Данный материал является фрагментом электронного учебника
по информационной безопасности и может обновляться.
При цитировании рекомендуется использовать ссылку:

Амелин Р. В. Информационная безопасность. Конспект лекций // Амелин Р.В. Лаборатория преподавателя [Электронный ресурс]. URL: rv-lab.ru (2017).

Лекция 5. Криптографические протоколы

5.1. Понятие криптографического протокола

Протокол – это последовательность шагов, которые предпринимают две или большее количество сторон для совместного решения некоторой задачи. Все шаги предпринимаются в порядке строгой очередности, и ни один из них не может быть сделан прежде, чем закончится предыдущий.

Криптографические протоколы используются для выполнения некоторых действий по обмену информацией в ситуации, когда цели участников могут быть нарушены злоумышленником (например, под угрозой оказывается конфиденциальность, целостность или подтверждаемость сообщений). Шаги криптографического протокола позволяют осуществить информационный обмен таким образом, что цели участников оказываются выполненными, а цели злоумышленника – нет.

Приведенная в предыдущем разделе последовательность шагов по созданию и проверке электронной цифровой подписи может служить примером криптографического протокола, в котором целью участников являются гарантия подлинности сообщения, а целью злоумышленника – его фальсификация.

Криптографические протоколы широко используют шифрование, хэширование, односторонние функции, генерацию случайных чисел.

5.2. Протоколы аутентификации

Если модель безопасности системы предусматривает управление доступом, в базе данных системы должны храниться идентификаторы зарегистрированных пользователей и назначенные им права доступа к ресурсам и функциям системы (подробно механизм доступа был рассмотрен в главе 3). Создание идентификаторов и назначение прав происходит в момент регистрации пользователя в системе, которая может выполняться администратором (например, в организации, при наделении полномочиями нового сотрудника) или самим пользователем (например, при регистрации на сайте – в этом случае, разумеется, пользователю предоставляется стандартный набор прав).

Для того, чтобы войти в систему, пользователь предъявляет ей свой идентификатор (например, вводит свой *логин* – имя пользователя). Система проверяет, хранится ли такой идентификатор в базе. Если нет, то пользователь считается нелегальным и ему отказывается в доступе к системе. Эта процедура называется *идентификацией*.

Аутентификация – процедура проверки подлинности, которая позволяет достоверно убедиться в том, что субъект, предъявивший системе идентификатор, является именно тем, кому этот идентификатор принадлежит.

В случае, если аутентификация проходит успешно, выполняется процедура *авторизации* – предоставление субъекту возможности доступа к ресурсам и функциям системы в соответствии с назначенными ему правами.

Протоколы аутентификации должны обеспечивать защиту от потенциального злоумышленника, цель которого – выдать себя за другого пользователя. В частности, протокол аутентификации не должен позволить проверяющему получить такую информацию о

стороне, доказывающей свою подлинность (аутентичность), которая впоследствии помогла бы ей выдать себя за нее.

В руководстве NTSC-TG-017, выпущенном Национальным центром компьютерной безопасности США, приводятся три основных вида аутентификации, в зависимости от того, каким именно образом пользователь (или другая система) доказывает, что он ~~не верблюд~~ именно тот, за кого себя выдает.

1. Authentication by Knowledge – на основе знания чего-либо. Например, пользователь вводит пароль или PIN-код, который (как предполагается) знает он один.
2. Authentication by Ownership – на основе владения чем-либо. Это может быть смарт-карта, электронный ключ и т. д.
3. Authentication by Characteristic – на основе биометрических характеристик: отпечатка пальцев, сетчатки глаза, голоса, почерка и т. д.

В соответствии с упомянутым руководством, выделяют протоколы аутентификации типа 1, типа 2 и типа 3, а также протоколы многофакторной аутентификации, когда для того, чтобы подтвердить свою подлинность, субъект должен продемонстрировать, например, и знание (тип 1) и обладание (тип 2). Так, чтобы снять деньги со своего счета через банкомат, клиент банка в процессе аутентификации использует и смарт-карту и PIN-код – это называется аутентификацией типа 12. Очевидно, что многофакторная аутентификация более надежна, чем однофакторная (злоумышленник должен не просто узнать PIN-код или похитить карту, а сделать и то и другое). А самой надежной будет аутентификация типа 123.

Наиболее простым протоколом аутентификации типа 1 является *аутентификация на основе открытого пароля*. Пользователь вводит логин и пароль, а система проверяет, содержится ли такая пара в ее базе. Если да, авторизация проходит успешно.

Такой протокол небезопасен по двум причинам. Во-первых, если осуществляется удаленный доступ к системе, то логин и пароль передаются по сети в открытом виде, а следовательно, немного постаравшись злоумышленник сможет их перехватить. Достаточно вспомнить, что маршрут прохождения пакетов через Интернет может быть каким угодно и практически кто угодно может анализировать содержимое этих пакетов на различных промежуточных пунктах. А технологии локальных сетей (например, Ethernet) построены по такому принципу, что сообщение, направленное определенному получателю, получат все узлы этой локальной сети. Они сверяют MAC-адрес получателя со своим собственным и игнорируют кадры, предназначенные для других узлов. Но никто не мешает злоумышленнику настроить свой (подключенный к этой локальной сети) компьютер таким образом, чтобы изучать весь циркулирующий в сети трафик. Во-вторых, поскольку логин и пароль в открытом виде хранятся в системе, злоумышленнику достаточно каким-либо образом получить доступ к системе хотя бы один раз, чтобы впоследствии успешно выдавать себя за любого пользователя.

Протокол аутентификации на основе открытого пароля и другие аналогичные протоколы простой аутентификации типа 1 не являются криптографическими. Примером простого и часто используемого криптографического протокола аутентификации является *аутентификация на основе хэшированного пароля*. В этом случае после того, как пользователь вводит логин и пароль, вычисляется хэш пароля и именно он передается по сети. В

системе хранятся пары вида «логин – хэш пароля» и для того, чтобы пользователь успешно авторизовался, необходимо, чтобы хранящийся в системе хэш совпал с хэшем введенного пароля. Поскольку хэш-функция является стойкой к коллизиям первого рода, противник, даже если перехватит хэш, не сможет вычислить соответствующий ему пароль. Такая аутентификация используется в большинстве современных операционных систем.

Слабость такого протокола в том, что он легко подвержен ***атаке повторного использования***. Она основана на перехвате сетевого трафика между пользователем и системой. Затем злоумышленник просто воспроизводит перехваченные пакеты для того, чтобы выдать себя за пользователя и получить его права доступа. В данном случае злоумышленнику не нужно отгадывать пароль – достаточно перехватить хэш, а затем просто предъявить его системе (конечно, придется слегка модифицировать клиентскую программу, чтобы она при входе не вычисляла хэш от хэша или написать свою, но это как раз тривиальная задача).

Хороший криптографический протокол строится исходя из предположения, что противник может перехватывать любые сообщения, передающиеся по сети, а также может блокировать их и посылать свои собственные. В хорошем протоколе аутентификации по сети не передается ничего, что позволило бы злоумышленнику получить дополнительную информацию, открывающую перед ним новые возможности.

Протоколы аутентификации типа 1, в которых пользователь подтверждает знание некоторого ключа, но сам ключ в ходе обмена информацией не раскрывается, называются *протоколами строгой аутентификации*.

Рассмотрим примеры таких протоколов. Стороной A будем называть пользователя или систему, который должен доказать свою подлинность (проверяемый), а стороной B – пользователя или систему, который должен убедиться в подлинности A (проверяющий).

Строгая односторонняя аутентификация на основе случайных чисел

Обе стороны знают общий ключ K и выбрали симметричный алгоритм шифрования.

1. A сообщает B о своем желании получить доступ. Например, пользователь вводит свой логин на рабочей станции. Логин передается по сети в открытом виде.
2. B генерирует случайное число *r* и отправляет его A.
3. A шифрует полученное число ключом K и отправляет результат B.
4. B расшифровывает сообщение ключом K и убеждается в том, что в результате получается число r.

Поскольку предполагается, что ключ K знают только A и B, сторона B посредством данного протокола убеждается в том, что имеет дело со стороной A. При этом даже если противник перехватывает передающиеся по сети сообщения, он не сможет воспользоваться ими, чтобы выдать себя за A, так как K в явном виде не передается, а каждый сеанс аутентификации использует новое случайное число.

Вычисления по алгоритму шифрования на шаге 3 может выполнять специальное программное обеспечение, установленное на рабочей станции, с которой пытается авторизоваться пользователь A, а может специальное аппаратное устройство. Такая программа или устройство называется *OTP-токеном* (аббревиатура OTP означает One-Time Password – одноразовый пароль). Сам ключ может храниться в OTP-токене, а может вводиться

пользователем с клавиатуры. Второй вариант предпочтительнее, поскольку запомнить сложный ключ, который невозможно подобрать, очень трудно. Самое надежное – это когда OTP-токен представляет собой смарт-карту или USB-устройство, ключ хранится внутри (в зашифрованном виде), а для того, чтобы выполнить третий шаг протокола, необходимо ввести PIN-код. При такой реализации аутентификация типа 1 превращается в аутентификацию типа 12.

Данный протокол может быть описан с помощью следующей наглядной схемы:

1. A → B: A
2. B → A: r
3. A → B: $[r]_K$

Строгая двусторонняя аутентификация на основе случайных чисел

В предыдущем протоколе сторона B убеждается в подлинности стороны A, но A не может быть уверена, что имеет дело именно с B. Протокол легко усовершенствовать таким образом, чтобы обе стороны убеждались в подлинности друг друга. Будем использовать следующую схему:

1. A → B: A
2. B → A: r_1
3. A → B: $[B \mid r_1 \mid r_2]_K$
4. B → A: $[r_1 \mid r_2]_K$

Здесь $[B \mid r_1 \mid r_2]_K$ означает сообщение, состоящее из имени B, случайного числа, полученного от B на втором шаге и другого случайного числа, сгенерированного стороной A, причем все это сообщение целиком зашифровано с помощью ключа K. Получив это сообщение, сторона B убеждается, что имеет дело со стороной A (поскольку только A может зашифровать только что присланное ей r_1 с использованием K). В свою очередь, на шаге 4 A убеждается в подлинности B, поскольку никто иной не мог бы расшифровать предыдущее сообщение, получить из него r_2, а затем подготовить другое сообщение, содержащее r_2 и зашифрованное тем же ключом K.

Заметим, что шаг 4 мог бы выглядеть и так:

4. B → A: r_2

Таким образом B показывает A, что знает ключ K, так как смог расшифровать число r_2. Однако данная версия протокола позволяет противнику, наблюдающему за сетью, получить пару «открытый текст – зашифрованный текст» (зашифрованный текст передавался на шаге 3, а соответствующий ему открытый текст можно составить из сообщений, передающихся на шагах 2 и 4. A с помощью пары «открытый текст – закрытый текст» уже можно попытаться осуществить атаку на сам алгоритм шифрования.

Аутентификация на основе асимметричного алгоритма

В следующем протоколе r – случайное число, H – хэш-функция, а *PK(A)* – открытый ключ стороны A. Хэш-функция используется на шаге 2, чтобы A могла убедиться в том, что сообщение не было искажено случайно или умышленно. Авторизация осуществляется

на том основании, что только закрытый ключ стороны A известен только ей (а значит никто больше не смог бы извлечь r из сообщения $[r, B]_{PK(A)}$

1. A → B: A
2. B → A: $H(r), B, [r, B]_{PK(A)}$
3. A → B: r

Приведенная схема протокола на самом деле является упрощением. Для того, чтобы такой протокол мог применяться в реальности, необходимо гарантировать, что открытый ключ *PK(A)* на самом деле принадлежит стороне A, то есть, в протоколе должен участвовать удостоверяющий центр.

В двух предыдущих протоколах, основанных на алгоритмах симметричного шифрования, нерешенным вопросом остается то, каким образом A и B разделяют один и тот же ключ K. Для этих целей можно использовать протокол обмена ключами.

5.3. Протоколы обмена ключами

Протокол обмена ключами – это такой протокол, с помощью которого знание некоторого секретного ключа (который может впоследствии использоваться для шифрования с помощью симметричного алгоритма) разделяется между двумя или более сторонами, причем противник, имеющий возможность перехватывать пересылаемые сообщения, не способен этот ключ получить.

Различают три вида протоколов обмена ключами: протоколы передачи ключей (уже сгенерированных), протоколы совместной выработки общего ключа и схемы предварительного распределения ключей.

Схема предварительного распределения ключей состоит из двух алгоритмов: распределения исходной ключевой информации и формирования ключа. С помощью первого алгоритма генерируется открытая часть исходной ключевой информации, которая размещается на общедоступном сервере и секретные части (для каждой стороны). Второй предназначен для вычисления действующего ключа для взаимодействия между абонентами по имеющимся у них секретной и общей открытой части исходной ключевой информации. Применяется для уменьшения объема хранимой и распределяемой секретной ключевой информации. Схема предварительного распределения ключей должна быть устойчивой, то есть учитывать возможность раскрытия части ключей при компрометации (утечке), обмане или сговоре части абонентов и гибкой – допускать возможность быстрого восстановления путем удаления скомпрометированных ключей и подключения новых абонентов[1].

Алгоритм Диффи-Хеллмана

Это один из самых известных протоколов обмена ключами. Он является весьма надежным для обмена ключами по каналу, исключающему возможность модификации (т.е. злоумышленник имеет возможность перехватывать данные, но не изменять их). Стойкость алгоритма проистекает из сложности дискретного логарифмирования: не суще-

[1] Б.А. Погорелов, А.В.Черемушкин, С.И.Чечета. Об определении основных криптографических понятий. http://www.ict.edu.ru/ft/002455/pogorelov.pdf

ствует эффективного алгоритма решения уравнения $a^x \bmod n = b$ (для простого n такое $x < n$ существует и единственно).

1. Участники обмена ключами выбирают два больших числа p и q (эти числа могут быть определены заранее и даже быть фиксированными, например, «зашитыми» в программное обеспечение).
2. Каждый участник генерирует случайное простое число (a и b соответственно).
3. Первый участник вычисляет значение $q^a \bmod p$ и пересылает его второму, а второй вычисляет $q^b \bmod p$ и передает первому.
4. Первый участник возводит полученное значение в степень x по модулю n, а второй участник – в степень y по модулю n. В результате оба участника получают одно и то же число $q^{ab} \bmod p$. Оно и берется в качестве секретного ключа.

Располагая значениями p, q, $q^a \bmod p$ и $q^b \bmod p$, но не зная x и y, злоумышленник не может вычислить ключ $q^{ab} \bmod p$.

Сокращенно схема обмена данными будет выглядеть так:

1. A → B: $p, q, q^a \bmod p$
2. B → A: $q^b \bmod p$
3. A, B: $q^{ab} \bmod p$

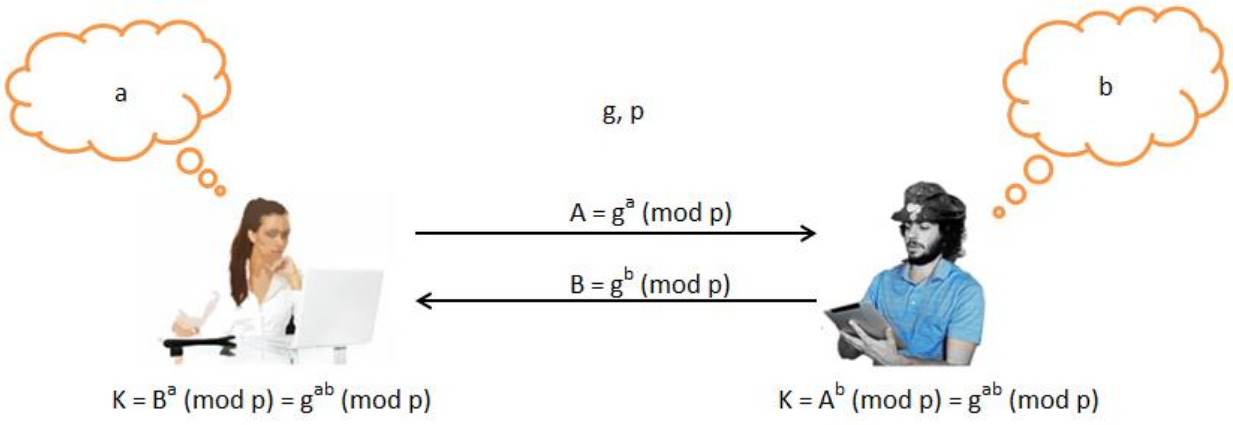

*Рис ***. Алгоритм Диффи-Хеллмана.*

Чтобы понять сильные и слабые стороны криптографических протоколов, важно понимать, что противник может использовать для атаки на протокол не только лобовой подход. Так, протокол Диффи-Хеллмана действительно не вскрывается «в лоб», однако легко поддается атаке «человек посередине» – в том случае, если противник может не только перехватывать, но и блокировать сообщения, а также отправлять свои собственные.

Атака «человек посередине» (Man in the Middle)

Идея атаки «человек посередине» заключается в том, что противник вклинивается в сеанс связи A и B, перехватывая все передаваемые между ними сообщения и модифицируя их таким образом, чтобы одновременно выдавать себя стороне B за сторону A, а стороне A – за строну B.

Например, в протоколе Диффи-Хеллмана противник перехватывает сообщение 1:

1. A → B: $p, q, q^a \bmod p$

далее он генерирует число z и подменяет перехваченное сообщение на:

1. A {E} → B: p, q, $q^c \bmod p$.

Сообщение

2. B → A: $q^b \bmod p$

подменяется сообщением:

2. B {E} → A: $q^c \bmod p$

В результате сторона A вычислит секретный ключ $q^{ac} \bmod p$, а сторона B – $q^{bc} \bmod p$. Эти же ключи будет знать злоумышленник, следовательно, он сможет изучать всю зашифрованную переписку. Чтобы остаться необнаруженным как можно дольше, он должен передавать B все сообщения, полученные от A и наоборот (предварительно расшифровав их одним ключом и снова зашифровав другим.

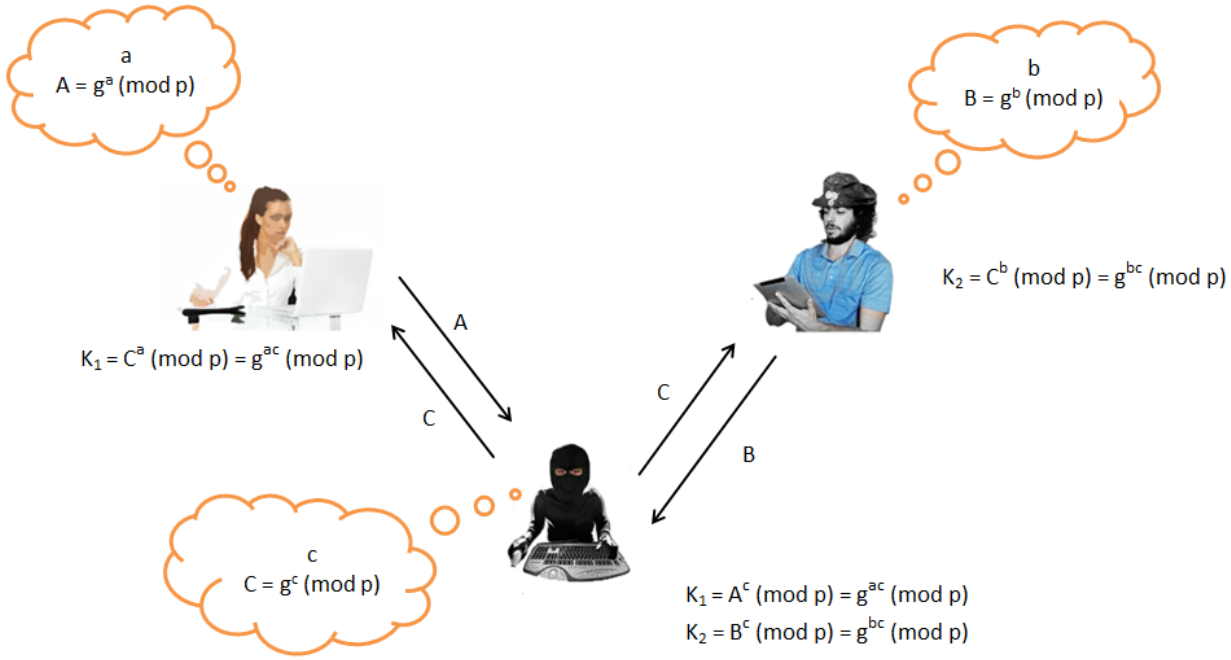

Рис. Атака «человек посередине»
на протокол обмена ключами по алгоритму Диффи-Хеллмана.

5.4. Протокол Kerberos

Kerberos – один из общедоступных протоколов аутентификации, созданный в Массачусетском технологическом институте (MIT). Пятая версия протокола получила статус стандарта IETF[2] и была реализована как основной протокол аутентификации в Windows 2000.

Участниками протокола Kerberos являются:

Клиент – пользователь или система, желающие получить доступ к ресурсам или функциям некоторого сервера (например, сервера электронной почты). В результате выполнения шагов протокола клиент должен подтвердить свою подлинность.

[2] IETF – Internet Engineering Task Force – открытое международное сообщество, которое занимается развитием протоколов и архитектуры Интернета.

Сервер – система, которая должна убедиться в подлинности клиента, прежде чем предоставлять ему свои услуги.

KDC (Key Distribution Center) – центр распределения ключей – сторонний посредник между клиентом и сервером, который «ручается» за подлинность клиента. В центре распределения ключей хранятся так называемые *долговременные ключи* каждого пользователя системы (клиента). Например, в Windows 2000 эти ключи генерируются на основе паролей, которые пользователи указывают при входе в систему.

Когда клиенту необходимо получить доступ к определенному серверу, он должен получить от KDC специальный ключ, который в терминологии Kerberos называется *мандатом* или *билетом* (в оригинале – ticket). Именно этот мандат и будет использоваться для связи с сервером – таким образом отсутствует необходимость регистрации каждого клиента на каждом сервере и управления большим количеством долговременных ключей.

Прежде чем получить мандат, клиент подтверждает свою подлинность **службе аутентификации (AS)**, которая является составной частью KDC. В результате клиент получит TGT – *мандат на получение мандатов*. Эта процедура будет выглядеть следующим образом:

1. Клиент → AS: ID клиента.

   Таким образом клиент сообщает, что хотел бы воспользоваться услугами некоторых серверов (пока неважно, каких), требующих аутентификации.

2. AS → Клиент: $[\text{сеансовый ключ}]_{PK(\text{Клиент})}$, $[\text{ID клиента, время, адрес клиента, сеансовый ключ}]_{PK(TGS)}$.

   Предварительно AS проверяет, что клиент с данным ID есть в базе. Первое сообщение, отправленное клиенту, представляет собой кратковременный сеансовый ключ, предназначенный для связи со **службой выдачи мандатов (TGS)**, также являющейся составной частью KDC. Разделение KDC на AS и TGS служит для повышения безопасности и удобства пользования системой. Дело в том, что зачастую клиенту во время одного сеанса работы может понадобиться связь с несколькими серверами. Для того, чтобы не проводить аутентификацию (и не требовать ввода пароля, на основе которого система пользователя воссоздает его долговременный ключ, ибо хранить его в самой системе небезопасно) при обращении к каждому серверу, клиент и получает сеансовый ключ, а затем использует его (а не долговременный ключ), чтобы подтвердить свою подлинность службе выдачи мандатов и получить уже от нее мандат для доступа к конкретному серверу.

   Важно отметить, что сеансовый ключ должны получить как клиент, так и TGS. AS могла бы отправить этот ключ TGS напрямую (т. е. один экземпляр ключа клиенту, второй – службе выдачи мандатов). Но такой подход неудобен (в частности потому, что сообщения по сети идут разными маршрутами и может оказаться, что клиент попытается связаться с TGS до того, как TGS успеет получить от AS сеансовый ключ). Поэтому передача сеансового ключа службе выдачи мандатов поручается самому клиенту. Для этого используется второе сообщение, полученное клиентом от AS на шаге 2, которое и представляет собой мандат на получение мандатов (TGT). Расшифровать его и каким-либо об-

разом изменить не может ни сам клиент, ни противник (если перехватит), поскольку TGT зашифрован с помощью открытого ключа TGS.

В Microsoft Windows эта часть протокола называется Authentication Service Exchange (сокращенно AS Exchange).

Далее клиент обращается к TGS, чтобы получить мандат на доступ к конкретному серверу. Последовательность шагов будет следующей:

1. Клиент → TGS: ID сервиса, TGT, $[\text{ID клиента, время}]_{\text{сеансовый ключ}}$
   Служба выдачи мандатов извлекает из TGT ID клиента, сеансовый ключ, время выдачи TGT, а затем убеждается, что данное сообщение пришло от того самого клиента, а не просто скопировано злоумышленником. Для этого необходимо расшифровать третью часть сообщения сеансовым ключом и убедиться, что ID клиента совпадает с тем, который содержится в TGT. В соответствии с протоколом аутентификации (приведенным выше), только сам клиент мог расшифровать сеансовый ключ, полученный от AS и воспользоваться им.
2. TGS → Клиент: $\text{сеансовый ключ для связи с сервером, [ID клиента, адрес клиента, время, сеансовый ключ для связи с сервером]}_{\text{РК(сервер)}}$
   Легко увидеть, что мандат на доступ к серверу (вторая часть сообщения) по структуре идентичен мандату на выдачу мандатов.

В Microsoft Windows эта часть протокола называется Ticket-Granting Service Exchange (сокращенно TGS Exchange).

Теперь клиенту остается обратиться к серверу, используя полученный мандат:

1. Клиент → Сервер: $\text{мандат, [ID клиента, время]}_{\text{сеансовый ключ для связи с сервером}}$
   Сервер убеждается в подлинности клиента точно также, как TGS на первом шаге предыдущей части протокола.
2. Сервер → Клиент: $[\text{время} + 1]_{\text{сеансовый ключ для связи с сервером}}$
   На данном шаге сервер подтверждает свою подлинность клиенту. Клиент убеждается в этом постольку, поскольку только обладатель закрытого ключа сервера мог расшифровать мандат, извлечь из него временную метку и сеансовый ключ для связи с сервером и создать соответствующее зашифрованное сообщение.

В Microsoft Windows эта часть протокола называется Client/Server Exchange (сокращенно CS Exchange).

Таким образом, двусторонняя аутентификация клиента на сервере прошла успешно. Клиент может доверять серверу и посылать ему запросы, а сервер соответственно – предоставлять клиенту требуемый сервис. Для обращения к другому серверу клиент должен заново выполнить вторую часть протокола, обратившись к TGS за новым мандатом.

Для успешной работы протокола Kerberos важно, чтобы часы всех участников были синхронизированы. Поскольку злоумышленник может перехватывать, а затем повторять от своего имени пересылаемые по сети сообщения, то в целях обеспечения дополнительной безопасности как служба выдачи мандатов TGS, так и сервер откажутся продолжать

протокол, если временная метка в TGT или соответственно мандате будет слишком старой (например, отставать от текущего времени TGS или сервера более чем на пять минут).

Протокол Kerberos – пример надежного криптографического протокола. Однако используя такие «надежные» схемы, никогда не следует забывать, что безопасность должна быть комплексной. Протокол выполняется участниками не вручную, а с помощью соответствующего программного обеспечения. Так, в 2000 году в программной реализации протокола, разработанной в самом MIT, были обнаружены ошибки переполнения буфера, позволяющие удаленному атакующему получить полный контроль над системой (привилегии root), где эта программа выполнялась.

5.5. Специфические протоколы

Протоколы аутентификации и протоколы обмена ключами – наиболее многочисленные классы криптографических протоколов. Однако существует ряд протоколов, предназначенных для решения других специфических задач, в частности:

Протоколы голосования. Предназначены для обеспечения проведения выборов, в ходе которых каждый участник может анонимно подать свой голос. При этом ни один участник не может проголосовать дважды; голосовать могут только зарегистрированные участники; каждый участник может проверить, правильно ли учтен его голос.

Протокол безопасного голосования основывается на использовании двух доверенных сторон – агентства по проверке голосующего T_1 и агентства для подведения итогов голосования T_2. Перед проведением голосования T_1 должно отослать T_2 список всех разрешенных идентификаторов участников голосования. Каждый голосующий (i) посылает T_1 некоторую идентифицирующую его информацию, после чего, если голосующему разрешено голосовать, то T_1 отправляет ему значение $E_1(i)$ – идентификатор голосующего и фиксирует факт участия в выборах. Далее голосующий вычисляет секретный идентификатор $E_2(i)$ и результат голосования $E_3(i)$ и посылает T_2 набор ($E_1(i)$, $E_2(i)$, $E_3(i)$). T_2 проверяет, существует ли $E_1(i)$ в списке разрешенных идентификаторов голосующих; если существует, то добавляет $E_2(i)$ к списку голосующих за $E_3(i)$. Преобразования E_1, E_2, и E_3 основаны на асимметричных алгоритмах или необратимых функциях[3].

Протоколы одновременной подписи. Цель участников: подписать некоторый документ таким образом, чтобы каждая сторона имела гарантию, что если она поставит свою подпись, это сделает и другая сторона. При этом участники могут быть удалены друг отдруга и подписывать документ при помощи ЭЦП.

Протоколы групповой подписи. Только члены группы могут подписывать сообщение, при этом получатель подписи может убедиться, что сообщение подписано членом группы, но не может определить – каким именно. Тем не менее, при споре подпись может быть раскрыта для определения личности подписавшего.

Самый простой протокол групповой подписи – с использованием доверенного арбитра. Арбитр генерирует большое количество пар открытых и закрытых ключей и раздает их членам группы (по m ключей каждому из n членов). Список открытых ключей пуб-

[3] См. Петров А.А. Компьютерная безопасность. Криптографические методы защиты. — М.: ДМК, 2000. С. 158.

ликуется. Чтобы подписать сообщение, член группы выбирает любой из своих закрытых ключей. Чтобы убедиться, что сообщение подписано одним из членов группы, достаточно проверить, что открытый ключ, с помощью которого проверяется ЭЦП, входит в набор открытых ключей группы. Наконец, проверка личности подписавшего определяется посредством обращения к арбитру. Единственная слабость этого протокола – сам арбитр, который может подделывать подписи всех участников.

Неоспариваемая подпись. Отличается от обычной цифровой подписи тем, что для ее проверки необходимо разрешение подписавшего.

Слепая подпись. Обладает свойствами электронной цифровой подписи, но при этом подписывающий не может ознакомиться с содержанием документа (пример: заверение завещания у нотариуса).

Протоколы разделения секрета. Позволяют разделить сообщение на несколько частей между членами группы таким образом, что каждый член группы не сможет извлечь никакой информации из своей части и только собравшись вместе участники группы смогут прочитать сообщение.

Наиболее распространенный протокол разделения секрета требует участия арбитра, который генерирует множество бессмысленных сообщений, которые дают исходное, будучи сложенными вместе операцией XOR. Например, чтобы разделить сообщение между двумя участниками арбитр генерирует случайное число R той же длины, что и исходное сообщение M, а затем вычисляет $R \oplus M = S$. Части R и S раздаются участникам. Чтобы получить исходное сообщение, выполняется операция $R \oplus S = M$.

5.6. Генерация случайных чисел

Большое значение для криптографии имеет генерация истинно случайных (непредсказуемых) чисел. Случайные числа используются при генерации сеасовых ключей (а также открытых и закрытых ключей асимметричных алгоритмов), во многих криптографических протоколах. Многие приложения имеют уязвимости связанные именно с предсказуемостью генерируемых паролей, сеансовых ключей и т.д.

Проблема заключается в том, что программное приложение не может генерировать абсолютно случайные числа в силу свойства детерминированности алгоритма. Поэтому для целей криптографии используются генераторы псевдослучайных чисел.

Генератор псевдослучайных чисел (ГПСЧ) – алгоритм, генерирующий последовательность чисел, элементы которой почти независимы друг от друга и подчиняются заданному распределению (обычно равномерному).

Генератор псевдослучайных чисел инициализируется некоторым ключом (начальным значением), после чего начинает выдавать последовательность, *зависящую от этого ключа* (т.е. если запустить генератор несколько раз с одним и тем же стартовым значением, получим на выходе одну и ту же последовательность). Как правило, очередное значение генератора вычисляется на основе одного или нескольких предыдущих. Следовательно, такая последовательность является *периодической* (как только в последовательности встретится цепочка n идущих подряд элементов, которая уже встречалась ранее, где n –

число последних элементов, на основе которых рассчитывается следующий, то n+1й, n+2й элемент и т.д.).

Криптографически стойкие генераторы псевдослучайных чисел должны обладать следующими свойствами:

1) ***Непредсказуемость***. Основное свойство генератора. Зная алгоритм генератора и все предыдущие элементы последовательности (но не зная секретного ключа, инициализирующего генератор) противник не может вычислить следующий элемент (если не пройден период ГПСЧ).

2) Достаточно большой ***период*** генератора. Дело в том, что последовательность, выдаваемая генератором псевдослучайных чисел является периодической, т.е. начиная с очередного элемента она начинает повторяться. Период должен быть достаточно большим, чтобы злоумышленник не мог найти

— Существенно большой период генератора (последовательность, выдаваемая ГПСЧ, является периодической).
— Последовательно выдаваемые элементы последовательности независимы друг от друга.
— Все элементы генерируемой последовательности являются одинаково «случайными» (т.е. можно говорить о равномерном распределении). К примеру, если ГПСЧ генерирует нам битовую последовательность, то количество нулей и единиц в этой последовательности должно быть примерно одинаковым.
— Непредсказуемость.

Рассмотрим примеры ГПСЧ:

1. Линейный конгруэнтный генератор. Выбираются три константы $m > 0$ (модуль), a (множитель) и c (приращение), такие что $0 \le a < m$, $0 \le c < m$. Выбирается начальное значение x_0. Значения элементов последовательности вычисляются по формуле:

$$x_{n+1} = (ax_n + c) \bmod m$$

При этом число m должно быть очень большим, для a предпочтительный диапазон $0.01 \le a < 0.99m$, а c не должно иметь общего множителя с m. Генератор следует использовать для получения не более, чем m/1000 элементов последовательности, далее их случайность уменьшается.

Линейный конгруэнтный генератор никогда не следует использовать в криптографии.

2. Смешанный квадратичный генератор. Применяется для генерации битовой последовательности b_1, b_2, ... по следующим соотношениям:

$$x_{n+1} = x_n^2 \bmod m,\ b_n = x_n \cdot z \bmod 2,$$

где произведение $x \cdot z$ – скалярное произведение чисел x и z, представленных в двоичной форме, т.е. $x \cdot z = (x_r x_{r-1} \ldots x_0) \cdot (z_r z_{r-1} \ldots z_0) = x_r z_r + x_{r-1} z_{r-1} + x_0 z_0$. Кроме того m должно представлять собой произведение двух больших простых чисел вида $4k + 3$, а x_0 взаимно просто с m.

В мощных криптосистемах военного применения используются действительно случайные генераторы чисел, основанные на физических процессах. Они представляют собой платы, либо внешние устройства, подключаемые к ЭВМ через порт ввода-вывода. Два ос-

новных источника белого Гауссовского шума – высокоточное измерение тепловых флуктуаций и запись радиоэфира на частоте, свободной от радиовещания.