

Оглавление

Предисловие	6
Введение	7
Глава 1. Алгоритмы хеширования	9
1.1 Основные понятия и определения	10
1.1.1 Структура алгоритмов хеширования	10
1.1.2 Настройки над алгоритмами хеширования.....	14
1.2 Методы криптоанализа и атаки на алгоритмы хеширования.....	18
1.2.1 Цели атак на алгоритмы хеширования	19
1.2.2 Атаки методом «грубой силы»	21
1.2.3 Словарные атаки и цепочки хеш-кодов	22
1.2.4 Радужные таблицы.....	26
1.2.5 Парадокс «дней рождения» и поиск коллизий	27
1.2.6 Дифференциальный криптоанализ	30
1.2.7 Алгебраический криптоанализ.....	34
1.2.8 Атаки, использующие утечки данных по побочным каналам.....	35
1.2.9 Другие виды атак	35
1.3 Наиболее известные алгоритмы хеширования	38
1.3.1 Алгоритмы семейства MD	38
1.3.2 Алгоритмы семейства RIPEMD	59
1.3.3 Алгоритмы семейства SHA.....	69
1.3.4 Отечественные стандарты хеширования.....	84
Глава 2. Алгоритмы электронной подписи на эллиптических кривых	91
2.1 Математические основы	91
2.2 Эллиптические кривые.....	95
2.2.1 Определение эллиптической кривой	95
2.2.2 Основные операции над точками эллиптической кривой	96
2.2.3 Основные характеристики эллиптической кривой.....	99
2.2.4 Примеры эллиптических кривых	101
2.2.5 Задача дискретного логарифмирования в группе точек эллиптической кривой.....	105
2.2.6 Альтернативные формы представления эллиптических кривых ...	107
2.3 Основные алгоритмы электронной подписи	111
2.3.1 Алгоритм ECDSA	111
2.3.2 ГОСТ Р 34.10–2012	112

2.3.3 Некоторые особенности алгоритмов ECDSA и ГОСТ Р 34.10–2012	114
2.3.4 Алгоритм EdDSA.....	116
2.3.5 Алгоритм BLS	118

Глава 3. Основные принципы работы блокчейн-технологий... 122

3.1 Базовые механизмы блокчейн-систем.....	123
3.1.1 Транзакции.....	123
3.1.2 Упаковка транзакций в блоки	127
3.1.3 Применение деревьев Меркля при формировании блоков.....	130
3.2 Механизмы консенсуса	131
3.2.1 Консенсус доказательства работы Proof of Work	131
3.2.2 Консенсус доказательства владения долей Proof of Stake.....	135
3.2.3 Консенсус на основе решения задачи византийских генералов ...	136
3.2.4 Другие механизмы достижения консенсуса	137
3.3 Выстраивание цепочки блоков	139
3.3.1 Принципы формирования цепочки	139
3.3.2 Ветвления цепочки блоков.....	142
3.4 Смарт-контракт.....	145
3.5 Основные виды блокчейн-систем	148
3.5.1 Публичный блокчейн.....	148
3.5.2 Приватный блокчейн.....	149
3.6 Криптовалютные кошельки	150
3.6.1 Программы-кошельки.....	150
3.6.2 Аппаратные кошельки.....	152

Глава 4. Основные блокчейн-платформы..... 153

4.1 Биткойн	153
4.1.1 Введение в устройство блокчейн-системы Биткойн	154
4.1.2 Особенности механизма консенсуса в системе Биткойн.....	156
4.1.3 Форки в системе Биткойн	156
4.1.4 Транзакции.....	159
4.1.5 Кошельки в системе Биткойн.....	203
4.1.6 Создание и использование иерархических детерминированных ключей.....	206
4.2 Эфириум	209
4.2.1 Глобальное состояние	209
4.2.2 Консенсус.....	210
4.2.3 Газ.....	211
4.2.4 Адреса и кошельки.....	212
4.2.5 Транзакции.....	213
4.2.6 Структура блока	213
4.2.7 Эволюция системы Эфириум.....	214
4.2.8 Основная и тестовые сети платформы Эфириум	218
4.2.9 Запуск сети Эфириум.....	219
4.2.10 Смарт-контракты в системе Эфириум	234
4.3 Hyperledger	245

4.3.1 Основные особенности системы	245
4.3.2 Проекты экосистемы Hyperledger	246
4.3.3 Архитектура Hyperledger Fabric	247
4.3.4 Пример смарт-контракта для Hyperledger	248
4.4 Обзор других платформ	253
4.4.1 EOSIO	253
4.4.2 Краткий обзор прочих блокчейн-платформ.....	255
4.4.3 Обзор отечественных решений	257
Приложение 1. Таблицы констант алгоритмов хеширования	261
П1.1 Таблица замен алгоритма MD2	261
П1.2 Индексы используемых в итерациях слов блока сообщения алгоритма MD4.....	262
П1.3 Константы алгоритма MD5	263
П1.4 Константы алгоритма MD6	267
П1.5 Константы алгоритмов семейства SHA-2	268
П1.6 Раундовые константы алгоритмов семейства SHA-3	270
П1.7 Константы алгоритма ГОСТ Р 34.11–2012	271
Список сокращений	275
Перечень рисунков	281
Перечень таблиц	286
Перечень источников	289

Предисловие

В последние десятилетия криптографические методы проникают в самые различные сферы нашей жизнедеятельности, связанные с передачей, обработкой и хранением информации. Цепная запись данных, распределенные реестры, интернет вещей, облачные вычисления в той или иной мере используют криптографические алгоритмы и протоколы. Появление криптовалюты биткойн привлекло внимание научного сообщества к технологии блокчейн, или технологии цепной записи данных. Последовала разработка блокчейн-платформ для использования в самых различных областях: финансовой и банковской сфере, медицине, торговле и т. п. При этом успешное применение технологий, основанных на криптографии, невозможно без понимания математических основ, на которых базируются криптографические алгоритмы, используемые в данной технологии.

Приведенная в этой книге информация позволяет получить представление по наиболее важным вопросам построения блокчейн-платформ, таким как принципы построения и использования функций хеширования, схем электронной подписи на основе эллиптических кривых, механизмов консенсуса. Приведенные математические основы используемых в технологии блокчейн криптографических алгоритмов помогают глубже понять заложенные в ней механизмы обеспечения безопасности информации. Данная книга может быть интересна самому широкому кругу читателей, желающих понять принципы построения и использования блокчейн-технологий.

*Сергей Васильевич Матвеев,
эксперт ТК-26 «Криптографическая защита информации»*

Введение

С момента появления блокчейн-технологий прошло менее 15 лет, но их активное развитие в течение этого времени предопределило вхождение данных технологий в весьма различные сферы деятельности.

Основной сферой применения блокчейн-технологий можно считать финансовую: они лежат в основе криптовалют, которые, похоже, уже достаточно прочно вошли в нашу жизнь. Буквально в последний год мы могли наблюдать всплеск интереса к криптовалютам после многократного удорожания основной из них – биткойна – в течение 2020 – начала 2021 года.

При этом постоянно модернизируются и совершенствуются как сама платформа Биткойн и лежащая в ее основе блокчейн-система, так и данные технологии в принципе. Их развитие способствует и развитию множества смежных технологий и направлений: от криптографических алгоритмов до вычислительных ресурсов, применяемых пользователями подобных систем.

Помимо финансового сектора, блокчейн-технологии востребованы в различных системах государственных организаций, в частности:

- ведение различных реестров, например государственной регистрации прав на недвижимое имущество, землю и т. п.;
- выпуск цифровых удостоверений личности на основе блокчейн-технологий;
- удаленное голосование, опробованное, в частности, в России в 2019 и 2020 годах.

Можно предполагать, что в дальнейшем приведенный выше, далеко не полный перечень применений блокчейнов будет только расширяться.

Отметим, что распределенные реестры были известны и ранее, но именно блокчейн-технологии, обеспечивающие, с одной стороны, возможность модификации общих данных различными пользователями распределенных систем и, с другой стороны, контроль целостности и непротиворечивости данных на основе определенных правил, предопределили масштабное развитие подобных технологий и появление таких принципиально новых направлений, как криптовалюты.

Про блокчейн-технологии, особенно в части их применений в криптовалютах, издано достаточно много книг. Специфика этой книги в том, что при ее создании мы изначально ставили своей целью рассмотреть и проанализировать именно криптографические механизмы, лежащие в основе блокчейн-технологий и предопределяющие их основные качества, способствующие столь бурному развитию и предполагаемому в будущем широчайшему применению данных технологий.

Понимая и разделяя интерес многих потенциальных читателей к криптовалютам, мы не обошли их стороной, но рассмотрели именно с точки зрения реализованных в них криптоалгоритмов и прочих методов, обеспечивающих технические составляющие безопасного использования криптовалют.

В первой главе книги описаны алгоритмы хеширования, обеспечивающие контроль целостности данных в блокчейне. Рассмотрены основные принципы данных алгоритмов, возможные проблемы при их реализации и использовании, включая известные атаки на алгоритмы хеширования. Приведено подробное описание наиболее известных алгоритмов хеширования, включая используемые в распространенных блокчейн-платформах.

Вторая глава также посвящена криптографическим алгоритмам – на этот раз алгоритмам электронной подписи, являющимся одним из важнейших элементов, обеспечивающих связь в цепочках данных блокчейна, и не только. Рассмотрены эллиптические кривые, лежащие в основе современных алгоритмов электронной подписи и наиболее часто применяемые из данных алгоритмов.

Третья глава описывает базовые механизмы построения цепочек данных – основы блокчейн-технологий. Значительная часть главы посвящена описанию различных механизмов достижения консенсуса, легитимизирующих действия пользователей с данными блокчейна.

Наконец, в последней главе рассмотрены примеры построения блокчейн-платформ на основе алгоритмов и методов, описанных в предыдущих главах. В частности, дано подробное описание системы Биткойн, обеспечивающей оборот одноименной криптовалюты, наиболее широко используемой в мире.

Надеемся, что изложенная в нашей книге информация оправдает ваши ожидания от книги, окажется интересной и принесет пользу в вашей деятельности.

Авторы выражают глубокую признательность известному специалисту по прикладной криптографии Олегу Геннадьевичу Тараскину (компания Waves) за предоставленную для публикации в данной книге главу 2, без материала которой книга была бы неполной, а также за множество полезных замечаний, позволивших значительно улучшить книгу.

Авторы также благодарны эксперту технического комитета по стандартизации «Криптографическая защита информации» (ТК-26) Сергею Васильевичу Матвееву за предисловие к книге и ценные замечания по ее материалу.

Будем рады вашим письмам по изложенным в книге вопросам, а также замечаниям к содержанию книги и предложениям по ее возможному усовершенствованию. Адреса электронной почты для связи с авторами:

serg@panasenko.ru (Сергей Панасенко) и

jekky82@mail.ru (Евгения Ищукова).

Глава 1

Алгоритмы хеширования

Алгоритмы хеширования (или функции хеширования, хеш-функции) позволяют по определенным правилам выполнить свертку входных данных произвольной длины в битовую строку фиксированного размера, называемую хеш-кодом [15] (распространены также термины «хеш» или «хеш-значение»).

Фактически хеш-функции выполняют контрольное суммирование данных, которое может происходить как с участием некоего секретного ключа, так и без него. Обычно алгоритмы ключевого хеширования представляют собой надстройки над алгоритмами хеширования, не использующими ключ. Однако существуют и такие хеш-функции, которые изначально разрабатывались с учетом использования секретного ключа в качестве дополнительного параметра преобразований.

Такое контрольное суммирование достаточно широко применяется в области защиты компьютерной информации, в том числе:

- для подтверждения целостности данных;
- для свертки данных перед вычислением или проверкой их электронной подписи;
- в различных протоколах аутентификации пользователей;
- в процедурах генерации псевдослучайных последовательностей и производных ключей.

Алгоритмы хеширования применяются и с другими целями, не относящимися к задачам защиты информации. В частности, они применяются для вычисления уникальных идентификаторов данных и построения на их основе хеш-таблиц, существенно ускоряющих поиск требуемых данных в больших массивах. Однако в подобных случаях к алгоритмам хеширования предъявляются иные требования, чем при их криптографических применениях.

Алгоритмы хеширования активно используются в блокчейн-технологиях, наиболее часто – для свертки данных перед их подписанием электронной подписью. В ряде случаев хеш-функции используются и с другими целями – например, в биткойне для подтверждения проделанной работы используются найденные (получаемые путем перебора) хеш-коды специального формата – с количеством лидирующих битовых нулей не менее заданного.

Конкретные применения алгоритмов хеширования в технологиях блокчейн-на будут описаны в главах 3 и 4, а в данной главе рассмотрим подробно структуру алгоритмов хеширования, предъявляемые к ним требования, а также основные методы и результаты их криптоанализа.

1.1 ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ

Функции хеширования позволяют выполнить однонаправленное преобразование входного массива произвольного размера в выходную битовую строку (хеш-код) фиксированного размера.

Криптографические хеш-функции должны обладать, как минимум, следующими свойствами [44, 202].

1. Хеш-код сообщения должен однозначно соответствовать сообщению и должен изменяться при любой модификации сообщения.
2. Должно быть вычислительно сложно найти прообраз, т. е. такое сообщение M , хеш-код которого был бы равен заданному значению h :

$$h = f(M),$$

где $f()$ – функция хеширования.

3. Должно быть вычислительно сложно найти второй прообраз, т. е. такое сообщение M_2 , хеш-код которого был бы равен хеш-коду заданного сообщения M_1 :

$$f(M_1) = f(M_2).$$

4. Должно быть вычислительно сложно найти коллизию, т. е. такие два сообщения M_1 и M_2 , хеш-коды которых были бы эквивалентны.

1.1.1 Структура алгоритмов хеширования

Хотя все множество алгоритмов хеширования достаточно разнообразно по структурам их формирования, наиболее известные алгоритмы хеширования основаны на нескольких типовых структурах, которые рассмотрим в данном разделе далее.

Схема Меркля–Дамгорда

Многие из широко используемых алгоритмов хеширования имеют схожую между собой структуру, которая была предложена в 1988–1989 гг. независимо двумя известными криптологами: Ральфом Мерклем (Ralph Merkle) [177] и Айвеном Дамгордом (Ivan Damgård) [102]. Она получила название «схема Меркля–Дамгорда» (Merkle-Damgård construction) – см. рис. 1.1.

В соответствии со схемой Меркля–Дамгорда работают, в частности, алгоритмы MD4 [210], MD5 [212] и SHA [125], а также отечественный стандарт хеширования ГОСТ Р 34.11–2012 [15], которые будут подробно рассмотрены в этой главе далее.

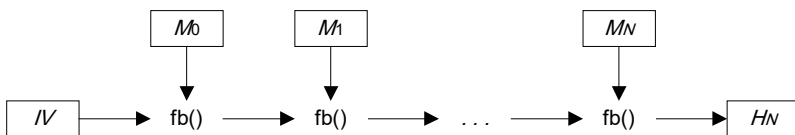


Рисунок 1.1. Схема Меркля–Дамгорда

Хешируемое сообщение M разбивается на блоки определенной длины (например, по 512 байт в алгоритме SHA-1) $M_0 \dots M_N$. Выполняется (по-разному

в различных алгоритмах) дополнение сообщения M до размера, кратного данной длине блока (при этом количество блоков может увеличиться – см., например, описание алгоритма SHA далее).

Каждый i -й блок сообщения обрабатывается функцией сжатия $fb()$ (функция сжатия является криптографическим ядром алгоритма хеширования), причем данная функция накладывает результат этой обработки на текущий (промежуточный) хеш-код H_{i-1} (т. е. результат обработки предыдущих блоков сообщения):

$$H_i = fb(H_{i-1}, M_i).$$

Результатом работы алгоритма хеширования $hash()$ (т. е. хеш-кодом сообщения M) является значение H_N :

$$hash(M) = H_N.$$

Начальное значение H_{-1} обычно является константным и различным в разных алгоритмах хеширования; оно часто обозначается как IV (от Initialization Vector – вектор инициализации).

Алгоритмы хеширования на основе криптографической губки

Схема алгоритмов хеширования на основе «криптографической губки» (cryptographic sponge) была предложена рядом криптологов в работе [69]. Авторами данной схемы являются Гвидо Бертони (Guido Bertoni), Джоан Деймен (Joan Daemen), Михаэль Петерс (Michaël Peeters) и Жиль Ван Аске (Gilles Van Assche).

Алгоритмы хеширования, использующие конструкцию криптографической губки, содержат две фазы преобразований (рис. 1.2):

- «впитывание» (absorbing) – поблочная обработка входного сообщения, в процессе которой внутреннее состояние алгоритма хеширования «впитывает» свертку данных очередного блока;
- «выжимание» (squeezing) – сжатие внутреннего состояния для извлечения из него результирующего хеш-кода.

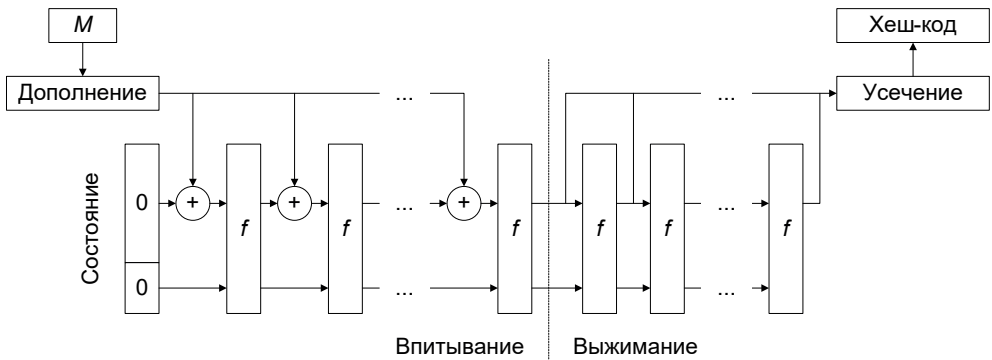


Рисунок 1.2. Фазы преобразований криптографической губки

Конструкция криптографической губки, по мнению ее авторов, имеет как минимум следующие преимущества [69, 71]:

- возможность точного и доказуемого определения минимальной трудоемкости основных атак на криптоалгоритмы, основанные на данной конструкции;
- относительная простота криптоалгоритмов;
- легкое построение алгоритмов с переменным размером входных и выходных данных;
- универсальность конструкции – с ее помощью можно создавать криптоалгоритмы различного назначения;
- легкое увеличение расчетного уровня криптостойкости алгоритма за счет изменения его параметров и снижения его быстродействия.

Надстройки над алгоритмами блочного шифрования

Существуют также варианты создания стойких однонаправленных хеш-функций на основе алгоритмов блочного шифрования. В частности, в работе [202] подробно описаны 12 алгоритмов, представляющих собой хеширующие надстройки над блочными шифрами.

Рассмотрим две из таких надстроек более подробно. Первая из них – это алгоритм Матиаса–Мейера–Осиса (Matyas-Meyer-Oseas – ММО), функция сжатия которого представлена на рис. 1.3. В данном алгоритме нижележащий блочный шифр используется следующим образом [174]:

- блок хешируемых данных M_i подается на вход алгоритма шифрования в качестве открытого текста (P);
- ключом блочного шифра (K) служит текущее состояние алгоритма хеширования H_{i-1} , обработанное некоторой функцией $g()$ (см. далее);
- на результат шифрования (C) блока M_i на ключе $g(H_{i-1})$ накладывается операцией XOR сам блок M_i , в результате чего получается новое состояние алгоритма хеширования H_i .

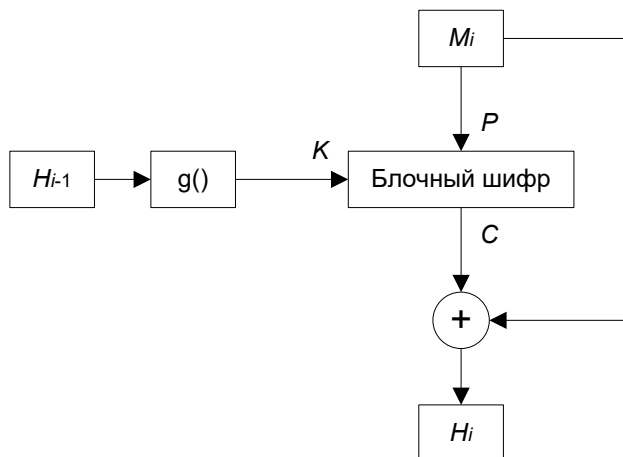


Рисунок 1.3. Схема алгоритма ММО

Поскольку размеры блока шифруемых данных и ключа алгоритма шифрования могут быть различными (и чаще всего они действительно различаются),

а размер состояния алгоритма хеширования является фиксированным, в алгоритме ММО существует необходимость в функции $g()$, задача которой состоит лишь в адаптации значения H_{i-1} под требуемый размер ключа шифрования. Так как функция $g()$ не влияет на безопасность алгоритма, она может быть максимально простой и, например, выполнять только дополнение H_{i-1} до требуемого размера [246].

Формально алгоритм ММО представляется так:

$$H_i = E(M_i, g(H_{i-1})) \oplus M_i,$$

где $E(P, K)$ – блочный шифр, шифрующий открытый текст P на ключе K .

Существует также вариант алгоритма ММО, который вместо операции XOR использует сложение 32-битовых субблоков M_i и C по модулю 2^{32} . В работе [217] такой вариант называется предпочтительным, в частности для алгоритмов MD4 и MD5.

В качестве блочного шифра может использоваться также функция сжатия какого-либо алгоритма хеширования, поскольку, как и блочный шифр, функции сжатия алгоритмов хеширования обычно принимают на вход два параметра (состояние и хешируемый блок данных вместо ключа шифрования и шифруемого блока), выдавая в качестве результата модифицированное состояние. В этом случае описанные в данном разделе схемы можно рассматривать как надстройки, усиливающие криптографические свойства нижележащего алгоритма хеширования.

Еще один вариант формирования алгоритма хеширования на основе блочного шифра – алгоритм Мягучи–Пренеля (Miyaguchi-Preneel), который отличается от предыдущего только тем, что предыдущее состояние алгоритма хеширования также участвует в операции XOR с блоком сообщения и результатом его зашифрования (рис. 1.4) [174]:

$$H_i = E(M_i, g(H_{i-1})) \oplus M_i \oplus H_{i-1}.$$

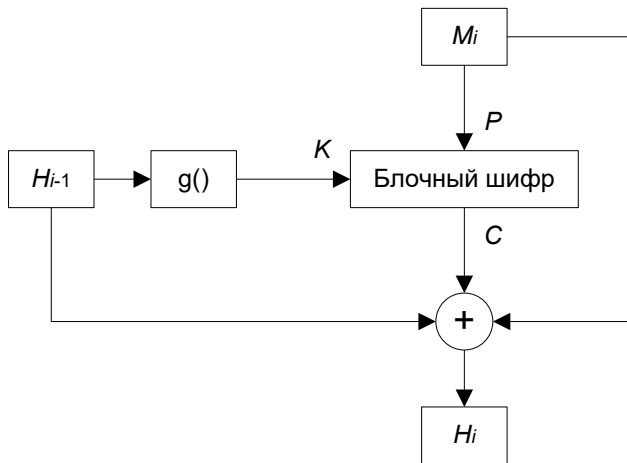


Рисунок 1.4. Схема алгоритма Мягучи–Пренеля

Аналогично предыдущей схеме, здесь также возможна (и рекомендуется для тех же алгоритмов MD4 и MD5 [217]) замена операции XOR на операцию сложения субблоков операндов по модулю 2^{32} .

1.1.2 Надстройки над алгоритмами хеширования

В свою очередь, алгоритмы хеширования достаточно часто используются в качестве основы для различных кодов аутентификации сообщений (Message Authentication Codes – MAC).

Коды аутентификации сообщений на основе алгоритмов хеширования и их варианты

Код аутентификации сообщения фактически представляет собой криптографическую контрольную сумму сообщения. MAC вычисляется на основе ключа и текста сообщения произвольной длины и используется для проверки целостности сообщения [246]. Для вычисления MAC используются ключевые алгоритмы хеширования (или ключевые надстройки над алгоритмами хеширования), алгоритмы блочного шифрования или специальные алгоритмы вычисления кодов аутентификации сообщений.

В отличие от алгоритмов электронной подписи (ЭП), в которых для вычисления ЭП используется закрытый ключ, а для проверки ЭП – вычисляемый из закрытого открытый ключ, при вычислении и проверке MAC применяется один и тот же секретный ключ. Следовательно, в отличие от ЭП, MAC не используется для установления авторства сообщения, поскольку секретный ключ принадлежит более, чем одному пользователю.

Схема использования MAC приведена на рис. 1.5.

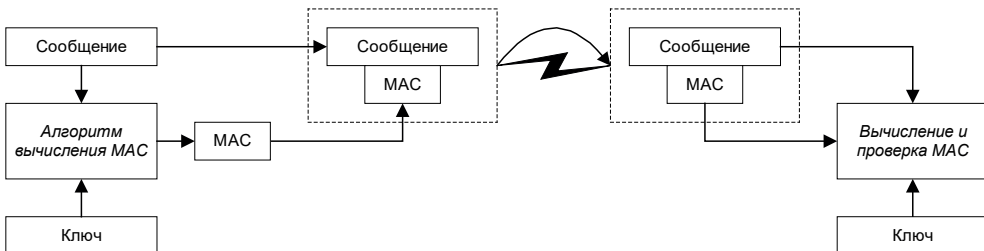


Рисунок 1.5. Схема использования MAC

Простейшие варианты создания MAC на основе произвольного бесключевого алгоритма хеширования выглядят так (при этом все три описанных далее варианта были признаны небезопасными) [202, 203]:

1. «Секретный префикс» (secret prefix):

$$\text{MAC}(k, m) = \text{hash}(k || m),$$

где:

- k – ключ;
- m – сообщение;
- $\text{hash}()$ – нижележащая бесключевая хеш-функция.

2. «Секретный суффикс» (secret suffix):

$$\text{MAC}(k, m) = \text{hash}(m \parallel k).$$

3. «Конверт» (envelope):

$$\text{MAC}(k_1, k_2, m) = \text{hash}(k_1 \parallel m \parallel k_2),$$

где подключи k_1 и k_2 являются различными.

Более безопасным способом создания MAC на основе алгоритмов хеширования является алгоритм HMAC (Hash-based Message Authentication Code – код аутентификации сообщения на основе хеширования), который позволяет вычислять хеш-коды с использованием некоего секретного ключа с помощью практически произвольного бесключевого алгоритма хеширования [163].

В основе алгоритма HMAC лежит функция хеширования, которая позволяет вычислить код аутентификации сообщения следующим образом (см. рис. 1.6):

1. Размер ключа выравнивается с размером блока используемого алгоритма хеширования:
 - если ключ key длиннее блока, он укорачивается путем применения к нему используемого алгоритма хеширования hash (этот вариант не показан на рис. 1.6):

$$key = \text{hash}(key);$$

отметим, что размер выходного значения алгоритма хеширования обычно много меньше размера блока хешируемых данных – например, соответственно, 128 и 512 бит для алгоритма MD4; в [163] рекомендуется минимальный размер ключа, равный размеру выходного значения алгоритма хеширования;

- если размер ключа меньше размера блока, то выровненный ключ k получается путем дополнения до размера блока нулевыми битами исходного (или укороченного) ключа key .
2. Выровненный ключ k складывается по модулю 2 с константой $C1$, которая представляет собой блок данных, заполненный байтами с шестнадцатеричным значением 36; аналогичным образом ключ k также складывается с константой $C2$, которая представляет собой блок данных, заполненный байтами с шестнадцатеричным значением 5C:

$$ki = k \oplus C1;$$

$$ko = k \oplus C2.$$

3. Вычисляется хеш-код t от результата конкатенации модифицированного ключа ki и сообщения m :

$$t = \text{hash}(ki \parallel m).$$

4. Выходным значением алгоритма HMAC является хеш-код от результата конкатенации модифицированного ключа ko и полученного на предыдущем шаге значения t :

$$hmac = \text{hash}(ko \parallel t).$$

Таким образом, при вычислении HMAC используемый алгоритм хеширования применяется дважды; каждый раз с участием модифицированного ключа. Размер выходного значения алгоритма HMAC равен размеру выходного значения алгоритма хеширования, а общая формула вычисления HMAC (без учета выравнивания ключа) выглядит следующим образом:

$$\text{HMAC}(k, m) = \text{hash}((k \oplus C2) \parallel \text{hash}((k \oplus C1) \parallel m)).$$

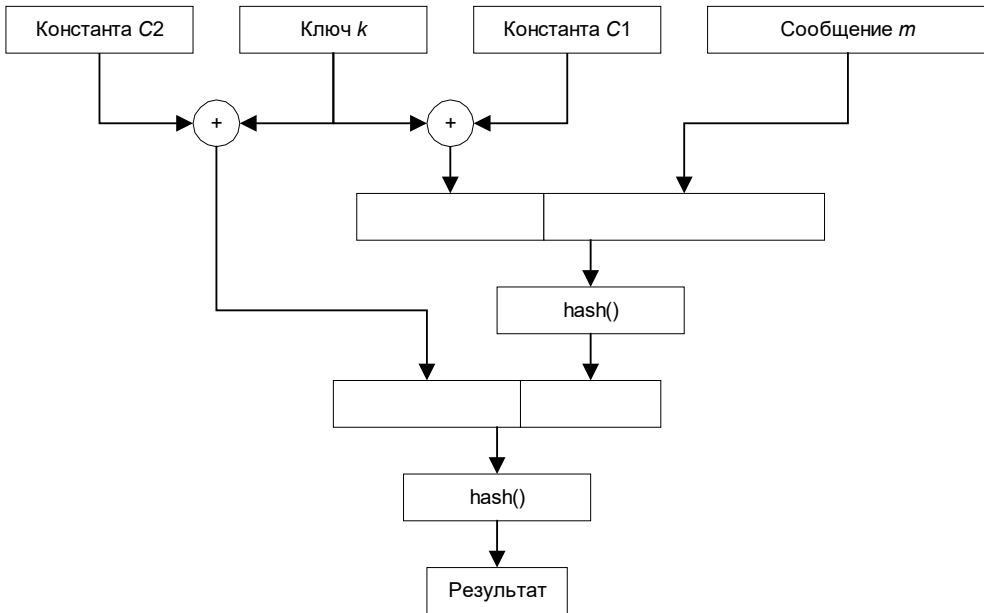


Рисунок 1.6. Вычисление HMAC

Алгоритмы, построенные с помощью HMAC, традиционно обозначают следующим образом:

- «HMAC-х», где «х» – используемый алгоритм хеширования, например HMAC-MD4;
- «HMAC-х-к» в тех случаях, где выходное значение алгоритма хеширования может быть усечено (т. е. может использоваться частично); в данном случае «к» – размер выходного значения алгоритма HMAC в битах; пример – алгоритм HMAC-SHA1-80 [163].

Стоит отметить, что быстродействие HMAC ненамного хуже, чем у используемой функции хеширования, что особенно заметно при хешировании длинных сообщений, поскольку само сообщение при использовании HMAC обрабатывается однократно [158].

Алгоритм NMAC (Nested Message Authentication Code – «вложенный» код аутентификации сообщения) был предложен авторами алгоритма HMAC в работе [62]. Данный алгоритм позволяет использовать при аутентификации сообщений два ключа: k_1 и k_2 . Формула вычисления NMAC крайне проста:

$$\text{NMAC}(k_1, k_2, m) = \text{hash}(k_2, \text{hash}(k_1, m)),$$

где второй параметр функции $\text{hash}()$ обозначает хешируемое сообщение, а первый параметр – нестандартный вектор инициализации, в качестве которого используется ключ (соответственно, k_1 и k_2 обозначают уже выровненные до размера вектора инициализации ключи).

NMAC фактически является обобщением HMAC, поскольку ключи k_1 и k_2 можно использовать независимо, а можно представить как производные от единственного ключа k , вычисляемые следующим образом [158]:

$$k_1 = h(k \oplus C_1);$$

$$k_2 = h(k \oplus C_2),$$

где $h()$ – функция сжатия используемого алгоритма хеширования; в данном случае применяется стандартный вектор инициализации алгоритма.

Схема NMAC приведена на рис. 1.7.

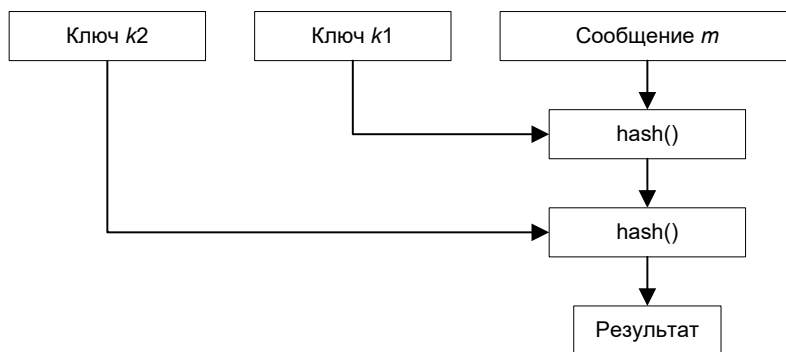


Рисунок 1.7. Вычисление NMAC

Хеширование с использованием деревьев Меркля

Дерево Меркля (Merkle tree) представляет собой бинарное дерево, листья которого содержат хеш-коды блоков данных, а узлы содержат хеш-коды от конкатенации хеш-кодов дочерних листьев или узлов.

Данная структура была предложена Ральфом Мерклем в 1979 г. и запатентована в 1982 г. в патенте [178]. Общий вид дерева Меркля приведен на рис. 1.8.

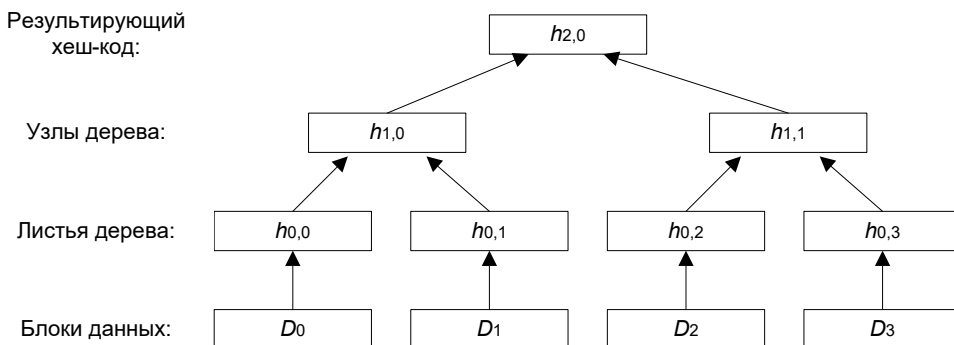


Рисунок 1.8. Дерево Меркля

В таком дереве вычисление хеш-кодов должно выполняться «снизу вверх» следующим образом:

1. Вычисление хеш-кодов данных для каждого листа дерева:

$$h_{0,i} = \text{hash}(D_i),$$

где:

- D_i – i -й блок данных;
 - $h_{0,i}$ – хеш-код i -го блока данных;
 - hash – используемая функция хеширования.
2. Поочередное вычисление хеш-кодов узлов дерева, начиная с более нижних уровней до вычисления корневого хеш-кода, который является результирующим:

$$h_{j,k} = \text{hash}(h_{j-1,2k} \parallel h_{j-1,2k+1}),$$

где:

- $h_{a,b}$ – b -й хеш-код a -го уровня дерева;
- \parallel – операция конкатенации.

Дерево Меркля активно применяется при хешировании данных в различных приложениях (включая ряд известных криптовалют – см. главу 4) в связи с наличием у него ряда положительных свойств (по сравнению с хешированием всех данных за один проход, т. е. одним вызовом функции hash), включая следующие:

- вычисление хеш-кодов с использованием дерева Меркля достаточно хорошо распараллеливается;
- при изменении одного из блоков данных не требуется пересчет хеш-кодов всех данных – достаточно пересчитать только хеш-коды, соответствующие той ветви дерева, которой принадлежит измененный блок данных;
- для проверки целостности одного из блоков данных также не требуется проверять целостность всех данных – достаточно проверить только соответствующую блоку ветвь дерева.

При большом количестве блоков данных пересчет одной ветви дерева по сравнению со всеми данными блоков требует значительно меньшей трудоемкости, в частности в криптовалютных приложениях различия в трудоемкости могут достигать нескольких порядков.

Схожий с деревом Меркля структурный элемент применяется и непосредственно в ряде алгоритмов хеширования. В качестве примера таких алгоритмов можно привести алгоритм хеширования MD6, который будет подробно рассмотрен в данной главе далее.

1.2 МЕТОДЫ КРИПТОАНАЛИЗА И АТАКИ НА АЛГОРИТМЫ ХЕШИРОВАНИЯ

Рассмотрим в данном разделе цели, которые преследуют атакующие при анализе алгоритмов хеширования, а также существующие методы криптоанализа и некоторые из основных результатов атак на хеш-функции.

1.2.1 Цели атак на алгоритмы хеширования

Опишем основные и промежуточные цели атак на алгоритмы хеширования.

Основные цели атак на алгоритмы хеширования

Нахождение коллизии

Коллизией (collision) является ситуация, в которой два различных сообщения m_1 и m_2 имеют один и тот же хеш-код $h = \text{hash}(m_1) = \text{hash}(m_2)$, где $\text{hash}()$ – используемый алгоритм хеширования [242, 246].

Предотвратить существование коллизий принципиально невозможно, поскольку для n -битового алгоритма хеширования размер множества возможных хеш-кодов составляет 2^n , тогда как размер множества хешируемых сообщений является бесконечным. Трудоемкость нахождения коллизий является одной из основных характеристик криптостойкости хеш-функций: для стойкого алгоритма хеширования поиск коллизии должен требовать порядка $2^{n/2}$ операций хеширования.

Существует также понятие мультиколлизии (multicollision) – множества из r сообщений, каждое из которых при хешировании дает один и тот же результат [153, 188]. В работе [153] показано, что трудоемкость поиска мультиколлизии лишь незначительно (логарифмически) превышает трудоемкость поиска коллизии для того же алгоритма хеширования. Мультиколлизии будут подробно описаны далее.

Нахождение прообраза

Атака, направленная на поиск прообраза (preimage attack), ставит своей целью нахождение сообщения, хеш-код которого соответствует заданному хеш-коду h . Существуют два типа таких атак [242, 246]:

- поиск первого прообраза (first preimage attack) – поиск сообщения m , для которого $\text{hash}(m) = h$;
- поиск второго прообраза (second preimage attack) – при имеющемся сообщении m_1 поиск другого сообщения m_2 , удовлетворяющего условию: $\text{hash}(m_2) = \text{hash}(m_1)$.

Считается, что алгоритм хеширования является стойким против данных атак, если для поиска прообраза необходимо не менее 2^n операций хеширования алгоритмом, вычисляющим n -битовые хеш-коды.

Атаки, направленные на поиск прообраза, существенно более опасны, чем атаки, ставящие своей целью поиск коллизий, поскольку они могут быть использованы, в частности, для следующих целей [166]:

- компрометации схем проверки целостности;
- подделки электронной подписи;
- поиска паролей по их известным хеш-кодам.

Определение секретного ключа

Определение секретного ключа как цель атаки актуально для ключевых алгоритмов хеширования или ключевых надстроек над бесключевыми алгоритмами хеширования.

Промежуточные цели атак на алгоритмы хеширования

Нахождение near-коллизии

Near-коллизией называется пара сообщений, хеш-коды которых являются почти эквивалентными, с различиями только в нескольких битах [72].

Само по себе обнаружение near-коллизии не может быть опасным. Однако поиск near-коллизий может использоваться в контексте различных атак на хеш-функции. В качестве примера использования near-коллизии для поиска реальной коллизии можно привести пример атаки на алгоритм SHA, направленной на поиск многоблочной коллизии и описанной в [73]. Поэтому считается, что криптографически стойкие хеш-функции должны также не быть подверженными поиску near-коллизий [174].

Нахождение псевдопрообраза

Псевдопрообраз (pseudo-preimage) – это сообщение m , для которого $\text{hash}(IV', m) = h$, где:

- h – заданный хеш-код;
- IV' – начальный хеш-код, использованный при вычислении хеш-кода h .

Таким образом, псевдопрообраз отличается от первого прообраза тем, что при его поиске не фиксируется начальный хеш-код, т. е. может быть использовано другое начальное значение, нежели прописанное в алгоритме хеширования (например, для алгоритма SHA начальное значение – это совокупность исходных значений регистров $A...E$ – см. описание алгоритма SHA в разделе 1.3). Значение IV' , используемое для поиска псевдопрообраза, может иметь различные специфические свойства, существенно упрощающие поиск псевдопрообраза по сравнению с поиском прообраза (при стандартном начальном значении) [166].

Как и near-коллизии, псевдопрообразы сами по себе не являются опасными (если не рассматривать заведомо ошибочные варианты реализации алгоритмов хеширования, позволяющие изменять или загружать начальные значения), однако псевдопрообразы могут являться промежуточным шагом других атак, в частности они могут быть успешно использованы в контексте атак, посвященных поиску прообраза ([103, 166]).

Нахождение псевдоколлизии

По аналогии с псевдопрообразом, псевдоколлизия определяется как ситуация, в которой два различных сообщения m_1 и m_2 имеют один и тот же хеш-код $h = \text{hash}(m_1, IV_1) = \text{hash}(m_2, IV_2)$, но, в отличие от «классической» коллизии, псевдоколлизия возникает при различных начальных значениях алгоритма хеширования: IV_1 и IV_2 .

В работе [250] утверждается, что криптографические алгоритмы хеширования должны быть стойкими к поиску псевдоколлизий (т. е. трудоемкость поиска псевдоколлизии не должна быть ниже $2^{n/2}$ операций для n -битового алгоритма хеширования), поскольку нахождение псевдоколлизии также может являться промежуточным шагом, снижающим итоговую трудоемкость различных атак на алгоритмы хеширования.

Рассмотрим далее различные методы криптоанализа алгоритмов хеширования и атаки на них, приводящие к достижению описанных выше целей.

1.2.2 Атаки методом «грубой силы»

Метод «грубой силы» (brute-force attack) предполагает перебор всех возможных вариантов каких-либо объектов (в частности, хешируемых сообщений) до нахождения искомого значения (например, сообщения, соответствующего заданному хеш-коду).

Атаки методом «грубой силы» могут быть применены для достижения всех возможных целей атак на алгоритмы хеширования:

- поиска коллизии;
- поиска прообраза;
- определения секретного ключа (для ключевых алгоритмов хеширования).

Рассмотрим последнюю из рассматриваемых целей. Пусть размер секретного ключа в битах равен b . Соответственно, существует 2^b вариантов ключа. Криптоаналитик должен методично перебрать все возможные ключи, т. е. (если рассматривать b -битовую последовательность как число) применить в качестве ключа значение 0, затем 1, 2, 3 и т. д. до максимально возможного ($2^b - 1$). В результате секретный ключ обязательно будет найден, причем в среднем такой поиск потребует $2^{b/2}$, т. е. 2^{b-1} тестовых операций [34].

В качестве критерия корректности найденного ключа используется N пар сообщений и их хеш-кодов. Поскольку при переборе ключей возможны коллизии, в [202] показано, что требуемое число пар для определения верного ключа «несколько превышает» (slightly larger) значение b/n для n -битового алгоритма хеширования.

Защита от атак методом «грубой силы» весьма проста – достаточно лишь увеличить размер ключа: увеличение размера ключа на 1 бит увеличит количество ключей (и среднее время атаки) в 2 раза.

Существуют различные методы усиления эффективности атаки методом «грубой силы», в частности [84]:

- атака методом «грубой силы» простейшим образом распараллеливается: при наличии, скажем, миллиона компьютеров, участвующих в атаке, ключевое множество делится на миллион равных фрагментов, которые распределяются между участниками атаки;
- скорость перебора ключей может быть во много раз увеличена, если в переборе участвуют не компьютеры общего назначения, а специализированные устройства.

Следует учесть, что с развитием вычислительной техники требования к размеру секретного ключа постоянно возрастают. В качестве примера (применительно к симметричному шифрованию) можно привести тот факт, что в той же работе [84] был рекомендован 90-битовый размер ключа в качестве абсолютно безопасного (причем с 20-летним запасом) на конец 1995 г. В 2000 г. эксперты посчитали безопасным с примерно 80-летним запасом использование ключей размером от 128 бит [101].

В известной работе Даниэля Бернштейна (Daniel J. Bernstein) [66] указано, что, несмотря на всю силу параллельных атак методом «грубой силы», криптографы уделяют данной проблеме достаточно мало внимания, допуская следующие ошибки при проектировании криптографических алгоритмов и протоколов и их реализации:

- криптографы часто сильно преувеличивают реальную стойкость своих криптосистем из-за того, что рассматривают только последовательные (т. е. использующие один атакующий компьютер) атаки на криптосистему, не уделяя внимания более сильным параллельным атакам;
- в случае если криптосистема архитектурно зависима от решения, параллельным или последовательным атакам она должна противостоять, часто делается неверный выбор в пользу последовательных атак.

Атака методом «грубой силы» является «мерилом эффективности» других атак – атака считается тем эффективнее, чем быстрее она достигает требуемой цели по сравнению с атакой методом «грубой силы».

1.2.3 Словарные атаки и цепочки хеш-кодов

Словарная атака (dictionary attack) – это метод вскрытия какой-либо информации путем перебора возможных значений данной информации (здесь и далее для определенности в качестве мишени словарных атак будем рассматривать пароли пользователей, хранящиеся в виде их хеш-кодов; при этом область применения словарных атак достаточно широка) [246].

Название атаки произошло благодаря тому, что основу множества перебираемых паролей изначально составляли слова какого-либо языка, а словарные атаки эксплуатировали присущую большинству пользователей тенденцию к использованию легко запоминаемых паролей, к которым часто относятся различные слова и их варианты (например, замена части букв слова на похожие по написанию цифры или спецсимволы).

Словарные атаки часто классифицируют как один из вариантов атаки методом «грубой силы», поскольку здесь также выполняется перебор вариантов пароля или ключа. По сравнению с методом «грубой силы» словарные атаки осуществляют перебор по существенно меньшему (но наиболее вероятному ввиду того, что пользователи нередко выбирают простые и легко запоминающиеся пароли) множеству возможных значений.

Далее рассмотрим некоторые варианты словарных атак и методы противодействия им.

Словарная атака, основанная на предварительных вычислениях

Смысл данного варианта словарной атаки состоит в предварительном вычислении хеш-кодов паролей, принадлежащих какому-либо множеству.

Путем такого вычисления атакующий получает таблицу соответствий входящих в множество паролей и их хеш-кодов (см. рис. 1.9). Для нахождения искомого пароля, соответствующего известному злоумышленнику хеш-коду h , выполняется поиск хеш-кодов в таблице, в результате которого делается один из следующих выводов:

- если в таблице нашлась запись, соответствующая искомому значению h , то пароль p (для которого $h = \text{hash}(p)$) найден; при этом стоит учитывать, что поскольку возможно возникновение коллизий, найденный пароль может быть не искомым паролем, а паролем с эквивалентным искомому хеш-кодом;

Конец ознакомительного фрагмента.

Приобрести книгу можно

в интернет-магазине

«Электронный универс»

e-Univers.ru