9.2. Взаимная аутентификация

Данные протоколы применяются для взаимной *аутентификации* участников и для обмена *ключом сессии*.

Основной задачей таких протоколов является обеспечение конфиденциального распределения *ключа сессии* и гарантирование его своевременности, то есть протокол не должен допускать повторного использования старого *ключа сессии*. Для обеспечения конфиденциальности *ключи сессии* должны передаваться в зашифрованном виде. Вторая задача, обеспечение своевременности, важна, потому что существует угроза перехвата передаваемого сообщения и повторной его пересылки. Такие повторения в худшем случае могут позволять взломщику использовать скомпрометированный *ключ сессии*, при этом успешно подделываясь под другого участника. Успешное повторение может, как минимум, разорвать операцию *аутентификации* участников.

Такие повторы называются *replay-атаками*. Рассмотрим возможные примеры подобных *replay-атак*:

1. Простое повторение: противник просто копирует сообщение и повторяет его позднее.
2. Повторение, которое не может быть определено: противник уничтожает исходное сообщение и посылает скопированное ранее сообщение.

Один из возможных подходов для предотвращения *replay-атак* мог бы состоять в присоединении последовательного номера (sequence number) к каждому сообщению, используемому в *аутентификационном* обмене. Новое сообщение принимается только тогда, когда его последовательный номер правильный. Трудность данного подхода состоит в том, что каждому участнику требуется поддерживать значения sequence number для каждого участника, с которым он взаимодействует в данный момент. Поэтому обычно sequence number не используются для *аутентификации* и обмена ключами. Вместо этого применяется один из следующих способов:

1. *Отметки времени*: участник А принимает сообщение как не устаревшее только в том случае, если оно содержит *отметку времени*, которая, по мнению А, соответствует текущему времени. Этот подход требует, чтобы часы всех участников были синхронизированы.
2. Запрос/ответ: участник А посылает в запросе к В случайное число (***nonce*** - number only once) и проверяет, чтобы ответ от В содержал корректное значение этого *nonce*.

Считается, что подход с *отметкой времени* не следует использовать в приложениях, ориентированных на соединение, потому что это технически трудно, так как таким протоколам, кроме поддержки соединения, необходимо будет поддерживать синхронизацию часов различных процессоров. При этом возможный способ осуществления успешной атаки может возникнуть, если временно будет отсутствовать синхронизация часов

одного из участников. В результате различной и непредсказуемой природы сетевых задержек распределенные часы не могут поддерживать точную синхронизацию. Следовательно, процедуры, основанные на любых *отметках времени*, должны допускать окно времени, достаточно большое для приспособления к сетевым задержкам, и достаточно маленькое для минимизации возможности атак.

С другой стороны, подход запрос/ответ не годится для приложений, не устанавливающих соединения, так как он требует предварительного рукопожатия перед началом передач, тем самым отвергая основное свойство транзакции без установления соединения. Для таких приложений доверие к некоторому безопасному серверу часов и постоянные попытки каждой из частей синхронизировать свои часы с этим сервером может быть оптимальным подходом.

Использование симметричного шифрования

Для обеспечения *аутентификации* и распределения *ключа сессии* часто используется двухуровневая иерархия ключей симметричного шифрования. В общих чертах эта стратегия включает использование доверенного центра распределения ключей (*KDC*). Каждый участник разделяет секретный ключ, называемый также *мастер-ключом*, с *KDC*. *KDC* отвечает за создание ключей, называемых *ключами сессии*, и за распределение этих ключей с использованием *мастер-ключей*. *Ключи сессии* применяются в течение короткого времени для шифрования только данной сессии между двумя участниками.

Большинство алгоритмов распределения секретного ключа с использованием *KDC*, включает также возможность *аутентификации* участников.

Протокол Нидхэма и Шредера

Предполагается, что секретные *мастер-ключи* K_A и K_B разделяют соответственно A и *KDC* и B и *KDC*. Целью протокола является безопасное распределение *ключа сессии* KS между A и B. Протокол представляет собой следующую последовательность шагов:

1. $A \rightarrow KDC: \ ID_A \| ID_B \| N_1$
2. $KDC \rightarrow A: \ E_{Ka} [K_S \| ID_B \| N_1 \| E_{Kb} [K_S \| ID_A]]$
3. $A \rightarrow B: \ E_{Kb} [K_S \| ID_A]$
4. $B \rightarrow A: \ E_{KS} [N_2]$
5. $A \rightarrow B: \ E_{KS} [f (N_2)]$

A запрашивает у *KDC ключ сессии* для установления защищенного соединения с B. Сообщение включает идентификацию A и B и уникальный идентификатор данной транзакции, который обозначен как N_1 и называется *nonce*. *Nonce* может быть временной меткой, счетчиком или случайным числом; минимальное требование состоит в том, чтобы он отличался для каждого запроса. Кроме того, для предотвращения подделки желательно, чтобы противнику было трудно предугадать *nonce*. Таким образом, случайное число является лучшим вариантом для *nonce*.

1. *KDC* отвечает сообщением, зашифрованным ключом K_A. Таким образом, только A может расшифровать сообщение, и A уверен, что оно получено от *KDC*, так как предполагается, что кроме A и *KDC* этот ключ не знает никто. Это сообщение включает следующие элементы, предназначенные для A:
    - Одноразовый *ключ сессии*.
    - Идентификатор B.
    - *nonce*, который идентифицирует данную сессию .

A должен убедиться, что полученный *nonce* равен значению *nonce* из первого запроса. Это доказывает, что ответ от *KDC* не был модифицирован при пересылке и не является повтором некоторого предыдущего запроса. Кроме того, сообщение включает два элемента, предназначенные для B:

- Одноразовый *ключ сессии* K_S.
- Идентификатор A ID_A.

Эти два последних элемента шифруются *мастер-ключом*, который *KDC* разделяет с B. Они посылаются B при установлении соединения и доказывают идентификацию A.

2. A сохраняет у себя *ключ сессии* и передает B информацию от *KDC*, предназначенную B: E_{Kb} [K_S || ID_A]. Так как эта информация зашифрована K_B, она защищена от просмотра. Теперь B знает *ключ сессии* (K_S), знает, что другим участником является A, (ID_A) и что начальная информация передана от *KDC*, т.к. она зашифрована с использованием K_B.

B этой точке *ключ сессии* безопасно передан от A к B, и они могут начать безопасный обмен. Тем не менее, существует еще два дополнительных шага:

3. Используя созданный *ключ сессии*, B пересылает A *nonce* N_2.
4. Также используя K_S, A отвечает f (N_2), где f - функция, выполняющая некоторую модификацию N_2.

Эти шаги гарантируют B, что сообщение, которое он получил, не изменено и не является повтором предыдущего сообщения.

Заметим, что реальное распределение ключа включает только шаги 1 - 3, а шаги 4 и 5, как и 3, выполняют функцию *аутентификации*.

A безопасно получает *ключ сессии* на шаге 2. Сообщение на шаге 3 может быть дешифровано только B. Шаг 4 отражает знание B ключа K_S, и шаг 5 гарантирует B знание участником A ключа K_S и подтверждает, что это не устаревшее сообщение, так как используется *nonce* N_2. Шаги 4 и 5 призваны предотвратить общий тип *replay-атак*. В частности, если противник имеет возможность захватить сообщение на шаге 3 и повторить его, то это должно привести к разрыву соединения.

Разрывая рукопожатие на шагах 4 и 5, протокол все еще уязвим для некоторых форм атак повторения. Предположим, что противник X имеет возможность скомпрометировать старый *ключ сессии*. Маловероятно, чтобы противник мог сделать больше, чем просто копировать сообщение шага 3. Потенциальный риск состоит в том, что X может заставить взаимодействовать A и B, используя старый *ключ сессии*. Для этого X просто

повторяет сообщение шага 3, которое было перехвачено ранее и содержит скомпрометированный *ключ сессии*. Если B не запоминает идентификацию всех предыдущих *ключей сессий* с A, он не сможет определить, что это повтор. Далее X должен перехватить сообщение рукопожатия на шаге 4 и представиться A в ответе на шаге 5.

Протокол Деннинга

Деннинг предложил преодолеть эту слабость модификацией протокола Нидхэма и Шредера, которая включает дополнительную *отметку времени* на шагах 2 и 3:

1. $A \rightarrow KDC: \ ID_A \ || \ ID_B$
2. $KDC \rightarrow A: \ E_{Ka} [K_S \ || \ ID_B \ || \ T \ || \ E_{Kb} [K_S \ || \ ID_A \ || \ T]]$
3. $A \rightarrow B: \ E_{Kb} [K_S \ || \ ID_A \ || \ T]$
4. $B \rightarrow A: \ E_{KS} [N_1]$
5. $A \rightarrow B: \ E_{KS} [f (N_1)]$

T – это *отметка времени*, которая гарантирует A и B, что *ключ сессии* является только что созданным. Таким образом, и A, и B знают, что распределенный ключ не является старым. A и B могут верифицировать *временную отметку* проверкой, что $|Clock - T| < \Delta t_1 + \Delta t_2$

где $\Delta \ t_1$ - оцениваемое нормальное расхождение между часами *KDC* и локальными часами (у A или B) и t_2 - ожидаемая сетевая задержка времени. Каждый участник может установить свои часы, ориентируясь на определенный доверенный источник. Поскольку *временная отметка* T шифруется с использованием секретных *мастер-ключей*, взломщик, даже зная старый *ключ сессии*, не сможет достигнуть цели повторением шага 3 так, чтобы B не заметил искажения времени.

Шаги 4 и 5 не были включены в первоначальное представление, но были добавлены позднее. Эти шаги подтверждают A, что B получил *ключ сессии*.

Протокол Деннинга обеспечивает большую степень безопасности по сравнению с протоколом Нидхэма и Шредера. Однако данная схема требует доверия к часам, которые должны быть синхронизированы в сети. В этом есть определенный риск, который состоит в том, что распределенные часы могут рассинхронизироваться в результате диверсии или повреждений. Проблема возникает, когда часы отправителя спешат по отношению к часам получателя. В этом случае противник может перехватить сообщение от отправителя и повторить его позднее, когда *отметка времени* в сообщении станет равной времени на узле получателя. Это повторение может иметь непредсказуемые последствия.

Один способ вычисления атак повторения состоит в требовании, чтобы участники регулярно сверяли свои часы с часами *KDC*. Другая альтернатива, при которой нет необходимости всем синхронизировать часы, состоит в доверии протоколам рукопожатия, использующим *nonce*.

Протокол аутентификации с использованием билета

Данный протокол пытается преодолеть проблемы, возникшие в предыдущих двух протоколах. Он выглядит следующим образом:

1. $A \rightarrow B: \quad ID_A \parallel N_a$
2. $B \rightarrow KDC: ID_B \parallel N_b \parallel E_{Kb} [ID_A \parallel N_a \parallel T_b]$
3. $KDC \rightarrow A: E_{Ka} [ID_B \parallel N_a \parallel K_S \parallel T_b] \parallel E_{Kb} [ID_A \parallel K_S \parallel T_b] \parallel N_b$
4. $A \rightarrow B: \quad E_{Kb} [ID_A \parallel K_S \parallel T_b] \parallel E_{KS} [N_b]$

A инициализирует *аутентификационный* обмен созданием *nonce* N_a и посылкой его и своего идентификатора к B в незашифрованном виде. Этот *nonce* вернется к A в зашифрованном сообщении, включающем *ключ сессии*, гарантируя A, что *ключ сессии* не старый.

1. B сообщает *KDC*, что необходим *ключ сессии*. Это сообщение к KDC включает идентификатор B и *nonce* N_b. Данный *nonce* вернется к B в зашифрованном сообщении, которое включает *ключ сессии*, гарантируя B, что *ключ сессии* не устарел. Сообщение B к *KDC* также включает блок, зашифрованный секретным ключом, разделяемым B и *KDC*. Этот блок используется для указания *KDC*, когда заканчивается время жизни данного *ключа сессии*. Блок также специфицирует намеченного получателя и содержит *nonce*, полученный от A. Этот блок является своего рода "верительной грамотой" или "билетом" для A.
2. *KDC* получил *nonces* от A и B и блок, зашифрованный секретным ключом, который B разделяет с *KDC*. Блок служит билетом, который может быть использован A для последующих *аутентификаций*. *KDC* также посылает A блок, зашифрованный секретным ключом, разделяемым A и *KDC*. Этот блок доказывает, что B получил начальное сообщение A (ID_B), что в нем содержится допустимая *отметка времени* и нет повтора (N_a). Этот блок обеспечивает A *ключом сессии* (K_S) и устанавливает ограничение времени на его использование (T_b).
3. A посылает полученный билет B вместе с *nonce* B, зашифрованным *ключом сессии*. Этот билет обеспечивает B *ключом сессии*, который тот использует для дешифрования и проверки *nonce*. Тот факт, что *nonce* B расшифрован *ключом сессии*, доказывает, что сообщение пришло от A и не является повтором.

Данный протокол аутентифицирует A и B и распределяет *ключ сессии*. Более того, протокол предоставляет в распоряжение A билет, который может использоваться для его последующей *аутентификации*, исключая необходимость повторных контактов с *аутентификационным* сервером. Предположим, что A и B установили сессию с использованием описанного выше протокола и затем завершили эту сессию. Впоследствии, но до истечения лимита времени, установленного протоколом, A может создать новую сессию с B. Используется следующий протокол:

1. $A \rightarrow B: \quad E_{Kb} [ID_A \parallel K_S \parallel T_b], N_a'$
2. $B \rightarrow A: \quad N_b', E_S [N_a']$
3. $A \rightarrow B: \quad E_S [N_b']$

Когда B получает сообщение на шаге 1, он проверяет, что билет не просрочен. Заново созданные *nonces* N_a' и N_b' гарантируют каждому участнику, что не было атак повтора. Время T_b является временем относительно часов B. Таким образом, эта временная метка не требует синхронизации, потому что B проверяет только им самим созданные *временные отметки*.

Использование шифрования с открытым ключом

Протокол аутентификации с использованием аутентификационного сервера.

Рассмотрим протокол, использующий *отметки времени* и *аутентификационный* сервер:

1. $A \rightarrow AS: \ ID_A \| ID_B$
2. $AS \rightarrow A: \ E_{KRas} [ID_A \| KU_a \| T] \| E_{KRas} [ID_B \| KU_b \| T]$
3. $A \rightarrow B: \ E_{KRas} [ID_A \| KU_a \| T] \| E_{KRas} [ID_B \| KU_b \| T] \| E_{KUb} [E_{KRa} [K_S \| T]]$

B данном случае *третья доверенная сторона* является просто *аутентификационным* сервером AS, потому что *третья сторона* не создает и не распределяет секретный ключ. AS просто обеспечивает сертификацию открытых ключей участников. *Ключ сессии* выбирается и шифруется A, следовательно, не существует риска, что AS взломают и заставят распределять скомпрометированные *ключи сессии*. *Отметки времени* защищают от повтора скомпрометированных *ключей сессии*.

Данный протокол компактный, но, как и прежде, требует синхронизации часов.

Протокол аутентификации с использованием KDC

Другой подход использует *nonces*. Этот протокол состоит из следующих шагов:

1. $A \rightarrow KDC: \ ID_A \| ID_B$
2. $KDC \rightarrow A: \ E_{KRkdc} [ID_B \| KU_b]$
3. $A \rightarrow B: \ E_{KUb} [N_a \| ID_A]$
4. $B \rightarrow KDC: \ ID_B \| ID_A \| E_{KUkdc} [N_a]$
5. $KDC \rightarrow B: \ E_{KRkdc} [ID_A \| KU_a] \| E_{KUb} [E_{KRkdc} [N_a \| K_S \| ID_B]]$
6. $B \rightarrow A: \ E_{KUa} [E_{KRkdc} [N_a \| K_S \| ID_B] \| N_b]$
7. $A \rightarrow B: \ E_{KS} [N_b]$

На первом шаге A информирует *KDC*, что хочет установить безопасное соединение с B. *KDC* возвращает A сертификат открытого ключа B (шаг 2). Используя открытый ключ B, A информирует B о создании защищенного соединения и посылает *nonce* N_a (шаг 3). На 4-м шаге B спрашивает *KDC* о сертификате открытого ключа A и запрашивает *ключ сессии*. B включает *nonce* A, чтобы *KDC* мог пометить *ключ сессии* этим *nonce*. *Nonce* защищен использованием открытого ключа *KDC*. На 5-м шаге *KDC* возвращает B сертификат открытого ключа A плюс информацию $\{N_a, K_S, ID_B\}$. Эта информация означает, что K_S является секретным ключом, созданным *KDC* в интересах B и связан с N_a. Связывание K_S и N_a гарантирует A, что K_S не устарел. Эта тройка шифруется с использованием закрытого ключа *KDC*, это

гарантирует B, что тройка действительно получена от *KDC*. Она также шифруется с использованием открытого ключа B, чтобы никто другой не мог подсмотреть *ключ сессии* и использовать эту тройку для установления соединения с A. На шаге 6 тройка $\{N_a, K_S, ID_B\}$, зашифрованная закрытым ключом *KDC*, передается A вместе с *nonce* N_b, созданным B. Все сообщение шифруется открытым ключом A. A восстанавливает *ключ сессии* K_S, использует его для шифрования N_b, который возвращает B. Это последнеее сообщение гарантирует B, что A знает *ключ сессии*.

Это достаточно безопасный протокол при различного рода атаках. Однако авторы предложили пересмотренную версию данного алгоритма:

1. A -> KDC: $ID_A \| ID_B$
2. KDC -> A: $E_{KRauth} [ID_B \| KU_b]$
3. A -> B: $E_{KUb} [N_a \| ID_A]$
4. B -> KDC: $ID_B \| ID_A \| E_{KUauth} [N_a]$
5. KDC -> B: $E_{KRauth} [ID_A \| KU_a] \| E_{KUb} [E_{KRauth} [N_a \| K_S \| ID_A \| ID_B]]$
6. B -> A: $E_{KUa} [E_{KRauth} [N_a \| K_S \| ID_A \| ID_B] \| N_b]$
7. A -> B: $E_{KS} [N_b]$

Добавляется идентификатор A ID_A к данным, зашифрованных с использованием закрытого ключа *KDC* на шагах 5 и 6 для идентификации обоих участников сессии. Это включение ID_A приводит к тому, что значение *nonce* N_a должно быть уникальным только среди всех *nonces*, созданных A, но не среди *nonces*, созданных всеми участниками. Таким образом, пара $\{ID_A, N_a\}$ уникально идентифицирует соединение, созданное A.

Односторонняя аутентификация

Существует специфическое приложение - электронная почта, для которого шифрование также имеет большое значение. Особенность e-mail состоит в том, что отправителю и получателю нет необходимости быть на связи в одно и то же время. Вместо этого сообщения направляются в почтовый ящик получателя, где они хранятся до тех пор, пока у того не появится возможность получить их.

Заголовок сообщения должен быть незашифрованным, чтобы сообщение могло пересылаться протоколами e-mail, такими как X.400 или SMTP. Однако желательно, чтобы протоколы управления почтой не имели бы доступа к самому сообщению. Соответственно, e-mail сообщение должно быть зашифровано так, чтобы система управления почтой могла бы не знать ключ шифрования.

Вторым требованием является *аутентификация* сообщения. Обычно получателю нужна определенная гарантия того, что сообщение пришло от законного отправителя.

Использование симметричного шифрования

При использовании симметричного шифрования сценарий централизованного распределения ключей в полном объеме непригоден. Эти схемы требуют, чтобы в двух заключительных шагах отправитель посылал запрос получателю, ожидая ответа с созданным *ключом сессии*, и только после этого отправитель может послать сообщение.

С учетом перечисленных ограничений протоколы использования *KDC* являются возможными кандидатами для шифрования электронной почты. Для того чтобы избежать требования к получателю B находиться на связи в то же самое время, когда и отправитель A, шаги 4 и 5 должны быть опущены. Таким образом, остается последовательность шагов:

1. A -> KDC: $ID_A \| ID_B \| N_1$
2. KDC -> A: $E_{Ka} [K_S \| ID_B \| N_1 \| E_{Kb} [K_S \| ID_A]]$
3. A -> B: $E_{Kb} [K_S , ID_A] \| E_{KS} [M]$

Данный подход гарантирует, что только требуемый получатель сообщения сможет прочитать его. Это также обеспечивает определенный уровень *аутентификации*, что отправителем является A. Очевидно, что протокол не защищает от атак повтора. Некоторая мера защиты может быть обеспечена включением *отметки времени* в сообщение. Однако поскольку существуют потенциальные задержки в процессе передачи e-mail сообщений, такие *временные отметки* имеют ограниченный срок действия.

Использование шифрования с открытым ключом

При использовании шифрования с открытым ключом требуется, чтобы отправитель знал открытый ключ получателя (для обеспечения конфиденциальности), получатель знал открытый ключ отправителя (для обеспечения *аутентификации*), или и то, и другое (для обеспечения конфиденциальности и *аутентификации*).

Если требуется конфиденциальность, то может быть использована следующая схема: A -> B: $E_{KUb} [K_S] \| E_{KS} [M]$

В этом случае сообщение шифруется одноразовым секретным ключом. A шифрует этот одноразовый ключ открытым ключом B. Только B имеет соответствующий закрытый ключ для получения одноразового ключа и использования этого ключа для дешифрования сообщения. Эта схема более эффективна, чем простое шифрование всего сообщения открытым ключом B.

Если требуется *аутентификация*, то цифровая подпись может быть создана по такой схеме: A -> B: $M \| E_{KRa} [H (M)]$

Этот метод гарантирует, что A не сможет впоследствии отвергнуть полученное сообщение. Однако данная технология открыта для другого типа подделок. Например, можно получить доступ к почтовой очереди перед доставкой, вырезать подпись отправителя, вставить свою и опять поставить сообщение в очередь на доставку.

Чтобы этого не допустить, сообщение и подпись можно зашифровать ключом симметричного шифрования, который в свою очередь шифруется открытым ключом получателя: A -> B: $E_{KS} [M \| E_{KRa} [H (M)]] \| E_{KUb} [K_S]$

Следующая схема не требует, чтобы B знал открытый ключ A. В этом случае должен использоваться сертификат открытого ключа:

A -> B: $M \| E_{KRa} [H (M)] \| E_{KRas} [T \| ID_A \| KU_a]$

В конце сообщения A посылает B подпись, зашифрованную закрытым ключом A, и сертификат A, зашифрованный закрытым ключом *аутентификационного* сервера. Получатель применяет сертификат для получения открытого ключа отправителя и затем использует открытый ключ

отправителя для проверки самого сообщения. Конфиденциальность может быть добавлена аналогично предыдущей схеме.