Лекція 15. Технологія аутентифікації Kerberos; інфраструктура відкритого колюча X.509.

Kerberos

Kerberos является аутентификационным сервисом, разработанным как часть проекта Athena в MIT. В настоящее время Kerberos реализован также в Windows 2000. Kerberos решает следующую задачу: предположим, что существует открытое распределенное окружение, в котором пользователи, работающие за своими компьютерами, хотят получить доступ к распределенным в сети серверам. Серверы должны иметь возможность предоставлять доступ только авторизованным пользователям, т.е. аутентифицировать запросы на предоставление тех или иных услуг. В распределенном окружении рабочая станция не может быть доверенной системой, корректно идентифицирующей своих пользователей для доступа к сетевым серверам. В частности, существуют следующие угрозы:

1. Пользователь может получить физический доступ к какой-либо рабочей станции и попытаться войти под чужим именем.
2. Пользователь может изменить сетевой адрес своей рабочей станции, чтобы запросы, посылаемые с неизвестной рабочей станции, приходили с известной рабочей станции.
3. Пользователь может просматривать трафик и использовать replay-атаку для получения доступа на сервер или разрыва соединения законных пользователей.

Во всех этих случаях неавторизованный пользователь может получить доступ к сервисам и данным, не имея на то права. Для того чтобы не встраивать тщательно разработанные протоколы аутентификации на каждый сервер, Kerberos создает централизованный аутентифицирующий сервер, в чьи функции входит аутентификация пользователей для серверов и серверов для пользователей. В отличие от большинства схем аутентификации, Kerberos применяет исключительно симметричное шифрование, не используя шифрование с открытым ключом.

Существует две версии Kerberos. Наиболее широко используется версия 4. Версия 5, в которой исправлены некоторые недостатки версии 4, описана в RFC 1510.

Причины создания Kerberos

Если пользователи используют не соединенные в сеть компьютеры, то пользовательские и системные ресурсы и файлы можно уберечь от злоумышленников, обеспечив физическую защиту каждого компьютера. Когда пользователи обслуживаются централизованной системой разделения времени, безопасность должна обеспечивать именно она. Операционная система может проводить политику управления доступом на основе идентификатора пользователя и поддерживать строго определенную процедуру входа для идентификации и аутентификации пользователя.

В условиях сетевого взаимодействия подобные сценарии неприемлемы. Наиболее общим случаем является распределенная архитектура, состоящая из рабочих станций пользователя (клиентов) и распределенных серверов. В подобном окружении могут использоваться три подхода к обеспечению безопасности:

1. Аутентификация пользователя на своей рабочей станции.
2. Аутентификация пользователя на каждом сервере, к которому он должен иметь доступ.
3. Аутентификация пользователя на специальном сервере, который выдает соответствующий билет (Ticket) для доступа на сервер.

В небольших закрытых окружениях, в которых все системы работают в единственной организации, первой и, возможно, второй стратегии оказывается достаточно. Но в более открытом окружении, где поддерживаются сетевые соединения между компьютерами, для защиты информации и ресурсов пользователей необходим третий подход. Этот третий подход поддерживается Kerberos. Kerberos предполагает наличие распределенной архитектуры клиент/сервер и использует один или более серверов Kerberos для предоставления аутентификационных сервисов.

Kerberos должен удовлетворять следующим требованиям:

1. **Безопасность**: при просмотре трафика у оппонента не должно быть возможности получить необходимую информацию для того, чтобы сыграть роль законного пользователя.
2. **Надежность**: для всех сервисов, которые используют Kerberos для управления доступом, отсутствие сервера Kerberos означает отсутствие поддерживаемых сервисов. Следовательно, Kerberos должен иметь высокую надежность и реализовывать распределенную серверную архитектуру, в которой одна система может быть вспомогательной для другой.
3. **Прозрачность**: в идеале пользователь не должен знать, что имеет место аутентификация, кроме того случая, когда требуется ввести пароль.
4. **Масштабируемость**: система должна иметь возможность поддерживать большое число клиентов и серверов. Это требование означает, что Kerberos должен иметь модульную, распределенную архитектуру.

Для реализации этих требований Kerberos использует трехсторонний аутентификационный диалог, основанный на протоколе Нидхэма и Шредера. Такая система является доверенной в том случае, если клиенты и серверы доверяют Kerberos быть посредником при аутентификации. Этот аутентификационный диалог безопасен в той же степени, в какой безопасен сам сервер Kerberos.

Kerberos версии 4

Версия 4 Kerberos в качестве алгоритма симметричного шифрования использует DES. Рассматривая протокол в целом, трудно оценить необходимость многих содержащихся в нем элементов. Поэтому сначала рассмотрим простейший вариант протокола, затем будем добавлять отдельные элементы для ликвидации конкретных уязвимостей.

Простой аутентификационный протокол

В незащищенном сетевом окружении любой клиент может использовать любой сервер в качестве сервиса. В этом случае существует очевидный риск для системы безопасности. Оппонент может попытаться представиться другим клиентом и получить неавторизованные привилегии на сервере. Для того чтобы избежать этой опасности, сервер должен иметь возможность проверить идентификацию клиента, который запрашивает сервис. Практически не представляется возможным, чтобы каждый сервер выполнял эту задачу при соединении с каждым клиентом.

Альтернативой является использование аутентификационного сервера AS, который знает пароли всех пользователей и хранит их в специальной базе данных. Кроме того, AS разделяет уникальный секретный ключ с каждым сервером. Эти ключи распределяются физически или некоторым другим безопасным способом. Рассмотрим следующий протокол, который является скорее гипотетичеким:

1. $C \rightarrow AS: \quad ID_C \parallel P_C \parallel ID_S$
2. $AS \rightarrow C: \quad \text{Ticket}$
3. $C \rightarrow S: \quad ID_C \parallel \text{Ticket}$

$$\text{Ticket} = E_{Ks}\left[ID_C \parallel AD_C \parallel ID_S\right]$$

Где:

C – клиент;

AS – аутентификационный сервер;

S – сервер;

ID_C – идентификатор пользователя на C;

ID_S – идентификатор S;

P_C – пароль пользователя на C;

AD_C – сетевой адрес C;

K_S – секретный ключ шифрования, разделяемый AS и S.

В данном сценарии предполагается, что пользователь входит на рабочую станцию и хочет получить доступ к серверу S. Клиентский модуль C на пользовательской рабочей станции запрашивает пользовательский пароль и затем посылает сообщение AS, которое включает идентификатор пользователя, идентификатор сервера и пароль пользователя. AS проверяет в своей базе данных правильность пароля пользователя и то, что данному пользователю разрешен доступ к серверу S. Если обе проверки выполнены успешно, AS считает, что пользователь аутентифицирован, и должен теперь убедить сервер, что это так. Для того, чтобы это сделать, AS создает билет (ticket), который содержит идентификатор пользователя, сетевой адрес, с которого вошел пользователь, и идентификатор сервера. Этот билет шифруется с использованием секретного ключа, разделяемого AS и S. Он посылается C. Так как билет зашифрован, его не может изменить ни C, ни оппонент.

Имея данный билет, C может теперь обращаться к S за сервисом. Для этого он посылает серверу сообщение, содержащее идентификатор C и билет. S расшифровывает билет и проверяет, совпадают ли идентификатор пользователя в билете и незашифрованный идентификатор пользователя в сообщении. Если это соответствие выполняется, то сервер считает пользователя аутентифицированным и предоставляет соответствующий сервис.

Каждая часть сообщения (3) важна. Билет зашифрован для предотвращения изменения или подделки. Идентификатор сервера ID_S включается в билет, чтобы сервер мог убедиться, что он расшифровал билет корректно. ID_C включается в билет, чтобы определить, что данный билет послан от имени C. Наконец, AD_C служит для предотвращения следующей угрозы. Оппонент может перехватить билет, передаваемый в сообщении (2), затем использовать имя ID_C и передать сообщение в форме (3) с другой рабочей станции. Сервер получит законный билет, в котором совпадают идентификаторы пользователя, и предоставит доступ пользователю с другой рабочей станции. Для предотвращения подобной атаки AS включает в билет сетевой адрес, с которого приходит первоначальный запрос. Теперь билет действителен только в том случае, если он передан с той же самой рабочей станции, с которой первоначально запрашивался.

Более безопасный аутентификационный протокол

Хотя описанный сценарий и решает часть проблем аутентификации в открытых сетевых окружениях, многие проблемы все еще остаются. В частности, следует решить следующие две задачи. Во-первых, сделать так, чтобы пользователю приходилось вводить пароль минимальное количество раз. Пока предполагается, что каждый билет может использоваться только один раз. Если пользователь C хочет проверить свою почту на почтовом сервере, он должен предоставить пароль для получения билета на почтовый сервер. Если C хочет проверить почту несколько раз в течение дня, каждое обращение к почтовому серверу требует повторного ввода пароля. Эту процедуру можно усовершенствовать, разрешив переиспользовать билеты. При первой входной сессии рабочая станция может запомнить полученный билет сервера и использовать его от имени пользователя в дальнейшем при доступе к этому серверу.

Однако при такой схеме пользователю необходим новый билет для каждого нового сервера. Если пользователь хочет получить доступ к серверу печати, почтовому серверу, файловому серверу и т.д., то при первом доступе к каждому серверу будет требоваться ввод пароля.

Вторая проблема состоит в том, что ранее рассмотренный сценарий включает незашифрованную передачу пароля в первом сообщении. Оппонент может перехватить пароль и использовать любой доступный данному пользователю сервис.

Для решения этих проблем рассмотрим схему, которая не допускает незашифрованных паролей, и введем новый сервер, называемый сервером предоставления билетов (ticket-granting server – TGS). Этот сценарий состоит в следующем:

Один раз при входе пользователя:

1. $C \rightarrow AS:\ \ ID_C \parallel ID_{TGS}$
2. $AS \rightarrow C:\ \ E_{Kc}\left[\mathrm{Ticket}_{\mathrm{TGS}}\right]$

Один раз для каждого типа сервиса:

3. $C \rightarrow TGS:\ \ ID_C \parallel ID_S \parallel \mathrm{Ticket}_{\mathrm{TGS}}$
4. $TGS \rightarrow C:\ \ \mathrm{Ticket}_{\mathrm{S}}$

Один раз для каждого доступа к сервису:

5. $C \rightarrow S:\ \ ID_C \parallel \mathrm{Ticket}_{\mathrm{S}}$

$$\mathrm{Ticket}_{\mathrm{TGS}} = E_{Ktgs}\left[ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_1 \parallel LT_1\right]$$

$$\mathrm{Ticket}_{\mathrm{S}} = E_{Ks}\left[ID_C \parallel AD_C \parallel ID_S \parallel TS_2 \parallel LT_2\right]$$

Пользователь первым делом получает билет, гарантирующий билет, $\mathrm{Ticket}_{\mathrm{TGS}}$ от AS. Этот билет хранится в модуле клиента на рабочей станции пользователя. Сервер TGS выдает билеты пользователям, которые перед этим были аутентифицированы AS. Каждый раз, когда пользователю требуется новый сервис, клиентский модуль обращается к TGS, используя предоставленный AS билет, и TGS выдает билет для конкретного сервиса. Клиентский модуль хранит каждый гарантирующий сервис билет и использует его для аутентификации на сервере всякий раз, когда требуется конкретный сервис. Рассмотрим данную схему подробнее:

1. Клиентский модуль запрашивает билет, гарантирующий билет, посылая идентификатор пользователя к AS вместе с идентификатором TGS, который будет в дальнейшем использоваться для получения билета, гарантирующего сервис.
2. AS в ответ присылает билет, зашифрованный ключом, полученным из пароля пользователя. Когда этот ответ поступает в клиентский модуль, он просит пользователя ввести свой пароль, создает ключ и пытается расшифровать полученное сообщение. Если используется корректный пароль, то билет успешно извлекается.

Так как только законный пользователь должен знать пароль, только законный пользователь и может получить билет. Таким образом, пароль используется для получения доверительной грамоты от Kerberos без передачи пароля в незашифрованном виде. Сам билет включает идентификатор и сетевой адрес пользователя и идентификатор TGS. Это соответствует первому сценарию. Необходимо, чтобы такой билет мог использоваться клиентским модулем для запроса нескольких билетов, гарантирующих предоставление сервиса. Следовательно, билет, гарантирующий билет, с одной стороны, должен быть переиспользуемым. Но с другой стороны, необходимо добиться того, чтобы оппонент не мог перехватывать этот билет и использовать его. Рассмотрим следующий сценарий: оппонент перехватывает билет и ждет до тех пор, пока пользователь не завершит регистрацию на своей рабочей станции. Тогда оппонент пытается получить доступ к этой рабочей станции или сконфигурировать свою рабочую станцию с тем же сетевым адресом, что и у законного пользователя. После этого оппонент будет иметь возможность переиспользовать билет для обмана TGS. Чтобы этого не произошло, билет

включает отметку времени, определяющую дату и время, когда был получен билет (TS_1), и время жизни (LT_1), определяющую величину времени, в течение которого билет является действительным (например, 8 часов). Таким образом, теперь клиентский модуль имеет переиспользуемый билет, и нет необходимости требовать от пользователя ввода пароля для получения нового сервиса. В заключении заметим, что билет, гарантирующий билет, шифруется секретным ключом, известным только AS и TGS. Это предотвращает модификацию билета. Билет повторно зашифровывается ключом, основанным на пароле пользователя. Это гарантирует, что билет может быть восстановлен только законным пользователем, прошедшим аутентификацию.

Теперь, когда клиентский модуль имеет билет, гарантирующий билет, доступ к любому серверу можно получить, выполнив шаги (3) и (4):

3. Клиентский модуль запрашивает билет, гарантирующий сервис. Для этой цели клиентский модуль передает TGS сообщение, содержащее идентификатор пользователя, идентификатор требуемого сервиса и билет, гарантирующий билет.
4. TGS расшифровывает входящий билет и проверяет успешность дешифрования по наличию своего идентификатора. Также необходимо убедиться, что время жизни данного билета не истекло. Затем TGS сравнивает идентификатор пользователя и его сетевой адрес со значениями, полученными из билета. После этого TGS выдает билет, гарантирующий доступ к нужному сервису.

Билет, гарантирующий сервис, имеет ту же структуру, что и билет, гарантирующий билет. Этот билет также содержит отметку времени и время жизни. Если пользователь захочет получить доступ к тому же самому сервису позднее, клиентский модуль может просто использовать ранее полученный билет, гарантирующий сервис, и нет необходимости повторно запрашивать пароль пользователя. Заметим, что билет зашифрован с помощью секретного ключа E_{Ks}, известного только TGS и серверу, что предотвращает его изменение.

Наконец, с билетом, гарантирующим сервис, клиентский модуль может получить доступ к соответствующему сервису:

5. Клиентский модуль запрашивает доступ к сервису от имени пользователя. С этой целью клиентский модуль передает сообщение серверу, содержащее идентификатор пользователя и билет, гарантирующий сервис. Сервер аутентифицирует пользователя, используя содержимое билета.

Этот новый сценарий позволяет сделать так, чтобы за время работы пользователя пароль запрашивался только один раз, и обеспечивает защиту пароля пользователя.

Аутентификационный протокол версии 4

Хотя рассмотренный сценарий усиливает безопасность по сравнению с первым сценарием, остаются еще две проблемы. Первая проблема состоит в том, что время жизни связано с билетом, гарантирующем билет. Если это время жизни очень короткое (т.е. минуты), то пароль у пользователя будет запрашиваться повторно. Если время жизни большое (т.е. часы), то оппонент имеет больше возможностей для совершения различных replay-атак. Злоумышленник должен просматривать сеть и перехватить билет, гарантирующий билет, и затем ждать, когда законный пользователь выйдет. Затем оппонент может подделать сетевой адрес законного пользователя и послать сообщение шага (3) к TGS. Это откроет ему неограниченный доступ к ресурсам и файлам законного пользователя.

Аналогично, если оппонент перехватил билет, гарантирующий сервис, и использует его прежде, чем истечет время его действия, он имеет доступ к соответствующему сервису.

Таким образом, можно сформулировать следующее дополнительное требование. Сетевой сервис (т.е. TGS или прикладной сервис) должны иметь возможность убедиться в том, что билет использует тот, кто получил его.

Вторая проблема состоит в том, что должна быть возможность аутентификации самих серверов для пользователей. Без подобной аутентификации оппонент может изменить конфигурацию таким образом, чтобы сообщение к серверу перенаправлялось по другому адресу. Этот ложный сервер будет затем выполнять действия в качестве реального сервера, перехватывать любую информацию от пользователя или не предоставлять ему необходимый сервис.

Сначала рассмотрим проблему перехвата билетов, гарантирующих билеты, и необходимость гарантировать, что представленный билет является тем же самым билетом, который был выдан клиенту. Для решения этой проблемы AS должен безопасным способом обеспечить как клиентский модуль, так и TGS некоторой секретной информацией. После этого клиентский модуль может доказать TGS свою идентичность, предоставляя безопасным способом эту секретную информацию. Здесь может использоваться некий разделяемый секрет, который в дальнейшем будет применяться в качестве ключа шифрования. Этот разделяемый секрет называется ключом сессии.

Рассмотрим обмен сообщениями по протоколу Kerberos версии 4:

(а) Обмен службы аутентификации: получение билета, гарантирующего билет

1. $C \to AS:\ ID_C \parallel ID_{TGS} \parallel TS_1$ – клиентский модуль запрашивает билет, гарантирующий билет

TS_1 – отметка времени, позволяет AS проверить, синхронизированы ли часы клиента с часами AS.

2. $AS \to C:\ E_{KC}\left[K_{C,TGS} \parallel ID_{TGS} \parallel TS_2 \parallel LT_2 \parallel \mathrm{Ticket}_{\mathrm{TGS}}\right]$ – AS возвращает билет, гарантирующий билет

$\mathrm{Ticket}_{\mathrm{TGS}} = E_{Ktgs}\left[K_{C,TGS} \parallel ID_C \parallel AD_C \parallel ID_{TGS} \parallel TS_2 \parallel LT_2\right]$

K_C – ключ шифрования, основанный на пользовательском пароле, применение которого позволяет AS и клиентскому модулю аутентифицировать пользователя и защитить содержимое сообщения (2).

$K_{C,TGS}$ – ключ сессии, созданный AS для обеспечения безопасного обмена между клиентским модулем и TGS.

K_{tgs} – ключ, которым зашифрован билет и который известен только AS и TGS.

ID_{TGS} – подтверждение того, что данный билет предназначен для TGS.

AD_C – адрес пользователя, предотвращает использование билета с любой рабочей станции, кроме той, с которой он был первоначально получен.

TS_2 – время создания билета.

LT_2 – время жизни билета.

(б) Обмен службы выдачи билетов: получение билета на доступ к сервису

3. $C \to TGS:\ ID_S \parallel \mathrm{Ticket}_{\mathrm{TGS}} \parallel \mathrm{Authenticator}_{\mathrm{C}}$ – клиентский модуль запрашивает билет, гарантирующий сервис

$\mathrm{Ticket}_{\mathrm{TGS}}$ – гарантирует TGS, что данный пользователь аутентифицирован AS.

$\mathrm{Authenticator}_{\mathrm{C}}$ – создается клиентом для подтверждения законности ключа.

$\mathrm{Authenticator}_{\mathrm{C}} = E_{KC,TGS}\left[ID_C \parallel AD_C \parallel TS_3\right]$

TS_3 – время создания аутентификатора.

4. $TGS \to C:\ E_{KC,TGS}\left[K_{C,S} \parallel ID_S \parallel TS_4 \parallel \mathrm{Ticket}_{\mathrm{S}}\right]$ – TGS возвращает билет, гарантирующий сервис

$\mathrm{Ticket}_{\mathrm{S}} = E_{KTGS,S}\left[K_{C,S} \parallel ID_C \parallel AD_C \parallel ID_S \parallel TS_4 \parallel LT_4\right]$

$\mathrm{Ticket}_{\mathrm{S}}$ – билет, используемый клиентом для доступа к серверу S.

$K_{TGS,S}$ – ключ, разделяемый S и TGS.

$K_{C,S}$ – ключ сессии, который создается TGS для обеспечения безопасного обмена между клиентским модулем и сервером без необходимости разделения ими постоянного ключа.

ID_S – доказательство того, что этот ключ предназначен для сервера S.

TS_4 – время создания билета.

LT_4 – время жизни билета.

(в) Обмен аутентификации клиента/сервера: получение сервиса

5. $C \to S:\ \mathrm{Ticket}_{\mathrm{S}} \parallel \mathrm{Authenticator}_{\mathrm{C}}$ – клиент запрашивает сервис

$\mathrm{Authenticator}_{\mathrm{C}} = E_{KC,S}\left[ID_C \parallel AD_C \parallel TS_5\right]$

$\mathrm{Ticket}_{\mathrm{S}}$ – гарантирует серверу, что данный клиент аутентифицирован AS.

$\mathrm{Authenticator}_{\mathrm{C}}$ – создается клиентом для подтверждения законности ключа.

TS_5 – время создания аутентификатора.

6. $S \to C:\ E_{KC,S}\left[TS_5 + 1\right]$ – дополнительная аутентификация сервера для клиента

$TS_5 + 1$ – гарантирует C, что не было replay-атак.

Прежде всего, клиентский модуль посылает сообщение к AS с требованием доступа к TGS. AS отвечает сообщением, зашифрованным ключом, полученным из пароля пользователя, который

содержит билет. Зашифрованное сообщение также содержит ключ сессии $K_{C,TGS}$, где индексы определяют, что это ключ сессии между C и TGS. Таким образом, ключ сессии безопасно передан как C, так и TGS.

К первой фазе сценария добавлено несколько дополнительных элементов информации. Сообщение (1) включает отметку времени, так что AS знает, что сообщение своевременно. Сообщение (2) включает несколько элементов билета в форме, доступной C. Это необходимо C для подтверждения того, что данный билет предназначен для TGS и для определения момента истечения срока его действия.

Теперь, имея билет и ключ сессии, C может обратиться к TGS. Как и раньше, C посылает TGS сообщение, которое включает билет и идентификатор требуемого сервиса (сообщение (3)). Дополнительно C передает аутентификатор, который включает идентификатор и адрес пользователя C, а также отметку времени. В отличие от билета, который является переиспользуемым, аутентификатор применяется только один раз и не имеет времени жизни (LT). Теперь TGS расшифровывает билет с помощью ключа, который он разделяет с AS. Этот билет содержит ключ сессии $K_{C,TGS}$. В действительности билет устанавливает, что любой, кто использует $K_{C,TGS}$, должен быть C. TGS задействует ключ сессии для дешифрования аутентификатора. TGS может затем сравнить имя и адрес из аутентификатора с тем, которое содержится в билете, и с сетевым адресом входящего сообщения. Если все совпадает, то TGS может быть уверен, что отправитель билета является настоящим его собственником. В действительности аутентификатор устанавливает, что до времени TS_3 возможно использование $K_{C,TGS}$. Заметим, что билет не доказывает чью-либо идентичность, а является способом безопасного распределения ключей. Аутентификатор является доказательством идентификации клиента. Так как аутентификатор может быть использован только один раз, опасности, что оппонент украдет аутентификатор для пересылки его позднее, не существует.

Сообщение (4) от TGS имеет вид сообщения (2). Сообщение зашифровано ключом сообщения, разделяемым TGS и C, и включает ключ сессии, который разделяется C и сервером S, идентификатор S и отметку времени билета. Билет включает тот же самый ключ сессии.

C теперь имеет переиспользуемый билет, гарантирующий сервис S. Когда C представляет этот билет, как и в сообщении (5), он также посылает аутентификатор. Сервер может расшифровать билет, получить ключ сессии $K_{C,S}$ и расшифровать аутентификатор.

Если требуется взаимная аутентификация, сервер посылает сообщение (6), в котором возвращает значение отметки времени из аутентификатора, увеличенное на единицу и зашифрованное ключом сессии. C может расшифровать это сообщение для получения увеличенной отметки времени. Так как сообщение может быть расшифровано ключом сессии, C уверен, что оно может быть создано только S. Это гарантирует C, что replay-атаки не было.

Наконец, в завершение клиент и сервер разделяют секретный ключ. Этот ключ может быть использован для шифрования будущих сообщений между ними или для обмена новым случайным ключом сессии.

Области Kerberos

Полнофункциональное окружение Kerberos, состоящее из сервера Kerberos, некоторого числа клиентов и некоторого числа прикладных серверов, требует следующего:

1. Сервер Kerberos должен иметь в своей базе данных идентификаторы и пароли всех зарегистрированных пользователей, т.е. все пользователи регистрируются на сервере Kerberos.
2. Сервер Kerberos должен разделять секретный ключ с каждым сервером, т.е. все серверы регистрируются на сервере Kerberos.

Такое окружение называется областью (realm). Сети из клиентов и серверов в различных административных организациях обычно образовывают различные области. Однако пользователи из одной области могут нуждаться в доступе к серверам из других областей, и некоторые серверы готовы предоставлять сервисы пользователям из других областей, если эти пользователи будут аутентифицированы.

Kerberos предоставляет механизм для поддержки такой аутентификации между областями. Для двух областей, поддерживающих межобластную аутентификацию, добавлено следующее требование:

3. Сервер Kerberos для каждой из взаимодействующих областей разделяет секретный ключ с сервером Kerberos в другой области. Другими словами, два сервера Kerberos регистрируют друг друга.

Схема требует, чтобы сервер Kerberos в одной области доверял серверу Kerberos в другой области аутентифицировать своих пользователей. Более того, серверы во второй области также должны быть согласны доверять серверу Kerberos в первой области.

Если эти основополагающие условия выполняются, можно ввести следующий механизм доступа пользователей к серверам из других областей. Пользователю, который хочет получить сервис на сервере из другой области, необходим билет для этого сервера. Клиентский модуль следует обычным процедурам получения доступа к локальному TGS и затем запрашивает билет, гарантирующий билет, для удаленного TGS (TGS в другой области). Затем клиентский модуль вызывает удаленный TGS для получения билета, гарантирующего сервис, на требуемый сервер в области, которую обслуживает удаленный TGS.

Билет, предоставляемый удаленному серверу, кроме всего остального, содержит идентификатор области, в которой пользователь был аутентифицирован. Сервер определяет, является ли данный удаленный запрос законным.

При таком подходе традиционная проблема состоит в том, что если существует N областей, то должно быть $N \cdot (N-1)/2$ безопасных обменов ключей, чтобы каждая область Kerberos могла взаимодействовать со всеми остальными областями.

Kerberos версии 5

Версия 5 Kerberos описана в RFC 1510 и предоставляет ряд улучшений по сравнению с версией 4.

Различия между версиями 4 и 5

Версия 5 предназначена для преодоления недостатков проектирования и технических недоработок версии 4.

Версия 4 Kerberos была разработана для использования в окружении проекта Athena и, следовательно, не предназначалась для использования в общих целях. Это привело к следующим **недостаткам проектирования**:

1. **Зависимая система шифрования**: версия 4 требует использования DES. Сначала существовали экспортные ограничения DES, в настоящий момент основным недостатком является сомнение в силе DES. В версии 5 зашифрованный текст помечен типом шифрования, что позволяет задействовать любой алгоритм симметричного шифрования. Ключ шифрования помечен типом и длиной, что также позволяет применять различные алгоритмы.
2. **Зависимость от Internet-протоколов**: версия 4 требует использования IP-адресации. Другие типы адресов, такие как сетевые адреса ISO, не поддерживаются. В версии 5 сетевой адрес помечен типом и длиной, что позволяет задействовать любой тип сетевого адреса.
3. **Упорядочивание байтов сообщения**: в версии 4 отправитель сообщения использует упорядочивание байтов по своему выбору и помечает сообщение, чтобы определить, левый или правый байт расположен в младшем адресе. В версии 5 структура всех сообщений определяется на основании ASN.1 и BER, что обеспечивает однозначную последовательность байтов.
4. **Время жизни билета**: в версии 4 значения времени жизни хранятся в 8-битовых блоках по 5 минут. Таким образом, максимальное время жизни, которое может быть получено, составляет $2^8 \cdot 5 = 1280$ минут или меньше 21 часа. Для некоторых приложений этого может быть недостаточно. В версии 5 билет включает явное время начала и время конца, допуская произвольное время жизни билета.
5. **Перенаправление аутентификации**: версия 4 не позволяет, чтобы доверительные грамоты, полученные для одного клиента, были перенаправлены другому хосту и использовались другим клиентом. Эта особенность не дает клиенту возможность получить доступ к серверу, чтобы затем сервер получил доступ к другому серверу от имени данного клиента. Например, клиент сделал запрос на сервер печати, после чего необходим доступ к файл-серверу за файлом клиента, с помощью доверительной грамоты клиента. Версия 5 обеспечивает такую возможность.
6. **Аутентификация между областями**: в версии 4 взаимодействие N областей требует порядка N^2 взаимосвязей Kerberos-to-Kerberos. Версия 5 поддерживает метод, который требует меньшего числа взаимосвязей.

Существуют также следующие **технические недостатки** в самом протоколе версии 4:

1. **Двойное шифрование**: сообщения 2 и 4, которые поставляют клиентам билеты, зашифрованы дважды, один раз секретным ключом сервера назначения, а затем опять секретным ключом, известным клиенту. Второе шифрование не является обязательным и вычислительно расточительно.
2. **Ключи сессии**: каждый билет включает ключ сессии, который используется клиентом для шифрования аутентификатора, посылаемого серверу. Дополнительно ключ сессии может впоследствии использоваться клиентом и сервером для защиты сообщений, передающихся в течение данной сессии. В версии 5 появилась возможность вести переговоры о ключе подсессии, который используется только для одного соединения. Для нового соединения будет применяться новый ключ шифрования.
3. **Атаки на пароль**: одним из слабых мест, присущих обеим версиям, являются атаки на пароль. Сообщение от AS клиенту включает нечто, зашифрованное ключом, основанным на пароле клиента. Оппонент может перехватить это сообщение и попытаться расшифровать его, используя различные пароли. Если результат дешифрования будет иметь корректный формат,

то это означает, что оппонент раскрыл пароль клиента и может последовательно использовать его для получения доверительной грамоты от Kerberos. Версия 5 обеспечивает механизм, называемый предаутентификацией, который затрудняет атаки на пароли, но не предотвращает их.

Аутентифікація на основі сертифікатів. Служба аутентифікації X.509.

Введение в PKI

Цифровая подпись обеспечивает аутентификацию участника, а также служит доказательством того, что электронное сообщение было послано конкретным участником. Второе свойство является более сильным, чем аутентификация, так как аутентификация может выполняться и на основе разделяемого секрета.

Одним из требований к алгоритмам цифровой подписи является требование, чтобы было вычислительно невозможно, зная открытый ключ KU, определить закрытый ключ KR. Казалось бы, открытый ключ KU можно распространять по небезопасным сетям и хранить в небезопасных репозиторях. Но при этом следует помнить, что при использовании цифровой подписи необходимо быть уверенным, что субъект, с которым осуществляется взаимодействие с использованием алгоритма открытого ключа, является собственником соответствующего закрытого ключа. В противном случае возможна атака, когда оппонент заменяет открытый ключ законного участника своим открытым ключом, оставив при этом идентификатор законного участника без изменения. Это позволит ему создавать подписи от имени законного участника и читать зашифрованные сообщения, посланные законному участнику, используя для этого свой закрытый ключ, соответствующий подмененному открытому ключу. Для предотвращения такой ситуации следует использовать *сертификаты,* которые *являются структурами данных, связывающими значения открытого ключа с субъектом*. Для связывания необходимо наличие доверенного удостоверяющего (или сертификационного) центра (Certification Authority – CA), который проверяет идентификацию субъекта и подписывает его открытый ключ и некоторую дополнительную информацию своим закрытым ключом.

Целью Инфраструктуры Открытого Ключа (Public Key Infrastructure – PKI) является предоставление доверенного и действительного открытого ключа участника, а также управление всем жизненным циклом сертификата открытого ключа.

Основным понятием Инфраструктуры Открытого Ключа является понятие сертификата.

Сертификат участника, созданный CA, имеет следующие характеристики:

1. Любой участник, имеющий открытый ключ CA, может восстановить открытый ключ участника, для которого CA создал сертификат.
2. Никто другой, кроме данного удостоверяющего центра, не может модифицировать сертификат без обнаружения этого проверяющей стороной.

Мы будем рассматривать сертификаты X.509, хотя существует достаточно много сертификатов других форматов.

CCITT (Consultative Committee for International Telegraphy and Telephony) разработал серию рекомендаций для создания так называемой службы каталога. Каталог является сервером или распределенным набором серверов, которые поддерживают распределенную базу данных, содержащую информацию о пользователях и других объектах, которая называется Информационной Базой Каталога (Directory Information Base – DIB). Данная информация может включать имя пользователя или объекта, а также другие атрибуты. Эти рекомендации носят название стандарта X.500.

Стандарт X.509 первоначально являлся частью стандарта X.500 и описывал основные требования к аутентификации в каталоге X.500. Но X.509 используется не только в контексте службы каталога X.500. Сертификаты, определяемые данным стандартом, используются практически всеми программными продуктами, относящимися к обеспечению сетевой безопасности.

Стандарты ITU-T X.509 и ISO/IEC 9594-8, которые впервые были опубликованы в 1988 году как часть рекомендаций каталога X.500, определили формат сертификата X.509. Формат сертификата в стандарте 1988 года называется форматом версии 1 (v1). Стандарт X.500 был пересмотрен в 1993 году, в результате чего было добавлено несколько новых полей в формат сертификата X.509, который был назван форматом версии 2 (v2).

Опыт реализации первой и второй версий говорит о том, что форматы сертификата v1 и v2 имеют ряд недостатков. Самое важное, что для хранения различной информации требуется больше полей. В результате ISO/IEC, ITU-T и ANSI X9 разработали формат сертификата X.509 версии 3 (v3). Формат v3 расширяет формат v2 путем добавления заготовок для дополнительных полей расширения. Конкретные типы полей расширения могут быть определены в стандартах или определены и зарегистрированы любой организацией или сообществом. В июне 1996 года стандартизация базового формата v3 была завершена.

Однако расширения стандарта ISO/IEC, ITU-T и ANSI X9 являются слишком общими, чтобы применять их на практике. Для того чтобы разрабатывать интероперабельные реализации систем, взаимно использующие сертификаты X.509 v3, необходимо четко специфицировать формат сертификата X.509 v3. Специалисты IETF разработали профиль сертификата X.509 v3 и опубликовали его в RFC 3280.

В табл. 13.2 рассмотрены основные элементы сертификата.

Таблица 13.2. Основные элементы сертификата

Пояснение	**Параметры сертификата**	**Версия**		
Целое число, идентифицирующее данный сертификат, которое должно быть уникальным среди всех сертификатов, выпущенных данным CA	Серийный номер	v1	v2	v3
CA, который создал и подписал сертификат	Имя CA, выпустившего сертификат			
Период действительности состоит из двух временных значений, в промежутке между которыми сертификат считается действительным	Не раньше			
	Не позже			
Конечный участник, для которого создан данный сертификат	Имя субъекта (конечного участника)			
Открытый ключ субъекта и алгоритм, для которого этот ключ был создан	Алгоритм			
	Параметры			
	Открытый ключ субъекта			
	Уникальный идентификатор CA			
	Уникальный идентификатор субъекта			
	Расширения			
Подпись охватывает все остальные поля сертификата и состоит из хэш-кода других полей, зашифрованного закрытым ключом CA	Подпись, созданная закрытым ключом CA для всех полей сертификата	Все версии		

Часто используется следующая нотация для обозначения сертификата:

CA <<A>>

– сертификат пользователя A, выданный сертификационным центром CA.

CA подписывает сертификат своим закрытым ключом. Если соответствующий открытый ключ известен пользователю, то пользователь может проверить, что сертификат, подписанный CA, действителен.

Так как сертификаты не могут быть изменены без обнаружения этого, их можно разместить в общедоступной директории и пересылать по открытым каналам связи без опасения, что кто-то может их изменить.

В любом случае если **B** имеет сертификат **A**, **B** уверен, что сообщение, которое он расшифровывает открытым ключом **A**, никто не мог просмотреть, и что сообщение, подписанное закрытым ключом **A**, не изменялось.

При большом количестве пользователей неразумно подписывать сертификаты всех пользователей у одного CA. Кроме того, если существует единственный CA, который подписывает сертификаты, каждый пользователь должен иметь копию открытого ключа CA, чтобы проверять подписи. Этот открытый ключ должен быть передан каждому пользователю абсолютно безопасным способом (с обеспечением целостности и аутентификации), чтобы пользователь был уверен в подписанных им сертификатах. Таким образом, в случае большого количества пользователей лучше иметь несколько CA, каждый из которых безопасно предоставляет свой открытый ключ некоторому подмножеству пользователей.

Теперь предположим, что **A** получил сертификат от уполномоченного органа $\mathbf{X^1}$, и **B** получил сертификат от уполномоченного органа $\mathbf{X^2}$. Если **A** не знает безопасным способом открытый ключ $\mathbf{X^2}$, то сертификат **B**, полученный от $\mathbf{X^2}$, для него бесполезен. **A** может прочитать сертификат **B**, но не в состоянии проверить подпись. Тем не менее, если два CA могут безопасно обмениваться своими открытыми ключами, возможна следующая процедура для получения **A** открытого ключа **B**.

1. **A** получает из каталога сертификат $\mathbf{X^2}$, подписанный $\mathbf{X^1}$. Так как **A** знает открытый ключ $\mathbf{X^1}$ надежным способом, **A** может получить открытый ключ $\mathbf{X^2}$ из данного сертификата и проверить его с помощью подписи $\mathbf{X^1}$ в сертификате.
2. Затем **A** возвращается обратно в каталог и получает сертификат **B**, подписанный $\mathbf{X^2}$. Так как **A** теперь имеет открытый ключ $\mathbf{X^2}$ надежным способом, **A** может проверить подпись и безопасно получить открытый ключ **B**.

Для получения открытого ключа **B** **A** использует цепочку сертификатов. В приведенной выше нотации эта цепочка выглядит следующим образом:

X^1 <<X^2>> X^2 <<B>>

Аналогично **B** может получить открытый ключ **A** с помощью такой же цепочки:

X^2 <<X^1>> X^1 <<A>>

Данная схема не обязательно ограничена цепочкой из двух сертификатов. Для получения цепочки может использоваться путь CA произвольной длины. Цепочка, содержащая N элементов, выглядит следующим образом:

X^1 <<X^2>> X^2 <<X^3>> … X^N <<B>>

В этом случае каждая пара CA в цепочке (X^i, X^{i+1}) должна создать сертификаты друг для друга.

Все эти сертификаты CA необходимо разместить в каталоге, и пользователи должны иметь информацию о том, как они связаны друг с другом, чтобы получить путь к сертификату открытого ключа другого пользователя. Это определяет Инфраструктуру Открытого Ключа.

Терминология PKI

Мы будем использовать следующие термины и понятия.

1. Public Key Infrastructure (PKI) – инфраструктура открытого ключа – это множество аппаратуры, ПО, людей, политик и процедур, необходимых для создания, управления, хранения, распределения и отмены сертификатов, основанных на криптографии с открытым ключом.
2. End-entity (EE) – конечный участник, для которого выпущен данный сертификат. Важно заметить, что здесь под конечными участниками подразумеваются не только люди и используемые ими приложения, но также и исключительно сами приложения (например, для безопасности уровня IP). Этот фактор влияет на протоколы, которые используют операции управления PKI; например, ПО приложения следует более точно знать требуемые расширения сертификата, чем человеку.
3. Certificate Authority (CA) – удостоверяющий (сертификационный) центр – это уполномоченный орган, который создает и подписывает сертификаты открытого ключа. Дополнительно CA может создавать пары закрытый/открытый ключ конечного участника. Важно заметить, что CA отвечает за сертификаты открытого ключа в течение всего времени их жизни, а не только в момент выпуска.
4. Public Key Certificate (PKC) – сертификат открытого ключа или просто сертификат – это структура данных, содержащая открытый ключ конечного участника и другую информацию, которая подписана закрытым ключом CA, выпустившим данный сертификат.
5. Registration Authority (RA) – регистрационный центр – необязательный участник, ответственный за выполнение некоторых административных задач, необходимых для регистрации конечных участников. Такими задачами могут быть: подтверждение идентификации конечного участника; проверка значений, которые будут указаны в создаваемом сертификате; проверка, знает ли конечный участник закрытый ключ, соответствующий открытому ключу, указанному в сертификате. Заметим, что RA сам также является конечным участником.

Причины, которые объясняют наличие RA, могут быть разделены на имеющие технические факторы и те, которые являются организационными. К числу технических относятся следующие причины:

- Если используется аппаратный маркер, то не все конечные участники имеют необходимое оборудование для его инициализации; оборудование RA может включать необходимую функциональность (это также может быть вопросом политики).
- Не все конечные участники могут иметь возможность опубликовать сертификаты; для этого может использоваться RA.
- RA может иметь возможность выпускать подписанные запросы отмены от имени конечного участника, связанного с ним, тогда как конечный участник не всегда имеет возможность сделать это (если пара ключей потеряна).

Некоторые организационные причины для использования RA могут быть следующие:

- Может оказаться более эффективным сконцентрировать некоторую функциональность в оборудовании RA, чем иметь данную функциональность для всех конечных участников (особенно если используется специальное оборудование для инициализации маркера).
- Установление RA в организации может уменьшить число необходимых CA.
- RA могут быть лучше размещены для идентификации людей с их «электронными» именами, особенно если CA физически удален от конечного участника.
- Для многих приложений уже существуют определенные административные структуры, которые могут играть роль RA.

6. Выпускающий CRL (Certificate Revocation List) – необязательный компонент, которому CA делегирует функции опубликования списков отмененных сертификатов.
7. Certificate Policy (CP) – политика сертификата – поименованное множество правил, которое определяет применимость сертификата открытого ключа для конкретного сообщества или класса приложений с общими требованиями безопасности. Например, конкретная политика сертификата может указывать применимость типа сертификата открытого ключа для аутентификации транзакций данных при торговле товарами в данном ценовом диапазоне.
8. Certificate Practice Statement (CPS) – регламент сертификационной практики, в соответствии с которой сотрудники удостоверяющего центра выпускают сертификаты открытого ключа.
9. Relying party (RP) – проверяющая сторона – пользователь или агент (например, клиент или сервер), который использует сертификат для надежного получения открытого ключа субъекта и, быть может, некоторой дополнительной информации.

10. Root CA – CA, которому непосредственно доверяет EE; это означает безопасное приобретение значения открытого ключа корневого CA, требующее некоторых внешних шагов. Этот термин не означает, что корневой CA обязательно должен быть вершиной иерархии, просто показывает, что CA является непосредственно доверенным.
11. Subordinate CA – «подчиненный CA» представляет собой CA, который не является корневым CA для EE. Часто подчиненный CA не является корневым CA для любого участника, но это не обязательно.
12. Top CA – вершина иерархии PKI. **Замечание**: часто он также называется корневым CA, так как в терминах структур данных и в теории графов узел на вершине дерева называется корнем. Однако во избежание путаницы в данном документе будем называть данный узел «вершиной CA», а «корневой CA» зарезервируем за CA, непосредственно тем, кому доверяет EE. Данный термин не является общепринятым.
13. Репозиторий – система или набор распределенных систем, которые хранят сертификаты и CRL и предназначены для распределения этих сертификатов и CRL между конечными участниками.

Архитектура PKI

Основой для достижения безопасности в Internet являлось создание протоколов безопасности, таких как TLS, SSH и IPSec. Все эти протоколы используют криптографию с открытым ключом для предоставления таких сервисов как конфиденциальность, целостность данных, аутентификация исходных данных и невозможность отказа. Целью PKI является предоставление доверенного и действительного ключа и управление сертификатом открытого ключа, который необходим для аутентификации, невозможности отказа и конфиденциальности.

Пользователи систем, основанных на открытом ключе, должны быть уверены, что когда они используют открытый ключ, субъект, с которым они связываются, является собственником соответствующего закрытого ключа. Эта уверенность достигается благодаря использованию PKC, которые являются структурами данных, связывающих значения открытого ключа с субъектами. Связывание обеспечивается при наличии доверенного CA, который проверяет идентификацию субъекта и подписывает каждый PKC.

PKC имеет ограниченное время жизни, которое указывается в подписанном содержимом. Так как подпись и своевременность могут быть независимо проверены клиентом, использующим сертификат, PKC могут распространяться по небезопасным коммуникациям и серверным системам и кэшироваться в небезопасном хранилище в системах, использующих сертификаты.

PKC применяются в процессе проверки действительности подписанных данных. Имеется определенная специфика, относящаяся к используемому алгоритму, но общий процесс выглядит следующим образом (заметим, что не существует особенностей, относящихся к порядку, в котором должны выполняться перечисленные проверки; разработчики свободны в выборе наиболее эффективного способа для своих систем).

- Получатель подписанных данных должен убедиться, что предоставленная идентификация пользователя соответствует идентификации, содержащейся в PKC.
- Получатель проверяет, не отменен ли в полученной цепочке сертификатов какой-либо PKC (например, посредством получения соответствующего текущего CRL или on-line запроса статуса сертификата) и все ли PKC находились в рамках своих периодов действительности в момент подписывания данных.
- Получатель должен убедиться, что данные не содержат никаких значений, для которых подписывающая сторона не имеет авторизации.
- Получатель проверяет, не изменялись ли данные после подписывания, используя открытый ключ в PKC.
- Если все эти проверки проходят, получатель может считать, что данные были подписаны указанной подписывающей стороной. Процесс для ключей, используемых для шифрования, аналогичен.

Замечание: конечно, возможно, что данные были подписаны кем-то еще, если, например, закрытый ключ подписывающей стороны скомпрометирован. Безопасность зависит от всех частей системы, использующей сертификаты; она включает: физическую безопасность размещения компьютеров; безопасность персонала (например, возможность доверять людям, которые реально разрабатывают, инсталлируют, выполняют и сопровождают систему); безопасность, обеспечиваемую CA. Нарушение в любой из этих областей может послужить причиной нарушения безопасности всей системы.

PKI определяется как:

Множество аппаратуры, ПО, людей, политик и процедур, необходимых для создания, управления, хранения, распределения и отмены PKC, основываясь на криптографии с открытым ключом.

PKI состоит из следующих типов компонентов:

- CA, которые выпускают и отменяют PKC.
- RA, которые ручаются за связь между открытыми ключами и идентификациями держателей сертификатов, а также другими атрибутами.

- Держатели PKC, для которых выпущен сертификат, могут подписывать и шифровать документы, используя закрытые ключи.
- Клиенты, которые проверяют цифровые подписи и соответствующие верификационные пути с помощью известного открытого ключа доверенного CA и дешифруют документы, используя открытые ключи из сертификатов держателей PKC.
- Репозитории, которые хранят и делают доступными PKC и CRL.

На рис. 13.1 показан упрощенный взгляд на архитектурную модель PKI.

Рис. 13.1. Участники PKI

Профили сертификата и списка отмененных сертификатов

Сертификат X.509 версии 3

Пользователям открытого ключа требуются гарантии, что соответствующий закрытый ключ знает конкретный субъект, который указан владельцем данного открытого ключа. Это будет означать, что только названный субъект может использовать данный закрытый ключ для дешифрования или создания цифровой подписи. Такая гарантия достигается посредством использования сертификатов открытого ключа, которые являются структурами данных, связывающие значения открытого ключа с субъектами. Связывание осуществляется доверенным CA, который подписывает сертификат. При этом CA должен гарантировать, что субъект имеет соответствующий закрытый ключ (например, использовать доказательство обладания закрытым ключом с помощью протокола запрос-ответ), либо непосредственно предоставить закрытый ключ (вместе с сертификатом) субъекту.

Сертификат всегда имеет ограниченный срок действительности. В первую очередь это связано с тем, что любой криптографический алгоритм подвержен атакам с помощью простого перебора всего пространства ключей. Время действительности указывается в самом сертификате. Так как подпись сертификата и его своевременность могут быть независимо проверены клиентом, использующим сертификат, сертификаты могут распределяться по открытым коммуникациям и серверным системам, и могут быть кэшированы в небезопасной памяти системы, использующей сертификаты.

Расширения сертификата могут содержать такие данные как информация о дополнительной идентификации субъекта, информация об атрибутах ключа, информация о политике и ограничения на сертификационный путь.

Сертификационные пути и доверие

Пользователь сервиса безопасности является проверяющей стороной, которой требуется знание открытого ключа некоторого субъекта. В этом случае проверяющей стороне необходимо получить сертификат, содержащий требуемый открытый ключ и проверить его действительность. Если проверяющая сторона в настоящий момент не имеет заверенной копии открытого ключа CA, который подписал сертификат, не знает имени CA и, быть может, некоторой дополнительной информации, то ей требуется дополнительный сертификат для получения открытого ключа данного CA. В общем случае может потребоваться цепочка из нескольких сертификатов, содержащая сертификат открытого ключа конечного участника, подписанный одним CA, и ноль или более дополнительных сертификатов CA, подписанных другими CA. Такая цепочка, называемая сертификационным путем, необходима, потому что проверяющая сторона изначально обладает ограниченным количеством открытых ключей CA, полученных надежным способом.

Существуют различные способы, с помощью которых CA могут быть сконфигурированы для того, чтобы проверяющая сторона могла построить сертификационные пути. Первоначально планировалась жесткая иерархическая структура CA. При этом предполагалось существование трех типов уполномоченных органов сертификации:

1. Регистрационный уполномоченный орган политики Internet (IPRA): данный уполномоченный орган, функционирующий под руководством Internet-сообщества, действует в качестве корневого органа в сертификационной иерархии. Он выпускает сертификаты только для уполномоченных органов следующего уровня, PCA. Все сертификационные пути начинаются с IPRA.
2. Уполномоченные органы сертификатов политики (PCA): PCA являются вторым уровнем иерархии, каждый PCA сертифицирован IPRA. PCA должны установить и опубликовать утверждение о своей политике в отношении сертифицируемых пользователей или подчиненных уполномоченных органов сертификации. Различные PCA могут удовлетворять те или иные потребности пользователей. Например, один PCA может выпускать сертификаты для электронной почты, другой PCA может иметь политику, которая удовлетворяет более строгим законодательным требованиям в отношении цифровых подписей.
3. Уполномоченные органы сертификации (CA): CA являются третьим уровнем иерархии, но могут иметь и более низкий уровень. Те, что находятся на третьем уровне, сертифицируются PCA. CA представляют, например, отдельные организации, отдельные единицы организации (департаменты, отделы, лаборатории) или отдельные географические единицы.

RFC 1422 определяет правило подчинения имен, которое требует, чтобы CA мог выпускать сертификаты только тем пользователям, чьи имена являются подчиненными (в дереве именования X.500) по отношению к имени самого CA. Доверие, связанное с сертификационным путем, определяется именем PCA. Правило подчинения имен гарантирует, что CA, подчиняющиеся конкретному PCA, ограничены множеством участников, которых они могут сертифицировать (т.е. CA организации может сертифицировать только сотрудников данной организации). Системы сертификации пользователей должны иметь возможность автоматически проверять, выполняется ли правило подчинения имен.

RFC 1422 использует форматы сертификатов X.509 v1. Ограничения X.509 v1 требуют нескольких структурных ограничений на четкое связывание информации о политике или ограничения на использование сертификатов. Эти ограничения включают:

- Строгую иерархию сверху вниз, все сертификационные пути начинаются от IPRA.
- Правило подчинения имен ограничивает имена субъектов CA.
- Использование понятия PCA, которое требует знания индивидуальных PCA, что встроено в логику проверки цепочки сертификатов. Знание индивидуальных PCA требуется при определении возможности принятия цепочки сертификатов.

В X.509 v3 большинство ограничений, определенных в RFC 1422, может быть указано в расширениях сертификата без необходимости иметь строгую иерархическую структуру CA. В частности, расширения сертификата, касающиеся политик, устраняют необходимость применения правила подчинения имен. Как результат, третья версия может поддерживать более гибкую архитектуру:

- сертификационные пути начинаются с открытого ключа CA в собственном домене пользователя или с открытого ключа верхнего уровня иерархии. Начало сертификационного пути с открытого ключа CA в собственном домене пользователя имеет определенные преимущества. В некоторых окружениях больше доверяет локальному домену.
- Ограничения имени могут определяться с помощью явного включения в сертификат расширения ограничения имени, но это не обязательно.
- Расширения политики и отображения политики заменяют понятие PCA, что допускает большую степень автоматизации. Приложение может определить, является ли сертификационный путь действительным, на основании содержимого сертификатов вместо априорного знания открытых ключей PCA. Это позволяет лучше автоматизировать обработку сертификационного пути.

Отмена сертификата

Когда сертификат выпущен, считается, что он будет использоваться в течение всего своего периода действительности. Однако могут произойти события, в результате которых сертификат станет недействительным до завершения своего периода действительности, указанного в сертификате. К таким событиям относится изменение имени, изменение связывания между субъектом и CA (например, сотрудник увольняется из организации) и компрометация или предположение компрометации соответствующего закрытого ключа. При таких обстоятельствах CA должен отменить сертификат.

X.509 определяет один метод отмены сертификата. Данный метод предполагает, что каждый CA периодически выпускает подписанную структуру данных, называемую списком отмененных сертификатом (CRL). CRL является помеченным временем списком, который подписан CA или специальным выпускающим CRL и в котором перечислены отмененные сертификаты. Данный список становится свободно доступен в открытом репозитории. Каждый отмененный сертификат идентифицируется в CRL своим серийным номером. Когда использующая сертификат система получает сертификат (например, для проверки цифровой подписи удаленного пользователя), эта система не только проверяет подпись сертификата и действительность, но также получает свежий CRL и проверяет, не находится ли в этом CRL серийный номер сертификата. Понятие «свежий» может зависеть от

локальной политики, но обычно означает самый последний выпущенный CRL. Новый CRL выпускается регулярно (например, каждый час, день или неделю). При получении уведомления об отмене запись добавляется в CRL.

Преимущество данного метода отмены состоит в том, что CRL могут распространяться теми же способами, что и сами сертификаты, а именно через небезопасные серверы и коммуникации.

При этом недостатком данного метода отмены CRL является то, что точность отмены ограничена периодом выпуска CRL. Например, если об отмене сообщено в текущий момент, то об отмене не будут надежно оповещены использующие сертификат системы до тех пор, пока не будет выпущен новый CRL – это может занять час, день или неделю, в зависимости от частоты создания CRL.

Как и формат сертификата X.509 v3, для того, чтобы обеспечить интероперабельность реализаций различных производителей, необходимо иметь строгий формат X.509 CRL v2. Существуют также специальные протоколы и соответствующие им форматы сообщений, поддерживающих on-line оповещения об отмене. On-line методы оповещения об отмене могут применяться в некоторых окружениях как альтернатива CRL. On-line проверка отмены может существенно уменьшить промежуток между отправкой сообщения об отмене и получением этой информации проверяющей стороной. После того как CA принимает и аутентифицирует сообщение от субъекта сертификата или от RA о необходимости отмены данного сертификата, любой запрос к on-line сервису будет корректно отображать действительность сертификата. Однако эти методы навязывают новые требования безопасности: проверяющая сторона должна доверять on-line сервису действительности, в то время как репозиторий не обязательно должен быть доверяемым.