Низкоуровневые криптографические операции

Н. Д. Байков[1], А. Н. Годунов[2]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nknikita@niisi.ras.ru;

[2]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, nkag@niisi.ras.ru

Аннотация. Целью работы является обзор базовых низкоуровневых криптографических операций, лежащих в основе современных криптографических протоколов. Рассмотрены примеры широко применяемых операций криптографического хэширования, шифрования и формирования электронно-цифровой подписи.

Ключевые слова: криптографическая хэш-функция, шифрование, цифровая подпись

1. Введение

Архитектура безопасности большого числа современных систем передачи и хранения информации строится на трех главных видах криптографических операций:

- криптографическое хэширование данных (свертка данных);
- шифрование данных;
- формирование цифровой подписи.

За последние несколько десятилетий было разработано обширное множество различных подходов к реализации этих операций, каждый со своими собственными особенностями, отражающими веяния современных тенденций в криптографии и покрывающими все новые и новые обнаруживаемые уязвимости. Подобное многообразие особенностей и подходов на ранних этапах ознакомления с предметной областью существенно повышает порог входа для новичков, целью которых является внедрение систем безопасности в собственные программные продукты и системы.

Задачей настоящей работы является ознакомление с фундаментальными идеями, лежащими в основе алгоритмов выполнения криптографических операций, и областью применения каждой из этих операций. Изложение преимущественно фокусируется на примерах классических алгоритмов, чтобы сформировать у читателя общее представление об их различиях и предоставить ему возможность самостоятельно определиться с направлением дальнейшего изучения.

Раздел 2 посвящен описанию области применения операции криптографического хэширования данных. В частности, рассмотрено применение алгоритмов в задаче хранения аутентификационных данных (паролей). Сформулированы основные требования к алгоритмам криптографического хэширования, обеспечивающие им свойство криптографической стойкости, т.е. способности алгоритма противостоять атакам на основе методов криптографического анализа. Стойким в данном контексте считается, алгоритм, атака на который требует настолько значительных (или вовсе недостижимых) ресурсов атакующего, что затраты на них значительно превосходят ценность защищаемых данных. Приведены примеры зарубежных алгоритмов бесключевого хэширования, включающие в себя семейства алгоритмов MD [1], SHA [2], [3] и отечественного алгоритма Стрибог [4], [5].

В разделе 3 рассмотрены алгоритмы шифрования и их использование для обеспечения конфиденциальности данных при их передаче (например, по протоколу HTTP с поддержкой Transport Layer Security (TLS) [6]). Описаны основные идеи семейств алгоритмов симметричного шифрования на основе общего секретного ключа и алгоритмов асимметричного шифрования на основе пары из публичного ключа для шифрования данных и секретного ключа для их расшифровки. На примерах алгоритмов симметричного шифрования также продемонстрированы различия между шифрованием данных блоками и поточным шифрованием, описаны достоинства и недостатки этих подходов.

В разделе 4 приведено описание операции формирования цифровой подписи и ее применение для обеспечения гарантий целостности передаваемых данных. Рассмотрены особенности протоколов использования цифровых подписей для проверки подлинности получаемых по сети публичных ключей при установлении зашифрованного соединения с сервером посредством асимметричного алгоритма шифрования.

В разделе 5 рассматриваются примеры готовых программных реализаций всех трех видов операций в составе библиотек и утилит. Примеры включают в себя как зарубежные программные продукты (такие как OpenSSL [7], NaCl [8] или утилиту Crypt4GH, реализующую

стандарт GA4GH [9]), так и отечественные средства криптографической защиты (например, криптопровайдер КриптоПро CSP [10]).

2. Криптографическое хэширование данных

Под хэшированием в широком смысле подразумевается любой односторонний процесс преобразования входных данных в выходные по некоторому заданному алгоритму. Дополнительные требования к алгоритму зависят от области его применения. В криптографии операция хэширования обычно применяется для обеспечения целостности передаваемых данных, а также для повышения безопасности при хранении аутентификационных данных. Для этого алгоритму хэширования требуется обладать дополнительным свойством — свойством криптографической стойкости, которое на практике достигается за счет выполнения следующих основных требований:

- при применении алгоритма к одним и тем же входным значениям всегда должны получаться одни и те же выходные значения (не обязательно уникальные);
- восстановление возможного прообраза по полученным выходным значениям должно иметь крайне высокую трудоемкость и требовать очень большого времени;
- малейшее изменение входных значений должно приводить к существенному и трудно предсказуемому изменению получаемых в результате выходных значений;
- алгоритм должен минимизировать вероятность возникновения коллизий, когда для двух различных входных значений в результате получаются совпадающие выходные значения;
- выходные данные не должны нести в себе какого-либо дополнительного смысла, кроме того, что получение двух различных выходных значений свидетельствует о несовпадении исходных входных значений.

Рассмотрим использование операций хэширования для обеспечения безопасного хранения аутентификационных данных. Общепринятым способом аутентификации пользователя является использование паролей. В современных информационных системах хранение паролей учетных записей пользователей в открытом виде считается нежелательной практикой, т.к. в случае утечки базы данных злоумышленнику не потребуется прикладывать дополнительные усилия для получения доступа к системе. Криптографическое хэширование призвано усложнить злоумышленнику задачу подбора пароля и создать дополнительный запас времени, за который можно было бы успеть обнаружить утечку и изменить пароль до того, как он станет известен злоумышленнику и будет нанесен ущерб системе или пользователю.

Для этой цели значения паролей в базе данных информационной системы заменяются значениями их криптографических хэшей. При этом по-прежнему возможно достаточно надежно аутентифицировать пользователя при входе в систему по значению его пароля: нужно вычислить хэш пароля, а затем сравнить полученный хэш со значением хэша указанного пользователя в базе. Вместе с этим подобная замена значительно понижает риск мгновенной компрометации учетных записей пользователей в случае утечки базы, т.к. для подбора исходного пароля злоумышленнику необходимо решить задачу восстановления прообраза хэша, что сделать достаточно трудно.

На практике только этих мер для обеспечения безопасности оказывается недостаточно. Хотя хэширование само по себе является эффективным средством для сокрытия значения исходного пароля пользователя, оно не учитывает влияние человеческого фактора. Часто пользователи используют в качестве пароля типовые легко запоминающиеся комбинации символов. Если злоумышленнику известен используемый в системе алгоритм криптографического хэширования, он может составить список из наиболее часто встречающихся паролей, заблаговременно вычислить значения всех их хэшей и упорядочить полученную таблицу хэшей для быстрого поиска по ней. Тогда, если произойдет утечка базы, ему будет достаточно сопоставить аутентификационные данные системы с имеющимся у него упорядоченным перечнем хэшей, чтобы в кратчайшие сроки выявить возможные учетные записи со слабыми паролями и получить доступ к системе. Хотя ответственность за использование сложного пароля прежде всего лежит на самом пользователе, тем не менее в качестве дополнительной меры противодействия данной угрозе был предложен метод автоматического усложнения пользовательских паролей, позволяющий еще больше замедлить злоумышленника. Так появилось понятие «соли» (salt) — последовательности символов, добавляемой к паролю при каждом вычислении хэша. Соль случайным образом формируется в момент регистрации пользователя в системе. Сформированные значения сохраняются в базе данных системы в открытом виде, но держатся в секрете от любых внешних участников информационного обмена. Единственная задача соли состоит в том, чтобы за счет ее добавления (конкатенации) усилить пароль и превратить его в более длинную и нечитаемую строку, которой не окажется в заранее

подготовленном и оптимизированном для быстрого поиска перечне злоумышленника. При этом следует понимать, что утечка значений соли вместе с другими аутентификационными данными по-прежнему ставит под угрозу безопасность системы. Отличие только в том, что злоумышленнику понадобится больше времени на взлом, т.к. ему будет необходимо повторно перебрать все типовые пароли и вычислить их хэши с учетом ставшей известной ему соли.

Классическими примерами семейств (бесключевых) криптографических хэш-функций являются семейства MD-функций (MD2, MD4, MD5 [1] и MD6), а также семейство функций SHA (Secure Hash Algorithm). Часть из указанных хэш-функций за прошедшее с момента изобретения время уже утратили свойства криптографической стойкости, тогда как оставшиеся функции до сих пор активно применяются на практике. К последним можно отнести разновидности SHA-2 [2] для различных длин хэшей, а также относительно новый алгоритм SHA-3 (Keccak) [3]. Отечественным вариантом криптографического хэширования выступает алгоритм «Стрибог», описанный в межгосударственном криптографическом стандарте ГОСТ 34.11-2018 [4], разработанном на основе национального стандарта Российской Федерации ГОСТ Р 34.11.2012 [5].

3. Шифрование данных

Другой важной с точки зрения безопасности операцией является шифрование. Ее основное назначение — соблюдение условия конфиденциальности при передаче информации от отправителя к получателю. Требуется, чтобы никакой сторонний субъект, которому удастся перехватить сообщение, не был способен прочесть скрытую в сообщении информацию. Для достижения поставленной цели также используется алгоритм преобразования исходных данных, но — в отличие от алгоритма криптографического хэширования — задача восстановления прообраза зашифрованных данных получателем должна быть однозначно разрешима. Конфиденциальность же обеспечивается тем условием, что для расшифровки данных алгоритму необходим дополнительный аргумент — персональный ключ шифрования, который получатель должен хранить в тайне от других участников информационного обмена и для которого алгоритм шифрования должен гарантировать недопустимо высокую трудоемкость его подбора по структуре зашифрованных данных. Отправителю для шифрования данных также необходим парный к ключу получателя ключ, который в зависимости от используемого алгоритма может быть как секретным, так и публично доступным. Принято выделять два основных семейства криптографических алгоритмов шифрования:

- алгоритмы на основе секретного симметричного ключа, используемого одновременно для шифрования и расшифровки данных;

- алгоритмы асимметричного шифрования с отдельным публичным ключом для шифрования данных и отдельным секретным ключом для расшифровки данных.

Простейшим примером алгоритма симметричного шифрования является предложенный в 1917 году шифр Вернама, предназначавшийся для защищенной передачи телеграфных сообщений. Алгоритм состоит всего лишь из одного действия — применения логической операции XOR. Аргументами операции выступают исходное шифруемое сообщение и секретный ключ, длина которого должна совпадать с длиной сообщения. Для расшифровки данных принимающей стороне также достаточно всего лишь применить операцию XOR при условии, что ей известен секретный ключ. Данный алгоритм, как было показано Шенноном, обладает абсолютной криптографической стойкостью. Однако для его применения на практике необходимо, чтобы в каждой операции всегда был задействован новый случайно генерируемый ключ, длина которого бы совпадала и длиной шифруемых данных. В противном случае алгоритм становится уязвим к методам статистического анализа, т.к. XOR между двумя зашифрованными одним и тем же ключом сообщениями позволяет исключить из рассмотрения значение ключа и определить позиции совпадающих и несовпадающих битов исходных сообщений. Если шифр используется для передачи текстовых данных, поиск совпадений длиной в один или несколько байтов позволит определить позиции совпадающих символов, на место которых можно попытаться подставить наиболее распространенные символы используемого алфавита. Также злоумышленник получит возможность отслеживать отправку повторяющихся сообщений, а взлом шифра хотя бы одного из сообщений сразу же позволит расшифровать все оставшиеся сообщения.

Практическая реализация формирования и безопасной передачи очень длинных и при этом абсолютно случайных ключей весьма затратна. По этой причине в чистом виде шифр Вернама практически не используется. Вместо этого алгоритмы шифрования обычно прибегают к псевдослучайному формированию шифрующей последовательности. Упрощенно это выглядит следующим образом:

- выбирается секретный симметричный ключ

шифрования фиксированной длины;

- задается функция криптографического хэширования, для которой секретный ключ должен являться дополнительным параметром;

- задается последовательность целых неотрицательных чисел *nonce*, *nonce*+1, *nonce*+2, … Точка отсчета *nonce* выбирается таким образом, чтобы среди элементов последовательности не встречались уже ранее использовавшиеся в других операциях шифрования элементы. Как альтернатива также встречается использование фиксированной последовательности 0, 1, 2, 3… с использованием *nonce* в роли одноразового дополнительного параметра для инициализации начального состояния функции хэширования наравне с секретным ключом;

- используя секретный симметричный ключ, путем вычисления хэшей от *nonce*+*i* формируется шифрующая последовательность;

- преобразование исходных данных в шифр осуществляется через XOR с шифрующей последовательностью;

- принимающей стороне зашифрованные данные передаются вместе со значением *nonce*, с помощью которого она должна восстановить шифрующую последовательность из собственного значения секретного ключа.

Описанную выше процедуру принято называть поточным шифрованием, т.к. она позволяет шифровать исходные данные произвольной длины побитово и свободно перемещаться между участками данных при шифровании и расшифровке. Примерами используемых на практике алгоритмов поточного симметричного шифрования являются Salsa20 [11], XSalsa20 (расширенный вариант Salsa20 с увеличением размера *nonce* с 64 до 192 бит), а также ChaCha20 [12].

Альтернативой алгоритмам поточного шифрования выступают алгоритмы блочного шифрования. В них входные данные передаются функции шифрования порциями в виде отдельных блоков фиксированной длины. Широко известными примерами алгоритмов блочного шифрования являются ныне устаревший алгоритм DES (Data Encryption Standard) с длиной ключа 56 бит и пришедший ему на смену AES (Advanced Encryption Standard) с поддерживаемыми длинами ключей 128/192/256 бит [13]. В частности, AES использует блоки размера 128 бит, из которых функция шифрования сначала формирует квадрат 4x4 байта для того, чтобы далее выполнять на нем различные операции перестановки.

Случаи, когда длина передаваемого сообщения точно совпадает с длиной одного блока, на практике встречаются достаточно редко. Гораздо чаще пересылаемые данные имеют произвольную длину, значительно превышающую размеры блока. Простейшим решением в таких случаях было бы дополнение исходного сообщения служебными битами до длины, кратной длине блока, а затем последовательное применение функции шифрования к каждому блоку сообщения. Такой режим использования алгоритма шифрования AES получил название Electronic Codebook (ECB). В реальных системах AES-ECB почти не встречается из-за заложенного в него недостатка — низкой эффективности при работе с данными, имеющими периодическую структуру и содержащими большое количество повторяющихся блоков. Проблемой является то, что для одинаковых входных блоков функция шифрования сформирует одинаковый шифр. Этим она частично раскроет для сторонних наблюдателей структуру исходных данных. Особенно наглядно эту проблему демонстрируют примеры использования AES-ECB для зашифрованной передачи изображений, где через шифр могут визуально угадываться контуры исходного изображения. Для борьбы с данным недостатком блочных алгоритмов были разработаны иные режимы функционирования [14], наиболее известными из которых являются:

- Cipher Block Changing (CBC) или режим сцепления блоков шифротекста;

- Counter Mode (CTR) или режим счетчика;

- Galois/Counter Mode (GCM).

В случае CBC проблема повторяющихся блоков решается добавлением к шифруемым данным «шумов». Алгоритм добавления в этом случае выглядит следующим образом:

- перед началом работы случайным образом выбирается вектор инициализации (IV). Вектор должен иметь длину одного блока и задает начальное значение для шума. Получателю IV передается в открытом виде;

- блоки передаваемого сообщения шифруются последовательно;

- шаг алгоритма состоит из получения очередного блока исходного текста, добавления к нему текущего значения шума с помощью операции XOR и передачи полученного результата на вход функции шифрования блока (AES);

- после каждого шифрования полученный в результате зашифрованный блок устанавливается как новое текущее значение шума.

Хотя данный режим эффективно решает проблему шифрования повторяющихся блоков, у него есть собственные недостатки. Среди них невозможность распараллеливания операции шифрования (хотя при этом присутствует возможность параллельной расшифровки данных), а также наличие угроз вида Padding Oracle Attack [15]. Последнюю рассмотрим более подробно.

Для реализации атаки Padding Oracle Attack достаточно следующих условий:

- злоумышленник способен перехватывать зашифрованные сообщения отправителя;

- злоумышленнику известна используемая длина блока (для AES это 16 байт), поэтому он может разделить сообщение на отдельные блоки и расшифровывать их по отдельности друг от друга;

- злоумышленнику известен используемый при шифровании формат дополнения сообщения служебными байтами до длины, кратной длине блока. Как правило, используется формат, когда к последнему блоку длины 15 байт добавляется один байт со значением 1, для блока длины 14 байт — два байта со значением 2, для блока длины 13 байт — три байта со значением 3 и т.д. Если длина сообщения кратна 16, используется добавочный блок длины 16, у которого все байты имеют значение 16. Ясно, что в этом случае сообщение, последний блок которого оканчивается, например, двумя байтами со значениями 3 и 2, не будет являться корректным;

- злоумышленник может неограниченное отправлять серверу поддельные сообщения; и

- злоумышленник каким-то образом (например, по разнице во времени отклика сервера на его запросы) способен определить, посчитал ли сервер его сообщение некорректным из-за неправильной последовательности добавочных байтов или нет.

Взлом шифра в этом случае производится поблочно. Блоки взламываются побайтово. В каждом блоке байты взламываются в порядке от конца к началу. Взлом каждого байта выполняется методом перебора. Для взлома последнего байта злоумышленник формирует сообщения, состоящие ровно из двух блоков:

- вторым блоком сообщения является взламываемый блок;

- в первом блоке первые 15 байт задаются случайным образом;

- значения последнего байта перебираются в порядке от 0 до 255, до тех пор, пока не обнаружится, что для какого-то из значений сервер посчитал сообщение корректным.

Корректность сообщения означает, что по результатам применения операции расшифровки второго блока по алгоритму AES (или другому блочному алгоритму шифрования) и вычисления XOR между ним и значением первого блока хвостовые байты полученного в результате блока образуют последовательность, которую сервер интерпретирует как правильную последовательность добавочных байтов. Таковой гарантированно является последовательность из одного последнего байта со значением 1. Для некоторых блоков дополнительно возможны последовательности 2-2, 3-3-3 и т.д., отфильтровать которые можно путем модификации первых 15 байтов первого блока поддельного сообщения.

Информации о том, что последний байт после применения процедуры расшифровки в соответствии с режимом CBC для поддельного сообщения имеет значение 1 достаточно для его взлома в исходном сообщении. Для этого нужно последовательно сделать две операции XOR сначала между единичным байтом и последним подобранным байтом первого блока в подставном сообщении злоумышленника, а затем между получившимся результатом и последним байтом предыдущего блока в исходном сообщении отправителя.

Далее, используя уже полученное поддельное сообщение, необходимо применить аналогичную процедуру подбора предпоследнего байта в первом блоке с целью получить после применения алгоритма расшифровки CBC хвостовую последовательность вида 2-2, затем последовательность 3-3-3 для взлома третьего байта с конца и т.д. Таким образом расшифровывается каждый байт в блоке.

Пример с Padding Oracle Attack наглядно демонстрирует, что достаточно малейшего несовершенства реализации алгоритма (например, диагностируемого несовпадения во времени обработки отдельных ветвей алгоритма), чтобы сделать систему полностью уязвимой

Режим CTR фактически превращает работу с блоками в алгоритм поточного шифрования, где функция шифрования блока используется как функция хэширования:

- перед началом работы выбирается одноразовый целочисленный параметр *nonce*, на основе которого строится последовательность *nonce*, *nonce*+1, *nonce*+2, … по числу блоков исходного сообщения;

- к каждому элементу последовательности *nonce*+*i* применяется функция шифрования для получения очередного блока шифрующей последовательности;

- блоки шифра получаются путем суммирования (XOR) блоков исходного текста передаваемого сообщения с полученными блоками шифрующей последовательности;

- получателю вместе с зашифрованными данными в открытом виде передается значение *nonce*, чтобы он при помощи собственной копии симметричного ключа имел возможность воспроизвести значения шифрующей последовательности и расшифровать сообщения.

Данный режим обладает всеми достоинствами и недостатками алгоритмов поточного шифрования. К ограничениям следует отнести недопустимость повторного использования

nonce для шифрования нескольких блоков данных, т.к. это автоматически позволяет выявить позиции совпадающих и несовпадающих битов зашифрованных сообщений любому, кто перехватит шифр. Поэтому в режиме CTR каждый ключ шифрования может быть использован лишь ограниченное число раз и, если *nonce* выбирается в режиме счетчика, должен быть заменен при переполнении счетчика.

Режим GCM (Galois/Counter Mode) можно считать улучшенной версией режима CTR, добавляющей возможность проверки целостности и аутентификации передаваемых данных. Т.к. GCM основан на CTR, его также можно отнести к семейству алгоритмов поточного шифрования. Среди всех перечисленных режимов GCM является в настоящий момент наиболее используемым. Его распространению поспособствовало обнаружение в CBC вышеупомянутых уязвимостей, из-за чего в конечном итоге CBC был исключен из спецификации Transport Layer Security (TLS) в версии 1.3. Результатом этого стало то, что все присутствующие на сегодняшний день в спецификации TLS 1.3 шифры являются поточными [6].

Вторым важным семейством алгоритмов шифрования являются алгоритмы асимметричного шифрования, также известные как алгоритмы шифрования на основе открытого ключа. Их идея заключается в наличии пары ключей, один из которых является общедоступным и используется для шифрования данных, а второй держится в секрете и используется для расшифровки сообщений. Известнейшим примером алгоритма асимметричного шифрования является алгоритм RSA (аббревиатура от фамилий Rivest, Shamir и Adleman), основанный на высокой вычислительной сложности задачи разложения на множители больших полупростых чисел. Процедура формирования RSA-ключей выглядит следующим образом:

- случайно выбираются два простых числа p и q (длины 1024 бит и более);
- вычисляется модуль $n = pq$;
- вычисляется функция Эйлера модуля $\varphi(n) = (p-1)(q-1)$;
- выбирается публичный показатель степени e как любое число, взаимно простое с $\varphi(n)$. Рекомендуется использовать простые числа с небольшим количеством единичных битов в двоичной записи для ускорения операций возведения в степень. Например, 65537;
- из условия $ed \equiv 1(mod\ \varphi(n))$ вычисляется секретный показатель степени d. Ее существование обеспечивается выполнением предыдущего условия. Для нахождения может использоваться расширенный алгоритм Евклида;
- пара (e, n) назначается публичным ключом;
- пара (d, n) назначается секретным ключом.

Тогда операции шифрования и расшифровки для произвольного числа $0 \le m < n$ задаются следующими симметричными друг другу формулами:

$$c = Enc(m) = m^e\ (mod\ n)$$
$$m = Dec(c) = c^d\ (mod\ n)$$

Обоснование их работоспособности приведено в Приложении к статье. Криптографическая стойкость шифра обеспечивается трудностью вычисления множителей p и q по значению n, а следовательно, и трудностью подбора секретного показателя степени d.

Недостатки RSA схожи с недостатками алгоритмов блочного шифрования — шифр для повторяющихся исходных данных будет одинаков. По этой причине на практике обычно используется комбинированный подход, когда RSA используется на начальном этапе для защищенной передачи симметричного случайно формируемого сеансового ключа, с помощью которого уже осуществляется непосредственное шифрование данных.

Важным достоинством RSA в сравнении с алгоритмами симметричного шифрования является то, что отправителю данных не требуется заранее иметь собственную копию секретного ключа для обмена данными с получателем. Получателю достаточно прислать свой публичный ключ по любому открытому каналу данных. Тогда с его помощью отправитель сможет зашифровать данные для получателя и тем самым обеспечить их конфиденциальность. Однако при этом возникает дополнительная угроза, связанная невозможностью установления реального владельца секретного ключа, от которого был получен публичный ключ. Главной опасностью в этом сценарии являются атаки вида «человек посередине» (Man in the Middle; MITM), в которых злоумышленник:

- формирует собственную пару из публичного и секретного RSA-ключей;
- встраивается в канал между отправителем и получателем данных;
- перехватывает публичный ключ, которым должны шифроваться данные для получателя, при его передаче отправителю;
- подменяет перехваченный ключ своим собственным экземпляром публичного ключа, для которого ему известен секретный ключ.

В таком случае злоумышленник получает возможность расшифровки всех сообщений отправителя, т.к. они шифруются публичным ключом злоумышленника. При этом для отправителя этот факт может остаться незамеченным, т.к. злоумышленник способен самостоятельно перешифровать скомпрометированные данные пере-

хваченным публичным ключом исходного получателя и переслать их дальше. Для борьбы с этим недостатком был разработан метод защиты, основанный на использовании электронно-цифровой подписи (ЭЦП, далее — цифровая подпись), которая должна принадлежать третьей доверенной стороне.

4. Цифровая подпись

Цифровая подпись — это третий вид низкоуровневых криптографических операций, направленный прежде всего на обеспечение защиты целостности передаваемых данных. Выше при описании алгоритма асимметричного шифрования RSA подчеркивалось, что операции шифрования и расшифровки имеют идентичный друг другу вид, из которого следует, что публичный ключ RSA может быть использован не только для шифрования данных, но также с его помощью можно выполнять обратную операцию — расшифровку данных, зашифрованных при помощи секретного ключа. Если передаваемые в открытом виде исходные данные необходимо защитить от искажения или подмены при передаче, можно сделать это следующим образом:

- к передаваемому в открытом виде сообщению прикрепляется дополнительное поле, содержащее хэш-сумму передаваемых данных;
- хэш-сумма дополнительно шифруется секретным ключом владельца данных.

В этом случае любой обладатель публичного ключа при получении данных может самостоятельно вычислить хэш-сумму полученного сообщения, а затем сравнить ее с хэш-суммой, получаемой в результате расшифровки переданного вместе с сообщением дополнительного поля шифротекста. Совпадение значений будет свидетельствовать о том, что сообщение было получено от обладателя секретного ключа и доставлено получателю в неизменном виде.

Однако в случае атаки MITM (Man in the Middle) исходными данными являются сами публичные ключи сервисов сети Интернет, поэтому применить описанный выше алгоритм проверки подписи не представляется возможным и для решения задачи необходимо привлечение третьей стороны. В качестве одного из таких решений было предложено помещать публичные ключи каждого сервиса внутрь специализированного документа — сертификата, дополнительно хранящего информацию о владельце сервиса. При этом право выпуска сертификатов для регистрируемых в сети сервисов предоставляется только ограниченному кругу доверенных организаций — удостоверяющих центров (Certification authority, CA). Для защиты от подделки каждый сертификат заверяется электронной подписью удостоверяющего центра (или подписью нижестоящей в цепочке сертификации организации, сертификат которой заверен подписью удостоверяющего центра). При этом используется описанный выше алгоритм. Публичные ключи корневых удостоверяющих центров, необходимые для проверки подлинности сертификата, общедоступны и, как правило, изначально зашиты в дистрибутивы программ (браузеров), с помощью которых осуществляется доступ к информационным системам в сети Интернет. Таким образом, предустановленные публичные ключи удостоверяющих центров позволяют пользователям программ в автономном режиме проверять целостность публичных ключей внутри получаемых от сервисов сертификатов путем проверки их подписей, которые также должны принадлежать удостоверяющим центрам. Тем самым обеспечивается доверие к отправителю сертификата и исключается возможность подмены публичного ключа на этапе установления защищенного соединения. Подробно структуру сертификата описывает формат X.509 [16].

Детальное описание всех процедур, связанных с использованием алгоритмов асимметричного шифрования, можно найти в серии спецификаций Public-Key Cryptography Standards (PKCS), созданных корпорацией RSA. Среди них:

- PKCS #1 — описание алгоритма RSA и формата его ключей (RFC 8017 [17]);
- PKCS #3 — описание алгоритма Диффи-Хеллмана выработки общего секретного ключа [18];
- PKCS #7 — описание формата зашифрованного и/или подписанного криптографического сообщения (обычно представлен файлом с расширением .p7b) [19];
- PKCS #8 — описание формата секретного ключа (RFC 5958 [20]; обычно представлен файлом с расширением .key);
- PKCS #12 — описание формата экспорта секретного ключа вместе с сертификатом и путём сертификации (RFC 7292 [21]; обычно представлен файлом с расширением .pfx или .p12).

5. Программные реализации

Следует отметить, что даже при наличии готовых криптографических алгоритмов их практическая реализация остается весьма нетривиальной задачей. Как было отмечено в примере с Padding Oracle Attack, для взлома шифра может быть достаточно малейшего неучтенного канала данных. Даже несовпадения времени выполнения операции расшифровки блоков с корректным и некорректным дополнением служебными

хвостовыми байтами оказывается достаточно для проведения тайминговых атак и взлома алгоритма. По этой причине собственная разработка криптографических примитивов при отсутствии должного уровня квалификации часто нежелательна, а при использовании готовых решений следует тщательно подходить к выбору и полагаться только на проверенные реализации алгоритмов, предоставляемые сторонними библиотеками и программными продуктами.

Одним из важнейших примеров готовой реализации является библиотека OpenSSL [7], которая является библиотекой с открытым исходным кодом и известна прежде всего тем, что содержит в себе открытую реализацию протокола TLS. Помимо этого, библиотека поддерживает множество алгоритмов шифрования и криптографического хэширования, включая алгоритмы AES, ChaCha20, RSA и алгоритмы MD5, SHA-2, SHA-3, а также многие другие.

Другим известным примером является библиотека Networking and Cryptography Library (NaCl) [8], созданная Дэниэлом Бернштейном — автором многих известных криптографических алгоритмов. Ядром библиотеки NaCl выступают три основных элемента:

- эллиптическая кривая Curve25519, посредством которой реализуется разновидность протокола Диффи-Хеллмана выработки общего симметричного ключа (ECDH);

- потоковое шифрование на основе алгоритма Salsa20;

- функция Poly1305, используемая для аутентификации сообщений.

При создании библиотеки авторы ставили своей основной целью обеспечение высокой производительности при сохранении небольшого размера библиотеки. Также особое внимание уделено борьбе с тайминговыми атаками по типу описанной Padding Oracle Attack. Для их предотвращения при реализации библиотеки выбирались операции с фиксированным временем выполнения, не зависящим от подаваемых на вход алгоритмам данных.

Среди отечественных криптопровайдеров следует отметить КриптоПро CSP [10], выступающего в роли хранилища ключей и реализующего работу с ними через различные криптографические алгоритмы — хэширование, шифрование и формирование электронных подписей. Достоинством КриптоПро CSP является поддержка как зарубежных, так и отечественные криптографических алгоритмов, а также совместимость с большинством известных разновидностей ключевых носителей. В частности, поддерживаются отечественные алгоритмы криптографического хэширования ГОСТ Р 34.11-2012 (алгоритм «Стрибог»), блочного шифрования ГОСТ Р 34.12-2015 (алгоритмы «Кузнечик» и «Магма») [22] и формирования цифровой подписи ГОСТ Р 34.10-2012 [23].

Другим направлением развития средств криптографии является разработка специализированных библиотек и утилит, направленных на расширение условий применения базовых криптографических операций. Например, во всех в рассмотренных выше подходах к шифрованию неявно предполагалось, что доступ к данным в расшифрованном виде имеют только два участника информационного обмена — владелец данных, выступающий в роли отправителя, а также их получатель, обладающий копией секретного ключа шифрования. При этом шифрование в описанных подходах было прежде всего необходимо для безопасной передачи данных от отправителя к получателю, а вопрос хранения данных отдельно не рассматривался. Часто, когда исходные данные обладают высоким уровнем конфиденциальности, их хранение в открытом виде оказывается нежелательным, поэтому для защиты от утечки данных также применяют шифрование. Если появляется необходимость предоставить дополнительный доступ к зашифрованным данным, существует несколько подходов к реализации этого требования:

- расшифровать исходные данные, чтобы повторно зашифровать их ключом получателя;

- зашифровать ключом получателя ключ, позволяющий расшифровать данные, и передать его вместе с зашифрованными данными (данный подход принято называть envelope encryption).

Развитием второго подхода служит формат Global Alliance for Genomics and Health (GA4GH [9]), разработанный для безопасного хранения и передачи генетических данных пациентов и масштабирующий предложенную концепцию на случай большего количества получателей данных. В GA4GH сообщения имеют блочную структуру и состоят из блоков заголовков (по количеству получателей данных) и блоков зашифрованных данных. Упрощенно формат расшифровки сообщения в GA4GH выглядит следующим образом:

- отправитель и все получатели сообщения имеют собственную пару из публичного и секретного ключей схемы асимметричного шифрования. Публичные ключи получателей предварительно передаются отправителю для шифрования;

- получатель сообщения поочередно перебирает заголовки сообщения в поисках предназначающегося ему заголовка. Каждый заголовок содержит в себе публичный ключ отправителя и блок зашифрованных данных. При обнаружении заголовка, данные которого получится расшиф-

ровать, процедура перебора приостанавливается;

- для расшифровки заголовка необходим симметричный ключ. Для его вычисления используется алгоритм Диффи-Хеллмана выработки общего секретного ключа. На вход алгоритму передаются публичный ключ отправителя из заголовка и секретный ключ получателя (для усиления безопасности в GA4GH значение общего ключа дополнительно хэшируется вместе со значениями публичных ключей по алгоритму Blake2b; см. [9], [24]);

- целостность данных и правильность их расшифровки контролируются с помощью Message Authentication Code (MAC) на основе функции функция Poly1305;

- в случае успеха из заголовка извлекается ключ шифрования данных, с помощью которого расшифровываются блоки передаваемых данных сообщения.

Следует отметить, что получатели данных имеют возможность не только расшифровать данные, но также добавить к сообщению новых получателей, для чего им достаточно знать публичный ключ нового получателя. В этом случае к сообщению добавляется новый блок заголовка, внутрь которого помещается публичный ключ того, кто этот заголовок добавляет.

Одной из известных реализаций формата GA4GH является утилита Crypt4GH на языке программирования Python.

6. Заключение

В работе рассмотрены примеры популярных низкоуровневых криптографических операций. На этих примерах продемонстрированы основные идеи, благодаря которым обеспечивается выполнение требований безопасности.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).

Приложение

Покажем корректность задания операций шифрования и расшифровки в алгоритме RSA с помощью формул

$$c = Enc(m) = m^e \ (mod\ n)$$
$$m = Dec(c) = c^d \ (mod\ n)$$

Требуется доказать, что для любого $0 \le m < n$ выполнено следующее тождество:

$$Dec\big(Enc(m)\big) = m^{ed} \equiv m \ (mod\ n)$$

Доказательство этого утверждения опирается на использование нескольких вспомогательных теорем.

Теорема 1 (малая теорема Ферма). *Пусть p — простое число и a — натуральное число, которое не делится на p. Тогда*

$$a^{p-1} \equiv 1 \ (mod\ p)$$

Теорема 2 (китайская теорема об остатках). *Пусть $a_1, a_2, \dots, a_n$ — набор попарно взаимно простых натуральных чисел. Пусть также $r_1, r_2, \dots, r_n$ — набор целых чисел таких, что $0 \le r_i \le a_i$ для всех $i \in \{1, 2, \dots, n\}$. Тогда существует такое натуральное N, что для любого $i \in \{1, 2, \dots, n\}$*

$$N \equiv r_i \ (mod\ a_i)$$

При этом, если найдутся два таких числа N_1 и N_2, то

$$N_1 \equiv N_2 \ (mod\ a_1 \cdot a_2 \cdot \dots \cdot a_n)$$

Следствие 1. *Пусть есть два натуральных числа N и M такие,* что *для любого $i \in \{1, 2, \dots, n\}$*

$$N \equiv M \ (mod\ a_i)$$

где $a_1, a_2, \dots, a_n$ — набор попарно взаимно простых натуральных чисел. Тогда

$$N \equiv M \ (mod\ a_1 \cdot a_2 \cdot \dots \cdot a_n)$$

Перейдем к доказательству утверждения. Из условия $ed \equiv 1(mod\ \varphi(n))$ имеем, что для некоторого целого *k* выполнено тождество

$$ed = 1 + k(p-1)(q-1)$$

Тогда

$$m^{ed} = m(m^{p-1})^{k(q-1)}$$

Если *m* не делится на простое число *p*, из малой теоремы Ферма имеем, что

$$m^{p-1} \equiv 1(mod\ p)$$

Отсюда получаем, что

$$m^{ed} \equiv m\ (mod\ p)$$

Если m делится на p, также имеем

$$m^{ed} \equiv 0\ \equiv m\ (mod\ p)$$

Аналогично для простого числа q имеем

$$m^{ed} \equiv m\ (mod\ q)$$

Отсюда по следствию из китайской теоремы об остатках для $n = pq$ получаем

$$m^{ed} \equiv m\ (mod\ n)$$

■

Low Level Cryptographic Operations

N. D. Baykov, A. N. Godunov

Abstract. The goal of this manuscript is to provide an overview of the basic low level cryptographic operations at the heart of modern cryptographic protocols. We consider the examples of some widely used operations of cryptographic hashing, encryption and the digital signature calculation.

Keywords: cryptographic hash function, encryption, digital signature

Литература

1. R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321. 1992. url: https://www.rfc-editor.org/info/rfc1321 (дата обращения 13 октября 2023). doi: 10.17487/RFC1321.

2. T. Hansen. US Secure Hash Algorithms (SHA and HMAC-SHA). RFC 4634. 2006. url: https://datatracker.ietf.org/doc/html/rfc4634 (дата обращения 13 октября 2023). doi: 10.17487/RFC4634.

3. M.J. Dworkin. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Federal Inf. Process. Stds. (NIST FIPS) – 202. 2015. doi: 10.6028/NIST.FIPS.202.

4. ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования». ФГУП «СТАНДАРТИНФОРМ». Москва. 2018.

5. ГОСТ Р 34.11-2012 «Информационная технология. Криптографическая защита информации. Функция хэширования». ФГУП «СТАНДАРТИНФОРМ». Москва. 2013.

6. E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.3. RFC 8446. 2018. url: https://www.rfc-editor.org/info/rfc8446 (дата обращения 13 октября 2023). doi: 10.17487/RFC8446.

7. OpenSSL. url: https://www.openssl.org (дата обращения 13 октября 2023).

8. NaCl: Networking and Cryptography library. url: http://nacl.cr.yp.to (дата обращения 13 октября 2023).

9. GA4GH File Encryption Standard. 2019. url: http://samtools.github.io/hts-specs/crypt4gh.pdf (дата обращения 13 октября 2023).

10. КриптоПро CSP. url: https://www.cryptopro.ru/products/csp (дата обращения 13 октября 2023).

11. D.J. Bernstein. The Salsa20 Family of Stream Ciphers. New Stream Cipher Designs. Lecture Notes in Computer Science. V 4986 (2008). 84-97. doi: 10.1007/978-3-540-68351-3_8.

12. Y. Nir, A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 8439. 2018. url: https://www.rfc-editor.org/info/rfc8439 (дата обращения 13 октября 2023). doi: 10.17487/RFC8439.

13. Advanced Encryption Standard. Federal Inf. Process. Stds. (NIST FIPS) - 197. 2001. doi: 10.6028/NIST.FIPS.197.

14. D. Blazhevski, A. Bojinovski, B. Stojcevska, V. Pachovski. Modes of Operation of the AES Algorithm. “Third International Conference on Computational Intelligence and Information Technology (CIIT 2013)”, India, Mumbai, 2013. url: https://www.researchgate.net/publication/236656798_MODES_OF_OPERATION_OF_THE_AES_ALGORITHM (дата обращения 13 октября 2023).

15. Heaton R. The Padding Oracle Attack. 2013. url: https://robertheaton.com/2013/07/29/padding-oracle-attack/ (дата обращения 13 октября 2023).

16. D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280. 2008. url: https://www.rfc-editor.org/info/rfc5280 (дата обращения 13 октября 2023). doi: 10.17487/RFC5280.

17. K.Ed. Moriarty, B. Kaliski, J. Jonsson, A. Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017. 2016. url: https://www.rfc-editor.org/info/rfc8017 (дата обращения 13 октября 2023). doi: 10.17487/RFC8017.

18. R. Merkle. Secure Communications Over Insecure Channels. Communications of the ACM. V. 21 (1978). № 4. doi:10.1145/359460.359473.

19. R. Housley. Cryptographic Message Syntax (CMS). RFC 5652. 2009. url: https://www.rfc-editor.org/info/rfc5652 (дата обращения 13 октября 2023). doi: 10.17487/RFC5652.

20. S. Turner. Asymmetric Key Packages. RFC 5958. 2010. url: https://www.rfc-editor.org/info/rfc5958 (дата обращения 13 октября 2023). doi: 10.17487/RFC5958.

21. K.Ed. Moriarty, M. Nystrom, S. Parkinson, A. Rusch, M. Scott. PKCS #12: Personal Information Exchange Syntax v1.1. RFC 7292. 2014. url: https://www.rfc-editor.org/info/rfc7292 (дата обращения 13 октября 2023). doi: 10.17487/RFC7292.

22. ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры». ФГУП «СТАНДАРТИНФОРМ». Москва. 2016.

23. ГОСТ Р 34.10-2012 «Информационная технология. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи». ФГУП «СТАНДАРТИНФОРМ». Москва. 2012.

24. M-J. Ed. Saarinen, J-P. Aumasson. The BLAKE2 Cryptographic Hash and Message Authentication Code (MAC). RFC 7693. 2015. url: https://www.rfc-editor.org/info/rfc7693 (дата обращения 13 октября 2023). doi: 10.17487/RFC7693.