Лекция 11: Хеширование и целостность данных

Тема: Целостность данных, цифровые подписи и PKI

Технический университет Молдовы

Лектор: Максим Масютин

Введение

Здравствуйте. Добро пожаловать на лекцию 11. Сегодня мы рассмотрим криптографические хеш-функции -- пожалуй, самые универсальные инструменты в арсенале криптографа. Если шифрование защищает конфиденциальность, а подписи подтверждают подлинность, то хеш-функции -- это незаметные рабочие лошадки, которые делают возможной большую часть современной безопасности.

Вы сталкиваетесь с хеш-функциями десятки раз в день, даже если не осознаёте этого. Когда вы входите на веб-сайт, ваш пароль проверяется с помощью хеша. Когда ваш браузер подключается к защищённому сайту, хеши проверяют целостность сертификатов. Когда вы скачиваете программное обеспечение, вы можете убедиться, что оно не было изменено, с помощью хеша. Когда вы фиксируете код в Git, каждый коммит идентифицируется хешем.

Сегодня мы разберём, что делает криптографическую хеш-функцию надёжной, изучим основные алгоритмы от MD5 до SHA-3 и BLAKE3, научимся использовать хеш-функции для аутентификации сообщений и погрузимся в специализированный мир хеширования паролей, где медлительность -- это на самом деле цель.

Часть 1: Основы криптографических хеш-функций

Что такое хеш-функция?

Криптографическая хеш-функция принимает произвольный объём входных данных и создаёт выход фиксированного размера, называемый хешем, дайджестом или отпечатком. Этот выход детерминирован -- одни и те же входные

данные всегда дают одинаковый результат -- и при этом выглядит совершенно случайным.

Например, SHA-256 всегда создаёт 256-битный (32-байтовый) выход, хешируете ли вы один символ или целый видеофайл. Хеш строки "Hello" всегда будет одинаковым, но хеш "Hello!" будет совершенно другим.

Математическая запись проста: H(m) = h, где m -- сообщение (произвольной длины), а h -- хеш (фиксированной длины).

Основные свойства безопасности

Криптографические хеш-функции должны удовлетворять трём фундаментальным свойствам безопасности. Понимание этих свойств критически важно для того, чтобы знать, когда и как правильно использовать хеш-функции.

1. Устойчивость к преобразу (свойство односторонности)

По данному хеш-значению h должно быть вычислительно невозможно найти какое-либо сообщение m, такое что H(m) = h.

Представьте это так: если я дам вам хеш, вы не должны иметь возможности восстановить то, что его породило. Единственным вариантом должен быть перебор -- хешировать случайные входные данные, пока не найдётся совпадение. Для 256-битного хеша это означает перебор в среднем 2^255 входных значений, что вычислительно невозможно.

Почему это важно? Если вы храните хеши паролей (а не сами пароли), злоумышленник, похитивший базу хешей, не должен иметь возможности восстановить пароли.

2. Устойчивость ко второму преобразу

По данному сообщению m1 и его хешу H(m1) должно быть вычислительно невозможно найти другое сообщение m2, такое что H(m2) = H(m1).

Это тонко отличается от устойчивости к преобразу. Здесь злоумышленник знает и исходное сообщение, и его хеш, и пытается найти другое сообщение с таким же хешем.

Почему это важно? Если вы подписываете документ, подписывая его хеш, злоумышленник не должен иметь возможности создать другой документ с тем же хешем и утверждать, что вы его подписали.

3. Устойчивость к коллизиям

Должно быть вычислительно невозможно найти два любых различных сообщения m1 и m2, таких что H(m1) = H(m2).

Отметим, что коллизии математически неизбежны -- мы отображаем бесконечное множество возможных входов в конечное множество выходов. Но нахождение коллизий должно быть практически невозможным.

Парадокс дней рождения делает поиск коллизий проще, чем атаки на преобраз. Для хеша с n-битным выходом нахождение коллизии требует приблизительно 2^(n/2) вычислений хеша. Поэтому SHA-256 обеспечивает 128 бит устойчивости к коллизиям, а не 256 бит.

Лавинный эффект

Хорошая хеш-функция демонстрирует лавинный эффект: изменение одного бита входных данных изменяет приблизительно половину бит выходных данных. Выходные значения выглядят статистически независимыми даже для очень похожих входных данных.

Например, хеши SHA-256 строк "test1" и "test2" различаются примерно в 130 битах (из 256), без какой-либо различимой закономерности. Это свойство является ключевым для безопасности -- любая закономерность дала бы злоумышленникам информацию, которую они могли бы использовать.

Конструкция Меркла-Дамгорда

Большинство классических хеш-функций (MD5, SHA-1, SHA-2) используют конструкцию Меркла-Дамгорда:

1. Дополнить сообщение до размера, кратного размеру блока
2. Разбить на блоки
3. Обработать каждый блок последовательно с помощью функции сжатия
4. Каждая функция сжатия принимает предыдущее состояние и текущий блок, создавая новое состояние
5. Финальное состояние и есть хеш

Эта конструкция обладает математическим свойством: если функция сжатия устойчива к коллизиям, то и общая хеш-функция устойчива к коллизиям. Однако у неё есть слабости, включая атаки удлинением сообщения, когда знание H(m) позволяет вычислить H(m || дополнение || m') без знания m.

Часть 2: Устаревшие хеш-функции

MD5 (Message Digest 5)

MD5 был разработан Рональдом Ривестом в 1991 году для замены MD4. Он создаёт 128-битный хеш и широко использовался более десяти лет.

Крушение MD5 произошло через последовательные криптоаналитические атаки:

- 1996: обнаружены слабости в функции сжатия

- 2004: первая практическая коллизия найдена Wang и др.
- 2008: коллизии с выбранным префиксом позволили создать поддельный сертификат удостоверяющего центра
- Сегодня: коллизии генерируются за секунды на ноутбуке

MD5 полностью взломан в отношении устойчивости к коллизиям. Злоумышленник может создать два различных документа с одинаковым хешем MD5. Это было наглядно продемонстрировано в 2008 году, когда исследователи создали поддельный SSL-сертификат, эксплуатируя коллизии MD5.

Текущий статус: MD5 никогда не должен использоваться для каких-либо целей безопасности. Он допустим лишь как некриптографическая контрольная сумма для обнаружения случайного повреждения данных (когда злоумышленник отсутствует).

SHA-1 (Secure Hash Algorithm 1)

SHA-1 был разработан NSA (National Security Agency, Агентство национальной безопасности) и опубликован NIST (National Institute of Standards and Technology, Национальный институт стандартов и технологий) в 1995 году. Он создаёт 160-битный хеш и был наиболее широко используемой хеш-функцией на протяжении почти двух десятилетий.

Закат SHA-1 был медленнее, но неизбежен:

- 2005: опубликованы теоретические коллизионные атаки
- 2017: первая практическая коллизия (атака "SHAttered" компанией Google)
- Стоимость: приблизительно 110 GPU-лет

Атака SHAttered создала два различных PDF-файла с идентичными хешами SHA-1. Это были не случайные данные, а реальные PDF-документы, которые могли отображать различное содержимое.

Текущий статус: SHA-1 считается устаревшим для всех приложений безопасности. Основные браузеры отклоняют сертификаты SHA-1, и Git начал переход с SHA-1. Используйте SHA-256 или лучше для любых новых приложений.

Часть 3: Семейство SHA-2

Обзор

SHA-2, разработанный NSA и опубликованный в 2001 году, на самом деле представляет собой семейство хеш-функций с различными размерами выхода:

- **SHA-224**: 224-битный выход, основан на SHA-256

- **SHA-256**: 256-битный выход, наиболее широко используемый
- **SHA-384**: 384-битный выход, основан на SHA-512
- **SHA-512**: 512-битный выход
- **SHA-512/224** и **SHA-512/256**: усечённые варианты SHA-512

Несмотря на происхождение из той же организации, что и SHA-1, SHA-2 использует принципиально иной дизайн и остаётся безопасным. За почти 25 лет анализа не было обнаружено существенных криптографических слабостей.

SHA-256

SHA-256 -- рабочая лошадка современной криптографии. Он обрабатывает 512-битные блоки и поддерживает 256-битное состояние через 64 раунда операций смешивания.

Каждый раунд использует:

- Побитовые операции (AND, OR, XOR, NOT)
- Сложение по модулю
- Циклические сдвиги вправо

Алгоритм использует расписание сообщений для расширения каждого 512-битного блока в 64 32-битных слова, обеспечивая влияние каждого входного бита на множество выходных бит.

SHA-256 обеспечивает:

- 256-битную устойчивость к преобразу
- 256-битную устойчивость ко второму преобразу
- 128-битную устойчивость к коллизиям (граница дней рождения)

SHA-256 используется в сертификатах TLS, майнинге Bitcoin, конструкциях HMAC, цифровых подписях и бесчисленных других приложениях.

SHA-512 и варианты

SHA-512 обрабатывает 1024-битные блоки и создаёт 512-битный хеш. Он использует 80 раундов вместо 64 и работает с 64-битными словами вместо 32-битных.

На 64-битных процессорах SHA-512 часто быстрее SHA-256, потому что обрабатывает больше данных за операцию при том же количестве инструкций. Это кажется нелогичным -- более длинный хеш быстрее -- но имеет смысл с учётом 64-битной архитектуры процессора.

SHA-512/256 -- это SHA-512, усечённый до 256 бит с другими начальными значениями. Он обеспечивает:

- Устойчивость к атакам удлинением сообщения (в отличие от обычного SHA-256)
- Часто быстрее SHA-256 на 64-битных системах
- Такую же эффективную безопасность, как SHA-256

SHA-384 -- это SHA-512, усечённый до 384 бит, обычно используемый там, где требуется более 128 бит устойчивости к коллизиям.

Когда использовать SHA-2

SHA-256 остаётся рекомендуемой хеш-функцией для большинства приложений в 2025-2026 годах:

- Цифровые подписи
- Отпечатки сертификатов
- Конструкция HMAC
- Выработка ключей (с HKDF)
- Общая проверка целостности

Используйте SHA-512, когда:

- Работа ведётся на 64-битных системах, где он быстрее
- Требуется более 128 бит устойчивости к коллизиям
- Протокол его указывает (некоторые наборы шифров TLS)

Часть 4: SHA-3 (Keccak)

Предпосылки и конкурс

Хотя SHA-2 не был взломан, NIST был обеспокоен тем, что все его стандарты основаны на одном семействе конструкций. Если бы была обнаружена фундаментальная слабость в конструкции SHA-2, запасного варианта бы не было.

В 2007 году NIST объявил конкурс на SHA-3, аналогичный конкурсу AES. После пяти лет публичного криптоанализа алгоритм Keccak (произносится "кечак") был выбран в 2012 году и стандартизирован как SHA-3 в 2015 году.

Конструкция губки

SHA-3 использует радикально иную конструкцию, называемую губкой. Вместо последовательного сжатия по Мерклу-Дамгорду:

1. Инициализировать состояние в 1600 бит (200 байт)

2. **Фаза поглощения**: выполнить XOR входных блоков с состоянием, применить перестановку после каждого
3. **Фаза отжима**: извлечь выходные блоки из состояния, применяя перестановку между извлечениями

Состояние разделено на скорость (биты, взаимодействующие с входом/выходом) и ёмкость (биты, обеспечивающие запас безопасности). Ёмкость определяет безопасность: SHA3-256 имеет ёмкость 512 бит, обеспечивая 256-битную безопасность.

Варианты SHA-3

Стандарт SHA-3 определяет:

- **SHA3-224**: 224-битный выход, 448-битная ёмкость
- **SHA3-256**: 256-битный выход, 512-битная ёмкость
- **SHA3-384**: 384-битный выход, 768-битная ёмкость
- **SHA3-512**: 512-битный выход, 1024-битная ёмкость

Они обеспечивают прямую замену функций SHA-2 с эквивалентной безопасностью.

SHAKE128 и SHAKE256

SHA-3 также определяет функции с расширяемым выходом (XOF -- Extendable Output Functions):

- **SHAKE128**: 128-битная безопасность, произвольная длина выхода
- **SHAKE256**: 256-битная безопасность, произвольная длина выхода

В отличие от хеш-функций с фиксированным выходом, XOF могут создавать столько выходных данных, сколько необходимо. SHAKE128 с 256 битами выхода -- это не то же самое, что SHAKE128 с 512 битами -- они начинаются одинаково, но далее различаются.

XOF полезны для:

- Выработки ключей (замена HKDF в некоторых контекстах)
- Функций генерации масок
- Детерминированной генерации случайных чисел
- Создания хеш-функций нестандартных размеров

SHA-3 против SHA-2

Когда следует использовать SHA-3 вместо SHA-2?

Предпочитайте SHA-2 (SHA-256, SHA-512), когда:

- Требуется максимальная совместимость
- Доступно аппаратное ускорение (SHA-NI)
- Необходима максимально быстрая программная реализация

Предпочитайте SHA-3, когда:

- Необходима устойчивость к атакам удлинением (SHA-3 изначально к ним невосприимчив)
- Желательно разнообразие в отношении SHA-2 для эшелонированной защиты
- Требуется выход переменной длины (используйте SHAKE)
- Важна устойчивость к атакам по побочным каналам (дизайн Keccak помогает в этом)

На практике и SHA-2, и SHA-3 являются отличным выбором. Криптографическое сообщество тщательно проанализировало оба алгоритма, и оба остаются безопасными.

Часть 5: BLAKE2 и BLAKE3

BLAKE2

BLAKE2, опубликованный в 2012 году, -- это хеш-функция, оптимизированная для скорости при сохранении безопасности. Она основана на BLAKE, финалисте конкурса SHA-3, но перепроектирована для практического применения.

BLAKE2 имеет два основных варианта:

- **BLAKE2b**: оптимизирован для 64-битных платформ, создаёт хеши до 512 бит
- **BLAKE2s**: оптимизирован для 32-битных платформ, создаёт хеши до 256 бит

Ключевые особенности:

- Быстрее MD5 в программной реализации (да, быстрее взломанного алгоритма)
- Настраиваемая длина выхода
- Встроенная работа с ключами (может действовать как код аутентификации сообщений без обёртки HMAC)
- Встроенная персонализация и соль
- Режим древовидного хеширования для параллельной обработки

BLAKE2 используется в:

- libsodium (в качестве хеш-функции по умолчанию)
- WireGuard VPN

- Хешировании паролей Argon2
- Многих инструментах проверки файлов

BLAKE3

BLAKE3, выпущенный в 2020 году, развивает идеи BLAKE2 дальше:

- Приблизительно в 4 раза быстрее BLAKE2
- Параллелизуемый по конструкции (использует структуру дерева Меркла)
- Единый алгоритм работает на всех платформах (нет вариантов b/s)
- Фиксированный 256-битный выход (режим XOF для более длинного выхода)
- Встроенное хеширование с ключом и выработка ключей

BLAKE3 достигает своей скорости за счёт:

- Уменьшенного числа раундов (7 вместо 10-12 у BLAKE2)
- Древовидной структуры, обеспечивающей параллелизм
- Операций, дружественных к SIMD

BLAKE3 отлично подходит для:

- Высокоскоростного хеширования файлов
- Проверки больших объёмов данных
- Приложений, которые могут использовать параллелизм
- Замены MD5/SHA-1 в некриптографических контрольных суммах

Сравнительная сводка

Алгоритм	Безопасность	Скорость	Параллелизм	Область применения
SHA-256	Отличная	Хорошая	Последовательный	Соответствие стандартам
SHA-512	Отличная	Хорошая	Последовательный	64-битные системы
SHA3-256	Отличная	Умеренная	Последовательный	Разнообразие относительно SHA-2
BLAKE2b	Отличная	Очень высокая	Ограниченный	Общее назначение
BLAKE3	Отличная	Чрезвычайно высокая	Полный	Высокоскоростное хеширование

Часть 6: Коды аутентификации сообщений

Хеширование с ключом и без ключа

Стандартная (бесключевая) хеш-функция, такая как SHA-256, принимает один вход -- данные -- и выдаёт дайджест фиксированного размера: дайджест = H(данные). Поскольку функция публична и детерминирована, любой с теми же данными может независимо вычислить тот же дайджест. Это делает бесключевое хеширование полезным для **проверки целостности** (обнаружения изменений данных), но бесполезным для **аутентификации** (подтверждения того, кто создал или одобрил данные). Злоумышленник, модифицировавший сообщение, может просто пересчитать хеш модифицированного сообщения.

Хеширование с ключом решает эту проблему, включая секретный ключ в качестве дополнительного входа: тег = MAC(ключ, данные). Только тот, кто обладает секретным ключом, может вычислить правильный тег для данного сообщения. Получатель, разделяющий ключ, может проверить тег и убедиться как в том, что данные не были изменены (целостность), так и в том, что они были созданы кем-то, кто знает ключ (подлинность). Это и есть фундаментальное назначение кода аутентификации сообщений (MAC).

Различие имеет значение на практике. Если вы храните SHA-256 хеш файла для обнаружения повреждений, любой злоумышленник, модифицировавший файл, может также обновить хеш. Но если вы храните тег HMAC-SHA256, вычисленный с секретным ключом, злоумышленник, модифицировавший файл, не сможет создать валидный тег без знания ключа. Хеширование с ключом превращает пассивную проверку целостности в активный механизм аутентификации.

Концепция хеширования с ключом появляется и в хешировании паролей (Лекция 9): "перец" -- это секретное значение, подмешиваемое к паролю перед хешированием, фактически превращающее хеш в ключевую операцию, где перец выполняет роль ключа. Без перца злоумышленник, получивший базу данных хешей, может проверять предполагаемые пароли офлайн.

HMAC (код аутентификации на основе хеша)

HMAC, стандартизированный в RFC 2104, строит код аутентификации из любой криптографической хеш-функции. Формула:

HMAC(K, m) = H((K XOR opad) || H((K XOR ipad) || m))

Где:

- K -- секретный ключ (дополненный до размера блока)
- opad и ipad -- фиксированные константы
- || обозначает конкатенацию

Структура двойного хеширования обеспечивает безопасность, даже если базовая хеш-функция имеет некоторые слабости. HMAC-MD5 на самом деле всё ещё считается безопасным для аутентификации (хотя и не рекомендуется для новых приложений).

Распространённые реализации:

- **HMAC-SHA256**: наиболее широко используемый, 256-битный выход
- **HMAC-SHA384**: используется, когда необходима безопасность 128+ бит
- **HMAC-SHA512**: максимальная безопасность

HMAC используется в:

- TLS для аутентификации записей
- Аутентификации API (подписи HMAC-SHA256)
- JWT (JSON Web Token)
- Выработке ключей HKDF

KMAC (код аутентификации на основе Keccak)

Конструкция губки SHA-3 позволяет напрямую поглощать ключ без обёртки HMAC. KMAC, стандартизированный в NIST SP 800-185, предоставляет код аутентификации непосредственно из Keccak.

Преимущества KMAC:

- Однопроходный (в отличие от двойного хеширования HMAC)
- Встроенное разделение областей применения
- Выход переменной длины (режим XOF)

KMAC определён в:

- **KMAC128**: 128-битная безопасность
- **KMAC256**: 256-битная безопасность

KMAC является более новым и менее широко развёрнутым, чем HMAC, но становится всё более распространённым в системах, уже использующих SHA-3.

Poly1305

Poly1305, разработанный Дэниелом Бернштейном, -- это одноразовый аутентификатор, используемый с ChaCha20 (ChaCha20-Poly1305) и AES (AES-GCM использует GHASH, но Poly1305 доступен как альтернатива).

Ключевые свойства:

- Чрезвычайно быстрый (особенно в программной реализации)
- Создаёт 128-битный тег

- Требует новый ключ для каждого сообщения (обычно выводится из шифра)
- Теоретико-информационно безопасен при одноразовом использовании

Poly1305 не используется самостоятельно, а как часть конструкций аутентифицированного шифрования. ChaCha20-Poly1305 выводит ключ Poly1305 из ключа шифрования и одноразового номера для каждого сообщения.

Когда что использовать

- **HMAC-SHA256**: выбор по умолчанию, максимальная совместимость
- **HMAC-SHA512**: когда важна скорость на 64-битных системах
- **KMAC256**: при уже используемом SHA-3, если нужны возможности XOF
- **Poly1305**: как часть аутентифицированного шифрования (ChaCha20-Poly1305)

Никогда не используйте:

- Хеш без ключа для аутентификации (любой может пересчитать)
- HMAC с MD5 или SHA-1 для новых приложений
- Самодельные конструкции кодов аутентификации

Часть 7: Хеширование паролей

Хеширование паролей принципиально отличается от хеширования общего назначения. Для проверки целостности файлов мы хотим быстрые хеши. Для паролей мы хотим медленные хеши.

Проблема хеширования паролей

Когда пользователи создают пароли, они выбирают их плохо. Согласно ежегодным отчётам NordPass, компании-разработчика менеджера паролей, и исследованиям Verizon Data Breach Investigations Report (DBIR), большинство паролей выбираются из относительно небольшого набора:

- Словарные слова с простыми модификациями
- Клавиатурные шаблоны (qwerty, 123456)
- Личная информация (имена, даты)
- Ранее утёкшие пароли

Злоумышленник с базой хешей в первую очередь попробует эти распространённые пароли. При использовании быстрых хешей вроде SHA-256 злоумышленник может перебирать миллиарды паролей в секунду на потребительском оборудовании. Даже умеренно сложный пароль будет подобран быстро.

Решение -- намеренно медленные хеш-функции, которые делают атаки методом перебора затратными.

Требования к хешированию паролей

Хорошая функция хеширования паролей должна:

1. **Быть медленной**: настраиваемый коэффициент трудоёмкости, чтобы не отставать от развития оборудования
2. **Использовать память**: предотвращать ускорение на GPU и ASIC
3. **Использовать соль**: предотвращать атаки с помощью радужных таблиц и атаки на несколько целей
4. **Быть детерминированной**: один и тот же пароль должен проверяться стабильно

bcrypt

bcrypt, опубликованный в 1999 году, был разработан специально для хеширования паролей. Он основан на затратной процедуре настройки ключей шифра Blowfish.

Ключевые особенности:

- Настраиваемый коэффициент стоимости (2^cost итераций)
- 128-битная соль, встроенная в выходные данные
- 184-битный хеш на выходе
- Максимальная длина пароля 72 байта

Формат bcrypt: $2b$cost$salthashvalue

Пример: $2b$12$salt22characters...hash31characters...

bcrypt остаётся безопасным и широко используемым. Однако у него есть ограничения:

- Лимит пароля в 72 байта
- Не использует память (подвержен атакам на GPU)
- Коэффициент стоимости влияет только на время процессора

scrypt

scrypt, опубликованный в 2009 году, добавляет стойкость к объёму памяти в задачу хеширования паролей. Он спроектирован так, чтобы быть затратным как по времени процессора, так и по памяти.

Параметры:

- **N**: стоимость CPU/памяти (итерации, должна быть степенью 2)
- **r**: размер блока (множитель использования памяти)
- **p**: коэффициент параллелизма

scrypt использует память целенаправленно -- алгоритм должен хранить в памяти большой массив и обращаться к нему случайным образом. Это делает атаки на GPU затратными, поскольку GPU имеют ограниченный объём памяти на ядро.

scrypt используется в:

- Майнинге криптовалют (Litecoin, Dogecoin)
- Зашифрованных файловых системах
- Хранилищах паролей

Argon2

Argon2, победитель конкурса хеширования паролей (Password Hashing Competition) в 2015 году, представляет собой современное состояние искусства. Он был разработан для устранения ограничений как bcrypt, так и scrypt.

Три варианта:

- **Argon2d**: зависящий от данных доступ к памяти, самый быстрый, использовать только на серверной стороне
- **Argon2i**: независящий от данных доступ к памяти, устойчив к атакам по побочным каналам
- **Argon2id**: гибрид d и i, рекомендуется для большинства применений

Параметры:

- **t**: стоимость по времени (итерации)
- **m**: стоимость по памяти (килобайты)
- **p**: параллелизм (потоки)

Преимущества Argon2:

- Стойкость к объёму памяти (как scrypt)
- Независимая настройка времени/памяти/параллелизма
- Вариант, устойчивый к атакам по побочным каналам (Argon2i, Argon2id)
- Нет ограничения на длину пароля
- Стандартизирован IETF (Internet Engineering Task Force, Инженерный совет интернета) в RFC 9106

Рекомендуемые параметры Argon2id (2025):

- Минимальные: m=47104 (46 MiB), t=1, p=1
- Интерактивный вход: m=65536 (64 MiB), t=3, p=4

- Высокая безопасность: m=2097152 (2 GiB), t=1, p=4

Лучшие практики хеширования паролей

1. **Всегда используйте специализированную функцию хеширования паролей** (Argon2id, bcrypt, scrypt)
2. **Никогда не используйте хеш-функции общего назначения** (SHA-256 сам по себе не подходит)
3. **Всегда включайте соль** (все современные хеширующие функции для паролей делают это автоматически)
4. **Настраивайте параметры под ваше оборудование** (цель -- 100 мс - 1 с для интерактивного входа)
5. **Увеличивайте коэффициенты стоимости со временем** (оборудование становится быстрее)
6. **Используйте сравнение с постоянным временем** (предотвращение атак по времени)
7. **Реализуйте ограничение частоты запросов** (предотвращение онлайн-перебора)

Миграция хешей паролей

Если ваша система использует устаревшее хеширование паролей (MD5, SHA-1, слабый bcrypt):

Вариант 1: обернуть существующие хеши

- Новый хеш = Argon2id(старый_хеш)
- Проверка: вычислить старый хеш, затем Argon2id
- При следующем входе обновить до Argon2id(пароль) напрямую

Вариант 2: принудительный сброс пароля

- Более разрушительный, но чистый
- Подходит для систем высокой безопасности

Вариант 3: постепенная миграция

- Хешировать новые и изменённые пароли с помощью Argon2id
- Обновлять существующих пользователей при входе
- В конечном итоге заставить оставшихся пользователей сбросить пароль

Часть 8: Практические применения

Проверка целостности файлов

Хеш-функции проверяют, что файлы не были изменены:

```
sha256sum important_file.dat > important_file.sha256
# Позднее...
sha256sum -c important_file.sha256
```

Для распространения программного обеспечения:

- Издатель вычисляет хеш релиза
- Хеш подписывается ключом издателя
- Пользователи скачивают файл и проверяют совпадение хеша

Идентификация коммитов Git

Git использует SHA-1 (переходит на SHA-256) для идентификации:

- Каждого файла (blob)
- Каждого каталога (tree)
- Каждого коммита

Хеш коммита зависит от:

- Хеша дерева (структура каталогов)
- Хеша(ей) родительского коммита
- Информации об авторе и коммиттере
- Сообщения коммита
- Метки времени

Любое изменение любой части истории изменяет все последующие хеши. Именно это делает историю Git устойчивой к подделке.

Доказательство работы (блокчейн)

Криптовалюты используют хеш-функции для доказательства работы:

- Найти одноразовый номер (nonce) такой, что H(блок || nonce) начинается с множества нулей
- Требуется множество вычислений хеша (майнинг)
- Проверка дёшева (единственный хеш)

Bitcoin использует двойной SHA-256: SHA256(SHA256(блок))

Хранилище с адресацией по содержимому

Системы вроде IPFS используют хеши в качестве идентификаторов содержимого:

- CID = хеш содержимого
- Одинаковое содержимое всегда имеет одинаковый CID
- Содержимое может быть проверено по его CID
- Обеспечивает дедупликацию и проверку целостности

Схемы обязательств

Хеш-функции позволяют реализовать криптографические обязательства:

1. Фаза фиксации: отправить H(секрет || одноразовый номер)
2. Фаза раскрытия: отправить секрет и одноразовый номер
3. Проверка: убедиться в совпадении хеша

Это используется в:

- Аукционах с запечатанными заявками
- Протоколах голосования
- Честной генерации случайных чисел

Часть 9: Сводка лучших практик

Выбор алгоритма

Для хеширования общего назначения:

- **По умолчанию**: SHA-256
- **Высокая безопасность**: SHA-384 или SHA-512
- **Критична скорость**: BLAKE3 или BLAKE2b
- **Экосистема SHA-3**: SHA3-256

Для кодов аутентификации:

- **По умолчанию**: HMAC-SHA256
- **С SHA-3**: KMAC256
- **Аутентифицированное шифрование**: используйте встроенный MAC режима шифрования

Для паролей:

- **По умолчанию**: Argon2id
- **Совместимость с устаревшим**: bcrypt
- **Никогда**: простой SHA-256, MD5, SHA-1

Чего никогда не делать

1. Никогда не используйте MD5 или SHA-1 для каких-либо целей безопасности
2. Никогда не используйте простые хеши для паролей
3. Никогда не используйте хеш для аутентификации без ключа (используйте HMAC)
4. Никогда не реализуйте собственную хеш-функцию
5. Никогда не усекайте хеши ниже границы дней рождения
6. Никогда не игнорируйте атаки по времени при сравнении хешей

Замечания по реализации

1. Используйте криптографические библиотеки, а не собственные реализации
2. Используйте сравнение с постоянным временем для проверки кодов аутентификации
3. Включайте версию протокола/разделение областей во входные данные
4. Учитывайте гибкость в выборе хеш-алгоритма при проектировании протоколов
5. Следите за объявлениями об устаревании алгоритмов

Заключение

Сегодня мы исследовали мир криптографических хеш-функций -- от фундаментальных свойств, которые делают их безопасными, до современных алгоритмов, защищающих нашу цифровую инфраструктуру.

Ключевые выводы:

1. **Хеш-функции обеспечивают целостность, а не подлинность** -- используйте HMAC для аутентификации
2. **MD5 и SHA-1 взломаны** в отношении устойчивости к коллизиям и не должны использоваться для обеспечения безопасности
3. **SHA-256 -- стандарт** -- широко развёрнут, хорошо проанализирован, безопасен
4. **SHA-3 обеспечивает разнообразие** относительно SHA-2 с другой конструкцией

5. **BLAKE2/BLAKE3 предлагают скорость** для приложений, где важна производительность
6. **Хеширование паролей -- особый случай** -- используйте Argon2id, bcrypt или scrypt, но никогда хеш-функции общего назначения
7. **Лавинный эффект и устойчивость к коллизиям** -- вот что делает хеш-функции полезными для безопасности

На следующей неделе мы объединим всё, что узнали о хешировании, симметричном шифровании и асимметричной криптографии, чтобы изучить цифровые подписи и инфраструктуру открытых ключей -- как мы аутентифицируем личность в цифровом мире.

Вопросы для обсуждения

1. Каковы последствия того, что организации продолжают использовать устаревшие хеш-функции, такие как MD5 и SHA-1, в своих системах?
2. Как организациям следует подходить к миграции хранилища паролей со старых методов хеширования на современные алгоритмы вроде Argon2id?
3. Какую роль играют хеш-функции в технологии блокчейн и каковы последствия для безопасности?
4. Компания обнаруживает, что их база данных паролей использовала хеши MD5 без соли. Какие шаги им следует предпринять и насколько срочно?
5. Почему некоторые веб-сайты до сих пор показывают вам ваш старый пароль, когда вы нажимаете "забыл пароль"? Что это говорит о том, как они хранят пароли?

Благодарю за внимание. Увидимся на следующей неделе.

Контрольные вопросы

1. Перечислите три основных свойства безопасности криптографических хеш-функций и объясните каждое.
2. Почему MD5 и SHA-1 считаются взломанными для целей безопасности? Какие атаки были продемонстрированы?
3. Что такое HMAC и чем он отличается от простого добавления ключа к сообщению перед хешированием?
4. Почему небезопасно хешировать пароль напрямую хеш-функцией общего назначения вроде SHA-256, даже если алгоритмы хеширования паролей, такие как bcrypt и Argon2id, внутренне используют хеш-функции?

5. Что такое bcrypt и как его фактор стоимости влияет на безопасность?
6. Что такое Argon2id и почему он является рекомендуемым алгоритмом для хеширования паролей в настоящее время?
7. Что такое атака удлинением сообщения и какие конструкции хеш-функций к ней уязвимы?

Ключевые термины

- **Argon2id**: алгоритм хеширования паролей со стойкостью к объёму памяти, победитель конкурса хеширования паролей
- **Лавинный эффект**: свойство, при котором малые изменения входных данных вызывают значительные изменения выходных данных
- **bcrypt**: функция хеширования паролей, основанная на шифре Blowfish
- **Атака дней рождения**: криптографическая атака, которая использует математику теории вероятностей для нахождения коллизий хешей быстрее, чем полный перебор
- **BLAKE2**: быстрая криптографическая хеш-функция, быстрее MD5 при сохранении безопасности
- **BLAKE3**: параллелизуемая хеш-функция, производная от BLAKE2, предлагающая чрезвычайно высокую скорость
- **Устойчивость к коллизиям**: свойство, при котором вычислительно невозможно найти два входных значения с одинаковым хешем
- **Хеш-функция**: функция, отображающая вход произвольного размера в выход фиксированного размера
- **HMAC**: код аутентификации сообщений на основе хеша, обеспечивающий как целостность, так и подлинность
- **KMAC**: код аутентификации сообщений на основе Keccak, построенный на семействе SHA-3
- **MD5**: Message Digest 5, взломанная хеш-функция, которая не должна использоваться для обеспечения безопасности
- **Конструкция Меркла-Дамгорда**: конструкция, используемая хеш-функциями семейств MD5, SHA-1 и SHA-2
- **Хеширование паролей**: использование намеренно медленных хеш-функций для защиты хранимых паролей
- **Устойчивость к преобразу**: свойство, при котором по данному хешу невозможно найти исходные входные данные

- **Соль**: случайные данные, добавляемые ко входным данным перед хешированием для предотвращения атак с помощью радужных таблиц
- **scrypt**: функция хеширования паролей со стойкостью к объёму памяти, спроектированная для устойчивости к аппаратному перебору
- **SHA-1**: Secure Hash Algorithm 1, взломан в отношении устойчивости к коллизиям с 2017 года
- **SHA-256**: член семейства SHA-2, создающий 256-битные хеши, широко используемый и безопасный
- **SHA-3**: третье поколение Secure Hash Algorithm, основанное на конструкции губки Keccak
- **SHAKE/XOF**: функции SHA-3 с расширяемым выходом (SHAKE128, SHAKE256), создающие хеш-значения переменной длины
- **Конструкция губки**: конструкция, лежащая в основе SHA-3, устойчивая к атакам удлинением сообщения