую экспоненту
- Атака при малом объеме возможных сообщений
- Атака на малую секретную экспоненту
- Атака с использованием мультипликативного свойства шифра РША
- Циклическая атака
- Атака на общие модули

Для большинства побочных атак были приведены небольшие численные примеры их выполнения.

В качестве алгоритма нахождения целочисленного корня n -ой степени при выполнении атаки на малую открытую экспоненту был предложен численный метод Ньютона.

Особое внимание в данной главе было уделено атаке Винера, ввиду сложности ее алгоритма. Приведена методика решения задачи нахождения малой секретной экспоненты, используя непрерывные дроби.

Аналогичным образом подробно была рассмотрена атака на общие модули и приведена методика нахождения делителей модуля криптосистемы при наличии обеих экспонент, используя теорему Эйлера, свойства ключей РША и свойства составных чисел.

Материал, изложенный в данной главе, явился основой для разработки программы, демонстрирующей выполнение побочных атак на практике. Процесс разработки программы описан в следующей главе.

Глава 3. Разработка лабораторной работы «Побочные атаки на криптосистему RSA»

3.1 Задачи, решаемые при разработке программного обеспечения

В рамках данной работы был реализован специальный программный комплекс (далее – программа) для наглядной демонстрации выполнения побочных атак на криптосистему RSA. Программа позволяет смоделировать работу криптосистемы и увидеть на практике применение побочных атак, рассмотренных на лекционных занятиях. Интуитивно понятный графический интерфейс позволяет легко понять последовательность выполняемых действий и облегчить работу с программой.

Структурно программа состоит из следующих частей:

- общий управляемый генератор случайных чисел
- классы криптосистем, отличающиеся способами генерирования ключей
- модули для выполнения шести побочных атак

Более подробно о проектировании программы написано в этой главе.

3.2 Инструменты для разработки программного обеспечения

Поскольку асимметричные алгоритмы, в частности RSA, работает медленнее из-за сложности операций и оперирования большими числами, в качестве языка разработки был выбран язык C++. Также желательным было разработать кроссплатформенное программное обеспечение, поэтому для программа написана таким образом, чтобы она могла быть откомпилирована в трех основных операционных системах: Microsoft Windows, Linux и Apple Mac OS X.

Разработка программы велась в ОС Windows, а в качестве среды разработки использовалась Microsoft Visual Studio 2013. Данная среда включает в себя редактор исходного кода, встроенный отладчик и прочие инструменты, облегчающие разработку программ. Visual Studio также предоставляет удобные инструменты для разработки графического

интерфейса, однако в рамках данной работы для обеспечения кроссплатформенности разработанного программного продукта при разработке интерфейса использовалась сторонняя открытая библиотека.

Для реализации программы, которая могла бы моделировать криптосистему RSA с вполне реальными длинами ключей потребовалось выбрать математическую библиотеку, поддерживающую длинную арифметику. В качестве возможных для использования в C++ математических библиотек рассматривались такие свободные библиотеки, как:

- Crypto++
- MIRACL (Multiprecision Integer and Rational Arithmetic Cryptographic Library)
- FLINT (Fast Library for Number Theory)
- NTL (Number Theory Library)
- GMP (GNU Multi-Precision Library)
- MPIR (Multiple Precision Integers and Rationals)

Поскольку для демонстрации выполнения атаки на малую открытую экспоненту требуется вычисление кубического корня, выбор был сделан в сторону библиотек GMP/MPIR, в которых алгоритм вычисления корня n -ой степени реализован с помощью метода Ньютона [9]. MPIR является ответвлением от GMP, поэтому код, написанный с использованием MPIR обладает обратной совместимостью с библиотекой GMP. В частности, при выполнении данной дипломной работы использовалась библиотека MPIR, поскольку сборка библиотеки GMP в операционной системе MS Windows сильно осложнена из-за отсутствия файла-проекта для сборки в ОС Windows. При необходимости разработанная программа может быть откомпилирована в Linux и Mac OS X с использованием GMP или MPIR.

В качестве возможных для реализации графического интерфейса библиотек рассматривались:

- проприетарная библиотека Qt

- свободная библиотека FLTK (Fast, Light Toolkit)
- свободная библиотека wxWidgets

Qt – очень популярный на сегодняшний день набор инструментов для разработки кроссплатформенных программ, однако его выбор являлся бы неоправданно громоздким решением в рамках решения данной задачи. FLTK – очень компактная по своим размерам библиотека, предоставляющая возможности для построения крайне минималистичного интерфейса, который выглядит одинаково на всех платформах. В свою очередь wxWidgets является неким промежуточным решением, сочетающим в себе небольшой в сравнении с Qt размер и красоту элементов интерфейса разработанной программы. При использовании wxWidgets дизайн элементов соответствует дизайну используемой операционной системы. Если в Qt такой же эффект достигается при помощи использования различных стилей элементов интерфейса, то wxWidgets для отображения элементов использует встроенные вызовы системы везде, где это возможно, поэтому элементы не просто похожи на родные элементы платформы – они ими и являются [14].

Таблица 3.1 показывает четыре концептуальных уровня wxWidgets:

1. Внешний API wxWidgets
2. Основной порт
3. API платформы, используемой этим портом
4. Целевая операционная система

Таблица 3.1

wxWidgets API					
Порт wxWidgets					
wxMSW	wxGTK	wxX11	wxMotif	wxMac/ wxCarbon	wxCocoa
API целевой платформы					
Win32	GTK+	Xlib	Motif/Lesstif	Carbon	Cocoa
Операционная система					
Windows	Unix/Linux			Mac OS X	

Программист работает непосредственно с верхним уровнем wxWidgets API. Далее, в зависимости от того, под какую операционную систему будет компилироваться программа, используется соответствующий порт wxWidgets. В свою очередь порт wxWidgets использует системные вызовы для отображения элементов интерфейса во всех местах, где это возможно, или использует реализацию нетипичного элемента под конкретную операционную систему. На рис. 3.1-3.3 в качестве примера приведены изображения одного и того же элемента wxChoicebook в операционных системах Windows XP, Linux и Mac OS X соответственно.

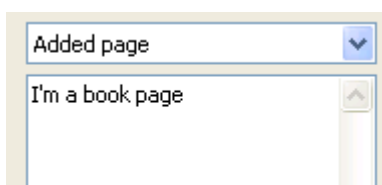


Рис. 3.1 Отображение элемента wxChoicebook в ОС Windows XP (порт wxMSW)

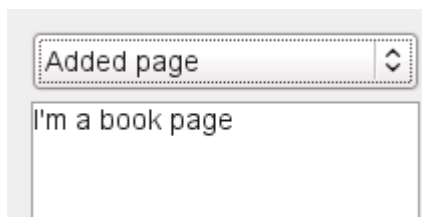


Рис. 3.2 Отображение элемента wxChoicebook в ОС Linux (порт wxGTK)

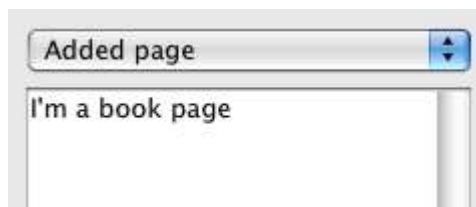


Рис. 3.3 Отображение элемента wxChoicebook в ОС Mac OS X (порт wxCocoa)

Отметим, что wxWidgets постоянно обновляется и также используется такими известными открытыми кроссплатформенными проектами, как аудиоредактор Audacity, FTP-клиент FileZilla, торрент-клиент BitTorrent, среда разработки Code::Blocks и другие.

Таким образом в качестве инструментов были выбраны язык C++, свободная математическая библиотека MPFR для длинной арифметики и открытая библиотека wxWidgets для построения интерфейса программы. Все это при условии статической линковки позволяет разработанной программе

быть переносимой на популярные операционные системы, сохранять родной вид и поведение, а также запускаться без установки каких-либо дополнительных динамических библиотек или виртуальных машин.

3.3 Моделирование криптосистемы RSA

В работе необходимо было реализовать несколько криптосистем с различными параметрами. Основное отличие между криптосистемами заключается в способе генерирования ключей, поэтому был создан базовый класс `RsaBase`, в котором объявлены структуры хранимых данных и реализованы методы шифрования и дешифрования сообщений, а функция генерирования ключей является чисто виртуальной. Конструктор базового класса принимает значение битовой длины модуля криптосистемы, которую хочет сгенерировать пользователь и использует данное значение для генерирования простых чисел p и q соответствующих размеров.

Путем наследования от базового класса `RsaBase` образованы следующие 4 класса:

- `RsaCommon`
- `RsaHastad`
- `RsaWiener`
- `RsaDefinedModulus`

Класс `RsaCommon` использует обычный метод генерирования ключей. В таком случае последовательность выполняемых действий следующая:

1. Генерируется 2 случайных простых числа p и q одинаковой битовой длины.
2. Вычисляется модуль криптосистемы $n = p \cdot q$.
3. Вычисляется функция Эйлера $\varphi(n) = (p - 1)(q - 1)$.
4. Генерируется случайная открытая экспонента e , где $1 \leq e \leq \varphi(n)$ и $\gcd(e, \varphi(n)) = 1$.
5. Вычисляется закрытая экспонента $d = e^{-1} \bmod \varphi(n)$.

Класс `RsaCommon` используется при демонстрации большинства атак, реализованных в данной работе:

- Атака при малом числе возможных сообщений (п. 2.2)
- Атака с использованием мультипликативного свойства шифра RSA (п. 2.4)
- Циклическая атака (п. 2.5)
- Атака на общие модули (п.2.6)

Для демонстрации атаки при малой открытой экспоненте (п. 2.1) используется класс `RsaHastad`, где ключи генерируются следующим образом:

1. Открытая экспонента выбирается $e=3$.
2. Генерируется 2 случайных простых числа p и q одинаковой битовой длины.
3. Вычисляется модуль криптосистемы $n = p \cdot q$.
4. Вычисляется функция Эйлера $\varphi(n) = (p-1)(q-1)$.
5. Вычисляется секретная экспонента $d = e^{-1} \bmod \varphi(n)$.

Для демонстрации атаки Винера (п. 2.3) на малую секретную экспоненту используется класс `RsaWiener`. Конструктор данного класса помимо желаемой длины модуля криптосистемы также принимает значение задаваемой пользователем битовой длины секретной экспоненты. Пользователь также может выбрать, по модулю какой функции вычисляется обратный элемент: по модулю функции Эйлера или функции Кармайкла (см. п. 1.3.1). Таким образом метод генерирования ключей в классе `RsaWiener` выполняется следующим образом:

1. Генерируется случайная открытая экспонента d , битовая длина которой задается пользователем.
2. Генерируется 2 случайных простых числа p и q одинаковой битовой длины.
3. Вычисляется модуль криптосистемы $n = p \cdot q$.
4. Вычисляется значение функции $\varphi(n) = (p-1)(q-1)$ или $\lambda(n) = \text{lcm}(p-1, q-1)$ в зависимости от выбора пользователя.

5. Вычисляется секретная экспонента, как $d = e^{-1} \bmod \varphi(n)$ или $d = e^{-1} \bmod \lambda(n)$.

Последний класс `RsaDefinedModulus` используется, для генерирования ключей криптосистемы с заранее заданными простыми числами p и q , иными словами с заранее заданным модулем n . Данный класс используется при демонстрации атаки на общие модули (п. 2.6), где одна из криптосистем генерирует все ключи случайным образом и является объектом класса `RsaCommon`, а другая криптосистема использует p и q первой криптосистемы, но генерирует свои собственные экспоненты e и d и является объектом класса `RsaDefinedModulus`. Таким образом, метод генерирования ключей в классе `RsaDefinedModulus` определяется следующим образом:

1. Простые числа p и q не генерируются случайным образом, а выбираются равными тем, что поступили на вход конструктора класса `RsaDefinedModulus`.
2. Вычисляется модуль криптосистемы $n = p \cdot q$.
3. Вычисляется значение функции $\varphi(n) = (p - 1)(q - 1)$.
4. Генерируется случайная открытая экспонента e , где $1 \leq e \leq \varphi(n)$ и $\gcd(e, \varphi(n)) = 1$.
5. Вычисляется закрытая экспонента $d = e^{-1} \bmod \varphi(n)$.

Исходный код, отвечающие за генерирование ключей в вышеописанных классах приводятся в приложении 1. Фрагменты исходного кода, отвечающие за выполнение алгоритмов побочных атак приводятся в приложении 2.

3.4 Разработка графического интерфейса

При компилировании `wxWidgets` в операционной системе семейства MS Windows используется порт `wxMSW`. Данный порт поддерживает 32-разрядные и 64-разрядные версии операционных систем, включая Windows 98/NT/2000, Windows XP, Windows Vista, и Windows 7/8. Для совместимости откомпилированного исполняемого файла для операционных систем от

Windows XP до Windows 8 в настройках проекта Visual Studio в качестве набора инструментов выбирается «v120_xp».

Разработанное программное обеспечение состоит из одного главного окна и шести панелей, предназначенных для демонстрации шести побочным атак на криптосистему RSA. В каждый момент времени в главном окне может отображаться только одна панель. Тип атаки выбирается при помощи меню главного окна, как показано на рис. 3.4:

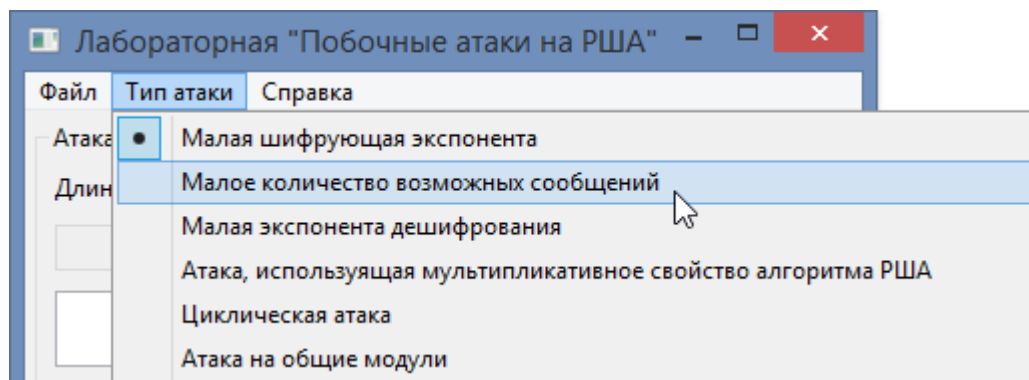


Рис. 3.4 Выбор типа атаки

Меню также позволяет управлять состоянием генератора случайных чисел и вызывать справочную информацию по выбранной в текущий момент времени атаке. Также внизу главного окна находится статус-панель, иногда отображающая дополнительную информацию и подсказки при наведении мыши на некоторые элементы управления или по завершению определенных операций.

В целях отладки была реализована функция, позволяющая вручную проинициализировать генератор случайных чисел. Это позволяет получить одинаковые ключи генерируемой криптосистемы, запустив программу ни один раз. Для задания начального состояния генератора случайных чисел используется пункт меню «Инициализация ГСЧ», как показано на рис. 3.5.

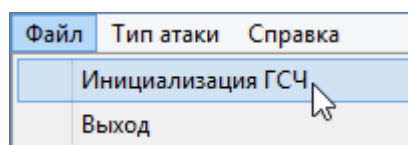


Рис. 3.5 Инициализация генератора случайных чисел

При выборе соответствующего пункта меню появляется диалоговое окно управления ГСЧ (рис. 3.6)

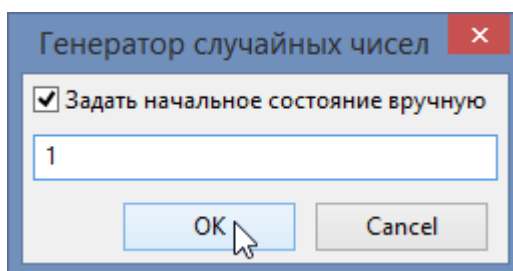


Рис. 3.6 Управление ГСЧ

Если в диалоговом окне инициализации ГСЧ отметить галочку, то текстовое поле, как показано на рис. 3.6, становится активным, и появляется возможность вручную задать начальное значение ГСЧ. Если снять галочку, то, как и при запуске программы, генератор случайных чисел инициализируется текущим системным временем.

При кроссплатформенной разработке необходимо принимать в расчет различия размеров отдельных элементов управления в различных операционных системах. Также элементы должны уметь адаптироваться к изменению размеров окна. По этим причинам вместо задания статического положения элементов и их абсолютных размеров для создания компоновки панелей используются система сайзеров (англ. sizer). Алгоритм компоновки, используемый сайзерами в wxWidgets родственен системам компоновки таких библиотек для разработки интерфейса, как GTK+ и QT для языка C++ или AWT для языка Java. Он основан на предоставлении отдельными окнами информации об их минимально требуемом размере, а также об их возможностях растягиваться при изменении размеров родительского окна.

При использовании сайзеров программист чаще всего не задает самостоятельно размер главного окна, вместо этого к главному окну привязан некоторый сайзер, который и обеспечивает рекомендованный размер. Этот сайзер, в свою очередь, для получения требуемой информации опрашивает дочерние элементы, которые могут быть элементами управления, панелями, пустыми пространствами и другими сайзерами. Сайзеры образуют свою

собственную, отдельную от оконной, иерархию. Например, панель для демонстрации атаки на криптосистемы с общими модулями имеет следующую иерархию сайзеров (рис. 3.7):

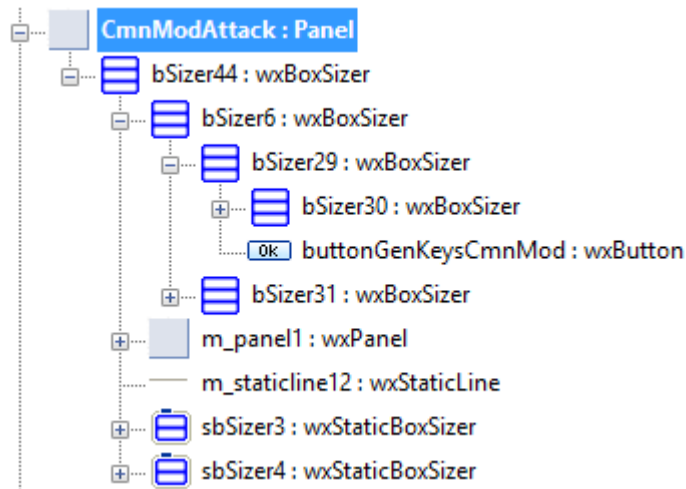


Рис. 3.7 Иерархия дочерних элементов панели «Атака на общие модули»

3.4.1 Окно демонстрации атаки на малую экспоненту шифрования

Перед началом работы с программой окно для демонстрации атаки на малую экспоненту шифрования (п. 2.1) выглядит, как показано на рис. 3.8.

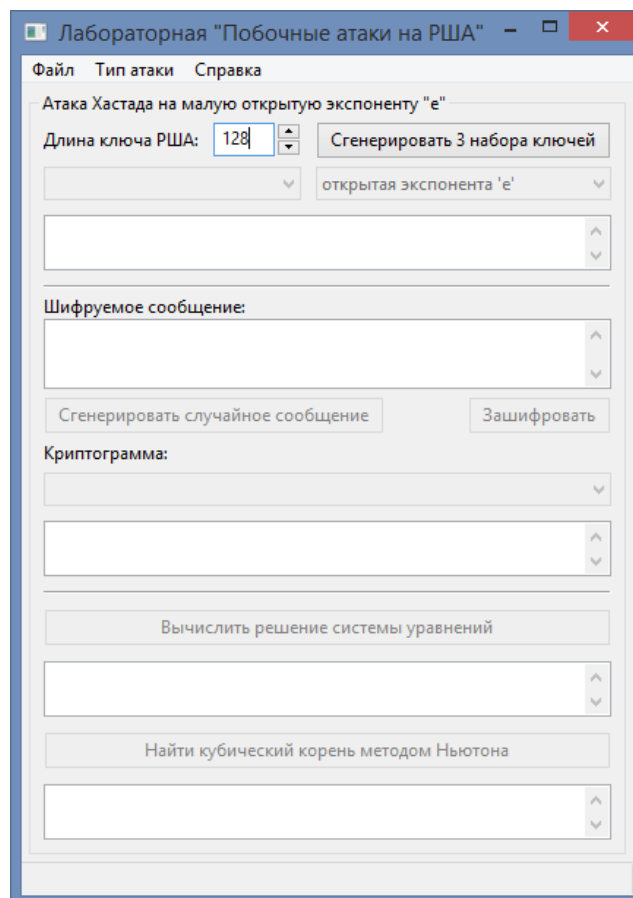


Рис. 3.8 Окно демонстрации атаки на малую шифрующую экспоненту

Разберем подробно элементы данного окна. Как видно из рис. 3.8, в первую очередь предлагается сгенерировать три криптосистемы, где длина модуля криптосистемы задается вручную, а шифрующая экспонента $e=3$ для трех генерируемых криптосистем. Для примера сгенерируем 3 криптосистемы с длиной модуля, равной 128 бит.

После генерирования криптосистем становятся доступными два выпадающих списка, служащие для просмотра сгенерированных параметров криптосистемы. Первый выпадающий список определяет криптосистему, параметр которой хочет просмотреть пользователь, второй выпадающий список определяет сам параметр. Как показано на рис. 3.9, можно просмотреть обе экспоненты e и d , модуль криптосистемы n и простые сомножители p и q . Значение выбранного параметра отображается в текстовом поле, расположенном сразу после выпадающих списков.

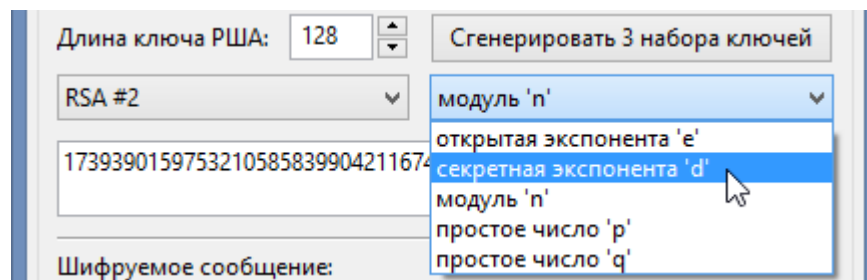


Рис. 3.9 Просмотр параметров сгенерированных криптосистем

Ниже расположены две кнопки «сгенерировать случайное сообщение» и «зашифровать», позволяющие выбрать случайное сообщение и вычислить 3 криптограммы, используя сгенерированные на предыдущем шаге открытые ключи трех криптосистем (рис. 3.10).

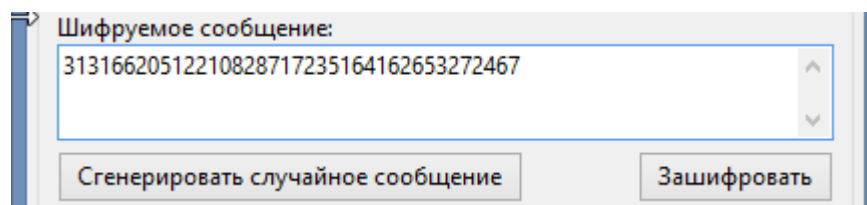


Рис. 3.10 Выбор случайного сообщения для передачи

После нажатия на кнопку «Зашифровать» ниже становится доступным для использования еще одно выпадающее меню, состоящее из трех пунктов.

Каждый пункт меню служит для отображения вычисленной криптограммы в текстовом поле, которое находится сразу после выпадающего меню (рис. 3.11).

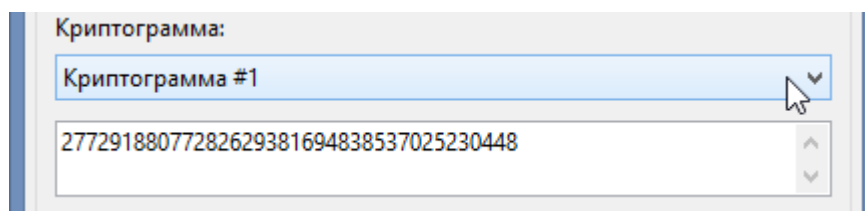


Рис. 3.11 Просмотр вычисленных криптограмм

Ниже расположена кнопка «Вычислить решение системы уравнений». При условии, что злоумышленник перехватил 3 криптограммы (рис. 3.11), соответствующие одному и тому же сообщению (рис. 3.10), нажатие на кнопку «Вычислить решение системы уравнений» приведет к вычислению решения системы (19), используя китайскую теорему об остатках (20). Полученное решение выводится в текстовом поле ниже (рис. 3.12).

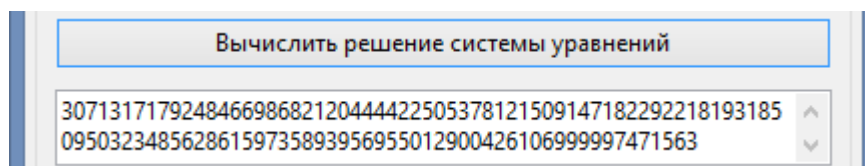


Рис. 3.12 Вычисление решения системы, используя китайскую теорему об остатках

Теперь, вычислив кубический корень из решения системы уравнений (рис. 3.12), злоумышленник получит исходное передаваемое сообщение. Для выполнения этой операция ниже расположена кнопка «Найти кубический корень методом Ньютона». Нажатие на эту кнопку приводит к выводу найденного кубического корня в текстовом поле ниже (рис. 3.13).

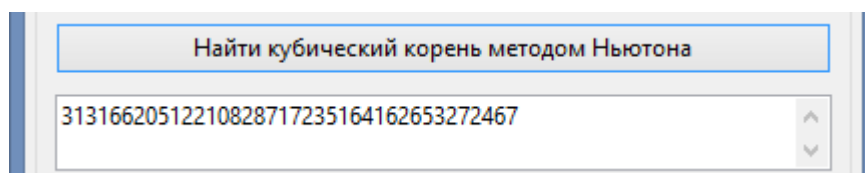


Рис. 3.13 Нахождение кубического корня методом Ньютона

Таким образом, глядя на полученный результат на рис. 3.13 и исходное сообщение на рис. 3.10, можно убедиться в том, что атака была выполнена успешно и вычисленный кубический корень в точности совпадает с исходным сообщением.

После выполнения атаки на малую шифрующую экспоненту окно программы выглядит следующим образом (рис. 3.14):

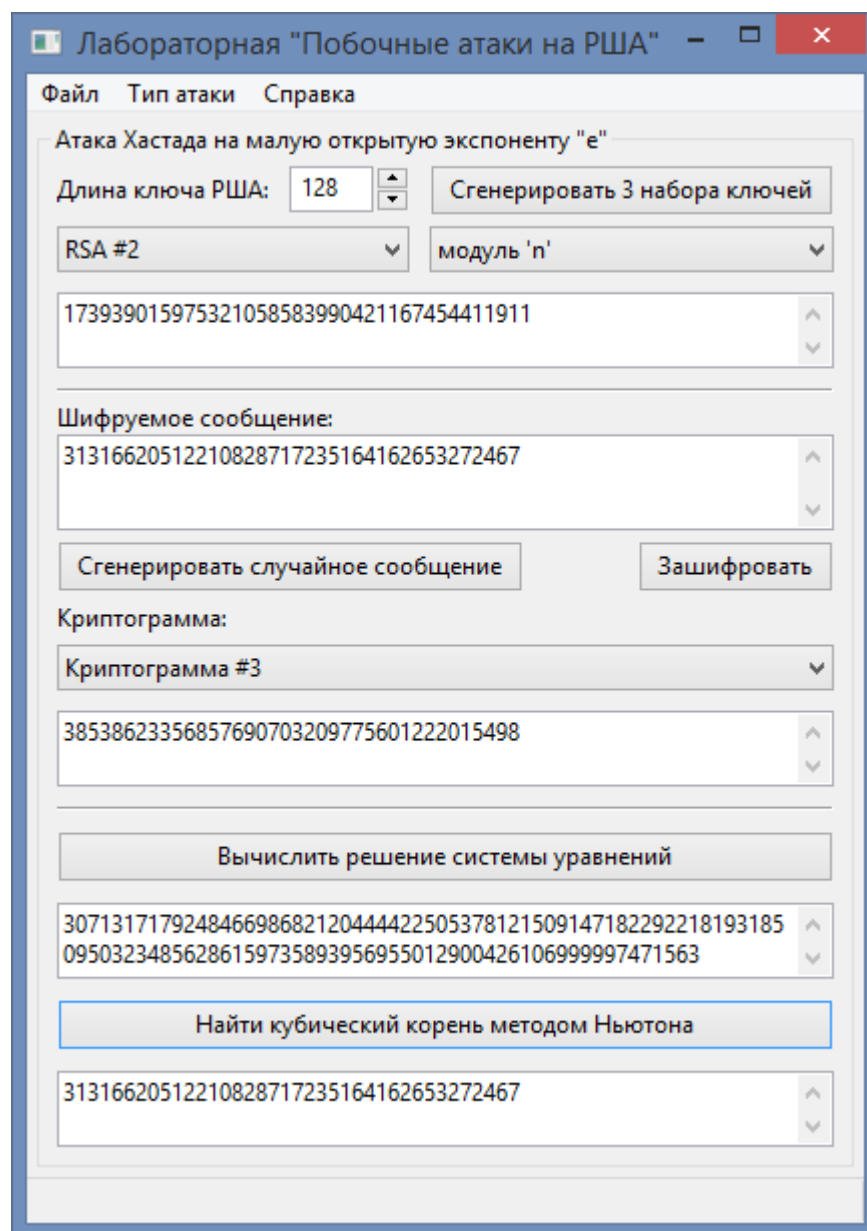


Рис. 3.14 Успешное выполнения атаки на малую шифрующую экспоненту

3.4.2 Окно демонстрации атаки при малом числе возможных сообщений

Перед началом работы с программой окно для демонстрации атаки при малом количестве возможных сообщений (п. 2.2) выглядит, как показано на рис. 3.15.

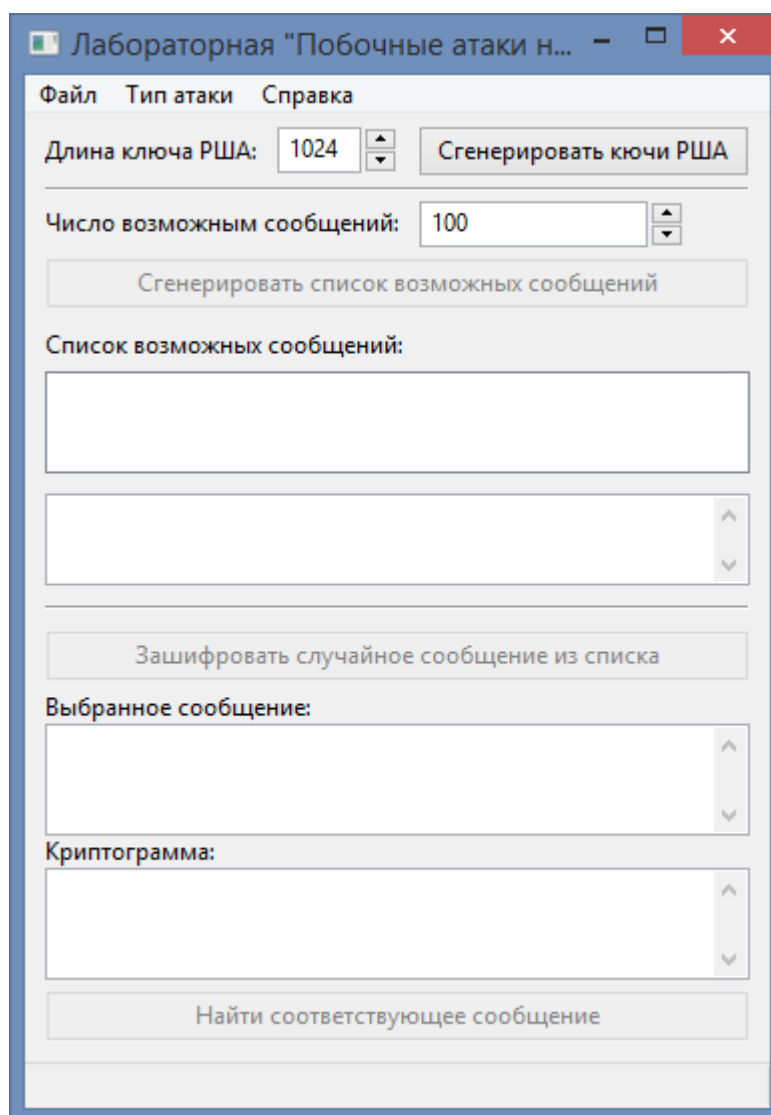


Рис. 3.15 Окно демонстрации атаки при малом количестве возможных сообщений

В первую очередь пользователю предлагается выбрать длину модуля моделируемой криптосистемы и сгенерировать ключи. После генерирования ключей становится доступна кнопка «Сгенерировать список возможных сообщений», нажатие которой приводит к заполнению списка последовательно пронумерованных сообщений. При этом генерируемые сообщения не больше модуля криптосистемы. Число возможных сообщений также выбирается пользователем. Выбор номера возможного сообщения из списка приводит к отображению самого сообщения в текстовом поле ниже (рис. 3.16).

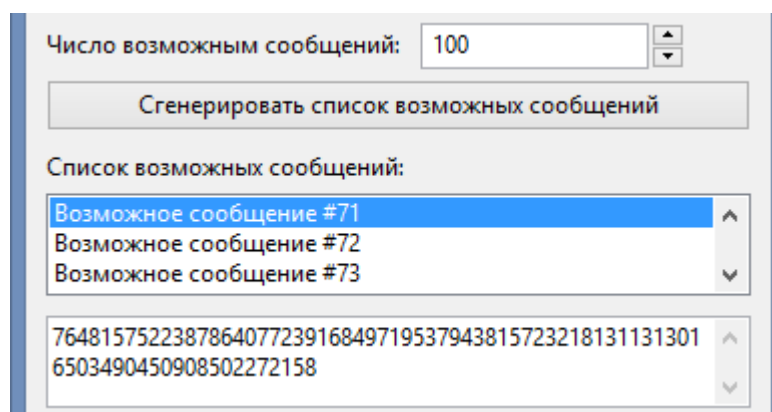


Рис. 3.16 Список возможных сообщений

После того, как был сгенерирован список возможных сообщений становится доступна кнопка «Зашифровать случайное сообщение из списка». При этом случайным образом происходит выбор и шифрование одного сообщения из списка возможных. Выбранное сообщение и соответствующая ему криптограмма отображаются в соответствующих текстовых полях (рис. 3.17).

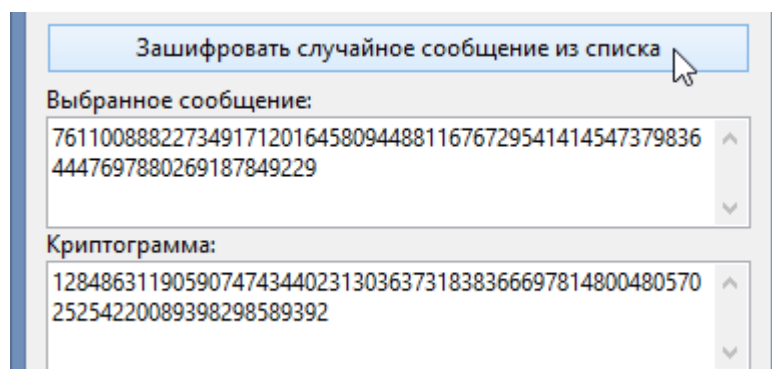


Рис. 3.17 Выбранное сообщение и соответствующая ему криптограмма

После этого становится доступна последняя кнопка «Найти соответствующее сообщение». После ее нажатия программа приступает к последовательному шифрованию каждого возможного сообщения при помощи открытого ключа и сравнению вычисленной криптограммы с имеющейся. Как только криптограмма очередного сообщения совпадет с имеющейся криптограммой вычисления прекращаются и появляется информационное окно, свидетельствующее об успешном выполнении атаки (рис. 3.18), а найденное сообщение автоматически выделяется в списке возможных сообщений (рис. 3.19). Таким образом пользователь может

наглядно убедиться в том, что методом последовательного перебора удалось обнаружить передаваемое сообщение.

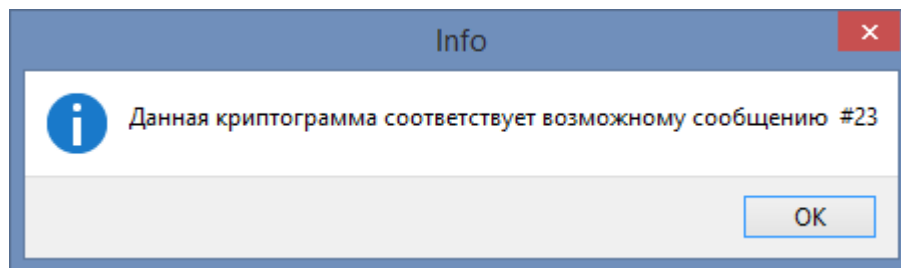


Рис. 3.18 Сообщение об успешном выполнении атаки

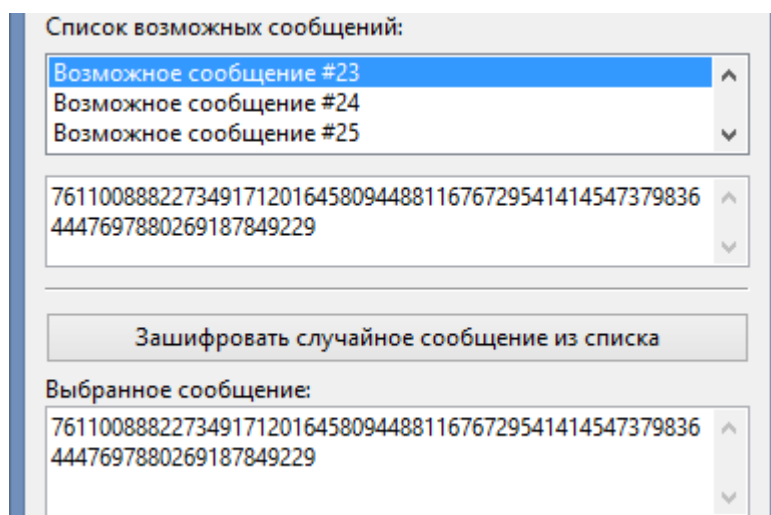


Рис. 3.19 Автоматическое выделение найденного сообщения

В случае, если пользователь задаст слишком большее количество возможных сообщений, используя при этом большой модуль, программа прекратит поиск исходного сообщений через 10 секунд безуспешной работы и покажет соответствующее информационное сообщение.

3.4.3 Окно демонстрации атаки на малую дешифрующую экспоненту

Перед началом работы с программой окно для демонстрации атаки на малую секретную экспоненту (п. 2.3) выглядит так, как показано на рис. 3.20. Пользователю предлагается выбрать битовую длину модуля криптосистемы n и секретной экспоненты d . Также имеется возможность использования функции Кармайкла вместо функции Эйлера для вычисления обратной (т.е. открытой) экспоненты e при генерировании ключей (см. п. 1.3.1).

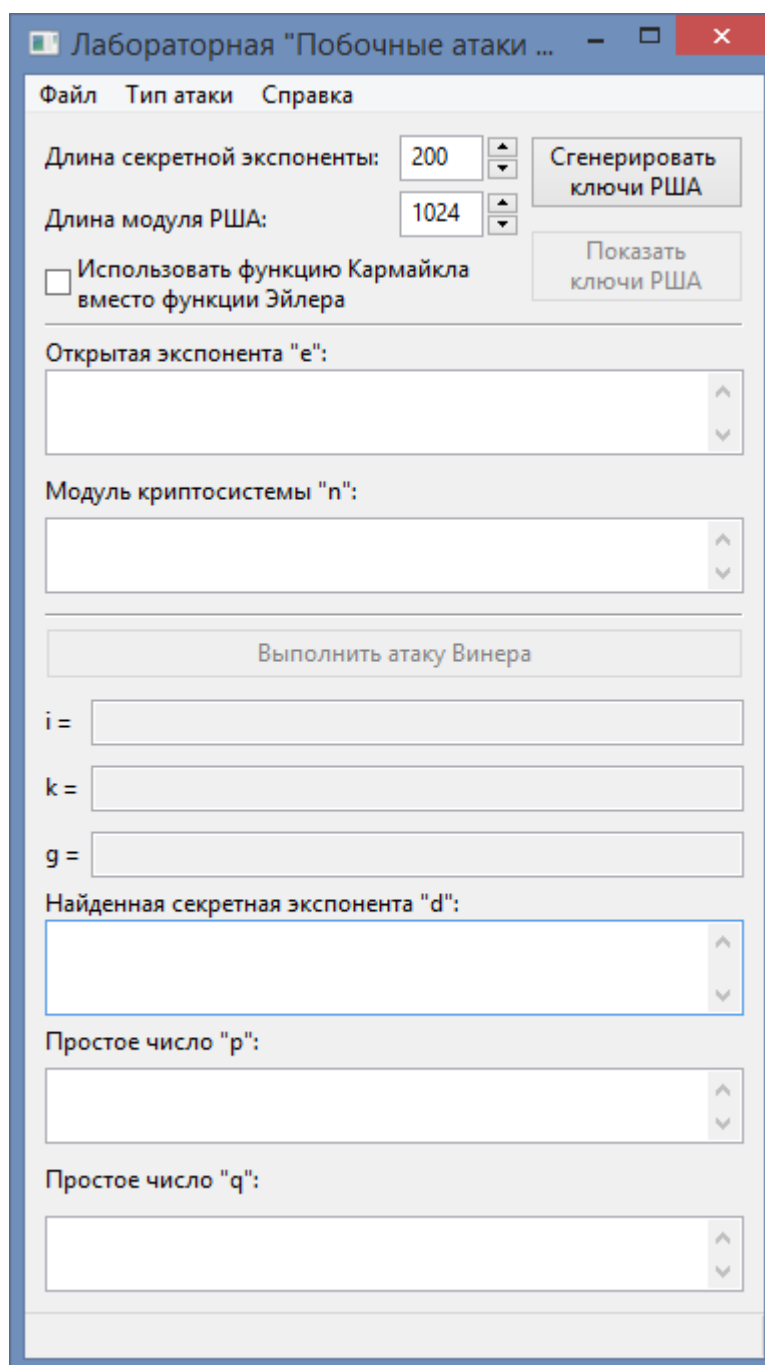


Рис. 3.20 Окно демонстрации атаки на малую дешифрующую экспоненту

После такого, как пользователь сгенерировал ключи криптосистемы становится доступна кнопка «Выполнить атаку Винера», а открытый ключ в виде шифрующей экспоненты и модуля криптосистемы отображается в двух текстовых полях ниже.

Нажатие на кнопку «Выполнить атаку Винера» приводит к выполнению «алгоритма непрерывных дробей» над известным открытым ключом и поиску секретной экспоненты d . Если секретная экспонента была достаточно мала для

успешного выполнения атаки Винера, то найденная секретная экспонента и делители модуля криптосистемы отображаются в соответствующих текстовых полях ниже (рис. 3.21). Помимо этого также выводятся значения чисел k и g , входящих в состав отношения $\frac{k}{d \cdot g}$, а также число i , при котором предположение о $\frac{k}{d \cdot g}$ оказалось верным (см. п. 2.3).

Выполнить атаку Винера

$i =$ 19

$k =$ 569066927

$g =$ 6

Найденная секретная экспонента "d":
924727129

Простое число "p":
18095914062619146223

Простое число "q":
9902356935241853431

Рис. 3.21 Успешное выполнение атаки Винера

После успешного выполнения атаки на малую секретную экспоненту под кнопкой генерирования ключей становится доступна кнопка «Показать ключи РША». Нажав на нее, пользователь увидит диалоговое окно, текстовые поля которого отображают обе секретные экспоненты, модуль криптосистемы и простые делители модуля (числа p и q) (рис. 3.22). Таким образом пользователь может самостоятельно убедиться в том, что найденная секретная экспонента и полученная факторизация модуля (рис. 3.21) в действительности соответствует исходным параметрам криптосистемы (рис. 3.22).

The screenshot shows a window titled "Ключи RSA" (RSA Keys) with a close button in the top right corner. The window contains several input fields for RSA parameters:

- Открытый ключ** (Public key) section:
 - Открытая экспонента 'e': 18378818908722748319984645645528426299
 - РША модуль 'n': 179192200117517285768228695865323241113
- Секретная экспонента** (Secret exponent) section:
 - Секретная экспонента 'd': 924727129
 - Простое число 'p': 9902356935241853431
 - Простое число 'q': 18095914062619146223

Рис. 3.22 Исходные параметры криптосистемы с малой секретной экспонентой

Если пользователь задаст такие параметры криптосистемы, при которых атаке Винера не удастся найти секретную экспоненту, то будет показано соответствующее сообщение об ошибке.

3.4.4 Окно демонстрации атаки, связанной с мультипликативным свойством шифра RSA.

Перед началом работы с программой окно для демонстрации атаки связанной с мультипликативным свойством шифра RSA (п. 2.4) выглядит, как показано на рис. 3.23. Пользователю предлагается сгенерировать криптосистему с заданной длиной модуля, сгенерировать случайное сообщение, зашифровать его, а затем произвести криптоанализ для расшифровки сообщения, зная только криптограмму. После того, как пользователь сгенерирует сообщение, следует скопировать его куда-либо, поскольку после нажатия на кнопку «Зашифровать» криптограмма появляется в том же текстовом поле, куда выводилось шифруемое сообщение.

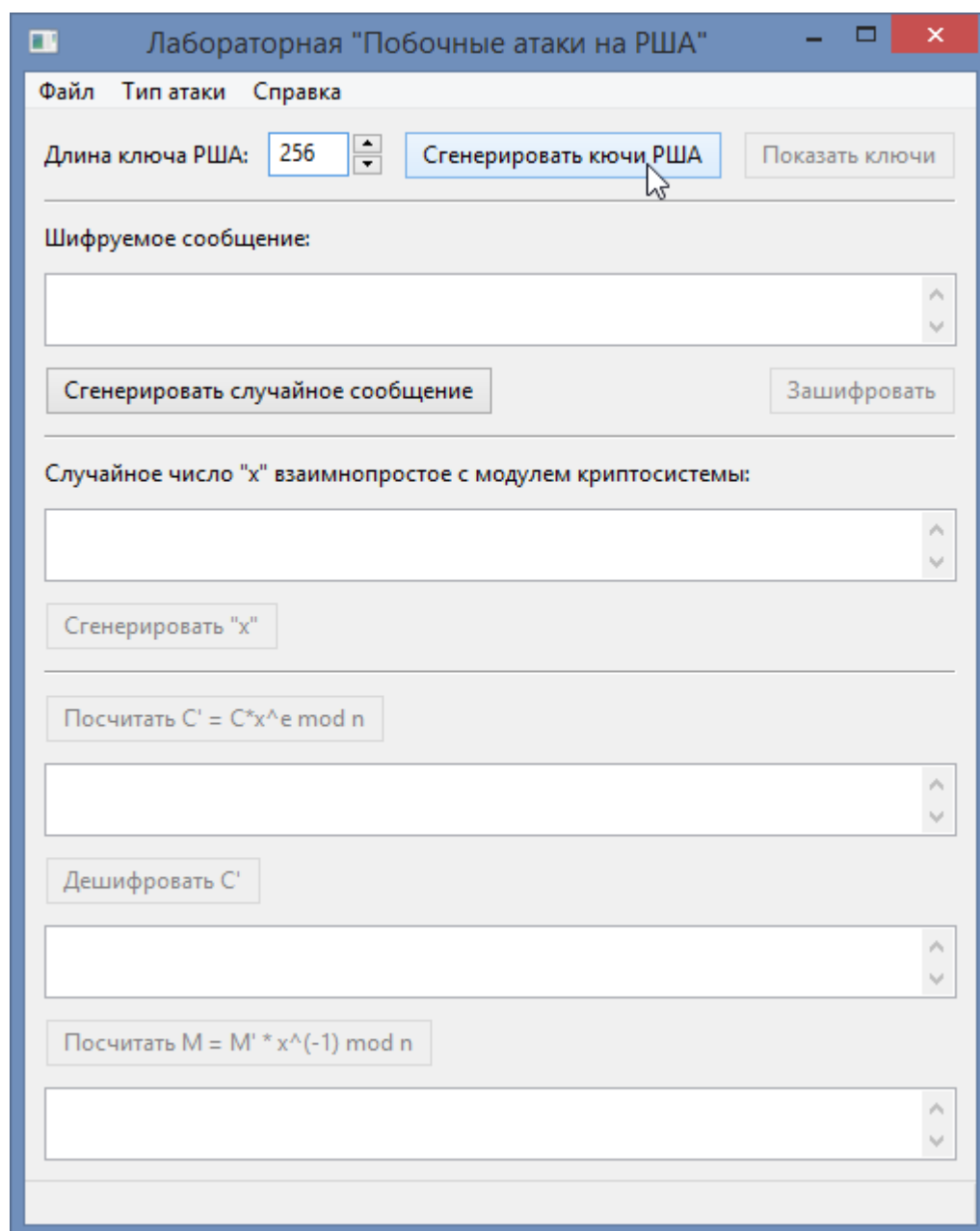


Рис. 3.23 Окно демонстрации атаки, связанной с мультипликативным свойством шифра RSA

Допустим, пользователь сгенерировал сообщение, показанное на рис. 3.24.

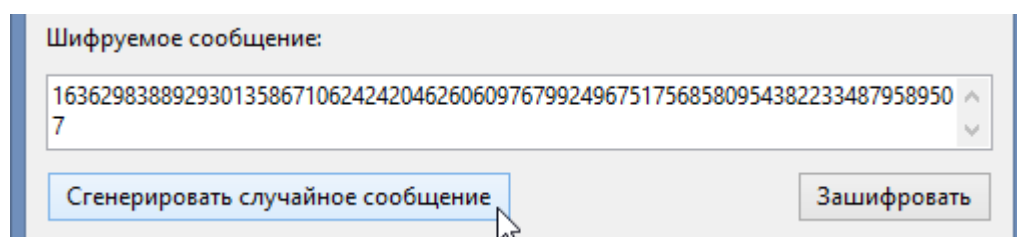
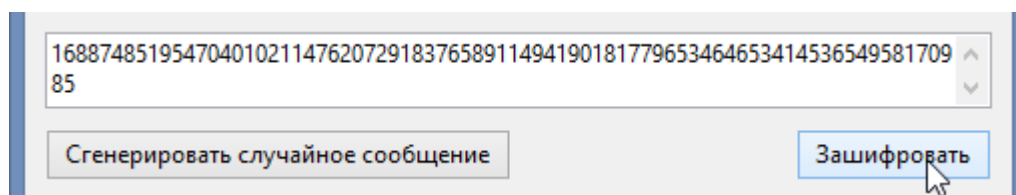


Рис. 3.24 Шифруемое сообщение

Соответствующая криптограмма, полученная нажатием кнопки «зашифровать» показана на рис. 3.25.



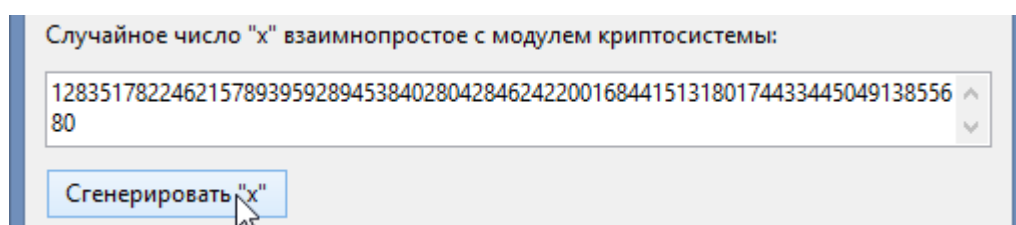
168874851954704010211476207291837658911494190181779653464653414536549581709
85

Сгенерировать случайное сообщение

Зашифровать

Рис. 3.25 Криптограмма

Далее пользователю предлагается сгенерировать любое число x взаимно простое с модулем криптосистемы n (рис. 3.26) и вычислить C' (рис. 3.27) по формуле (48).

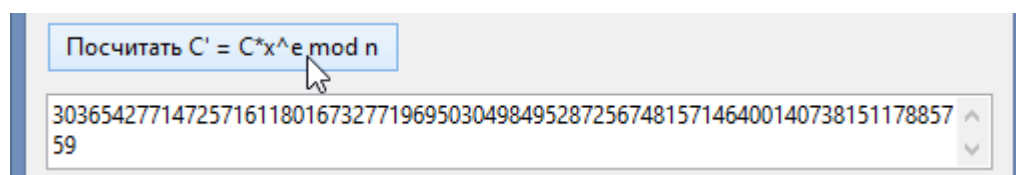


Случайное число "x" взаимно простое с модулем криптосистемы:

128351782246215789395928945384028042846242200168441513180174433445049138556
80

Сгенерировать "x"

Рис. 3.26 Случайное число x взаимно просто с модулем n

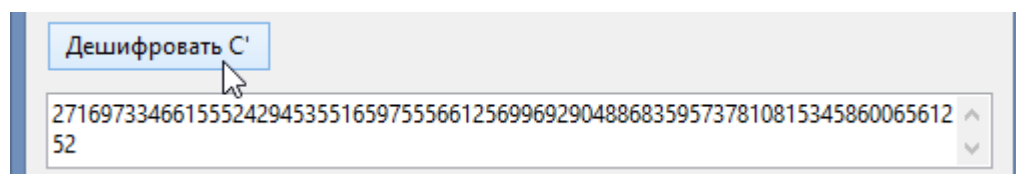


Посчитать $C' = C \cdot x^e \bmod n$

303654277147257161180167327719695030498495287256748157146400140738151178857
59

Рис. 3.27 Вычисление C'

Затем предполагается, что легальный пользователь соглашается расшифровать криптограмму C' для злоумышленника, используя свой закрытый ключ. Передача получившегося сообщения M' злоумышленнику на первый взгляд не представляет никакой опасности, поскольку сообщение выглядит, как случайный набор символов (рис. 3.28).



Дешифровать C'

271697334661555242945355165975556612569969290488683595737810815345860065612
52

Рис. 3.28 Дешифрование C'

Теперь, имея сообщение M' , злоумышленник вычисляет исходное сообщение M по формуле (50) (рис. 3.29).

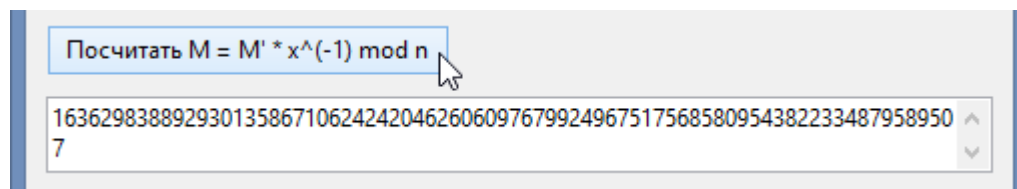


Рис. 3.29 Вычисление исходного сообщения M

Нетрудно убедиться, что исходное сообщение (рис. 3.24) и сообщение, полученное в результате использования данной атаки (рис. 3.29), совпадают. Пользователь также может узнать, какие ключи использовались моделируемой криптосистемой, нажав на кнопку «Показать ключи» (рис. 3.23).

3.4.5 Окно демонстрации циклической атаки

Перед началом работы с программой окно для демонстрации циклической атаки (п. 2.5) выглядит, как показано на рис. 3.30.

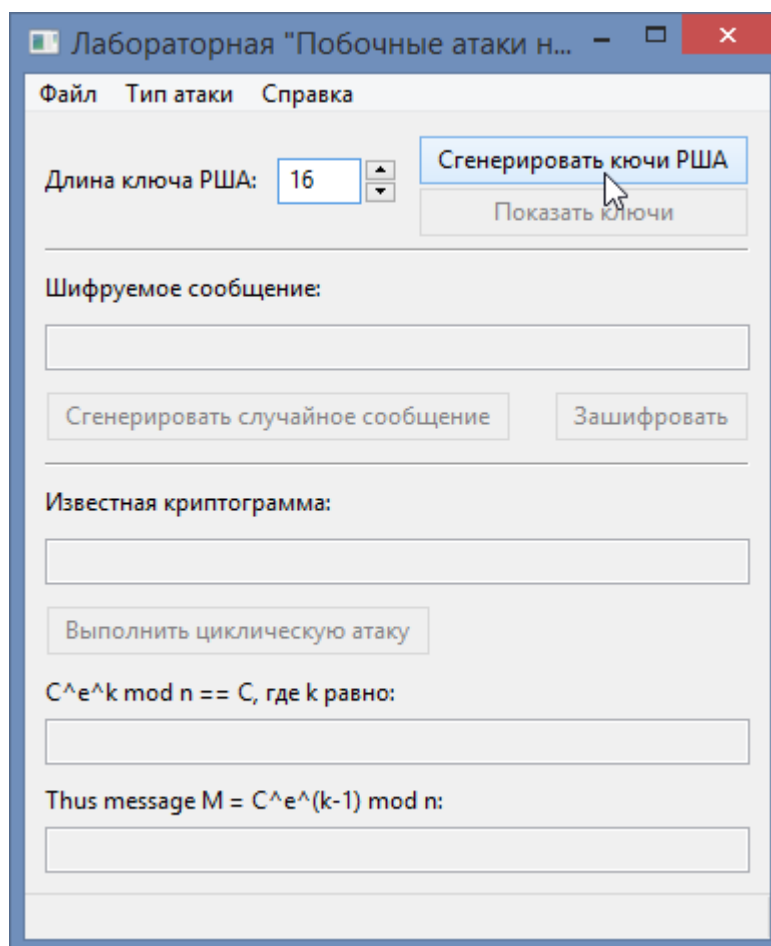
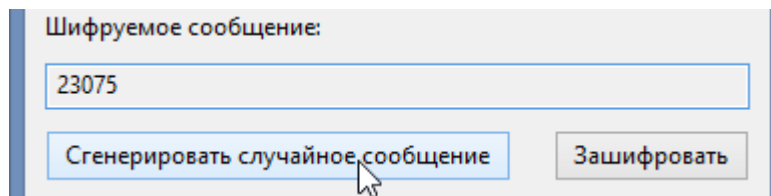


Рис. 3.30 Окно демонстрации циклической атаки

Данное окно содержит небольшие однострочные поля, размеров которых достаточно, поскольку циклическая атака выполнима в обозримое время

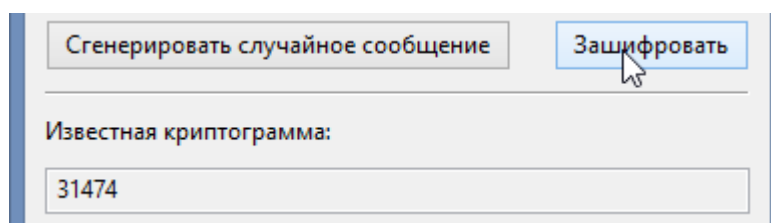
только при небольшой длине модуля криптосистемы. Изначально пользователю предлагается сгенерировать криптосистему с длиной модуля 16 бит, после чего становится доступна кнопка для генерирования случайного сообщения (рис. 3.31).



The screenshot shows a web interface with a title 'Шифруемое сообщение:' (Encrypted message:). Below the title is a text input field containing the number '23075'. Underneath the input field are two buttons: 'Сгенерировать случайное сообщение' (Generate random message) and 'Зашифровать' (Encrypt). A mouse cursor is pointing at the 'Сгенерировать случайное сообщение' button.

Рис. 3.31 Генерирование случайного сообщения

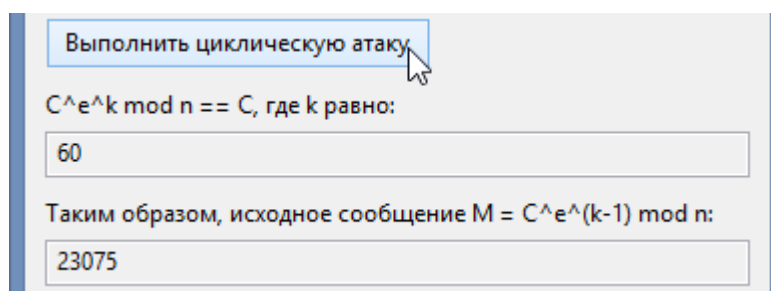
После того, как пользователь сгенерирует сообщение, становится доступна соседняя кнопка «Зашифровать». Нажатие на нее приводит к шифрованию сгенерированного сообщения, а криптограмма выводится в текстовом поле ниже:



The screenshot shows a web interface with two buttons at the top: 'Сгенерировать случайное сообщение' (Generate random message) and 'Зашифровать' (Encrypt). The 'Зашифровать' button is highlighted with a mouse cursor. Below the buttons is a section titled 'Известная криптограмма:' (Known ciphertext:). Under this title is a text input field containing the number '31474'.

Рис. 3.32 Шифрование сообщения

После шифрования исходного сообщения пользователю становится доступна кнопка для выполнения циклической атаки. Нажатие на эту кнопку приводит к последовательному применению операции шифрования к известной криптограмме до тех пор, пока результат вычисления не совпадет с исходной криптограммой. Ниже в первом текстовом поле будет выведено число повторений операции шифрования, а во втором текстовом поле сообщение, найденное с помощью данной атаки (рис. 3.33).



The screenshot shows a web interface with a button 'Выполнить циклическую атаку' (Perform cyclic attack) at the top, which is highlighted with a mouse cursor. Below the button is a text label 'C^e^k mod n == C, где k равно:' (C^e^k mod n == C, where k is equal to:). Under this label is a text input field containing the number '60'. Below this is another text label 'Таким образом, исходное сообщение M = C^e^(k-1) mod n:' (Thus, the original message M = C^e^(k-1) mod n:). Under this label is a text input field containing the number '23075'.

Рис. 3.33 Результат выполнения циклической атаки

В итоге сообщение, полученное в результате применения циклической атаки (рис. 3.33), совпадает с исходным сообщением (рис. 3.31).

В данном окне пользователь может генерировать криптосистему с длиной модуля от 10 до 128 бит. При этом, если выполнение атаки занимает больше десяти секунд, вычисления прерываются, и пользователь получает соответствующее сообщение об ошибке.

3.4.6 Окно демонстрации атаки на общие модули

Перед началом работы с программой окно для демонстрации атаки на общие модули (п. 2.6) выглядит, как показано на рис. 3.34.

Лабораторная "Побочные атаки на ..."

Файл Тип атаки Справка

Длина ключа RSA: 128 Ключи 1го пользователя

Сгенерировать 2 набора ключей с общим модулем Ключи 2го пользователя

Факторизация "n", зная обе экспоненты "e" и "d"

Найти делители "n"

$g^{(k \cdot 2^i)} \bmod n$ равно 1,
при этом $x = g^{(k \cdot 2^i - 1)} \bmod n$ не равно 1 или -1,
где g равно:

и нечетное число $k = (e \cdot d - 1) / 2^t$:

Тогда $p = \gcd(x + 1, n)$:

и $q = \gcd(x + 1, n)$:

Вычисление "d" 2го пользователя

Вычислить "d"

Рис. 3.34 Окно демонстрации атаки на общие модули

Пользователю предлагается сгенерировать ключи для двух криптосистем с заданной длиной модуля, причем модуль первой и второй криптосистемы совпадает. Затем предполагается факторизация общего модуля легитимным пользователем первой криптосистемы и вычисление секретной экспоненты пользователя второй криптосистемы. После генерирования ключей становится доступна кнопка «Найти делители "n"», нажатие на которую приводит к выполнению алгоритма факторизации (см. п. 2.6). В текстовых полях ниже будут выведены (рис. 3.35):

- случайное число g , используемой для нахождения нетривиального корня из единицы,
- нечетное число k , являющееся делителем числа $ed-1$, представленного в виде (61),
- найденные простые делители модуля криптосистемы.

Факторизация "n", зная обе экспоненты "e" и "d"

Найти делители "n"

$g^{(k \cdot 2^3)} \bmod n$ равно 1,
при этом $x = g^{(k \cdot 2^2)} \bmod n$ не равно 1 или -1,
где g равно:

25936042808257807119813820802601891286

и нечетное число $k = (e \cdot d - 1) / 2^5$:

790144075087551933618835334729874607730935247765929448
2607142770008208439

Тогда $p = \gcd(x+1, n)$:

10483817642281600789

и $q = \gcd(x+1, n)$:

3032628278085233497

Рис. 3.35 Выполнение факторизации модуля n при известных экспонентах e и d

После нахождения простых делителей общего модуля пользователю программы предлагается вычислить секретный ключ второй криптосистемы (рис. 3.36).

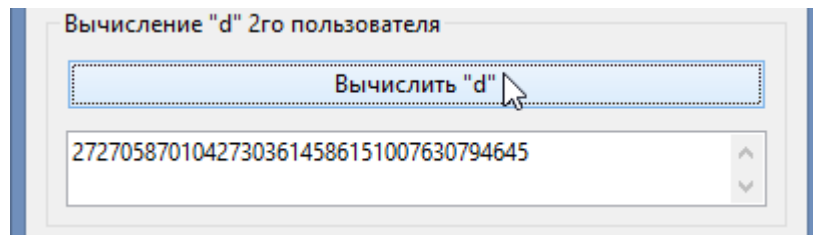


Рис. 3.36 Вычисление секретной экспоненты второй криптосистемы

Рядом с кнопкой генерирования двух наборов ключей расположены также кнопки для просмотра всех параметров обеих криптосистем (рис. 3.34), вызывающие диалоги, показанные на рис. 3.37. Данные диалоговые окна позволяют пользователю программы убедиться в том, что обе криптосистемы использовали общий модуль, а вычисленная секретная экспонента (рис. 3.36) в действительности соответствует секретной экспоненте второго пользователя.

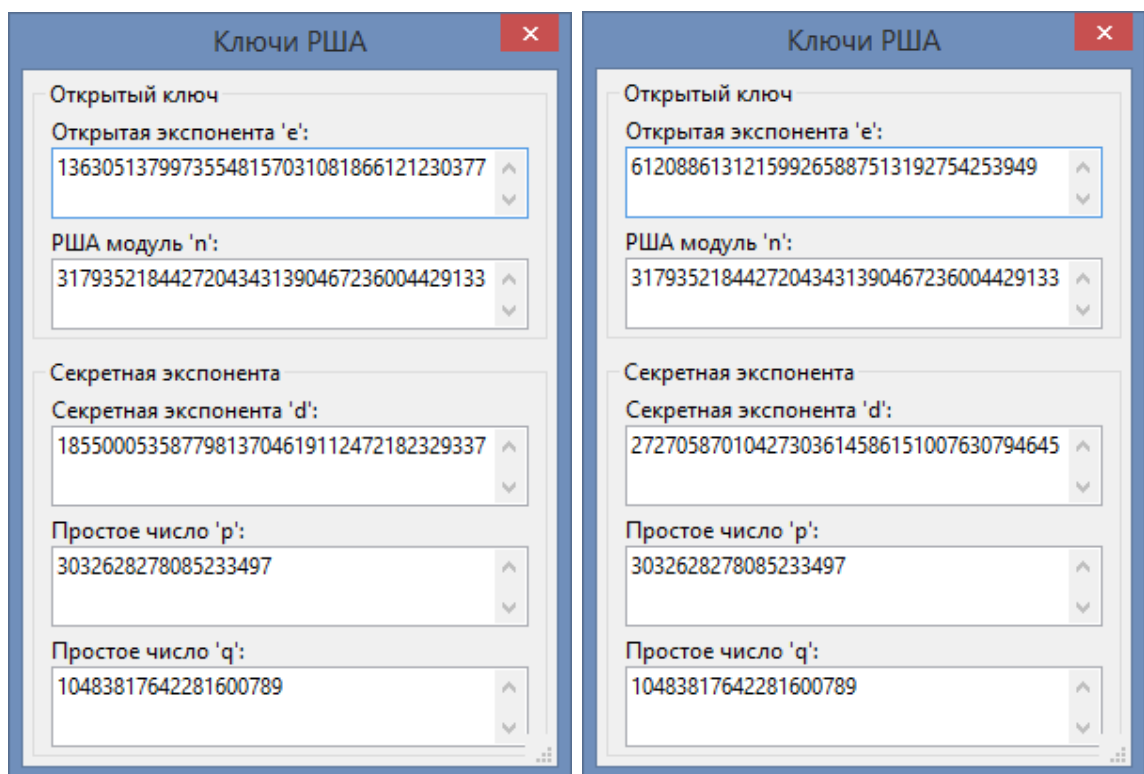


Рис. 3.37 Ключи первой и второй криптосистемы соответственно

3.5 Локализация программы

Интерфейс разработанной программы включает в себя поддержку двух языков: английского и русского. Реализация интернационализации в библиотеке wxWidgets очень похожа на ее реализацию в пакете GNU gettext.

wxWidgets использует каталоги сообщений, заменяя строки из исходного кода переведенными строками из соответствующих каталогов.

Существует два типа каталогов сообщений:

- исходный каталог, который представляет из себя текстовый файл с переводом, имеющий расширение .po (Portable Object),
- бинарный каталог, компилируемый из .po файла в удобный для чтения программой и имеющий расширение .mo (Machine Object).

Для каждого языка требуется свой каталог сообщений. При непосредственном выполнении программы используются только бинарные каталоги.

Такой подход к интернационализации программного обеспечения очень удобен, поскольку переводчики могут заниматься локализацией программы, работая только с файлами с расширением .po, без необходимости заново компилировать программу.

Для выполнения перевода и поддержки каталогов в актуальном состоянии удобно пользоваться программой poEdit, которая по сути является интерфейсом к gettext [14]. Файл каталога создается автоматически из исходных кодов разрабатываемой программы. Строки, которые необходимо перевести, в исходных кодах программист оборачивает в макрос `_()`, эквивалентный функции `wxGetTranslation()`. Например:

```
_("Sample text")
```

poEdit находит все подобные строки в исходных кодах и составляет каталог сообщений, которые необходимо перевести. По умолчанию языком разработчика считается «английский США» (English - UNITED STATES), поэтому строки текста в исходном коде изначально написаны на английском языке. Локализация программы осуществлялась на русский язык при помощи вышеупомянутого инструмента poEdit (рис. 3.38). Также был создан каталог сообщения для обычного английского языка (English), который может быть использован в будущем для внесения поправок без перекомпилирования программы.

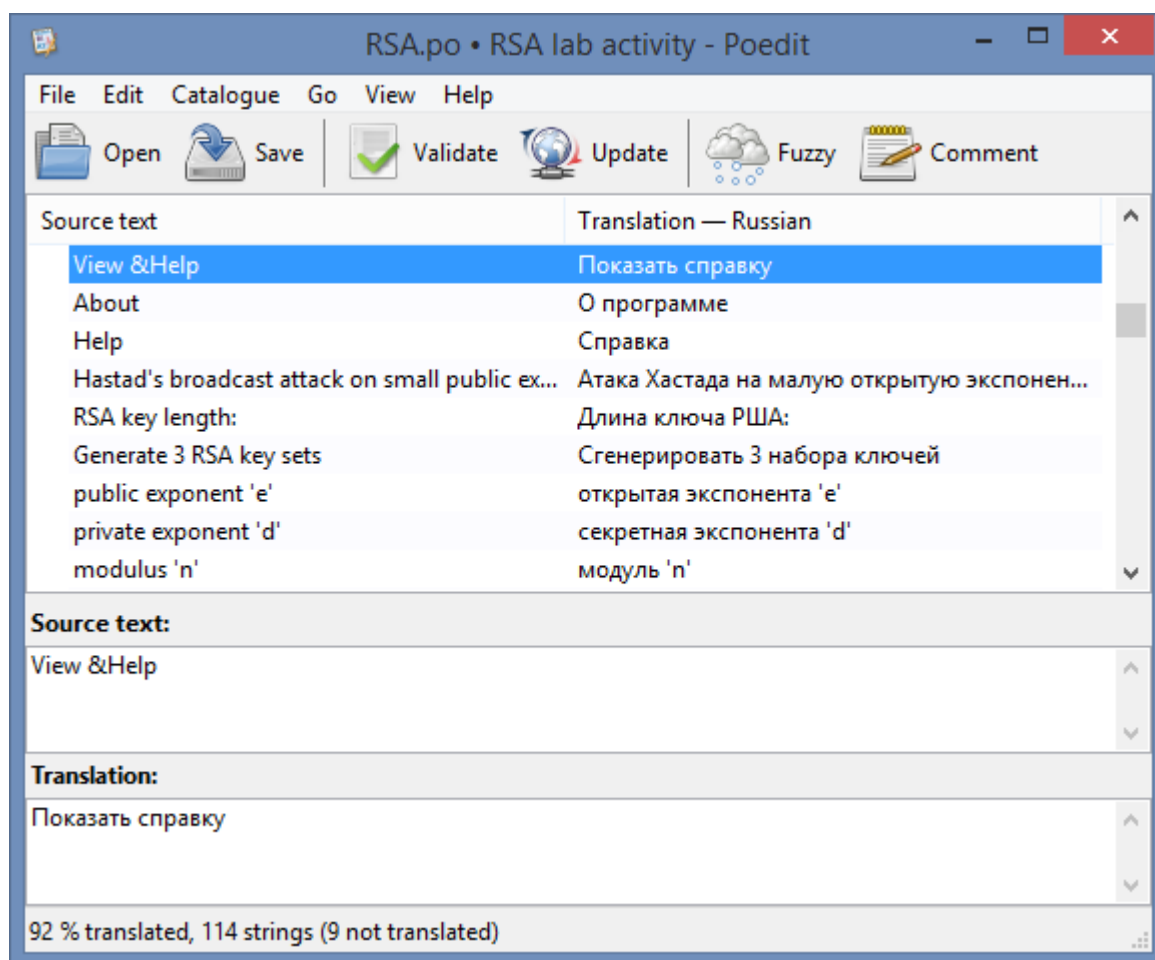


Рис. 3.38 Процесс локализации разработанной программы

После запуска разработанной программы появляется небольшое диалоговое окно, позволяющее выбрать один из двух доступных языков (рис. 3.39).

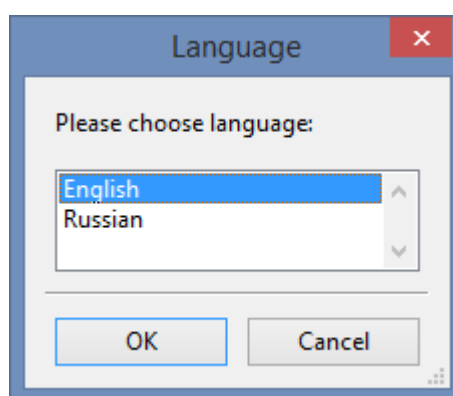


Рис. 3.39 Выбор языка интерфейса разработанной программы

Если проигнорировать окно, показанное на рис. 3.39 (закрыть его или нажать «Cancel»), то программа выберет язык в зависимости от настроек интернационализации текущей операционной системы.

Также предусмотрена возможность запуска программы с указанием необходимого языка в качестве опции командной строки сразу после имени исполняемого файла. Например, можно создать ярлык с параметрами:

"C:\RSA attacks lab.exe" en,

где en – сокращенное название английского языка. В таком случае программа сразу запустится, выбрав английский в качестве языка интерфейса. Помимо en можно также использовать ru и en_us. Выбор последней опции означает, что программа будет использовать текстовые строки непосредственно из исходных кодов.

Выводы по главе

В ходе выполнения данной дипломной работы был разработан программный комплекс для выполнения лабораторной работы на тему «Побочные атаки на криптосистему RSA» по курсу «Криптография с открытым ключом». Он предназначен для наглядной демонстрации угроз, которые появляются при неправильном выборе параметров криптосистемы.

Было разработано 6 модулей, предоставляющих пользователю возможность смоделировать работу криптосистемы с заданными параметрами и совершить ту или иную побочную атаку. Практическая работа позволяет лучше понять и усвоить, какие параметры и особенности проектирования влияют на стойкость криптосистемы RSA.

Разработанное программное обеспечение имеет дружелюбный интерфейс и поддерживает два языка: русский и английский. Помимо этого, за счет выбора кроссплатформенных инструментов разработки есть возможность откомпилировать исходный код без каких-либо изменений в другой операционной системе, если это необходимо. В рамках данной дипломной работы исполняемый файл был откомпилирован под операционную систему MS Windows, а все необходимые библиотеки были привязаны к исполняемому файлу статически.

Глава 4. Методические указания по проведению лабораторной работы

4.1 Рекомендации по подготовке к лабораторной работе

В учебную программу студентов специальности «Защищенные системы связи» входят два больших раздела: симметричные криптосистемы и криптосистемы с открытым ключом. Помимо лекций и теоретического материала по особо важным параграфам предусмотрены лабораторные работы, в ходе которых учащиеся получают возможность не только лучше понять и усвоить принципы работы алгоритмов, но и увидеть их работу на практике.

Разработанный программный комплекс послужит прекрасным дополнением к разделу «Криптосистемы с открытым ключом», поскольку изучение криптосистемы RSA является очень важным этапом в изучении данного раздела. Более того, криптосистема RSA, как отмечалось ранее, находит активное применение на практике как для шифрования, так и для выполнения электронной цифровой подписи.

Перед выполнением лабораторного практикума предлагается прослушать лекционный материал, а также изучить соответствующие разделы учебника «Основы криптографии» [1]. Рекомендуется обратить особое внимание на следующие темы:

- Делимость. Алгоритм Евклида
- Операции по числовому модулю
- Китайская теорема об остатках
- Малая теорема Ферма и теорема Эйлера
- Тестирование простых чисел
- Генерирование пар ключей в КС RSA
- Побочные атаки на КС RSA
- Выбор параметров для КС RSA

Ниже представлены методические указания, специально разработанные для решения задач, поставленных на лабораторную работу.

4.2 Методические указания к лабораторной работе

Лабораторная работа ИССЛЕДОВАНИЕ ПОБОЧНЫХ АТАК НА КРИПТОСИСТЕМУ РША

Цель работы

Изучить влияние параметров и способов проектирования криптосистемы РША на возможность ее криптоанализа, используя побочные атаки, а также закрепить знания, полученные на лекциях курса «Основы криптографии с открытым ключом» по теме: «Побочные атаки на КС РША».

Используемое программное обеспечение

Для выполнения работы используется специально разработанный программный комплекс «RSA attacks lab.exe»

Задание

1. Для выполнения каждой из атак сгенерировать набор ключей РША (или несколько наборов).
2. Выполнить необходимые для осуществления криптоанализа шаги.
3. Проанализировать результаты выполнения атак.

Порядок выполнения

Для начала работы перейти в каталог, содержащий описание лабораторных работ, и убедиться в установке программы «RSA attacks lab». Прodelать нижеописанные шаги сначала для небольших чисел, которые можно занести в отчет (длина модуля РША ~20 бит). Затем убедиться в том, что данные атаки выполняются также быстро и при больших длинах ключей (за исключением циклической атаки).

1. Запустить программу «RSA attacks lab.exe». После запуска сразу откроется окно для демонстрации атаки на малую шифрующую экспоненту.
2. Сгенерировать 3 набора ключей RSA с одинаковой малой открытой экспонентой $e=3$.
3. Сгенерировать случайное сообщение и зашифровать его.
4. Вычислить решение системы сравнений, используя китайскую теорему об остатках.
5. Найти кубический корень методом Ньютона из решения системы, полученного на предыдущем шаге. Убедиться в том, что это и есть исходное сообщение, сгенерированное в п.3.
6. Переключиться на окно демонстрации атаки при малом количестве возможных сообщений.
7. Сгенерировать ключи криптосистемы и список возможных сообщений заданной длины.
8. Зашифровать случайным образом одно из возможных сообщений.
9. Путем последовательного шифрования возможных сообщений по списку и сравнения с криптограммой из п.8 найти исходное сообщение. Оценить скорость выполнения данной атаки в зависимости от количества возможных сообщений и длины модуля.
10. Переключиться на окно демонстрации атаки Винера (атака на малую секретную экспоненту).
11. Сгенерировать ключи криптосистемы.
12. Выполнить атаку Винера. Убедиться в том, что найденные в результате выполнения атаки секретная экспонента и делители модуля криптосистемы действительно соответствуют параметрам заданной криптосистемы.
13. Задавая различные длины модуля и секретной экспоненты, убедиться в том, что атака Винера дает результат при битовой длине секретной экспоненты приблизительно меньше четверти битовой длины модуля криптосистемы.

13. Переключиться на окно демонстрации атаки, связанной с мультипликативным свойством шифра РША.
12. Сгенерировать ключи криптосистемы.
13. Сгенерировать случайное сообщение M и зашифровать его.
14. Сгенерировать случайное число x взаимно простое с модулем криптосистемы и вычислить специальную криптограмму C' .
15. Дешифровать C' , получив тем самым некое сообщение M' .
16. Извлечь исходное сообщение M из M' . Убедиться в том, что полученное в результате выполнения атаки сообщение совпадает с исходным сообщением из п.13.
17. Переключиться на окно демонстрации циклической атаки.
18. Сгенерировать ключи криптосистемы.
19. Сгенерировать случайное сообщение и зашифровать его.
20. Выполнить циклическую атаку. Убедиться в том, что найденное сообщение соответствует исходному сообщению из п.19.
21. Увеличив длину модуля криптосистемы, убедиться в том, что алгоритм выполнения данной атаки обладает не полиномиальной сложностью.
22. Переключиться на окно демонстрации атаки на общие модули.
23. Сгенерировать 2 набора ключей с общим модулем.
24. Выполнить факторизацию общего модуля, зная обе секретные экспоненты.
25. Вычислить секретную экспоненту второго набора ключей, зная делители общего модуля криптосистемы. Просмотрев исходные параметры обоих наборов ключей и результаты выполнения атаки, убедиться в том, что атака на общие модули привела к взлому криптосистемы.

Отчет

1. Титульный лист.
2. Сгенерированные параметры криптосистем.
3. Сгенерированные сообщения и соответствующие им криптограммы.

4. Результаты промежуточных вычислений, выводимые программой.

5. Результаты выполнения атак.

Пример отчета

1. Атака на малую шифрующую экспоненту

Параметры криптосистем:

набор ключей	модуль n	экспонента e	экспонента d	простое p	простое q
1	110767	3	73387	431	257
2	243043	3	161339	677	359
3	81247	3	53611	719	113

Шифруемое сообщение: 64945

Криптограмма 1: 19255

Криптограмма 2: 176455

Криптограмма 3: 1363

Общее решение системы сравнений: 273928464708625

Кубический корень из общего решения: 64945

2. Атака Винера

Шифрующая экспонента e : 51470893

Модуль криптосистемы n : 1071303463

Найденная секретная экспонента: 37

Найденные делители модуля: 33931 и 31573

$i = 37, k = 16, g = 9$

3. Атака с использованием мультипликативного свойства шифра РША

Параметры криптосистемы:

- Открытая экспонента e : 692923
- Секретная экспонента d : 327283
- Модуль криптосистемы n : 697483

Выбранное сообщение M : 81662

Зашифрованное сообщение C : 511286

Случайное число x взаимно простое с модулем n : 449665

Комбинированная криптограмма $C' = C \cdot x^e \bmod n = 253342$

Расшифрованное M' : 155729

Исходное сообщение $M = M' \cdot x^{-1} \bmod n = 81662$

4. Циклическая атака

Открытая экспонента e : 112031

Модуль криптосистемы n : 117197

Шифруемое сообщение M : 31990

Криптограмма C : 64361

$C^{e^k} \bmod n = C$, где $k = 3222$

Исходное сообщение $M = C^{e^{k-1}} \bmod n = 31990$

5. Атака на общие модули

Открытая экспонента e_1 : 374263

Открытая экспонента e_2 : 294143

Секретная экспонента d_1 : 16327

Общий модуль: 385241

Случайное число g , используемое для факторизации: 302073

$e_1 \cdot d_1 - 1 = k \cdot 2' = 5967375 \cdot 2^{10}$

Делители модуля: 601 и 641

Секретная экспонента d_2 : 139007

Контрольные вопросы

1. Каковы необходимые условия для выполнения атаки на малую шифрующую экспоненту?
2. Каким образом можно противостоять атаке при малом числе возможных сообщений?
3. Какова приблизительная длина секретной экспоненты, при которой атака Винера выполняется успешно?
4. Каково главное условие выполнения атаки, связанной с мультипликативным свойством шифра RSA?
5. Почему с увеличением модуля криптосистемы время выполнения циклической атаки возрастает экспоненциально?

Выводы по главе

В данной главе были представлены методические указания к лабораторной работе, составленные на основе разработанной программы.

В методических указаниях описаны цели и задачи лабораторной работы, и приведена последовательность ее выполнения. Также приведен пример выполнения отчета по лабораторной работе.

В отчет рекомендуется записать примеры с небольшими длинами ключей и сообщений, при этом разработанная программа позволяет убедиться в скорости выполнения атак даже при большой длине ключа и осознать, насколько опасны данные атаки.

Специально созданный «дружественный» графический интерфейс позволит лучше понять последовательность выполняемых действий и упростит процесс изучения причин возникновения побочных атак и сценариев их использования.

Лабораторный комплекс адаптирован для использования в учебном процессе при изучении дисциплины «Основы криптографии с открытым ключом». Дополнительный теоретический материал, необходимый для выполнения лабораторной работы, содержится в справочной информации к программному комплексу. Лабораторная работа рассчитана на выполнение в рамках стандартного практического занятия, занимающего по продолжительности два академических часа.

Заключение

Целью данной дипломной работы являлась разработка лабораторной работы для изучения побочных атак на криптосистему RSA.

Криптосистема RSA имеет важное значение в изучении курса криптосистем с открытым ключом. Являясь первой криптосистемой, рассматриваемой в разделе КОК, она довольно проста для понимания, но в то же время активно применяется в настоящее время для шифрования и цифровой подписи. Наличие в курсе данной лабораторной работы позволит студентам лучше разобраться в криптоанализе шифра RSA, рассмотрев выполнение алгоритмов побочных атак на практическом примере.

Для разработки лабораторной работы были подробно изучены сама криптосистема RSA и побочные атаки на данную криптосистему.

Интересным выводом из первой главы является то, что в современных стандартах предписывается вычисление обратной экспоненты по модулю функции Кармайкла, а не функции Эйлера, о чем для простоты изучения почти никогда не упоминается в учебных пособиях по криптографии. Хотя и доказательство справедливости алгоритма атаки Винера было приведено для случая генерирования обратной экспоненты с использованием функции Кармайкла, в разработанной программе оба метода генерирования обратной экспоненты были реализованы в модуле для демонстрации атаки Винера, и в обоих случаях успешность атаки зависит только от битовой длины секретной экспоненты относительно модуля. То есть алгоритм, разработанный Винером находит ту секретную экспоненту из возможных, которая является малой при заданных условиях, независимо от метода ее вычисления при генерировании ключей.

Во второй главе дипломной работы были приведены корректные доказательства работы побочных атак, представлены четкие алгоритмы их выполнения, приведены примеры, а также предложен метод Ньютона в качестве алгоритма, используемого для нахождения целочисленного корня n -

ой степени. Вторая глава явилась основой для разработки программы, реализующей вышеупомянутые алгоритмы.

Третья глава описывает особенности разработки кроссплатформенного программного обеспечения с дружественным графическим интерфейсом. Были использованы свободные C++ библиотеки: MPFR для длинной арифметики и wxWidgets для графического интерфейса. Интерфейс программы поддерживает интернационализацию без перекомпилирования исходных кодов благодаря системе каталогов сообщений gettext (в рамках дипломной работы в программе была реализована поддержка английского и русского языков). Выбранный язык программирования позволяет разработанной программе производить большие вычисления максимально быстро, что немаловажно при моделировании асимметричных криптосистем, скорость работы которых ниже, чем у симметричных.

В итоге был разработан полноценный программный комплекс, готовый к использованию при выполнении практической работы по теме «Побочные атаки на криптосистему RSA». Он содержит все необходимые средства для наглядной демонстрации работы побочных атак, отображая большинство промежуточных вычислений. Возможность генерирования ключей с задаваемой длиной ключа позволит учащимся убедиться в том, что большинство атак выполняется очень быстро даже при больших длинах модуля криптосистемы.

Список литературы:

1. В.И. Коржик, В.П. Просихин, Основы криптографии / СПб.:СПБГУТ - 2008
2. Menezes, A.J. et all. Handbook of Applied Cryptography. - New-York, London, Tokyo : CRC Press, 1996
3. W. Diffie, M. Hellman, New directions in cryptography. IEEE Trans. Inform. Theory IT-22, (Nov. 1976), 644-654.
4. R. Rivest, A. Shamir, L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 21 (2), pp. 120-126, February 1978
5. PKCS #1: RSA Cryptography Standard v2.2, <http://www.emc.com/emc-plus/rsa-labs/pkcs/files/h11300-wp-pkcs-1v2-2-rsa-cryptography-standard.pdf>
6. RFC 3447, <http://www.ietf.org/rfc/rfc3447.txt>
7. H. Riesel. Prime numbers and Computer Methods for Factorization: Second Edition / Springer Science+Business Media - 2012
8. D. Boneh, Twenty Years of Attacks on the RSA Cryptosystem / Notices of the AMS, 46(2):203-213, 1999
9. T. Granlund, GMP Development Team, The Multiple Precision Integers and Rationals Library manual: Edition 2.6.0 / 2012
10. K. Atkinson, An introduction to numerical analysis: Second Edition / John Wiley & Sons, Inc - 1989
11. http://e-maxx.ru/algo/roots_newton
12. M. Wiener, Cryptanalysis of short RSA secret exponents, IEEE Trans. Inform.Theory 36 (1990), 553–558.
13. J. DeLaurentis, A further weakness in the common modulus protocol for the RSA cryptalgorithm / Cryptologia Volume 8, Number 3 – 1984, 253-259
14. J. Smart, K. Hock, Cross-Platform GUI Programming with wxWidgets / Prentice Hall – 2005

15. Н. К. Логвинова, В. Я. Павлов, А. В. Алексеев, Дипломная работа - методические указания к выполнению, СПбГУТ, 2004

Приложение 1. Классы криптосистем

Листинг 1.1: RsaBase.h

```
#pragma once

#include <mpir.h>
#include <mpirxx.h>
#include <iostream>

class RsaBase
{
public:
    typedef struct _pubKey
    {
        mpz_class n;
        mpz_class e;
    }pubKey;

    typedef struct _privKey
    {
        mpz_class p;
        mpz_class q;
        mpz_class d;
    }privKey;

    pubKey m_PubKey;
    privKey m_PrivKey;

    mpz_class primeLen;
    gmp_randclass * pGenerator;

    mpz_class GetRandomPrime(mpz_class primeLen);
    virtual void GenerateKeys() = 0;

    void Encode(mpz_class &cyphertext, mpz_class &plaintext);
    void Decode(mpz_class &plaintext, mpz_class &cyphertext);

    RsaBase(gmp_randclass *randclass, int primeLength);
    ~RsaBase(void);
};
```

Листинг 1.2: RsaBase.cpp

```
#include "RsaBase.h"

using namespace std;

RsaBase::RsaBase(gmp_randclass *randclass, int primeLength)
{
    pGenerator = randclass;
    primeLen = primeLength / 2;
}

RsaBase::~RsaBase()
{
}

mpz_class RsaBase::GetRandomPrime(mpz_class primeLen)
{
    mpz_class temp;
    do
```

```

        {
            temp = pGenerator->get_z_bits(primeLen);
        } while (mpz_probab_prime_p(temp.get_mpz_t(), 10) == 0);
        return temp;
    }

void RsaBase::Encode(mpz_class &cyphertext, mpz_class &plaintext)
{
    if (plaintext > m_PubKey.n)
    {
        cout << "Plaintext is larger than modulus N" << endl;
    }
    mpz_powm(cyphertext.get_mpz_t(), plaintext.get_mpz_t(),
m_PubKey.e.get_mpz_t(), m_PubKey.n.get_mpz_t());
}

void RsaBase::Decode(mpz_class &plaintext, mpz_class &cyphertext)
{
    mpz_powm(plaintext.get_mpz_t(), cyphertext.get_mpz_t(),
m_PrivKey.d.get_mpz_t(), m_PubKey.n.get_mpz_t());
}

```

Листинг 1.3: RsaCommon.h

```

#pragma once

#include "RsaBase.h"

class RsaCommon : public RsaBase
{
public:
    RsaCommon(gmp_randclass *randclass, int primeLength);
    ~RsaCommon();

    void GenerateKeys();
private:
};

```

Листинг 1.4 RsaCommon.cpp

```

#include "RsaCommon.h"

RsaCommon::RsaCommon(gmp_randclass *randclass, int primeLength) :
RsaBase(randclass, primeLength)
{
    GenerateKeys();
}

RsaCommon::~~RsaCommon()
{
}

void RsaCommon::GenerateKeys()
{
    mpz_class phi_n;
    m_PrivKey.p = GetRandomPrime(primeLen);
    m_PrivKey.q = GetRandomPrime(primeLen);
    m_PubKey.n = m_PrivKey.p * m_PrivKey.q;
    phi_n = (m_PrivKey.p - 1) * (m_PrivKey.q - 1);
    do
    {
        m_PubKey.e = pGenerator->get_z_range(phi_n);
    }
}

```

```

        } while (m_PubKey.e < 5 || mpz_invert(m_PrivKey.d.get_mpz_t(),
m_PubKey.e.get_mpz_t(), phi_n.get_mpz_t()) == 0);
        ~phi_n;
    }

```

Листинг 1.5 RsaDefinedModulus.h

```

#pragma once

#include "RsaBase.h"

class RsaDefinedModulus : public RsaBase
{
public:
    RsaDefinedModulus(gmp_randclass *randclass, int primeLength, mpz_class
&p, mpz_class &q);
    ~RsaDefinedModulus();

    void GenerateKeys();
};

```

Листинг 1.6 RsaDefinedModulus.cpp

```

#include "RsaDefinedModulus.h"

RsaDefinedModulus::RsaDefinedModulus(gmp_randclass *randclass, int
primeLength, mpz_class &p, mpz_class &q) : RsaBase(randclass, primeLength)
{
    m_PrivKey.p = p;
    m_PrivKey.q = q;
    m_PubKey.n = m_PrivKey.p * m_PrivKey.q;

    mpz_class phi_n;
    phi_n = (m_PrivKey.p - 1) * (m_PrivKey.q - 1);
    do
    {
        m_PubKey.e = pGenerator->get_z_range(phi_n);
    } while (m_PubKey.e < 5 || mpz_invert(m_PrivKey.d.get_mpz_t(),
m_PubKey.e.get_mpz_t(), phi_n.get_mpz_t()) == 0);
    ~phi_n;
}

RsaDefinedModulus::~RsaDefinedModulus()
{
}

void RsaDefinedModulus::GenerateKeys()
{
}

```

Листинг 1.7 RsaHastad.h

```

#pragma once

#include "RsaBase.h"

class RsaHastad : public RsaBase
{
public:
    RsaHastad(gmp_randclass *randclass, int primeLength);
    ~RsaHastad();

    void GenerateKeys();
}

```



```
private:
};
```

Листинг 1.8 RsaHastad.cpp

```
#include "RsaHastad.h"

RsaHastad::RsaHastad(gmp_randclass *randclass, int primeLength) :
RsaBase(randclass, primeLength)
{
    GenerateKeys();
}

RsaHastad::~RsaHastad()
{
}

void RsaHastad::GenerateKeys()
{
    m_PubKey.e = 3;

    mpz_class phi_n;
    mpz_class gcdTemp;

    do
    {
        m_PrivKey.p = GetRandomPrime(primeLen) - 1; // -1 for gcd test
        purposes
        mpz_gcd(gcdTemp.get_mpz_t(), m_PrivKey.p.get_mpz_t(),
m_PubKey.e.get_mpz_t());
    } while (gcdTemp != 1);
    m_PrivKey.p = m_PrivKey.p + 1; // +1 to get original prime after test

    ~gcdTemp;

    do
    {
        m_PrivKey.q = GetRandomPrime(primeLen);
        phi_n = (m_PrivKey.p - 1) * (m_PrivKey.q - 1);
    } while (mpz_invert(m_PrivKey.d.get_mpz_t(), m_PubKey.e.get_mpz_t(),
phi_n.get_mpz_t()) == 0); // while "==" 0" means inverse does not exist
    m_PubKey.n = m_PrivKey.p * m_PrivKey.q;

    ~phi_n;
}
}
```

Листинг 1.9 RsaWiener.h

```
#pragma once

#include "RsaBase.h"

enum inv_exp_method
{
    RSA_USE_EULER_TOTIENT,
    RSA_USE_CARMICHAEL
};

class RsaWiener : public RsaBase
{
public:
```

```

        RsaWiener(gmp_randclass* randclass, int primeLength, int d_length,
inv_exp_method inv_exp);
        ~RsaWiener();

        int d_len;
        inv_exp_method my_inv_exp;

        void GenerateKeys();
private:
};

```

Листинг 1.10 RsaWiener.cpp

```

#include "RsaWiener.h"

RsaWiener::RsaWiener(gmp_randclass* randclass, int primeLength, int d_length,
inv_exp_method inv_exp) : RsaBase(randclass, primeLength)
{
    d_len = d_length;
    my_inv_exp = inv_exp;
    GenerateKeys();
}

RsaWiener::~RsaWiener()
{
}

void RsaWiener::GenerateKeys()
{
    mpz_class mod_for_inv;

    m_PrivKey.p = GetRandomPrime(primeLen);
    m_PrivKey.q = GetRandomPrime(primeLen);
    m_PubKey.n = m_PrivKey.p * m_PrivKey.q;

    switch (my_inv_exp)
    {
    case RSA_USE_EULER_TOTIENT:
        mod_for_inv = (m_PrivKey.p - 1) * (m_PrivKey.q - 1);
        break;
    case RSA_USE_CARMICHAEL:
        m_PrivKey.p = m_PrivKey.p - 1;
        m_PrivKey.q = m_PrivKey.q - 1;
        mpz_lcm(mod_for_inv.get_mpz_t(), m_PrivKey.p.get_mpz_t(),
m_PrivKey.q.get_mpz_t());
        m_PrivKey.p = m_PrivKey.p + 1;
        m_PrivKey.q = m_PrivKey.q + 1;
        break;
    }

    do
    {
        m_PrivKey.d = pGenerator->get_z_bits(d_len);
    } while (mpz_invert(m_PubKey.e.get_mpz_t(), m_PrivKey.d.get_mpz_t(),
mod_for_inv.get_mpz_t()) == 0);
    ~mod_for_inv;
}

```

Приложение 2. Реализация алгоритмов побочных атак

Листинг 2.1: Реализация алгоритма атаки на малую открытую экспоненту (фрагмент файла GUI_HastadPanel.cpp)

```
void GUI_HastadPanel::OnCRTSolClicked( wxCommandEvent& event )
{
    if (CRT_Sol == NULL)
    {
        CRT_Sol = new mpz_class;
    }
    *CRT_Sol = 0;

    mpz_class N(1);
    for (size_t i = 0; i < vec_RSA.size(); i++) // N = mul (ni)
    {
        N = N * vec_RSA.at(i)->m_PubKey.n;
    }

    mpz_class NiTemp, Ni_invTemp;
    for (size_t i = 0; i < vec_RSA.size(); i++)
    {
        NiTemp = N / vec_RSA.at(i)->m_PubKey.n;
        mpz_invert(Ni_invTemp.get_mpz_t(), NiTemp.get_mpz_t(),
vec_RSA.at(i)->m_PubKey.n.get_mpz_t());
        *CRT_Sol = (*CRT_Sol + *vec_C.at(i) * NiTemp * Ni_invTemp) % N;
//x = summ(C*Mi*Mi_1) mod M
    }
    ~N;
    ~NiTemp;
    ~Ni_invTemp;
    m_CRTsolText->SetValue(CRT_Sol->get_str());
    m_nthRottButton->Enable(true);
}

void GUI_HastadPanel::OnNthRootClicked( wxCommandEvent& event )
{
    if (NthRoot == NULL)
    {
        NthRoot = new mpz_class;
    }
    mpz_nthroot(NthRoot->get_mpz_t(), CRT_Sol->get_mpz_t(), vec_RSA.size());
    m_nthRootText->SetValue(NthRoot->get_str());
}
}
```

Листинг 2.2: Реализация алгоритма атаки при малом числе возможных сообщений (фрагмент файла GUI_KnownMsgsPanel.cpp)

```
void GUI_KnownMsgsPanel::OnFindClicked( wxCommandEvent& event )
{
    mpz_class tempCipher;
    time_t endTime = time(NULL) + LIMIT_SECONDS;
    for (size_t i = 0; i < vec_M.size(); i++)
    {
        CommonRsaInst->Encode(tempCipher, *vec_M.at(i));
        if (tempCipher == *C)
        {
            m_RandMsgList->SetSelection(i);
            m_textChosenMsg->SetValue(vec_M.at(i)->get_str());
            wxMessageBox(wxString::Format(_("Given ciphertext
corresponds to possible message #i"), i + 1), wxT("Info"),
```

```

        wxOK | wxICON_INFORMATION, this);
        ~tempCipher;
        return;
    }
    if (time(NULL) > endTime)
    {
        wxMessageBox(wxString::Format(_("Failed to find message in
%i seconds using linear search! Aborted.\nGenerate a new list of possible
messages with lower number of messages or decrease the size of RSA
modulus."), LIMIT_SECONDS), wxT("Info"),
        wxOK | wxICON_INFORMATION, this);
        ~tempCipher;
        return;
    }
    wxMessageBox(_("None of given possible messages corresponds to given
ciphertext"), wxT("Info"),
        wxOK | wxICON_INFORMATION, this);
    ~tempCipher;
}

```

**Листинг 2.3: Реализация алгоритма атаки на малую секретную экспоненту (фрагмент
файла GUI_WienersPanel.cpp)**

```

void GUI_WienersPanel::OnWienerAttackClicked( wxCommandEvent& event )
{
    unsigned int i = 0;
    mpz_class qi, ri_num, ri_denom, temp_ri_denom;
    mpz_class ni, di, ni_1, di_1, ni_2, di_2;
    mpz_class guessOf_k, guessOf_dg;
    mpz_class guessOf_p_plus_q_2, guessOf_p_minus_q_2;
    mpz_class guess_temp, guessOf_g;

    do
    {
        if (i == 0)
        {
            qi = RsaWienIns->m_PubKey.e / RsaWienIns->m_PubKey.n;
            ri_num = RsaWienIns->m_PubKey.e % RsaWienIns->m_PubKey.n;
            ri_denom = RsaWienIns->m_PubKey.n;

            ni = qi;
            di = 1;

            // qi is replaced with qi+1, because zero is odd
            guessOf_k = qi + 1;
            guessOf_dg = di;
        }
        else if (i == 1)
        {
            ni_1 = ni;
            di_1 = di;
            temp_ri_denom = ri_num;
            //mpz_cdiv_qr(qi.get_mpz_t(), ri_num.get_mpz_t(),
            ri_denom.get_mpz_t(), ri_num.get_mpz_t());
            qi = ri_denom / ri_num;
            ri_num = ri_denom % ri_num;
            ri_denom = temp_ri_denom;

            ni = ni*qi + 1; //n1=q0*q1+1, where q0=n0
            di = qi;
        }
    }
}

```

```

// No modification of convergent here, because 1 is odd.
guessOf_k = ni;
guessOf_dg = qi;
}
else
{
    ni_2 = ni_1;
    di_2 = di_1;
    ni_1 = ni;
    di_1 = di;
    temp_ri_denom = ri_num;
    //mpz_cdiv_qr(qi.get_mpz_t(), ri_num.get_mpz_t(),
ri_denom.get_mpz_t(), ri_num.get_mpz_t());
    qi = ri_denom / ri_num;
    ri_num = ri_denom % ri_num;
    ri_denom = temp_ri_denom;

    ni = qi*ni_1 + ni_2;
    di = qi*di_1 + di_2;

    if (!(i % 2)) // x is even
    {
        guessOf_k = (qi + 1)*ni_1 + ni_2;
        guessOf_dg = (qi + 1)*di_1 + di_2;
    }
    else
    {
        guessOf_k = ni;
        guessOf_dg = di;
    }
}

guess_temp = guessOf_dg * RsaWienIns->m_PubKey.e; // guess of edg
e*dg
guessOf_g = guess_temp % guessOf_k; // guess of g = edg mod k
guess_temp = guess_temp / guessOf_k; // guess of (p-1)(q-1)
if ((RsaWienIns->m_PubKey.n + guess_temp + 1) % 2 != 0) // (p+q)/2
is not integer
{
    i++;
    continue;
};
guess_temp = guessOf_p_plus_q_2 = (RsaWienIns->m_PubKey.n -
guess_temp + 1) / 2; // guess of (p+q)/2 = (n - (p-1)(q-1) + 1)/2
guess_temp = guess_temp * guess_temp - RsaWienIns->m_PubKey.n; //
guess of ((p-q)/2)^2 = ((p+q)/2)^2 - n
if (mpz_perfect_square_p(guess_temp.get_mpz_t()) == 0) // ((p-
q)/2)^2 is not perfect square
{
    i++;
    continue;
}
mpz_sqrt(guessOf_p_minus_q_2.get_mpz_t(), guess_temp.get_mpz_t());
mpz_class found_d, found_p, found_q;
found_d = guessOf_dg / guessOf_g;
found_p = guessOf_p_plus_q_2 + guessOf_p_minus_q_2;
found_q = guessOf_p_plus_q_2 - guessOf_p_minus_q_2;

m_text_i->SetValue(wxString::Format(wxT("%d"), i));
m_text_k->SetValue(guessOf_k.get_str());
m_text_g->SetValue(guessOf_g.get_str());
m_textFound_d->SetValue(found_d.get_str());

```

```

        m_textFound_p->SetValue(found_p.get_str());
        m_textFound_q->SetValue(found_q.get_str());
        ~found_d; ~found_p; ~found_q;
        //wxMessageBox(_("Success!"), wxT("Info"),
        //      wxOK | wxICON_INFORMATION, this);
        break;
    } while (ri_num != 0);
    if (ri_num == 0)
    {
        m_text_i->SetValue(wxString::Format(wxT("%d"), i));
        m_text_k->SetValue(RESULT_NA);
        m_text_g->SetValue(RESULT_NA);
        m_textFound_d->SetValue(RESULT_NA);
        m_textFound_p->SetValue(RESULT_NA);
        m_textFound_q->SetValue(RESULT_NA);
        wxMessageBox(_("Failed to apply Wiener's attack!"), wxT("Info"),
            wxOK | wxICON_INFORMATION, this);
    }

    ~qi, ~ri_num, ~ri_denom, ~temp_ri_denom;
    ~ni, ~di, ~ni_1, ~di_1, ~ni_2, ~di_2;
    ~guessOf_k, ~guessOf_dg;
    ~guessOf_p_plus_q_2, ~guessOf_p_minus_q_2;
    ~guess_temp, ~guessOf_g;

    m_ShowKeys->Enable(true);
}

```

Листинг 2.4: Реализация алгоритма атаки с использованием мультипликативного свойства шифра РША (фрагмент файла GUI_MultiplicativePanel.cpp)

```

void GUI_MultiplicativePanel::OnGenerateXClicked( wxCommandEvent& event )
{
    GUI_MainFrame * Myparent = wxDynamicCast(GetParent()->GetParent(),
    GUI_MainFrame);
    if (BlindingRsaInst == NULL)
    {
        wxMessageBox(_("You should generate keys first"), wxT("Info"),
            wxOK | wxICON_INFORMATION, this);
        return;
    }
    if (x == NULL)
    {
        x = new mpz_class;
    }
    mpz_class tempInvert;
    do
    {
        *x = Myparent->MyRndGen.myRandclass->get_z_range(BlindingRsaInst->
        m_PubKey.n);
    } while (mpz_invert(tempInvert.get_mpz_t(), x->get_mpz_t()),
    BlindingRsaInst->m_PubKey.n.get_mpz_t()) == 0); // == 0 means inverse does
    not exist, generate next number
    ~tempInvert;
    m_textX->SetValue(x->get_str());
    m_calcCustC->Enable(true);
}

void GUI_MultiplicativePanel::OnCalcCcClicked( wxCommandEvent& event )
{
    if (CustC == NULL)
    {

```

```

        CustC = new mpz_class;
    }
    mpz_powm(CustC->get_mpz_t(), x->get_mpz_t(), BlindingRsaInst-
>m_PubKey.e.get_mpz_t(), BlindingRsaInst->m_PubKey.n.get_mpz_t()); //  $x^e \bmod n$ 
    *CustC = *cyphertext * *CustC % BlindingRsaInst->m_PubKey.n; //
C'=(C*x^e) mod n
    m_textCtrl20->SetValue(CustC->get_str());
    m_DecryptCustC->Enable(true);
}

void GUI_MultiplicativePanel::OnDecryptCcClicked( wxCommandEvent& event )
{
    if (CustM == NULL)
    {
        CustM = new mpz_class;
    }
    BlindingRsaInst->Decode(*CustM, *CustC);
    m_textCtrl21->SetValue(CustM->get_str());
    m_CalcM->Enable(true);
}

void GUI_MultiplicativePanel::OnCalcMCClicked( wxCommandEvent& event )
{
    if (M == NULL)
    {
        M = new mpz_class;
    }
    mpz_invert(M->get_mpz_t(), x->get_mpz_t(), BlindingRsaInst-
>m_PubKey.n.get_mpz_t()); //  $x^{-1} \bmod n$ 
    *M = *CustM * *M % BlindingRsaInst->m_PubKey.n;
    m_textCtrl22->SetValue(M->get_str());
}

```

Листинг 2.5: Реализация алгоритма циклической атаки (фрагмент файла

GUI_CyclicPanel.cpp)

```

void GUI_CyclicPanel::OnFindFixedPointClicked( wxCommandEvent& event )
{
    mpz_class C_ = *C;
    mpz_class k, found_M;
    time_t endTime = time(NULL) + LIMIT_SECONDS;
    do
    {
        found_M = C_;
        RsaInst->Encode(C_, C_);
        k++;
    } while (C_ != *C && time(NULL) < endTime);
    if (time(NULL) < endTime)
    {
        m_text_k->SetValue(k.get_str());
        m_textFoundM->SetValue(found_M.get_str());
    }
    else
    {
        m_text_k->SetValue(RESULT_NA);
        m_textFoundM->SetValue(RESULT_NA);
        wxMessageBox(wxString::Format(_("Failed to apply Cyclic attack in
%i seconds!"), LIMIT_SECONDS), wxT("Info"),
            wxOK | wxICON_INFORMATION, this);
    }
    ~k; ~C_; ~found_M;
}

```

}

Листинг 2.6: Реализация алгоритма атаки на общие модули (фрагмент файла

GUI_CmnModAttack.cpp)

```

void GUI_CmnModAttack::OnFactorModClicked( wxCommandEvent& event )
{
    GUI_MainFrame * Myparent = wxDynamicCast(GetParent()->GetParent(),
    GUI_MainFrame);
    mpz_class g;
    mpz_class powm_temp, powm_temp_prev;
    bool successfulFactoring = false;

    mpz_class k = firstRsa->m_PubKey.e * firstRsa->m_PrivKey.d - 1; // k =
ed-1
    unsigned int t = 0;
    do
    {
        t++;
    } while (mpz_divisible_2exp_p(k.get_mpz_t(), t)); // while k is
divisible by 2 2^t
    t--;
    mpz_cdiv_q_2exp(k.get_mpz_t(), k.get_mpz_t(), t); // divides k by 2^t

    while (!successfulFactoring)
    {
        do
        {
            g = Myparent->MyRndGen.myRandclass->get_z_range(firstRsa-
>m_PubKey.n - 1);
        } while (g < 2); // choose random g such that 1 < g < n-1

        mpz_powm(powm_temp.get_mpz_t(), g.get_mpz_t(), k.get_mpz_t(),
firstRsa->m_PubKey.n.get_mpz_t()); // g^k mod n, where k is odd
        if (powm_temp == 1)
        {
            //cout << "g^k mod n = 1, algorithm failed with randomly
chosen g." << endl;
            continue;
        }

        for (unsigned int i = 1; i <= t; i++)
        {
            powm_temp_prev = powm_temp;
            if (powm_temp_prev == firstRsa->m_PubKey.n - 1)
            {
                //cout << "g^(k*2^t) mod n = n - 1, algorithm failed
with randomly chosen g." << endl;
                break;
            }
            //cout << "Previous pow was: " << powm_temp_prev << endl;
            mpz_powm(powm_temp.get_mpz_t(), powm_temp_prev.get_mpz_t(),
mpz_class(2).get_mpz_t(), firstRsa->m_PubKey.n.get_mpz_t());
            //cout << "Current pow is: " << powm_temp << endl;

            if (powm_temp == 1)
            {
                if (factored_p1 == NULL)
                {
                    factored_p1 = new mpz_class;
                }
                if (factored_q1 == NULL)

```



```

        {
            factored_q1 = new mpz_class;
        }
        powm_temp_prev = powm_temp_prev - 1; // nontrivial
solutuion minus 1
            mpz_gcd(factored_p1->get_mpz_t(),
powm_temp_prev.get_mpz_t(), firstRsa->m_PubKey.n.get_mpz_t()); // gcd (x-1
,n)
            powm_temp_prev = powm_temp_prev + 2; // nontrivial
solutuion plus 1
            mpz_gcd(factored_q1->get_mpz_t(),
powm_temp_prev.get_mpz_t(), firstRsa->m_PubKey.n.get_mpz_t()); // gcd (x+1
,n)
            powm_temp_prev = powm_temp_prev - 1; // nontrivial
solutuion itself
            //cout << "g^(k*2^" << t << ") mod n = 1, so x =
g^(k*2^" << t - 1 << ") mod n" << endl; // nontrivial x
            m_staticText35-
>SetLabelText(wxString::Format(_("g^(k*2^%i) mod n is equal to 1,\nwhile x =
g^(k*2^%i) mod n is not equal to 1 or -1,\nwhere g:"), i, i - 1));
            m_staticText36->SetLabelText(wxString::Format(_("and
odd number k = (e*d-1) / 2^%i:"), t));
            m_text_rnd_g->SetLabelText(g.get_str());
            m_text_k->SetLabelText(k.get_str());
            m_text_p->SetLabelText(factored_p1->get_str());
            m_text_q->SetLabelText(factored_q1->get_str());
            successfulFactoring = true;
            m_buttonCalcD->Enable(true);
            break;
        }
    }
}
~g; ~powm_temp; ~powm_temp_prev; ~k;
}

void GUI_CmnModAttack::OnCalcDClicked( wxCommandEvent& event )
{
    mpz_class phi = (*factored_p1 - 1) * (*factored_q1 - 1);
    mpz_class d2;
    mpz_invert(d2.get_mpz_t(), secondRsa->m_PubKey.e.get_mpz_t(),
phi.get_mpz_t());
    m_text_2nd_d->SetLabelText(d2.get_str());
    ~phi; ~d2;
}

```