Основы компьютерной безопасности

Симметричные шифры

Бобровский Андрей группа 20207

Введение

Информационные системы и вообще любое программное обеспечение требует наличие защиты от кибер-атак. Цифровые системы так или иначе работают с данными, а также, в современных реалиях, осуществляют сетевое взаимодействие для обеспечения многопользовательского доступа, возможности распределённой работы или удалённого доступа к данным. Все эти факторы могут создать угрозу целостности используемых данных, угрозу утечки приватных данных, а также угрозу целостности работы самого приложения. Всё это становится возможным, поскольку вредоносное программное обеспечение через общую сеть может получить доступ к используемым приложением данным, или даже загрузиться на устройство, на котором оно работает в виде исполняемого файла, и осуществлять свою вредоносную работу прямо на устройстве.

Для борьбы со всеми этими угрозами важно обеспечить работу необходимых технологий защиты систем - такие, как брандмауэры, антивирусное ПО, технологии шифрования данных, аутентификация и пр.

В данном реферате будут описаны общие определения в компьютерной безопасности и её основные задачи. Затем будет рассмотрено устройство симметричных шифров от общего, до устройства конкретных: *DES, 3DES, AES*.

Определение компьютерной безопасности

Разберёмся с определением компьютерной безопасности:

- **Компьютерная безопасность** - это совокупность защитных технологий информационных систем, целями которых является сохранение *целостности*, *доступности* и *конфиденциальности* ресурсов информационной системы: оборудование, программное обеспечение, информацию/данные и телекоммуникации.

Из данного определения следует три ключевых аспекта компьютерной безопасности:

- **Конфиденциальность** - подразделяется на две составляющих:
  - Конфиденциальность данных - гарантия того, что приватная или конфиденциальная информация не будет доступна не авторизированным пользователям.
  - Приватность - гарантия того, что пользователи, могут собирать и хранить относящуюся к ним информацию, а также контролировать то, кому эта информация может быть передана.
- **Целостность** - данный термин содержит две концепции:
  - целостность данных - гарантия того, что любое изменение данных осуществляется в рамках строго за регламентированных действий.
  - системная целостность - функционирование системы не может быть несанкционированно изменено.
- **Доступность** - гарантия того, что система быстро и безотказно предоставляет данные в соответствии с необходимой для этого авторизацией.

Эти три концепции считаются основными при реализации компьютерной безопасности информационных систем.

Для полноты картины разумно будет добавить ещё два определения:

- **Подлинность** - подразумевает достоверность действий определённого пользователя. Проще говоря, способность однозначно определить, что действия определённого пользователя - это его и только его действия. Например - отправка сообщения в мессенджере.
- **Подотчётность** - протоколирование действий объекта в системе. Это нужно для отслеживания нарушений безопасности до ответственной стороны.

Приведём некоторые примеры приложений, которые реализуют указанные требования.

Например, база медицинских данных обязана отвечать требованиям *конфиденциальности*, эта информация охраняется законом и становится доступной только согласно предусмотренным этим законом протоколам, например - лечащему врачу. Сайт технической спецификации должен отвечать требованиям *целостности*, поскольку для реализации услуг необходимо обладать актуальной и достоверной информацией о соответствующей детали или устройстве. *Доступность* должна быть реализована в высоко нагруженных сервисах, когда большое кол-во пользователей с соответствующей авторизацией присылают запросы на получение данных.

Сложности реализации компьютерной безопасности

Реализация компьютерной безопасности на практике предполагает решение довольно сложных задач. Основные сложности, с которыми на практике сталкивается разработчик, перечислены

ниже:

1. На первый взгляд, требования кажутся вполне простыми: действительно, большинству основных требований к службам безопасности можно присвоить понятные ярлыки: конфиденциальность, аутентификация, неразглашение, целостность. Но механизмы, используемые для удовлетворения этих требований, могут быть довольно сложными.
2. При разработке конкретного механизма безопасности или алгоритма всегда необходимо учитывать потенциальные атаки на эти функции безопасности. Во многих случаях успешные атаки разрабатываются путем совершенно иного подхода к проблеме, что позволяет использовать неожиданную слабость механизма.
3. Разработав различные механизмы безопасности, необходимо решить, где их использовать. Это верно как с точки зрения физического размещения (например, в каких точках сети необходимы определенные механизмы безопасности), так и в логическом смысле [например, на каком уровне или слоях архитектуры, такой как TCP/IP (Протокол управления передачей/Интернет-протокол) должны быть установлены механизмы].
4. Механизмы безопасности обычно включают в себя нечто большее, чем определенный алгоритм или протокол. Они также требуют, чтобы участники владели некоторой секретной информацией (например, ключом шифрования), что вызывает вопросы о создании, распространении и защите этой секретной информации. Также может возникнуть зависимость от коммуникационных протоколов, поведение которых может усложнить задачу разработки механизма безопасности. Например, если для надлежащего функционирования механизма безопасности требуется установить временные ограничения на время прохождения сообщения от отправителя к получателю, то любой протокол или сеть, которые вводят переменные, непредсказуемые задержки, могут сделать такие временные ограничения бессмысленными.
5. Компьютерная безопасность - это, по сути, битва умов между злоумышленником, который пытается найти дыры, и разработчиком или администратором, который пытается их закрыть. Огромное преимущество, которым обладает злоумышленник, заключается в том, что ему нужно найти только одно слабое место, в то время как разработчик должен найти и устранить все слабые места для достижения идеальной безопасности.
6. Безопасность требует регулярного, даже постоянного мониторинга, а это сложно в сегодняшней краткосрочной, перегруженной среде.
7. Безопасность по-прежнему слишком часто является второстепенной задачей, которая должна быть включена в систему после завершения проектирования, вместо того чтобы быть неотъемлемой частью процесса проектирования. (**ЧУТЬ ПОДРОБНЕЕ**)
8. Любые пользователи и даже администраторы служб безопасности рассматривают надежную защиту как препятствие эффективной и удобной работе информационной

системы.

Модель компьютерной безопасности

Для дальнейшего повествования требуется ввести некоторую терминологию:

- **Противник** (агент угрозы) - Объект, который атакует систему или представляет угрозу для нее.
- **Атака** - посягательство на безопасность системы, являющееся результатом угрозы; то есть разумное действие, представляющее собой преднамеренную попытку обойти службы безопасности и нарушить политику безопасности системы. Тот, кто осуществляет атаку, называется **злоумышленником**.
- **Контрмера** - Действие, устройство, процедура или метод, которые уменьшают угрозу, уязвимость или атаку путем их устранения, предотвращения, минимизации вреда, который они могут причинить, или путем обнаружения и сообщения о них, чтобы можно было предпринять соответствующие действия.
- **Риск** - ожидание потерь, выраженное как вероятность того, что конкретная угроза воспользуется конкретной уязвимостью с конкретным вредным результатом.
- **Политика безопасности** - набор правил и практик, которые определяют то, как система предоставляет услуги безопасности для защиты конфиденциальных и критически важных *системных ресурсов*.
- **Системный ресурс** - данные, содержащиеся в информационной системе; услуга, предоставляемая системой; возможности системы, такие как: вычислительная мощность, полоса пропускания канала связи; или элемент системного оборудования (т.е. системный компонент — аппаратное обеспечение, встроенное ПО, документация и пр.); или объект, в котором размещаются системные операции и оборудование.
- **Угроза** - потенциальная возможность нарушения безопасности, которая существует при наличии обстоятельств, которые могут нарушить безопасность и причинить вред. То есть угроза - это возможная опасность, которая может использовать уязвимость.
- **Уязвимость** - недостаток или слабая сторона в проектировании, внедрении или эксплуатации системы, которые могут быть использованы для нарушения её политики безопасности.

Начнём с более подробного рассмотрения понятия *системные ресурсы*. Именно их пользователь и обладатель стремятся защитить. Эти ресурсы можно разделить на следующие категории:

- **Аппаратное обеспечение** - части компьютерной системы, физические устройства обработки хранения и передачи данных.

- **Программное обеспечение** - операционная система, системные утилиты и приложения.
- **Данные** - файлы и базы данных, а также данные, связанные с безопасностью, такие как файлы паролей.
- **Средства связи и сети** - каналы связи в локальных и глобальных сетях: мосты, маршрутизаторы и так далее.

Ресурсы компьютерной системы могут быть подвержены определённым *уязвимостям*, основные из которых это:

- **повреждение** - часть системы более не может правильно выполнять свою функцию, либо данные некорректны.
- **утечка** - данные попали к лицу, по протоколу безопасности, не имеющему доступ к этим данным.
- **недоступность** - либо доступ к ресурса с очень медленной скоростью.

Эти три общих типа уязвимости соответствуют концепциям целостности, конфиденциальности и доступности, перечисленным ранее в этом разделе. Различным типам уязвимостей, в свою очередь, соответствуют угрозы, способные использовать эти уязвимости. Угрозы представляют *атаки* на безопасность. Атаки можно разделить на два типа:

- **Активные атаки** - попытка изменить системные ресурсы или повлиять на их работу.
- **Пассивные атаки** - попытка использовать информацию компьютерной системы, не влияющей на системные ресурсы.

Атаки также можно классифицировать в зависимости от источника:

- **внутренние атаки** - инициируются субъектом, находящимся внутри периметра безопасности *инсайдером*. Инсайдер имеет право доступа к системным ресурсам, но использует их способом, не одобренным теми, кто предоставил разрешение.
- **внешние атаки** - инициируются за пределами периметра безопасности неавторизованным или нелегитимным пользователем системы *посторонним*.

И, наконец, *контрмера* - это любые средства, применяемые для борьбы с атакой на систему безопасности. Когда предотвратить атаку невозможно либо это терпит неудачу, цель состоит в том, чтобы обнаружить атаку и затем оправиться от ее последствий.

Перейдём к более подробному рассмотрению угроз и атак. Сначала мы рассмотрим типы угроз безопасности, с которыми необходимо бороться, а затем приведем несколько примеров типов угроз, применимых к различным категориям ресурсов компьютерной системы.

Для каждого типа атаки сопоставим возможные последствия для компьютерной системы:

последствия	атака
несанкционированное разглашение данных	**раскрытие** - конфиденциальные данные связываются с не авторизированным субъектом. **перехват** - неавторизованный субъект получает прямой доступ к конфиденциальным данным, передаваемым между авторизованными источниками и пунктами назначения. **вывод** - неавторизованный субъект косвенно получает доступ к конфиденциальным данным (но не обязательно к самому содержанию сообщения), исходя из характеристик или побочных продуктов сообщений.
обман - событие, при котором уполномоченный орган получит ложные данные и поверит, что они соответствуют действительности	**маскарад** - выдача неавторизированного субъекта за авторизированный. **фальсификация** - предоставление ложных данных.
нарушение - событие, которое препятствуют корректной работе системных функций	**вывод из строя** - отключение системного компонента. **повреждение** - нежелательно изменяет работу системы путем неблагоприятного изменения системных функций или данных. **препятствие** - действие угроз, которое прерывает предоставление системных служб
узурпация - событие, которое приводит к контролю системных функций не авторизированным лицом	**незаконное присвоение** - объект несанкционированно берёт на себя контроль над системным ресурсом. **неправильное использование**, результатом которого является выполнение системной функции, наносящей ущерб безопасности.

Фундаментальные принципы дизайна систем безопасности

Несмотря на годы исследований и разработок, так и не удалось разработать методы проектирования и внедрения систем безопасности, которые систематически исключали бы недостатки безопасности и предотвращали все несанкционированные действия. В отсутствие таких надежных технологий полезно иметь набор широко согласованных принципов проектирования, которыми можно руководствоваться при разработке механизмов защиты.

В качестве таких фундаментальных принципов проектирование систем безопасности можно перечислить следующее:

- Компактность реализации
- Отказоустойчивые значения по умолчанию
- Полное посредничество
- Открытый дизайн
- Разделение привилегий
- Наименьшая привилегия
- Наименьший общий механизм
- Прозрачность
- Изоляция
- Инкапсуляция
- Модульность
- Многослойность
- Интуитивность

Компактность реализации

Данный принцип означает, что разработка мер безопасности, встроенных как в аппаратное, так и в программное обеспечение, должна быть максимально простой и компактной.

Мотивация для применения этого принципа заключается в том, что относительно простую и компактную конструкцию легче тщательно протестировать и верифицировать. К тому же, при сложном дизайне у противника появляется гораздо больше возможностей обнаружить едва заметные слабые места для использования, которые может быть трудно обнаружить заранее. Чем сложнее механизм, тем больше вероятность того, что он будет содержать уязвимые места.

Простые механизмы, как правило, имеют меньше уязвимых недостатков и требуют меньшего технического обслуживания. Кроме того, поскольку упрощаются вопросы управления его конфигурацией, то обновление или замена простого механизма становится менее трудоемким процессом.

На практике это, пожалуй, самый сложный принцип для соблюдения, поскольку создать грамотный компактный дизайн, без лишних надстроек в архитектуре, который будет удовлетворять всем необходимым требованиям безопасности довольно сложно.

Отказоустойчивые значения по умолчанию

Этот принцип означает, что решение о доступе принимается основываясь на разрешении. То есть, ситуация по умолчанию - это отсутствие доступа, и система защиты определяет условия

при которых доступ разрешён. Режим сбоя для этого подхода предоставляет гораздо лучший сценарий, чем в случае, если поведением по умолчанию было бы разрешение доступа.

Полное посредничество

Полное посредничество означает, что каждый доступ должен быть сверен с механизмом контроля доступа. Системы не должны полагаться на решения о доступе, полученные из кэша. Чтобы полностью реализовать полное посредничество, каждый раз, когда пользователь считывает поле или запись в файле или элемент данных в базе данных, система должна осуществлять контроль доступа. Этот ресурсоёмкий подход используется редко.

Открытый дизайн

Открытый дизайн означает, что дизайн механизма безопасности должен быть открытым, а не секретным. Например, хотя ключи шифрования должны быть секретными, алгоритмы шифрования должны быть открыты для общественного контроля. Затем алгоритмы могут быть рассмотрены многими экспертами, и, следовательно, пользователи могут быть уверены в них с высокой степенью достоверности. Это философия, лежащая в основе программы Национального института стандартов и технологий (NIST) по стандартизации алгоритмов шифрования и хеширования.

Разделение привилегий

Разделение привилегий - это практика, при которой для получения доступа к ресурсу с ограниченным доступом требуется несколько атрибутов привилегий. Хорошим примером этого является многофакторная аутентификация пользователя, которая требует использования нескольких методов, таких как пароль и смарт-карта.

Этот термин теперь также применяется к любому методу, при котором программа разделяется на части, ограниченные конкретными привилегиями, необходимыми им для выполнения конкретной задачи.

Используется для уменьшения потенциального ущерба от атаки на компьютерную безопасность.

Наименьшая привилегий

Принцип наименьшей привелегии подразумевает, что каждый процесс и каждый пользователь системы должны работать, используя наименьший набор привилегий, необходимых для выполнения задачи.

В более общем плане, любая система контроля доступа должна предоставлять каждому пользователю только те привилегии, которые авторизованы для этого пользователя. Существует

также временный аспект принципа наименьших привилегий. Например, системные программы или администраторы, обладающие особыми привилегиями, должны пользоваться ими только в случае необходимости; когда они выполняют обычные действия, эти привилегии должны быть отменены.

Наименьший общий механизм

Данный принцип означает, что дизайн должен сводить к минимуму функции, совместно используемые разными пользователями. Он помогает сократить количество непреднамеренных каналов связи и уменьшает количество аппаратного и программного обеспечения, от которого зависят все пользователи, тем самым облегчая проверку наличия каких-либо нежелательных последствий для безопасности.

Прозрачность

Подразумевает, что механизмы безопасности не должны чрезмерно вмешиваться в работу пользователей, в то же время удовлетворяя потребности тех, кто разрешает доступ. Там, где это возможно, механизмы безопасности должны быть прозрачными для пользователей системы или, создавать минимальные препятствия.

Изолированность

Это принцип, который применяется в трех контекстах. Во-первых, системы публичного доступа должны быть изолированы от критически важных ресурсов (данных, процессов и т.д.), чтобы предотвратить их закрытие или несанкционированное вмешательство.

Во-вторых, процессы и файлы отдельных пользователей должны быть изолированы друг от друга, за исключением случаев, когда это явно желательно. Все современные операционные системы предоставляют средства для такой изоляции, так что отдельные пользователи имеют отдельное, изолированное пространство процессов, пространство памяти и файловое пространство с защитой от несанкционированного доступа.

И, наконец, механизмы безопасности должны быть изолированы в том смысле, что доступ к этим механизмам защищён. Например, логическое управление доступом может обеспечивать средство изоляции криптографических программ от других частей системы и для защиты этих программ от взлома, а ключей - от замены или раскрытия.

Инкапсуляция

Инкапсуляцию можно рассматривать как специфическую форму изоляции, основанную на объектно-ориентированных принципах. Защита обеспечивается путем инкапсуляции процедур и объектов данных в отдельный домен таким образом, что внутренняя структура объекта

доступна только процедурам защищаемой подсистемы, и сами процедуры могут вызываться только в определенных точках входа в домен.

Модульность

Модульность в контексте безопасности относится как к разработке функций безопасности в виде отдельных защищенных модулей, так и к использованию модульной архитектуры для проектирования и реализации механизмов безопасности.

Многочисленные протоколы и приложения используют криптографические функции. Вместо реализации таких функций в каждом протоколе или приложении более безопасный дизайн обеспечивается за счет разработки общего криптографического модуля, который может быть вызван многочисленными протоколами и приложениями.

Каждый механизм безопасности должен быть способен поддерживать переход на новую технологию или обновление новых функций, не требуя полной перестройки системы.

Многослойность

При использовании множества перекрывающихся подходов к защите сбой или обход любого индивидуального подхода не оставит систему незащищенной.

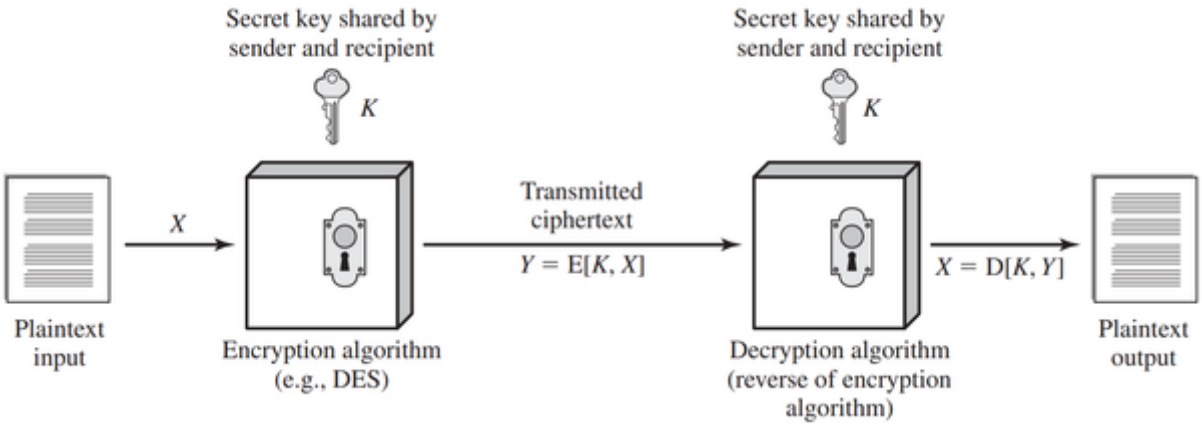
Интуитивность

Данный принцип подразумевает, что цели механизмов безопасности должны быть интуитивно понятными для пользователей. Например, механизм авторизации должен быть достаточно прозрачным. Также, у пользователя должно быть понимание того, как цели системы безопасности соотносятся с механизмами, которые их реализуют.

Криптографический инструментарий

Важнейшим элементом служб компьютерной безопасности является использование криптографических алгоритмов. В качестве примера, здесь будет представлен обзор различных алгоритмов симметричного шифрования.

Симметричное шифрование



- **Симметричное шифрование** - способ шифрования, в котором для шифрования и расшифрования применяется один и тот же криптографический ключ.

Симметричное шифрование оперирует несколькими определениями:

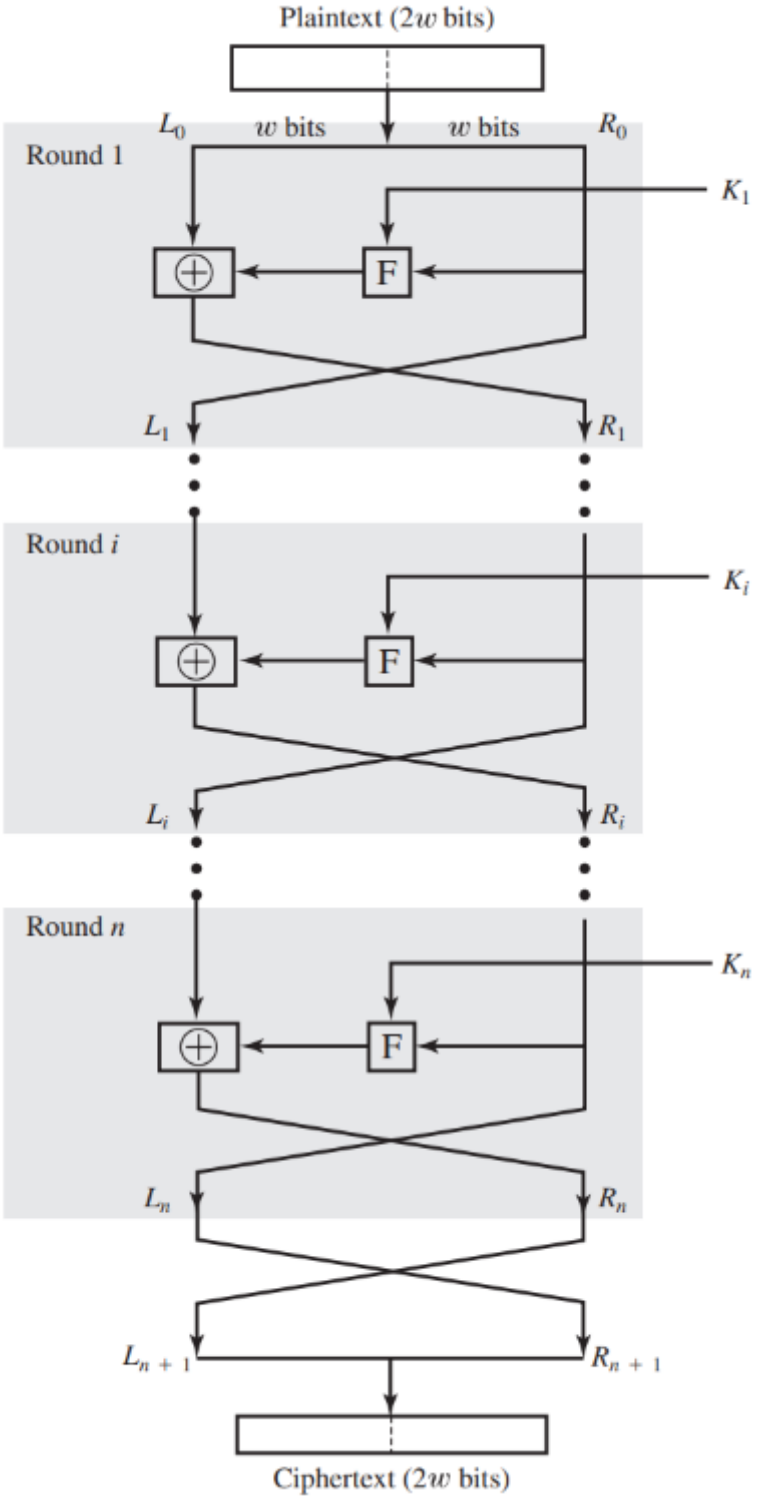
- **открытый текст** - исходный текст сообщения.
- **алгоритм шифрования** - алгоритм преобразования исходного текста в зашифрованный вид.
- **секретный ключ** - включается в алгоритм шифрования, замены и преобразования выполняемые алгоритмом зависят от секретного ключа.
- **шифротекст** - зашифрованное выходное сообщение, зависит от открытого текста и секретного ключа.
- **алгоритм дешифрования** - набор операций обратный алгоритму шифрования, для преобразования шифротекста обратно в исходное сообщение.

у безопасного алгоритма симметричного шифрования существует два требования:

1. Противник не должен быть в состоянии расшифровать зашифрованный текст или обнаружить ключ, даже если он или она владеет несколькими зашифрованными текстами вместе с открытым текстом, из которого был создан каждый зашифрованный текст.
2. Отправитель и получатель должны получить копии секретного ключа безопасным способом и должны хранить этот ключ в безопасности. Если кто-то может обнаружит ключ и узнает алгоритм шифрования, все сообщения, использующие этот ключ, будут доступны для чтения.

Наиболее часто используемыми алгоритмами симметричного шифрования являются блочные шифры. Блочный шифр обрабатывает вводимый открытый текст блоками фиксированного размера и создает блок зашифрованного текста одинакового размера для каждого блока открытого текста.

Наиболее важными симметричными алгоритмами, все из которых являются блочными шифрами, являются Стандарт шифрования данных (DES), тройной DES и расширенный стандарт шифрования (AES)



Входными данными для алгоритма шифрования являются блок открытого текста длиной $2w$ бит и ключ K. Блок открытого текста разделен на две половины, L_0 и R_0. Две половины данных проходят через n раундов обработки, а затем объединяются для получения блока зашифрованного текста.

Каждый i'й раунд имеет в качестве входных данных L_{i-1} и R_{i-1}, полученные из предыдущего раунда, а также подключ K_i, полученный из общего K. В общем, все подключи K_i отличаются от K и друг от друга и генерируются из исходного ключа с помощью алгоритма генерации подключей.

Все раунды имеют одинаковую структуру. $L_i = R_{i-1}$; $R_i = XOR(L_{i-1}, F(K_{i-1}, R_{i-1}))$. Где F - это *раундовая функция*.

Дешифрование шифротекста в открытый происходит в обратном порядке: $R_{i-1} = L_i$; $L_{i-1} = XOR(R_i, F(R_{i-1}, K_{i-1}))$. Напомним, что св-во XOR следующее - $a\ XOR\ b\ XOR\ b = a$, из него и следует предыдущее равенство.

DES

Рассмотрим один из базовых алгоритмов симметричного шифрования под названием *DES - Data Encription Standard*. Данный алгоритм описывается следующим образом: открытый текст разбивается на блоки по 64 бита. Сам алгоритм состоит из 16 раундов и имеет секретный ключ длинной 56 бит. Из этого ключа генерируется 16 подключей соответственно. Дальше алгоритм применяется для каждого блока открытого текста, как описано в предыдущем разделе.

Тройной DES

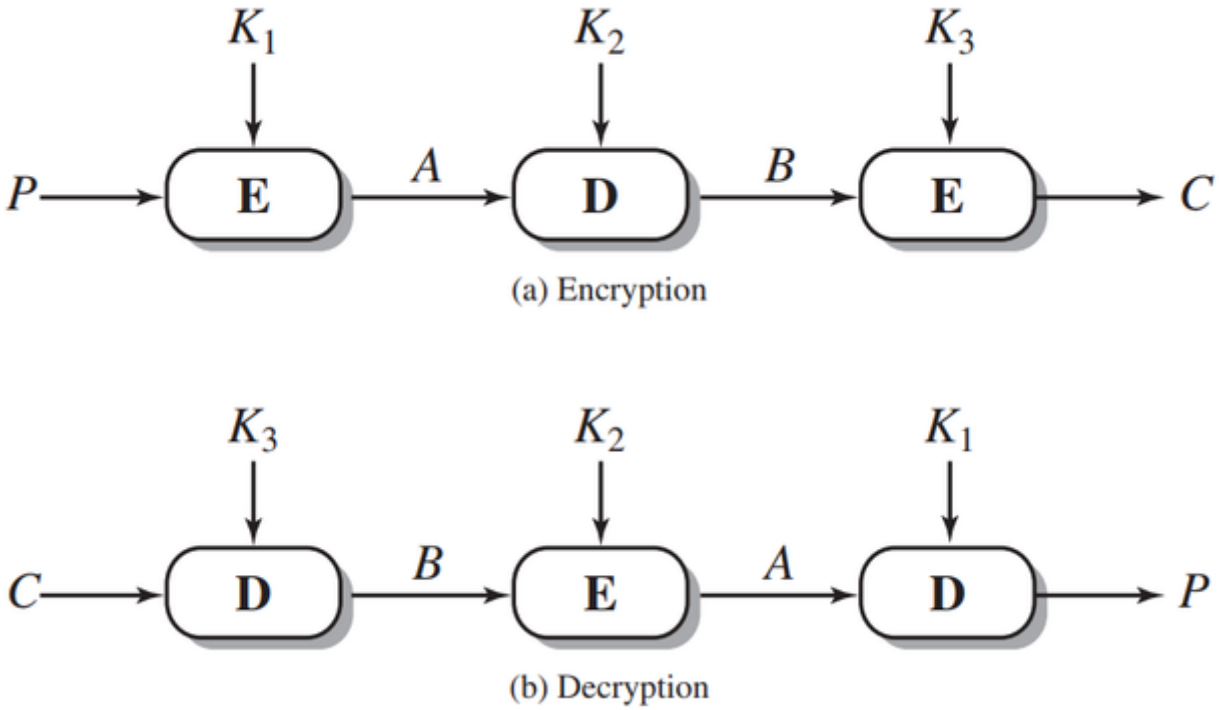
Тройной DES представляет из себя композицию из обычных DES. Перед рассмотрением устройства данного алгоритма введём некоторые обозначения:

- C - шифротекст.
- P - открытый текст.
- $E[K, X]$ - шифрование текста X ключом K алгоритма DES
- $D[K, Y]$ - дешифрование Y ключом K алгоритма DES

Всего существует три варианта *Triple DES*:

- **DES-EEE3** (операции шифрование-шифрование-шифрование)
  - шифрование - $C = E[K_3, E[K_2, E[K_1, P]]]$
  - дешифрование - $P = D[K_1, D[K_2, D[K_3, \mathrm{C}]]]$
- **DES-EDE3** (операции шифровка-расшифровка-шифровка с тремя разными ключами)
  - шифрование - $C = E[K_3, D[K_2, E[K_1, P]]]$
  - дешифрование - $P = D[K_1, E[K_2, D[K_3, \mathrm{C}]]]$
- **DES-EEE2** (аналогично первому варианту, за исключением того, что на первом и третьем шаге используется одинаковый ключ)
  - шифрование - $C = E[K_1, E[K_2, E[K_1, P]]]$

    - дешифрование - $P = D[K_1, D[K_2, D[K_1, \mathrm{C}]]]$
- **DES-EDE2** (аналогично второму варианту, за исключением того, что на первом и третьем шаге используется одинаковый ключ)
    - шифрование - $C = E[K_1, D[K_2, E[K_1, P]]]$
    - дешифрование - $P = D[K_1, E[K_2, D[K_1, \mathrm{C}]]]$



Благодаря трем различным ключам эффективная длина ключа *DES-EDE3* / *DES-EEE3* составляет 168 бит. При $K_1 = K_3$ как в *DES-EEE2* / *DES-EDE2*, эффективная длина ключа будет равна 112 бит.

AES

Расширенный стандарт шифрования (*AES - Advanced Encryption Standard*) был выпущен в качестве федерального стандарта обработки информации (FIPS 197). Он предназначен для замены DES и triple DES алгоритмом, который является более безопасным и эффективным.

вспомогательные определения и процедуры:

- **шифрключ** - секретный ключ, из которого производятся *раундовые ключи*.
- **State** - промежуточный результат шифрования, который может быть представлен как прямоугольный массив байтов, имеющий 4 строки и Nb столбцов.
- **Key Expansion** - процедура генерации раундовых ключей из шифрключа.
- **Nb** - число столбцов, составляющих *State*.
- **раундовый ключ** - применяются к State для шифрования и расшифрования.

- **S-box** - нелинейная таблица замен, использующаяся в нескольких трансформациях замены байтов и в процедуре Key Expansion для взаимно-однозначной замены значения байта.
- **Nk** - число 32-битных слов, составляющих шифроключ. Для AES Nk = 4, 6, или 8.
- **Nr** - число раундов, которое является функцией Nk и Nb. Для AES Nr = 10, 12 или 14.
- **AddRoundKey()** - трансформация при шифровании и дешифровании, при которой осуществляется XOR для State и раундового ключа.
- **MixColumns()** - трансформация при шифровании, которая берёт все столбцы State и смешивает их данные (независимо друг от друга), чтобы получить новые столбцы
- **RotWord()** - функция, использующаяся в процедуре Key Expansion, которая берёт 4-байтовое слово и производит над ним циклическую перестановку.
- **ShiftRows()** - трансформации при шифровании, которые обрабатывают State, циклически смещая последние три строки State на разные величины.
- **SubBytes()** - трансформации при шифровании, которые обрабатывают State, используя нелинейную таблицу замещения байтов (S-box), применяя её независимо к каждому байту State.
- **SubWord()** - функция, используемая в процедуре Key Expansion, которая берёт на входе четырёх байтовое слово и, применяя S-box к каждому из четырёх байтов, выдаёт выходное слово.
- **InvMixColumns()** - трансформация при расшифровании, которая является обратной к операции *MixColumns()*.
- **InvSubBytes()** - трансформация при расшифровании, которая является обратной по отношению к *SubBytes()*.
- **InvShiftRows()** - трансформация при расшифровании, которая является обратной по отношению к *ShiftRows()*.

шифрование:

Для AES длина входного блока данных (input) и блока State постоянна и равна 128, а длина шифрключа составляет 128, 192, 256 бит.

С начала *input* копируется в массив *State* по следующему правилу: $state[i, j] = input[i + 4j]$ при $0 \leq i < 4$ и $0 \leq j < Nb$. После этого, к полученному *State* применяется процедура *AddRoundKey()*, после чего, *State* проходит через 10, 12 или 14 раундов трансформации соответственно. Нужно учитывать, что последний раунд несколько отличается от предыдущих.

Отдельный раунд включает в себя последовательное применение процедур *SubBytes*, *ShiftRows*, *MixColumns* и *AddRoundKey* к текущему *State*. Последний раунд имеет небольшое отличие - он не содержит этапа применения процедуры *MixColumns*.

В итоге после завершения последнего раунда *State* копируется в *output* по следующему правилу: $output[i + 4j] = state[i, j]$ при $0 \leq i < 4$ и $0 \leq j < Nb$. Таким образом, мы получаем наш шифротекст.

Перед тем, как рассмотреть каждую упомянутую процедуру отдельно нужно вспомнить теоретические основы, на которых эти процедуры строятся.

Операции в поле $GF(2^8)$: для описания алгоритма используются вычисления в конечном поле Галуа, построенном как расширение поля $GF(2) = \{0, 1\}$ по модулю неприводимого многочлена $m(x) = x^8 + x^4 + x^3 + x + 1$. Элементами поля являются многочлены вида $b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x^1 + b_0$ степень которых меньше 8, а коэффициенты b_i берутся из множества: $\{0, 1\}$. Все операции в поле выполняются по модулю $m(x)$.

Таким образом, байт $b_7b_6b_5b_4b_3b_2b_1b_0$ можно представить, как многочлен в поле Галуа $GF(2^8)$ с соответствующими коэффициентами: $01101011 \sim x^6 + x^5 + x^3 + x + 1$. Таким образом, все операции с байтами происходят так - каждый байт представляется в виде многочлена, затем над ним производится указанная операция в поле Галуа, после чего результат преобразуется обратно в байт.

Ещё нужно упомянуть нахождение обратного элемента. Для любого ненулевого многочлена в поле Галуа $b(x)$ существует обратный по умножению элемент $b^{-1}(x)$ то есть: $b^{-1}(x)b(x) \equiv 1 \ (mod \ m(x))$. Ищется он при помощи расширенного Алгоритма Евклида, через него находятся такие многочлены $a(x)$ и $c(x)$, что $a(x)b(x) + c(x)m(x) = 1 \implies b^{-1}(x) \equiv a(x) \ (mod \ m(x))$

Также, в данном алгоритме будет использоваться, многочлены с коэффициентами из поля $GF(2^8)$, т.е. их коэффициенты являются байтами. Операции с такими многочленами не представляют особой сложности. *Сложение* осуществляется путём применения операции XOR к соответствующим коэффициентам. Умножение соответствует умножению многочленов, где при группировке при соответствующих степенях x используется XOR, а коэффициенты перемножаются как многочлены с коэффициентами из поля $GF(2)$:

$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$
$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$
$c(x) = a(x)b(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x^1 + c_0$
$c_0 = a_0b_0$
$c_1 = a_1b_0 \oplus a_0b_1$
$c_2 = a_2b_0 \oplus a_1b_1 \oplus a_0b_2$
$c_3 = a_3b_0 \oplus a_2b_1 \oplus a_1b_2 \oplus a_0b_3$

$c_4 = a_3b_1 \oplus a_2b_2 \oplus a1b_3$
$c_5 = a_3b_2 \oplus a_2b_3$
$c_6 = a_3b_3$

Рассмотрим отдельно каждую упомянутую процедуру:

- **SubBytes()** - данная операция представляет нелинейную замену байтов, выполняемую независимо с каждым байтом матрицы *State*. Данная замена обратима и построена путем комбинации двух преобразований над входным байтом:
  1. нахождение обратного элемента в поле Галуа для этого байта, так, как было описано выше.
  2. выполнение аффинного преобразования: умножение инвертированного байта на многочлен $a(x) = x^4 + x^3 + x^2 + x + 1$, затем происходит суммирование с многочленом $b(x) = x^6 + x^5 + x + 1$ в поле $F_2[x]/x^8 + 1$

  В матричном виде, данное преобразование записывается следующим образом:

$$y = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 10001111 \\ 11000111 \\ 11100011 \\ 11110001 \\ 11111000 \\ 01111100 \\ 00111110 \\ 00011111 \end{pmatrix} * \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}^{-1} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

  нелинейность данного преобразования обусловлена нелинейностью взятия обратного элемента x^{-1}, а обратимость - обратимостью матрицы.

  Созданную на основе этой операции специальную таблицу замен байтов в шестнадцатеричной системе называют **S-box**. Она представляет из себя матрицу 16x16, каждый элемент которой кодирует соответствующий байт Например, *SubBytes*(8A)={7E}:

	S-box															
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F**
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	FO	AD	D4	A2	AF	9C	A4	72	CO
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	AO	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	DO	EF	AA	FB	43	4D	33	85	45	F9	02	F7	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DE
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	CB	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	IF	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	OE	61	35	57	B9	86	C1	1D	9E
E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	OD	BF	E6	42	68	41	99	2D	OF	BO	54	BB	16

- **ShiftRows()** - операция применяется к строкам матрицы *State* для введение в шифр дополнительной диффузии. Первая строка остаётся неподвижной, а оставшиеся 3 циклически сдвигаются влево на 1, 2 и 3 позиции соответственно. Очевидно данное преобразование обратимо, для этого просто требуется осуществить обратные сдвиги.

$$\begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{10} & s_{11} & s_{12} & s_{13} \\ s_{20} & s_{21} & s_{22} & s_{23} \\ s_{30} & s_{31} & s_{32} & s_{33} \end{pmatrix} -> \begin{pmatrix} s_{00} & s_{01} & s_{02} & s_{03} \\ s_{11} & s_{12} & s_{13} & s_{10} \\ s_{22} & s_{23} & s_{20} & s_{21} \\ s_{33} & s_{30} & s_{31} & s_{32} \end{pmatrix}$$

- **MixColumns()** - при помощи этой операции байты матрицы *State* перемешиваются. Каждый столбец матрицы *State* принимается за многочлен над полем $GF(2^8)$. Т.е. коэффициенты данного многочлена являются байтами. Далее, данный многочлен умножается на фиксированный $c(x) = c_3x^3 + c_2x^2 + cx + c_0 = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}c_0$ по модулю многочлена $x^4 + 1$. Данную операцию можно записать в виде матрицы и она, также, является обратимой:
  s_i - элементы столбца матрицы *State*

$$\begin{pmatrix} c_0c_1c_2c_3 \\ c_1c_0c_3c_2 \\ c_2c_1c_0c_3 \\ c_3c_2c_1c_0 \end{pmatrix} * \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} 01\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} * \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \end{pmatrix} = \begin{pmatrix} s'_0 \\ s'_1 \\ s'_2 \\ s'_3 \end{pmatrix}$$

- **AddRoundKey()** - данная функция побитово складывает (применяет XOR) элементы переменной *RoundKey* и элементы переменной *State* по принципу: i-й столбец данных $i = 0..3$ складывается с определенным 4-байтовым фрагментом *расширенного ключа*

$W[4r + 1]$, где r – номер раунда алгоритма. При шифровании первое сложение раундового ключа происходит до первого выполнения операции *SubBytes*.
Расширенный ключ вырабатывается из ключа шифра K при помощи процедуры расширения. Работает это следующим образом: Каждый раундовый ключ имеет длину 128 бит, или 4 четырёхбайтных слова. Всего раундов 11 - столько же и ключей. В совокупности они дают один *расширенный ключ*. Первые четыре слова по 4 байта каждое $w_0, ..., w_3$ заполняются байтами ключа шифра K. Таким образом, *раундовым ключом* для нулевого раунда будет являться ключ шифра.

Новые слова $w_{i+4}, w_{i+5}, w_{i+6}, w_{i+7}$ следующего раундового ключа определяются на основе предыдущего $w_i, w_{i+1}, w_{i+2}, w_{i+3}$ по следующему правилу:
$w_{i+5} = w_{i+4} \oplus w_{i+1}$
$w_{i+6} = w_{i+5} \oplus w_{i+2}$
$w_{i+7} = w_{i+6} \oplus w_{i+3}$

Первое слово w_{i+4} в каждом раундовом ключе изменяется подругому:
$w_{i+4} = w_i \oplus g(w_i + 3)$

Действие функции g сводится к последовательному применению следующих трёх шагов:

1. циклический сдвиг четырёхбайтного слова влево на один байт.
2. замена каждого байта полученного слова на соответствующий байт из таблицы **S-box**.
3. суммирование байтов слова по модулю 2 с раундовой постоянной $R_{con}[i] = (RC[i], 0, 0, 0)$. $RC[1] = 1,\ RC[i] = 2 * RC[i - 1], i = 1, ..10:$ на каждый раунд соответственно.

Цель суммирования с раундовыми константами – разрушить любую симметрию, что может возникнуть на разных этапах разворачивания ключа и привести к появлению слабых ключей, как в алгоритме DES.

дешифрование:

Для расшифрования шифротекста в алгоритме *AES* все используемые шифрующие преобразования могут быть инвертированы и применены в обратном порядке.

Заключение

В данном реферате был произведён обзор основных определений компьютерной безопасности, задач и проблем, возникающих при её реализации в различных системах. Также, мы познакомились с симметричными шифрами, изучили их общее устройство, а также разобрали нескольких распространённых примеров шифров этого типа: *DES, 3DES, AES*.

Ссылки

- (Triple DES: [сайт]. URL: https://ru.wikipedia.org/wiki/Triple_DES)
- (AES: [сайт]. URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard)
- Сушко С.А Практическая криптология лекция 9; общее описание криптоалгоритма AES
- William Stallings, Lawrie Brown. - UNSW Canberra at the Australian Defence Force Academy: Computer Security
  Principles and Practice (Third Edition)