Лекция 6.

Уважаемые студенты, я рад вас приветствовать на шестой лекции по компьютерным сетям. На прошлом уроке мы разобрали такие технологии как NAT и VPN, а на семинаре познакомились с шифрованным VPN. А сегодня давайте более подробно поговорим про само шифрование, разберем что такое сертификат и как работают протоколы HTTP и HTTPS. А также ознакомимся с протоколом DNS.

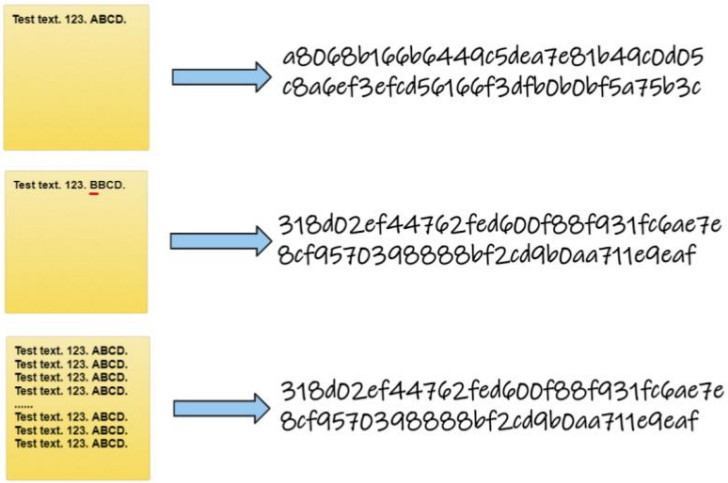
Итак, давайте для начала рассмотрим основные проблемы безопасности при передачи данных в сетях. Представим, что есть Алиса, которая хочет по каналу связи передать в банк Боба сообщение, что со счета А на счет В надо перевести 100 долларов “A->B:100$”. Если передавать такое сообщение по открытому каналу связи, где некая Ева может прослушивать его. То у нас нарушается конфиденциальность. То есть Ева может все это услышать и использовать эти данные в своих корыстных целях. Проблема конфиденциальности решается как раз таки шифрованием. Если Алиса зашифрует сообщение, то Ева получит какой-то непонятный и нечитаемый набор данных. Собственно шифрование - это некий алгоритм, который преобразует исходный текст с помощью специального ключа шифрования в нечитаемый набор данных, который можно расшифровать . Один из самых древних шифров - это шифр Цезаря. Ключом здесь играет роль некоторое число N, а сам алгоритм предполагает сдвиг по алфавиту на N

символов каждой буквы в сообщении. Если мы возьмем за N=3 и попробуем зашифровать наше сообщение от Алисы, то нам надо букву A заменить на D, букву B на E. Если мы представим что наш алфавит состоит не только из букв но и из символов и цифр, то и их можно тоже заменить, например > на #, - на & и т.д. Тогда наше сообщение примет примерно следующий вид “D#&E*433^”. Такое сообщение Еве не прочитать. И проблема конфиденциальности решается
На самом деле шифр Цезаря ломается довольно легко, но мы сейчас опустим проблему криптографической стойкости, так как современные шифры довольно устойчивы ко взломам. Но для большей наглядности будем использовать простой шифр Цезаря, принцип от этого не меняется.

Проблема целостности данных.

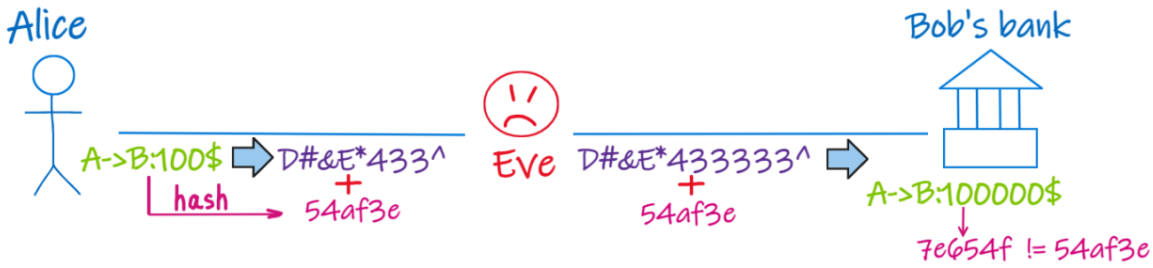
Но, что может произойти, если Алиса будет передавать сообщение через канал связи, который контролируется Евой? Ева может проанализировать проходящие через нее сообщения и увидеть один и тот же паттерн. И она может попробовать его поменять. Например увеличить количество “3” в нашем сообщении. Тогда банк получит что-то вроде “D#&E*433333^”. И вместо 100$ переведет 100000$.

Хэш-функции.



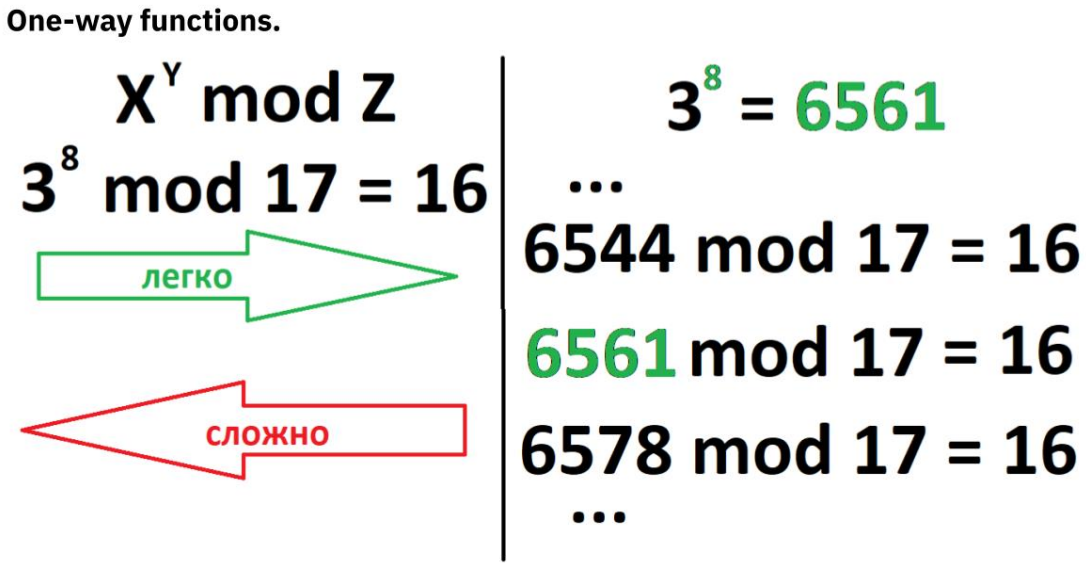
Эта проблема называется проблемой целостности данных. Чтобы её решить, используются известные нам хэш-функции, только более сложные, которые разрабатывают математики. Напомню: хэш функция - это функция, которая принимает на вход данные любой длины (хоть 1 байт, хоть 1 Гигабайт, не важно), а на выходе выдает значение фиксированной длины (например 256 бит). При этом из этого полученного значения получить исходные данные невозможно. Также если мы изменим хоть один бит в наших исходных данных, то значение хэша мы получим совершенно другое, непохожее на первое значение. Важно понимать, что эти функции ничего не шифруют, они как раз нужны для того, чтобы следить за тем, чтобы данные не были изменены, при этом алгоритмы функций стандартизированы, открыты и доступны всем.

Проверка целостности данных.



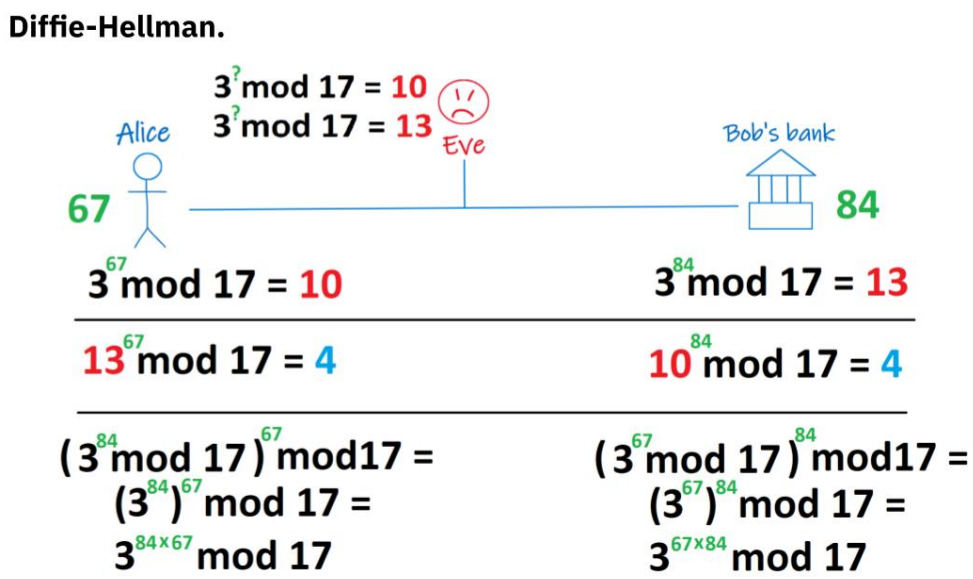
Итак, как хэширование поможет защитить целостность данных в нашем случае? Алиса должна перед отправкой данных, сделать хэш для них. Допустим получилось значение “54af3e” (это 16-чный вид записи хэша). После этого Алиса уже шифрует данные и отправляет в банк шифрованное сообщение плюс хэш сообщения: “D#&E*433^” + “54af3e”. Теперь если злоумышленник изменит шифрованное сообщение и добавит троек, то банк получит что-то вроде “D#&E*433333^” + “54af3e”. Расшифровав сообщение банк получит “A->B:100000$”, посчитает к нему хэш и получит например “7e654f” сравнит его с полученным. Но они уже будут разные. Поэтому банк поймет, что данные были искажены в процессе передачи и отбросит их. Так, с помощью хэш-функций, мы решаем проблему целостности данных. Злоумышленник может попробовать угадать хэш для измененного сообщения (которого он знает), но вероятность этого практически равна нулю.

Также существует проблема аутентификации, а именно: банку Боба нужно подтвердить Алисе, что именно он является банком Боба, а не Ева прикинулась им. Это решается благодаря механизму сертификатов, которые позволяют доверять участнику и убеждаться в его подлинности (а также устанавливать шифрованный канал связи между участниками в недоверенной среде).



Чтобы понять как работает система сертификатов давайте пойдем от простого к сложному и поговорим снова про шифрование. И сначала мы рассмотрим механизм, который позволяет нам вырабатывать общие ключи шифрования в недоверенной среде. Этот механизм был придуман учеными Diffie и Hellman. Раньше, до изобретения этого алгоритма, стояла проблема обмена ключами шифрования в недоверенной среде. Ведь если мы передадим ключ по открытому каналу связи мы его можем скомпрометировать, так как Ева всегда может перехватить его и использовать в дальнейшем для дешифровки сообщений. Так вот Diffie и Hellman для общей выработки ключа шифрования использовали механизм так называемых однонаправленных функций. Это функции, которые легко вычисляются в одну сторону и сложно в обратную. В нашем случае они использовали функцию X^Y mod Z. Это означает что мы берём X возводим в степень Y и делим на Z (где Z - простое число), получая остаток от деления. Этот остаток от деления и есть результат функции. Если мы возьмем наш пример со слайда и возведем 3^8=6561 и поделим на 17 мы получим остаток от деления 16. Важно понимать что помимо числа 6561 остаток от деления 16 нам даст также и число 6561+17 и число 6561+17+17 и т.д. Ровно как и 6561-17 , 6561-17-17 и т.д. То есть у нас есть бесконечное число чисел, которые дают нам 16 в остатке при делении на 17.

Это пример функции которые вычисляются легко в одну сторону и сложно в другую, ведь имея на руках число 16 нам довольно-таки сложно будет получить число 8. Для этого нам надо перебирать бесконечное число значений, которые дают при делении на 17 остаток 16, и для каждого такого значения нам ещё надо будет посчитать логарифм по основанию 3, что является нетривиальной задачей для больших чисел с точки зрения вычислений на компьютере. Если брать маленькие числа, то в принципе это задачи решаемые методом перебора, но если мы будем брать большие числа, например где 200 цифр, то такая задача может решаться очень долго - настолько долго, что взломщику становится нецелесообразно ее решать в принципе.



Так вот Diffie и Hellman взяли за основу эту функцию и придумали следующий алгоритм, рассмотрим его на примере:

1. Пусть Алиса и Боб вырабатывают каждый по-своему секретному числу В нашем случае это 67 и 84.
2. Далее они договариваются об общем основании - числе, которое будут возводить в степень, например 3. Пусть Ева перехватывает это основание.
3. Затем они договариваются об общем делителе, которое будет нам возвращать остаток от деления на него, например 17. Пусть Ева перехватывает этот делитель.

4. Далее Алиса берет и возводит основание 3 в степень своего секретного числа 67 и делит на 17, получая остаток от деления равный 10. Она передает это число 10 Бобу. Пусть Ева перехватывает его.
5. Далее Боб делает тоже самое, берет и возводит основание 3 в степень своего секретного числа 84 и делит на 17, получая остаток от деления равный 13. Он передает это число 13 Алисе. Пусть Ева перехватывает его.
6. Далее Алиса берет и возводит полученное от Боба число 13 в свою секретную степень 67 и делит на 17, получая остаток от деления 4.
7. Боб делает тоже самое и возводит полученное от Алисы число 10 в свою секретную степень 84 и делит на 17, получая остаток от деления тоже 4 !
8. Это число 4 и есть общий секретный ключ шифрования, выработанный в недоверенной среде. Ведь Еве будет трудно получить это число. Для этого ей надо сначала получить чье-либо секретное число, что является нереальной задачей для огромных чисел.
9. Почему Алиса и Боб будут получать одинаковые числа? Давайте в уравнении 13^67mod17=4 заменим 13 на (3^84mod17 - напомню, так вычислял его Боб). Получим выражение (3^84mod17)^67mod17, а Диффи и Хеллман показали, что mod 17 можно вынести за скобки, тогда получим:

   (3^84)^67mod17 = 3^(84*67)mod 17.

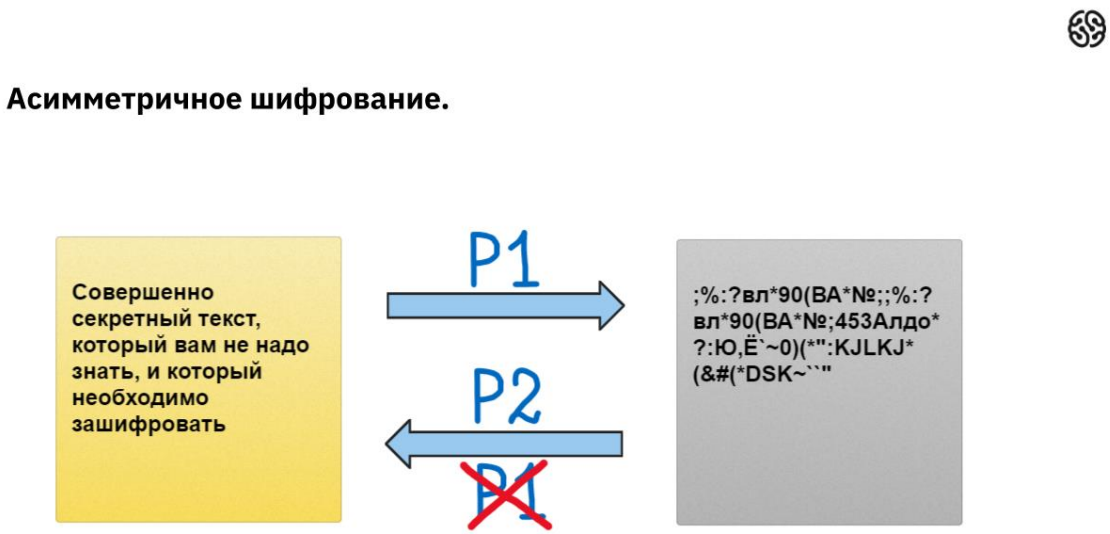
   Если мы сделаем тоже самое в уравнени Боба, то получим

   (3^67mod17)^84mod17 = 3^(67*84)mod 17

   А от перемены мест множителей произведение не меняется, следовательно мы имеем одинаковые числа.

Diffie-Hellman практически не используется сам по себе, так как он не предоставляет механизм аутентификации. В нашем примере Алиса не может с уверенностью заявить, что общается с Бобом. Ева может представиться от имени Боба и Алиса этого не заметит. Но Diffie-Hellman очень показателен в обучении, на его примере видна сама возможность существования математического механизма, который позволяет безопасно вырабатывать общие ключи шифрования для 2 участников в сети, где их могут прослушивать.

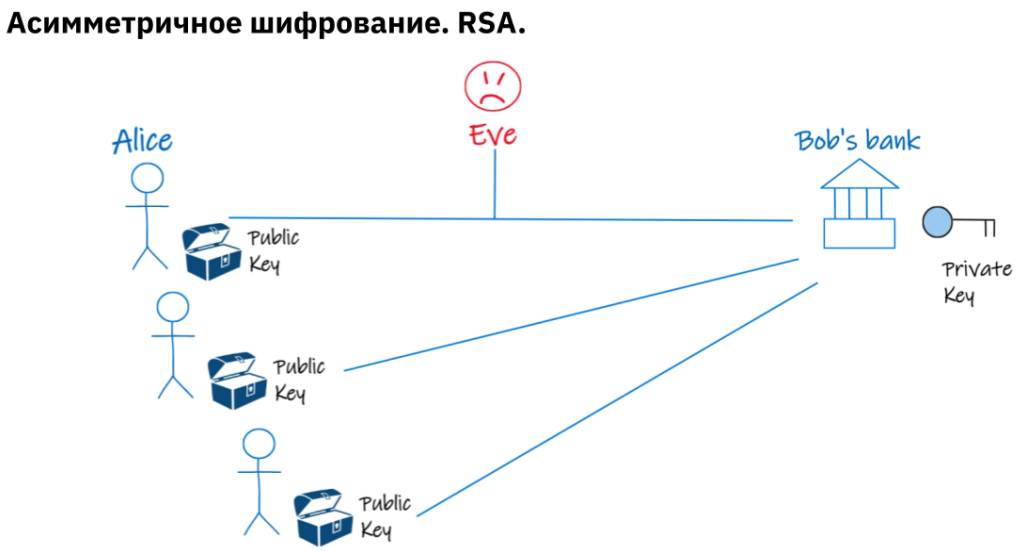
У Diffie-Hellman также есть еще один минус, ключ вырабатывается только для двух участников в сети, если это банк с миллионами клиентов, то для каждого клиента ему надо вырабатывать ключ и хранить его. Плюс, с течением времени, ключ лучше менять, потому что чем больше времени проходит с момента выработки ключа, тем больше вероятность что злоумышленник посередине может его всё-таки подобрать. Куда удобнее было бы иметь один ключ для дешифровки сообщений.



С этой этой проблемой нам позволяет бороться ассиметричное шифрование! До изобретения асимметричного шифрования мы шифровали и расшифровывали сообщения с помощью одного и того же ключа. А ассиметричное шифрование позволяет шифровать с помощью одного ключа, а расшифровывать с помощью другого.

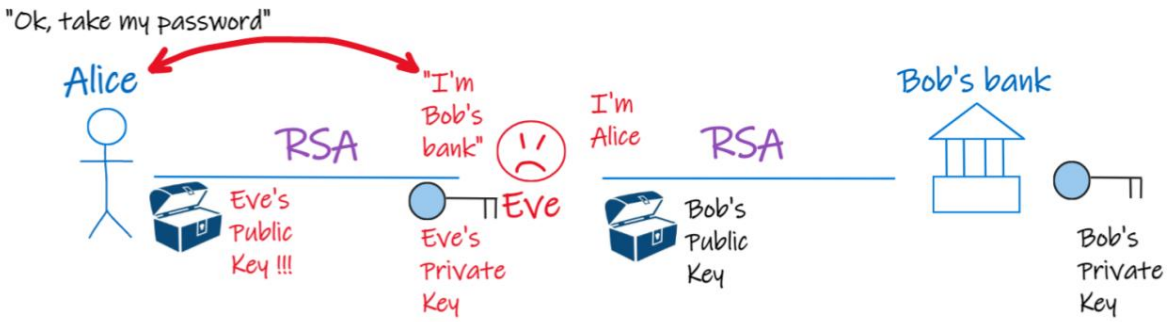
И британский ученый Клиффорд Кокс, а затем и трое американских ученых Ривест, Шамир и Адельман, независимо друг от друга придумали специальный алгоритм, похожий на Diffie-Hellman. Суть этого алгоритма следующая: У нас есть два огромных (примерно 500 знаков) простых числа p1 и p2 связанных между собой специальным образом, таким что если мы зашифруем наше сообщение m с помощью ключа p1, то расшифровать мы его сможем уже только с помощью ключа p2 (расшифровать с помощью того же p1 нельзя). И наоборот, если мы

зашифруем наше сообщение m с помощью ключа p2, то расшифровать мы его сможем уже только с помощью ключа p1 (расшифровать с помощью того же p2 нельзя). Этот алгоритм назвали RSA в честь американских ученых, потому что британская реализация была засекречена.



Чтобы понять как это работает, проще представить один ключ в виде сундука с самозащелкивающимся замком. А второй ключ собственно в виде ключа, который может открывать этот замок. Предположим Банк Боба создал такую пару. Он отсылает сундук с замком для Алисы, и она кладет в него сообщение и защелкивает замок. Все, даже сама Алиса не может открыть этот сундук. Только Боб, владелец ключа сможет его открыть. Более того, Боб может наклепать сколь угодно много таких сундуков и рассылать их для своих клиентов, а открывать (расшифровывать) только с помощью одного ключа.

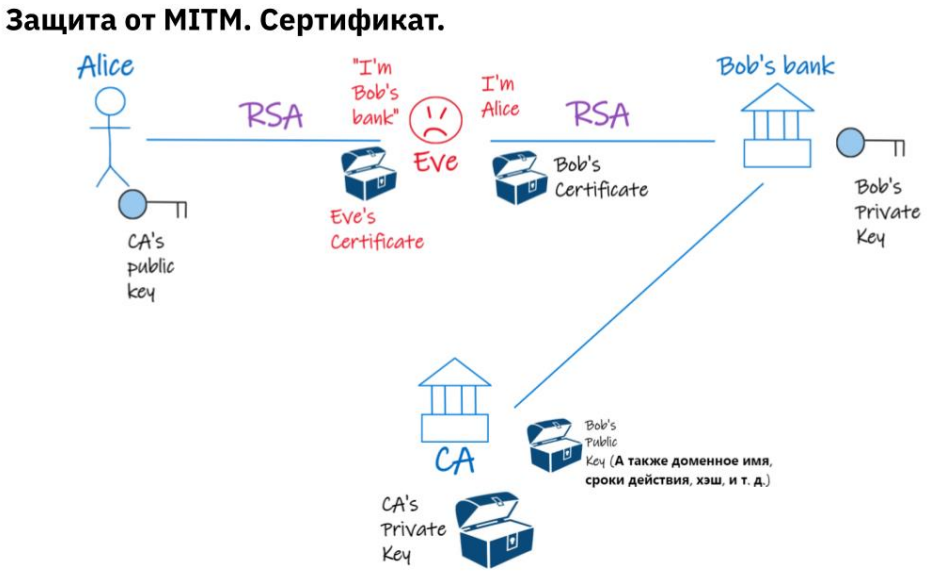
Man-In-The-Middle Attack. MITM.



Но что если Ева будет не просто прослушивать трафик, а встанет посередине и со стороны Алисы она представится Бобом, а со стороны Боба представится Алисой, и с обеих сторон она использует алгоритм RSA? Тогда она сможет обмануть систему и расшифровывать все сообщения и менять в них всё что угодно. Это так называемый класс атак Man-In-The-Middle (MITM). Чтобы защититься от этого нам надо заранее иметь публичный ключ Боба.

Защита от MITM.

Допустим наша ОС, например Windows, заранее позаботится и установится с публичным ключом Боба (то есть компания Microsoft встретится с представителем Банка Боба и внесет их публичный ключ в Windows, чтобы на каждом PC с Windows оказался этот ключ). Тогда, как только Ева нам отправит зашифрованное сообщение под видом Боба, Алиса не сможет его расшифровать ключом Боба и она поймет, что где-то что-то не чисто. Так и наоборот, Ева будет получать от Алисы зашифрованные сообщения Бобовским ключом, и не сможет их расшифровать, т.к. приватный ключ Боба только у него самого.



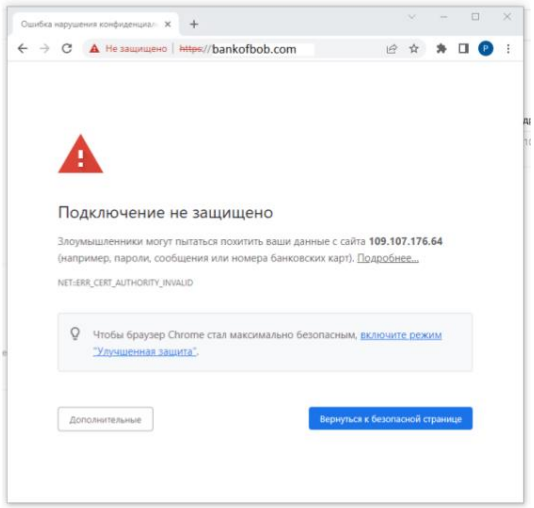
Но собирать публичные ключи всех компаний в мире задача нереальная. И для того, чтобы решить эту проблему существует третья сторона - CA (Certification Authority) или УЦ (Удостоверяющий Центр). Они занимаются тем, что тоже вырабатывают свою пару публичных и приватных ключей. Публичный ключ они передают в Microsoft. А затем они собирают от компаний публичные ключи этих компаний (то есть в нашем случае CA берет публичный ключ Боба) и шифруют своим приватным ключом (вместе с такой информацией как имя сайта банка Боба, хэш ключа, чтобы сохранить его целостность, сроки действия ключа и т. д.). Мы помним, что можно шифровать и приватным ключом, только расшифровать уже можно публичным! Поэтому на слайде приватный ключ CA - это сундук, который можно открыть только с помощью ключа, который CA раздает всем, причем один раз, то есть сундук ломается при

расшифровке. Так работает механизм “подписи”. Зашифровав эту информацию своим приватным ключом CA выпускает сертификат. Т.е. по сути сертификат - это зашифрованный приватным ключом УЦ открытый ключ Боба. Помимо этого удостоверяющий центр может запросить соответствующие документы и убедиться что сайт точно принадлежит компании, а также например попросить на сайте выложить специальную метку, чтобы убедиться, что обратившиеся к нему люди точно имеют доступ сайту и могут им управлять и администрировать его.

Удостоверяющий центр проделывает эту операцию для многих компаний. А свой публичный ключ они отдают вам в компьютер (через Microsoft, Linux, или в ваш браузер).

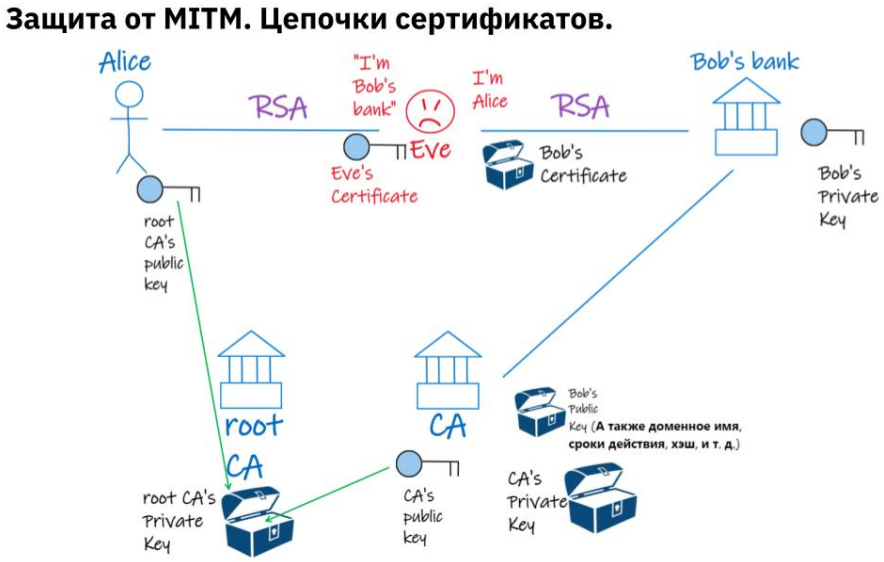
После чего как только вы связываетесь с сайтом Боба, Боб вам отдаёт свой сертификат, вы его расшифровываете публичным ключом УЦ и получаете публичный ключ Боба. Таким образом вы защищаетесь от подделки Евы. Ведь если она встанет посередине и подсунет свой левый сертификат (она не сможет получить его у УЦ никаким образом), то у вас попросту не будет ключа, с помощью которого вы сможете расшифровать сертификат Евы.

Защита от MITM. Неправильный Сертификат.



При этом в браузере перед посещением сайта будет выводиться огромное предупреждение, что вы имеете дело с сайтом на котором лежит

недействительный сертификат, подписанный недоверенным УЦ, и с просьбой не оставлять и не вводить логины и пароли на этом сайте, так как это может быть небезопасно.



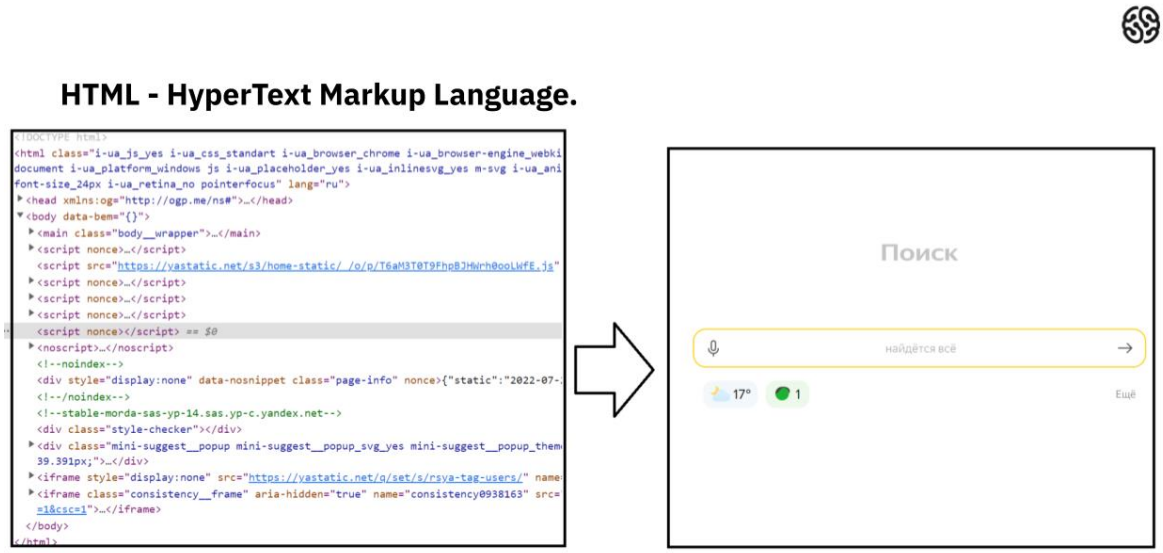
Конечно же один удостоверяющий центр на весь мир не будет справляться со всеми компаниями и банками в мире. Поэтому он может сделать сертификат для дочерних удостоверяющих центров. Получается цепочка сертификатов, которую можно расшифровать, имея сертификат корневого удостоверяющего центра.

Важно понимать, что корневые сертификаты основных удостоверяющих центров в мире всегда обычно предустановлены в ваших браузерах и операционных системах, на семинаре мы посмотрим где они храняться. То есть производители ваших браузеров и операционных систем (то есть компании Google, Microsoft и так далее) доверяют этим удостоверяющим центрам, так как сами провели с ними определенный комплекс мероприятий и убедились в чистоте их работы.

Таким образом, благодаря тому, что есть третья сторона, которая подтверждает подлинность публичных ключей различных компаний, мы можем избегать атак типа Man-In-The-Middle. До тех пор, пока вы не установите в корневые УЦ сертификат левого УЦ, который вам подсунул злоумышленник.

Инфраструктуру удостоверяющих центров вы можете сами развернуть для своих локальных ресурсов у себя. Тогда чтобы у ваших пользователей не выводилось это огромное пугающее сообщение, вы можете свой корневой сертификат подсунуть им в операционные системы. Но при этом, еще раз, ни в коем случае не подсовывайте левые сертификаты, которым вы не доверяете.

Важно отметить, что асимметричное шифрование является нетривиальной задачей для CPU, поэтому после создания соединения с ассиметричным шифрованием, создается симметричный ключ шифрования, так называемый сессионный ключ, и уже на нем шифруются данные, которые потом передаются.

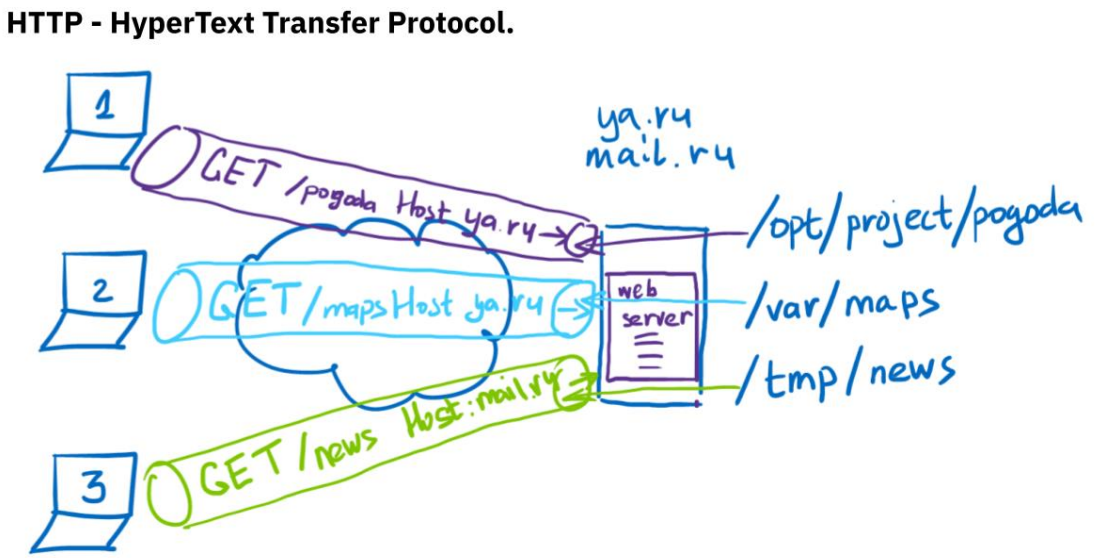


Что ж, разобравшись с асимметричным шифрованием и системой удостоверяющих центров, давайте поговорим теперь про протокол HTTP, который является основным протоколом для отображения веб-страничек в интернете.

Каждый сайт состоит из HTML разметки. HTML переводится как как Hyper Text Markup Language - язык разметки гипертекста. Когда-то в далёкие времена веб-странички представляли из себя простые сайты с гипертекстом внутри - то есть с текстом, в которых были ссылки на другие

тексты. Так вот HTML - это был стандарт, позволяющий оформлять этот текст и отображать в браузерах как сайт.

Для получения от сервера со стороны клиента определённых страниц, был придуман протокол передачи гипертекста - Hyper Text Transfer Protocol, или HTTP. Который собственно и позволяет клиенту запросить определенный контент с веб-сервера. Времена шли, странички усложнялись, но сам принцип не изменился до сих пор. Любой сложный контент прячется внутри HTML разметки, который мы получаем с помощью HTTP-запросов.



Давайте рассмотрим работу на примере. Предположим, что у нас есть сайт yandex.ru и mail.ru на сервере, и у нас в различных директориях лежит соответствующий контент:

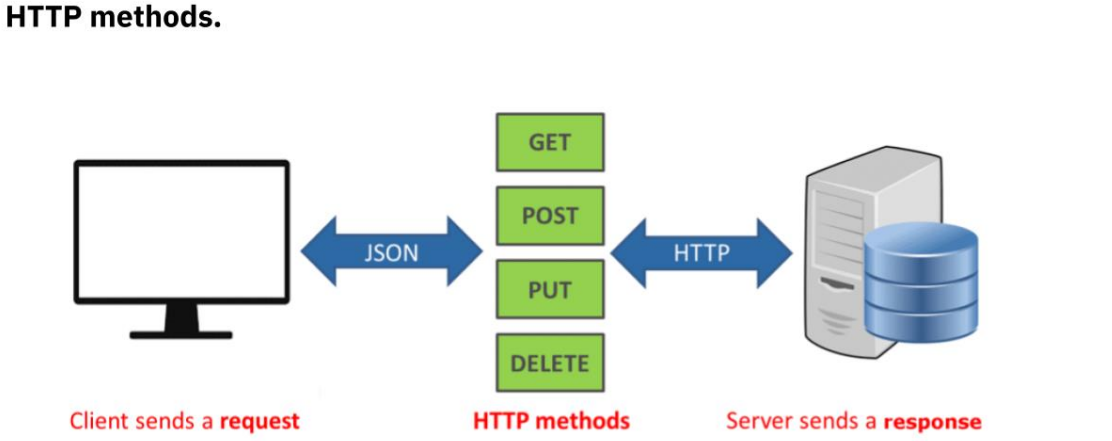
- для yandex.ru/pogoda код страницы лежит в /opt/project/pogoda/
- для yandex.ru/maps код страницы лежит в /var/maps/
- а для mail.ru/news код страницы лежит в /tmp/news/

Клиент №1 если захочет получить информацию о погоде, должен создать соответствующий запрос, который будет выглядеть примерно следующим образом: “GET /pogoda Host:yandex.ru”. Клиент №2 будет смотреть карты

и создаст запрос “GET /maps Host:yandex.ru”. Клиент №3 будет смотреть новости и создаст запрос “GET /news Host:mail.ru”.

Задачей веб-сервера, как программы, является создать Socket с каждым клиентом, разобрать эти запросы (то есть распарсить их) и выдать определенный контент каждому клиенту, согласно его запросу.

Можно сравнить веб-сервер с официантом в ресторане, который собирает заказы с разных столиков и относит их на кухню. На кухне заказы готовятся и официант разносит готовые заказы по столам, не путая заказы.



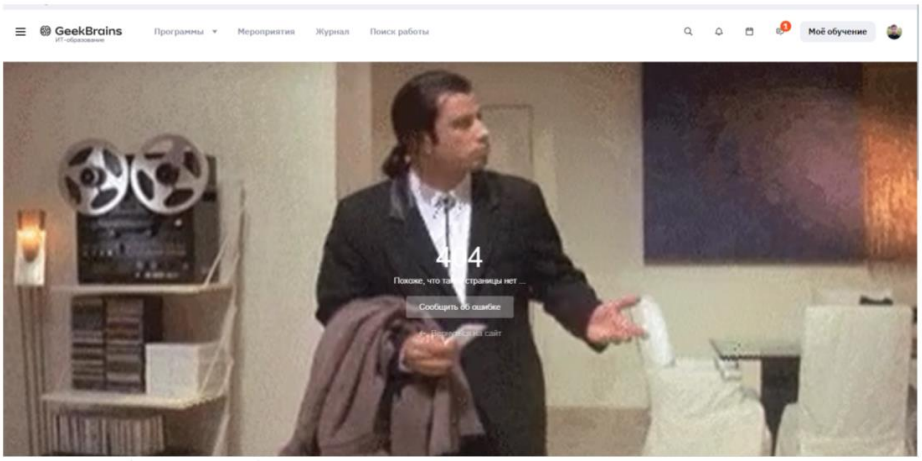
GET - это так называемый тип запроса, или метод запроса. HTTP позволяет не только получать данные, но и загружать данные на сервер с помощью других методов, например с помощью метода POST (знакомый нам с семинара с Wireshark’ом) мы можем передавать значения, или с помощью метода PUT мы можем передавать файлы на сервер, или с помощью метода DELETE, мы можем удалить какой-либо ресурс на сервере (если есть права) и так далее, есть еще некоторые другие методы, для более специфичных задач.

HTTP answer codes.

В ответ веб-сервер всегда сначала должен послать код ответа. Код состоит из трех цифр. И тип кода ответа зависит от 1 цифры:

- коды начинающиеся с “1” это информационные ответы.
- коды начинающиеся с “2” это коды успеха, например если страница существует и контент выдается, то код обычно равен 200.
- коды начинающиеся “3” означают перенаправление (redirect) на другую страничку.

HTTP 404.



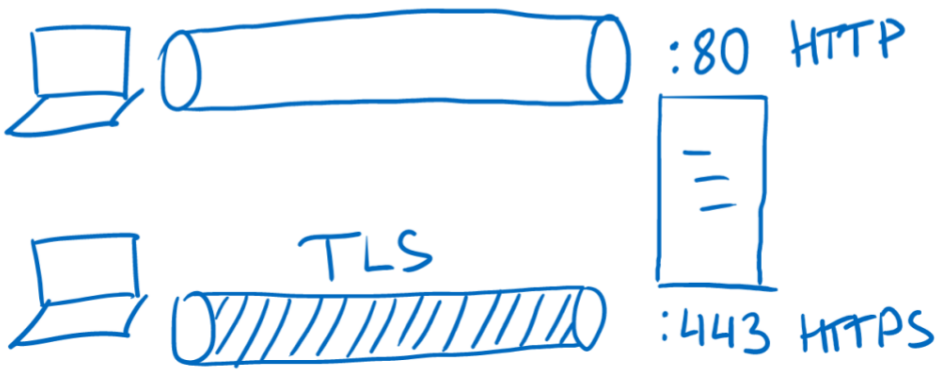
- коды начинающиеся с “4” это коды ошибки. Вы наверное иногда сталкивались с кодом 404, который говорит о том, что

запрашиваемая страница не найдена. Многие сайты делают на них прикольные картинки.

- коды начинающиеся с “5” это всякие ошибки веб-сервера.

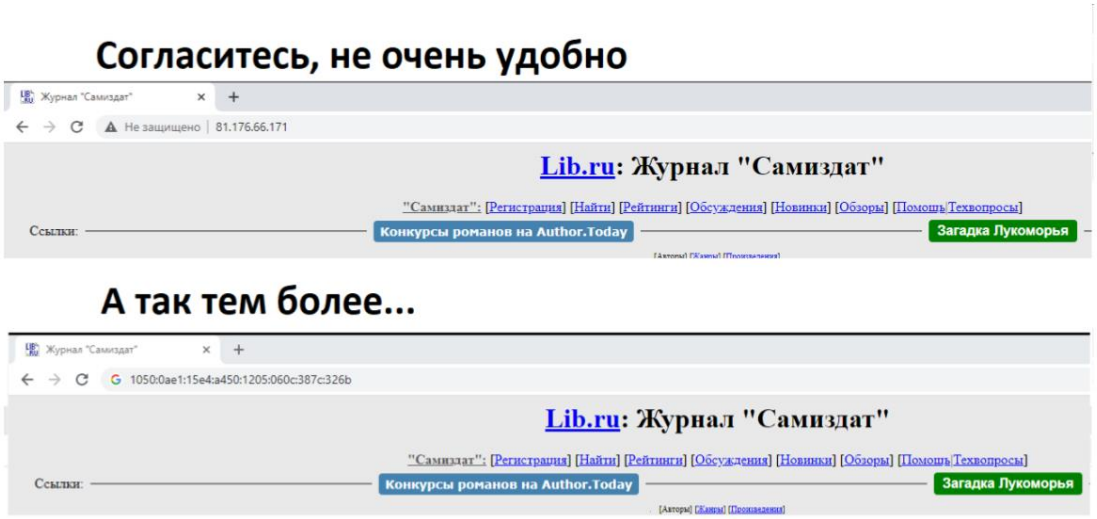
На семинаре мы попробуем создать свой собственный GET запрос и передать его в сокет какого-нибудь сайта. А также попробуем создать свой веб-сервер.

HTTP и HTTPS.



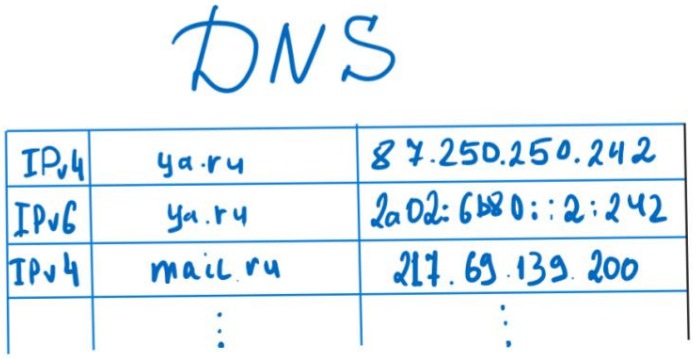
Обычно веб-сервера работают по умолчанию на 80 порту и на 443 порту. 80 порт как вам уже известно это не шифрованный Socket и все запросы, которые передаются в этом сокете, могут быть перехвачены и прочитаны злоумышленником. А 443 порт, это как раз-таки порт, по которому сначала создается шифрованное соединение с помощью сертификатов, а уже потом передаются запросы на веб-сервер в зашифрованном виде. Зашифрованный HTTP называется HTTPS, а процесс проверки сертификата и выработки сессионного ключа называется протокол TLS (Transport Layer Security)

DNS?



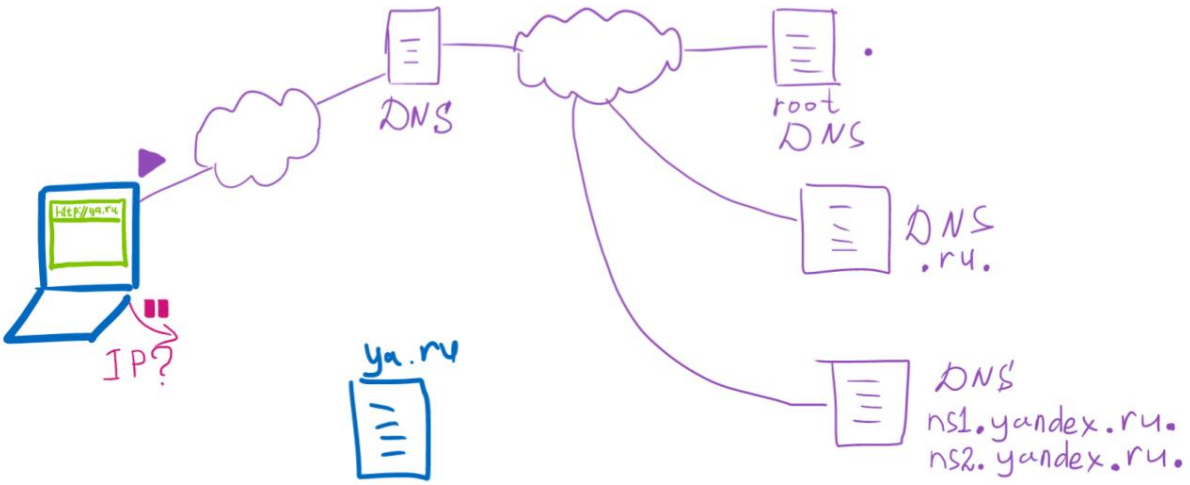
Теперь имея механизм сокетов и протокол HTTP, мы можем получать странички сайтов и отображать их у себя в браузерах. Но мы делали это всё время по конкретному IP адресу. Согласитесь, что если бы в интернете мы использовали чистые IP адреса - это было бы не очень удобно. Во-первых запомнить такие IP адреса было бы сложно, если бы мы например нашим бабушкам дедушкам говорили “заходи на 87.250.123.214”, вместо “Зайди на yandex.ru”, интернетом им было бы сложнее пользоваться. Да и про IPv6 можно было бы точно забыть. Во-вторых, если бы Яндекс сменил свой IP-адрес с 87.250.123.214 на какой-нибудь другой, например на 15.250.13.8 (по причине смены провайдера или датацентра), то он должен был бы сообщить об этом всему миру заранее, чтобы все знали, что теперь надо заходить по другому адресу.

Domain Name System



Поэтому чтобы ресурсы не привязываюсь к айпи адресом была придумана система DNS - domain name System. Эта система представляет из себя распределенную базу данных в которых есть ключ и значение. В виде ключа выступает доменное имя вида yandex.ru, в в значении лежит IP-адрес, привязанный к этому доменному имени. Эта база данных хранится на серверах DNS. которые обязаны выдавать значения по запрашиваемому ключу.

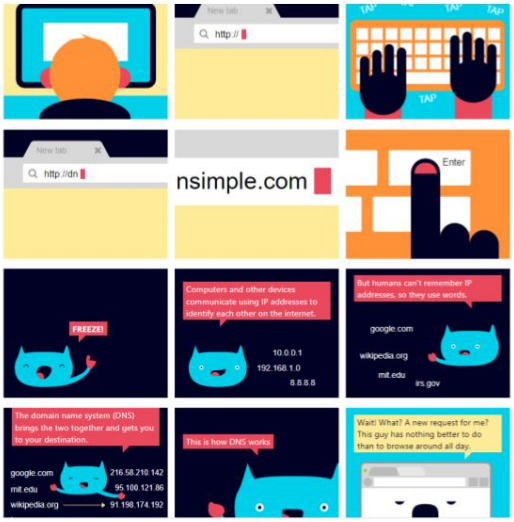
Как работает DNS.



Давайте рассмотрим как это работает.

1. Если к примеру вы обращаетесь к сайту yandex.ru и вбиваете его в браузер, а Ваш компьютер не знает какой IP адрес стоит за этим доменным именем, то прежде чем создавать сокет, он должен узнать IP адрес yandex.ru.
2. Ваш комп обращается к своему DNS серверу, который прописан у него в настройках. Например это может быть DNS-сервер который стоит у вашего провайдера. DNS запрос отправляется по протоколу UDP по 53 порту, в самом запросе собственно вопрос “А какой IP у сайта yandex.ru?”.
3. Если провайдерский DNS сервер не знает значения IP адреса для сайта yandex.ru, тон создаст запрос к вышестоящему DNS серверу или к корневому серверу.
4. Чаще всего вышестоящим DNS сервером является корневой сервер. Всего существуют 13 корневых DNS серверов в мире. Корневые DNS сервера хранят информацию о серверах различных зон 1ого уровня. В нашем запросе yandex.ru, зона первого уровня .ru
5. Наш DNS сервер получив информацию о DNS сервере .ru, пойдет спрашивать у него.
6. Если и он не знает, какой IP у yandex.ru, он выдаст список DNS серверов самого yandex.ru:
    - ns1.yandex.ru internet address = 213.180.193.1
    - ns2.yandex.ru internet address = 93.158.134.1
7. И тогда уже наш DNS сервер создаст запрос напрямую к этим серверам, они то уж точно подскажут, т.к. это их домен.
8. И после этого, получив IP адрес yandex.ru, мы его запомним (закэшируем) и выдадим нашему компу, который создаст соединение по этому адресу и уже после этого отправит HTTP запрос на сайт yandex.ru

Рекомендация - https://howdns.works/



Рекомендую также ознакомиться с сайтом, который в форме комикса разъясняет как работает DNS:

https://howdns.works/

Доменных уровней может быть много. За регистрацию доменов второго уровня отвечают специальные регистраторы, которые следят, чтобы записи не дублировались. Если вы владеете своим доменом второго уровня, вы можете создавать записи для доменов третьего уровня, делегируя вашим дочерним структурам например. И так далее. Более того, вы можете создать локально свою структуру DNS со своими адресами, при условии что ваши хосты из вашей сети будут обращаться в вашим DNS серверам. Например вы можете создать домен .mycooloffice и разместить там свои сервера с доменными именами:

- zabbix.mycooloffice
- 1c.mycooloffice
- games.mycooloffice
- и т. д.

Итак, на сегодня мы завершили знакомство с основными двумя протоколами HTTP(S) и DNS, которые позволяют нам пользоваться Интернетом, как пользуются им большинство людей - смотреть страницы

в браузерах. Но мы то теперь знаем, что на самом деле Интернет это больше чем просто странички в браузере. Мы понимаем что Интернет - это огромная мировая транспортная сеть, и теперь мы можем пользоваться этой транспортной сетью в своих целях, зная как она работает под капотом. За кадром осталось много еще интересных тем и протоколов. Но теперь вам не составит труда, зная базис, который мы рассмотрели, ознакомится с ними и разобраться, если это будет нужно.

Надеюсь эта серия уроков вам понравилась. А на сегодня все, и до встречи на семинаре! Всем пока!