МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1—98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ОТЧЕТ ПО ПРАКТИЧЕСКИМ ЗАНЯТИЯМ**

по дисциплине «Основы информационной безопасности»

вариант 10

**Исполнитель**

студент(ка) 2 курса \_7\_ группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Банкузов Михаил

подпись, дата

**Руководитель**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ржеутская Н. В.

должность, ученая степень, ученое звание подпись, дата

**Практическое задание № 1**

**Тема «Решение задачи разработки средств защиты для обеспечения максимальной эффективности объекта в условиях несанкционированного доступа»**

Цель: научиться решать задачи разработки средств защиты для обеспечения максимальной эффективности объекта в условиях несанкционированного доступа.

**Теоретическое введение**

Все методы защиты информации по характеру проводимых действий можно разделить на:

– законодательные (правовые);

– организационные;

– технические;

– комплексные.

Принципиальным вопросом при определении уровня защищенности объекта является выбор критериев. Рассмотрим один из них ‑ широко известный критерий "эффективность - стоимость".

Пусть имеется информационный объект, который при нормальном (идеальном) функционировании создает положительный эффект (экономический, политический, технический и т.д.). Этот эффект обозначим через *Е0*. Несанкционированный доступ к объекту уменьшает полезный эффект от его функционирования (нарушается нормальная работа, наносится ущерб из-за утечки информации и т.д.) на величину *ΔЕ*. Тогда эффективность функционирования объекта с учетом воздействия несанкционированного доступа:

|  |  |
| --- | --- |
|  | (1) |

Относительная эффективность:

|  |  |
| --- | --- |
|  | (2) |

Уменьшение эффективности функционирования объекта приводит к материальному ущербу для владельца объекта. В общем случае материальный ущерб есть некоторая неубывающая функция от ΔЕ:

|  |  |
| --- | --- |
|  |  |

Будем считать, что установка на объект средств защиты информации уменьшает негативное действие несанкционированного доступа на эффективность функционирования объекта. Обозначим снижение эффективности функционирования объекта при наличии средств защиты через ΔЕ3, а коэффициент снижения негативного воздействия несанкционированного доступа на эффективность функционирования объект ‑ через К, тогда:

|  |  |
| --- | --- |
|  |  |

где К≥1.

Выражения (1) – (2) примут вид:

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  |  |

Поскольку затраты на установку средств защиты можно рассматривать как ущерб владельцу объекта от возможности осуществления несанкционированного доступа, то суммарный ущерб объекту:

|  |  |
| --- | --- |
|  |  |

Если эффективность функционирования объекта имеет стоимостное выражение (доход, прибыль и т.д.), то UΣ непосредственно изменяет эффективность:

|  |  |
| --- | --- |
|  |  |

**Условие задания**

Решить задачу разработки средств защиты для обеспечения максимальной эффективности объекта в условиях несанкционированного доступа в соответствии с вариантом.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № варианта | *E*0 | *E* | *K* | *C* |
| 2 | 100000 | 90000 | 5 | 5000 |

**Исполнительная часть**

0,9



0,93

**Вывод:** Я научился решать задачи разработки средств защиты для обеспечения максимальной эффективности объекта в условиях несанкционированного доступа и выяснила, что с данными значениями моего варианта защита эффективна.

**Практическое задание № 2**

**Тема «Разработка политики информационной безопасности бизнес-компании»**

Цель: Разработать проект политики информационной безопасности бизнес-компании.

**Введение**

Основой мер **административного уровня,** то есть мер, предпринимаемых руководством организации, является политика безопасности.

Под **политикой безопасности** понимается совокупность документированных управленческих решений, направленных на защиту информации и ассоциированных с ней ресурсов.

Политика безопасности определяет стратегию организации в области информационной безопасности, а также ту меру внимания и количество ресурсов, которую руководство считает целесообразным выделить.

Определение политики ИБ должно сводиться к следующим практическим шагам:

1. Определение используемых руководящих документов и стандартов в области ИБ, а также основных положений политики ИБ, включая:

• управление доступом к средствам вычислительной техники, программа и данным;

• антивирусную защиту;

• вопросы резервного копирования;

• проведение ремонтных и восстановительных работ;

• информирование об инцидентах об области ИБ.

2. Определение подходов к управлению рисками: является ли достаточным базовый уровень защищенности или требуется проводить полный вариант анализа рисков.

3. Структуризация контрмер по уровням.

4. Порядок сертификации на соответствие стандартам в области ИБ. Должна быть определена периодичность проведения совещаний по тематике ИБ на уровне руководства, включая периодический пересмотр положений политики ИБ, а также порядок обучения всех категорий пользователей информационной системы по вопросам ИБ.

Для построения системы защиты информации необходимо определить границы системы, для которой должен быть обеспечен режим информационной безопасности. Соответственно система управления информационной безопасности (система защиты информации) должна строиться именно в этих границах.

Описание границ системы, для которой должен быть обеспечен режим информационной безопасности, рекомендуется выполнять по следующему плану.

1. Структура организации. Описание существующей структуры и изменений, которые предполагается внести в связи с разработкой или модернизации автоматизированной системы обработки информации.

2. Размещение средств вычислительной техники и поддерживающей инфраструктуры. Модель иерархии средств вычислительной техники.

3. Ресурсы информационной системы, подлежащие защите. Рекомендуется рассмотреть ресурсы автоматизированной системы следующих классов: средства вычислительной техники, данные, системное и прикладное программное обеспечение. Все ресурсы представляют ценность с точки зрения организации. Для их оценки должна быть выбрана система критериев и методология оценок по этим критериям.

4. Технология обработки информации и решаемые задачи. Для решаемых задач должны быть построены модели обработки информации в терминах ресурсов.

В результате должен быть составлен документ, в котором:

• зафиксированы границы и структура системы;

• перечислены ресурсы, подлежащие защите;

• дана система критериев для оценки их ценности.

Минимальным требованиям к режиму информационной безопасности соответствует базовый уровень. Обычной областью использования этого уровня являются типовые проектные решения. Существует ряд стандартов и спецификаций, в которых рассматривается минимальный (типовой) набор наиболее вероятных угроз, таких как вирусы, сбои оборудования, несанкционированный доступ и т.д. Для нейтрализации этих угроз обязательно должны быть приняты контрмеры вне зависимости от вероятности осуществления угроз и уязвимости ресурсов. Таким образом, характеристики угроз на базовом уровне рассматривать не обязательно.

В случае, когда нарушения информационной безопасности чреваты тяжелыми последствиями, базовый уровень требований к режиму информационной безопасности является недостаточным. Для того, чтобы сформулировать дополнительные требования, необходимо:

• определить ценность ресурсов;

• к стандартному набору добавить список угроз, актуальных для исследуемой информационной системы;

• оценить вероятности угроз;

• определить уровень уязвимости ресурсов.

Политика безопасности строится на основе анализа рисков, которые признаются реальными для информационной системы организации. Когда риски проанализированы, стратегия защиты определена, тогда составляется программа, реализация которой должна обеспечить информационную безопасность. Под эту программу выделяются ресурсы, назначаются ответственные, определяется порядок контроля выполнения программы и т.п.

Существуют различные подходы к оценке рисков. Выбор под хода зависит от уровня требований, предъявляемых в организации к режиму информационной безопасности, характера принимаемых во внимание угроз (спектра воздействия угроз) и эффективности потенциальных контрмер.

Процесс оценивая рисков содержит несколько этапов.

1. Идентификация ресурса и оценивание его количественных показателей (определение негативного воздействия).

2. Оценивание угроз.

3. Оценивание уязвимостей.

4. Оценивание существующих и предполагаемых средств обеспечения.

5. Оценивание рисков.

На основании оценивания рисков выбираются средства, обеспечивающие режим ИБ. Ресурсы, значимые для нормальной работы организации и имеющие определенную степень уязвимости, считаются подверженными риску, если по отношению к ним существует какая либо угроза. При оценивании рисков учитываются потенциальные негативные воздействия от нежелательных происшествий и показатели значимости рассматриваемых уязвимостей и угроз для этих ресурсов.

Риск характеризует опасность, которой может подвергаться система и использующая ее организация. Риск зависит от показателей ценности ресурсов, вероятности реализации угроз для ресурсов и степени легкости, с которой уязвимости могут быть использованы при существующих или планируемых средствах обеспечения информационной безопасности.

Цель оценивания рисков состоит в определении характеристик рисков для информационной системы и ее ресурсов. На основе таких данных могут быть выбраны необходимые средства управления ИБ.

При оценивании рисков учитывается:

• ценность ресурсов;

• оценка значимости угроз;

• эффективность существующих и планируемых средств защиты.

Показатели ресурсов или потенциальное негативное воздействие на деятельность организации можно определять несколькими способами:

• количественными (например, стоимостные);

• качественными (могут быть построены на использовании таких понятий, как, умеренный или чрезвычайно опасный);

• их комбинацией.

Для того, чтобы конкретизировать определение вероятности в течение которого предполагается защитить ресурс. Вероятность того, что угроза реализуется, определяется следующими факторами:

• привлекательность ресурса как показатель при рассмотрении угрозы от умышленного воздействия со стороны человека;

• возможность использования ресурса для получения дохода как показатель при рассмотрении угрозы от умышленного воздействия со стороны человека;

• технические возможности угрозы, используемые при умышленном воздействии со стороны человека;

• вероятность того, что угроза реализуется;

• степень легкости, с которой уязвимость может быть использована.

Вопрос о том, как провести границу между допустимыми и недопустимыми рисками, решается пользователем. Очевидно, что разработка политики безопасности требует учета специфики конкретных организаций.

На основании политики безопасности строится программа безопасности, которая реализуется на процедурном и программно техническом уровнях уровне.

К **процедурному уровню** относятся меры безопасности, реализуемые людьми.

Можно выделить следующие группы процедурных мер:

• управление персоналом;

• физическая защита;

• поддержание работоспособности;

• реагирование на нарушения режима безопасности;

• планирование восстановительных работ.

Управление персоналом заключается в выполнении следующих условий. Во-первых, для каждой должности существовать квалификационные требования по ИБ. Во-вторых, в должностные инструкции должны входить разделы, касающиеся информационной безопасности. В-третьих, каждого работника нужно научить мерам безопасности теоретически и на практике.

Меры физической защиты включают в себя защиту от утечки информации по техническим каналам, инженерные способы защиты и т.д.

Планирование восстановительных работ предполагает:

• слаженность действий персонала во время и после аварии;

• наличие заранее подготовленных резервных производственных площадок;

• официально утвержденную схему переноса на резервные площадки основных информационных ресурсов;

• -схему возвращения к нормальному режиму работы.

Поддержание работоспособности включает в себя создание инфраструктуры, включающий в себя как технические, так и процедурные регуляторы и способной обеспечить любой наперед заданный уровень работоспособности на всем протяжении жизненного цикла информационной системы.

Реагирование на нарушение режима безопасности может быть регламентировано в рамках отдельно взятой организации. В настоящее время, осуществляется только мониторинг компьютерных преступлений в национальном масштабе и на мировом уровне.

Основой **программно-технического уровня** являются следующие механизмы безопасности:

• идентификация и аутентификация пользователей;

• управление доступом;

• протоколирование и аудит;

• криптография;

• экранирование;

• обеспечение высокой доступности и т.д.

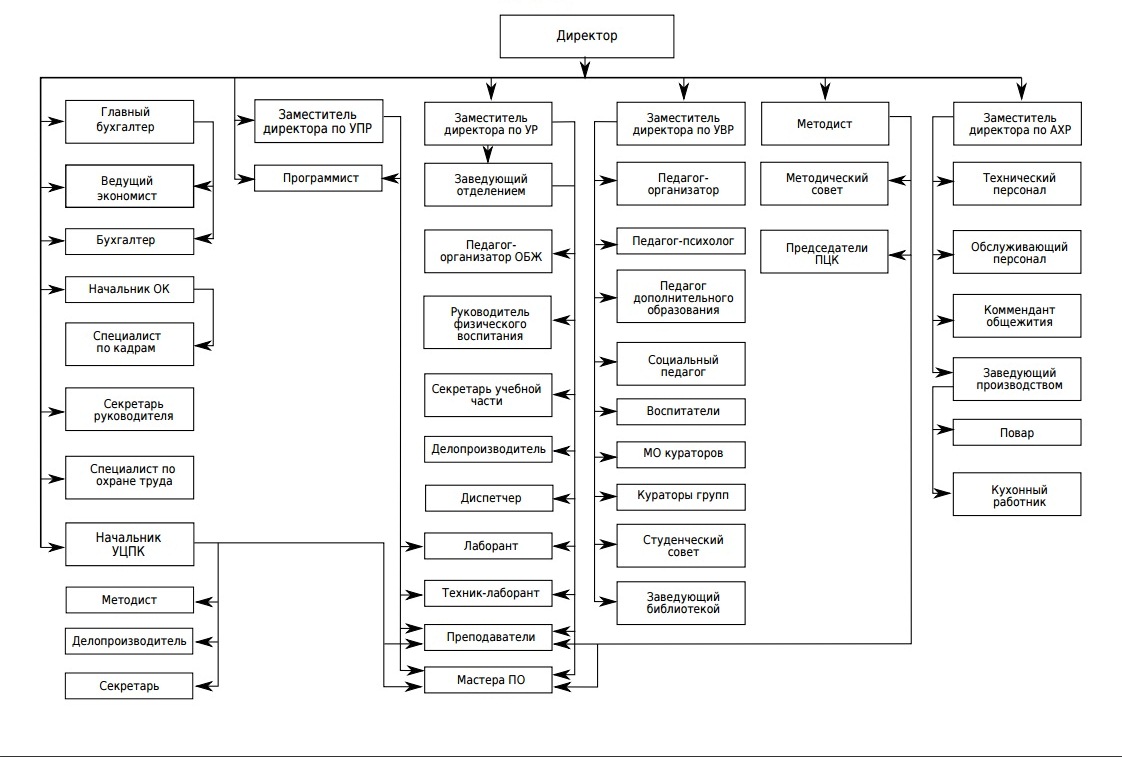
Важно управлять информационной системой в целом и механизмами безопасности в особенности. Упомянутые меры безопасности должны опираться на общепринятые стандарты, быть устойчивым к сетевым угрозам, учитывать специфику отдельных сервисов.

**Условие задания**

|  |  |
| --- | --- |
| 2. | Техникум |

**Структура техникума:**

1. Административное руководство: Включает директора или ректора техникума, который отвечает за общее управление учебным заведением. Рядом с ним могут быть заместители директора, административный персонал и другие административные должности, отвечающие за различные аспекты работы техникума.
2. Учебный совет: Это орган, состоящий из представителей администрации, преподавателей и студентов, который принимает решения по вопросам образования, учебного плана, кадрового обеспечения и другим важным аспектам обучения.
3. Кафедры и преподаватели: Техникум включает различные кафедры или отделения, каждое из которых специализируется в определенной области знаний или профессиональной области. Каждая кафедра имеет своего руководителя и преподавателей, которые осуществляют обучение студентов в соответствующих дисциплинах.
4. Учебные группы и студенты: Студенты техникума организуются в учебные группы, которые состоят из студентов, обучающихся вместе по определенной специальности или курсу. Каждая группа может иметь своего классного руководителя или наставника, который следит за успеваемостью и общим благополучием студентов.
5. Учебные и лабораторные помещения: Техникум обычно обладает различными учебными и лабораторными помещениями, где студенты получают практические навыки и проводят лабораторные работы в соответствующих областях знаний. Это могут быть аудитории, компьютерные классы, специализированные лаборатории и т. д.
6. Библиотека: Техникум обычно имеет библиотеку, где студенты и преподаватели могут получить доступ к необходимой учебной литературе, научным журналам и другим информационным ресурсам.
7. Административно-хозяйственный отдел: Включает различные службы, отвечающие за финансовое управление, кадровое делопроизводство, материально-техническое обеспечение, организацию питания студентов и другие аспекты, связанные с обеспечением нормального функционирования техникума.



**Оценка рисков для техникума может быть выполнена следующим образом:**

1. Идентификация ресурсов и оценка их количественных показателей:

* Системы вычислительной техники (количество компьютеров, серверов и других устройств).
* Данные (личная информация студентов, учетные записи, академические записи и т.д.).
* Программное обеспечение (системы управления учебным процессом, электронные ресурсы и т.д.).

1. Оценка угроз:

* Несанкционированный доступ к системам и данным (хакерские атаки, утечка паролей).
* Вирусы и вредоносное ПО (заражение компьютеров через вредоносные программы).
* Физические повреждения оборудования (пожар, наводнение, кража).

1. Оценка уязвимостей:

* Недостаточная защита сетевых соединений и устройств.
* Недостаточное обновление и поддержка программного обеспечения.
* Недостаточная обученность персонала по вопросам информационной безопасности.

1. Оценка существующих и предполагаемых средств обеспечения:

* Наличие антивирусной защиты и системы мониторинга сетевой активности.
* Регулярное резервное копирование данных.
* Политика управления доступом и парольная защита.

1. Оценка рисков:

* Вероятность нарушения информационной безопасности (высокая, средняя, низкая).
* Потенциальные последствия нарушений (утечка конфиденциальных данных, нарушение учебного процесса).

**На основе проведенной оценки рисков могут быть предприняты следующие меры для обеспечения информационной безопасности в техникуме**:

* Разработка и внедрение политики информационной безопасности, которая определит стратегию и меры защиты.
* Разработка и внедрение политики управления доступом, которая определит права доступа к информационным ресурсам в зависимости от должностных обязанностей и необходимости.
* Установка и обновление антивирусного программного обеспечения на всех компьютерах и серверах.
* Регулярное резервное копирование данных и их хранение на защищенных носителях.
* Обеспечение физической безопасности помещений, где находится вычислительная техника.
* Установка мощных паролей и регулярное их изменение.
* Мониторинг сетевой активности с целью выявления подозрительных действий или атак.
* Включать проверку сертификатов безопасности, заключение соответствующих соглашений о защите информации и мониторинг деятельности поставщиков и контрагентов.

**Также необходимо разработать административные меры защиты:**

* Разработка и внедрение политики информационной безопасности: Важно разработать документ, который определит стратегию и меры защиты информации в техникуме. Политика должна включать правила использования информационных систем, управление доступом, защиту конфиденциальных данных и другие важные аспекты безопасности.
* Установка ответственности: Необходимо определить ответственных лиц, которые будут отвечать за реализацию политики информационной безопасности и координацию мер по ее соблюдению. Это может быть комитет по информационной безопасности или назначенные сотрудники, которые будут отвечать за обеспечение безопасности информационных систем.
* Обучение и осведомленность: Регулярное обучение сотрудников, преподавателей и студентов основам информационной безопасности и правилам использования информационных систем является важным аспектом. Данные обучающие программы должны включать в себя обучение по защите паролей, распознаванию фишинговых атак, безопасному обращению с электронной почтой и другим аспектам безопасности.
* Управление доступом: Необходимо регулировать доступ к информационным системам и ресурсам на основе принципа "необходимости доступа". Это означает, что каждый пользователь должен иметь только необходимые права доступа для выполнения своих рабочих обязанностей. Управление доступом может осуществляться с помощью ролевой модели или других методов для определения прав доступа.

**Вывод:**

Политика безопасности является основой мер административного уровня, направленных на защиту информации и связанных с ней ресурсов в организации. Для эффективного построения системы защиты информации необходимо определить границы системы, ресурсы, подлежащие защите, и технологию обработки информации и решаемые задачи. Оценка рисков включает идентификацию угроз и уязвимостей, а также оценку существующих и предполагаемых средств обеспечения информационной безопасности. На основе оценки рисков могут быть предприняты соответствующие меры для обеспечения информационной безопасности, такие как разработка политики информационной безопасности, установка антивирусного программного обеспечения, регулярное резервное копирование данных, обучение персонала и мониторинг сетевой активности. Оценка рисков и принятие соответствующих мер по обеспечению информационной безопасности являются важными шагами для защиты конфиденциальной информации и предотвращения возможных инцидентов.

**Практическое занятие №3**

**Тема «Настройка Брандмауэра Windows»**

Цель: разобраться в настройках и использовании Брандмауэра.

**Теоретическое введение**

Брандмауэр (Межсетевой экран) - это аппаратный или программный комплекс, позволяющий проверять (фильтровать) входные и выходные потоки данных, проходящие через интернет или сеть. В случаи нарушения политики безопасности компьютера, брандмауэр блокирует эти данные.

Межсетевой экран является одним из основных компонентов защиты сетей. Наряду с Internet-протоколом межсетевого обмена (Internet Security Protocol — IPSec). Межсетевой экран является одним из важнейших средств защиты, осуществляя надежную аутентификацию пользователей и защиту от несанкционированного доступа. Отметим, что большая часть проблем с информационной безопасностью сетей связана с "прародительской" зависимостью коммуникационных решений от ОС UNIX – особенности открытой платформы и среды программирования UNIX сказались на реализации протоколов обмена данными и политики информационной безопасности. Вследствие этого ряд Internet-служб и совокупность сетевых протоколов (Transmission Control Protocol/Internet Protocol — TCP/IP) имеет "бреши" в защите.

К числу таких служб и протоколов относятся:

служба сетевых имен (Domain Name Server — DNS);

доступ к всемирной паутине WWW

программа электронной почты Send Mail

служба эмуляции удаленного терминала Telnet

простой протокол передачи электронной почты (Simple Mail Transfer Protocol — SMTP);

протокол передачи файлов (File Transfer Protocol);

графическая оконная система X Windows.

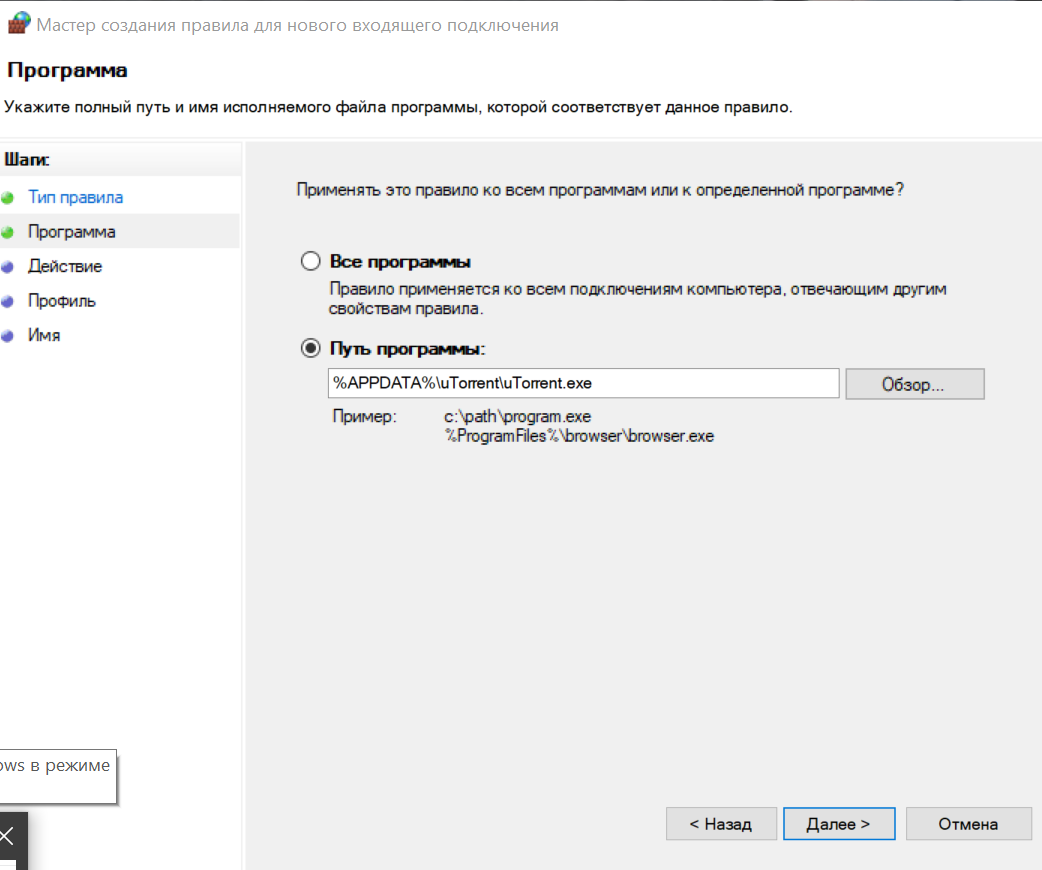
Настройки межсетевого экрана, т.е. решение пропускать или отсеивать пакеты информации, зависят от топологии распределенной сети и принятой политики информационной безопасности. В связи с этим политика реализации межсетевых экранов определяет правила доступа к ресурсам внутренней сети. Эти правила базируются на двух общих принципах – запрещать всё, что не разрешено в явной форме, и разрешать всё, что не запрещено в явной форме. Использование первого принципа дает меньше возможностей пользователям и охватывает жёстко очерченную область сетевого взаимодействия. Политика, основанная на втором принципе, является более мягкой, но во многих случаях она менее желательна, так как она предоставляет пользователям больше возможностей "обойти" межсетевой экран и использовать запрещенные сервисы через нестандартные порты (User Data Protocol – UDP), которые не запрещены политикой безопасности.

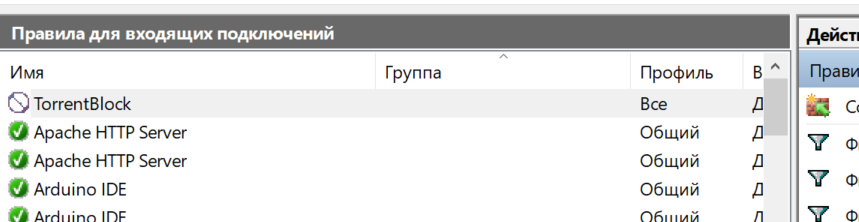
**Шпаргалка администратора ( Меню->Выполнить)**

Администрирование-control admintools  
Администратор источников данных-odbccp32.cpl  
Восстановление системных файлов-sfc /scannow  
Дефрагментация дисков-dfrg.msc  
Диспетчер проверки драйверов-verifier  
Дисптчер служебных программ-utilman  
Групповая политика-gpedit.msc  
Dr. Watson-drwtsn32  
Запросы операторов съемных ОЗУ-ntmsoprq.msc  
Защита БД учетных записей-syskey  
IExpress-iexpress  
Инфраструктура управления-wmimgmt.msc  
Проверка дисков-chkdsk  
Калькулятор-calc  
Командная строка-cmd  
Консоль управления-dcomcnfg  
Локальные параметры безопасности-secpol.msc  
Локальные пользователи и группы-lusrmgr.msc  
Мастер передачи файлов Bluetooth-fsquirt  
Настройка системы-msconfig  
Назначенные задания-control schedtasks  
Общие папки-fsmgmt.msc  
Общие ресурсы DDE-ddeshare  
Папка обмена-clipbrd  
Проверка подписи файла-sigverif  
Программа сетевого клиента SQL-cliconfg  
Производительность-perfmon.msc  
Просмотр событий-eventvwr.msc  
Подключение к рабочему столу-mstsc  
Результатирующая политика-rsop.msc  
Редактор системных файлов-sysedit  
Реестр-regedit  
Редактор личных символов-eudcedit  
Сертификаты-certmgr.msc  
Служба диагностики DirectX-dxdiag  
Службы-services.msc  
Службы компонентов-dcomcnfg  
Служба индексирования-ciadv.msc  
Съемные ЗУ-ntmsmgr.msc  
Телнет-telnet  
Управление дисками-diskmgmt.msc  
Управление рабочим столом-mstscoo  
Управление компьютером-compmgmt.msc  
Удаление вредоносных программ-mrt.exe

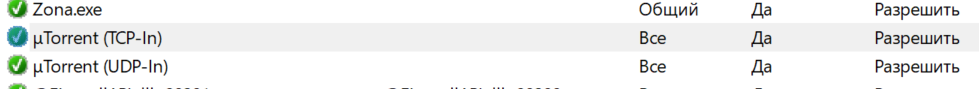
**Практическая часть**

1. Создать правила для входящих подключений (с помещением в электронный конспект копий экрана с пояснениями промежуточных действий):
   1. Для 1 программы (по выбору) на блокировку подключения;

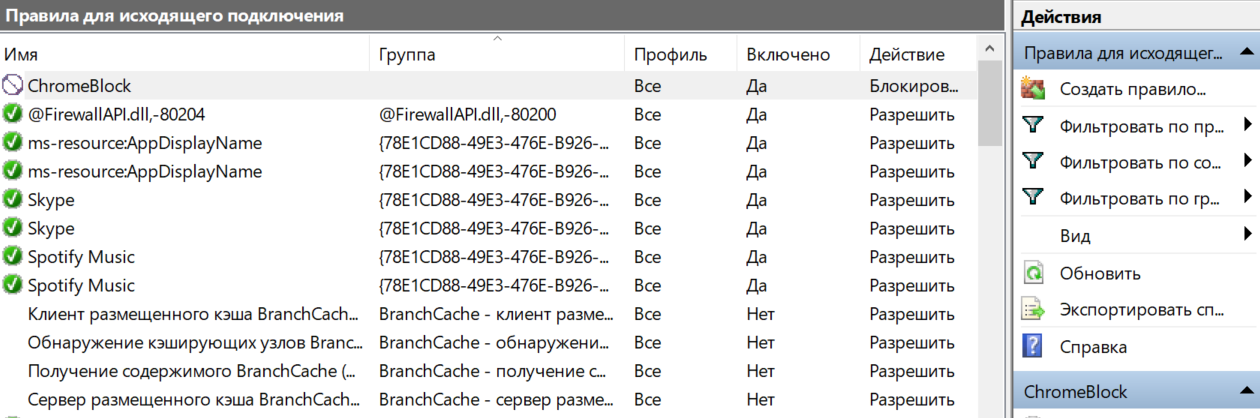


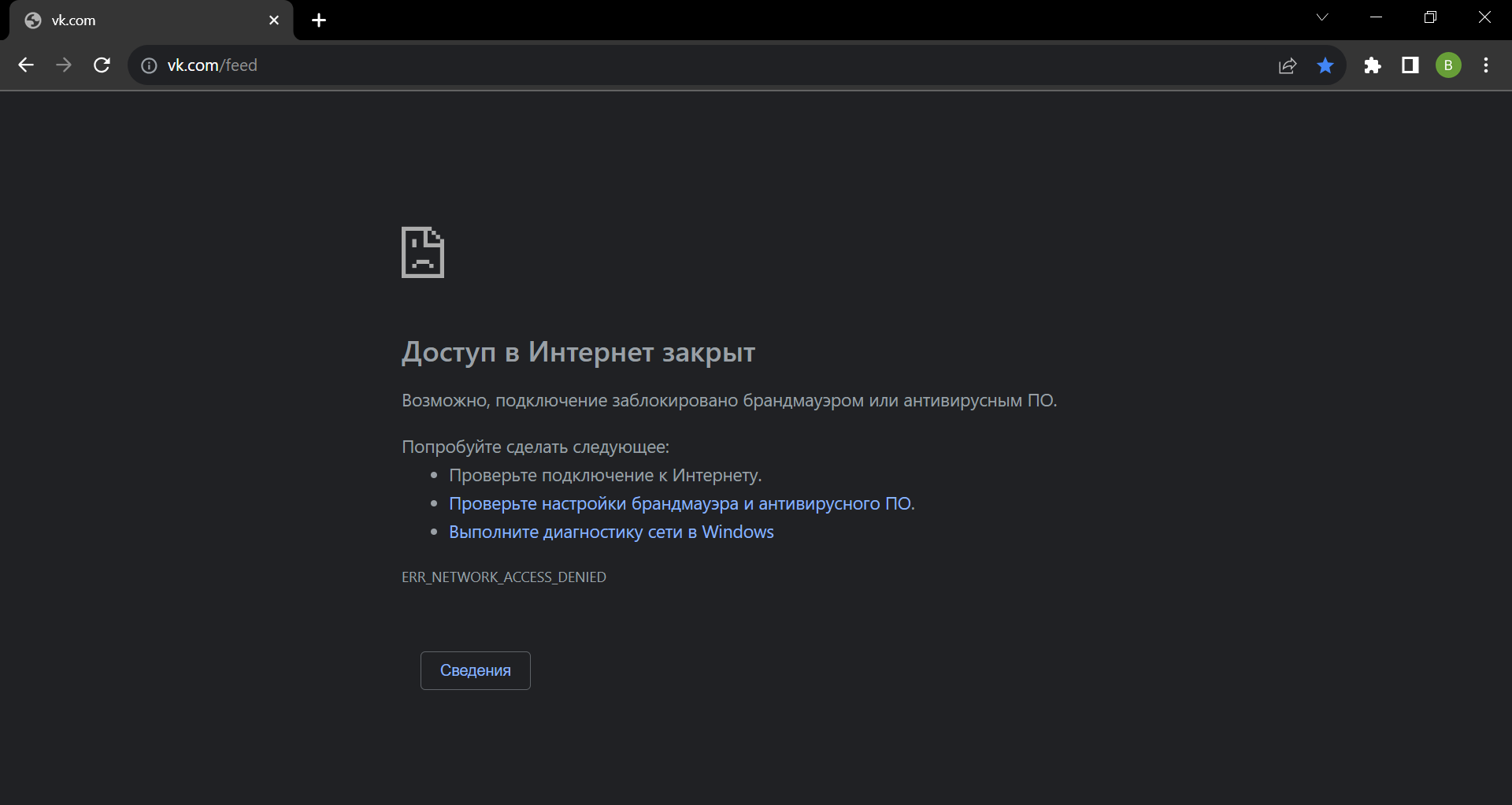


* 1. Для 1 программы (по выбору) на разрешение подключения;

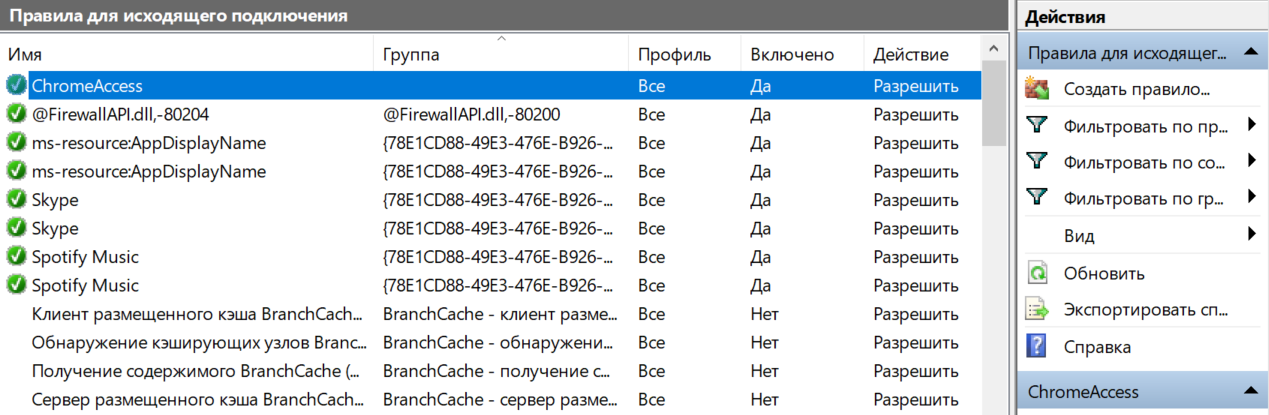


1. Создать правила для исходящих подключений (с помещением в электронный конспект копий экрана с пояснениями действий):
   1. Для 1 программы (по выбору) на блокировку подключения;

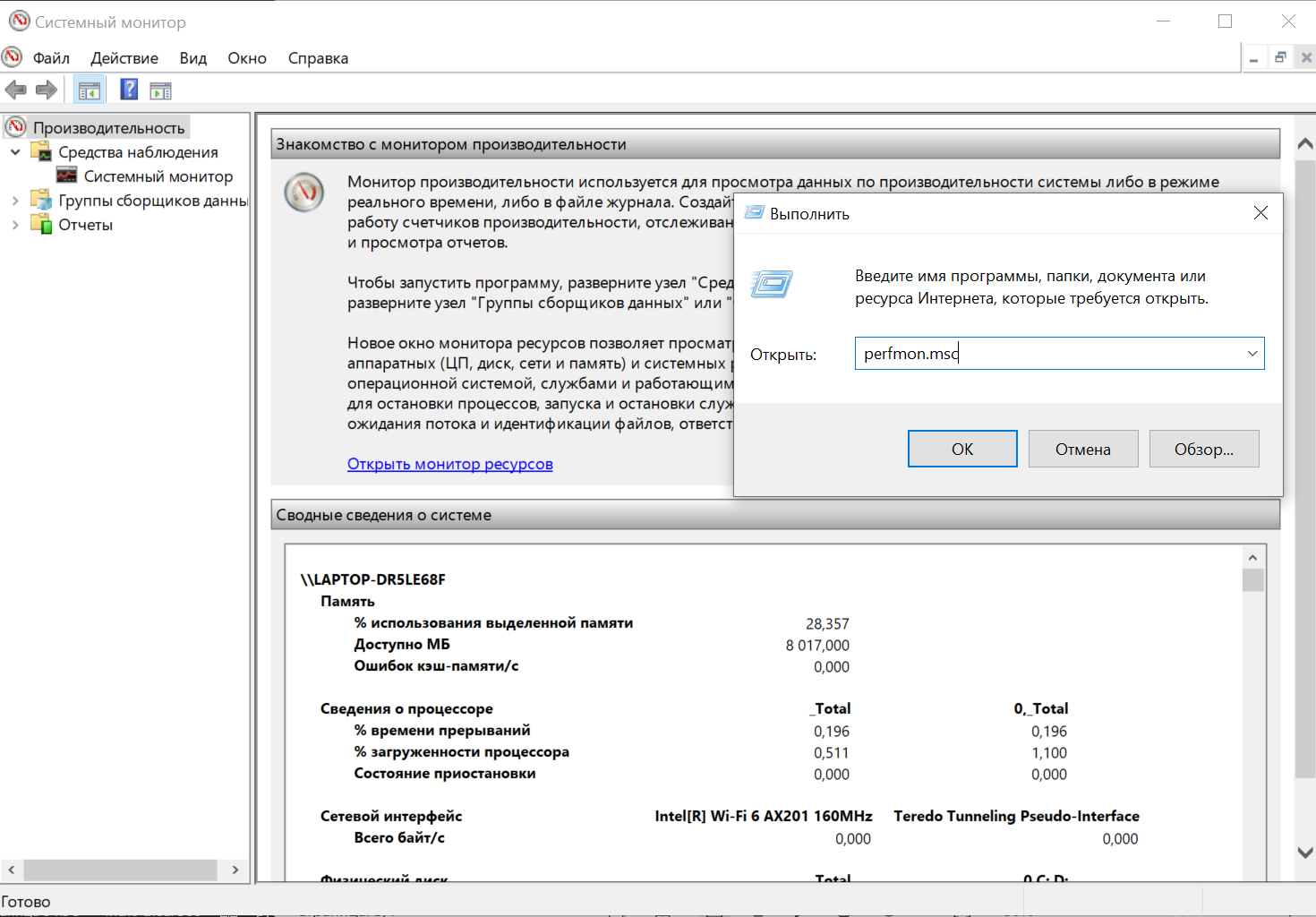




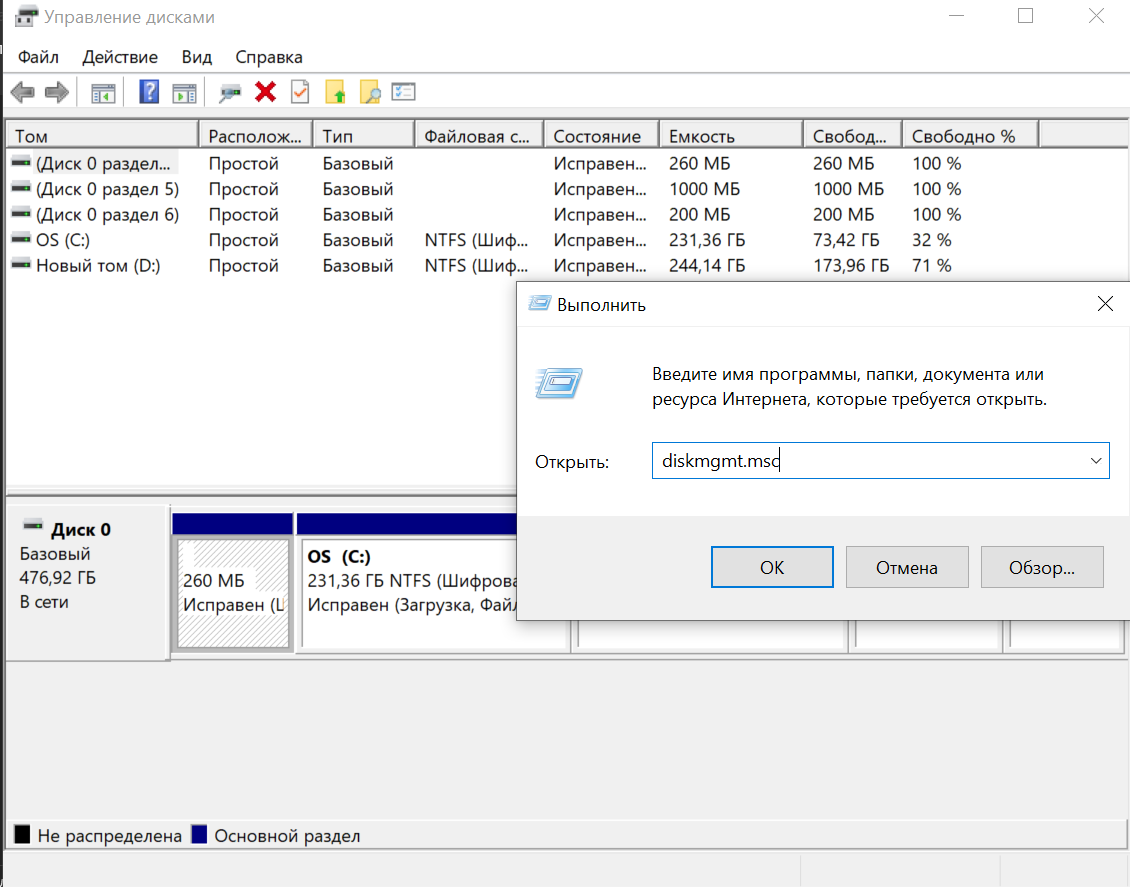
* 1. Для 1 программы (по выбору) на разрешение подключения;



1. Вернуть настройки Брандмауэра в исходное состояние до начала выполнения практического задания.
2. Опробовать действие нескольких команд (с помещением в электронный конспект копий экрана с пояснениями действий).
   1. Производительность



* 1. Управление дисками



**Практическое задание №4.1**

**Тема «Криптографическая защита информации»**

Цель: Овладение основными криптографическими алгоритмами симметричного шифрования .

**Теоретическое введение**

Криптография - наука о методах обеспечения конфиденциальности (невозможности прочтения информации посторонним) и аутентичности (целостности и подлинности авторства) информации.

Изначально криптография изучала методы шифрования информации – обратимого преобразования открытого (исходного) текста на основе секретного алгоритма и/или ключа в шифрованный текст (шифротекст). Традиционная криптография образует раздел симметричных криптосистем, в которых зашифрование и расшифрование проводится с использованием одного и того же секретного ключа.

Помимо этого современная криптография включает в себя асимметричные криптосистемы, системы электронной цифровой подписи, хеш-функции, управление ключами, получение скрытой информации, квантовую криптографию.

Шифрованием (encryption) называют процесс преобразования открытых данных (plaintext) в зашифрованные (шифртекст, ciphertext) или зашифрованных данных в открытые по определенным правилам с применением ключей.

В англоязычной литературе зашифрование / расшифрование – enciphering / deciphering.

Классификация алгоритмов шифрования

1. Симметричные (с секретным, единым ключом, одноключевые, single-key).

1.1. Потоковые:

· с одноразовым или бесконечным ключом (infinite-key cipher);

· с конечным ключом;

· на основе генератора псевдослучайных чисел.

1.2. Блочные:

1.2.1. Шифры перестановки (permutation, P-блоки);

1.2.2. Шифры замены (substitution, S-блоки):

· моноалфавитные;

· полиалфавитные;

2. Асимметричные (с открытым ключом, public-key):

· Диффи-Хеллман DH (Diffie, Hellman);

· Райвест-Шамир-Адлeман RSA (Rivest, Shamir, Adleman);

· Эль-Гамаль (ElGamal).

Симметричные алгоритмы шифрования (или криптография с секретными ключами) основаны на том, что отправитель и получатель информации используют один и тот же ключ. Этот ключ должен храниться в тайне и передаваться способом, исключающим его перехват.

Обмен информацией осуществляется в 3 этапа:

* отправитель передает получателю ключ (в случае сети с несколькими абонентами у каждой пары абонентов должен быть свой ключ, отличный от ключей других пар);
* отправитель, используя ключ, зашифровывает сообщение, которое пересылается получателю;
* получатель получает сообщение и расшифровывает его.

Если для каждого дня и для каждого сеанса связи будет использоваться уникальный ключ, это повысит защищенность системы.

При блочном шифровании информация разбивается на блоки фиксированной длины и шифруется поблочно. Блочные шифры бывают двух основных видов:

· шифры перестановки (transposition, permutation, P-блоки);

· шифры замены (подстановки, substitution, S-блоки).

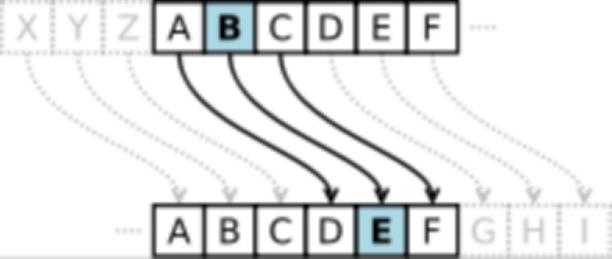
Шифры перестановок переставляют элементы открытых данных (биты, буквы, символы) в некотором новом порядке. Различают шифры горизонтальной, вертикальной, двойной перестановки, решетки, лабиринты, лозунговые и др.

Шифры замены заменяют элементы открытых данных на другие элементы по определенному правилу. Paзличают шифры простой, сложной, парной замены, буквенно-слоговое шифрование и шифры колонной замены. Шифры замены делятся на две группы:

· моноалфавитные (код Цезаря);

· полиалфавитные (шифр Видженера, цилиндр Джефферсона, диск Уэтстоуна, Enigma).

В моноалфавитных шифрах замены буква исходного текста заменяется на другую, заранее определенную букву. Например в коде Цезаря буква заменяется на букву, отстоящую от нее в латинском алфавите на некоторое число позиций.



Очевидно, что такой шифр взламывается совсем просто. Нужно подсчитать, как часто встречаются буквы в зашифрованном тексте, и сопоставить результат с известной для каждого языка частотой встречаемости букв.

В полиалфавитных подстановках для замены некоторого символа исходного сообщения в каждом случае его появления последовательно используются различные символы из некоторого набора. Понятно, что этот набор не бесконечен, через какое-то количество символов его нужно использовать снова. В этом слабость чисто полиалфавитных шифров.

В современных криптографических системах, как правило, используют оба способа шифрования (замены и перестановки). Такой шифратор называют составным (product cipher). Oн более стойкий, чем шифратор, использующий только замены или перестановки.

В асимметричных алгоритмах шифрования (или криптографии с открытым ключом) для зашифровывания информации используют один ключ (открытый), а для расшифровывания - другой (секретный). Эти ключи различны и не могут быть получены один из другого.

Схема обмена информацией такова:

· получатель вычисляет открытый и секретный ключи, секретный ключ хранит в тайне, открытый же делает доступным (сообщает отправителю, группе пользователей сети, публикует);

· отправитель, используя открытый ключ получателя, зашифровывает сообщение, которое пересылается получателю;

· получатель получает сообщение и расшифровывает его, используя свой секретный ключ.

**Задание к выполнению**

1. Изучить теоретические сведения по данной теме.
2. Зашифровать сообщение с использованием шифра Цезаря, Трисемуса, Плейфейра и Вижинера и полученного секретного ключа (по номеру варианта и ключевому слову «Защита»). В качестве сообщения использовать свою Фамилию Имя Отчество.

* Шифр Цезаря с ключом 5: Банкузов Михаил Олегович – Ёетпшмуж Снъенр Урйзужнь

public class CaesarCipher

{

const string alphabet = "АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

private string CodeEncode(string text, int k)

{

var fullAlphabet = alphabet + alphabet.ToLower();

var letterQty = fullAlphabet.Length;

var retVal = "";

for (int i = 0; i < text.Length; i++)

{

var c = text[i];

var index = fullAlphabet.IndexOf(c);

if (index < 0)

{

retVal += c.ToString();

}

else

{

var codeIndex = (letterQty + index + k) % letterQty;

retVal += fullAlphabet[codeIndex];

}

}

return retVal;

}

public string Encrypt(string plainMessage, int key)

=> CodeEncode(plainMessage, key);

public string Decrypt(string encryptedMessage, int key)

=> CodeEncode(encryptedMessage, -key);

}

* Шифр Трисемуса:

class TrisemusCipher

{

private char[,] trisemusSquare;

public TrisemusCipher(string key)

{

InitializeTrisemusSquare(key);

}

private void InitializeTrisemusSquare(string key)

{

string alphabet = "АБВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

trisemusSquare = new char[4, 8];

string uniqueKey = new string(key.ToUpper().Distinct().ToArray());

int index = 0;

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 8; j++)

{

if (index < uniqueKey.Length)

{

trisemusSquare[i, j] = uniqueKey[index++];

}

}

}

foreach (char ch in alphabet)

{

if (index < 32)

{

if (!uniqueKey.Contains(ch))

{

trisemusSquare[index / 8, index % 8] = ch;

index++;

}

}

else

{

break;

}

}

}

private Tuple<int, int> GetCharPosition(char ch)

{

for (int i = 0; i < 4; i++)

{

for (int j = 0; j < 8; j++)

{

if (trisemusSquare[i, j] == ch)

{

return Tuple.Create(i, j);

}

}

}

return null;

}

public string Encrypt(string plaintext)

{

plaintext = plaintext.ToUpper();

string ciphertext = "";

foreach (char ch in plaintext)

{

if (ch == ' ')

{

ciphertext += ' ';

continue;

}

var charPosition = GetCharPosition(ch);

if (charPosition != null)

{

int row = charPosition.Item1;

int col = charPosition.Item2;

ciphertext += trisemusSquare[(row + 1) % 4, col];

}

}

return ciphertext;

}

}

ЛЕЦУЬДЧМ ХЙЮЕЙФ ЧФПНЧМЙЗ

* Шифр Вижинера с ключом Защита:

public class VigenereCipher

{

const string defaultAlphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

readonly string letters;

public VigenereCipher(string alphabet = null)

{

letters = string.IsNullOrEmpty(alphabet) ? defaultAlphabet : alphabet;

}

private string GetRepeatKey(string s, int n)

{

var p = s;

while (p.Length < n)

{

p += p;

}

return p.Substring(0, n);

}

private string Vigenere(string text, string password, bool encrypting = true)

{

var gamma = GetRepeatKey(password, text.Length);

var retValue = "";

var q = letters.Length;

for (int i = 0; i < text.Length; i++)

{

var letterIndex = letters.IndexOf(text[i]);

var codeIndex = letters.IndexOf(gamma[i]);

if (letterIndex < 0)

{

retValue += text[i].ToString();

}

else

{

retValue += letters[(q + letterIndex + ((encrypting ? 1 : -1) \* codeIndex)) % q].ToString();

}

}

return retValue;

}

public string Encrypt(string plainMessage, string password)

=> Vigenere(plainMessage, password);

public string Decrypt(string encryptedMessage, string password)

=> Vigenere(encryptedMessage, password, false);

}

Банкузов Михаил Олегович – взлжотпб сдывщм рждбпбдю

* Шифр Плейфейра:

public abstract class PlayFairSettings

{

protected HashSet<char> HsAlphabet { get; private set; }

protected Dictionary<char, char> Replaces { get; private set; }

protected abstract char[] AlphabetChars { get; }

public abstract int Columns { get; }

public abstract int Rows { get; }

public abstract char Replacer { get; }

public IEnumerable<char> Alphabet

{

get

{

char[] chars = AlphabetChars;

foreach (var c in chars)

yield return c;

}

}

public PlayFairSettings()

{

HsAlphabet = new HashSet<char>();

Replaces = new Dictionary<char, char>();

foreach (var c in AlphabetChars)

HsAlphabet.Add(c);

}

public char GetChar(char Char)

{

Char = System.Char.ToLower(Char);

Char = Replaces.ContainsKey(Char) ? Replaces[Char] : Char;

return HsAlphabet.Contains(Char) ? Char : Replacer;

}

}

public class PlayFairRu56 : PlayFairSettings

{

public PlayFairRu56()

: base()

{

Replaces.Add('ё', 'е');

Replaces.Add('й', 'и');

Replaces.Add('ъ', 'ь');

}

protected override char[] AlphabetChars { get { return "абвгдежзиклмнопрстуфхцчшщьыэюя".ToCharArray(); } }

public override char Replacer { get { return 'х'; } }

public override int Columns { get { return 5; } }

public override int Rows { get { return 6; } }

}

public sealed class PlayFair

{

struct TablePosition

{

public int Row;

public int Column;

public TablePosition(int Row, int Column)

{

this.Row = Row;

this.Column = Column;

}

}

char[,] Matrix;

Dictionary<char, TablePosition> Positions;

public PlayFairSettings Settings { get; private set; }

public string Key { get; private set; }

public PlayFair(PlayFairSettings Settings, string Key)

{

this.Settings = Settings;

this.Key = Key;

Positions = new Dictionary<char, TablePosition>();

var items = MatrixItems().GetEnumerator();

Matrix = new char[Settings.Rows, Settings.Columns];

for (int r = 0; r < Settings.Rows; r++)

{

for (int c = 0; c < Settings.Columns; c++)

{

if (items.MoveNext())

{

Matrix[r, c] = items.Current;

Positions.Add(items.Current, new TablePosition(r, c));

}

else throw new ArgumentException("Алфавит слишком маленький");

}

}

}

IEnumerable<char> MatrixItems()

{

HashSet<char> used = new HashSet<char>();

foreach (char c in Key)

{

char rc = Settings.GetChar(c);

if (!used.Contains(rc))

{

used.Add(rc);

yield return rc;

}

}

foreach (char c in Settings.Alphabet)

{

if (!used.Contains(c))

{

used.Add(c);

yield return c;

}

}

}

public IEnumerable<char> Bigrams(string Text)

{

char prev = '\0';

bool even = false;

foreach (char c in Text)

{

char rc = Settings.GetChar(c);

if (!even)

{

prev = rc;

even = true;

}

else

{

if (prev == rc)

{

if (prev != Settings.Replacer)

{

yield return prev;

yield return Settings.Replacer;

prev = rc;

}

}

else

{

yield return prev;

yield return rc;

even = false;

}

}

}

if (even && prev != Settings.Replacer)

{

yield return prev;

yield return Settings.Replacer;

}

}

public string Crypt(string Text, bool ModeCrypt = true)

{

int shift = ModeCrypt ? 1 : -1;

StringBuilder sb = new StringBuilder();

var chars = Bigrams(Text).GetEnumerator();

while (chars.MoveNext())

{

var p1 = Positions[chars.Current];

chars.MoveNext();

var p2 = Positions[chars.Current];

int error = 0;

if (p1.Column == p2.Column)

{

p1.Row = Mod(p1.Row + shift, Settings.Rows);

p2.Row = Mod(p2.Row + shift, Settings.Rows);

error++;

}

else if (p1.Row == p2.Row)

{

p1.Column = Mod(p1.Column + shift, Settings.Columns);

p2.Column = Mod(p2.Column + shift, Settings.Columns);

error++;

}

if (error == 2)

throw new ArgumentException("Неверные биграммы");

sb.Append(Matrix[p1.Row, p2.Column]);

sb.Append(Matrix[p2.Row, p1.Column]);

}

return sb.ToString();

}

private int Mod(int x, int m)

{

return (x % m + m) % m;

}

public override string ToString()

{

StringBuilder sb = new StringBuilder();

sb.Append("Ключ: ");

sb.AppendLine(Key);

for (int r = 0; r < Settings.Rows; r++)

{

for (int c = 0; c < Settings.Columns; c++)

{

sb.Append(Matrix[r, c]);

sb.Append(' ');

}

sb.AppendLine();

}

return sb.ToString();

}

}

БА НК УЗ ОВ МИ ХА ИЛ ОЛ ЕГ ОВ ИЧ

ИА ЖУ ЁЗ ЦВ ХЫ ХЗ ИЕ БЛ МГ ЗК ЫЧ

1. Расшифровать следующие сообщения:

|  |
| --- |
|  |

|  |  |  |
| --- | --- | --- |
|  | **съчпщг окхчхиге ичлкпщг** | Расшифровать с помощью шифра Цезаря. Ключ 7 |

курить здоровью вредить

class Program

{

static void Main()

{

var cipher1 = new CaesarCipher();

Console.Write("Введите текст: ");

var message1 = Console.ReadLine();

Console.Write("Введите ключ: ");

var secretKey = Convert.ToInt32(Console.ReadLine());

var encryptedText = cipher1.Encrypt(message1, secretKey);

Console.WriteLine("Зашифрованное сообщение: {0}", encryptedText);

Console.WriteLine("Расшифрованное сообщение: {0}", cipher1.Decrypt(encryptedText, secretKey));

Console.WriteLine("-------------------------------------");

Console.WriteLine("Введите ключ: ");

string keyword = Console.ReadLine();

string additionalChars = "БВГДЕЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ";

string fullKeyword = keyword + additionalChars;

TrisemusCipher trisemus = new TrisemusCipher(fullKeyword);

Console.WriteLine("Введите текст для шифрования: ");

string plaintext = Console.ReadLine();

string ciphertext = trisemus.Encrypt(plaintext);

Console.WriteLine("Зашифрованный текст: " + ciphertext);

Console.WriteLine("-------------------------------------");

PlayFairSettings ps = new PlayFairRu56();

PlayFair pf = new PlayFair(ps, "Защита");

Console.WriteLine(pf);

string Text = "БанкузовМихаилОлегович";

Console.WriteLine("Исходный текст\r\n" + Text);

int i = 0;

foreach (char c in pf.Bigrams(Text))

{

Console.Write(c);

i++;

if (i % 2 == 0) Console.Write(' ');

if (i % 10 == 0) Console.WriteLine();

}

Text = pf.Crypt(Text, true);

Console.WriteLine("\r\nЗашифрованный текст\r\n" + Text);

Text = pf.Crypt(Text, false);

Console.WriteLine("\r\nРасшифрованный текст\r\n" + Text);

Console.WriteLine("-------------------------------------");

var cipher = new VigenereCipher("АБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯ");

Console.Write("Введите текст: ");

var inputText = Console.ReadLine().ToUpper();

Console.Write("Введите ключ: ");

var password = Console.ReadLine().ToUpper();

var encryptedText4 = cipher.Encrypt(inputText, password);

Console.WriteLine("Зашифрованное сообщение: {0}", encryptedText4);

Console.WriteLine("Расшифрованное сообщение: {0}", cipher.Decrypt(encryptedText4, password));

Console.WriteLine("-------------------------------------");

string encryptedMessage = "съчпщг окхчхиге ичлкпщг";

int key1 = 7;

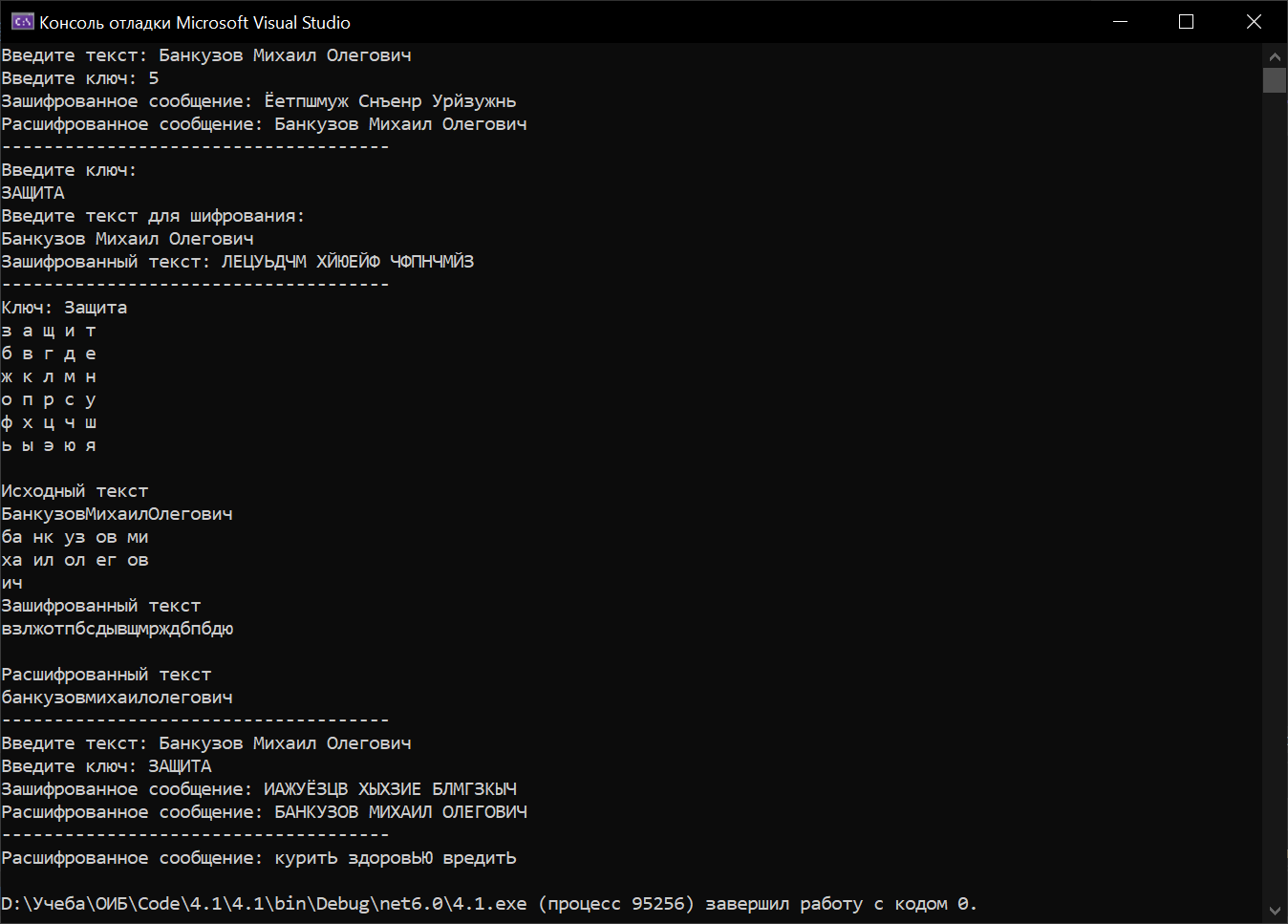
CaesarCipher cipher2 = new CaesarCipher();

string decryptedMessage = cipher2.Decrypt(encryptedMessage, key1);

Console.WriteLine("Расшифрованное сообщение: " + decryptedMessage);

}

}



**Практическое занятие №4.2**

**Тема « Криптографическая защита информации »**

Цель: Овладение основными криптографическими алгоритмами асимметричного шифрования.

**Теоретическое введение**

# Реализация элементов криптосистемы RSA

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Весь алгоритм расписан в таблице:

|  |  |  |
| --- | --- | --- |
| Этап | Описание операции | Результат операции |
| Генерация ключей | Выбрать два простых различных числа | p=3557,  q=2579 |
| Вычислить модуль (произведение) | n = p \cdot q = 3557 \cdot 2579 = 9173503 |
| Вычислить функцию Эйлера | \varphi(n) = (p-1) (q-1) = 9167368 |
| Выбрать открытую экспоненту | e = 3 |
| Вычислить секретную экспоненту | d = e^{-1} \mod \varphi(n)  d = 6111579 |
| Опубликовать открытый ключ | \{e, n\} = \{3,9173503 \} |
| Сохранить закрытый ключ | \{d, n\} = \{6111579, 9173503 \} |
| Шифрование | Выбрать текст для зашифровки | m = 111111 |
| Вычислить шифротекст | \begin{align} c &= E(m) \\  &= m^e \mod n \\  &= 111111^3   \mod 9173503 \\  &= 4051753 \end{align} |
| Расшифрование | Вычислить исходное сообщение | \begin{align} m &= D(c) = \\   &= c^d \mod n \\   &= 4051753^{6111579} \mod 9173503 \\   &= 111111 \end{align} |

# Реализация элементов схемы шифрования Эль-Гамаля

## **Генерация ключей**

1. Генерируется случайное простое число ~p длины ~n [битов](https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82).
2. Выбирается случайный примитивный элемент ~g.
3. Выбирается случайное целое число ~x такое, что ~1 < x < p-1.
4. Вычисляется ~y = g^x\,\bmod\,p.
5. Открытым ключом является тройка \left( p,g,y \right), закрытым ключом — число ~x.

## **Шифрование**

Сообщение ~M шифруется следующим образом:

1. Выбирается сессионный [ключ](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D1%8E%D1%87_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)) — случайное целое число ~k такое, что ~1 < k < p - 1
2. Вычисляются числа a = g^k\,\bmod\,p и b = y^k M\,\bmod\,p.
3. Пара чисел \left( a, b \right) является [шифротекстом](https://ru.wikipedia.org/wiki/%D0%A8%D0%B8%D1%84%D1%80%D0%BE%D1%82%D0%B5%D0%BA%D1%81%D1%82).

Нетрудно видеть, что длина шифротекста в схеме Эль-Гамаля длиннее исходного сообщения M вдвое.

## **Расшифрование**

Зная закрытый ключ ~x, исходное сообщение можно вычислить из шифротекста \left( a, b \right) по формуле:

M = b(a^x)^{-1}\,\bmod\,p.

При этом нетрудно проверить, что

~(a^x)^{-1}\equiv g^{-kx}\pmod{p}

и поэтому

~b(a^x)^{-1}\equiv (y^kM)g^{-xk}\equiv (g^{xk}M) g^{-xk}\equiv M \pmod{p}.

Для практических вычислений больше подходит следующая формула:

M = b(a^x)^{-1}\,\bmod\,p = b \cdot a^{(p-1-x)}\,\bmod\,p 

## **Пример**

**Шифрование**

Допустим, что нужно зашифровать сообщение ~M=5.

Произведем генерацию ключей :

пусть ~p=11, g=2. Выберем ~x=8 - случайное целое число ~x такое, что ~1 < x < p.

Вычислим ~y= g^x\bmod{p}=2^8\bmod{11}=3.

Итак, открытым является тройка ~(p,g,y)=(11,2,3), а закрытым ключом является число ~x=8.

Выбираем случайное целое число ~k такое, что 1 < k < (p − 1). Пусть ~k=9.

Вычисляем число ~a=g^k\bmod{p}=2^9 \bmod{11}=512 \bmod{11}=6.

Вычисляем число ~b=y^k M\bmod{p}=3^9 5 \bmod{11}=19683 \cdot 5 \bmod{11}=9.

Полученная пара ~(a,b)=(6,9) является шифротекстом.

**Расшифрование**

Необходимо получить сообщение ~M=5 по известному шифротексту ~(a,b)=(6,9) и закрытому ключу ~x=8.

Вычисляем M по формуле : ~M=b(a^x)^{-1}\bmod{p}=9(6^8)^{-1}\mod{11}=5

Получили исходное сообщение ~M=5.

# Реализация элементов схемы шифрования Дифи-Хеллмана

## **Генерация ключей**

В 1976 году после публичной критики алгоритма DES и указания на сложность обработки секретных ключей Уитфилд Диффи (Whitfield Diffie) и Мартин Хеллман (Martin Hellman) опубликовали свой алгоритм обмена ключами. Это была первая публикация на тему криптографии с открытым ключом и, возможно, самый большой шаг вперед в области криптографии, сделанный когда‑либо.

Из‑за невысокого быстродействия, свойственного асимметричным алгоритмам, алгоритм Диффи‑Хеллмана не предназначен для шифрования данных. Он был ориентирован на передачу секретных ключей DES, ARS или других подобных алгоритмов через небезопасную среду. В большинстве случаев алгоритм Диффи‑Хеллмана не используется для шифрования сообщений, потому что он, в зависимости от реализации, от 10 до 1000 раз медленнее алгоритма DES.

До алгоритма Диффи‑Хеллмана было сложно совместно использовать зашифрованные данные из‑за проблем хранения ключей и передачи информации. В большинстве случаев передача информации по каналам связи небезопасна, потому что сообщение может пройти десятки систем, прежде чем оно достигнет потенциального адресата, и нет никаких гарантий, что по пути никто не сможет взломать секретный ключ. Уитфилд Диффи и Мартин Хеллман предложили зашифровывать секретный ключ DES по алгоритму Диффи‑Хеллмана на передающей стороне и пересылать его вместе с сообщением, зашифрованным с использованием DES. Тогда на другом конце его сможет расшифровать только получатель сообщения.

На практике **обмен ключами** по алгоритму Диффи‑Хеллмана происходит по следующей схеме.

1. Два участника обмена договариваются о двух числах. Один выбирает большое простое число, а другой – целое число, меньшее числа первого участника. Переговоры они могут вести открыто, и это никак не отразится на безопасности.
2. Каждый из двух участников, независимо друг от друга, генерирует другое число, которое они будут хранить в тайне. Эти числа выполняют роль секретного ключа. Далее в вычислениях используются секретный ключ и два предыдущих целых числа. Результат вычислений посылается участнику обмена, и он играет роль открытого ключа.
3. Участники обмена обмениваются открытыми ключами. Далее они, используя собственный секретный ключ и открытый ключ партнера, конфиденциально вычисляют ключ сессии. Каждый партер вычисляет один и тот же ключ сессии.
4. Ключ сессии может использоваться как секретный ключ для другого алгоритма шифрования, например DES. Никакое третье лицо, контролирующее обмен, не сможет вычислить ключ сессии, не зная один из секретных ключей.

**Самое сложное в алгоритме** Диффи‑Хеллмана обмена ключами – это понять, что в нем фактически два различных независимых цикла шифрования. Алгоритм Диффи‑Хеллмана применяется для обработки небольших сообщений от отправителя получателю. Но в этом маленьком сообщении передается секретный ключ для расшифровки большого сообщения.

**Сильная сторона алгоритма** - никто не сможет скомпрометировать секретное сообщение, зная один или даже два открытых ключа получателя и отправителя. В качестве секретных и открытых ключей используются очень большие целые числа. Алгоритм Диффи‑Хеллмана основан на полезных для криптографии свойствах дискретных логарифмов.

## **Пример**

Ева — криптоаналитик. Она читает пересылку Боба и Алисы, но не изменяет содержимого их сообщений.

* s = секретный ключ. s = 2
* g = простое число меньшее p. g = 5
* p = открытое простое число. p = 23
* a = секретный ключ Алисы. a = 6
* A = открытый ключ Алисы. A = ga mod p = 8
* b = секретный ключ Боба. b = 15
* B = открытый ключ Боба. B = gb mod p = 19



З**адание №1**

Рассказать процесс работы алгоритма RSA.

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

using System;

using System.Security.Cryptography;

class Program

{

static void Main()

{

try

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

RSAParameters publicKey = rsa.ExportParameters(false);

RSAParameters privateKey = rsa.ExportParameters(true);

Console.Write("Введите текст для шифрования: ");

string message = Console.ReadLine();

byte[] encryptedData = Encrypt(message, publicKey);

string decryptedMessage = Decrypt(encryptedData, privateKey);

Console.WriteLine("\nОткрытый ключ:");

Console.WriteLine("Modulus (n): " + BitConverter.ToString(publicKey.Modulus));

Console.WriteLine("Exponent (e): " + BitConverter.ToString(publicKey.Exponent));

Console.WriteLine("\nЗакрытый ключ:");

Console.WriteLine("Modulus (n): " + BitConverter.ToString(privateKey.Modulus));

Console.WriteLine("Exponent (d): " + BitConverter.ToString(privateKey.D));

Console.WriteLine("\nЗашифрованное сообщение: " + BitConverter.ToString(encryptedData));

Console.WriteLine("Расшифрованное сообщение: " + decryptedMessage);

}

}

catch (CryptographicException e)

{

Console.WriteLine("Произошла ошибка при работе с RSA: " + e.Message);

}

}

static byte[] Encrypt(string message, RSAParameters publicKey)

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

rsa.ImportParameters(publicKey);

byte[] data = System.Text.Encoding.UTF8.GetBytes(message);

return rsa.Encrypt(data, false);

}

}

static string Decrypt(byte[] data, RSAParameters privateKey)

{

using (RSACryptoServiceProvider rsa = new RSACryptoServiceProvider())

{

rsa.ImportParameters(privateKey);

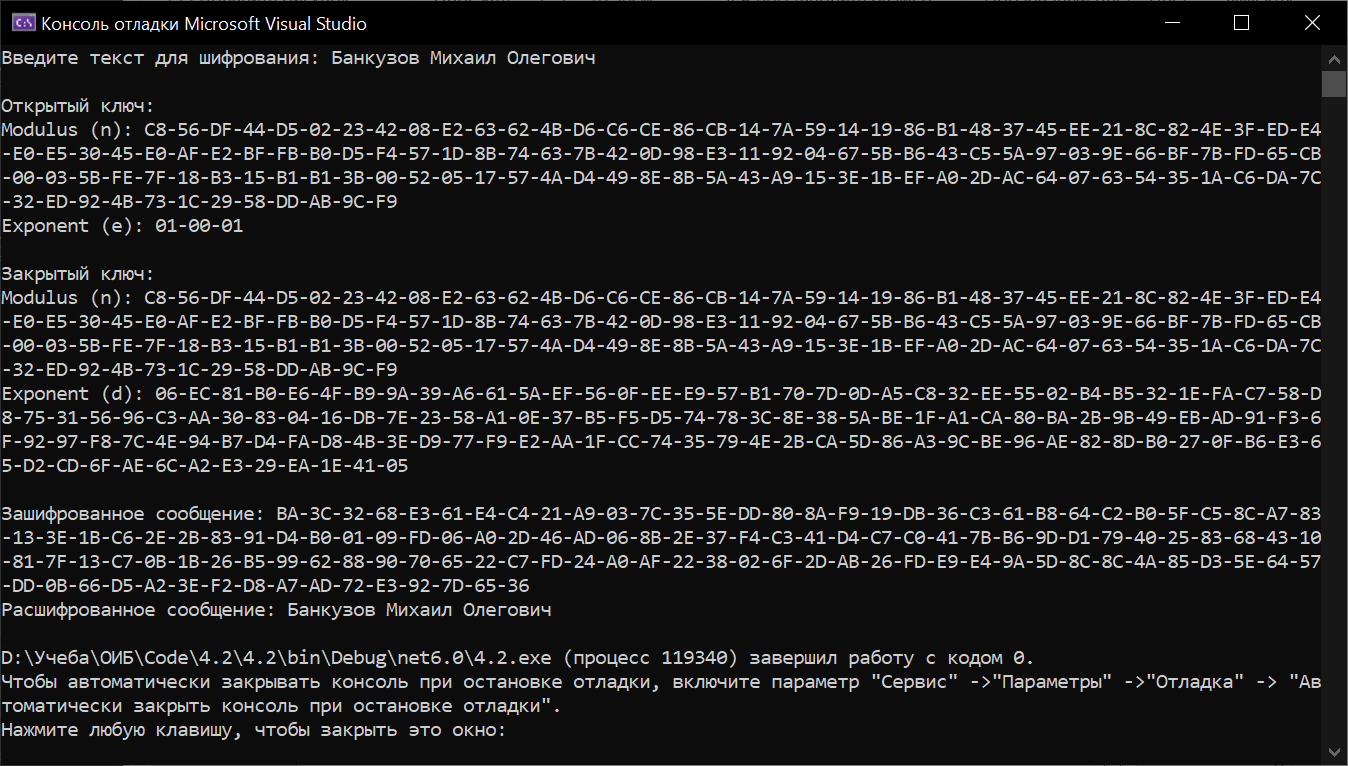
byte[] decryptedData = rsa.Decrypt(data, false);

return System.Text.Encoding.UTF8.GetString(decryptedData);

}

}

}

****

**Задание №2**

Рассказать процесс работы алгоритма Диффи-Хеллмана.

using System;

using System.Numerics;

public class DiffieHellman

{

private static Random random = new Random();

public static void RunDiffieHellman()

{

// Генерация простого числа p и первообразного корня g

int p = GeneratePrimeNumber();

int g = GeneratePrimitiveRoot(p);

// Алиса генерирует закрытый ключ a

int a = random.Next(2, p - 1);

// Боб генерирует закрытый ключ b

int b = random.Next(2, p - 1);

// Алиса и Боб вычисляют открытые ключи

BigInteger A = BigInteger.ModPow(g, a, p);

BigInteger B = BigInteger.ModPow(g, b, p);

// Алиса и Боб обмениваются открытыми ключами

BigInteger sharedKeyA = BigInteger.ModPow(B, a, p);

BigInteger sharedKeyB = BigInteger.ModPow(A, b, p);

// Общий ключ у Алисы и Боба теперь одинаков

Console.WriteLine($"p: {p}");

Console.WriteLine($"g: {g}");

Console.WriteLine($"Открытый ключ участника 1 (A): {A}");

Console.WriteLine($"Открытый ключ участника 2 (B): {B}");

Console.WriteLine($"Общий ключ у А: {sharedKeyA}");

Console.WriteLine($"Общий ключ у В: {sharedKeyB}");

}

// Генерация простого числа

private static int GeneratePrimeNumber()

{

int candidate = random.Next(100, 200); // Начнем с небольшого диапазона для упрощения примера

while (!IsPrime(candidate))

{

candidate++;

}

return candidate;

}

// Проверка на простоту

private static bool IsPrime(int number)

{

if (number < 2)

return false;

for (int i = 2; i <= Math.Sqrt(number); i++)

{

if (number % i == 0)

return false;

}

return true;

}

// Генерация первообразного корня

private static int GeneratePrimitiveRoot(int p)

{

for (int g = 2; g < p; g++)

{

bool isPrimitiveRoot = true;

for (int i = 1; i < p - 1; i++)

{

if (BigInteger.ModPow(g, i, p) == 1)

{

isPrimitiveRoot = false;

break;

}

}

if (isPrimitiveRoot)

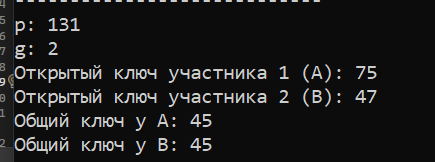
return g;

}

throw new Exception("Unable to find a primitive root.");

}

}



**Задание №3**

Рассказать процесс работы алгоритма Эль-Гамаля.

static int g\_main;

static BigInteger a;

public static bool Search\_g(int p, int g)

{

bool boolean = false;

List<BigInteger> array\_mod\_number = new List<BigInteger>();

BigInteger integer = ((BigInteger.Pow(g, 1)) % p);

array\_mod\_number.Add(integer);

for (int i = 2; i != p; i++)

{

integer = BigInteger.Pow(g, i) % p;

for (int j = 0; j != i - 1; j++)

{

if (array\_mod\_number[j] == integer)

{

g--;

array\_mod\_number.Clear();

i = 1;

integer = BigInteger.Pow(g, 1) % p;

array\_mod\_number.Add(integer);

break;

}

if ((j == i - 2) && (array\_mod\_number[j] != integer))

{

array\_mod\_number.Add(integer);

}

}

}

g\_main = g;

boolean = true;

return boolean;

}

public static int Search\_p()

{

Random random = new Random();

int p = 0;

Boolean boolean = false;

do

{

p = random.Next(2000, 2500);

for (int i = 2; i != p; i++)

{

if (i == p - 1)

{

boolean = Search\_g(p, p - 1);

break;

}

if (p % i == 0)

{

break;

}

}

}

while (boolean == false);

return p;

}

public static List<BigInteger> Cipher(string text, int p, BigInteger y)

{

List<BigInteger> array = new List<BigInteger>();

Random random = new Random();

int k = random.Next(1, p - 1);

for (int i = 0; i != text.Length; i++)

{

a = BigInteger.Pow(g\_main, k) % p;

array.Add((BigInteger.Pow(y, k) \* (int)text[i]) % p);

}

return array;

}

public static string Cipher\_RAZ(int length\_text, List<BigInteger> array\_number, int x, int p)

{

string save\_text = "";

BigInteger integer;

for (int i = 0; i != length\_text; i++)

{

integer = (array\_number[i] \* (BigInteger.Pow(a, p - 1 - x))) % p;

save\_text += (char)integer;

}

return save\_text;

}

int p = 0;

Random random = new Random();

p = Search\_p();

int x = random.Next(1, p - 1); //Генерирую закрытый ключ

BigInteger y = BigInteger.Pow(g\_main, x) % p; //Нахожу открытый ключ

Console.Write($"Числа [{p}, {g\_main}, {y}] ---> [p, g, y]\n");

Console.Write($"Открытый ключ - это <y>\n");

Console.Write($"Закрытый ключ: {x}\n");

Console.Write($"Введите текст: ");

string text = Console.ReadLine();

Console.WriteLine();

List<BigInteger> array\_cipher\_text = new List<BigInteger>();

array\_cipher\_text = Cipher(text, p, y);

Console.WriteLine("Зашифрованное сообщение: ");

for (int i = 0; i != text.Length; i++)

{

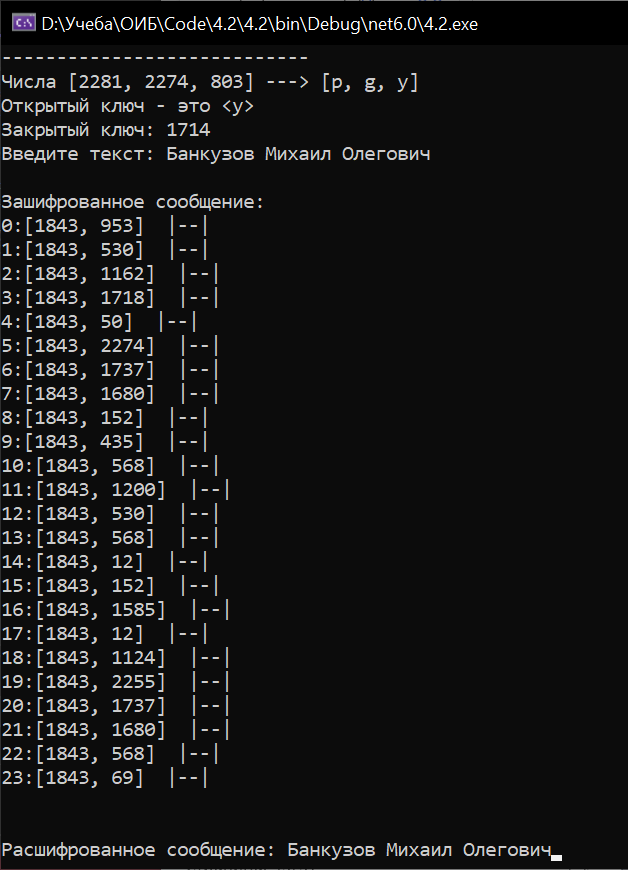
Console.WriteLine($"{i}:[{a}, {array\_cipher\_text[i]}] |--| ");

}

Console.WriteLine();

Console.Write($"\nРасшифрованное сообщение: {text = Cipher\_RAZ(text.Length, array\_cipher\_text, x, p)}");

Console.ReadKey();



**Практическое задание № 5**

**Тема «Криптографическая защита информации»**

**Цель:** изучить и закрепить умение реализации ЭЦП на примере RSA**.**

**Теоретические сведения**

# Реализация элементов ЭЦП RSA

Протоколы ЭЦП с одной стороны относят к протоколам аутентификации, т.к. гарантируют, что сообщение поступило от достоверного отправителя, а с другой стороны к протоколам контроля целостности, т.к. гарантируют, что сообщение пришло в неискаженном виде. Более того, получатель в дальнейшем может использовать ЭЦП как доказательство достоверности сообщения третьим лицам (арбитру) в том случае, если отправитель впоследствии попытается отказаться от него.

Говоря о схеме цифровой подписи, обычно имеют в виду следующую классическую ситуацию:

* отправитель знает содержание сообщения, которое он подписывает;
* получатель, зная открытый ключ проверки подписи, может проверить правильность подписи полученного сообщения в любое время без какого-либо разрешения и участия отправителя;
* безопасность схемы подписи гарантируется.

**Электронная цифровая подпись** – реквизит электронного документа, предназначенный для защиты данного документа от подделки, полученный в результате криптографического преобразования информации с использованием закрытого ключа ЭЦП и позволяющий идентифицировать владельца сертификата ключа подписи, а также установить отсутствие искажения информации в электронном документе (Федеральный закон "Об электронной цифровой подписи").

При создании цифровой подписи по классической схеме отправитель:

* применяет к исходному сообщению **T** хеш-функцию **h(T)** и получает хеш-образ r сообщения;
* вычисляет цифровую подпись **s по хеш-образу r с использованием своего закрытого ключа**;
* посылает сообщение **T** вместе с цифровой подписью s получателю.

Получатель, отделив цифровую подпись от сообщения, выполняет следующие действия:

* применяет к полученному сообщению **T** хеш-функцию **h(T)** и получает хеш-образ r сообщения;
* расшифровывает хеш-образ **r’** из цифровой подписи s с использованием открытого ключа отправителя;
* проверяет соответствие хеш-образов r и r’ и если они совпадают, то отправитель действительно является тем, за кого себя выдает, и сообщение при передаче не подверглось искажению.

Как видно из этой схемы, порядок использования ключей обратный тому, который используется при передаче секретных сообщений. Вначале отправитель использует свой закрытый ключ, а затем получатель применяет открытый ключ отправителя.

**Разновидности ЭЦП**

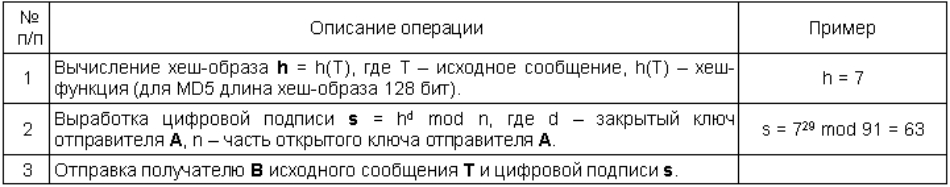
Кроме классической схемы ЭЦП различают еще несколько специальных:

* схема "конфиденциальной" (неотвергаемой) подписи – подпись не может быть проверена без участия сгенерировавшего ее лица;
* схема подписи "вслепую" ("затемненной" подписи) - отправитель не знает подписанного им сообщения;
* схема "мультиподписи" - вместо одного отправителя сообщение подписывает группа из нескольких участников;
* схема "групповой" подписи - получатель может проверить, что подписанное сообщение пришло от члена некоторой группы отправителей, но не знает, кем именно из членов группы оно подписано. В тоже время, в случае необходимости, отправитель может быть определен;
* и др.

**Этап 1.** Выработка ключей (выполняет отправитель **А**) - см. практическую работу 6 "Шифрование методом RSA".

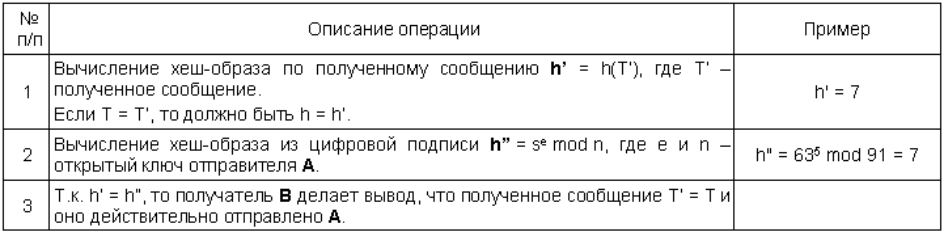
**Этап 2.** Отправка сообщения и электронной подписи (выполняет отправитель **А**).

Отправка сообщения и ЭЦП на базе алгоритма RSA



**Этап 3.** Получение сообщения и проверка электронной подписи (выполняет получатель **B**).

Получение сообщения и проверка ЭЦП на базе алгоритма RSA



1. Дайте определение понятию "электронная цифровая подпись".

**Электронная цифровая подпись** – реквизит электронного документа, предназначенный для защиты данного документа от подделки, полученный в результате криптографического преобразования информации с использованием закрытого ключа ЭЦП и позволяющий идентифицировать владельца сертификата ключа подписи, а также установить отсутствие искажения информации в электронном документе (Федеральный закон "Об электронной цифровой подписи").

1. Объясните какой порядок использования ключей (открытый; закрытый) при отправке и проверке ЭЦП.

Вначале отправитель использует свой закрытый ключ, а затем получатель применяет открытый ключ отправителя.

1. Перечислите специальные схемы ЭЦП. Перечислите недостатки алгоритма цифровой подписи RSA.

схема "конфиденциальной" (неотвергаемой) подписи – подпись не может быть проверена без участия сгенерировавшего ее лица;

схема подписи "вслепую" ("затемненной" подписи) - отправитель не знает подписанного им сообщения;

схема "мультиподписи" - вместо одного отправителя сообщение подписывает группа из нескольких участников;

схема "групповой" подписи - получатель может проверить, что подписанное сообщение пришло от члена некоторой группы отправителей, но не знает, кем именно из членов группы оно подписано. В тоже время, в случае необходимости, отправитель может быть определен;

Недостатки

1. При вычислении ключей для системы цифровой подписи RSA необходимо проверять ряд дополнительных условий. Невыполнение любого из этих условий делает возможным фальсификацию цифровой подписи со стороны того, кто обнаружит такое невыполнение.

2. Для обеспечения криптостойкости цифровой подписи RSA по отношению к попыткам фальсификации, например, на уровне алгоритма шифрования DES, необходимо использовать при вычислениях ключей очень большие целые числа, (около 10 в степени 154), что требует относительно больших вычислительных затрат, превышающих на 20-30% вычислительные затраты других алгоритмов цифровой подписи при сохранении того же уровня криптостойкости.

3. Цифровая подпись RSА уязвима к так называемой мультипликативной атаке. Иначе говоря, *алгоритм цифровой подписи RSA* позволяет злоумышленнику без знания секретного ключа сформировать подписи под теми документами, у которых результат хэширования можно вычислить как произведение результатов хэширования уже подписанных документов. Хотя следует заметить, что вероятность реализации такой атаки весьма незначительна.  
Для работы смарт-карт с цифровыми подписями RSA рекомендуется использование ключей с длиной модуля 1024 бит.

**Вывод:** Протоколы ЭЦП с одной стороны относят к протоколам аутентификации, т.к. гарантируют, что сообщение поступило от достоверного отправителя, а с другой стороны к протоколам контроля целостности, т.к. гарантируют, что сообщение пришло в неискаженном виде. Более того, получатель в дальнейшем может использовать ЭЦП как доказательство достоверности сообщения третьим лицам (арбитру) в том случае, если отправитель впоследствии попытается отказаться от него. Одним из самых популярных алгоритмов шифрования является RSA (от Rivest, Shamir, Adleman), однако и он имеет свои недостатки, которые могут по неосторожности или невыполнении требуемых условий выборки ключей привести к фальсификации или потере данных.

using System;

using System.Security.Cryptography;

namespace RSAExample

{

class Program

{

static void Main(string[] args)

{

using (RSA rsa = RSA.Create())

{

rsa.KeySize = 2048;

RSAParameters privateKeyParams = rsa.ExportParameters(true);

byte[] privateKeyBytes = ExportPrivateKey(privateKeyParams);

RSAParameters publicKeyParams = rsa.ExportParameters(false);

byte[] publicKeyBytes = ExportPublicKey(publicKeyParams);

byte[] message = System.Text.Encoding.UTF8.GetBytes("Банкузов Михаил Олегович");

Console.WriteLine("Исходное сообщение: " + System.Text.Encoding.UTF8.GetString(message));

Console.WriteLine();

byte[] signature = SignData(message, privateKeyParams);

bool isValid = VerifyData(message, signature, publicKeyParams);

if (isValid)

{

Console.WriteLine("Подпись действительна. Сообщение не было изменено.");

}

else

{

Console.WriteLine("Подпись недействительна. Сообщение было изменено или подпись неверна.");

}

}

}

static byte[] ExportPrivateKey(RSAParameters privateKeyParams)

{

using (RSACryptoServiceProvider rsaCsp = new RSACryptoServiceProvider())

{

rsaCsp.ImportParameters(privateKeyParams);

return rsaCsp.ExportCspBlob(true);

}

}

static byte[] ExportPublicKey(RSAParameters publicKeyParams)

{

using (RSACryptoServiceProvider rsaCsp = new RSACryptoServiceProvider())

{

rsaCsp.ImportParameters(publicKeyParams);

return rsaCsp.ExportCspBlob(false);

}

}

static byte[] SignData(byte[] data, RSAParameters privateKeyParams)

{

using (RSA rsa = RSA.Create())

{

rsa.ImportParameters(privateKeyParams);

byte[] signature = rsa.SignData(data, HashAlgorithmName.SHA256, RSASignaturePadding.Pss);

Console.WriteLine("Хеш исходного сообщения: " + BitConverter.ToString(data).Replace("-", ""));

Console.WriteLine();

Console.WriteLine("Подписанное сообщение (Signature): " + BitConverter.ToString(signature).Replace("-", ""));

Console.WriteLine();

return signature;

}

}

static bool VerifyData(byte[] data, byte[] signature, RSAParameters publicKeyParams)

{

using (RSA rsa = RSA.Create())

{

rsa.ImportParameters(publicKeyParams);

bool isValid = rsa.VerifyData(data, signature, HashAlgorithmName.SHA256, RSASignaturePadding.Pss);

Console.WriteLine("Верифицированное сообщение (подпись): " + BitConverter.ToString(data).Replace("-", ""));

Console.WriteLine();

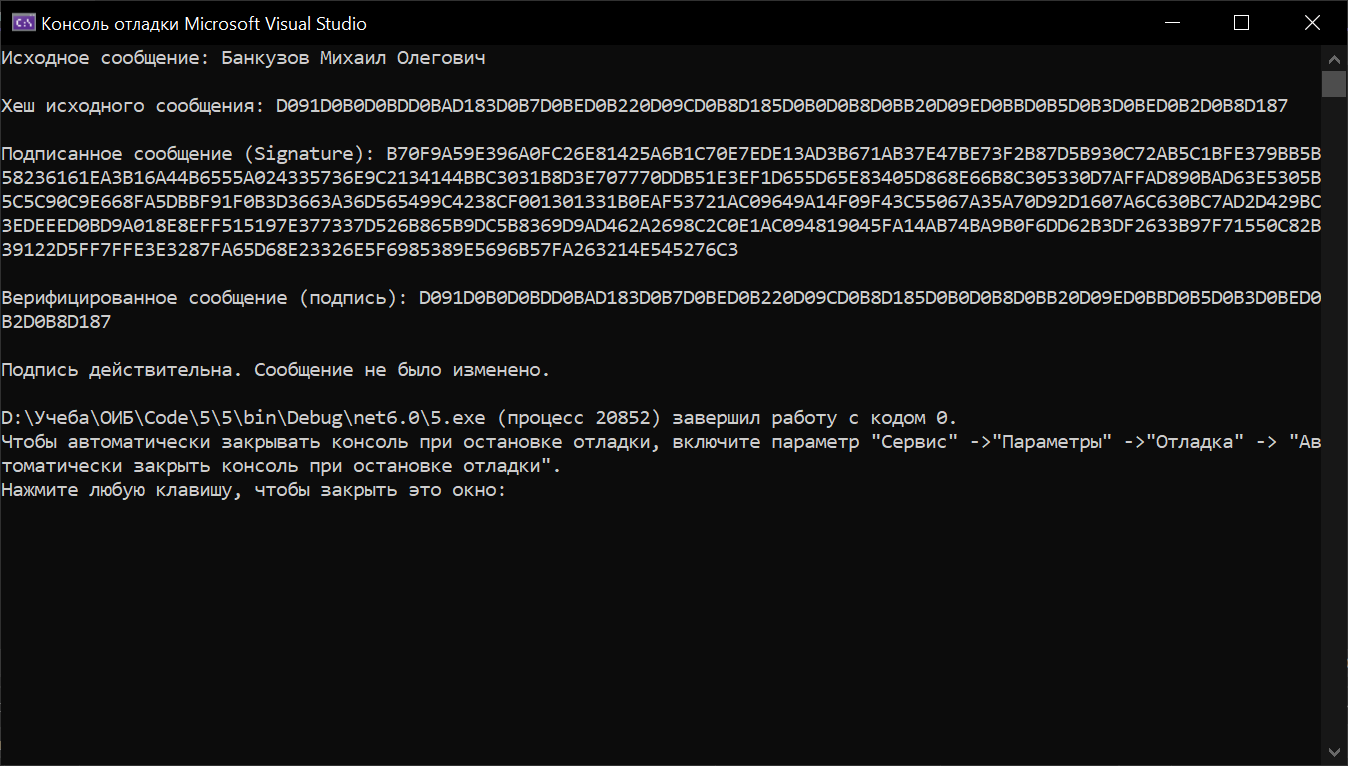
return isValid;

}

}

}

}



**Практическое занятие №6**

**Тема « Теория чисел»**

Цель**:**  получение основных сведений из курса теории чисел

**Вариант 2**

|  |  |
| --- | --- |
| 2. | 1-3. *а* = 5999801, *b* = 48685811  4. Найти остаток от деления C:\Users\mysha\AppData\Local\Temp\ksohtml58508\wps3.png на 17. |

using System;

using System.Collections.Generic;

namespace laboib6

{

public static class PrimeFactorization

{

public static void PrintPrimeFactorization(int num)

{

if (num <= 0)

{

Console.WriteLine("Число должно быть положительным.");

return;

}

Dictionary<int, int> primeFactors = new Dictionary<int, int>();

for (int factor = 2; factor <= num; factor++)

{

while (num % factor == 0)

{

if (primeFactors.ContainsKey(factor))

primeFactors[factor]++;

else

primeFactors[factor] = 1;

num /= factor;

}

}

foreach (var kvp in primeFactors)

{

Console.WriteLine($"{kvp.Key}^{kvp.Value}");

}

}

}

public class EvclidForNOD

{

public static int FindNOD(int a, int b)

{

while (b != 0)

{

int temp = b;

b = a % b;

a = temp;

}

return a;

}

}

public class PrimeFactorizationUtility

{

// Функция для разложения числа на простые множители и возврата словаря простых множителей и их степеней

public static Dictionary<int, int> PrimeFactorization(int num)

{

Dictionary<int, int> primeFactors = new Dictionary<int, int>();

for (int factor = 2; factor <= num; factor++)

{

while (num % factor == 0)

{

if (primeFactors.ContainsKey(factor))

primeFactors[factor]++;

else

primeFactors[factor] = 1;

num /= factor;

}

}

return primeFactors;

}

// Функция для нахождения НОД двух чисел на основе разложения на простые множители

public static int FindGCDUsingPrimeFactorization(int a, int b)

{

Dictionary<int, int> primeFactorsA = PrimeFactorization(a);

Dictionary<int, int> primeFactorsB = PrimeFactorization(b);

int gcd = 1;

foreach (var kvpA in primeFactorsA)

{

if (primeFactorsB.ContainsKey(kvpA.Key))

{

int commonPower = Math.Min(kvpA.Value, primeFactorsB[kvpA.Key]);

gcd \*= (int)Math.Pow(kvpA.Key, commonPower);

}

}

return gcd;

}

}

public class ExtendedEuclideanAlgorithm

{

public static void ExtendedGCD(int a, int b, out int nodplus, out int u, out int v)

{

if (b == 0)

{

nodplus = a;

u = 1;

v = 0;

}

else

{

ExtendedGCD(b, a % b, out nodplus, out u, out v);

int temp = u;

u = v;

v = temp - (a / b) \* v;

}

}

}

public class ModularPowerCalculator

{

public static long CalculateModularPower(long baseNumber, long exponent, long modulus)

{

if (modulus == 1)

{

return 0;

}

long result = 1;

baseNumber = baseNumber % modulus;

while (exponent > 0)

{

if (exponent % 2 == 1)

{

result = (result \* baseNumber) % modulus;

}

exponent = exponent >> 1;

baseNumber = (baseNumber \* baseNumber) % modulus;

}

return result;

}

}

class Program

{

static void Main(string[] args)

{

int a = 5999801;

int b = 48685811;

Console.WriteLine($"Число a: {a} ");

Console.WriteLine($"Число b: {b}");

Console.WriteLine("\nЗадание 1");

Console.WriteLine($"Каноническое разложение числа a ({a}):");

PrimeFactorization.PrintPrimeFactorization(a);

Console.WriteLine($"Каноническое разложение числа b ({b}):");

PrimeFactorization.PrintPrimeFactorization(b);

Console.WriteLine("\nЗадание 2 - а. Алгоритм Евклида");

int nod = EvclidForNOD.FindNOD(a, b);

Console.WriteLine($"НОД({a}, {b}) = {nod}");

Console.WriteLine("\nЗадание 2 - b. Разложение чисел на простые множители");

int gcd = PrimeFactorizationUtility.FindGCDUsingPrimeFactorization(a, b);

Console.WriteLine($"НОД({a}, {b}) = {gcd}");

Console.WriteLine("\nЗадание 3. С помощью расширенного алгоритма Евклида найти целые u, v, удовлетворяющие соотношению Безу");

int nodplus, u, v;

ExtendedEuclideanAlgorithm.ExtendedGCD(a, b, out nodplus, out u, out v);

Console.WriteLine($"НОД({a}, {b}) = {nodplus}");

Console.WriteLine($"u = {u}, v = {v}");

Console.WriteLine("\nЗадание 4. Найти остаток от деления 1995^2004 на 16");

long baseNumber = 2005;

long exponent = 2003;

long modulus = 17;

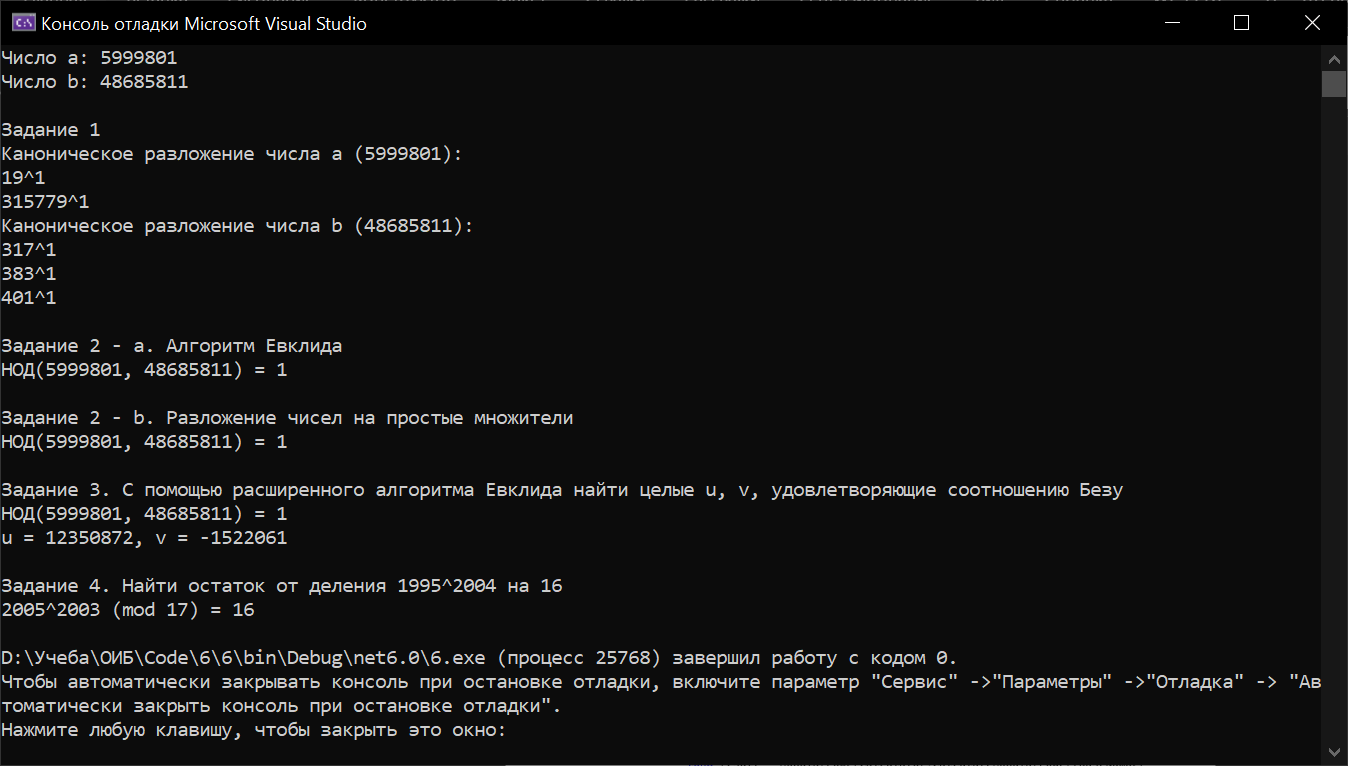
long result = ModularPowerCalculator.CalculateModularPower(baseNumber, exponent, modulus);

Console.WriteLine($"{baseNumber}^{exponent} (mod {modulus}) = {result}");

}

}

}



**Практическое задание № 7**

**Тема «**Настройка антивирусов**»**

Цель: Овладение навыками настройки и использования различных антивирусов.

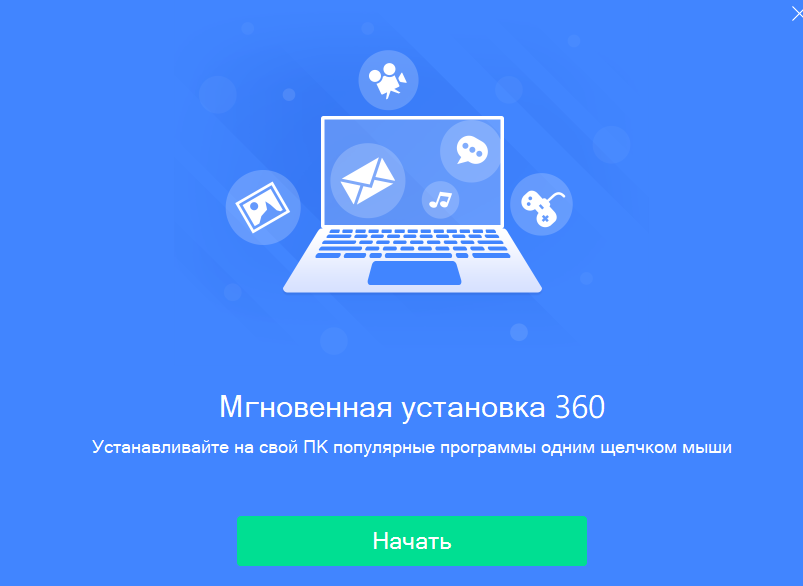
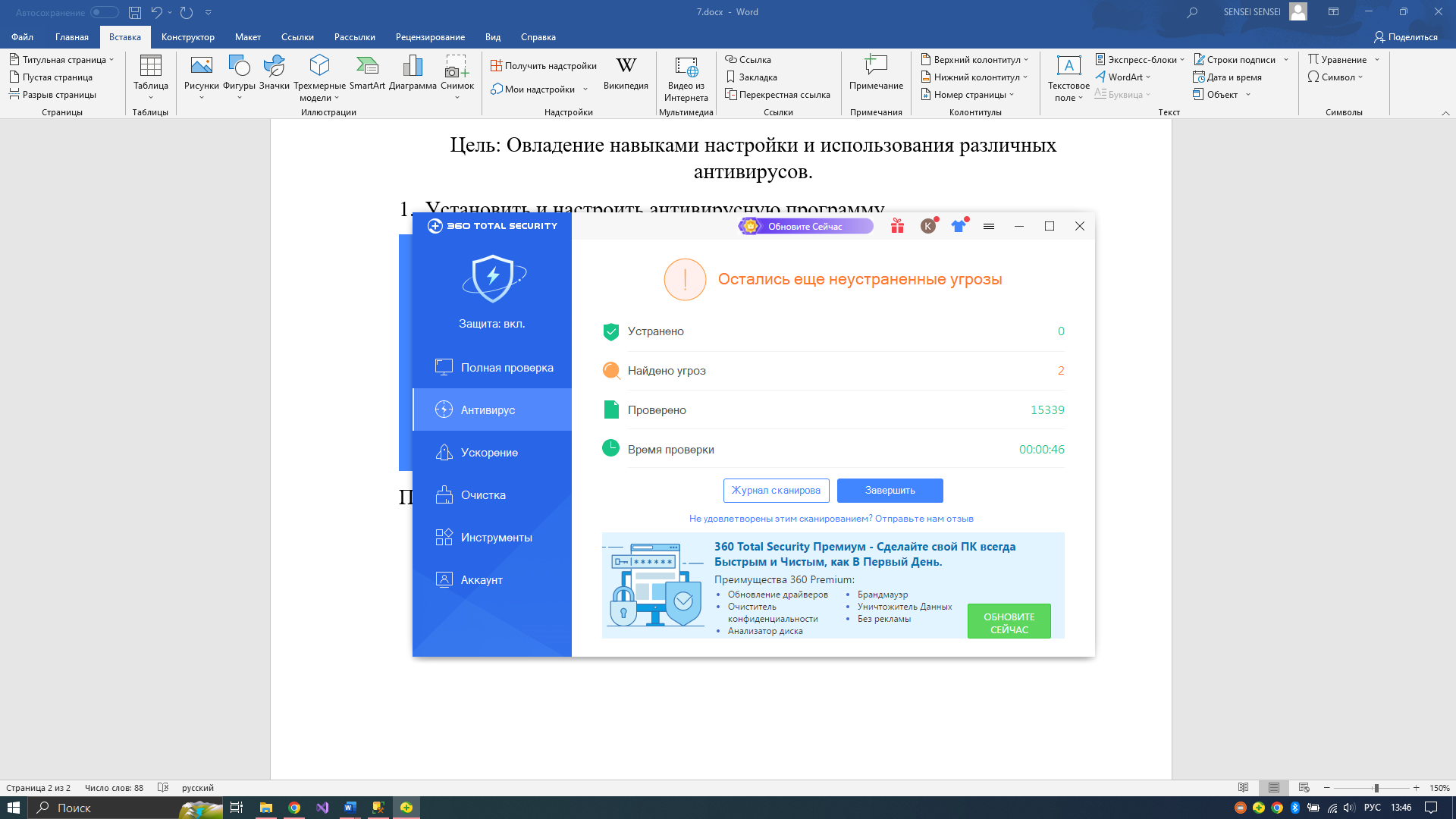
**Задание к выполнению**

1 Установить и настроить антивирусную программу по варианту.

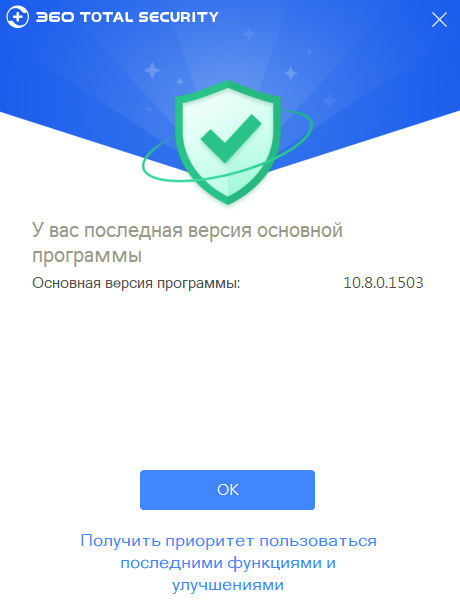
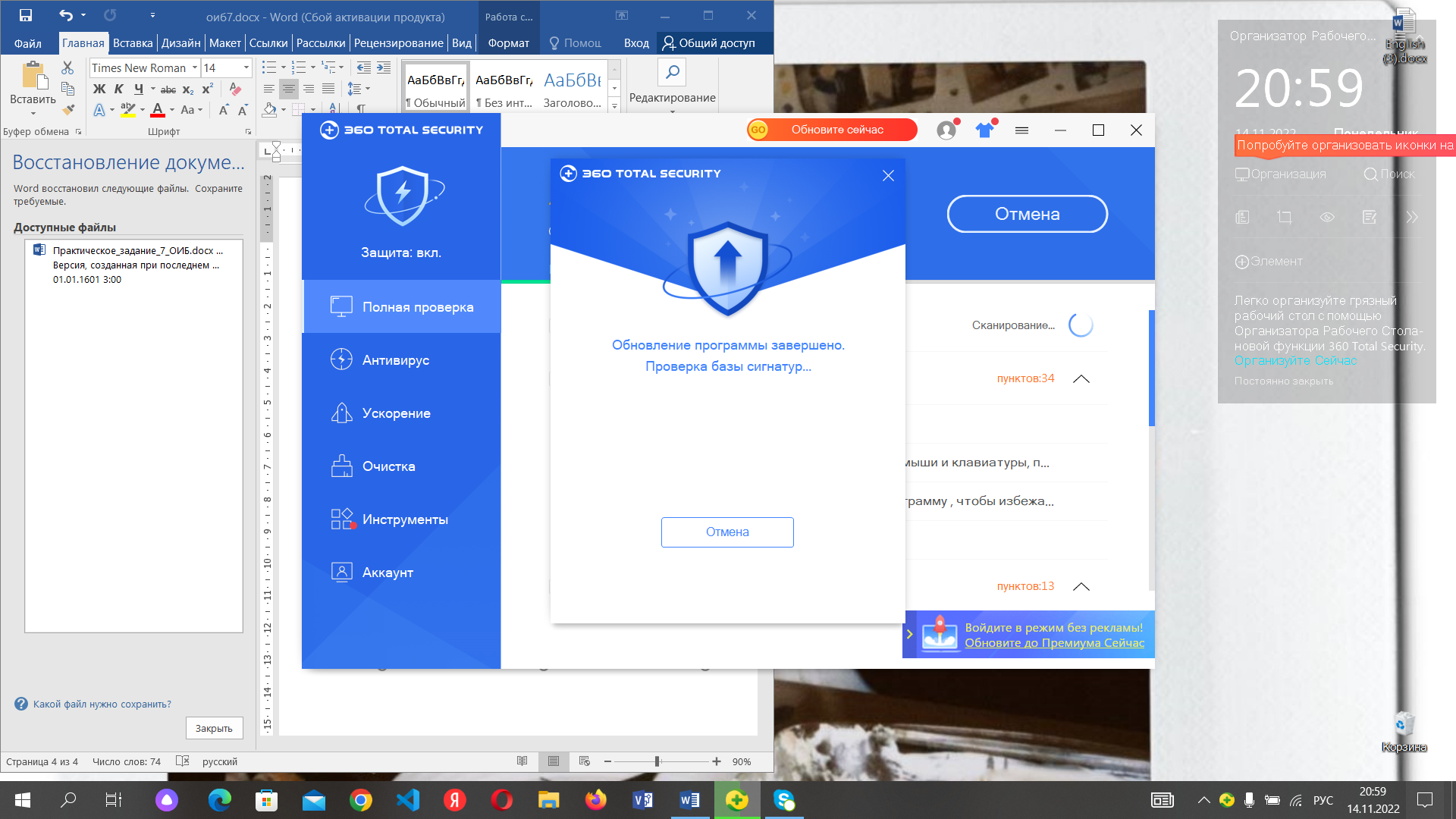
2 Обновить базу данных сигнатур вирусов.

3 Выполнить сканирование дисков.

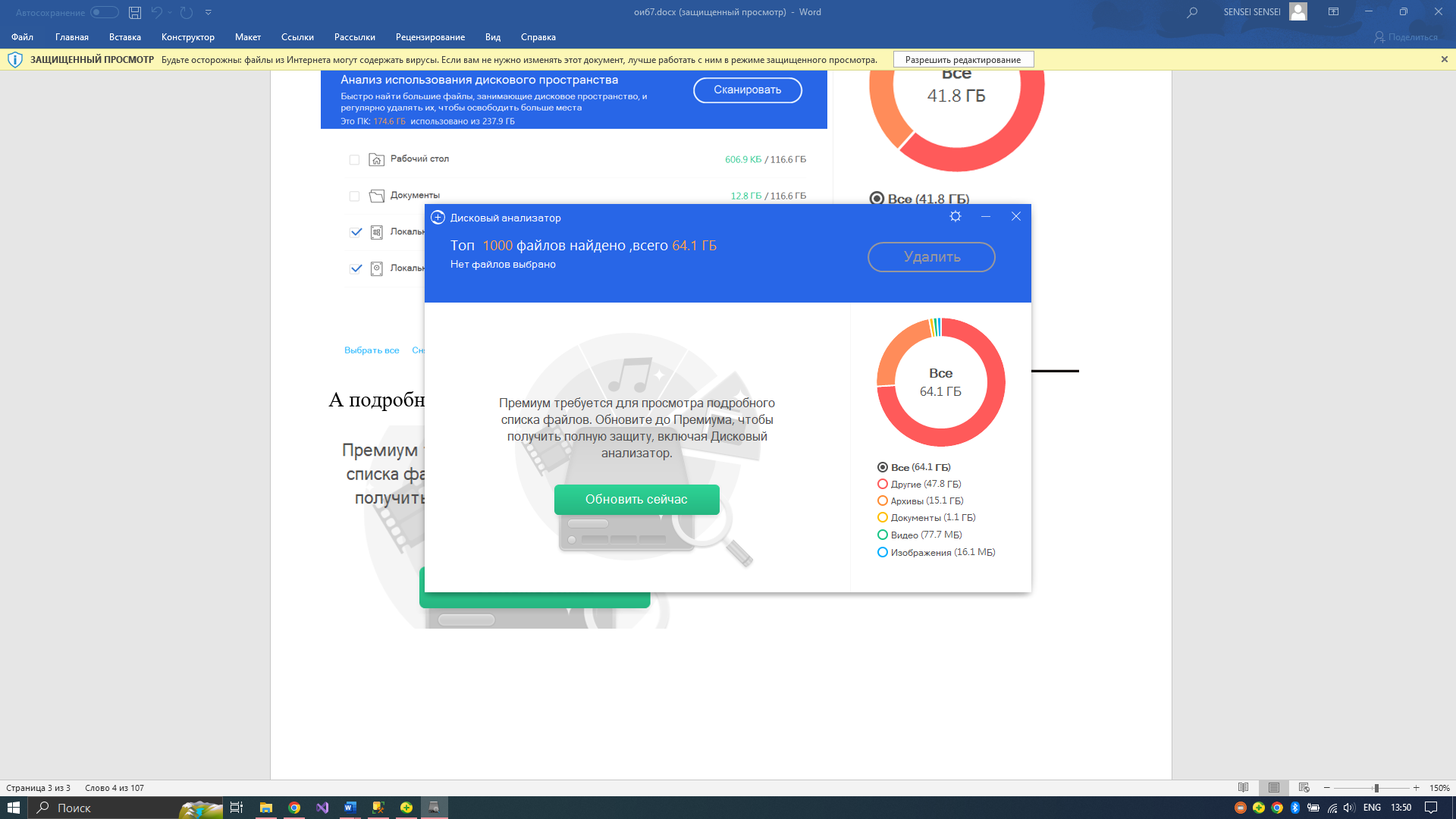
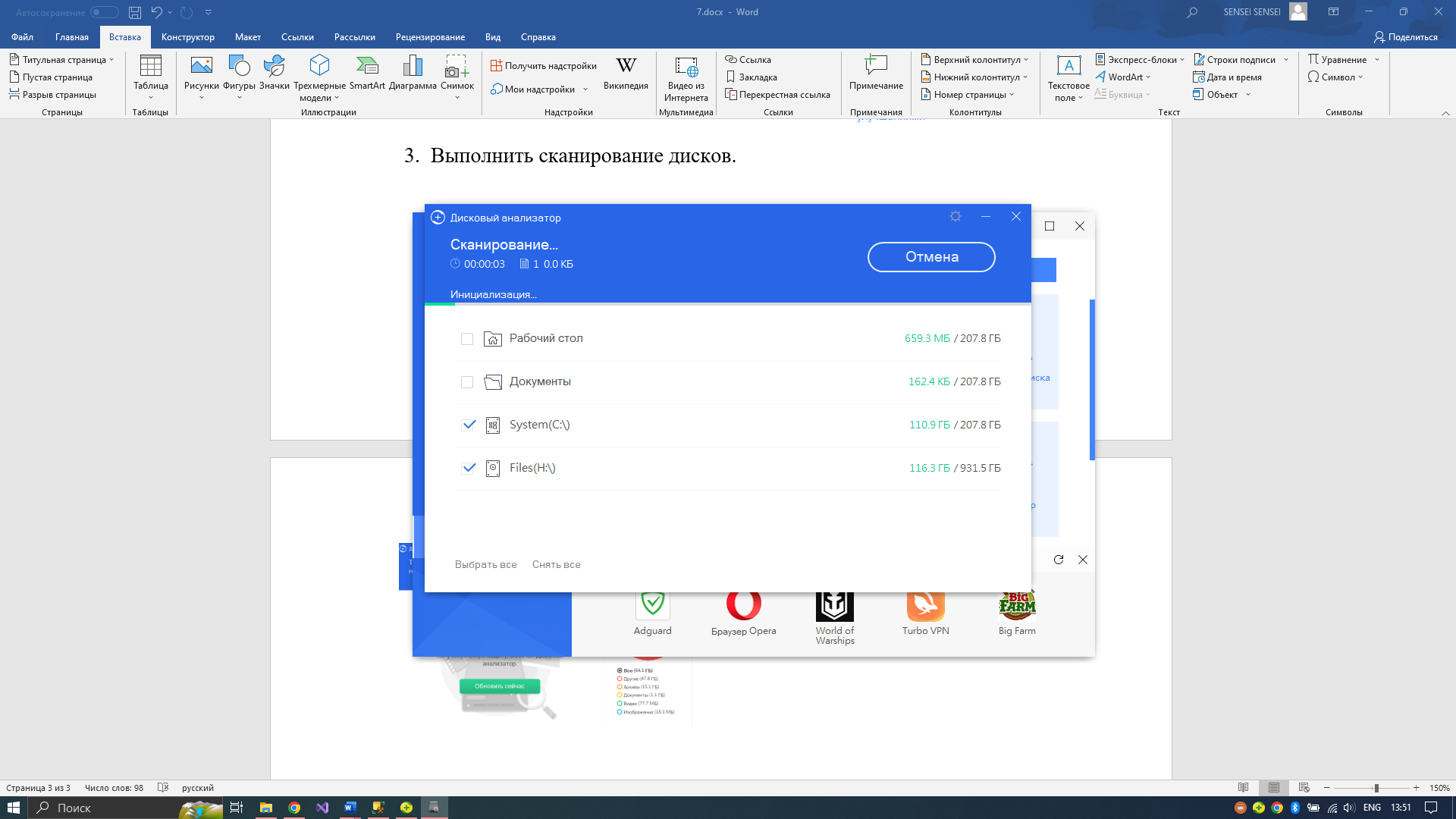
1. Установить и настроить антивирусную программу

2. Обновить базу данных сигнатур вирусов.



3. Выполнить сканирование дисков.

|  |
| --- |
| **Практическое занятие №8** |
| **Тема «Изучение стандартных средств для реализации приложений, использующих симметричное и ассиметричное шифрование с использованием библиотеки** [**System.Security.Cryptography**](https://docs.microsoft.com/ru-ru/dotnet/api/system.security.cryptography)**»** |
| Цель: Изучить модель криптографии .NET Framework, основные классы и структуры данных, разработать приложение для шифрования файлов использующих симметричные и ассиметричные алгоритмы шифрования |
| **Теоретические сведения** |

**Криптография в .Net**

В .Net Framework присутствует пространство имён для выполнения криптографических операций под названием System.Security.Cryptography. Данное пространство имён предоставляет криптографические службы, включающие безопасное кодирование и декодирование данных, а также другие операции, такие как хеширование сообщений, генерация случайных чисел и проверка подлинности сообщений. Данная библиотека предоставляет доступ для использования различных реализаций алгоритмов в основном это программные интерфейсы CryptoApi (CAPI) и Cryptography Next Generation API (CNG API) помимо этого для некоторых алгоритмов возможно использование реализаций на основе OpenSsl.

CryptoAPI — интерфейс программирования приложений, который обеспечивает разработчиков Windows-приложений стандартным набором функций для работы с криптопровайдером. Входит в состав операционных систем Microsoft. Большинство функций CryptoAPI поддерживается, начиная с Windows 2000.

Cryptography Next Generation стала долгосрочной заменой CAPI. Данный набор интерфейсов поддерживает все алгоритмы предлагаемые CAPI а также другие алгоритмы перечисленные в своде правил Suite B Агентства национальной безопасности США [1]. Данный интерфейс поддерживает следующие длины ключей или размерность хеша.

* RSA 512 бит to 16384 бит, с шагом 64 бит
* DH 512 бит to 16384 бит, с шагом 64 бит
* DSA 512 бит to 1024 бит, с шагом 64 бит
* ECDSA P-256, P-384, P-521 (NIST Curves)
* ECDH P-256, P-384, P-521 (NIST Curves)
* MD2 128 бит
* MD4 128 бит
* MD5 128 бит
* SHA-1 160 бит
* SHA-256 256 бит
* SHA-384 384 бит
* SHA-512 512 бит

Рассматривая структуру наследования для симметричных алгоритмов в .Net стоит упомянуть что SymmetricAlgorithm является абстрактным классом, от который наследуют абстрактные классы для реализаций каждого из алгоритмов. В свою очередь каждая из реализации алгоритма является производной от абстрактного класса алгоритма. Ниже представлена структура наследования.

SymmetricAlgorithm

Aes

AesCng

AesManeged

AesCryptoServiceProvider

Des

DesCng

DesManeged

DesCryptoServiceProvider

TripleDes

TripleDesCng

TripleDesManeged

TripleDesCryptoServiceProvider

Данная структура наследования повторяется для каждого из трех типов поддерживаемых криптографических операций: SymmetricAlgorithm, AsymmetricAlgorithm, HashAlgorithm.

Рассмотрим часть кода для шифрования сообщения используя алгоритм Aes. Как представлено ниже мы создаём объект шифратор на основе созданного экземпляра криптографического объекта aesAlg. Далее мы создаём потоки для шифрования. Стоит отметить CryptoStream который определяет поток, связывающий потоки данный с криптографическим преобразованием.

ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV);

// создание потоков используемого для шифрования

using (MemoryStream msEncrypt = new MemoryStream())

{

using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write))

//1-используемый поток, 2 - криптографические преобразование, 3 тип доступа

{

using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))

{

swEncrypt.Write(plainText);

}

encrypted = msEncrypt.ToArray();

}

}

}

Дешифрование выполняется аналогично за исключением изменения типа доступа CryptoStream и изменения потока с записи на чтение.

ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV);

using (MemoryStream msDecrypt = new MemoryStream(cipherText))

{

using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read))

{

using (StreamReader srDecrypt = new StreamReader(csDecrypt))

{

plaintext = srDecrypt.ReadToEnd();

}

}

}

Также стоит упомянуть что сложность большинства алгоритмов шифрование имеет сильную зависимость от длины ключа. Изменение длины ключа возможно в большинстве алгоритмов как симметричного, так и ассиметричного шифрования. Для того что бы изменить размер ключа после инициализации алгоритма необходимо изменить свойство KeySize, что представлено в листинге ниже.

using (TripleDES myDes = TripleDES.Create())

{

myDes.KeySize = 128;

EncryptStringToFile\_DES(route, "2 keys usedE.txt", myDes.Key, myDes.IV);

DecryptStringToFile\_DES("2 keys usedE.txt", "2 key deceypted.txt", myDes.Key, myDes.IV);

}

Что касается ассиметричных алгоритмов то они представлены в данной структуре наследования.

AsymmetricAlgorithm

Rsa

RsaCng

RsaOpenSsl

RsaCryptoServiceProvider

Dsa

DsaCng

DsaOpenSsl

DsaCryptoServiceProvider

ECDiffieHellman

ECDiffieHellmanCng

ECDiffieHellmanOpenSsl

Шифрование и дешифрование для асимметричных алгоритмов выполняется проще из за встроенных функций Encrypt и Decrypt. Так же стоит помнить, что Rsa является блочным алгоритмов и если длина данных не совпадает с длиной блока, то данные нужно дополнить до длины блока. Для этого в данном пространстве имён можно использовать разные режимы заполнения.

using (RSA myRsa = RSA.Create())

{

string publickey = myRsa.ToXmlString(false); //получим открытый ключ

string privatekey = myRsa.ToXmlString(true);//получим закрытый ключ

byte[] encrypted = myRsa.Encrypt(data, RSAEncryptionPadding.Pkcs1);

byte[] decrypted = myRsa.Decrypt(encrypted, RSAEncryptionPadding.Pkcs1);

Console.WriteLine("До шифрования: {0}", original);

Console.WriteLine("Зашифрованное: {0}", System.Text.Encoding.UTF8.GetString(encrypted));

Console.WriteLine("После дешифровки: {0}", System.Text.Encoding.UTF8.GetString(decrypted));

}

Как представлено в коде выше после создания экземпляра Rsa возможна сразу же зашифровать и дешифровать данные использую автоматически сгенерированные ключи. Также код представленный выше содержит пример для получения открытого и закрытого ключа в виде строки в формате Xml.

<RSAKeyValue>

<Modulus>6yEjtrItcUq1hoA01xc63EW5/P99kstIobXsxPCUfUODRn2dazzcyhJ5Quhw1oHodlOMvtDN3xJdOTWDbH3xdQ==

</Modulus>

<Exponent>AQAB

</Exponent>

</RSAKeyValue>

Пример сохранённого в файл открытого ключа

<RSAKeyValue>

<Modulus>6yEjtrItcUq1hoA01xc63EW5/P99kstIobXsxPCUfUODRn2dazzcyhJ5Quhw1oHodlOMvtDN3xJdOTWDbH3xdQ==

</Modulus>

<Exponent>AQAB

</Exponent>

<P>9TsbWvgvA2OqPZxUZ96PomUG8rJk2T0SiH6chz65zkc=

</P>

<Q>9XR41YP8/CUoajovRPKWQZou3J23n3usp1acC3v9dGM=

</Q>

<DP>lnjHJ0GD72t3KUjETdu1fKrK4Z5u3RFpTtGJkD7/2b8=

</DP>

<DQ>9Qv2ppNCuigOlR7JmjflslDPgAklDN9XmyhoWT7L5qk=

</DQ>

<InverseQ>gSJ6G275fGFrEMqwSdgJYvmUQhnpCTcX0T3imIVQwOE=

</InverseQ>

<D>z0gMwu+6zehNtP/rFT9eXXd+qgHWAwYAxsapr0hjrzSXv1qS9QJJ+062YbdHc24WZagrKqABfOLQ3hfLXP3JdQ==

</D></RSAKeyValue>

Пример сохранённого в файл открытого ключа

Как видно на представленных выше файлах открытого и закрытого ключа объём ключей отличается, но также совпадают modulus (n) и Exponent (e). Остальные параметры в закрытом ключе совпадают с общепринятыми обозначениями кроме DP=d mod (p-1), DQ=d mode (q-1) и D обозначающего r или закрытый показатель степени.

Xеширования в .Net выполняется проще остальных операций из за отсутствия необходимости в обратном преобразовании. В данном пространстве имён поддерживаются следующие алгоритмы:

* MD5
* SHA256
* SHA384
* SHA512

Пример кода представлен ниже.

byte[] hashValue = new byte[256];

using (SHA256 mysha256 = SHA256.Create())

{

hashValue = mysha256.ComputeHash(File.ReadAllBytes("sha.txt"));

File.WriteAllBytes("hash.txt", hashValue);

}

Как видно из представленного кода после инициализации экземпляра для шифрования нужно вызвать только один метод ComputeHash который выполнит хеширование данных.

Пример захешированных данных представлен ниже

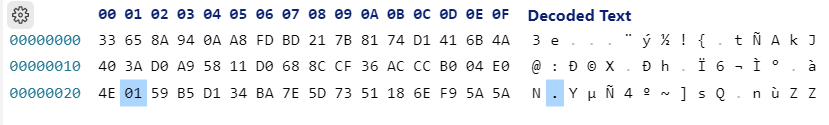


Рисунок 1

**Задание для выполнения:**

1. Ознакомиться с созданием криптографического приложения;
2. Выполнить шифрование, дешифрование и хеширование своей фамилии по указанным алгоритмам. Используя функции работы с файлами сохранить ключи шифрования, результаты шифрования и хеширования.
3. Для выше указанных алгоритмов используя Hex-редактор продемонстрировать ключи шифрования, зашифрованные и захешированные данные.

|  |  |  |
| --- | --- | --- |
| 2 | DES | SHA1 |

using System;

using System.IO;

using System.Security.Cryptography;

using System.Text;

public class CryptoExample

{

private static byte[] EncryptStringToBytes(string plainText, byte[] Key, byte[] IV)

{

using (DESCryptoServiceProvider des = new DESCryptoServiceProvider())

{

des.Key = Key;

des.IV = IV;

using (MemoryStream memoryStream = new MemoryStream())

{

using (CryptoStream cryptoStream = new CryptoStream(memoryStream, des.CreateEncryptor(), CryptoStreamMode.Write))

{

byte[] plaintextBytes = Encoding.UTF8.GetBytes(plainText);

cryptoStream.Write(plaintextBytes, 0, plaintextBytes.Length);

cryptoStream.FlushFinalBlock();

}

return memoryStream.ToArray();

}

}

}

private static string DecryptStringFromBytes(byte[] cipherText, byte[] Key, byte[] IV)

{

using (DESCryptoServiceProvider des = new DESCryptoServiceProvider())

{

des.Key = Key;

des.IV = IV;

using (MemoryStream memoryStream = new MemoryStream(cipherText))

{

using (CryptoStream cryptoStream = new CryptoStream(memoryStream, des.CreateDecryptor(), CryptoStreamMode.Read))

{

byte[] decryptedBytes = new byte[cipherText.Length];

int decryptedByteCount = cryptoStream.Read(decryptedBytes, 0, decryptedBytes.Length);

return Encoding.UTF8.GetString(decryptedBytes, 0, decryptedByteCount);

}

}

}

}

private static byte[] ComputeHash(string input)

{

using (SHA1 sha1 = SHA1.Create())

{

byte[] inputBytes = Encoding.UTF8.GetBytes(input);

return sha1.ComputeHash(inputBytes);

}

}

public static void Main()

{

string originalText = "Банкузов";

// Генерация ключа и вектора инициализации

byte[] key = new byte[8]; // 8 байт для DES

byte[] iv = new byte[8]; // 8 байт для DES

using (DES des = DES.Create())

{

key = des.Key;

iv = des.IV;

}

// Шифрование

byte[] encryptedData = EncryptStringToBytes(originalText, key, iv);

// Дешифрование

string decryptedText = DecryptStringFromBytes(encryptedData, key, iv);

// Хеширование

byte[] hash = ComputeHash(originalText);

// Сохранение ключей, зашифрованных данных и хеша в файлы

File.WriteAllBytes("encryption\_key.bin", key);

File.WriteAllBytes("encryption\_iv.bin", iv);

File.WriteAllBytes("encrypted\_data.bin", encryptedData);

File.WriteAllBytes("hash.bin", hash);

// Демонстрация ключей, зашифрованных данных и хеша в Hex-редакторе

string keyHex = BitConverter.ToString(key).Replace("-", "");

string ivHex = BitConverter.ToString(iv).Replace("-", "");

string encryptedDataHex = BitConverter.ToString(encryptedData).Replace("-", "");

string hashHex = BitConverter.ToString(hash).Replace("-", "");

Console.WriteLine("Encryption Key (Hex): " + keyHex);

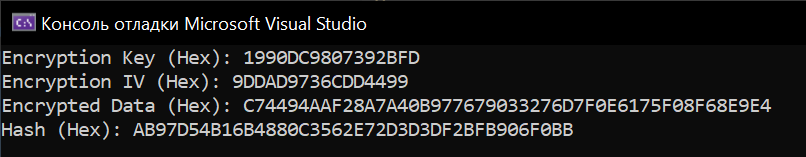
Console.WriteLine("Encryption IV (Hex): " + ivHex);

Console.WriteLine("Encrypted Data (Hex): " + encryptedDataHex);

Console.WriteLine("Hash (Hex): " + hashHex);

}

}



**Практическое задание № 9**

**Тема «Изучение стандартных средств для реализации симметричного и ассиметричного шифрование с использованием SubtleCrypto в js»**

Цель: Изучить интерфейс **SubtleCrypto**, Основные классы и структуры данных, разработать приложение для шифрования файлов использующих симметричные и ассиметричные алгоритмы шифрования

**Криптография в Js**

Web Crypto API - это интерфейс, позволяющий использовать криптографические примитивы для построения систем с использованием криптографии. Данный интерфейсвключают в себя возможность генерировать, использовать и применять пары криптографических ключей; шифровать и дешифровать сообщения; надежно генерировать случайные числа

Некоторые браузеры реализовали интерфейс под названием Crypto, без точной структуры. Чтобы избежать путаницы, методы и свойства этого интерфейса были удалены из браузеров, реализующих Web Crypto API, и все методы Web Crypto API доступны в новом интерфейсе: SubtleCrypto.

Интерфейс SubtleCrypto Web Crypto API предоставляет ряд низкоуровневых криптографических функций. Доступ к функциям SubtleCrypto осуществляется через объёкт Crypto.subtle .

Этот объект содержит набор методов для выполнения общих криптографических функций, таких как шифрование, хеширование, подписывание и генерация ключей. Поскольку все криптографические операции выполняются с необработанными двоичными данными, каждый метод SubtleCrypto имеет дело с типами ArrayBuffer и ArrayBufferView. Из-за того, что строки так часто становятся предметом криптографических операций, классы TextEncoder и TextDecoder будут часто использоваться вместе с SubtleCrypto для преобразования в строки и обратно.

Одной из проблем криптографии является генерация случайных чисел. Если будет использоватся Math.random() то будет сгенерировано псевдослучайное число которое использует генератор PRNG (pseudorandom number generator). Поскольку сгенерированное число внутреннее состояние PRNG то использование данного алгоритма для криптографии неприемлемо. Для решения данной задачи решить эту проблему, криптографически безопасный генератор псевдослучайных чисел (cryptographically secure pseudorandom number generator, CSPRNG) дополнительно включает в себя источник энтропии качестве входных данных, такие системных свойств, которые проявляют непредсказуемое поведение. Это алгоритм медленнее PRNG, но значения, генерируемые CSPRNG, являются остаточно непредсказуемыми для криптографических целей. Код для генерации несколких случайных чисел можно увидеть в листинге 1.

const array = new Uint8Array(1);

for (let i=0; i<5; ++i) {

console.log(crypto.getRandomValues(array));

}

Листинг 1 – Код генерации случайных чисел

**Хеширование**

Хеширование в SubtleCrypto представлено следующими алгоритмами.

* Sha1
* Sha256
* Sha385
* Sha512

Метод SubtleCrypto.digest() используется для создания хеша сообщения. Ниже представлен пример для хеширования сообщения представлен в листинге 2.

const text = 'Hash using sha256';

async function digestMessage(message) {

const encoder = new TextEncoder();

const data = encoder.encode(message);

const hash = await crypto.subtle.digest(**'SHA-256'**, data);

return hash;

}

digestMessage(text)

.then((digestBuffer) => console.log(digestBuffer.byteLength));

Листинг 2 – Пример хеширования данных

Как видно из примера выше то для указания алгоритма хеширования используется первый параметр. Также стоит упомянуть что результат хеширования обычно используется в 16-ой форме. Для преобразования стоит добавить код представленный в листинге 3.

const hashArray = Array.from(new Uint8Array(hash));

const hashHex = hashArray.map((b) => b.toString(16).padStart(2, '0')).join('');

return hashHex;

Листинг 3 – Пример преобразование хеша в 16-ричную форму.

**Генерация ключей**

Генерация случайного CryptoKey выполняется с помощью метода SubtleCrypto.generateKey(algorithm, extractable, keyUsages). В этот метод передается объект params, указывающий целевой алгоритм, логическое значение,указывающее, должен ли ключ извлекаться из объекта CryptoKey, и массив строк — keyUsages, указывающий, с какими методами

Поскольку разные алгоритмы используют разный набор данных для ключей то в первый параметр содержит сооствутсвующее название алгоритма.

* RSA (RSASSA-PKCS1-v1\_5, RSA-PSS, or RSA-OAEP )использует объект RsaHashedKeyGenParams.
* ECDSA и ECDHи спользует объект EcKeyGenParams.
* HMAC использует объект HmacKeyGenParams.
* AES (AES-CTR, AES-CBC, AES-GCM, AES-KW) использует объект AesKeyGenParams

Значение extractable является логическим значением и указывает на возможность экспорта ключа.

Третий параметр keyUsages описывает, с какими алгоритмами можно использовать ключ.

* encrypt: Ключ используется для шифрования сообщений.
* decrypt: Ключ используется для расшифровки сообщений.
* sign: Ключ используется для подписи сообщений.
* verify: Ключ используется для проверки подписанного сообщения.
* deriveKey: Ключ используется для получения ключа.
* deriveBits: Ключ используется для получения битов.
* wrapKey: Ключ используется для упаковки ключа
* unwrapKey: Ключ используется для распоковки ключа.

Пример генерация ключа представлен в листинге 4.

(async function() {

const params = {

name: 'AES-CTR',

length: 128

};

const keyUsages = ['encrypt', 'decrypt'];

const key = await crypto.subtle.generateKey(params, false, keyUsages);

console.log(key);

})();

Листинг 4 – Пример генерации ключей.

**Шифрование и дешифрование**

Объект SubtleCrypto позволяет использовать как открытый ключ, так и симметричные алгоритмы для шифрования и дешифрования сообщений. Это может быть выполнено с использованием методов SubtleCrypto.encrypt() и SubtleCrypto.decrypt() соответственно. Ниже представлена часть кода для шифрования и дешифрования данных где algoIdentifier это название алгоритма и представлена в листинге 5.

const originalPlaintext = (new TextEncoder()).encode('Crypto');

const encryptDecryptParams = {

name: algoIdentifier,

iv: crypto.getRandomValues(new Uint8Array(16))

};

const ciphertext = await crypto.subtle.encrypt(encryptDecryptParams, key,

originalPlaintext);

console.log(ciphertext);

// ArrayBuffer(32) {}

const decryptedPlaintext = await crypto.subtle.decrypt(encryptDecryptParams,

key, ciphertext);

console.log((new TextDecoder()).decode(decryptedPlaintext));

Листинг 5 – Часть кода для шифрования и дешифрования.

**Создание цифровой подписи и проверка сообщений**

Объект SubtleCrypto позволяет использовать алгоритмы с открытым ключом для генерации подписей с использованием закрытого ключа или для проверки подписей с использованием открытого ключа. Они выполняются с использованием методов SubtleCrypto.sign() и SubtleCrypto.verify() соответственно. Для подписания сообщения требуется объект params, чтобы указать алгоритм и любые необходимые значения, частный CryptoKey и ArrayBuffer или ArrayBufferView для подписи. В примере, представленном в листинге 6 можно увидеть процесс создания цифровой подписи и проверки сообщения с цифровой подписью.

async function() {

const keyParams = {

name: 'ECDSA',

namedCurve: 'P-256'

};

const keyUsages = ['sign', 'verify'];

const {publicKey, privateKey} = await crypto.subtle.generateKey(keyParams,

true, keyUsages);

const message = (new TextEncoder()).encode('Mes to sign');

const signParams = {

name: 'ECDSA',

hash: 'SHA-256'

};

const signature = await crypto.subtle.sign(signParams, privateKey, message);

const verified = await crypto.subtle.verify(signParams, publicKey, signature,

message);

console.log(verified); // true

})();

Листинг 6 – Пример создания и проверки сообщений с цифровой подписью.

**Упаковка и распаковка ключа**

Объект SubtleCrypto позволяет упаковывать и распаковывать ключи, чтобы обеспечить передачу по ненадежному каналу. Это выполняется с использованием методов SubtleCrypto.wrapKey() и SubtleCrypto.unwrapKey() соответственно. Для переноса ключа требуется строка форматирования, экземпляр CryptoKey для переноса, CryptoKey для выполнения переноса и объект params для указания алгоритма и любых необходимых значений. В примере представленном в листинге 7 симметричный ключ AES-GCM, упаковывается с помощью AES-KW и, распаковывается обратно.

(async function() {

const keyFormat = 'raw';

const extractable = true;

const wrappingKeyAlgoIdentifier = 'AES-KW';

const wrappingKeyUsages = ['wrapKey', 'unwrapKey'];

const wrappingKeyParams = {

name: wrappingKeyAlgoIdentifier,

length: 256};

const keyAlgoIdentifier = 'AES-GCM';

const keyUsages = ['encrypt'];

const keyParams = {

name: keyAlgoIdentifier,

length: 256};

const wrappingKey = await crypto.subtle.generateKey(wrappingKeyParams,

extractable, wrappingKeyUsages);

console.log(wrappingKey);

const key = await crypto.subtle.generateKey(keyParams, extractable, keyUsages);

console.log(key);

const wrappedKey = await crypto.subtle.wrapKey(keyFormat, key, wrappingKey,

wrappingKeyAlgoIdentifier);

console.log(wrappedKey);

const unwrappedKey = await crypto.subtle.unwrapKey(keyFormat, wrappedKey,

wrappingKey, wrappingKeyParams, keyParams, extractable, keyUsages);

console.log(unwrappedKey);

})()

Листинг 7 – Пример упаковки и распаковки ключа.

1. Ознакомиться с созданием криптографического приложения
2. Выполнить генерацию и вывод в консоль случайный чисел.
3. Выполнить шифрование, дешифрование и хеширование своей фамилии по указанным алгоритмам.

|  |  |  |
| --- | --- | --- |
| 2 | AES-CTR | SHA-256 |

1. Продемонстрировать упаковку и распаковку ключа, полученного в предыдущем задании используя алгоритм AES-KW.
2. Выполнить процедуру подписи сообщения и проверку подлинности с использованием RSA-PSS или ECDSA на выбор.

const crypto = require("crypto");  
const array = new Uint8Array(1);  
  
for (let i = 0; i < 5; ++i) {  
 crypto.*getRandomValues*(array);  
 console.log(array[0]);  
}  
console.log("---------------------------------------------------------------");  
  
function deriveKey(*password*, *salt*, *keyLength*) {  
 return crypto.*pbkdf2Sync*(*password*, *salt*, 100000, *keyLength*, 'sha256');  
}  
  
const encryptionKey = deriveKey('myEncryptionKey', 'mySalt', 32);  
  
*// Функция для упаковки ключа с использованием AES-KW*function wrapKey(*key*, *wrappingKey*) {  
 const cipher = crypto.*createCipheriv*('aes-256-ecb', *wrappingKey*, '');  
 const wrappedKey = Buffer.concat([cipher.update(*key*), cipher.final()]);  
 return wrappedKey;  
}  
  
*// Функция для распаковки ключа с использованием AES-KW*function unwrapKey(*wrappedKey*, *wrappingKey*) {  
 const decipher = crypto.*createDecipheriv*('aes-256-ecb', *wrappingKey*, '');  
 const unwrappedKey = Buffer.concat([decipher.update(*wrappedKey*), decipher.final()]);  
 return unwrappedKey;  
}  
  
*// Функция для шифрования строки с использованием AES-CTR*function encryptAESCTR(*text*, *key*) {  
 const iv = crypto.*randomBytes*(16); *// Генерация случайного инициализационного вектора* const cipher = crypto.*createCipheriv*('aes-256-ctr', *key*, iv);  
 const encrypted = Buffer.concat([cipher.update(*text*), cipher.final()]);  
 return {  
 iv: iv.toString('hex'),  
 encryptedData: encrypted.toString('hex')  
 };  
}  
  
*// Функция для дешифрования строки с использованием AES-CTR*function decryptAESCTR(*encryptedData*, *key*, *iv*) {  
 const decipher = crypto.*createDecipheriv*('aes-256-ctr', *key*, Buffer.from(*iv*, 'hex'));  
 const decrypted = Buffer.concat([decipher.update(Buffer.from(*encryptedData*, 'hex')), decipher.final()]);  
 return decrypted.toString();  
}  
  
*// Функция для хеширования строки с использованием SHA-256*function hashSHA256(*text*) {  
 const hash = crypto.*createHash*('sha256');  
 hash.update(*text*);  
 return hash.digest('hex');  
}  
  
*// Функция для подписи сообщения с использованием RSA-PSS*function signMessageRSA(*message*, *privateKey*) {  
 const sign = crypto.*createSign*('RSA-SHA256');  
 sign.update(*message*);  
 return sign.sign(*privateKey*, 'hex');  
}  
  
*// Функция для проверки подлинности подписи сообщения с использованием RSA-PSS*function verifyMessageRSA(*message*, *signature*, *publicKey*) {  
 const verify = crypto.*createVerify*('RSA-SHA256');  
 verify.update(*message*);  
 return verify.verify(*publicKey*, *signature*, 'hex');  
}  
  
*// Пример использования функций*const surname = 'Банкузов'; *// Замените на свою фамилию  
  
// Упаковка ключа*const wrappedKey = wrapKey(encryptionKey, encryptionKey);  
console.log('Упакованный ключ:', wrappedKey);  
  
*// Распаковка ключа*const unwrappedKey = unwrapKey(wrappedKey, encryptionKey);  
console.log('Распакованный ключ:', unwrappedKey);  
  
*// Шифрование*const encryptedData = encryptAESCTR(surname, unwrappedKey);  
console.log('Зашифрованные данные:', encryptedData);  
  
*// Дешифрование*const decryptedData = decryptAESCTR(encryptedData.encryptedData, unwrappedKey, encryptedData.iv);  
console.log('Расшифрованные данные:', decryptedData);  
  
*// Хеширование*const hashedData = hashSHA256(surname);  
console.log('Хеш данных:', hashedData);  
  
*// Подписание сообщения*const privateKey = crypto.*generateKeyPairSync*('rsa', {  
 modulusLength: 2048,  
 privateKeyEncoding: {  
 type: 'pkcs8',  
 format: 'pem'  
 },  
 publicKeyEncoding: {  
 type: 'spki',  
 format: 'pem'  
 }  
}).privateKey;  
const signature = signMessageRSA(surname, privateKey);  
console.log('Подпись сообщения:', signature);  
  
*// Проверка подлинности подписи*const publicKey = crypto.*createPublicKey*(privateKey);  
const isSignatureValid = verifyMessageRSA(surname, signature, publicKey);  
console.log('Проверка подлинности подписи:', isSignatureValid);  
console.log("---------------------------------------------------------------");

