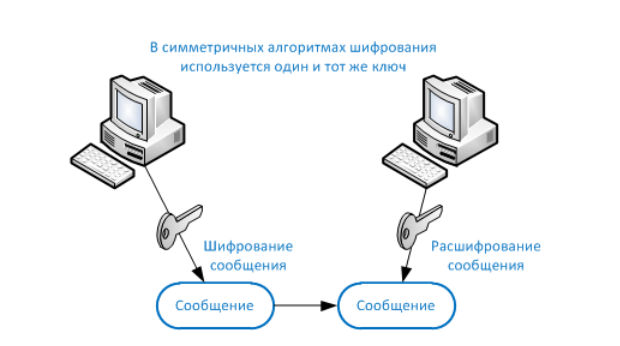
**1. Основные понятия информационной безопасности. Основные составляющие.** **Угрозы. Основные понятия: Защита информации** – это комплекс мероприятий, направленных на обеспечение  информационной безопасности  Информационная безопасность – защищенность информации и поддерживающей  инфраструктуры от случайных или преднамеренных воздействий естественного или  искусственного характера, которые могут нанести неприемлемый ущерб субъектам  информационных отношений, в том числе владельцам и пользователям информации и  поддерживающей инфраструктуры. Основные составляющие:   Доступность – это возможность за приемлемое время получить требуемую  информационную услугу. Целостность – актуальность и непротиворечивость информации, её защищенность от  разрушения и несанкционированного изменения. Целостность можно подразделить на:   ‐ статическую (неизменность информационных объектов)    ‐ динамическая (корректное выполнение сложных действий(транзакций)). Конфиденциальность – это защита от несанкционированного доступа к информации. Угрозы: Угроза – это потенциальная возможность определенным образом нарушить  информационную безопасность. Попытка реализации угрозы называется атакой, а тот, кто предпринимает такую попытку, ‐ злоумышленником. Потенциальные  злоумышленники называются источниками угрозы. Угроза чаще всего является  следствием наличия уязвимых мест в защите ИС (информационных систем).  Некоторые  угрозы нельзя считать следствием каких‐то ошибок или просчетов; они существуют в силу  самой природы современных ИС. Например, угроза отключения электричества   существует из‐за зависимости аппаратного обеспечения ИС от электричества. Промежуток  времени от момента, когда появляется возможность использовать слабое место, и до  момента, когда пробел ликвидируется называется окном опасности. Наиболее распространенные типы угроз:    ‐ Модификация данных (Tampering with data) Атаки этого типа предусматривают  злонамеренную порчу данных.   ‐ Отказ от авторства (Repudiation) Контрагент отказывается от совершенного им  действия (или бездействия), пользуясь тем, что у другой стороны нет никакого способа  доказать обратное.   ‐ Разглашение информации (Information disclosure) Подразумевается раскрытие  информации лицам, доступ к которой им запрещен, например, прочтение пользователем  файла, доступ к которому ему не предоставлялся, а  также способность злоумышленника  считывать данные при передаче между компьютерами.   ‐ Отказ от обслуживания (Denial of service) В атаках такого типа взломщик пытается  лишит доступа к сервису правомочных пользователей, например, сделав Web‐сервер  временно недоступным или непригодным для работы.   ‐ Повышение привилегий (Elevation of privilege) В данном случае  непривилегированный пользователь получает привилегированный доступ, позволяющий  ему «взломать» или даже уничтожить систему.  К повышению привилегий относятся  случаи, когда злоумышленник удачно проникает через защитные средства системы и  становится частью защищенной и доверенной подсистемы.

**2. Криптографические методы защиты информации. Симметричная криптография.**

В криптосистеме, в которой применяется симметричная криптография, отправитель и получатель используют два экземпляра одного и того же ключа для зашифрования и расшифрования информации, как показано на Рисунке 6-8. Таким образом, ключ имеет двойную функциональность и применяется как в процессе зашифрования, так и в процессе расшифрования. Симметричные ключи также называют *секретными ключами*, т.к. этот тип шифрования предполагает, что каждый из пользователей хранит ключ в секрете и надлежащим образом защищает его. Если атакующий получит этот ключ, он сможет расшифровать с его помощью любое перехваченное зашифрованное на нем сообщение.



Каждой паре пользователей, для защищенного с помощью симметричной криптографии обмена данными, требуется два экземпляра одного и того же ключа. Например, если Дену и Ирине нужно обмениваться данными, им обоим нужно получить копию одного ключа. Если Ден хочет также с использованием симметричной криптографии взаимодействовать с Нормом и Дейвом, ему нужно иметь три отдельных ключа – по одному на каждого друга. Это не является большой проблемой, пока Дену не потребуется взаимодействовать с сотней других людей за несколько месяцев и сохранять историю переписки. Ведь это потребует использования соответствующего ключа для переписки с каждым конкретным получателем. В таком случае это может стать сложнейшей задачей. Если десяти людям необходимо безопасно обмениваться данными друг с другом с использованием симметричной криптографии, им потребуется 45 ключей. Если же взаимодействовать нужно ста людям, им потребуется 4950 ключей. Формула для расчета необходимого количества симметричных ключей выглядит следующим образом:

Число ключей = N(N – 1)/2, где N – число абонентов

При использовании симметричных алгоритмов отправитель и получатель используют один и тот же ключ для процессов зашифрования и расшифрования информации. Безопасность таких алгоритмов полностью зависит от того, насколько хорошо пользователи защищают ключи. В таком случае безопасность полностью зависит от персонала, который должен хранить свои ключи в секрете. Если ключ скомпрометирован, все сообщения, зашифрованные на этом ключе, могут быть расшифрованы и прочитаны злоумышленником. В действительности, это еще больше усложняется, поскольку ключи необходимо безопасно распространять и обновлять их при необходимости. Если Дену нужно взаимодействовать с Нормом впервые, Ден должен решить, каким образом безопасно передать Норму ключ. Если он сделает это небезопасно, например, просто отправив ключ по электронной почте, этот ключ может быть легко перехвачен и использован злоумышленником. Поэтому Ден должен передать ключ Норму нестандартным способом. К примеру, Ден может записать ключ на флеш-накопитель и положить его на стол Норму или отправить его Норму с доверенным курьером. Процесс распространения симметричных ключей может стать очень сложной и громоздкой задачей.  
  
Поскольку оба пользователя используют один и тот же ключ для зашифрования и расшифрования сообщений, симметричные криптосистемы могут обеспечить конфиденциальность, но не аутентификацию или [неотказуемость](http://en.wikipedia.org/wiki/Nonrepudiation). Такой криптографический алгоритм не позволит доказать, кто реально отправил сообщение, т.к. оба пользователя используют один и тот же ключ.  
  
Но если симметричные криптосистемы имеют столько недостатков и проблем, почему они используются почти повсеместно? Потому что они обеспечивают очень высокую скорость обработки данных и их очень трудно взломать. Симметричные алгоритмы гораздо быстрее асимметричных. Они могут сравнительно быстро зашифровывать и расшифровывать большие объемы данных. Кроме того, данные, зашифрованные симметричным алгоритмом с использованием длинного ключа, очень сложно вскрыть.  
  
Следующий список описывает сильные и слабые стороны криптосистем с симметричными ключами:  
  
**Сильные стороны:**

* Гораздо быстрее асимметричных систем
* При использовании длинного ключа сложно взломать

**Слабые стороны:**

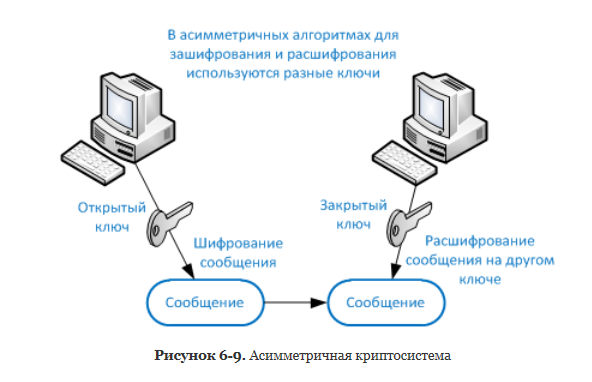
* Требует безопасного механизма передачи ключей
* Каждой паре пользователей нужен уникальный ключ; по мере увеличении количества пользователей, возрастающее число ключей может сделать управление ими просто нереальной задачей
* Обеспечивает конфиденциальность, но не обеспечивает аутентификацию или неотказуемость

Ниже приведены некоторые примеры симметричных алгоритмов, которые будут подробно рассмотрены позднее в разделе «Блочные и поточные шифры».

* [Data Encryption Standard](http://ru.wikipedia.org/wiki/DES) (DES)
* [Triple-DES](http://ru.wikipedia.org/wiki/3DES) (3DES)
* [Blowfish](http://ru.wikipedia.org/wiki/Blowfish)
* [IDEA](http://ru.wikipedia.org/wiki/IDEA)
* [RC4](http://ru.wikipedia.org/wiki/RC4), [RC5](http://ru.wikipedia.org/wiki/RC5) и [RC6](http://ru.wikipedia.org/wiki/RC6)
* [Advanced Encryption Standard](http://ru.wikipedia.org/wiki/Advanced_Encryption_Standard) (AES)

**3. Криптографические методы защиты информации. Ассиметричная криптография.**

В криптографии с симметричными ключами для зашифрования и расшифрования используется один и тот же секретный ключ, тогда как в системах с открытыми ключами для этих целей используются различные (***асимметричные***) ключи. При этом два отличающихся асимметричных ключа связаны между собой математически. Если сообщение зашифровано одним ключом, для его расшифрования требуется другой ключ.  
  
В системах с открытыми ключами, создается пара ключей, один из которых является закрытым, другой – открытым. *Открытый ключ* (public key) может быть известен всем, а *закрытый ключ* (private key) должен знать только его владелец. Часто открытые ключи хранятся в каталогах и базах данных адресов электронной почты, общедоступных всем желающим использовать эти ключи для зашифрования и расшифрования данных при взаимодействии с отдельными людьми. Рисунок 6-9 иллюстрирует использование отличающихся асимметричных ключей.  
Открытый и закрытый ключи асимметричной криптосистемы математически связаны, однако наличие у кого-то открытого ключа другого человека не позволяет узнать соответствующий ему закрытый ключ. Таким образом, если злоумышленник получит копию открытого ключа Боба, это вовсе не значит, что он с помощью какого-то математического волшебства сможет получить соответствующий ему закрытый ключ Боба. Однако, если кто-то получит закрытый ключ Боба, возникнет большая проблема. Поэтому никто кроме владельца не должен иметь доступа к закрытому ключу.



Если Боб зашифровал данные на своем закрытом ключе, получателю потребуется открытый ключ Боба, чтобы расшифровать их. Получатель может не только расшифровать сообщение Боба, но и ответить Бобу зашифрованным сообщением. Для этого ему нужно зашифровать свой ответ на открытом ключе Боба, тогда Боб сможет расшифровать этот ответ с помощью своего закрытого ключа. При использовании асимметричного алгоритма, невозможно зашифровывать и расшифровывать сообщение одним и тем же ключом, эти ключи, хотя и связаны математически, они не совпадают (в отличие от симметричных алгоритмов). Боб может зашифровать данные на своем закрытом ключе, тогда получатель сможет расшифровать их на открытом ключе Боба. Расшифровывая сообщение на открытом ключе Боба, получатель может быть уверен, что сообщение действительно исходит от Боба, ведь сообщение может быть расшифровано на открытом ключе Боба только в том случае, если оно было зашифровано на соответствующем закрытом ключе Боба. Это обеспечивает возможность аутентификации, т.к. Боб является (предположительно) единственным, кто имеет этот закрытый ключ. Если получатель хочет быть уверен, что единственным, кто сможет прочитать его ответ, будет Боб, он должен зашифровать свое сообщение Бобу на его открытом ключе. Тогда только Боб сможет расшифровать это сообщение, поскольку только у него есть необходимый для этого закрытый ключ.  
  
Кроме того, получатель может решить зашифровать данных на своем закрытом ключе, а не на открытом ключе Боба. Что это ему даст? Аутентификацию. Боб будет знать, что сообщение пришло от него и не могло придти ни от кого другого. Если он зашифровывает данные на открытом ключе Боба, это не обеспечит аутентификацию, т.к. кто угодно может получить открытый ключ Боба. Если он использует свой закрытый ключ для зашифрования данных, тогда Боб может быть уверен, что сообщение исходит именно от него. Симметричные ключи не обеспечивают аутентификацию, т.к. обе стороны используют один и тот же ключ, что не может гарантировать, что сообщение исходит от конкретного человека.  
  
Если отправителю в большей степени важна конфиденциальность передаваемой информации, ему следует зашифровать свое сообщение на открытом ключе получателя. Это называют ***безопасным форматом сообщения*** (secure message format), поскольку только человек, имеющий соответствующий закрытый ключ, сможет расшифровать это сообщение.  
  
Если же отправителю в большей степени важна аутентификация, ему следует зашифровывать передаваемые данные на своем закрытом ключе. Это позволит получателю быть уверенным в том, что зашифровал данные именно тот человек, который имеет соответствующий закрытый ключ. Если отправитель шифрует данные на открытом ключе получателя, это не обеспечивает возможность аутентификации, т.к. открытый ключ доступен всем.  
  
Шифрование данных на закрытом ключе отправителя называют ***открытым форматом сообщения*** (open message format), т.к. любой человек может расшифровать эти данные с помощью общедоступного открытого ключа отправителя. Конфиденциальность при этом не обеспечивается.  
  
Оба ключа, как закрытый, так и открытый, могут использоваться и для зашифрования, и для расшифрования данных. Не подумайте, что открытый ключ нужен только для зашифрования, а закрытый – только для расшифрования. При этом следует понимать, что если данные зашифрованы на закрытом ключе, они не могут быть расшифрованы на нем же. Зашифрованные на закрытом ключе данные могут быть расшифрованы на соответствующем ему открытом ключе. И наоборот.  
  
Асимметричный алгоритм работает медленнее, чем симметричный алгоритм, т.к. симметричные алгоритмы выполняют относительно простые математические функции над битами в процессах зашифрования и расшифрования. Они заменяют и перемешивают (перемещают) биты, что не очень сложно и не сильно загружает процессор. Причина их устойчивости к взлому заключается в том, что они выполняют эти функции много раз. Таким образом, в симметричных алгоритмах набор битов проходит более длинную серию замен и перестановок.  
  
Асимметричные алгоритмы медленнее симметричных алгоритмов, т.к. они используют гораздо более сложную математику для выполнения своих функций, что требует больше процессорного времени. Однако асимметричные алгоритмы могут обеспечить аутентификацию и неотказуемость в зависимости от используемого алгоритма. Кроме того, асимметричные системы позволяют использовать более простой и управляемый процесс распространения ключей, по сравнению с симметричными системами и не имеют проблем с масштабируемостью, которые есть у симметричных систем. Причина этих различий в том, что при использовании асимметричных систем вы можете отправлять свой открытый ключ всем людям, с которыми вы хотите взаимодействовать, а не использовать для каждого из них отдельный секретный ключ. Далее, в разделе «Гибридные методы шифрования» в этом Домене мы рассмотрим, как эти две системы могут использоваться совместно для получения наилучшего результата.

**ПРИМЕЧАНИЕ.** Криптография с открытым ключом – это асимметричная криптография. Эти термины взаимозаменяемы.

Ниже указаны сильные и слабые стороны алгоритмов с асимметричными ключами:  
  
**Сильные стороны**

* Лучше процесс распространения ключей, чем в симметричных системах
* Лучше масштабируемость, чем в симметричных системах
* Могут обеспечить аутентификацию и неотказуемость

**Слабые стороны**

* Работают гораздо медленнее симметричных систем
* Выполняют сложные математические преобразования

Ниже приведены примеры алгоритмов с асимметричными ключами.

* [RSA](http://ru.wikipedia.org/wiki/RSA)
* [Криптосистема на основе эллиптических кривых](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%BB%D0%B8%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) (ECC – Elliptic curve cryptosystem)
* [Алгоритм Диффи-Хеллмана](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B8%D1%84%D1%84%D0%B8_%E2%80%94_%D0%A5%D0%B5%D0%BB%D0%BB%D0%BC%D0%B0%D0%BD%D0%B0) Diffie-Hellman
* [Эль Гамаль](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%BB%D0%B8%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) (El Gamal)
* [Алгоритм цифровой подписи](http://ru.wikipedia.org/wiki/DSA) (DSA – Digital Signature Algorithm)
* Knapsack

**4. Инфраструктура открытых ключей. Назначение. Состав. Принцип работы.**

**http://cleric-23.narod.ru/Bos/bos19.htm**

**5. Инфраструктура открытых ключей. Сертификат открытого ключа. Проверка сертификата открытого ключа. Список отозванных сертификатов. Проверка списка отозванных сертификатов.**

**Сертификат открытого ключа** (сертификат [ЭП](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C), сертификат ключа подписи, сертификат ключа проверки электронной подписи (согласно ст. 2 Федерального Закона от 06.04.2011 «Об электронной подписи» № 63-ФЗ)) — электронный или бумажный документ, содержащий открытый ключ, информацию о владельце ключа, области применения ключа, подписанный выдавшим его [Удостоверяющим центром](https://ru.wikipedia.org/wiki/%D0%A3%D0%B4%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%B5%D1%80%D1%8F%D1%8E%D1%89%D0%B8%D0%B9_%D1%86%D0%B5%D0%BD%D1%82%D1%80) и подтверждающий принадлежность [открытого ключа](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC) владельцу.

Открытый ключ может быть использован для организации защищённого канала связи с владельцем двумя способами:

* для проверки [подписи](https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%86%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D0%BE%D0%B4%D0%BF%D0%B8%D1%81%D1%8C) владельца ([аутентификация](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F))
* для [шифрования](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC) посылаемых ему данных ([конфиденциальность](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D1%84%D0%B8%D0%B4%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%81%D1%82%D1%8C))

Существует две модели организации инфраструктуры сертификатов: централизованная ([PKI](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D1%80%D0%B0%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D1%85_%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%B9)) и децентрализованная (реализуемая на основе т. н. [сетей доверия](https://ru.wikipedia.org/w/index.php?title=%D0%A1%D0%B5%D1%82%D1%8C_%D0%B4%D0%BE%D0%B2%D0%B5%D1%80%D0%B8%D1%8F&action=edit&redlink=1)), получившая наибольшее распространение в сетях [PGP](https://ru.wikipedia.org/wiki/PGP).

**6. Электронная цифровая подпись (ЭЦП).**

Электро́нная по́дпись (ЭП) — [реквизит](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D0%B2%D0%B8%D0%B7%D0%B8%D1%82) [электронного документа](http://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B4%D0%BE%D0%BA%D1%83%D0%BC%D0%B5%D0%BD%D1%82), позволяющий установить отсутствие искажения информации в электронном документе с момента формирования ЭП и проверить принадлежность подписи владельцу [сертификата ключа ЭП](http://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%82_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BB%D1%8E%D1%87%D0%B0). Значение реквизита получается в результате криптографического преобразования информации с использованием *закрытого ключа ЭП*.

## Назначение и применение ЭЦП

Цифровая подпись предназначена для [аутентификации](http://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) лица, подписавшего электронный документ. Кроме этого, использование цифровой подписи позволяет осуществить:

* **Контроль** [**целостности**](http://ru.wikipedia.org/wiki/%D0%A6%D0%B5%D0%BB%D0%BE%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%B8) передаваемого документа: при любом случайном или преднамеренном изменении документа подпись станет недействительной, потому что вычислена она на основании исходного состояния документа и соответствует лишь ему.
* **Защиту от изменений (подделки) документа**: гарантия выявления подделки при контроле целостности делает подделывание нецелесообразным в большинстве случаев.
* **Невозможность отказа от авторства**. Так как создать корректную подпись можно, лишь зная закрытый ключ, а он должен быть известен только владельцу, то владелец не может отказаться от своей подписи под документом.
* **Доказательное подтверждение авторства документа**: Так как создать корректную подпись можно, лишь зная закрытый ключ, а он должен быть известен только владельцу, то владелец пары ключей может доказать своё авторство подписи под документом.

Все эти свойства ЭЦП позволяют использовать её для следующих целей:

## Алгоритмы

Существует несколько схем построения **цифровой подписи**:

* На основе алгоритмов [симметричного шифрования](http://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Данная схема предусматривает наличие в системе третьего лица — арбитра, пользующегося доверием обеих сторон. Авторизацией документа является сам факт зашифрования его секретным ключом и передача его арбитру.
* На основе алгоритмов [асимметричного шифрования](http://ru.wikipedia.org/wiki/%D0%90%D1%81%D0%B8%D0%BC%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D1%88%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). На данный момент такие схемы **ЭЦП** наиболее распространены и находят широкое применение..

## Использование хеш-функций

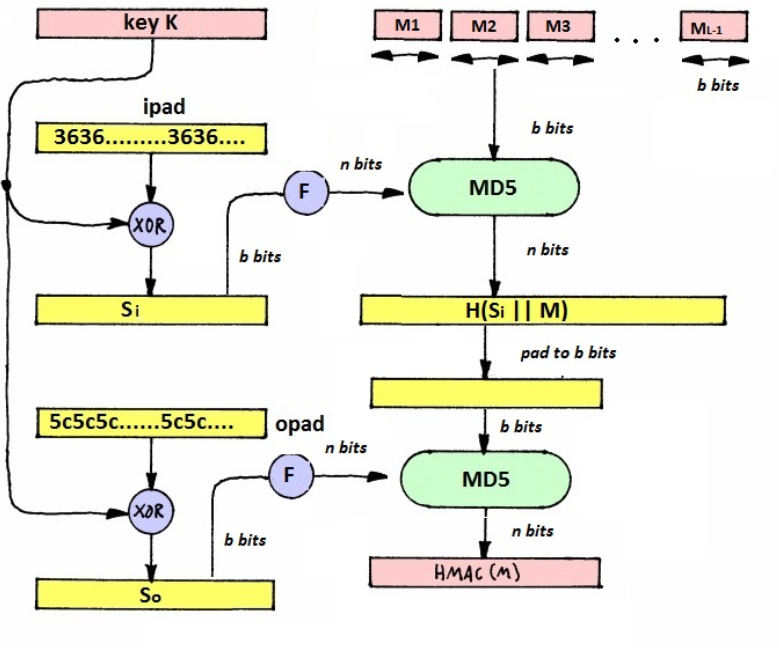
Поскольку подписываемые документы — переменного (и как правило достаточно большого) объёма, в схемах **ЭЦП** зачастую подпись ставится не на сам документ, а на его [хеш](http://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88). Для вычисления хэша используются криптографические хеш-функции, что гарантирует выявление изменений документа при проверке подписи. Хеш-функции не являются частью алгоритма **ЭЦП**, поэтому в схеме может быть использована любая надёжная хеш-функция.

Использование хеш-функций даёт следующие преимущества:

* **Вычислительная сложность**. Обычно хеш цифрового документа делается во много раз меньшего объёма, чем объём исходного документа, и алгоритмы вычисления хеша являются более быстрыми, чем алгоритмы **ЭЦП**.
* **Совместимость**. Большинство алгоритмов оперирует со строками бит данных, но некоторые используют другие представления. Хеш-функцию можно использовать для преобразования произвольного входного текста в подходящий формат.
* **Целостность**. Без использования хеш-функции большой электронный документ в некоторых схемах нужно разделять на достаточно малые блоки для применения **ЭЦП**. При верификации невозможно определить, все ли блоки получены и в правильном ли они порядке.

**7. Методы аутентификации сообщений. Алгоритм HMAC.**

**HMAC** ([сокращение](https://ru.wikipedia.org/wiki/%D0%90%D0%B1%D0%B1%D1%80%D0%B5%D0%B2%D0%B8%D0%B0%D1%82%D1%83%D1%80%D0%B0) от [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) ***h****ash-based* ***m****essage* ***a****uthentication* ***c****ode*, код [аутентификации](https://ru.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D0%B5%D0%BD%D1%82%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F) (проверки подлинности) сообщений, использующий [хеш](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%81%D1%83%D0%BC%D0%BC%D0%B0)-[функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5))) — в [информатике](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) ([криптографии](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)), один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними лицами (см. [человек посередине](https://ru.wikipedia.org/wiki/%D0%A7%D0%B5%D0%BB%D0%BE%D0%B2%D0%B5%D0%BA_%D0%BF%D0%BE%D1%81%D0%B5%D1%80%D0%B5%D0%B4%D0%B8%D0%BD%D0%B5)). Механизм HMAC использует [MAC](https://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%B8%D1%82%D0%BE%D0%B2%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0), описан в [RFC 2104](https://tools.ietf.org/html/rfc2104), в стандартах организаций [ANSI](https://ru.wikipedia.org/wiki/ANSI), [IETF](https://ru.wikipedia.org/wiki/IETF), [ISO](https://ru.wikipedia.org/wiki/ISO) и [NIST](https://ru.wikipedia.org/wiki/NIST). **MAC** — стандарт, описывающий способ обмена данными и способ проверки целостности передаваемых данных с использованием секретного ключа. Два клиента, использующие MAC, как правило, разделяют общий секретный ключ. HMAC — надстройка над MAC; механизм обмена данными с использованием секретного ключа (как в MAC) и [хеш-функций](https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F). В зависимости от используемой хеш-функции выделяют[[1]](https://ru.wikipedia.org/wiki/HMAC#cite_note-rfc2104-1) HMAC-[MD5](https://ru.wikipedia.org/wiki/MD5), HMAC-[SHA1](https://ru.wikipedia.org/wiki/SHA-1), HMAC-[RIPEMD128](https://ru.wikipedia.org/wiki/RIPEMD-128), HMAC-[RIPEMD160](https://ru.wikipedia.org/wiki/RIPEMD-160) и т. п.



**8. Методы идентификации и аутентификации пользователей. Парольная аутентификация (многоразовые пароли). Критерии выбора пароля.**

Для получения доступа к ресурсам, пользователь должен пройти процесс представления компьютерной системе, который включает две стадии:

***идентификацию*** - пользователь сообщает системе по ее запросу свое имя (идентификатор);

***аутентификацию*** - пользователь подтверждает идентификацию, вводя в систему уникальную, не известную другим пользователям информацию о себе (например, пароль).

Для проведения процедур идентификации и аутентификации пользователя необходимо наличие:

программы аутентификации, уникальной информации о пользователе.

Различают две формы хранения информации о пользователе:

- внешняя (например, пластиковая карта или голова пользователя)

- внутренняя (например, запись в базе данных)

**Методы идентификации:**

- использование механического ключа;

- организация пропускной системы обычного типа (пропуск с фотографией: охранник на входе);

- при помощи кодового замка с набором кода на клавиатуре;

- посредством различного вида карт (электронных, механических, магнитных), на которые нанесен код, определяемый специальным устройством (считывателем);

- при помощи особых устройств, генерирующих модулированный ультразвуковой, инфракрасный или радиосигнал;

- при помощи технических устройств (биометрические устройства), определяющих физиологические параметры человека, такие как голос, отпечатки пальцев и пр.;

- комбинированные методы.

**Методы аутентификации:**

**-** аутентификация по отпечаткам пальцев

- аутентификация по радужной оболочке глаза.

- аутентификация по сетчатке глаза.

- парольная аутентификация

- биометрия

- многофакторная аутентификация

- интеграция с системами управления доступом

- встроенная флэш-память

- генераторы одноразовых паролей

- Java-токены

**Парольная аутентификация**

Наиболее распространенными простыми и привычными являются методы аутентификации, основанные на паролях – секретных идентификаторах субъектов. Здесь при вводе субъектом своего пароля подсистема аутентификации сравнивает его с паролем, хранящимся в базе эталонных данных в зашифрованном виде. В случае совпадения паролей подсистема аутентификации разрешает доступ к ресурсам АС.

Парольные методы следует классифицировать по степени изменяемости паролей:

-        методы, использующие постоянные (многократно используемые) пароли,

-        методы, использующие одноразовые (динамично изменяющиеся) пароли.

В большинстве АС используются многоразовые пароли. В этом случае пароль пользователя не изменяется от сеанса к сеансу в течение установленного администратором системы времени его действительности. Это упрощает процедуры администрирования, но повышает угрозу рассекречивания пароля. Известно множество способов вскрытия пароля: от подсмотра через плечо до перехвата сеанса связи. Вероятность вскрытия злоумышленником пароля повышается, если пароль несет смысловую нагрузку (год рождения, имя девушки), небольшой длины, набран на одном регистре, не имеет ограничений на период существования и т. д. Важно, разрешено ли вводить пароль только в диалоговом режиме или есть возможность обращаться из программы.

В последнем случае, возможно запустить программу по подбору паролей – «дробилку».

Более надежный способ – использование одноразовых или динамически меняющихся паролей.

Известны следующие методы парольной защиты, основанные на одноразовых паролях:

-        методы модификации схемы простых паролей;

-        методы «запрос-ответ»;

-        функциональные методы.

В первом случае пользователю выдается список паролей. При аутентификации система запрашивает у пользователя пароль, номер в списке которого определен по случайному закону. Длина и порядковый номер начального символа пароля тоже могут задаваться случайным образом.

При использовании метода «запрос-ответ» система задает пользователю некоторые вопросы общего характера, правильные ответы на которые известны только конкретному пользователю.

Функциональные методы основаны на использовании специальной функции парольного преобразования http://www.sernam.ru/htm/book_ss/files/ss_23.files/image002.gif. Это позволяет обеспечить возможность изменения (по некоторой формуле) паролей пользователя во времени. Указанная функция должна удовлетворять следующим требованиям:

-        для заданного пароля x легко вычислить новый пароль http://www.sernam.ru/htm/book_ss/files/ss_23.files/image003.gif;

-        зная  х  и  y, сложно или невозможно определить функцию http://www.sernam.ru/htm/book_ss/files/ss_23.files/image002.gif.

Наиболее известными примерами функциональных методов являются: метод функционального преобразования и метод «рукопожатия».

Идея метода функционального преобразования состоит в периодическом изменении самой функции http://www.sernam.ru/htm/book_ss/files/ss_23.files/image002.gif. Последнее достигается наличием в функциональном выражении динамически меняющихся параметров, например, функции от некоторой даты и времени. Пользователю сообщается исходный пароль, собственно функция и периодичность смены пароля. Нетрудно видеть, что паролями пользователя на заданных http://www.sernam.ru/htm/book_ss/files/ss_23.files/image004.gif-периодах времени будут следующие: x, f(x),  f(f(x)),  ...,  f(x)n-1.

Метод «рукопожатия» состоит в следующем. Функция парольного преобразования известна только пользователю и системе защиты. При входе в АС подсистема аутентификации генерирует случайную последовательность x, которая передается пользователю. Пользователь вычисляет результат функции y=f(x) и возвращает его в систему. Система сравнивает собственный вычисленный результат с полученным от пользователя. При совпадении указанных результатов подлинность пользователя считается доказанной.

Достоинством метода является то, что передача какой-либо информации, которой может воспользоваться злоумышленник, здесь сведена к минимуму.

В ряде случаев пользователю может оказаться необходимым проверить подлинность другого удаленного пользователя или некоторой АС, к которой он собирается осуществить доступ. Наиболее подходящим здесь является метод «рукопожатия», так как никто из участников информационного обмена не получит никакой конфиденциальной информации.

Отметим, что методы аутентификации, основанные на одноразовых паролях, также не обеспечивают абсолютной защиты. Например, если злоумышленник имеет возможность подключения к сети и перехватывать передаваемые пакеты, то он может посылать последние как собственные

**9. Методы идентификации и аутентификации пользователей Парольная аутентификация (одноразовые пароли). Алгоритм генерирования одноразовых паролей S/KEY.**

На протяжении последних десятилетий наблюдается все более глубокое проникновение самых разнообразных аспектов человеческой деятельности во всемирную сеть, что позволяет нивелировать ограничения, налагаемые расстоянием, разделяющим субъекты, участвующие во взаимодействии. Однако вместе с тем все острее встает вопрос идентификации и аутентификации удаленного пользователя.

Существует ряд подходов к обеспечению идентификации и аутентификации личности, однако, в силу своей простоты и минимальных требований к оборудованию, наибольшее распространение получили методы, основанные на использование секретных паролей. При этом возникает вопрос как обеспечить безопасность передачи пароля по сети? Ведь даже в случае передачи пароля в зашифрованном виде или в виде хэш-значения не исключается возможность перехвата и последующего повторного использования этого значения (атака воспроизведения). Одним из решений данной проблемы является использование одноразовых ключей (аутентификация каждый раз производится с помощью нового значения ключа, т.о. даже если злоумышленнику удастся перехватить передаваемые по сети данные и извлечь из них значение ключа, используемого для аутентификации пользователя, это не даст ему возможности произвести аутентификацию от его имени, т.к. при следующей аутентификации сервер будет ожидать другое ключевое значение. При этом генерация всех одноразовых ключей производится на основе одного секретного пароля пользователя).

Одним из наиболее распространенных протоколов аутентификации на основе одноразовых паролей является стандартизованный в Интернете протокол S/Key. Алгоритм выглядит следующим образом:

1. При регистрации пользователя производится хэширование его пароля N раз с помощью какого-либо алгоритма хэширования (спецификация протокола S/Key определяет в качестве алгоритма хэширования алгоритм MD4, однако в общем случае может быть применен любой алгоритм хэширования). Результат N-кратного хэширования пароля сохраняется на стороне сервера вместе со значением N.
2. При прохождении пользователем процедуры аутентификации, сервер возвращает клиентскому приложению значение N-1. На стороне клиента производится хэширование пароля пользователя N-1 раз и результат передается серверному приложению.
3. Сервер производит однократное хэширование полученного от клиента значения и сравнивает его результат с ранее сохраненным значением. В случае если значения совпали, то принимается решение о успешной аутентификации пользователя и значения хранимого пароля и N заменяются на полученное от пользователя по сети хэш-значение и (N-1) соответственно, иначе принимается решение об отказе в аутентификации.

Т.о. очевидно, что даже в случае если злоумышленнику удастся перехватить передаваемое по сети значение одноразового пароля, то, в силу необратимости хэш-преобразований, ему не удастся ни вычислить значение следующего одноразового ключа, ни восстановить значение исходного пароля пользователя.

**10. Протокол аутентификации Kerberos.**

Kerberos - сетевой протокол аутентификации, позволяющий передавать данные через незащищённые сети для безопасной идентификации. Он ориентирован в первую очередь на клиент-серверную модель и обеспечивает взаимную аутентификацию — оба пользователя через сервер подтверждают личности друг друга. Данная модель является одним из вариантов Нидхем-Шрёдер-протокола аутентификации на основе доверенной третьей стороны и его модификациях, предложенных Denning и Sacco.

**Описание:**

**Вход пользователя в систему:**

1. Пользователь вводит имя и пароль на клиентской машине.
2. Клиентская машина выполняет над паролем одностороннюю функцию (обычно хэш), и результат становится секретным ключом клиента/пользователя.

**Аутентификация клиента:**

1. Клиент отсылает запрос (AS\_REQ) на СА для получения аутентификационных верительных данных и последующего их предоставления TGS серверу (впоследствии он будет их использовать для получения билетов без дополнительных запросов на применение секретного ключа пользователя.) Данный запрос содержит:

*Идентификатор клиента, его метка времени и идентификатор сервера.*

1. Если политика ЦРК требует предварительной аутентификации, то пользователь получает сообщение KRB\_ERROR, в ответ на которое посылает повторный запрос, но с уже данными для установления подлинности.
2. СА проверяет, есть ли такой клиент в базе. Если есть, то назад СА отправляет сообщение (AS\_REP) включающее:

* Сессионный ключ клиент/TGS, идентификатор TGS и время жизни билета зашифрованные секретным ключом клиента.
* TGT (который включает идентификатор и сетевой адрес клиента, метку времени ЦРК, период действия билета и сессионный ключ Клиент/TGS), зашифрованный секретным ключом TGS.

Если же нет, то клиент получает новое сообщение, говорящее о произошедшей ошибке.

Получив сообщение, клиент расшифровывает свою часть для получения Сессионного Ключа Клиент/TGS. Этот сессионный ключ используется для дальнейшего обмена с сервером TGS. (Важно: Клиент не может расшифровать TGT, так как оно зашифровано секретным ключом TGS) В этот момент у пользователя достаточно данных, чтобы авторизоваться на TGS.

**Авторизация клиента на TGS:**

1. Для запроса сервиса клиент формирует запрос на TGS (TGS\_REQ) содержащий следующие данные:

* TGT, полученный ранее и идентификатор сервиса.
* Аутентификатор (составленный из ID клиента и временного штампа), зашифрованный на Сессионном Ключе Клиент/TGS.

1. После получения TGS\_REQ, TGS извлекает из него TGT и расшифровывает его используя секретный ключ TGS. Это дает ему Сессионный Ключ Клиент/TGS. Им он расшифровывает аунтетификатор. Затем он генерирует сессионный ключ клиент/сервис и посылает ответ (TGS\_REP) включающий:

* Билет сервиса (который содержит ID клиента, сетевой адрес клиента, метку времени ЦРК, время действия билета и Сессионный Ключ клиент/сервис) зашифрованный секретным ключом сервиса.
* Сессионный ключ клиент/сервис, идентификатор сервиса и время жизни билета, зашифрованные на Сессионном Ключе Client/TGS.

**Запрос сервиса клиентом:**

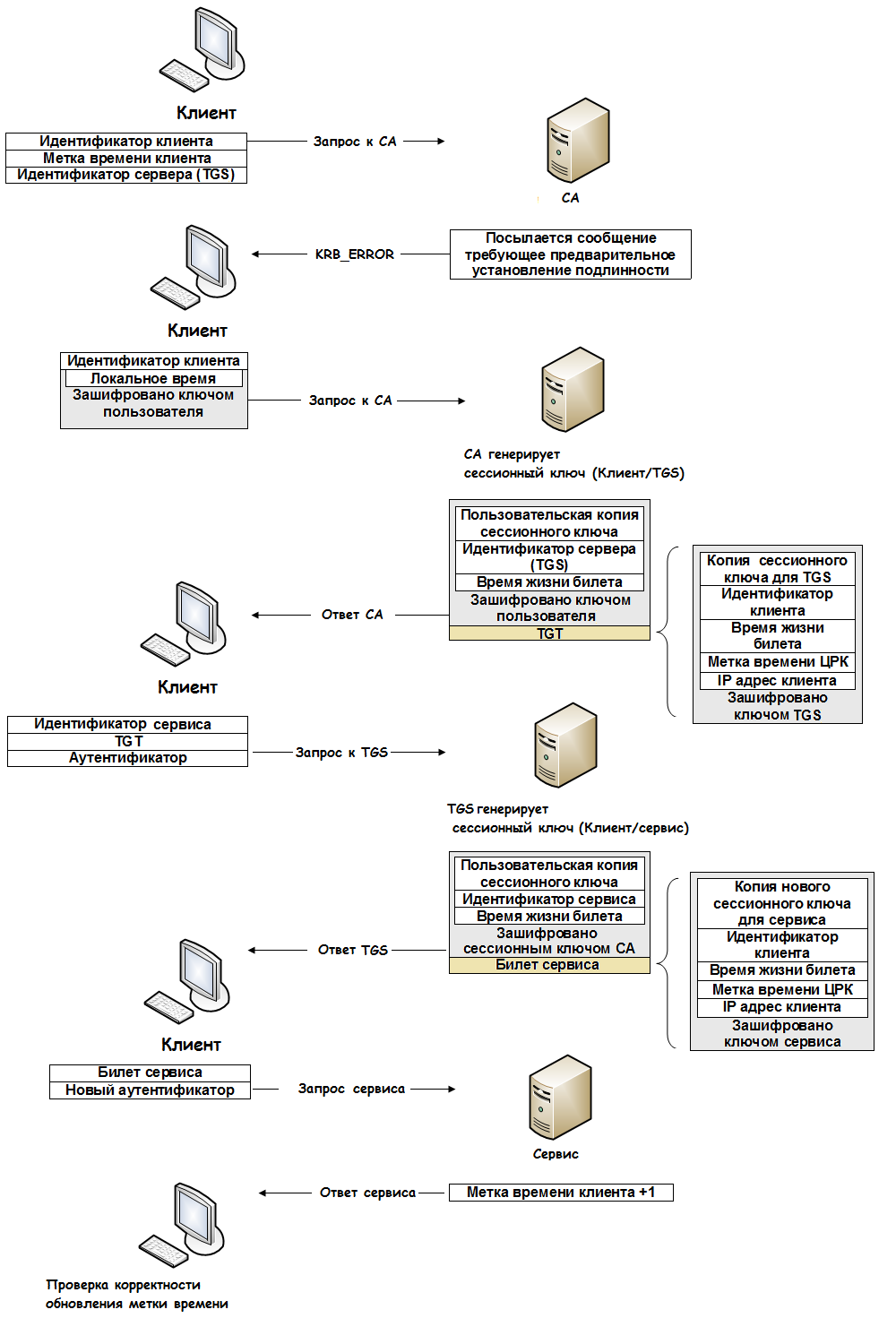
1. После получения TGS\_REP, у клиента достаточно информации для авторизации на сервисе. Клиент соединяется с ним и посылает сообщение содержащее:

* Зашифрованный билет сервиса полученный ранее.
* Новый аутентификатор, зашифрованный на сессионном ключе клиент/сервис, и включающий ID клиента и метку времени.

1. Сервис расшифровывает билет используя свой секретный ключ и получает сессионный ключ клиент/сервис. Используя новый ключ, он расшифровывает аутентификатор и посылает клиенту следующее сообщение для подтверждения готовности обслужить клиента и показать, что сервер действительно является тем, за кого себя выдает:

* Метку времени, указанную клиентом + 1, зашифрованную на сессионном ключе клиент/сервис.

1. Клиент расшифровывает подтверждение, используя сессионный ключ клиент/сервис и проверяет, действительно ли метка времени корректно обновлена. Если это так, то клиент может доверять серверу и может начать посылать запросы на сервер.
2. Сервер предоставляет клиенту требуемый сервис.



**11. Биометрическая идентификация и аутентификация пользователей.**

Биометрическое распознавание объекта основано на сравнении ***физиологических*** или ***психологических*** особенностей этого объекта с его характеристиками, хранящимися в базе данных системы.

Биометрические технологии можно разделить на две большие категории - ***физиологические*** и ***психологические (поведенческие)***.

В первом случае анализируются такие признаки, как черты лица, структура глаза, параметры пальцев, ладонь, форма руки.

Психологические характеристики - это голос человека, особенности его подписи, динамические параметры письма и особенности ввода текста с клавиатуры.

Логически биометрическую систему можно разделить на два модуля: ***модуль регистрации*** и ***модуль идентификации***.

***На этапе регистрации*** биометрические датчики сканируют необходимые физиологические или поведенческие характеристики человека и создают их цифровое представление. Специальный модуль обрабатывает это представление с тем, чтобы выделить характерные особенности и сгенерировать шаблон.

***На этапе идентификации*** биометрический датчик снимает характеристики человека, которого нужно идентифицировать, и преобразует эти характеристики в тот же цифровой формат, в котором хранится шаблон. Полученный шаблон сравнивается с хранимым, чтобы определить, соответствуют ли эти шаблоны друг другу.

Методы биометрической аутентификации делятся на  две группы:

***Статические методы -*** основываются на физиологической (статической) характеристике человека, то есть уникальной характеристике, данной ему от рождения и неотъемлимой от него:

* ***По отпечатку пальца.*** В основе этого метода лежит уникальность рисунка паппилярных узоров на пальцах.;
* ***По форме ладони.*** С помощью специального устройства строится трехмерный образ кисти;
* ***По форме лица.*** На лице выделяются контуры бровей, глаз, носа, губ и т.д., вычисляется расстояние между ними и строится не просто образ, а еще множество его вариантов на случаи поворота лица, наклона, изменения выражения;
* ***По сетчатке глаза***
* ***По радужной оболочке глаза.***
* ***По ДНК.***

***Динамические методы -*** основываются на поведенческой (динамической) характеристике человека, то есть построены на особенностях, характерных для подсознательных движений в процессе воспроизведения какого-либо действия.

* ***По рукописному почерку.*** Как привило, для этого вида идентификации человека используется его роспись.
* ***По клавиатурному почерку.*** Основная характеристика, по которой строится свертка для идентификации – динамика набора кодового слова;
* ***По голосу*** Основа - различные сочетания частотных и статистических  характеристик голоса;

Сравнивать описанные выше биометрические методы по показаниям ошибок первого рода (не пустить в систему «своего») очень сложно, так как они сильно разнятся из-за сильной зависимости от оборудования на котором они реализованы.

По показателям ошибок второго рода (пустить в систему чужого) общая сортировка методов биометрической аутентификации выглядит так (от лучших к худшим):

* ДНК;
* Радужная оболочка глаза, сетчатка глаза;
* Отпечаток пальца, термография лица, форма ладони;
* Форма лица, расположение вен на кисти руки и ладони;
* Подпись;
* Клавиатурный почерк;
* Голос.

**12. Методы управления доступом. Модель дискреционного контроля доступа (DAC).**

Управление доступом - это определение возможности субъекта оперировать над объектом.

Модель управления доступом – это структура, которая определяет порядок доступа субъектов к объектам. Для реализации правил и целей модели используются технологии управления доступом и механизмы безопасности. Существует три основных модели управления доступом: дискреционная, мандатная и недискреционная (ролевой).

Эти модели встроены в ядро различных операционных систем и во многих случаях поддерживаются приложениями. Каждая операционная система имеет ядро безопасности, которое реализует концепцию монитора обращений ([reference monitor](http://en.wikipedia.org/wiki/Reference_monitor)), которая зависит от встроенной в систему модели управления доступом. Для каждой попытки доступа, перед тем, как субъект сможет начать взаимодействовать с объектом, ядро безопасности проверяет правила модели управления доступом, чтобы определить, является ли запрос допустимым.

Если пользователь создает файл, он является владельцем этого файла. Идентификатор этого пользователя размещается в заголовке файла. Владение может быть также предоставлено определенному человеку. Дискреционное (произвольное) управление доступом (DAC) основывается на понятиях владельца объекта и списка доступа к объекту. Для получения соответствующих прав действий над объектом пользователь должен быть указан в списке доступа (ALC) к этому объекту либо персонально, либо в составе группы пользователей, для которой разрешены соответствующие действия над объектом. При отсутствии пользователя или его группы в списке при доступе к объекту используются права, применяемые по умолчанию.

Содержимое списков контроля доступа (ALC) определено владельцами. Работа ACL реализуется средствами операционной системы. Это может позволить пользователям использовать информацию динамически вместо более статичного мандатного или ролевого управления доступом. Большинство операционных систем основаны на модели DAC (например, системы Windows, Linux, Macintosh и большинство систем \*nix).

Однако DAC имеет существенный недостаток. Он заключается в том, что он не предоставляет полной гарантии того, что информация не станет доступна субъектам не имеющим к ней доступа. Это проявляется в том, что субъект имеющий право на чтение информации может передать ее другим субъектам, которые этого права не имеют, без уведомления владельца объекта. Система DAC не устанавливает никаких ограничений на распространение информации после того как субъект ее получил.

Еще одной особенностью DAC, которую можно отнести к недостаткам, является то, что все объекты в системе принадлежат субъектам, которые настраивают доступ к ним для других. На практике оказывается, что в большинстве случаев данные в системе не принадлежат отдельным субъектам, а всей системе. Наиболее распространенным примером такой системы является информационная система.

Классическая система дискреционного контроля доступа называется "закрытой" в том смысле, что изначально объект не доступен никому, и в списке прав доступа описывается список разрешений. Также существуют "открытые" системы, в которых по умолчанию все имеют полный доступ к объектам, а в списке доступа описывается список ограничений.

13. Методы управления доступом. Модель обязательного контроля доступа (MAC).

**Мандатное управление доступом** (Mandatory access control, MAC), также иногда переводится как **принудительный контроль доступа**. Суть мандатного принципа контроля доступа состоит в сопоставлении каждому объекту системы меток конфиденциальности (уровней секретности), и выдаче официальных разрешений (допуска) субъектам (пользователям) на обращение к информации с таким уровнем конфиденциальности. Модели построенные на мандатном управлении доступом:

**Модель Белла-Лападула.**

Правило 1: (No Read Up - Нет чтения вверх) Субъект A может читать объект О только в том случае, если уровень секретности А >= уровню секретности О. Субъект с уровнем доступа “Секретно” не может прочитать объект с уровнем доступа “Совершенно секретно”.

Правило 2: (No Write Down - Нет записи вниз) Субъект A может писать информацию в объект О только в том случае, если уровень секретности А <= уровню секретности О. Это правило разрешает проблему троянских коней, запрещая характерную для них запись информации на более низкий уровень. Таким образом субъект с уровнем доступа “Совершенно секретно” не сможет записать в “Неклассифицированный ” объект.

Данная модель разрабатывалась для военного ведомства США. Она заботиться о сохранении секретности, а не целостности. Секретность (конфиденциальность) - гарантия того, что конкретная информация доступна только тому кругу лиц, для кого она предназначена.

**Модель Кена Биба.**

Инверсия модели Белла-Лападула.

Правило 1: (No Read Down - Нет чтения вниз) Субъект A может читать объект О только в том случае, если уровень целостности А <= уровню целостности О. На более низких уровнях целостности может находиться некорректная или не актуальная информация.

Правило 2: (No Write Up - Нет записи вверх) Субъект A может писать информацию в объект О только в том случае, если уровень целостности А >= уровню целостности О.

Модель заботиться о сохранении целостности, а не секретности. Она разрабатывалась для нужд бизнеса и промышленности. Целостность - гарантия того, что информация сейчас существует в ее исходном виде, при ее хранении или передаче не было произведено несанкционированных изменений и она является актуальной, корректной.

**MLS (Multi-Level Security)** Multi-Level Security(MLS) - многоуровневая модель безопасности. Многоуровневая модель безопасности строится обычно на основе модели Белла-Лападула. Объекты в ней разделяются по уровням классификации (classification level), а субъекты причисляются к одному из уровней благонадежности (clearanсe level). Уровни классификации и благонадежности объединяют понятием уровни безопасности (security level).

В классической модели выделяют 4 уровня безопасности:

* Не классифицировано
* Конфиденциально
* Секретно
* Совершенно секретно

**Достоинства MAC.** 1)Динамичность. (гибкость в настройке прав).2)Более высокая степень надежности (централизация хранения меток; пользователь не может полностью управлять доступом к ресурсам, которые он создаёт и передать доступ неавторизованным пользователям; решена проблема троянских программ)

**Недостатки MAC.**1) Сложность реализации. 2) Значительные затраты ресурсов вычислительной системы.

14. Методы управления доступом. Модель ролевого контроля доступа (RBAC).

Модель [***ролевого управления доступом***](http://ru.wikipedia.org/wiki/%D0%A3%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B4%D0%BE%D1%81%D1%82%D1%83%D0%BF%D0%BE%D0%BC_%D0%BD%D0%B0_%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B5_%D1%80%D0%BE%D0%BB%D0%B5%D0%B9) (RBAC – Role-based Access Control), также называемая недискреционным управлением доступом (Nondiscretionary Access Control), использует централизованно администрируемый набор контролей, предназначенных для определения порядка взаимодействия субъекта с объектом. Этот тип модели разрешает доступ к ресурсам, основываясь на роли пользователя в компании. Это называют недискреционным подходом, поскольку назначение пользователю роли является неизбежным. Это означает, что если вам в компании назначена роль «Подрядчик», вы ничего с этим сделать не можете. Вы не определяете самостоятельно, какая роль вам будет назначена.  
 Более традиционное администрирование прав доступа основано на модели DAC, в которой управление доступом происходит на уровне объекта с использованием ACL. Этот подход более сложен, т.к. администратор должен перевести организационную политику компании в разрешения при настройке ACL. С ростом количества объектов и пользователей в компании у многих пользователей появляются (или остаются после изменения обязанностей) права доступа к некоторым ресурсам, которые им не требуются для работы. Это нарушает принцип минимальных привилегий и увеличивает риски компании. Подход RBAC позволяет избежать этого, так как разрешения управляются на уровне должностных ролей. В модели RBAC роль определяется в терминах операций и задач, которые она выполняет, тогда как в модели DAC описывается, какие субъекты могут иметь доступ к каким объектам.  
 Например, нам нужна роль аналитика. Мы разрабатываем эту роль, позволяя ей иметь доступ ко всем видам продукции и данным тестирования, а также, что более важно, определяем задачи и операции, которые эта роль может выполнять с этими данными. Когда пользователь, которому присвоена эта роль «Аналитик», делает запрос на доступ к новым результатам тестирования на файловом сервере, операционная система в фоновом режиме просматривает уровень доступа роли, прежде чем эта операция будет разрешена.

**ПРИМЕЧАНИЕ.** Введение ролей показывает разницу между правами, назначенными явно и неявно. В явном виде права и разрешения назначаются непосредственно конкретному пользователю. В неявном виде они назначаются роли или группе, а пользователь просто наследует эти полномочия.

Модель RBAC лучше всего подходит для компаний с большой «текучестью» кадров. Если увольняется сотрудник, которому была назначена определенная роль, то новому сотруднику, который займет его место, просто назначается та же роль. Таким образом, администратору не нужно постоянно вносить изменения в ACL отдельных объектов. Ему достаточно создать определенные роли, настроить необходимые этим ролям права и разрешения, а затем назначить пользователям эти роли.

**15. Методы защиты программного обеспечения от несанкционированного использования.**

Программное обеспечение – интеллектуальная собственность её автора, группы разработчиков либо компании, поставляющей продукт на рынок.

Существует ряд угроз, отсутствие защиты от которых может привести к серьёзным экономическим потерям.

**Средства защиты прав производителей программный продуктов**

1. Юридические средства
2. Этические средства
3. Технические средства.

**Средства защиты от несанкционированного использования:**

1. **Аппаратные средства** - Использование аппаратных ключей (“hardware dongles”) Интерфейс, которые крайне сложно скопировать или получить доступ и их содержимому. Во время выполнения программа периодически инициирует обмен данными с таким ключом, тем самым, проверяя его наличие как подтверждение факта своей легальности

**Недостатки**

* Возможность физической потери.
* Использование эмуляторов .
* Определение и отключение обращения к физическому ключу.
* Невозможность использования данного метода защиты при распространении ПО через Интернет.

1. **Программные методы**

* Регистрация копии программного продукта - Каждый серийный номер может обладать определённым
* Привязка программного продукта к носителю информации
* Аутентификация копии программного продукта через интернет
* Выполнение программного продукта на сервере

**Использование регистрации ПО**

Использование для каждой программы уникальных регистрационных номеров:

* Осуществление проверки программой наличия серийного номера, владение которым должно подтверждать легальность владения данной копией программы.
* Каждый серийный номер может обладать определённым сроком действия, продление которого, как правило,осуществляется через интернет на сайте разработчика после перевода соответствующей суммы за право использования продуктом, в результате чего генерируется новый номер.

**Привязка программного продукта к носителю информации**

* В процессе установки программа копируется на ПК лишь частично, оставляя большую часть важных для работы данных на носителе (дискета, компакт диск), на котором она распространяется.
* Программа всегда запускается с носителя информации.
* Механизм защиты реализуется за счёт невозможности скопировать программу на другой носитель.
* Применяется нестандартное форматирование дисков, преднамеренное повреждение рабочей поверхности на этапе изготовления. Примером такого типа защиты является система LaserLock, StarForce.

Защищаемый носитель содержит два раздела:

Первый раздел отформатирован стандартным образом и содержит программу-загрузчик;

Второй раздел отформатирован нестандартно и содержит саму защищаемую программу.

Когда загрузчик выполняется, он считывает второй раздел, копируя его в память, и после этого передаёт ему управление.

Средствами операционной системы не возможно прочитать второй раздел, программа является защищённой от копирования.

**Аутентификация копии программного продукта через интернет**

* Приложение при запуске или периодически во время своей работы посылает на сервер разработчика информацию о пользователе, получая в ответ подтверждение того, что данная копия используется легально, и в зависимости от ответа продолжает или прекращает работу.
* Необходимо иметь постоянную связь с сервером.

**Выполнение программного продукта на сервере**

* Пользователь получает только небольшую клиентскую часть программного продукта.
* Все основные вычисления производятся на сервере разработчика, а пользовательская часть программы осуществляет только ввод и пересылку исходных данных, а также приём и отображение результатов.
* Правомерность пользования программой проверяется при аутентификации клиентской части на сервере.

**Недостатки**

Необходимость постоянной связи между клиентским и серверным хостами.

Низкая скорость выполнения в случае необходимости обмена большим количеством данных.

**16.Методы защиты программного обеспечения от обратного проектирования и модификации**

Обратное проектирование ПО (reverse engineering) − это целенаправленный процесс анализа кода ПО с целью извлечения из него определенных функциональных возможностей, а также раскрытия алгоритмов или данных, используемых в программе, которые могут представлять интерес для взломщика. Следствием успешного анализа может стать кража интеллектуальной собственности (если некоторые алгоритмы защищены патентами) или модификация программы, т. е. преднамеренное изменение исполняемого кода, приводящее к отклонению программы от нормального хода выполнения. Исследование логики работы может выполняться в статическом и динамическом режиме. Сущность статического режима заключается в изучении листинга исходного текста программы на языке Assembler, IL и т.д. Для его получения выполняемый программный модуль подвергают дизассемблированию. Динамический режим изучения алгоритма предполагает выполнение трассировки программы, т. е. запуск с использованием специальных средств, позволяющих выполнять программу в пошаговом режиме, получать доступ к регистрам процессора, областям памяти и т. д. Обычно трассировка производится в автоматическом режиме, после чего взломщик исследует ее результаты. Для защиты программ от изучения необходимо иметь средства противодействия как дизассемблированию, так и трассировке. Наиболее распространенными методами защиты от обратного проектирования и модификации являются: 1. Обфускация (obfuscation – затемнение) − изменение программного кода таким образом, чтобы он был труден для изучения и модификации третьими лицами, при сохранении его функциональности. Подходы: 1.1. Лексическая обфускация – удаление комментариев, отступов, замена имен идентификаторов, добавление мусорных операций и т.д. 1.2. Обфускация данных – объединение переменных (например 4хCHAR в INT), реструктурирование массивов (разделение на подмассивы или объединение), изменение иерархии наследования классов и т. д. 1.3. Обфускация управления – запутывание последовательности выполнения программного кода. Используются непрозрачные предикаты (последовательности операций с неизвестным результатом), добавление недостижимого кода, встраивание функций в место вызова и т.д. 1.4. Превентивная обфускация – использование недостатков наиболее распространенных деобфускаторов. 2. Вынесение критического программного кода в защищенный модуль. Таким модулем может быть внешний физический носитель (эффективно при небольшом количестве пользователей программы) или доверенный сервер в сети. 3. Проверка целостности программного кода. С этой целью используют некоторую одностороннюю хэш-функцию (например, SHA1), проверяющую часть кода, являющуюся критичной. В случае обнаружения изменений программа аварийно завершается через какой-то отрезок времени. 4. Создание защищенной среды выполнения, которая берет на себя функции по контролю доступа к программному коду, его шифрованию, отслеживанию подозрительного поведения программы и проч.

**17.Атаки на переполнение буфера. Переполнение буфера в стеке.** **Переполнение буфера в области динамической памяти.**

Переполнение буфера — явление, возникающее, когда компьютерная программа записывает данные за пределами выделенного в памяти буфера. Переполнение буфера обычно возникает из-за неправильной работы с данными, полученными извне, и памятью, при отсутствии жесткой защиты со стороны подсистемы программирования (компилятор или интерпретатор) и операционной системы. В результате переполнения могут быть испорчены данные, расположенные следом за буфером (или перед ним). Использование данной уязвимости подразумевает изменение хода выполнения привилегированной программы, например, запуск командной оболочки с правами администратора. Реализация атаки требует решения двух подзадач:

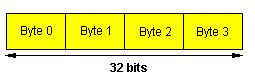
• Подготовка кода, который будет выполнятся в контексте привилегированной программы.

• Изменение последовательности выполнения программы с передачей управления подготовленному коду. Переполнение буфера в стеке: Данные добавляются в стек сверху и потом извлекаются. При атаке «переполнение буфера», направленной на стек, нарушитель добавляет в него больше данных, чем предусмотрено, при этом лишняя часть перезаписываются поверх данных, для которых разработчик программы не предусмотрел такой вариант. Так как вызову функции сопутствует занесение адреса возврата в стек, то при его подмене атакующим, управление передается по заданному им адресу. Рассмотрим методы защиты от подобных атак. Всем известно, что лучший способ защиты от любых атак подобного типа — безупречно написанные программы. Правильно написанные программы должны проверять длину входных данных, чтобы убедиться, что они не больше, чем выделенный буфер данных. Существует несколько схем для защиты от переполнения буфера. Простейшая схема предписывает компьютеру использовать стек и кучу только для данных и никогда не выполнять инструкции, обнаруженные в стеке и в куче. Другая популярная схема – это использовании осведомителя. Довольно таки хорошо с подобными проблемами справляются современные межсетевые экраны.

Буфера, расположенные в динамической памяти подвержены переполнению. В куче чаше всего встречаются переполняющиеся буфера двух типов: элементы структур и динамически выделяемые блоки памяти. пример структуры с переполняющимся буфером внутри strict demo { Int a; char buf[8]; int b; } Неосторожное обращение с обрабатываемыми данными (например, отсутствие нужных проверок в нужном месте) может привести к возможности переполнения буфера buf и как следствие – затиранию расположенных за ним переменных. В первую очередь это переменные-члены самой структуры (в данном случае – переменная b), стратегия модификация которых вполне типична и подчиняется тем же правилам – общим для всем переполняющихся буферов. Имеется возможность затирания ячеек памяти, лежащих за пределами выделенного блока памяти для буферов, монопольно владеющим всем выделенным блоком памяти. пример динамического блока памяти, подверженного переполнению #define MAX\_BUF\_SIZE 8 #define MAX\_STR\_SIZE 256 char \*p; … p = malloc(MAX\_BUF\_SIZE); … strncpy(p, MAX\_STR\_SIZE, str); Выделение и освобождение динамической памяти происходит довольно беспорядочно, и за концом нашего блока в произвольный момент времени может быть распложен любой другой блок. Даже при последовательном выделении нескольких блоков памяти, никто не может гарантировать, что при каждом запуске программы они будут выделяться в одном и том же порядке, поскольку это зависит от размера и порядка освобождения предыдущих выделяемых буферов. Структура служебных структур данных, пронизывающая динамическую память, предсказуема, хотя и меняется от одной версии библиотеки компилятора к другой. Существует множество реализаций динамической памяти и различные производители используют различные алгоритмы. Выделяемые блоки памяти могут быть нанизаны и на дерево, и на одно/двух связанный список, ссылки на который могут быть представлены как указателями, так и индексами, хранимыми либо в начале/конце каждого выделяемого блока, либо в отдельной структуре данных. Рассмотрим следующую организацию динамической памяти, при которой все выделяемые блоки соединены посредством двухсвязных списков, указатели на которых расположены в начале каждого блока (табл. ниже), причем смежные блоки памяти не обязательно должны находиться в соседних элементах списка, т. к. в процессе многократных операций выделения/освобождения список неизбежно фрагментируется, а постоянно дефрагментировать его себе дороже. указатель на следующий блок в цепочке указатель на предыдущий блок в цепочке размер блок памяти 1 статус (занят/свободен) память, выделенная блоку указатель на следующий блок в цепочке блок памяти 2 указатель на предыдущий блок в цепочке размер статус (занят/свободен) память, выделенная блоку … Табл - карта приблизительного распределения динамической памяти Переполнение буфера приводит к затиранию служебных структур следующего блока памяти и как следствие – возможности их модификации. Доступ к ячейкам всякого блока осуществляется по указателю, возращенному программе в момент его выделения, а отнюдь не по "служебному" указателю, который мы собираемся затирать! Служебные указатели используются исключительно функциями malloc/free (и другим подобными им функциям). Искажение указателя на следующий/предыдущий блок позволяет навязать адрес следующего выделяемого блока, например, "наложив" его на доступный нам буфер, но никаких гарантий, что это получится у нас нет – при выделении блока памяти, функция malloc ищет наиболее подходящий с ее точки зрения регион свободной памяти (обычно это первый свободный блок в цепочке, совпадающий по размеру с запрошенным), и не факт, что наш регион ей подойдет. Для уменьшения фрагментации динамической памяти функция free автоматически объединят текущий освобождаемый блок со следующим, если тот тоже свободен. Поскольку, смежные блоки могут находится на различных концах связывающего их списка, перед присоединением чужого блока памяти, функция free должна "выбросить" его из цепочки. Это осуществляется путем склейки предшествующего и последующий указателей, что в псевдокоде выглядит приблизительно так: \*указатель на след блок в цепочке = указатель на пред блок в цепочке. Обычно возможность записи в память используется для модификации таблицы импорта с целью подмены некоторой API-функции гарантированно вызываемой уязвимой программой вскоре после переполнения. Передать управление на переполняющийся буфер скорее всего не удастся, т. к. его адрес наперед неизвестен и тут приходится импровизировать: во-первых, злоумышленник может разместить shell-код в любом другом доступном ему буфере с известным адресом; во-вторых, среди функций уязвимой программы могут встретиться и такие функции(f), что передают управление на указатель, переданный им с тем или иным аргументом, после чего останется найти API-функцию, принимающую указатель на переполняющийся буфер и подменить ее адрес адресом функции f. В некоторых реализациях кучи можно встретиться не с указателями, а с индексами, в общем случае представляющие собой относительные адреса, отсчитываемые либо от первого байта кучи, либо от текущей ячейки памяти. Как говорилось выше, абсолютные адреса переполняющего буфера наперед неизвестны и непредсказуемым образом изменяются под воздействием ряда обстоятельств. Адреса же ячеек — абсолютны.

**18. Атаки на переполнение буфера. Целочисленное переполнение.Выполнение кода злоумышленника. Защита от атак типа переполнение буфера.**

В каждом языке программирования существуют типы данных. Основными являются char, string, pointer, float и integer. Integer это тип целого числа, например 2, 1024, 31337, и т.д. Согласно стандарту RFC1832, в 32 битных системах значения переменных данного типа могут лежать в пределах [-2147483648,2147483647]. Его структура выглядит следующим образом:



Как видно целое число занимает в памяти 32 бита, что ровно 4 байтам. Для нас Integer представляется в виде целого числа, но система работает с ним как с двоичным кодом. Где бит MSB устанавливается в единицу если число отрицательное. Если Integer используется как unsigned(только положительные значения), то значения попадут в пределы [0,4294967295].

Переполнение происходит при попытке записать в переменную значение превышающее максимально возможное число. Поведение программы в таких ситуациях полностью зависят от используемого компилятора. Потому, как согласно стандарту ISO C99 каждый компилятор при таком переполнении может делать все что угодно, от игнорирования до аварийного завершения программы. В большинства компиляторах какраз ничего и не делается :) Этот вид атаки опасен еще и тем, что приложение не может определить произошло переполнение или нет (вообще-то есть способ определить произошло переполнение или нет по регистрам...). Переполнения целых чисел можно использовать для влияния на значения некоторых критических данных, например размера буфера, индекса элемента массива, и т.д. Но на практике можно столкнутся с трудностью использования этой уязвимости. Все дело в том это переполнение в большинстве случяев не может непосредственно переписать область памяти(в отличии от Buffer Overflow и Heap overflow). Но применение этой уязвимости может легко вызвать и ошибки другого класса. В большинстве случаев возникает возможность переполнения буфера (Buffer Overflow). По словам ISO C99 уязвимость исчезает при использовании в вычислениях Unsigned Integer.

Пример:

unsigned int A=0xFFFFFFFF;

unsigned int B=0x1;

В результате выполнения операции (А+В) полученное значение, согласно стандарту ISO, не вмещается в 32 бита. В таком случае результатом будет (А+В) mod 0x100000000. mod - остаток от деления, например (8 mod 2 =0), а (8 mod 3=2). Следовательно, наш результат будет равен выражению:

result=(A+B) % 0x100000000;

подставив наши значения получим следующее:

result=(0xffffffff + 0x1) % 0x100000000;

//result=(0x100000000) % 0x100000000;

//result=0;

Переполнение буфера может привести к возможности запуска машинного кода злоумышленника от имени программы и с привилегиями учетной записи в которой выполняется программа.

Все атаки на переполнение буфера ставящие своей целью выполнение кода достигают цели через манипулирование значением счетчика команд процессора. Изменение счетчика команд злоумышленником возможно двумя способами – переполнение буфера на стеке и переполнение буфера в динамической памяти.

## Переполнение буфера на стеке.

Целью атаки на переполнения буфера на стеке является перезапись адреса возврата из функции в стековом кадре (stack frame) или перезапись указателя на функцию или обработчик исключения, вызываемых позже.

Переполнение буфера на стеке происходит при записи по адресу памяти стека программы за границами типов данных (чаще всего массив с фиксированной длинной). Из-за особенностей организации стека (стек «растет» вниз) подобные записи могут перезаписать данные стекового кадра.

|  |  |  |
| --- | --- | --- |
| Состояние стека до выполнения кода | Состояние стека после выполнение обычного кода | Состояние стека после переполнения буфера, приведшее к перезаписи адреса возврата из функции |
|  |  |  |

Если адрес перехода неизвестен, но известно что указатель на данные злоумышленника находится в одном из регистров, то используется техника «трамплина», когда адрес возврата заменяется на адрес машинной инструкции переходящей по адресу из регистра в котором хранится указатель на данные злоумышленника.

Проблему поиска точного расположения буфера возможно решить увеличением области перезаписываемых данных. Большие секции стека заполняются инструкциями no-op (инструкция пропуска такта) и в конце данных злоумышленника располагается инструкция относительного перехода к коду злоумышленника. Это позволяет злоумышленнику не вычислять местонахождение своего кода, а вычислить местонахождение любого участка заполненного инструкциями no-op, что гораздо легче. Распространенность данной техники привела к тому что многие системы обнаружения вторжения обнаруживают подобные участки кода.

## Переполнение буфера в динамической памяти.

Наиболее часто при атаке на переполнение буфера в динамической памяти для выполнения своего код, злоумышленник перезаписывает мета-данные алгоритмов выделения памяти, таких как системная функция malloc, что позже приводит к изменению счетчика команд процессора.

## Методы защиты от атаки на переполнение буфера.

* Предотвращение выполнения данных – техника современных ОС, предотвращающая выполнение кода из области памяти помеченной «только для чтения»
* Защита адресного пространства процесса
* Address Space Layout Randomization – техника меняющая расположение в адресном пространства важных структур программы (образа исполняемого файла, стека, кучи и т.д.)
* «Канарейки» – техника обнаружения переполнения буфера названная по аналогии с техникой обнаружения газа в шахтах. Между буфером и данными управления располагаются значения, за изменением которых наблюдают.
* Системы обнаружения вторжения
* Защита от повреждения стека (могут включать в себя несколько вышеперечисленных техник)
* Использование безопасных языков (наибольшее количество ошибок связанных с переполнением буфера находится в программах на языках C, C++)
* Использование безопасных библиотек

Стоит заметить что многие из вышеперечисленных способов защиты на работают против атак на переполнение буфера в динамической памяти.

Корректировка исходных кодов программы для устранения уязвимостей

Переполнение буфера происходит прежде всего из-за неправильного алгоритма работы программы, который не предусматривает проверок выхода за границы буферов. Также особую роль здесь играет язык программирования Си и его стандартные библиотеки. Так как Си не содержит средств контроля соответствия типов, то в переменную одного типа можно занести значение другого типа. Стандартные функции Си такие как strcpy, sprintf, gets работают со строками символов и не имеют в качестве аргументов их размеров, что легко приводит к переполнению буфера. Сложившийся годами стиль программирования, ориентированный на производительность программ, без выполнения дополнительных проверок также является причиной распространения данной уязвимости. В результате чего, для программистов выработано ряд методик и указаний по написанию программ, не содержащих уязвимости. Сформированы рекомендации по исправлению уже существующих программ (например, замена уязвимых функций: strcpy, spritnf на их аналоги strncpy, snprintf, в параметры которых входит размер строки). Существуют гибкие средства автоматически выполняющие действия имитирующие переполнение буфера на этапе отладки программы. Также следует упомянуть об утилитах автоматического поиска уязвимостей в исходном коде программы. Указанные методы и средства позволяют создавать более защищенные программы, но не решают проблему в принципе, а лишь минимизируют число уязвимостей по переполнению буфера.

Использование неисполнимых буферов

Суть метода заключается в запрещении исполнения кода в сегментах данных и стека, т.е. параметры сегментов данных и стека содержат только атрибуты записи и чтения, но не исполнения. Например, для реализации неисполняемого стека существуют “заплаты” для ОС Solaris и Linux. Однако ограничение на исполнение данных приводит к проблеме несовместимости. Исполняемый стек необходим для работы многим программам, так как на его основе генерируется код компиляторами, реализуются системные функции операционных систем, реализуется автоматическая генерация кода. Защита с использованием неисполнимых буферов предотвратит только атаки с внедрением кода, но не поможет при других видах атак.

Применение проверок выхода за границы

В основе данного метода лежит выполнение проверок выхода за границы переменной при каждом обращении к ней. Это предотвращает все возможные атаки по переполнению буфера, так как полностью исключает само переполнение. Проверки выхода за границы переменной опционально реализованы в некоторых компиляторах Си, например, Compaq C, cc в Tru64 Unix, cc в Alpha Linux. Следует отметить, что реализованные проверки ограничены только точными ссылками на элементы массивов, но не производятся для указателей. Существует также “заплата” для gcc, которая позволяет компилировать программы с полностью реализованной (включая проверку указателей) проверкой выхода за границы массивов. Однако, у этого решения есть существенный недостаток - значительное (до 30 раз) снижение производительности программы. Другие системы осуществляют проверки при доступе к памяти, выполняя вставки дополнительного объектного кода проверок во все места программы, где есть обращения к памяти. Вставки могут производится как до сборки объектных файлов (Purify) так и после (Pixie). Такие проверки сказываются на производительности с ее уменьшением от 2 до 5 раз и скорее подходят для отладки.

Применение проверок целостности

Данная методика заключается в изменении пролога и эпилога всех функций с целью проверки целостности адреса возврата из функции при помощи так называемого "canary word". Схема защиты изображена на рисунке 1.

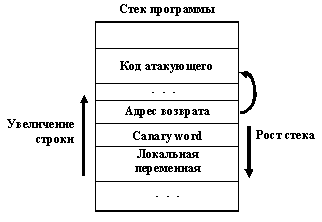


Рисунок 1.

Измененный пролог каждой функции выполняет занесение в стек “canary word”, а эпилог проверку содержимого стека, занесенного ранее и, в случае, нарушения останавливает программу с предупреждающим сообщением. При атаке с искажением адреса возврата неизбежно произойдет искажение “canary word”, что и будет признаком нарушения целостности. Таким образом, целостность адреса возврата определяется целостностью “canary word”. При известном значении “canary word” атакующий может организовать подмену адреса возврата без нарушения целостности. Поэтому “canary word” формируется особым образом:

1. имеет значения 0, CR, LF, EOF, что не позволит провести атаку при переполнении буфера в библиотечных функциях Си, так как данные значения являются признаками конца строки;
2. имеет псевдослучайное значение, генерируемое при каждом запуске программы.
3. **Проверка входных данных приложения.**

В основе проверки входных данных лежит непосредственное преобразование какого-то обобщенного типа (в основном это строчные типы данных для веб-приложений) в конкретный тип, который в дальнейшем будет являться элементом управления логики приложения. Именно поэтому нужно четко понимать, что данные, приходящие со стороны клиента, всегда несут угрозу, которая может в лучшем случае временно нейтрализовать работу приложения, в худшем выдать данные, которые являются секретными, т. е. пароли, ключи и т. д.

Данные правила касаются не только веб-ориентированных систем, но и обычных настольных приложений. Можно выделить 3 направления обработки входных данных:

* **проверка данных из внешних источников** – это основное направление, на котором нужно акцентировать внимание. Данные могут приходить не только от клиента, но и из базы данных или файла, который храниться на сервере, поэтому:

1. числовые данные должны попадать в указанный диапазон;
2. необходима проверка на корректную длину для строчных данных;
3. проверка на строгие форматы (даты, email, веб-ссылки и т. д.);
4. защита от исполняемого кода и SQL-инъекций.

* **проверка данных из внутренних источников –** это направление говорит о том, что необходимо проверять различные данные внутри реализации логики, т. е. параметры функций или методов класса. Данное направление является особенно важным для командной разработки и разработчиков различного рода библиотек, в которых необходимо проверять формат входных параметров.
* **реализация механизма вывода сообщений об ошибках –** необходимо реализовать способ сообщения пользователю об ошибке, но не о каждой, а только о той которую он сможет исправить, так как вывод всех ошибок может повлечь раскрытие механизмов реализации логики приложения.

1. **Атаки типа "Внедрение SQL-кода" (SQL-injection). Защита от атак типа"Внедрение SQL-кода" (SQL-injection).**

Атака типа SQL Injection — это атака, при которой производится вставка вредоносного кода в строки, передающиеся затем в экземпляр SQL Server для синтаксического анализа и выполнения. Любая процедура, создающая инструкции SQL, должна рассматриваться на предмет уязвимости к вставке небезопасного кода, поскольку SQL Server выполняет все получаемые синтаксически правильные запросы. Даже параметризованные данные могут стать предметом манипуляций опытного злоумышленника.

Основная форма атаки SQL Injection состоит в прямой вставке кода в пользовательские входные переменные, которые объединяются с командами SQL и выполняются. Менее явная атака внедряет небезопасный код в строки, предназначенные для хранения в таблице или в виде метаданных. Когда впоследствии сохраненные строки объединяются с динамической командой SQL, происходит выполнение небезопасного кода.

Существуют различные технические реализации SQL injection:

* Неотфильрованные терминальные символы (escape characters) - возникает если в программе не производится фильтрация терминальных символов (служебных для SQL Server, как например, кавычки, дефис, слеш). Например выборка пользователей производтся с помощью кода :

*statement = "SELECT \* FROM users WHERE name = '" + userName + "';"*

тогда добавив к строке "имя пользователя" ' or '1'='1' -- ' получим запрос, который вернет нам имена всех пользователей системы:

*SELECT \* FROM users WHERE name = '' OR '1'='1' -- ';*

* Использование UNION для выборки данных из другой таблицы:

*SELECT id\_news, header, body, author FROM news WHERE id\_news = ID*

воспользовавшись уязвимостью вернём вместо новостей логин и пароль админа:

*SELECT id\_news, header, body, author FROM news WHERE id\_news = -1 UNION SELECT 1,username,password,1 FROM admin*

* Отсутствие проверки типа - при вставке в таблицу не проверяются типы вставляемых значений (строка-число) :

*statement = "SELECT \* FROM userinfo WHERE id = " + a\_variable + ";"*

попытавшись вставить значение *1;DROP TABLE users* в поле удалим таблицу Users

*SELECT \* FROM userinfo WHERE id=1;DROP TABLE users;*

* Атака "вслепую" - когда вставка вредоносных данных не ведет напрямую к их исполнению, а вот при действиях на другой странице вредоносный код, записанный в поле будет выполнен (например операцией Select)

вставим в поле bookId текст OOk14cd' AND '1'='1 тогда запрос

*SELECT booktitle FROM booklist WHERE bookId='ID'*

вернет нам все книги вместо одной:

*SELECT booktitle FROM booklist WHERE bookId = 'OOk14cd' AND '1'='1';*

Внедрение вредоносного кода может проводиться через поля формы, Get-параметры, Cookie и другие способы взаимодействия приложения с данными.

«SQL Injection» атаки возможны в приложениях, не проверяющих полученные от пользователя данные. Уязвимое приложение использует полученные данные для построения динамического SQL запроса и последующего его выполнения.

string sqlQuery = "SELECT COUNT(\*) FROM Users WHERE UserName='"+username.Text+

"' AND Password='" + password.Text + "'"

SqlCommand sqlCmd = new SqlCommand(sqlQuery, sqlConn);

int userCount = (int)sqlCmd.ExecuteScalar();

if (1 == userCount) { // Авторизация пройдена успешно }

Одним из методов защиты является отказ от использования динамических SQL запросов. Параметризованные запросы и хранимые процедуры защитят ваше приложение от подобных атак.

string sqlQuery = "SELECT COUNT(\*) FROM Users WHERE UserName=@username AND Password=@password";

SqlCommand sqlCmd = new SqlCommand(sqlQuery, sqlConn);

sqlCmd.Parameters.Add("@username", SqlDbType.VarChar).Value = username.Text;

sqlCmd.Parameters.Add("@password", SqlDbType.VarChar).Value = password.Text;

int userCount = (int)sqlCmd.ExecuteScalar();

if (1 == userCount) { // Авторизация пройдена успешно }

Еще лучшим методом защиты БД является использование хранимых процедур. Защита базы данных в этом случае обеспечивается за счет того, что пользователю даются только права на выполнение хранимых процедур. Ваш код затем обращается к хранимым процедурам БД от имени подобной непривилегированной учетной записи. Хранимые процедуры в свою очередь выполняют операции INSERT/UPDATE/DELETE. Вызов хранимой процедуры для авторизации пользователя может выглядеть, например, так:

SqlCommand sqlCmd = new SqlCommand("CheckLogon", sqlConn);

sqlCmd.CommandType = CommandType.StoredProcedure;

sqlCmd.Parameters.Add ("@username", SqlDbType.VarChar).Value = username.Text;

sqlCmd.Parameters.Add ("@password", SqlDbType.VarChar).Value = password.Text;

int userCount = (int)sqlCmd.ExecuteScalar();

Также следует отключать подробные сообщения об ошибках выводимые ASP.NET, чтобы исключить использование этой информации злоумышленником. Именно из текста сообщения об ошибке злоумышленник может узнать о структуре БД.

В случае ошибки в веб-приложении пользователю должно выдаваться минимум информации. Ни в коем случае не следует выводить полное сообщение об ошибке. В общем случае достаточно информировать посетителя о ее наличии, а полный текст сообщения об ошибке занести в журнал событий и/или отправить уведомление администратору сайта по электронной почте.

Некоторые способы защиты в PHP:

1) для целых и дробных величин, перед их использованием в запросе достаточно привести величину к нужному типу.

$id=(int)$id; $total=(float)$total;

2) для строковых параметров, которые не используются в like, regexp и т.д., экранируем кавычки.

$str=addslashes($str); или mysql\_escape\_string($str);

1. в строках, которые предполагается использовать внутри like, regexp и тд, необходимо так же заэкранировать специальные символы, применяющиеся в этих операторах, если это необходимо.

**21. Методы и особенности защиты мультимедиа информации.**

Для защиты мультимедиа информации используется стеганография. Стеганография занимается разработкой методов передачи секретной информации, в которых скрывается само наличие передачи. Методы стеганографии позволяют встраивать секретные сообщения в безобидные послания так, чтобы невозможно было заподозрить существование встроенного тайного послания. Стеганографическая система, или стегосистема, – совокупность средств и методов, которые используются для формирования скрытого канала передачи информации. Направления: непосредственное сокрытие данных (сообщений), цифровые водяные знаки (watermark) и цифровые отпечатки (fingerprint).

Методы:

**LSB** (Least Significant Bit, наименьший значащий бит) — суть этого метода заключается в замене последних значащих битов в контейнере (изображения, аудио или видеозаписи) на биты скрываемого сообщения. Разница между пустым и заполненным контейнерами должна быть не ощутима для органов восприятия человека.

Методы LSB являются неустойчивыми ко всем видам атак и могут быть использованы только при отсутствии шума в канале передачи данных.

Другие методы скрытия информации в графических файлах ориентированы на форматы файлов с потерей, к примеру, JPEG. В отличие от LSB они более устойчивы к геометрическим преобразованиям.

**Метод Patchwork.** Данный метод используется для постановки водяных знаков. Он основан на внесении изменений в два участка изображения: на участке *А* яркость изображения незначительно увеличивается, а на участке *В* – уменьшается.

**Эхо-методы** применяются в цифровой аудиостеганографии и используют неравномерные промежутки между эхо-сигналами для кодирования последовательности значений. При наложении ряда ограничений соблюдается условие незаметности для человеческого восприятия. Эхо-методы устойчивы к амплитудным и частотным атакам, но неустойчивы к [атакам](http://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%B0%D1%82%D0%B0%D0%BA%D0%B0) по времени.

**Фазовое кодирование** (phase coding, фазовое кодирование) — также применяется в цифровой аудиостеганографии. Происходит замена исходного звукового элемента на относительную [фазу](http://ru.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B7%D0%B0_%D0%BA%D0%BE%D0%BB%D0%B5%D0%B1%D0%B0%D0%BD%D0%B8%D0%B9), которая и является секретным сообщением. Фаза подряд идущих элементов должна быть добавлена таким образом, чтобы сохранить относительную фазу между исходными элементами. Фазовое кодирование является одним из самых эффективных методов скрытия информации.

**Метод встраивания сообщения** заключается в том, что специальная случайная последовательность встраивается в контейнер, затем, используя согласованный фильтр, данная последовательность [детектируется](http://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%82%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Данный метод позволяет встраивать большое количество сообщений в контейнер, и они не будут создавать помехи друг другу. Метод заимствован из [широкополосной](http://ru.wikipedia.org/wiki/Broadband) связи.

1. **Стеганографические методы защиты графических файлов. Метод LSB.**

Метод замены наименее значащего бита (LSB – Least Significant Bit) наиболее распространен среди методов замены в пространственной области.

Младший значащий бит изображения несет в себе меньше всего информации. Известно, что человек в большинстве случаев не способен заметить изменений в этом бите. Фактически, LSB – это шум, поэтому его можно использовать для встраивания информации путем замены менее значащих битов пикселей изображения битами секретного сообщения. При этом для изображения в градациях серого (каждый пиксель изображения кодируется одним байтом) объем встроенных данных может составлять 1/8 от общего объема контейнера. Если же модифицировать два младших бита (что также практически незаметно), то данную пропускную способность можно увеличить еще вдвое.

Популярность данного метода обусловлена его простотой и тем, что он позволяет скрывать в относительно небольших файлах большие объемы информации (пропускная способность создаваемого скрытого канала связи составляет при этом от 12,5 до 30%). Метод зачастую работает с растровыми изображениями, представленными в формате без компрессии (например, BMP и GIF).

Метод LSB имеет низкую стеганографическую стойкость к атакам пассивного и активного нарушителей. Идея метода проста, однако в подобной тривиальной форме он крайне не эффективен. Недостатки: изменения изображения приводят к практически полной потери данных, факт внедрения легко обнаружить. Достоинства: большой объем встраиваемых данных (в идеале до 3/8 от размера изображения-контейнера), простота реализации. Рассмотрим возможные улучшения:

* Использования вместо всех трех компонент пикселя только одну (желательно синию т.к человеческий глаз менее всего воспринимает именно этот цвет)
* Использование специальных последовательностей битов для обозначения первого пикселя с которого начнется внедрение сообщения
* Использовать для внедрения не каждый последующий пиксель, а, например, каждый 3-й начиная с первого.
* Использовать для внедрения сообщения пиксели по заранее известной функции(правилу) начиная с первого (при этом порядок следования битов в сообщении может не совпадать с порядком следования пикселей изображения-контейнера, в который вшит бит), например: 2-бит сообщения внедряем в седьмой пиксель (относительно первого), 3-бит в пятый пиксель и.т.д
* Представление номера пекселя для внедрения бита, как функцию(правило), зависящей от номера первого пикселя (относительно начала изображения)
* Дублирование сообщения, помещенного в изображение-контейнер, для лучшей сохранности при изменении изображения.
* Использование самокорректирующий кодов

1. **Стеганографические методы защиты графических файлов. Метод Patchwork.**

Данный метод используется для постановки водяных знаков. Он основан на внесении незначительных изменений в изображение (незначительном увеличении яркости пикселей А и незначительном уменьшении яркости пикселей В).

Опишем основную идею, заложенную в данный метод. Если взять изображение с равномерно распределенной яркостью и высчитать разницу яркости S между двумя случайно взятыми точками (яркость которых соответственно равны a и b)

*S* = *a* – *b*,

то среднее значение разницы S (обозначим его *MS*)после многократного повторения данной процедуры будет равно 0. Т.е. после повторения данной процедуры *п* раз, полагая, что значения *а*, *b* и *S* на *i*-й итерации равны *ai*, *bi* и *Si* соответственно. Тогда *MS* выразится как



Если же мы внесем незначительные изменения яркости в пиксели, которые будут определяться с помощью секретного ключа пользователя по определенному алгоритму, то среднее значение разницы данных пикселей будет существенно отличаться от нуля, но распределение яркости по всему изображению практически не изменится. Это изменение будет незаметно для глаза человека и его невозможно будет определить без знания секретного ключа.

Алгоритм для данного метода следующий:

С помощью секретного ключа как начального значения для криптостойкого генератора псевдослучайных чисел, сгенерировать координаты пары точек (*ai*, *bi*).

1. Увеличить яркость изображения в точке *ai* на значение *,* обычно выбираемое в диапазоне от 1 до 5 для изображения с 256 уровнями яркости.
2. Уменьшить яркость изображения в точке *bi* на такое же значение *.*
3. Повторить шаги 1–3 *п* раз (*п* выбирается порядка 10 000).

После модификации значение ** при выборе точек по секретному ключу может быть выражено как

.

Таким образом, с каждым новым шагом приведенного выше алгоритма накапливается отклонение на величину .



Наличие подобного отклонения от ожидаемого значения свидетельствует о наличии встроенной в изображение метки. Таким образом, владелец может доказать свои интеллектуальные права на изображение, предъявив секретный ключ, который использовался для встраивания метки в изображения.

1. **Удаленные атаки на распределенные вычислительные системы. Типовые атаки. Атака типа "ложный ARP-сервер". Атака типа Smurf. Защита от удаленных атак.**

Итак, удаленные атаки можно классифицировать по следующим признакам:

**1. По характеру воздействия**

* пассивное (класс 1.1)
* активное (класс 1.2)

Пассивное - воздействие, которое не оказывает непосредственного влияния на работу системы, но может нарушать ее политику безопасности. Примером пассивного типового удаленного воздействия в РВС служит прослушивание канала связи в сети.

Активное - воздействие, оказывающее непосредственное влияние на работу системы (изменение конфигурации РВС, нарушение работоспособности и т. д.) и нарушающее принятую в ней политику безопасности. Практически все типы удаленных атак являются активными воздействиями.

**2. По цели воздействия**

* нарушение конфиденциальности информации либо ресурсов системы (класс 2.1)
* нарушение целостности информации (класс 2.2)
* нарушение работоспособности (доступности) системы (класс 2.3)

Этот классификационный признак является прямой проекцией трех основных типов угроз - раскрытия, целостности и отказа в обслуживании.

Перехват информации ведет к нарушению ее конфиденциальности. Примером перехвата информации может служить прослушивание канала в сети. В этом случае имеется несанкционированный доступ к информации без возможности ее искажения.

Возможность искажения информации означает либо полный контроль над информационным потоком между объектами системы, либо возможность передачи сообщений от имени другого объекта. Таким образом, очевидно, что искажение информации ведет к нарушению ее целостности.Примером удаленной атаки, цель которой нарушение целостности информации, может служить типовая удаленная атака (УА) "Ложный объект РВС".

Принципиально другой целью атаки является нарушение работоспособности системы. В этом случае основная цель - добиться, чтобы операционная система на атакуемом объекте вышла из строя и для всех остальных объектов системы доступ к ресурсам атакованного объекта был бы невозможен. Примером удаленной атаки, может служить типовая УА "Отказ в обслуживании”.

**3. По условию начала осуществления воздействия**

* Атака по запросу от атакуемого объекта (класс 3.1)

В этом случае атакующий ожидает передачи от потенциальной цели атаки запроса определенного типа, который и будет условием начала осуществления воздействия. Примером подобных запросов в ОС Novell NetWare может служить SAP-запрос (атака описана в [9]), а в сети Internet - DNS- и ARP-запросы Важно отметить, что данный тип удаленных атак наиболее характерен для распределенных ВС.

* Атака по наступлению ожидаемого события на атакуемом объекте (класс 3.2)

В этом случае атакующий осуществляет постоянное наблюдение за состоянием операционной системы удаленной цели атаки и при возникновении определенного события в этой системе начинает воздействие. Примером такого события может быть прерывание сеанса работы пользователя с сервером в ОС Novell NetWare без выдачи команды LOGOUT [9].

* Безусловная атака (класс 3.3)

В этом случае начало осуществления атаки безусловно по отношению к цели атаки, то есть атака осуществляется немедленно и безотносительно к состоянию системы и атакуемого объекта. Следовательно, в этом случае атакующий является инициатором начала осуществления атаки.

**4. По наличию обратной связи с атакуемым объектом**

* с обратной связью (класс 4.1)
* без обратной связи (однонаправленная атака) (класс 4.2)

Удаленная атака, осуществляемая при наличии обратной связи с атакуемым объектом, характеризуется тем, что между атакующим и целью атаки существует обратная связь, которая позволяет атакующему адекватно реагировать на все изменения, происходящие на атакуемом объекте. Подобные удаленные атаки наиболее характерны для распределенных ВС.

Удаленным атакам без обратной связи не требуется реагировать на какие-либо изменения, происходящие на атакуемом объекте. Атаки данного вида обычно осуществляются передачей на атакуемый объект одиночных запросов, ответы на которые атакующему не нужны. Примером однонаправленных атак является типовая УА "Отказ в обслуживании”.

**5. По расположению субъекта атаки относительно атакуемого объекта**

* внутрисегментное (класс 5.1)
* межсегментное (класс 5.2)

Важно отметить, что межсегментная удаленная атака представляет гораздо большую опасность, чем внутрисегментная. Это связано с тем, что в случае межсегментной атаки объект её и непосредственно атакующий могут находиться на расстоянии многих тысяч километров друг от друга, что может существенно воспрепятствовать мерам по отражению атаки.

**6. По уровню эталонной модели ISO/OSI, на котором осуществляется воздействие**

* физический (класс 6.1)
* канальный (класс 6.2)
* сетевой (класс 6.3)
* транспортный (класс 6.4)
* сеансовый (класс 6.5)
* представительный (класс 6.6)
* прикладной (класс 6.7)

**Ложный ARP сервер**

Базовым сетевым протоколом обмена данных является протокол IP. Это означает, что для передачи любого пакета данных между двумя любыми узлами сети, в заголовке пакета должен быть указан IP адрес получателя. В соответствии с моделью OSI при передаче пакетов между узлами имеет место вложенность пакетов. А именно: пакет TCP транспортного уровня вложен в пакет IP сетевого уровня, который вложен в пакет канального уровня (физическая адресация по MAC адресам). Поэтому каждый пакет данных, в конечном счете, посылается на аппаратный MAC адрес сетевого адаптера получателя. Первоначально узел может не иметь информации о MAC адресах других узлов, находящихся с ним в одном сегменте сети, в том числе не иметь информации и о MAC адресе маршрутизатора. Для решения этой задачи используется протокол ARP, предназначенный для определения адреса канального уровня (физическая адресация, MAC) модели OSI по известному адресу сетевого уровня (логическая адресация, IP) модели OSI. Соответствия адресов MAC и IP хранятся в ARP таблицах. При первом обращении к сетевым ресурсам узел отправляет широковещательный ARP запрос, где указывает IP адрес маршрутизатора и просит сообщить его MAC адрес. Широковещательный запрос получат все узлы в данном сегменте сети, в том числе и маршрутизатор. Маршрутизатор внесет запись о запросившем хосте в свою ARP таблицу, а затем отправит ответ, в котором сообщит свой MAC адрес. Полученный в ARP ответе MAC адрес будет занесен в ARP таблицу, расположенную на узле.

Атака «Ложный ARP сервер» сводится к тому, что атакующий узел, получив широковещательный ARP запрос, посылает ложный ARP ответ, в котором объявляет себя искомым узлом, маршрутизатором например. Затем атакующий узел посылает ARP запрос, указав в качестве своего IP адреса любой свободный в данном сегменте IP адрес, и в дальнейшем ведёт работу с данного IP адреса как с маршрутизатором, так и с обманутыми узлами (так работают и proxy сервера). Таким образом, все пакеты, которые обманутый узел отсылает и принимает через маршрутизатор, будут на самом деле проходить ещё и через атакующий узел, причём совершенно незаметно для самого обманутого узла. Маршрутизатор, отсылая пакеты, будет на самом деле слать их атакующему узлу, и обманутый узел, отсылая пакеты на маршрутизатор, также будет слать их атакующему узлу. Это схема полного перехвата.

Если атакующий узел не посылает ARP запроса, то получается петлевая, частичная, схема перехвата. Т.к. ARP запрос, переданный изначально обманутым узлом, дойдёт и до настоящего маршрутизатора, который, занеся данные в свою ARP таблицу, в таком случае, будет слать пакеты, предназначенные для обманутого узла, туда, куда нужно, минуя атакующий узел. И только те пакеты, которые исходят от обманутого узла, будут проходить через атакующий узел.

Атака является внутрисегментной и представляет угрозу, в случае если атакующий узел находится в одном сегменте сети с атакуемыми узлами. Осуществить внутрисегментную удаленную атаку значительно легче, чем межсегментную. По статистике нарушений информационной безопасности вычислительных сетей известно, что большинство сетевых атак произведены именно собственными сотрудниками, т.е. изнутри.

**Атака типа Smurf**

# SMURF

SMURF относится к атакам типа denial-of-service (DoS, отказ от обслуживания) и работает на основе Internet Control Message Protocol (ICMP).

Программа, реализующей одну из его функций - команда "PING".

Эта программа предназначена для определения доступности какого-либо хоста посылкой пакета эхо-запроса ICMP. Если получен пакет с эхо-ответом, то хост считается доступным. Но пакет может быть отправлен не по адресу конкретного хоста, а по широковещательному (broadcast) адресу сети.

В этом случае пакет будет доставлен всем машинам в этой сети. Очевидно, что если на широковещательный пакет ответят несколько тысяч машин, то компьютер-инициатор эхо-запроса может не справиться с обработкой эхо-ответов.

Схема DoS: посылается ICMP пакет, в котором адрес отправителя является адресом жертвы (спуфинг), а в качестве получателя указывается широковещательный адрес посредника. Компьютеры посредника отвечают на полученный эхо-запрос посылкой пакетов по адресу отправителя, т.е. жертве. В дальнейшем компьютер может временно оказаться неспособным работать в сети, может "зависнуть", возможно и нарушение функционирования самой сети из-за возросшего трафика.

Сделать невозможной атаку SMURF могут маршрутизаторы в сети посредника. Если они фильтруют широковещательный трафик, то компьютеры данной сети не будут посредниками в деструктивных действиях внешнего злоумышленника против неизвестной жертвы.

Но, инициатор атаки может находиться и внутри сети. В этом случае маршрутизаторы не помогут, и ответственность за защиту возлагается на хосты, которые также не должны отвечать на broadcast ICMP пакеты.

Smurf (атака, направленная на ошибки реализации TCP-IP протокола). Сейчас этот вид атаки считается экзотикой, однако раньше, когда TCP-IP протокол был достаточно новым, в нём содержалось некоторое количество ошибок, которые позволяли, например, подменять IP адреса. Однако, этот тип атаки применяется до сих пор. Некоторые специалисты выделяют TCP Smurf, UDP Smurf, ICMP Smurf. Конечно, такое деление основано на типе пакетов.

Рекомендации: коммутаторы CISCO предоставляют хорошую защиту, как и многие другие, а также свежее ПО и межсетевые экраны; необходимо блокировать широковещательные запросы.

1. **Типовые удаленные атаки**

Типовая удаленная атака – это удаленное информационное разрушающее воздействие, программно осуществляемое по каналам связи и характерное для любой распределенной вычислительной сети. Удаленная атака "анализ сетевого трафика" Тип удаленного воздействия, заключающийся в прослушивании канала связи. Анализ сетевого трафика позволяет: • изучить логику работы распределенной вычислительной сети, это достигается путем перехвата и анализа пакетов обмена на канальном уровне. • перехватить поток данных, которыми обмениваются объекты сети, т. е. удаленная атака данного типа заключается в получении несанкционированного доступа к информации, которой обмениваются пользователи. Удаленная атака "подмена доверенного объекта" В том случае, когда в вычислительной сети используют нестойкие алгоритмы идентификации удаленных объектов, то оказывается возможной типовая удаленная атака, заключающаяся в передаче по каналам связи сообщений от имени произвольного объекта или субъекта сети (т. е. подмена объекта или субъекта сети). Подмена доверенного объекта распределенной вычислительной сети является активным воздействием, совершаемым с целью нарушения конфиденциальности и целостности информации, по наступлению на атакуемом объекте определенного события. Удаленная атака "ложный объект" Принципиальная возможность реализации данного вида удаленной атаки в вычислительных сетях также обусловлена недостаточно надежной идентификацией сетевых управляющих устройств (например, маршрутизаторов). Целью данной атаки является внедрение в сеть ложного объекта путем изменения маршрутизации пакетов, передаваемых в сети. Внедрение ложного объекта в распределенную сеть может быть реализовано навязыванием ложного маршрута, проходящего через ложный объект. Получив контроль над проходящим потоком информации между объектами, ложный объект вычислительной сети может применять различные методы воздействия на перехваченную информацию, например: • селекция потока информации и сохранение ее на ложном объекте (нарушение конфиденциальности); • модификация информации: • подмена информации (нарушение целостности). Удаленная атака "отказ в обслуживании" В зависимости от различных параметров объектов вычислительной сети, основными из которых являются быстродействие ЭВМ, объем оперативной памяти и пропускная способность канала связи – количество одновременно устанавливаемых виртуальных подключений ограничено, соответственно, ограничено и число запросов, обрабатываемых в единицу времени. С этой особенностью работы вычислительных сетей связана типовая удаленная атака "отказ в обслуживании". Реализация этой угрозы возможна, если в вычислительной сети не предусмотрено средств аутентификации (проверки подлинности) адреса отправителя. В такой вычислительной сети возможна передача с одного объекта (атакующего) на другой (атакуемый) бесконечного числа анонимных запросов на подключение от имени других объектов. Результат применения этой удаленной атаки – нарушение на атакованном объекте работоспособности соответствующей службы предоставления удаленного доступа, то есть невозможность получения удаленного доступа с других объектов вычислительной сети – отказ в обслуживании.

1. **Удаленные атаки на распределенные вычислительные системы. Типовые атаки. SYN Flooding. Подмена одного из субъектов TCP-соединения в сети Internet. Защита от удаленных атак.**

**Подмена одного из субъектов TCP-соединения.**

Предположим, что хосту **А** необходимо создать TCP-соединение с хостом **В**. Тогда **А** посылает на **В** следующее сообщение:

1. **A - > B:**  SYN, ISSa

Это означает, что в передаваемом **A** сообщении установлен бит SYN (synchronize sequence number), а в поле Sequence Number установлено начальное 32-битное значение ISSa (Initial Sequence Number). В отвечает:

**2. B - > A:**  SYN, ACK, ISSb, ACK(ISSa+1)

В ответ на полученный от **А** запрос **В** отвечает сообщением, в котором установлен бит SYN и установлен бит ACK; в поле Sequence Number хостом **В** устанавливается свое начальное значение счетчика - ISSb; поле Acknowledgment Number содержит значение ISSa, полученное в первом пакете от хоста **А** и увеличенное на единицу. А, завершая рукопожатие (handshake), посылает:

**3. A - > B:**  ACK, ISSa+1, ACK(ISSb+1)

В этом пакете установлен бит ACK; поле Sequence Number содержит ISSa + 1; поле Acknowledgment Number содержит значение ISSb + 1. Посылкой этого пакета на хост **В** заканчивается трехступенчатый handshake, и TCP-соединение между хостами **А** и **В** считается установленным. Теперь хост **А** может посылать пакеты с данными на хост **В** по только что созданному виртуальному TCP-каналу:

**4. A - > B:**  ACK, ISSa+1, ACK(ISSb+1); DATA

Из рассмотренной выше схемы создания TCP-соединения видно, что единственными идентификаторами TCP-абонентов и TCP-соединения являются два 32-битных параметра Sequence Number и Acknowledgment Number. Следовательно, для формирования ложного TCP-пакета атакующему необходимо знать текущие идентификаторы для данного соединения - ISSa и ISSb.

Атакующему достаточно, подобрав соответствующие текущие значения идентификаторов TCP-пакета для данного TCP-соединения, послать пакет с *любого хоста в сети* *Internet* от имени одного из участников данного соединения (например, от имени клиента), и данный пакет будет воспринят как верный!

Для осуществления описанной выше атаки необходимым и достаточным условием является знание двух текущих 32-битных параметров ISSa и ISSb, идентифицирующих TCP-соединение. Рассмотрим возможные способы их получения.

В том случае, когда атакующий находится в одном сегменте с целью атаки или через его сегмент проходит трафик предполагаемого объекта атаки, то задача получения значений ISSa и ISSb является тривиальной и решается путем анализа сетевого трафика. Следовательно, надо четко понимать, что протокол TCP позволяет, в принципе, защитить соединение только в случае невозможности перехвата атакующим сообщений, передаваемых по данному соединению, то есть в случае нахождения атакующего в других сегментах относительно абонентов TCP-соединения.

Для защиты от таких атак необходимо использовать ОС, в которых начальное значение идентификатора генерируется действительно случайным образом. Также необходимо использовать защищённые протоколы типа SSL, S-HTTP, Kerberos и т.д.

**Защита от удалённых атак**

Удалённая сетевая атака - информационное разрушающее воздействие на распределённую вычислительную систему, осуществляемое программно по каналам связи. Методы защиты можно разделить на 2 группы:

**1) Административные методы защиты от удаленных атак в сети Internet**.

Они являются наиболее простыми и дешевыми методами защиты от информационно-разрушающих воздействий. Включают в себя (*задача* - решение):

1.*Защита от анализа сетевого трафика* - не допускать использование базовых протоколов TELNET и FTP для предоставления удаленного авторизованного доступа к ресурсам своих систем и применять стойкие криптоалгоритмы защиты IP-потока;

2.*Защита от ложного ARP-сервера* - создание сетевым администратором статической ARP-таблицы в виде файла (в ОС UNIX обычно /etc/ethers), куда необходимо внести соответствующую информацию об адресах;

3.*Защита от ложного DNS-сервера* - использовать для общения с хостами и с другими DNS-серверами только протокол TCP, а не UDP;

4.*Защита от навязывания ложного маршрута при использовании протокола ICMP* - в защищенном сегменте IP-сети не использовать ОС Linux 1.2.8, Windows '95 и Windows NT 4.0;

5.*Защита от отказа в обслуживании* - использовать как можно более мощные компьютеры. Чем больше число и частота работы процессоров, чем больше объем оперативной памяти, тем более надежной будет работа сетевой ОС;

6.*Защита от подмены одной из сторон при взаимодействии с использованием базовых протоколов семейства TCP/IP* - в качестве базового протокола использовать протокол TCP и сетевые ОС со случайным начальным значением TCP-соединения.

**2) Программно-аппаратные методы защиты от удаленных атак в сети Internet**.

К ним относятся (*метод* - описание):

1.*Аппаратные шифраторы сетевого трафика* – шифрование трафика протоколов Ethernet, Fibre Channel, SDH и IP на высоких скоростях;

2.*Методика Firewall, реализуемая на базе программно-аппаратных средств* – комплекс средств, осуществляющий контроль и фильтрацию проходящих через него сетевых пакетов в соответствии с заданными правилами;

3.*Защищенные сетевые криптопротоколы* –

- SSL - универсальный протокол защиты соединения, функционирующий на сеансовом уровне OSI. Использует криптографию с открытым ключом;

- SKIP - стандарт инкапсуляции IP-пакетов, позволяющий в стандарте IPv4 на сетевом уровне обеспечить защиту соединения и передачу данных;

- S-HTTP протокол позволяет обеспечить надежную криптозащиту только HTTP-документов Web-севера и функционирует на прикладном уровне модели OSI.

4.*Программно-аппаратные анализаторы сетевого трафика -* программа или программное устройство, предназначенное для перехвата и последующего анализа.

26. Защищенные сетевые протоколы. Протокол IPSec. Ассоциация безопасности.

IPsec – это комплект протоколов сетевого уровня, обеспечивающих аутентификацию и защиту данных. В отличие от протокола SSL, работающего на транспортном уровне (4 уровень сетевой модели OSI и выше), IPsec – протокол более низкого уровня. Он работает на сетевом уровне (3 уровень OSI) и обеспечивает безопасность передачи IP-пакетов. Наиболее известная область применения протокола IPsec – это организация виртуальных частных сетей (VPN), позволяющих пользователю удалённо получать безопасный доступ к защищенной сети. IPsec является обязательной частью протокола IPv6 и необязательным расширением IPv4.

Перед началом обмена защищёнными пакетами между двумя сторонами устанавливается одно или несколько безопасных соединений (называемых **SA** -Security Associations). Для каждого соединения стороны договариваются о контексте безопасности – используемых криптоалгоритмах и ключах. Результатом этого согласования является установление индекса параметров безопасности (**SPI**) для каждого конкретного SA. SPI – это некоторый номер, идентифицирующий канал SA и связанные с ним параметры безопасности. Этот номер передаётся вместе с каждым защищённым пакетом и указывает адресату, каким образом расшифровать полученный пакет (или проверить его аутентичность).

IPsec включает в себя 2 типа протоколов: протоколы, обеспечивающие защиту передаваемой информации (**AH** и **ESP)** и протоколы обмена криптографическими ключами (**IKE)**.

27. Защищенные сетевые протоколы. Протокол IPSec. Протокол AH. Транспортный режим работы. Туннельный режим работы.

Протокол **AH** служит только для аутентификации данных, а **ESP** обеспечивает и аутентификацию, и конфиденциальность передаваемой информации. Оба протокола могут работать в двух режимах:

1. **транспортный режим**, который обеспечивает защиту только передаваемых данных (полезной нагрузки IP-пакета, которая может представлять собой TCP, UDP или другой пакет);
2. **туннельный режим** – более сложный режим, обеспечивающий защиту всего IP-пакета, включая его заголовок. В этом режиме исходный IP-пакет вкладывается внутрь нового IP-пакета. Основное назначение этого режима – защита адресных полей исходного заголовка и обеспечение туннелирования, т.е. создания виртуального канала обмена информацией между двумя пользователями, адреса которых отличаются от указанных в несущем IP-пакете и (при наличии шифрования) остаются скрытыми от внешнего мира.

Протокол **AH (Authentication Header)** предполагает добавление к IP-пакету AH-заголовка, основные поля которого – это индекс SPI и данные аутентификации, представляющие собой специальным образом вычисленное хэш-значение от полезной нагрузки исходного IP-пакета и части его заголовка. В случае использования транспортного режима AH-заголовок вставляется между исходным IP-заголовком и его данными (представляющими, например, TCP-пакет). В туннельном режиме AH заголовок вместе со всем исходным IP-пакетом оборачиваются в новый IP-пакет.

Применение AH защищает от атак, связанных с несанкционированным изменением содержимого пакета, и в том числе от подмены исходного адреса сетевого уровня.

28. Защищенные сетевые протоколы. Протокол IPSec. Протокол ESP. Транспортный режим работы. Туннельный режим работы.

Протокол **ESP (Encapsulating Security Payload)** позволяет зашифровать передаваемые данные. При этом могут использоваться различные алгоритмы (например, DES, AES и Blowfish). Пакет ESP содержит, помимо прочего, поле SPI и поле зашифрованных данных, которые включают исходную полезную нагрузку (в транспортном режиме) либо целый исходный IP-пакет (в туннельном режиме). Также ESP может содержать данные аутентификации, которые дописываются в конец пакета. ESP-пакет оборачивается в новый IP-пакет и отсылается адресату. При работе ESP в туннельном режиме принимающая сторона дешифрует исходный пакет, извлекает из него исходный IP-адрес и отсылает вложенный пакет по этому адресу уже во внутренней сети.

**IKE** (Internet Key Exchange) – протокол обмена ключами, построенный на основе протоколов ISAKMP и Oakley. IKE поддерживает набор различных примитивных функций для использования в протоколах, среди которых - хэш-функция и псевдослучайная функция (PRF). IKE используется для установления соединений IPSec SA.

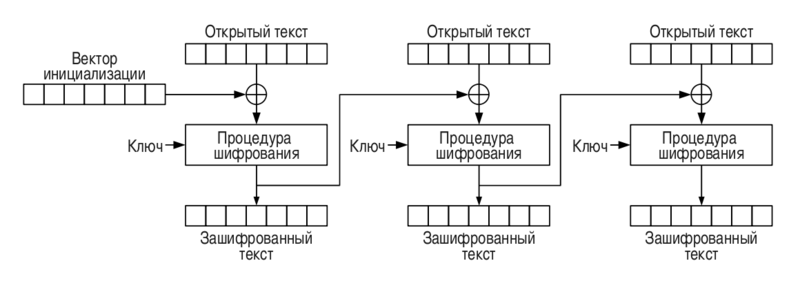
**29. Защищенные сетевые протоколы. Протокол SSL.**

**SSL** ( *Secure Sockets Layer* — уровень защищённых сокетов) — [криптографический протокол](http://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB), который обеспечивает установление безопасного соединения между клиентом и сервером. Протокол обеспечивает конфиденциальность обмена данными между клиентом и сервером, использующими TCP/IP, причём для шифрования используется асимметричный алгоритм с [открытым ключом](http://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81_%D0%BE%D1%82%D0%BA%D1%80%D1%8B%D1%82%D1%8B%D0%BC_%D0%BA%D0%BB%D1%8E%D1%87%D0%BE%D0%BC).

Протокол SSL 1.0/TLS 3.0 позволяет использовать шифрование симметричным ключом, используя либо блочные, либо потоковые шифры. На практике обычно используется блочные шифры. Принцип работы блочного шифра заключается в отображении блоков открытого текста в зашифрованные блоки того же размера. Проще всего представить блочный шифр в виде гигантской таблицы, содержащей 2^128 записей, каждая из которых содержит блок текста М и соответствующий ему зашифрованный блок С. Соответственно, для каждого ключа шифрования будет отдельная такая таблица. Далее мы будем обозначать шифрование в виде функции:   
C = E(Key, M), где M — исходные данные, Key — ключ шифрования, а C — полученные зашифрованные данные. Блоки имеют небольшой размер (как правило, 16 байт). Можно разбить сообщение на блоки одинаковой длины (те же самые 16 байт) и зашифровать каждый блок в отдельности. Такой подход называется режимом простой замены (ECB, Electronic codebook), но не используется, так как если мы будем шифровать два одинаковых по содержимому блока, то в результате и на выходе получим два одинаковых зашифрованных блока. Это влечет за собой проблему сохранения статистических особенностей исходного текста. Для избегания такого эффекта был разработан режим сцепления блоков шифротекста (СВС, Cipher-block chaining), в котором каждый следующий блок открытого текста XOR'ится с предыдущим результатом шифрования: Ci = E(Key, Mi xor Ci-1)

Во время шифрования первого блока исходный текст XOR’ится некоторым вектором инициализации (Initialization Vector, IV), который заменяет результат предыдущего шифрования. Однако эта теория описывает ситуацию для одного большого объекта, такого, например, как файл, который легко разбивается на блоки. В свою очередь, SSL/TLS является криптографическим протоколом — ему необходимо шифровать не отдельный файл, а серию пакетов. SSL/TLS-соединение может быть использовано для отправки серии HTTPS-запросов, каждый из которых может быть разбит на один или более пакетов, которые, в свою очередь, могут быть отправлены в течение как нескольких секунд, так и нескольких минут. В данной ситуации есть два способа использовать режим CBC:

1. обрабатывать каждое сообщение как отдельный объект, генерировать новый вектор инициализации и шифровать по описанной схеме.
2. обрабатывать все сообщения как будто они объединены в один большой объект, сохраняя CBC-режим между ними. В качестве вектора инициализации для сообщения n используется последний блок шифрования предыдущего сообщения (n-1).

Протокол SSL 3.0/TLS 1.0 использует второй вариант.  


30. Защищенные сетевые протоколы. Протокол SET.

Целью SET является обеспечение необходимого уровня безопасности для платежного механизма, в котором участвует три или более субъектов. При этом предполагается, что транзакция реализуется через Интернет. На базовом уровне SET осуществляет следующие функции:

1. Аутентификация. Все участники кредитных операций идентифицируются с помощью электронных подписей. Это касается клиента-покупателя, продавца, банкира, выдавшего кредитную карточку, и банкира продавца.
2. Конфиденциальность. Все операции производятся в зашифрованном виде.
3. Целостность сообщений. Информация не может быть подвергнута модификации по дороге в противном случае это будет сразу известно.
4. Подсоединение. SET позволяет подключить к базовому сообщению дополнительный текст и послать его одному из партнеров.
5. Безопасность. Протокол должен обеспечить максимально возможную безопасность операции, достижимую в имеющихся условиях.
6. Совместимость. Должна быть предусмотрена совместимость с любыми программными продуктами и с любыми сервис-провайдерами.
7. Независимость от транспортного протокола. Безопасность операций не должна зависеть от уровня безопасности транспортного протокола. Такие гарантии особенно важны, так как протокол SET ориентирован для работы в Интернет.

На более высоком уровне протокол SET поддерживает все возможности, предоставляемые современными кредитными карточками:

1. Регистрацию держателя карточки
2. Регистрацию продавца
3. Запрос покупки
4. Авторизацию платежа
5. Перевод денег
6. Кредитные операции
7. Возврат денег
8. Отмену кредита
9. Дебитные операции

Протокол работает с четырьмя субъектами: *владельцем кредитной карточки, банком, эту карточку выпустившим (issuer - эмитент), продавцом (merchant) и банком, где помещен счет продавца (acquirer)*. Помимо этих функциональных субъектов в процессе обычно (опционно) участвует центры сертификации, в задачу которых входит подтверждение подлинности предъявляемых параметров аутентификации, причем в случае крупных сделок с этими центрами должны взаимодействовать все участники. Основной целью сертификатов является подтверждение того, что присланный общедоступный ключ прибыл от настоящего отправителя, а не от самозванца.