Основные понятия надежности

1 Лекция

Основные показатели надежности:

- **доступность** (availability) — это свойство системы находиться в состоянии готовности к работе, показывает вероятность того, что система в данный момент времени будет правильно работать.
- **безотказность** (reliability) — свойство системы работать без отказов. В противоположность доступности безотказность определяется в понятиях временного интервала, а не момента времени.

- **безопасность** (safety) — определяет, насколько катастрофична ситуация временной неспособности системы должным образом выполнять свою работу. Например, на атомных электростанциях, должны обладать высокой степенью безопасности.
- **ремонтопригодность** (maintainability) — определяет, насколько сложно исправить неполадки в системе.

Виды отказов:

- **проходные** отказы (transient faults) — происходят однократно и больше не повторяются. Если повторить операцию, они не возникают.
- **перемежающиеся** отказы (intermittent faults) — появляются и пропадают, «когда захотят». Такие отказы трудно диагностировать.
- **постоянные** отказы (permanent faults) — продолжают свое существование до тех пор, пока отказавший компонент не будет заменен или исправлен.

Типы отказов:

- **поломка** (crash failure) — внезапная остановка сервера, при этом до момента остановки он работает нормально.Пример поломки — полное зависание операционной системы, когда единственным решением проблемы является перезагрузка.
- **пропуск данных** (omission failure) — возникает в том случае, когда сервер неправильно реагирует на запросы.

Типы отказов:

- а) **пропуск приема** {receive omission) — сервер может, например, не получать запросов. Отметим, что такая ошибка может произойти, в частности, и в том случае, когда соединение между клиентом и сервером установлено совершенно правильным образом, но на сервере не запущен процесс для приема приходящих запросов. Пропуск приема обычно не влияет на текущее состояние сервера, но сервер остается в неведении о посланных ему сообщениях.

- б) **пропуск передачи** (send omission) — происходит, когда сервер выполняет свою работу, но по каким-либо причинам не в состоянии послать ответ. Подобная ошибка может произойти, например, при переполнении буфера передачи, если сервер не готов к подобной ситуации. Отметим, что в противоположность пропуску приема в данном случае сервер может перейти в состояние, соответствующее полному выполнению услуги для клиента. Впоследствии, если обнаружится, что имел место пропуск передачи, сервер, вероятно, должен быть готов к тому, что клиент повторно пошлет свой последний запрос.

- Другие типы пропусков не имеют отношения к взаимодействию и могут быть вызваны ошибками в программе, такими как бесконечные циклы или некорректная работа с памятью, которые способны «подвесить» сервер.

ошибки синхронизации (timing failures)

- возникают при ожидании ответа дольше определенного временного интервала. Слишком раннее предоставление данных легко может вызвать у принимающей стороны проблемы, связанные с отсутствием места в буфере для хранения получаемых данных.
- Чаще, однако, сервер отвечает слишком поздно,
в этом случае говорят, что
произошла **ошибка производительности** (performance failure).

- **ошибки отклика** (response failures) — при которых ответы сервера просто неверны.

  а) **ошибки значения** (value failure) — сервер дает неверный ответ на запрос. Так, например, эту ошибку демонстрирует поисковая машина, систематически возвращающая адреса web-страниц, не связанных с запросом пользователя.

ошибки передачи состояния

- б) (state transition failures) — Этот тип ошибок характеризуется реакцией на запрос, не соответствующей ожиданиям.
- Так, например, если сервер получает сообщение, которое он не в состоянии распознать, и никаких мер по обработке подобных сообщений не предусмотрено, возникает ошибка передачи состояния.
- В частности, сервер может неправомерно осуществить по умолчанию некие действия, производить которые в данном случае не следовало бы.

произвольные ошибки (arbitrary failures)

- — Когда случается произвольная ошибка, клиент должен приготовиться к самому худшему.
- Например, может оказаться, что сервер генерирует сообщения, которые он в принципе не должен генерировать, но система не опознает их как некорректные. Хуже того, неправильно функционирующий сервер может, участвуя в работе группы серверов, приводить к появлению заведомо неверных ответов.
- Эта ситуация показывает, почему для надежных систем очень важна защита.

- Произвольные ошибки похожи на поломки. **Поломка** — наиболее распространенная причина остановки сервера. Поломки известны также под названием ошибок **аварийной остановки** .
- В действительности аварийно остановленный сервер просто прекращает генерировать исходящие сообщения. По этому признаку его остановка обнаруживается другими процессами.

- Например, по настоящему дружественный сервер может предупредить нас о том, что находится на грани поломки.
  Разумеется, в реальной жизни серверы, останавливаясь по причине пропуска данных или поломок, не настолько дружественны, чтобы оповестить нас о надвигающейся остановке.

- Другие процессы должны сами обнаружить «безвременную кончину» сервера.
- Однако в подобных системах остановки без уведомления другие процессы могут сделать неверный вывод об остановке сервера. Сервер может просто медленно работать, то есть может иметь место ошибка производительности.
  И, наконец, возможно, что сервер производит случайные сообщения, которые другие процессы считают абсолютным мусором. В этом случае мы имеем дело с наиболее простым случаем произвольной ошибки. Подобные ошибки называют **безопасными** .

Резервирование

Основные виды резервирования:

- Аппаратное резервирование (дублирование).
- Информационное резервирование (методы обнаружения и коррекции ошибок)
- Временное резервирование (методы альтернативной логики).
- Программное резервирование (кластер) (использование несколько функционально одинаковых программ, но, например, разных версий)

Основные подходы резервирования:

- **Кратность** резервирования (количество резервных "копий").
- По **времени "замены"**:
  - нагруженный (**горячий**) резерв — работает все время как и основной (RAID1);
  - облегчённый (**ждущий**) резерв — работает в ждущем режиме (например, резервный балансировщик нагрузки в кластере);
  - не нагруженный (**холодный**) резерв — "включены".
- В зависимости от **масштаба**:
  - **общий** резерв (например, приложения, файлы, БД)
  - **раздельный** (поэлементный) резерв - по отдельности (приложения, файлы, БД)
  - **смешанное** резервирование.

Контроль целостности данных

- Целостность информации (данных) — данные не были изменены (испорчены или подменены).

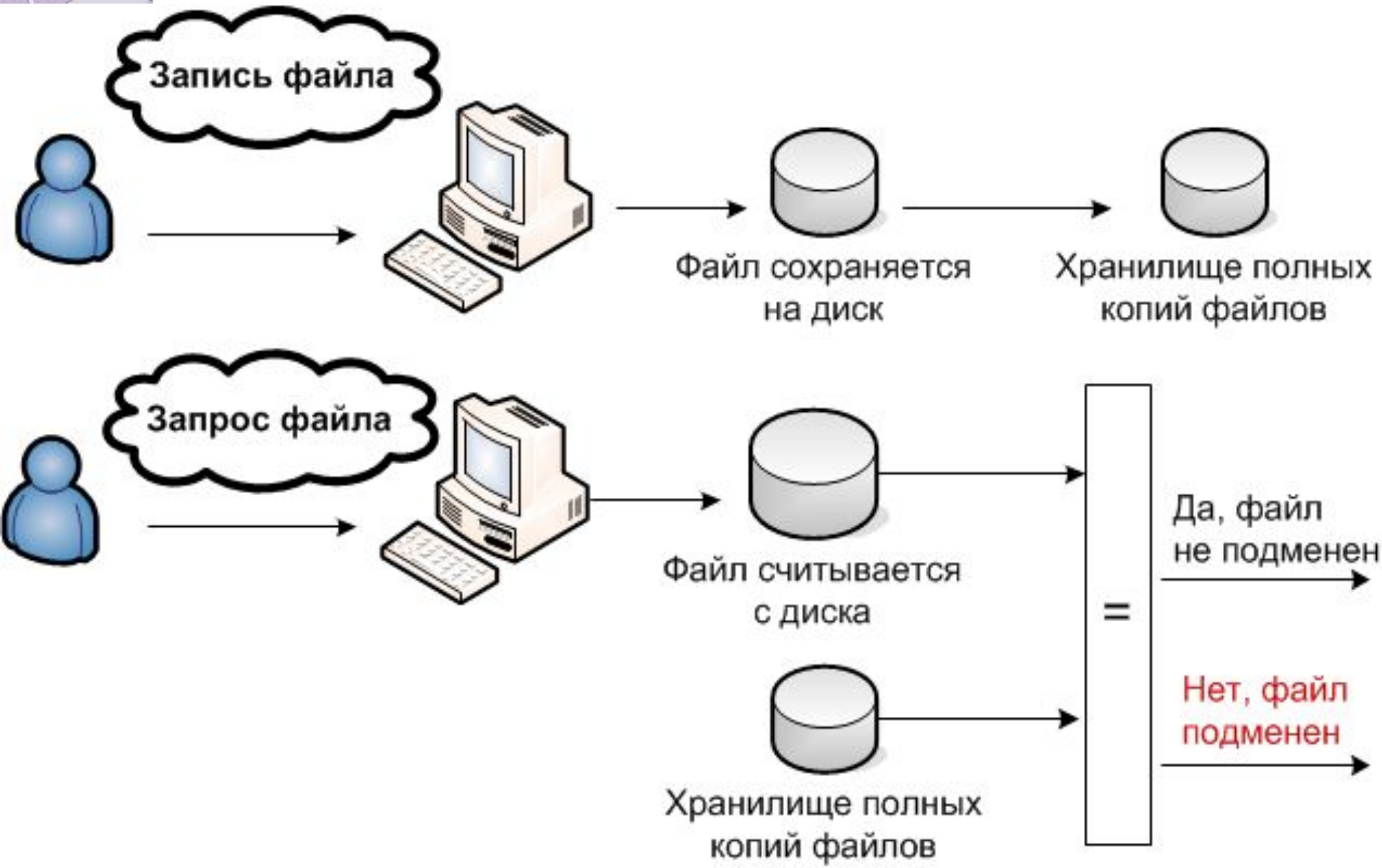
- Обнаружение ошибок - контроль целостности данных при записи/воспроизведении информации, при её передаче по линиям связи или хранении.
- Исправление ошибок (коррекция ошибок) — восстановление информации после чтения её из устройства хранения или канала связи.
- Хеширование (hashing) — преобразование входного массива данных произвольной длины в выходную битовую строку фиксированной длины.

- Целостности данных - при котором отсутствует любое ее изменение либо изменение осуществляется только преднамеренно субъектами, имеющими на него право.
- Методы контроля целостности данных:
- Полная копия данных
- Контрольная сумма
- Хеш
- Имитовставка
- ЭЦП

Полная копия данных.

- Создаются полные копии данных и потом сверяются.
- Преимущества:
- простота реализации
- полный контроль данных (до бита)
- Недостатки:
- большой объем
- копии можно подменить
- копиями можно воспользоваться (например: если данные - пароль)

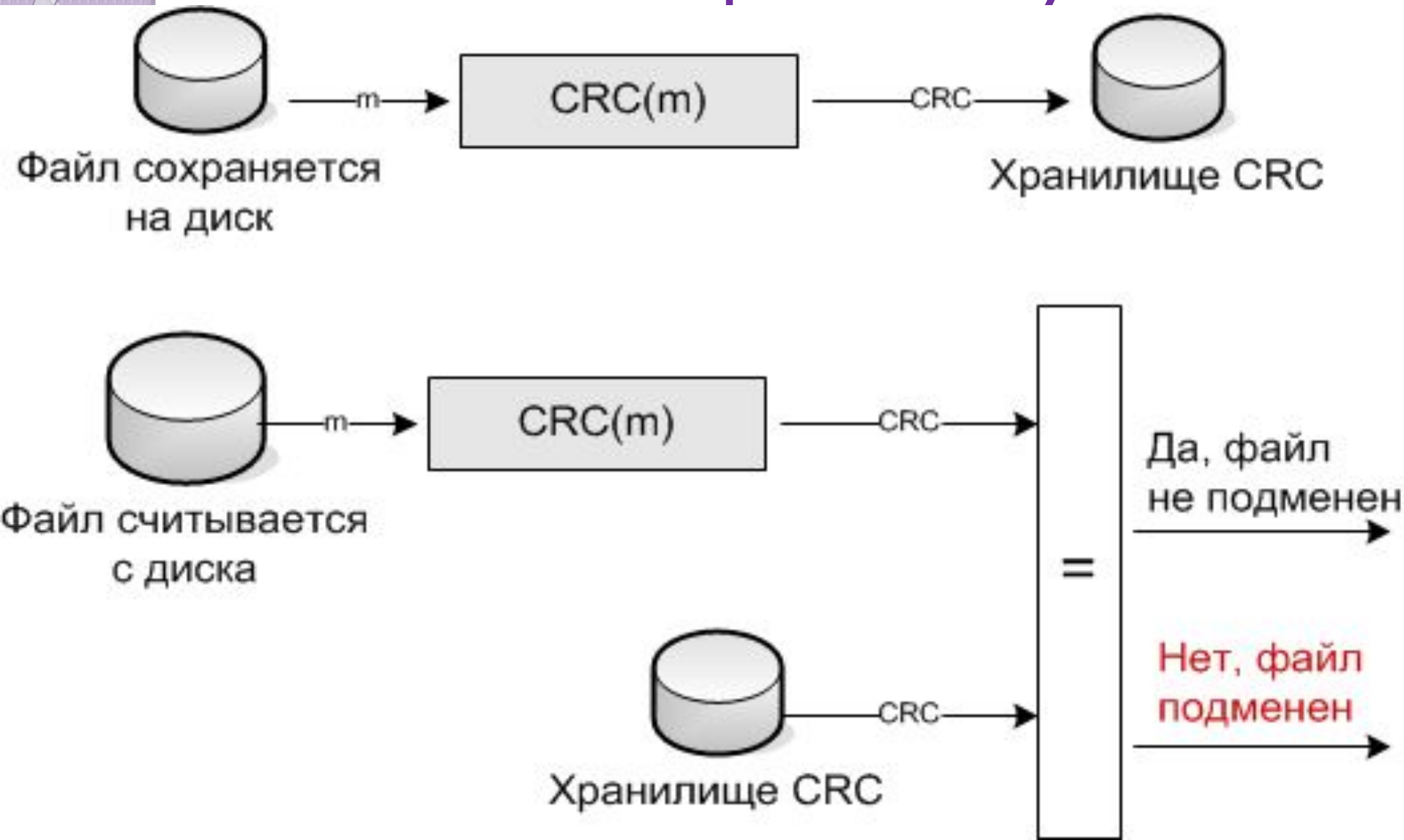
Контроль целостности с помощью полной копии данных



Контрольная сумма.

- Контрольная сумма - значение, рассчитанное по входным данным с помощью определённого алгоритма.
- Преимущества:
- высокая скорость вычисления
- малый размер
- стандартный размер
- Недостатки:
- можно подменить
- для одного значения существует множество исходных данных
- можно подобрать исходные данные к значению за приемлемое время (например: получить пароль)

Контроль целостности с помощью контрольной суммы

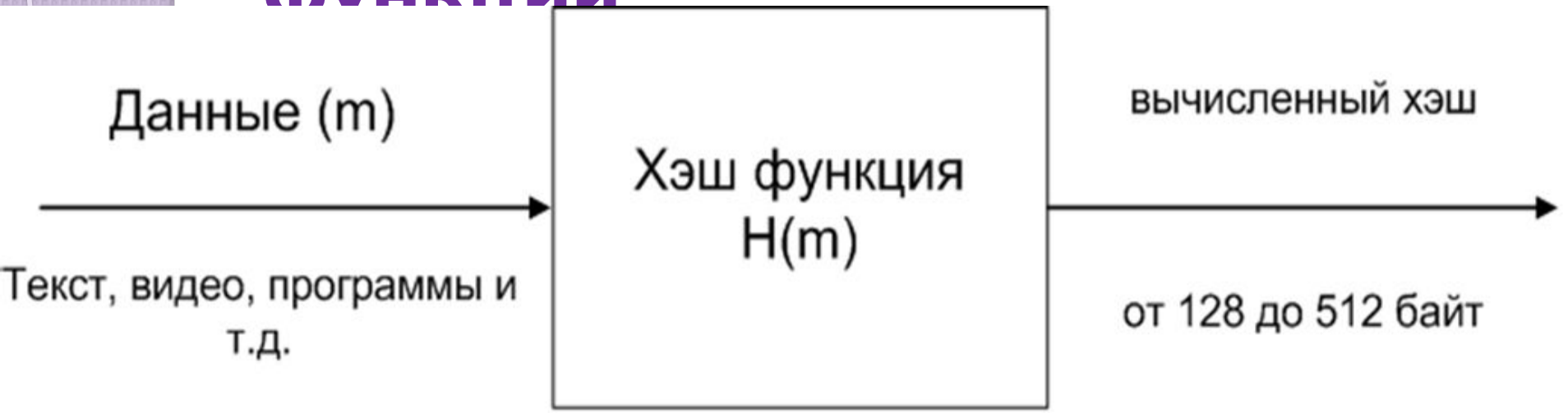


- Примеры контрольных сумм: CRC8, CRC16, CRC32
- Пример вычисления:
- исходный текст: Контроль целостности данных
- **crc32 (длина 32 бита)**
- b4feb5a5
-
- Применение:
- контроль целостности файлов
- контроль передаваемых данных по каналам связи
- контроль целостности при считывании данных (например: с HDD)

Хеш.

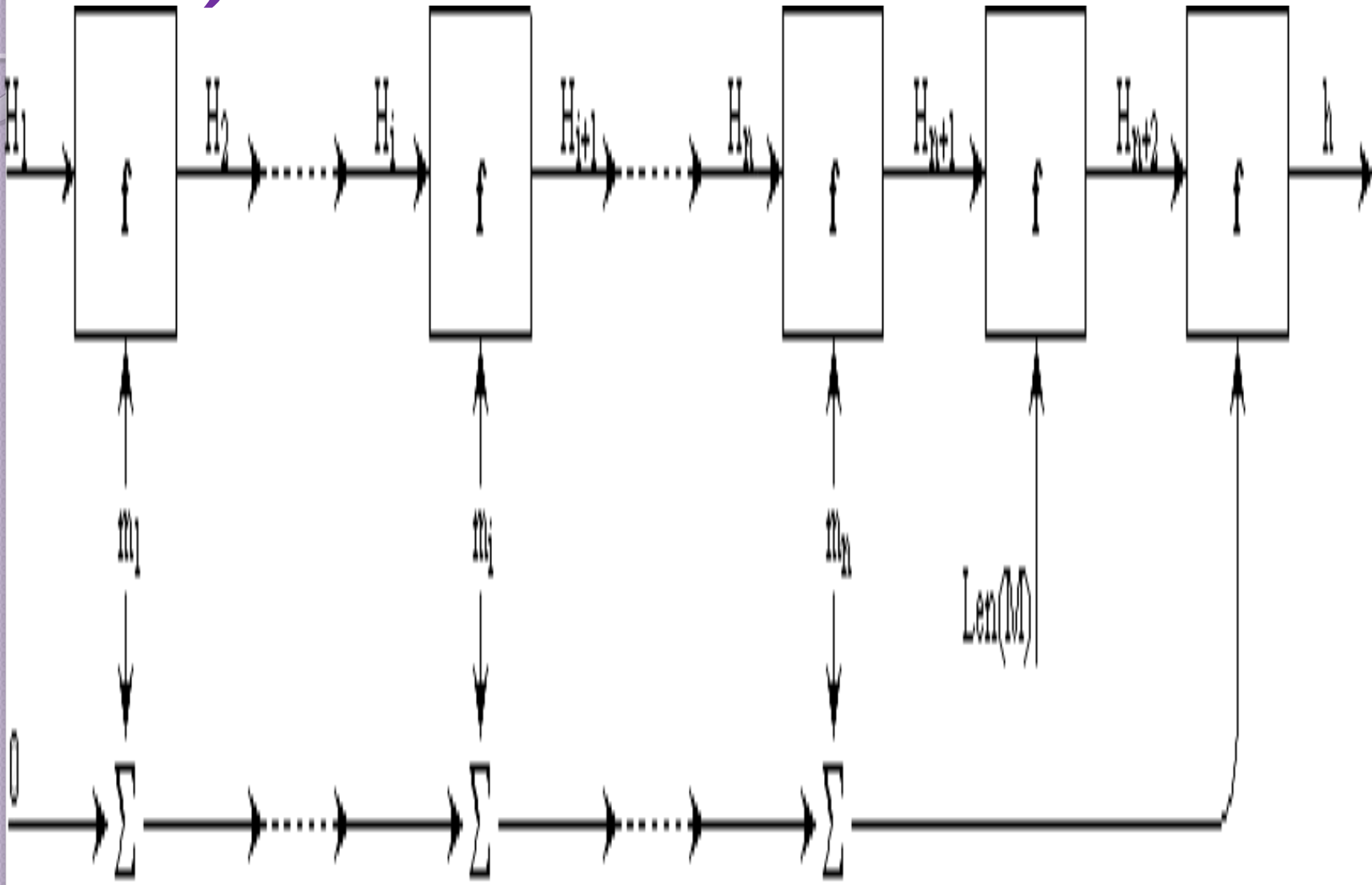
- Хеш (хэш, криптографический хеш) - значение, рассчитанное по входным данным с помощью криптографического алгоритма.
- Преимущества:
- малый размер
- стандартный размер
- нельзя подобрать исходные данные к значению за приемлемое время (например: получить пароль)
- Недостатки:
- низкая скорость вычисления (сопоставима с шифрованием)
- можно подменить
- для одного значения существует множество исходных данных

Основная задача хеш функций



- Вычисляют хеш шифрованием данных блочным алгоритмом в режимах CBC, но со стандартным (известным) ключом.
- Хешем является последний шифрованный блок.
- **ГОСТ Р 34.11-94**
- Входное сообщение M разделяется на блоки $m_n, m_{n-1}, m_{n-2}, \ldots, m_1$ по 256 бит. В случае если размер последнего блока m_n меньше 256 бит, то к нему приписываются слева нули для достижения заданной длины блока.
- Каждый блок сообщения подаётся на шаговую функцию для вычисления промежуточного значения хеш-функции $H_{out} = f(H_{in}, m_i)$ где H_{out}, H_{in}, m_i — блоки длины 256 бит.

Вычисление хеш по ГОСТ Р 34.11-94 (сравните с CBC)



Пример вычисления: исходный текст: Контроль целостности данных

- **md2 (длина 128 бит)**
- 7e347a5f3a3d8c6837d7accbed3e0b1e
- **md4 (длина 128 бит)**
- e37e491b9b4ea133df9964b25d7c7cd9
- **md5 (длина 128 бит)**
- 8ab8a5cf989e220ff8d39be415b903d5
- **sha1 (длина 160 бит)**
- 63cdc45bd8a857007d0be8c435e1e547653482d3
- **sha224 (длина 224 бит)**
- 670c98e5b7ce7bd2c7efb98b6ed448e0a6f3247e3fdeb678dde61bce
- **sha256 (длина 256 бит)**
- 1a97bcc0fbcf6722a8aac7b73d12700c0022778904790ce9eee7656f12d95dc2
- **sha384 (длина 384 бит)**
- b87b59cad0db141378d8ff04e46d00dd137113391d2a7009d640dd018680c459 310bfbdfef6b3258aeaa4b8146105698
- **sha512 (длина 512 бит)**
- 3f7b9d8b24fb1d764026fe2b0f72ad62d65fd1c5d77a18784108a69e8ad172c2 e6583989439aea658a2111525897d43af0f6c50cf299b360c21b3e02e8706f4e
- **ГОСТ Р 34.11-94 (длина 256 бит)**
- d38e4f1bc5d03601486f4aca83fed00c82e1a36fdac27806cce4b9464af1e9f9

Применение:

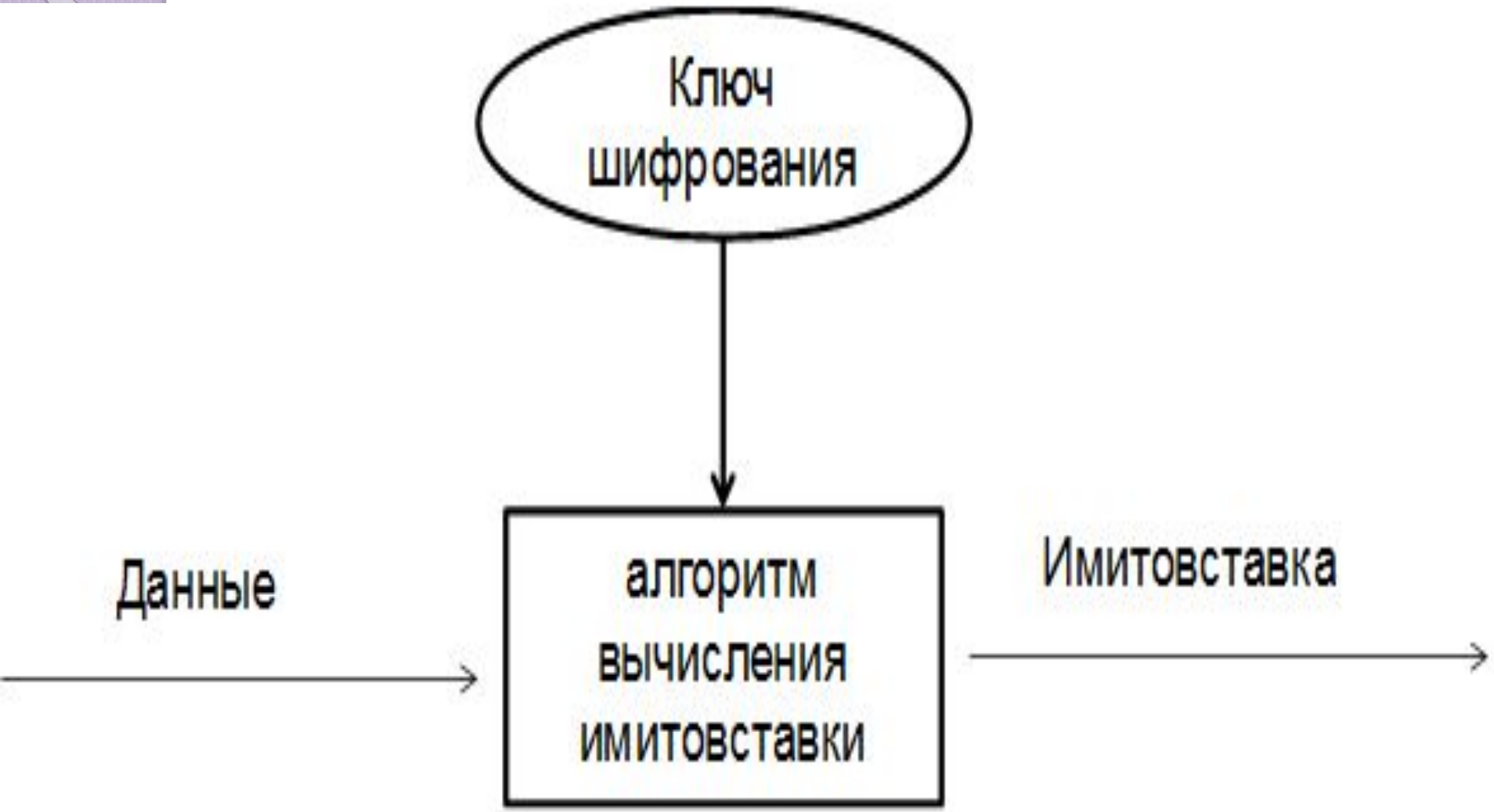
- контроль целостности файлов
- контроль передаваемых данных по каналам связи
- контроль целостности при считывании данных (например: с HDD)
- хеши паролей
- аутентификация (CRAM-MD5, DIGEST-MD5 и т.д.)

Имитовставка (MAC, message authentication code — код аутентичности сообщения)

- Имитовставка - значение, рассчитанное по входным данным с помощью криптографического алгоритма с использованием секретного элемента (ключа), известного только отправителю и получателю.

- **Преимущества:**
- малый размер
- стандартный размер
- нельзя подобрать исходные данные к значению за приемлемое время (например: получить пароль)
- нельзя подменить без секретного элемента (ключа)
- **Недостатки:**
- низкая скорость вычисления (сопоставима с шифрованием)
- для одного значения существует множество исходных данных
- секретный ключ известен как минимум двоим

Вычисляют имитовставку шифрованием данных блочным алгоритмом в режимах CBC. Имитовставкой является последний шифрованный блок.



Имитовставка по ГОСТ 28147-89

- Длина имитовставки от 1 до 32 бит.
- Открытый текст T_O разбивается на блоки длиной 64 бита. Последний блок в случае необходимости дополняется нулями.
- Первый блок шифруется, что и сообщение, но с применением 16 циклов вместо 32. Результат по битам по модулю 2 складывается с вторым блоком и так же шифруется. Результат складывается с третьим блоком... и так далее.
- Первые 32 бита получившегося блока составляют имитовставку. Спецификация шифра предусматривает использование в качестве имитовставки и меньшее количество бит по желанию, но не большее.

Проблема имитовставки

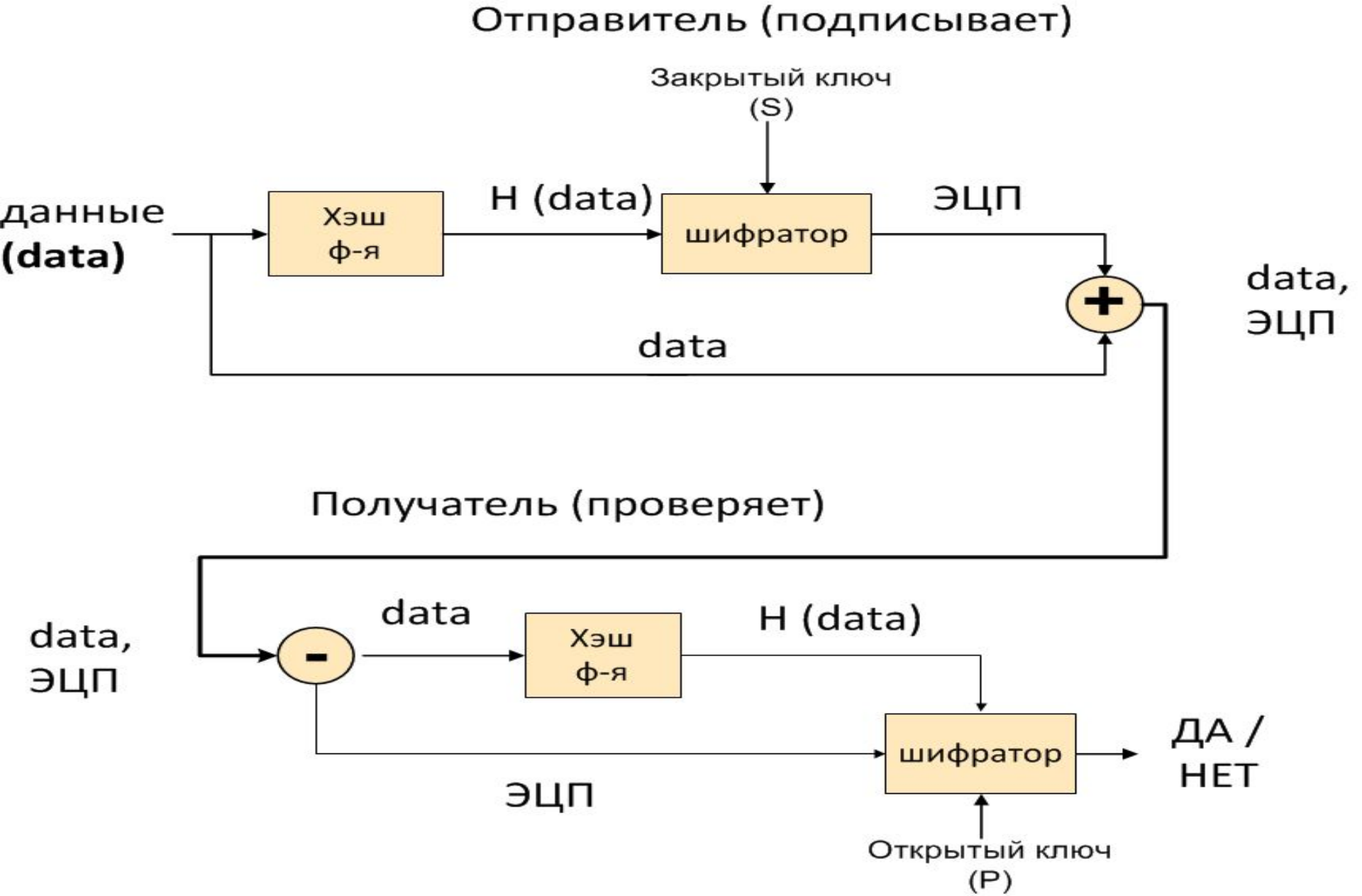
- Получатель должен знать ключ, и этот ключ позволяет ему генерировать сообщения с тем же значением имитовставки, что и у присланного сообщения, таким образом, имитовставка на основе симметричного шифра не дает знания — отправитель или получатель сформировал эту имитовставку.

- Обычные хэш-алгоритмы использовать для вычисления имитовставки нельзя (MD5 и т.д.) т.к. отсутствует секретный ключ.
- Поэтому создан **HMAC**.
- **HMAC (Hash-based Message Authentication Code)** - механизм включения секретного ключа в существующие хэш-алгоритмы.

ЭЦП.

- Электронная цифровая подпись - зашифрованное значение вычисленного хеша по входным данным.
- **Преимущества:**
- малый размер
- стандартный размер
- нельзя подобрать исходные данные к значению за приемлемое время (например: получить пароль)
- нельзя подменить без секретного элемента (ключа)
- секретный ключ известен одному
- **Недостатки:**
- низкая скорость вычисления (сопоставима с шифрованием)
- для одного значения существует множество исходных данных

Создание и проверка ЭЦП



- **Алгоритм:**
- вычисляется хеш
- шифруется хеш
-
- **Применение:**
- контроль целостности файлов
- контроль передаваемых данных по каналам связи
- аутентификации источника данных (кто создал подпись)