ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»
(Н И У « Б е л Г У »)

ИНСТИТУТ ИНЖЕНЕРНЫХ ТЕХНОЛОГИЙ И ЕСТЕСТВЕННЫХ НАУК

КАФЕДРА ИНФОРМАЦИОННО-ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМ И ТЕХНОЛОГИЙ

ИССЛЕДОВАНИЕ КРИПТОСТОЙКОСТИ МЕТОДОВ АУТЕНТИФИКАЦИИ ДАННЫХ

Выпускная квалификационная работа
обучающегося по направлению подготовки
11.04.02 Инфокоммуникационные технологии и системы связи,
магистерская программа «Системы и устройства радиотехники и связи»
очной формы обучения, группы 07001636
Гончарова Павла Александровича

Научный руководитель:
кандидат технических наук, с.н.с.,
доцент кафедры ИТСиТ
Буханцов А.Д.

Рецензент
кандидат технических наук, доцент,
доцент кафедры организации и
технологии защиты информации
Белгородского университета
кооперации, экономики и права
Земляченко В. В.

БЕЛГОРОД 2018

Список сокращений

АСС - агент системы справочника

АПС - агент пользователя справочника

ДИС - дерево информации справочника

ИБС - информационная база справочника

ИС - информационная система

ККОП - криптосистема ключа общего пользования

ПДС - протокол доступа к справочнику

УС - уполномоченный по сертификации

ЭП - электронная подпись

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Одним из столпов человеческой цивилизации является информационный обмен, который находится в непрерывном развитии со времен появления человека. Развитие методов информационного обмена являлось двигателем прогресса и позволяло человеку расширять горизонты знания за счет обмена и накопления информации об окружающем мире. С развитием методов информационного обмена росла и потребность в обеспечении достоверности информации. Так появление письменности и живописи привело к необходимости подтверждения подлинности или авторства.

Одним из наиболее древних методов подтверждения авторства является подпись – характерное для человека начертание символов. Однако данный метод хоть и является довольно устойчивым, имеет юридическую силу и активно используется по сегодняшний день, но обладает рядом недостатков: простота подделки или подмены, уничтожения или искажения, невозможность использования для подписи цифровых носителей. В связи с этим развивались другие методы подтверждения подлинности: печати, ключи, пароли.

Проверка подлинности усложнялась и развивалась, в современном мире процедуру проверки подлинности принято называть аутентификацией. В XXI веке с развитием методов цифровой передачи данных, обмен данных переходит на новый качественный уровень и требует особых подходов. В информационном обществе информация становится центральной ценностью создаваемой человечеством. Ее роль в современном мире настолько велика, что информационная индустрия стала одной из ведущих отраслей наших дней. Представленная в самых различных формах, информация, подобно другим товарам, производится, хранится, транспортируется к потребителю, продается, покупается и, наконец, потребляется, устаревает, портится.

Системы, работающие с информацией, получили название информационных (ИС). Сообщения, передаваемые в ИС требуется обеспечить надежными методами аутентификации, чтобы предотвратить их подлог, порчу и неправомерный доступ. Весь информационный обмен человека переносится в область ИС. Все больше защищаемой информации переносится в ИС. Современные информационные технологии не только обеспечивают новые возможности организации бизнеса, ведения государственной и общественной деятельности, но и создают значительные потребности в обеспечении безопасности для защиты информации.

Известно, что более 25 % злоупотреблений информацией в ИС совершаются внутренними пользователями, партнерами и поставщиками услуг, имеющими прямой доступ к ИС. До 70 % из них - случаи несанкционированного получения прав и привилегий, кражи и передачи учетной информации пользователей ИС, что становится возможным из-за несовершенства технологий разграничения доступа и аутентификации пользователей ИС. Совершенствование методов аутентификации одним из приоритетных направлений развития ИС.

Целью данной выпускной квалификационной работы является: исследование криптостойкости методов и алгоритмов аутентификации данных.

Для достижения поставленной цели необходимо сформировать следующие задачи, которые будут решены в ходе выполнения данной ВКР:

1) Провести анализ существующих методов и подходов к аутентификации данных;

2) Выполнить классификацию существующих подходов к аутентификации данных;

3) Выполнить программную реализацию хэш-функций: SHA-2, Keccak, ГОСТ Р 34.11-2012, используемых для аутентификации данных;

4) Провести сравнительные исследования реализованных хэш-функций для задач аутентификации данных;

5) Разработать рекомендации по использованию хэш-функций в задачах аутентификации данных.

ГЛАВА 1 АНАЛИЗ СУЩЕСТВУЮЩИХ ТЕХНОЛОГИЙ АУТЕНТИФИКАЦИИ В КОМПЬЮТЕРНЫХ СИСТЕМАХ

1.1 Простая аутентификация

Простая аутентификация предназначена для предоставления локальных полномочий на основе различительного имени пользователя, двусторонне согласованного (факультативно) пароля и двустороннего понимания способов использования и обработки этого пароля в пределах одного региона. Простая аутентификация предназначена в основном для локального использования, то есть для аутентификации равноправных логических объектов между одним АПС и одним АСС или между двумя АСС. Простая аутентификация может быть выполнена несколькими способами [20]:

a) передачей различительного имени пользователя и (факультативно) пароля в открытом (незащищенном) тексте получателю для оценки;

b) передачей различительного имени пользователя, пароля и случайного числа и/или отметкой времени и всего того, что защищено применением однонаправленной функции;

c) передачей защищенной информации, описанной в b), вместе со случайным числом и/или отметкой времени и всего того, что защищено применением однонаправленной функции.

Существует два замечания к изложенному материалу:

1. Не предъявляется никаких требований к тому, чтобы применение однонаправленных функций было различным.

2. Сигнализация процедур защищающих паролей может быть поводом для расширения документа.

Если пароли не защищены, то предполагается минимальная степень защиты от несанкционированного доступа. Это не должно рассматриваться как основа услуг защиты. Защита различительного имени пользователя и пароля обеспечивает большую степень защиты. Алгоритмы, которые должны

использоваться для механизма защиты, обычно не шифруются однонаправленными функциями, которые очень просты в реализации.

Общая процедура простой аутентификации приведена на рисунке 1.1.

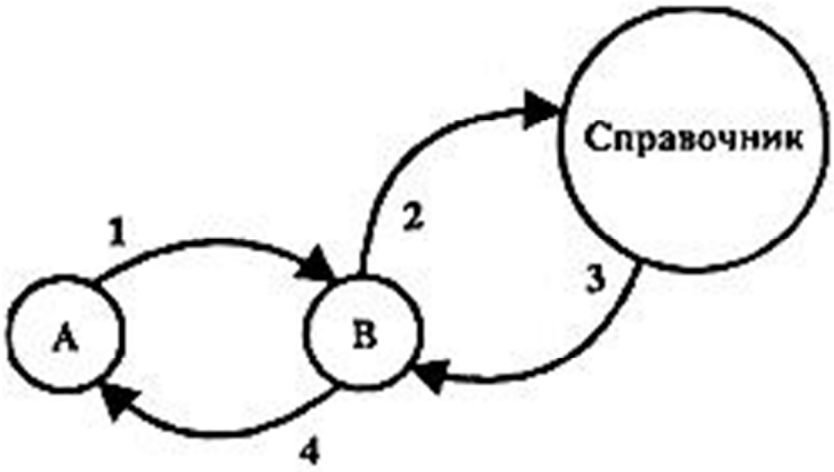


Рисунок 1.1 – Процедура незащищённой простой аутентификации

Процедура простой аутентификации состоит из следующих этапов:

1) пользователь - отправитель А посылает свои различительное имя и пароль пользователю - получателю В;

2) пользователь В посылает предполагаемое различительное имя и пароль пользователя А в справочник, где пароль проверяется относительно того пароля, который сохранен в виде атрибута парольПользователя в записи справочника для пользователя А, используя операцию сравнения справочника;

3) справочник подтверждает пользователю В (или отрицает) действительность удостоверения личности;

4) результат положительной (или отрицательной) аутентификации может быть передан пользователю А.

Самая простая форма аутентификации включает только шаг 1, а после проверки пользователем В различительного имени и пароля может выполняться шаг 4.

На рисунке 1.2 приведены два подхода генерации защищенной идентифицирующей информации f_1 и f_2 это однонаправленные функции

(одинаковые либо различные) и отметки времени, а случайные числа являются факультативными и подчиняются двусторонним соглашениям.

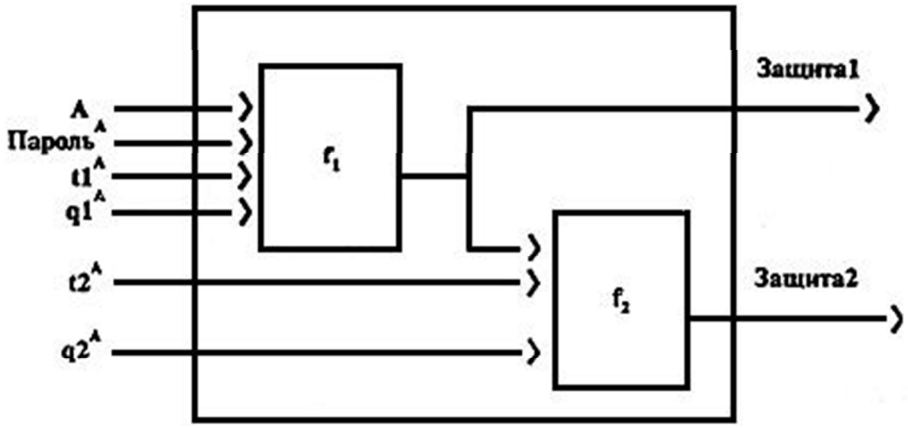


Рисунок 1.2 – Защищенная простая аутентификация

На рисунке 1.2 используются следующие обозначения: A – имя пользователя, t – отметка времени, Пароль A – пароль пользователя A, q – случайные числа.

На рисунке 1.3 представлена процедура защищенной простой аутентификации.

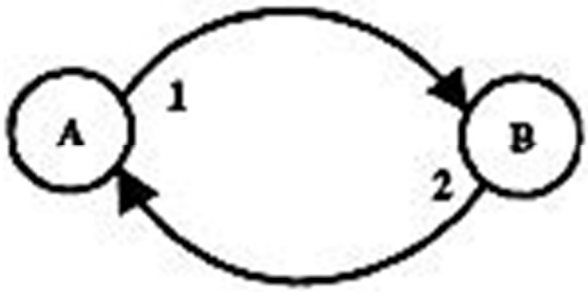


Рисунок 1.3 – Процедура защищенной простой аутентификации

Процедура защищенной простой аутентификации включает следующие шаги (первоначально использующие только f_1):

1) пользователь - отправитель A посылает свою защищенную идентифицирующую информацию (аутентификатор$_1$) пользователю B. Защита обеспечивается применением однонаправленной функции f_1, как на

рисунке 1.2, где отметка времени и/или случайное число (при его использовании) используются, чтобы уменьшить ответ и скрыть пароль.

Защита пароля пользователя А имеет вид:

$$Защита_1 = f_1(t_1^A, q_1^A, A, пароль_A) \quad (1.1)$$

При этом информация, переданная пользователю В, имеет вид:

$$Аутентификатор_1 = t_1^A, q_1^A, A, пароль_A \quad (1.2)$$

Пользователь В проверяет защищенную идентифицирующую информацию, предлагаемую пользователем А путем создания (используя различительное имя и факультативно отметку времени и/или случайное число, обеспечиваемые пользователем А, вместе с локальной копией пароля пользователя А) локальной защищенной копии пароля пользователя А (в виде Защита$_1$). Пользователь В сравнивает идентифицирующую информацию (Защита$_1$) с локально созданным значением;

2) пользователь В подтверждает пользователю А (или отрицает) наличие защищенной идентифицирующей информации.

С целью предоставления большей степени защиты процедура может быть изменена путем использования функций f_1 и f_2. Основные отличия состоят в следующем:

1) А посылает В дополнительно защищенную идентифицирующую информацию (Аутентификатор$_2$). Дополнительная защита достигается путем применения далее однонаправленной функции f_2, как показано на рисунке 1.2. Последующая защита имеет вид:

$$Защита_2 = f_2(t_2^A, q_2^A, Защита_1) \quad (1.3)$$

Информация, переданная В, имеет вид:

$$Аутентификатор_2 = (t_1^A, t_2^A, q_1^A, q_2^A A, Защита_2) \quad (1.4)$$

При выполнении операции сравнения В генерирует локальное значение дополнительно защищенного пароля и сравнивает его с значением Защита$_2$.

2) В подтверждает или отрицает для А верификацию защищенной идентифицирующей информации.

1.2 Строгая аутентификация

Принятый в спецификации ГОСТ Р ИСО/МЭК 9594-8-98 подход к строгой аутентификации основан на использовании свойств семейства криптографических систем, известных под названием "криптосистемы ключа общего пользования" (ККОП). Такие криптосистемы, описываемые также как асимметричные, используют пару ключей, один - секретный, а другой - общего пользования вместо одного ключа в соответствующих криптографических системах.

Эти основы аутентификации не предписывают использования конкретной криптосистемы. Задача состоит в том, чтобы эти основы могли быть применимы к любой приемлемой криптосистеме ключа общего пользования и тем самым поддерживали изменения в используемых методах, возникающие в результате дальнейших усовершенствований в криптографии, математических методах или вычислительных возможностях. Однако два пользователя, желающие осуществить аутентификацию, должны для правильного ее выполнения поддерживать один и тот же криптографический алгоритм. Таким образом, в контексте набора соответствующих применений выбор единственного алгоритма должен обеспечить расширение общества пользователей, способных к осуществлению аутентификации и обеспечению защищенного обмена.

Аутентификация рассчитана на любого пользователя [14], обладающего уникальным различительным именем. За распределение различительных имен несут ответственность уполномоченные по присвоению имен. Поэтому каждый пользователь должен доверить уполномоченным по присвоению имен право не выдавать дубликаты различительных имен.

Каждый пользователь идентифицируется на основе владения личным ключом. Другой пользователь может определить, обладает ли его партнер по обмену личным ключом, и может использовать это для подтверждения того,

что партнером по обмену действительно является данный пользователь. Достоверность этого подтверждения зависит от личного ключа, являющегося конфиденциальным для пользователя. [21]

Для того, чтобы пользователь мог определить, что партнер по обмену обладает личным ключом другого пользователя, он должен сам обладать ключом общего пользования этого пользователя. Если процедура получения значения ключа общего пользования непосредственно из записи пользователя в справочнике является достаточно прямолинейной, то проверка его правильности более проблематична. [18] Существует несколько способов решения этой проблемы: в разделе 8 описан процесс, с помощью которого ключ общего пользования может быть проверен путем обращения к справочнику. Такой процесс может действовать только в том случае, если между пользователями, нуждающимися в аутентификации, существует непрерывная цепочка доверительных точек в справочнике. Такая цепочка может быть построена путем идентификации общей точки доверия. Эта общая точка должна быть связана с каждым пользователем непрерывной цепочкой доверительных точек.

Для того, чтобы пользователь доверял процедуре аутентификации, он должен получить ключ общего пользования других пользователей от источника, которому он доверяет. Такой источник, называемый уполномоченным по сертификации (УС), для сертификации ключа общего пользования использует алгоритм ключа общего пользования. Сертификат, форма которого будет определена ниже в этом разделе, обладает следующими свойствами:

1) Любой пользователь, имеющий доступ к ключу общего пользования уполномоченного по сертификации, может восстановить ключ общего пользования, который был подтвержден;

2) Никто другой, кроме уполномоченного по сертификации, не может изменить сертификат без того, чтобы это не было обнаружено (сертификаты неподдельны).

Поскольку сертификаты неподдельны, их можно сделать общедоступными, поместив в справочнике без необходимости принятия специальных мер по их защите.

Уполномоченный по сертификации создает сертификат пользователя, подписывая собранную информацию, которая включает различительное имя пользователя и его ключ общего пользования, а также факультативный уникальный идентификатор, содержащий дополнительную информацию о пользователе. Точная форма содержимого уникального идентификатора здесь не определяется и оставлена на усмотрение уполномоченного по сертификации; она может иметь вид, например, идентификатора объекта, сертификата, даты или некоторый другой вид сертификата достоверности различительного имени. В частности, сертификат пользователя с различительным именем А и уникальным идентификатором УА, созданный уполномоченным по сертификации с именем УС и уникальным идентификатором УУС, имеет вид:

$$УС << A >> = УС\{B, ПН, ИА, УС, УУС, УА, A, A_0, T^A\} \quad (1.5)$$

где В – версия сертификата, ПН - порядковый номер сертификата, ИА – идентификатор алгоритма, используемый для подписания сертификата, УУС – факультативный уникальный идентификатор УС, УА – факультативный уникальный идентификатор пользователя А, $Т^А$ – указывает время действия сертификата и состоит из двух дат: начала и конца действия сертификата. Поскольку $Т^А$ может изменяться не чаще, чем через 24 ч, предполагается, что системы должны использовать в качестве базового эталонного времени Всемирное координированное время. Действительность подписи на сертификате может быть проверена любым пользователем, знающим $СУ_0$. Для представления сертификатов может быть использован тип данных АСН1.

Запись справочника каждого пользователя А, участвующего в строгой аутентификации, содержит сертификат(ы) А. Такой сертификат создается уполномоченным по сертификации пользователя А, являющимся логическим

объектом в ДИС. Уполномоченный по сертификации пользователя А, который может быть не уникальным, обозначается УС(А), или просто УС, если А подразумевается. Поэтому ключ общего пользования пользователя А может быть открыт любым пользователем, который знает ключ общего пользования УС. Таким образом, процесс раскрытия ключей общего пользования является рекурсивным.

Если пользователь А, пытаясь получить ключ общего пользования пользователя В, уже получил ключ общего пользования УС(В), то процесс заканчивается. Для того, чтобы А мог получить ключ общего пользования УС(В), запись справочника каждого уполномоченного по сертификации Х содержит несколько сертификатов. Существует два типа таких сертификатов. К первому типу относятся срочные сертификаты Х, созданные другими уполномоченными по сертификации. Ко второму - реверсивные сертификаты, созданные самим Х, которые являются заверенными общими ключами других уполномоченных по сертификации. Наличие таких сертификатов позволяет пользователям строить путь сертификации от одной точки к другой.

Список сертификатов, необходимый для того, чтобы конкретный пользователь мог получить общий ключ другого пользователя, называется "путь сертификации". Каждый элемент такого списка является сертификатом уполномоченного по сертификации следующего элемента в списке. Путь сертификации от А к В обозначим как $A \to B$.

Логически путь сертификации формирует непрерывную цепочку доверительных точек в дереве информации справочника между двумя пользователями, желающими выполнить аутентификацию. Точный метод, применяемый пользователями А и В для получения пути сертификации $A \to B$ и $B \to A$, может изменяться. Один из способов, ведущих к упрощению, состоит в том, чтобы построить иерархию УС, совпадающую с иерархией ДИС полностью или частично. Преимущество такого подхода состоит в том, что пользователи, имеющие УС в иерархии, могут, используя

справочник, установить между ними путь сертификации без какой-либо предварительной информации. Для того, чтобы достичь этого, каждый такой УС может хранить один сертификат и один реверсивный сертификат, предназначенный для передачи своему старшему УС.

1.2.1 Однонаправленная аутентификация

При однонаправленной аутентификации, рисунок 1.4, выполняются следующие шаги:

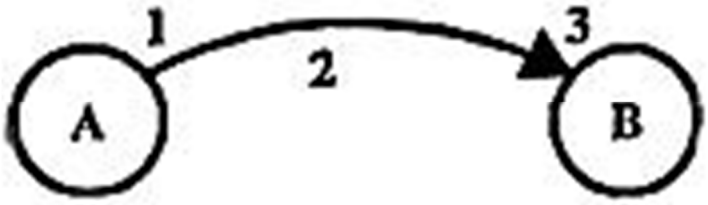


Рисунок 1.4 – Однонаправленная строгая аутентификация

Шаг 1. А создает r^A (неповторяющийся номер), который используется для обнаружения повторных угроз и предотвращения подделок;

Шаг 2. А посылает следующее сообщение к В:

$$B \to A, A\{t^A, r^A, B\}, \tag{1.6}$$

где t^A - отметка времени, которая состоит из одной или двух дат: даты создания маркера (который является факультативным) и даты истечения срока действия. Как вариант, если аутентификация отправителя данных "sgnData" должна обеспечиваться цифровой подписью, сообщение будет иметь вид:

$$B \to A, A\{t^A, r^A, B, \operatorname{sgn} Data\} \tag{1.7}$$

В тех случаях, когда передаваемая информация должна впоследствии использоваться в виде личного ключа (эта информация обозначается "encData"), сообщение будет иметь вид:

$$B \to A, A\{t^A, r^A, B, \operatorname{sgn} Data, B_0[encData]\} \tag{1.8}$$

Использование "encData" в виде личного ключа предполагает, что он должен очень внимательно выбираться, например, быть строгим ключом для любой используемой криптосистемы, как указано в поле "sgnData" маркера.

Шаг 3. Пользователь B выполняет следующие действия:

- получает A_0 *от* $B \rightarrow A$, проверяя, что срок сертификата пользователя A не истек;
- проверяет подпись, и тем самым целостность подписанной информации;
- проверяет, что он сам является назначенным получателем;
- проверяет, что отметка времени имеет значение “текущее”;
- проверяет факультативно, что r^A повторно не использован. Это может быть достигнуто, например, введением в r^A последующей части, которая проверяется локальной реализацией на уникальность ее значений.

Параметр r^A имеет силу до истечения даты, указанной t^A. При этом r^A всегда сопровождается последовательной частью, которая указывает, что A не должен повторять маркер в течение временного диапазона t^A и поэтому проверка значения самого r^A не требуется.[27]

В любом случае для стороны B целесообразно хранить в течение временного диапазона t^A последовательную часть вместе с отметкой времени t^A в открытом состоянии и вместе с хэшированной частью маркера.

1.2.2 Двунаправленная аутентификация

При двунаправленной аутентификации, рисунок 1.5, выполняются следующие шаги [27]:

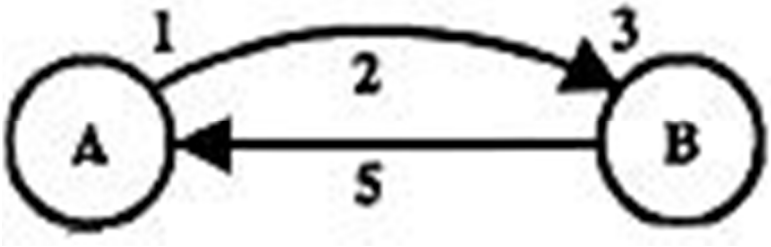


Рисунок 1.5 – Двунаправленная строгая аутентификация

Шаги 1-3 аналогичны однонаправленной строгой аутентификации.

Шаг 4. B создает r^B (неповторяющийся номер), который используется для обнаружения повторных угроз и предотвращения подделок;

Шаг 5. В посылает к А маркер последующей аутентификации:

$$B\{t^B, r^B, A, r^A\} \tag{1.9}$$

где t^B - отметка времени, определенная тем же способом что и t^A.

Как вариант, если аутентификация отправителя данных "sgnData" должна обеспечиваться цифровой подписью, маркер будет иметь вид:

$$B\{t^B, r^B, A, r^A, \mathrm{sgn}\, Data\} \tag{1.10}$$

В тех случаях, когда передаваемая информация должна впоследствии использоваться в виде личного ключа (эта информация обозначается "encData"), маркер будет иметь вид:

$$B\{t^B, r^B, A, r^A, \mathrm{sgn}\, Data, Ap[encData]\} \tag{1.11}$$

Использование "encData" в виде личного ключа предполагает, что он должен очень внимательно выбираться, например, быть строгим ключом для любой используемой криптосистемы, как указано в поле "sgnData" маркера.

Шаг 6. Пользователь А выполняет следующие действия:

- проверяет подпись, и тем самым целостность подписанной информации;
- проверяет, что он сам является назначенным получателем;
- проверяет, что отметка времени имеет значение “текущее”;
- проверяет факультативно, что r^A повторно не использован.

1.2.3 Трехнаправленная аутентификация

При трехнаправленной аутентификации, рисунок 1.6, выполняются следующие шаги [27]:

Шаг 1 выполняется аналогично двунаправленной строгой аутентификации.

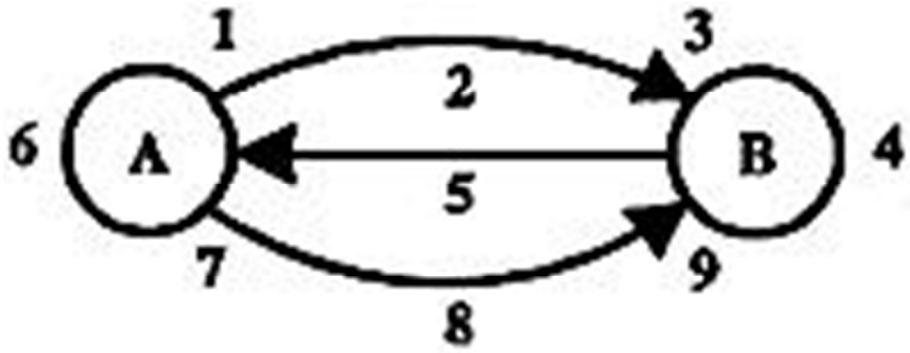


Рисунок 1.6 – Трехнаправленная строгая аутентификация

Шаг 2 выполняется аналогично двунаправленной строгой аутентификации. Отметка времени t^A может быть нулевой.

Шаг 3 выполняется аналогично двунаправленной строгой аутентификации за исключением того, что отметка времени не должна проверяться.

Шаг 4 выполняется аналогично двунаправленной строгой аутентификации.

Шаг 5 выполняется аналогично двунаправленной строгой аутентификации. Отметка времени t^B может быть нулевой.

Шаг 6 выполняется аналогично двунаправленной строгой аутентификации за исключением того, что отметка времени не должна проверяться.

Шаг 7 проверяет идентичен ли полученный r^A переданному r^A.

Шаг 8 А подсылает к В следующей маркер аутентификации:

$$A\{r^B, B\} \tag{1.12}$$

Шаг 9 В выполняет следующие действия:

- проверяет подпись и, тем самым, целостность подписанной информации;

- проверяет, что полученный r^B идентичен , r^B переданному стороной B.

1.3 Хэш-функции в задачах аутентификации

В настоящее время практически ни одно приложение криптографии не обходится без использования хэширования.

Хэш-функции – это функции, предназначенные для «сжатия» произвольного сообщения или набора данных, записанных, как правило, в двоичном алфавите, в некоторую битовую комбинацию фиксированной длины, называемую сверткой. [51] Хэш-функции имеют разнообразные применения при проведении статистических экспериментов, при тестировании логических устройств, при построении алгоритмов быстрого поиска и проверки целостности записей в базах данных. Основным требованием к хэш-функциям является равномерность распределения их значений при случайном выборе значений аргумента [31].

Криптографической хэш-функцией называется всякая хэш-функция, являющаяся криптостойкой, то есть удовлетворяющая ряду требований специфичных для криптографических приложений. В криптографии хэш-функции применяются для решения следующих задач:

— построения систем контроля целостности данных при их передаче или хранении,

— аутентификация источника данных.

Хэш-функцией называется всякая функция

$$h: X \to Y, \tag{1.13}$$

для которой бы выполнялось следующее условие, свертка для любого сообщения M, должна иметь фиксированную длину.

$$h(M) = H, \tag{1.14}$$

где X - множество сообщений, Y - множество двоичных векторов фиксированной длинны.

Как правило хэш-функции строят на основе так называемых одношаговых сжимающих функций $y = f(x_1, x_2)$ двух переменных где x_1, x_2, и y – двоичные векторы длины m, n и n соответственно, причем n – длина свертки, а m – длина блока сообщения. [22]

Для получения значения $h(M)$ сообщение сначала разбивается на блоки длинной m (при том, если длина сообщения не кратна m, то последний блок дополняется нулями до полного), затем к полученным блокам M_1, $M_2,...,M_N$ применяется последовательная процедура свертки:

$$\begin{aligned} &H_0 = v, \\ &H_i = f(M_i, H_{i-1}), i = 1,..,N, \\ &h(M) = H_N \end{aligned} \qquad (1.15)$$

Здесь v — некоторая константа, часто ее называют инициализирующим вектором. Она выбирается из различных соображений и может представлять собой секретную константу или набор случайных данных (выборку даты и времени, например). При таком подходе свойства хэш-функции полностью определяются свойствами одношаговой сжимающей функции. [34]

Выделяют два важных вида криптографических хэш-функций — ключевые и бесключевые [23]. Ключевые хэш-функции называют кодами аутентификации сообщений. Они дают возможность без дополнительных средств гарантировать как правильность источника данных, так и целостность данных в системах с доверяющими друг другу пользователями.

Бесключевые хэш-функции называются кодами обнаружения ошибок [38]. Они дают возможность с помощью дополнительных средств (шифрования, например) гарантировать целостность данных. Эти хэш-функции могут применяться в системах как с доверяющими, так и не доверяющими друг другу пользователями.

Основным требованием к хэш-функциям является равномерность распределения их значений при случайном выборе значений аргумента. Для криптографических хэш-функций также важно, чтобы при малейшем

изменении аргумента значение функции сильно изменялось. Это называется лавинным эффектом [17].

К ключевым функциям хэширования предъявляются следующие требования:

— невозможность фабрикации,

— невозможность модификации.

Первое требование означает высокую сложность подбора сообщения с правильным значением свертки. Второе — высокую сложность подбора для заданного сообщения с известным значением свертки другого сообщения с правильным значением свертки.

К бесключевым функциям предъявляют требования:

— однонаправленность,

— устойчивость к коллизиям,

— устойчивость к нахождению второго прообраза.

Под однонаправленностью понимают высокую сложность нахождения сообщения по заданному значению свертки [21]. Следует отметить, что на данный момент нет используемых хэш-функций с доказанной однонаправленностью.

Под устойчивостью к коллизиям понимают сложность нахождения пары сообщений с одинаковыми значениями свертки. Обычно именно нахождение способа построения коллизий криптоаналитиками служит первым сигналом устаревания алгоритма и необходимости его скорой замены [19].

Под устойчивостью к нахождению второго прообраза понимают сложность нахождения второго сообщения с тем же значением свертки для заданного сообщения с известным значением свертки.

1.4 Электронные подписи

При передаче электронных документов или сообщений по незащищенным каналам возникает угроза их перехвата. Злоумышленник может заполучить электронный документ, модифицировать его и отправить первоначальному адресату в целях извлечения выгоды. Чтобы избежать подобных ситуаций была разработана цифровая подпись. Правильно применённая цифровая подпись даёт получателю уверенность в том, что электронный документ достоверный и выслан указанным отправителем. Цифровая подпись для электронных документов является аналогом ручной – для бумажных носителей информации. Цифровая подпись также обеспечивает свойство неотказуемости. [27] Это означает, что подписывающий документ не сможет впоследствии отказаться от своей подписи.

Механизм электронной подписи заключается в следующем: информация подписывается путем добавления к ней зашифрованной сводки информации. Эта сводка вырабатывается с использованием однонаправленной хэш-функции, а шифрование выполняется с использованием личного ключа подписывающего лица, что проиллюстрировано на рисунке 1.7.

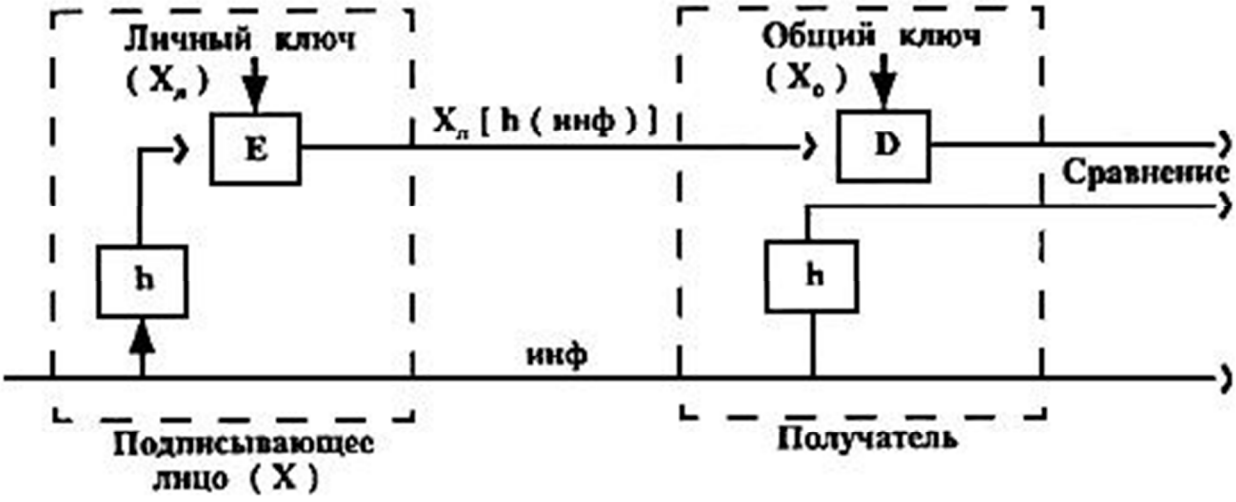


Рисунок 1.7 – Формирование электронной подписи [28]

Существует несколько схем построения цифровой подписи:

- На основе алгоритмов симметричного шифрования. Данная схема предусматривает наличие в системе третьего лица — арбитра, пользующегося доверием обеих сторон. Авторизацией документа является сам факт зашифрования его секретным ключом и передача его арбитру.
- На основе алгоритмов асимметричного шифрования. На данный момент такие схемы ЭП наиболее распространены и находят широкое применение.

Кроме этого, существуют другие разновидности цифровых подписей (групповая подпись, неоспоримая подпись, доверенная подпись), которые являются модификациями описанных выше схем. Их появление обусловлено разнообразием задач, решаемых с помощью ЭП. [7]

Поскольку подписываемые документы — переменного (и как правило достаточно большого) объёма, в схемах ЭП зачастую подпись ставится не на сам документ, а на его хэш. Для вычисления хэша используются криптографические хэш-функции, что гарантирует выявление изменений документа при проверке подписи. Хэш-функции не являются частью алгоритма ЭП, поэтому в схеме может быть использована любая надёжная хэш-функция. [7]

Использование хэш-функций даёт следующие преимущества:

- Вычислительная сложность. Обычно хэш цифрового документа делается во много раз меньшего объёма, чем объём исходного документа, и алгоритмы вычисления хэша являются более быстрыми, чем алгоритмы ЭП. Поэтому формировать хэш документа и подписывать его получается намного быстрее, чем подписывать сам документ.
- Совместимость. Большинство алгоритмов оперирует со строками бит данных, но некоторые используют другие представления. Хэш-функцию можно использовать для преобразования произвольного входного текста в подходящий формат.
- Целостность. Без использования хэш-функции большой электронный документ в некоторых схемах нужно разделять на достаточно

малые блоки для применения ЭП. При верификации невозможно определить, все ли блоки получены и в правильном ли они порядке. [7]

Использование хэш-функции не обязательно при электронной подписи, а сама функция не является частью алгоритма ЭП, поэтому хэш-функция может использоваться любая или не использоваться вообще. [33]

В большинстве ранних систем ЭП использовались функции с секретом, которые по своему назначению близки к односторонним функциям. Такие системы уязвимы к атакам с использованием открытого ключа, так как, выбрав произвольную цифровую подпись и применив к ней алгоритм верификации, можно получить исходный текст. [12] Чтобы избежать этого, вместе с цифровой подписью используется хэш-функция, то есть, вычисление подписи осуществляется не относительно самого документа, а относительно его хэша. В этом случае в результате верификации можно получить только хэш исходного текста, следовательно, если используемая хэш-функция криптографически стойкая, то получить исходный текст будет вычислительно сложно, а значит атака такого типа становится невозможной. [41]

1.4.1 Симметричная схема формирования подписи

Симметричные схемы ЭП менее распространены, чем асимметричные, так как после появления концепции цифровой подписи не удалось реализовать эффективные алгоритмы подписи, основанные на известных в то время симметричных шифрах. Первыми, кто обратил внимание на возможность симметричной схемы цифровой подписи, были основоположники самого понятия ЭП Диффи и Хеллман [19], которые опубликовали описание алгоритма подписи одного бита с помощью блочного шифра. Асимметричные схемы цифровой подписи опираются на вычислительно сложные задачи, сложность которых ещё не доказана, поэтому невозможно определить, будут ли эти схемы сломаны в ближайшее

время, как это произошло со схемой, основанной на задаче об укладке ранца. [31] Также для увеличения криптостойкости нужно увеличивать длину ключей, что приводит к необходимости переписывать программы, реализующие асимметричные схемы, и в некоторых случаях перепроектировать аппаратуру. Симметричные схемы основаны на хорошо изученных блочных шифрах.

В связи с этим симметричные схемы имеют следующие преимущества:

- Стойкость симметричных схем ЭП вытекает из стойкости используемых блочных шифров, надежность которых также хорошо изучена.
- Если стойкость шифра окажется недостаточной, его легко можно будет заменить на более стойкий с минимальными изменениями в реализации.

Однако у симметричных ЭП есть и ряд недостатков:

- Нужно подписывать отдельно каждый бит передаваемой информации, что приводит к значительному увеличению подписи. Подпись может превосходить сообщение по размеру на два порядка.
- Сгенерированные для подписи ключи могут быть использованы только один раз, так как после подписывания раскрывается половина секретного ключа.

Из-за рассмотренных недостатков симметричная схема ЭЦП Диффи-Хелмана [19] не применяется, а используется её модификация, разработанная Березиным и Дорошкевичем, в которой подписывается сразу группа из нескольких бит. Это приводит к уменьшению размеров подписи, но к увеличению объёма вычислений. Для преодоления проблемы «одноразовости » ключей используется генерация отдельных ключей из главного ключа.

1.4.2 Асимметричная схема формирования подписи

Асимметричные схемы ЭП относятся к криптосистемам с открытым ключом. [25] В отличие от асимметричных алгоритмов шифрования, в

которых шифрование производится с помощью открытого ключа, а расшифровка — с помощью закрытого, в асимметричных схемах цифровой подписи подписание производится с применением закрытого ключа, а проверка подписи — с применением открытого.

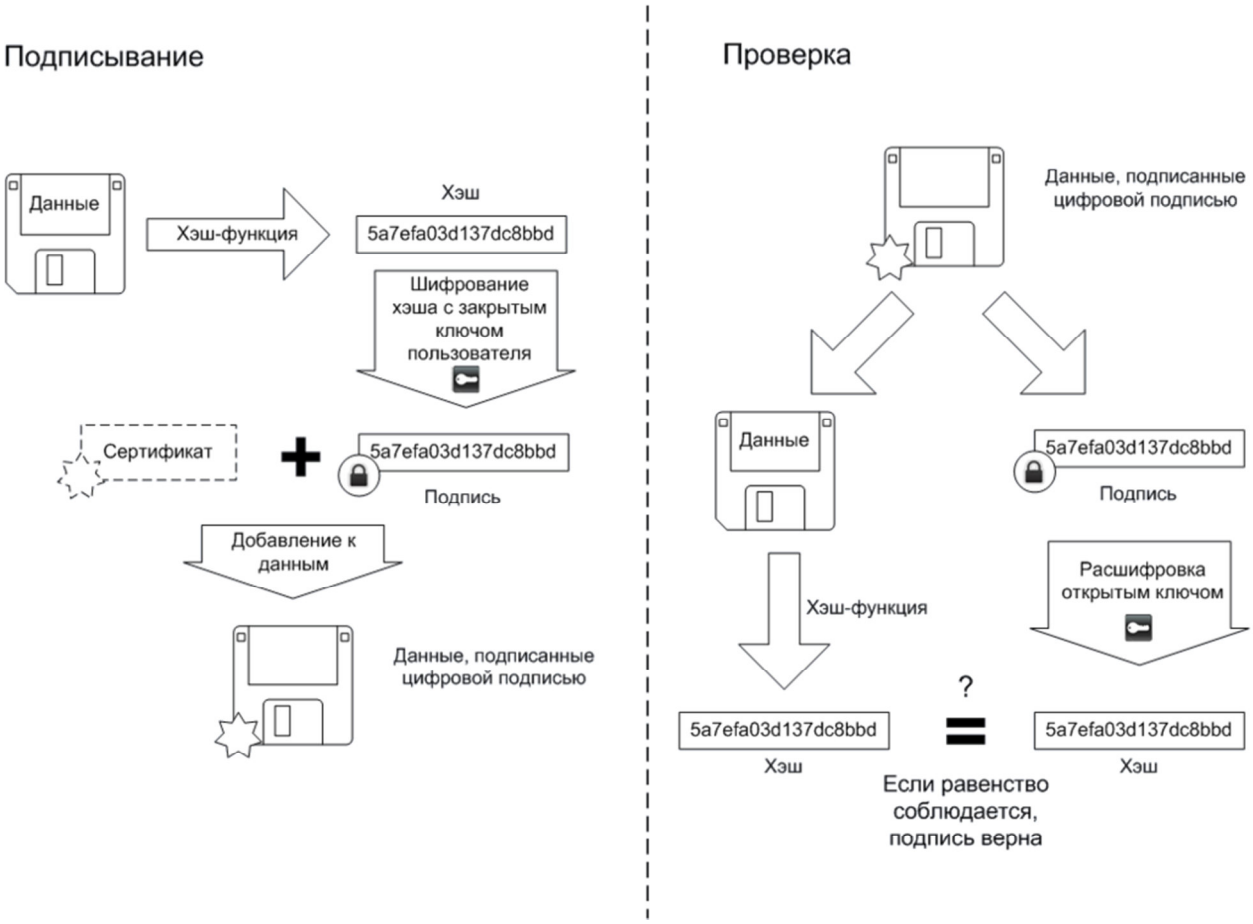


Рисунок 1.3 – Процедура защищенной простой аутентификации

Общепризнанная схема цифровой подписи охватывает три процесса:

1) Генерация ключевой пары. При помощи алгоритма генерации ключа равновероятным образом из набора возможных закрытых ключей выбирается закрытый ключ, вычисляется соответствующий ему открытый ключ.

2) Формирование подписи. Для заданного электронного документа с помощью закрытого ключа вычисляется подпись.

3) Проверка (верификация) подписи. Для данных документа и подписи с помощью открытого ключа определяется действительность подписи.

Для того, чтобы использование цифровой подписи имело смысл, необходимо выполнение двух условий:

- Верификация подписи должна производиться открытым ключом, соответствующим именно тому закрытому ключу, который использовался при подписании.
- Без обладания закрытым ключом должно быть вычислительно сложно создать легитимную цифровую подпись.

Следует отличать электронную цифровую подпись от кода аутентичности сообщения (MAC).

Вычисления тоже могут производиться двумя способами: на базе математического аппарата эллиптических кривых (ГОСТ Р 34.10-2012, ECDSA) и на базе полей Галуа (ГОСТ Р 34.10-94, DSA). В настоящее время самые быстрые алгоритмы дискретного логарифмирования и факторизации являются субэкспоненциальными. Принадлежность самих задач к классу NP-полных не доказана. [26]

Алгоритмы ЭП подразделяются на обычные цифровые подписи и на цифровые подписи с восстановлением документа. При верификации цифровых подписей с восстановлением документа тело документа восстанавливается автоматически, его не нужно прикреплять к подписи. Обычные цифровые подписи требуют присоединение документа к подписи. Ясно, что все алгоритмы, подписывающие хэш документа, относятся к обычным ЭП. К ЭП с восстановлением документа относится, в частности, RSA. [27]

Схемы электронной подписи могут быть одноразовыми и многоразовыми. В одноразовых схемах после проверки подлинности подписи необходимо провести замену ключей, в многоразовых схемах это делать не требуется. [51]

Также алгоритмы ЭП делятся на детерминированные и вероятностные. Детерминированные ЭП при одинаковых входных данных вычисляют одинаковую подпись. [37] Реализация вероятностных алгоритмов более

сложна, так как требует надежный источник энтропии, но при одинаковых входных данных подписи могут быть различны, что увеличивает криптостойкость. В настоящее время многие детерминированные схемы модифицированы в вероятностные.

В некоторых случаях, таких как потоковая передача данных, алгоритмы ЭП могут оказаться слишком медленными. В таких случаях применяется быстрая цифровая подпись. [23] Ускорение подписи достигается алгоритмами с меньшим количеством модульных вычислений и переходом к принципиально другим методам расчёта.

1.5 Задачи исследования

На основе проведенного анализа состояния вопроса относительно методов аутентификации и формирования хэш-функций были сформулированы следующие задачи:

1) Провести анализ существующих методов и подходов к аутентификации данных;

2) Выполнить классификацию существующих подходов к аутентификации данных;

3) Выполнить программную реализацию хэш-функций: SHA-2, Keccak, ГОСТ Р 34.11-2012, используемых для аутентификации данных.

4) Провести сравнительные исследования реализованных хэш-функций для задач аутентификации данных.

5) Разработать рекомендации по использованию хэш-функций в задачах аутентификации данных.

ГЛАВА 2 ИССЛЕДОВАНИЕ СОВРЕМЕННЫХ МЕТОДОВ ХЭШИРОВАНИЯ ДАННЫХ

2.1 Методы и функции хэширования, применяемые в системах аутентификации данных

Криптографические хэш-функции — это выделенный класс хэш-функций, который имеет определенные свойства, делающие его пригодным для использования в криптографии [24,35,41].

В общем случае в основе построения хэш-функции лежит итеративная последовательная схема, которая проиллюстрирована рисунком 2.1. Ядром алгоритма является сжимающая функция — преобразование k входных в n выходных бит, где n — разрядность хэш-функции, а k — произвольное число, большее n. При этом сжимающая функция должна удовлетворять всем условиям криптостойкости. [35,43]

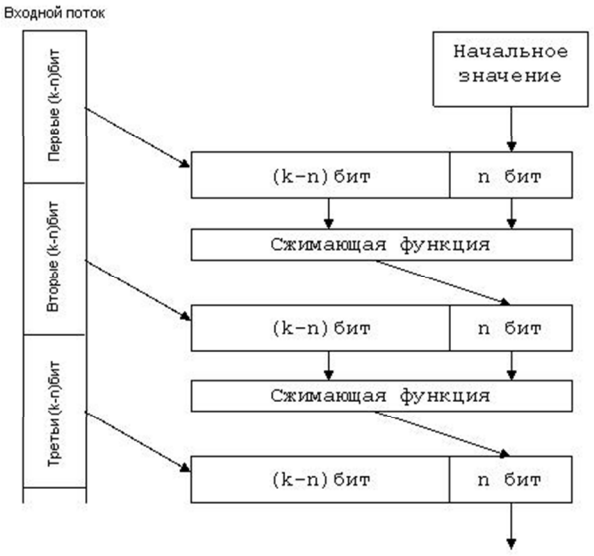


Рисунок 2.1 – Итеративная последовательная схема

Входной поток разбивается на блоки по (k − n) бит. Алгоритм использует временную переменную размером в n бит, в качестве начального значения которой берется некое общеизвестное число. Каждый следующий блок данных объединяется с выходным значением сжимающей функции на

предыдущей итерации. Значением хэш-функции являются выходные n бит последней итерации [35]. Каждый бит выходного значения хэш-функции зависит от всего входного потока данных и начального значения. Таким образом достигается лавинный эффект.

При проектировании хэш-функций на основе итеративной схемы возникает проблема с размером входного потока данных. Размер входного потока данных должен быть кратен $(k - n)$. Как правило, перед началом алгоритма данные расширяются неким, заранее известным, способом [37].

Помимо однопроходных алгоритмов, существуют многопроходные алгоритмы, в которых ещё больше усиливается лавинный эффект. В этом случае данные сначала повторяются, а потом расширяются до необходимых размеров.

2.2 Сжимающая функция на основе симметричного блочного алгоритма

В качестве сжимающей функции можно использовать симметричный блочный алгоритм шифрования. Для обеспечения большей безопасности можно использовать в качестве ключа блок данных, предназначенный к хэшированию на данной итерации, а результат предыдущей сжимающей функции — в качестве входа. Тогда результатом последней итерации будет выход алгоритма. В таком случае безопасность хэш-функции базируется на безопасности используемого алгоритма [28].

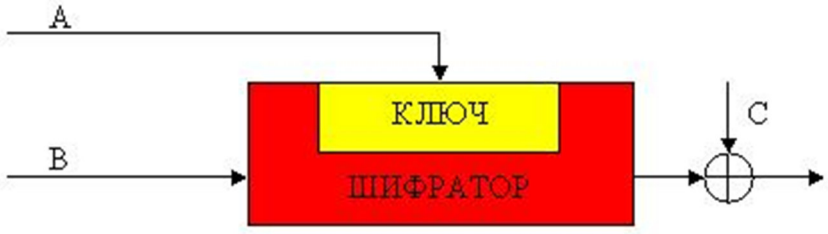


Рисунок 2.2 – Обобщенная схема формирования хэш-функции

Обычно при построении хэш-функции используют более сложную систему. Обобщённая схема симметричного блочного алгоритма шифрования изображена на рисунке 2.2.

Таким образом, мы получаем 64 варианта построения сжимающей функции. Большинство из них являются либо тривиальными, либо небезопасными. Ниже, на рисунке 2.3, изображены четыре наиболее безопасные схемы хэширования [28].

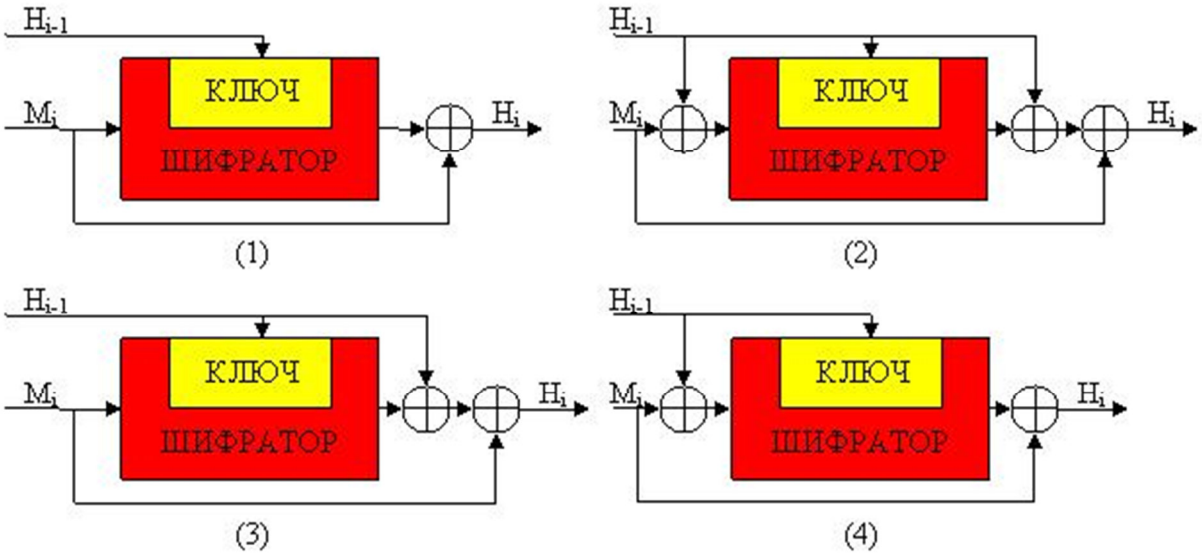


Рисунок 2.3 – Схемы безопасного хэширования

Основным недостатком хэш-функций, спроектированных на основе блочных алгоритмов, является низкая скорость работы. Необходимую криптостойкость можно обеспечить и за меньшее количество операций над входными данными. Существуют более быстрые алгоритмы хэширования, спроектированных самостоятельно, с нуля, исходя из требований криптостойкости. Наиболее распространенные из них — MD5, SHA-1, SHA-2.

2.3 Требования к криптографическим хэш-функциям

К криптографическим хэш-функциям предъявляются следующие требования:

1. Стойкость к поиску первого прообраза — отсутствие эффективного полиномиального алгоритма вычисления обратной функции, т.е. нельзя восстановить текст m по известной его свертке *H(m)* за реальное время (необратимость). Это свойство эквивалентно тому, что хэш-функция является односторонней функцией [31].

2. Стойкость к поиску второго прообраза — вычислительно невозможно, зная сообщение m и его свертку *H(m)*, найти такое другое сообщение $m' \neq m$, чтобы *H(m) = H(m')*.

3. Стойкость к коллизиям. Коллизией для хэш-функции называется такая пара значений *m* и *m'*, $m \neq m'$, для которой *H(m) = H (m')*. Так как количество возможных открытых текстов больше числа возможных значений свертки, то для некоторой свертки найдется много прообразов, а, следовательно, коллизии для хэш-функций обязательно существуют. Например, пусть длина хэш-прообраза 6 битов, длина свертки 4 бита. Тогда число различных сверток – $2^4 = 16$, а число хэш-прообразов – $2^6 = 64$, т.е. в 4 раза больше, значит хотя бы одна свертка из всех соответствует 4 прообразам.

Стойкость хэш-функции к коллизиям означает, что нет эффективного полиномиального алгоритма, позволяющего находить коллизии.

Стоит отметить, что данные свойства не являются независимыми:

- Обратимая функция неустойчива к восстановлению второго прообраза и коллизиям.
- Функция, нестойкая к восстановлению второго прообраза, нестойка к коллизиям; обратное неверно.
- Функция устойчивая к коллизиям, устойчива к нахождению второго прообраза.

- Устойчивая к коллизиям хэш-функция не обязательно является односторонней.

Для криптографии важно, чтобы значения хэш-функции сильно изменялись при малейшем изменении аргумента (лавинный эффект). Значение хэша не должно давать утечки информации даже об отдельных битах аргумента.

При разработке современного российского стандарта ГОСТ Р 34.11-2012 (Стрибог) к криптографическим хэш-функциям были сформулированы следующие требования [18,28,32]:

1. Сложность к вычислению прообраза: если известно значение функции, тогда должно быть сложно найти такое сообщение, хэш-функция от которого равна известному;

2. Стойкость вычисления второго прообраза: пусть есть одно значение, и известен хэш-код этого значения. Тогда злоумышленнику должно быть сложно найти еще одно такое значение, чтобы его хэш-функция совпадала с хэш-функцией первого значения;

3. Сложность к поиску коллизий: должно быть сложно найти два таких сообщения, которые не равны, но у них равны хэш-коды;

4. Стойкость к удлинению прообраза: если злоумышленник не знает сообщение, но знает его длину и хэш-код от него, то ему должно быть сложно подобрать такое сообщение, которое, будучи дописанным к оригинальному, даст какую-нибудь известную хэш-функцию. Другими словами, не должно быть возможно злоумышленнику что-то менять путем дополнения в сообщении, получая известны выход. Это можно сформулировать по-другому: хэш-функция не должна быть хорошо аддитируема.

2.4 Идеальная криптографическая хэш-функция

Идеальной криптографической хэш-функцияей является такая криптографическая хэш-функция, к которой можно отнести пять основных свойств:

1. Детерминированность. При одинаковых входных данных результат выполнения хэш-функции будет одинаковым (одно и то же сообщение всегда приводит к одному и тому же хэшу);

2. Высокая скорость вычисления значения хэш-функции для любого заданного сообщения;

3. Невозможность сгенерировать сообщение из его хэш-значения, за исключением попыток создания всех возможных сообщений;

4. Наличие лавинного эффекта. Небольшое изменение в сообщениях должно изменить хэш-значения, так широко, что новые хэш-значения не совпадают со старыми хэш-значениями;

5. Невозможность найти два разных сообщения с одинаковыми хэш-значениями.

Таким образом, идеальная криптографическая хэш-функция, у которой длина n (то есть на выходе n бит), для вычисления прообраза должна требовать как минимум 2^n операций.

Злоумышленник будет искать прообраз для идеальной хэш-функции следующим образом: у него есть число *h*, и ему надо найти такое m, что $H(m)=h$. Если это идеальная хэш-функция, то злоумышленнику остается лишь перебирать все возможные *M* и проверять, чему равна хэш-функция от этого сообщения. Результат вычисления, если m перебирается полностью, есть фактически случайное число. Если число *h* лежит в диапазоне от 0 до 2^n, то тогда в среднем на поиски нужного h злоумышленник будет тратить 2^{n-1} итераций. Таким образом, вычисление прообраза займёт в два раза меньше итераций, чем в идеальном случае.

Вычисление второго прообраза останется 2^n. В поиске коллизий оценка даст 2^n, причём это не совсем точный результат. Данная оценка идет из оценки так называемого «Парадокса дней рождения». [27]

Если злоумышленник хочет написать программу по поиску коллизий, ему будет оптимально вначале завести себе словарь коллизий. Соответственно, дальше он вычисляет хэш-функцию от очередного сообщения и проверяет, принадлежит эта хэш-функция очередному сообщению или нет. Если принадлежит, то коллизия найдена, и тогда можно найти по словарю исходное сообщение с данным хэш-кодом. Если нет, то он пополняет словарь. На практике такой способ не реализуется, потому что не хватило бы памяти для подобного словаря.

2.5 Семейство хэш-функций MD и SHA

На сегодняшний день подавляющую долю применений хэш-функций берут на себя алгоритмы MD5, SHA-1, SHA-256, а в России еще и ГОСТ Р 34.11-2012 (Стрибог). Конечно, существует и множество других менее известных, или распространенных только в узких сообществах алгоритмов (например, RIPEMD, TIGER, Panama и др.), однако эти алгоритмы не так распространены. Ниже представлен анализ хэш-функций MD4, которая была предшественником MD5, а также хэш-функции SHA.

Таблица 2.1 – Описание основных хэш-функций MD и SHA

Тип	Описание
MD4	Самая быстрая, оптимизирована для 32-битных машин среди семейства MD-функций. Хэш-функция, разработанная профессором Массачусетского университета Рональдом Ривестом в 1990 году и впервые описанная в RFC 1186. Содержит три цикла по 16 шагов каждый. В 1993 году был описан алгоритм взлома MD4, поэтому на сегодняшний день данная функция не рекомендована для использования с реальными приложениями.

Тип	Описание
MD5	Наиболее распространенная из семейства MD-функций. Похожа на MD4, но средства повышения безопасности делают ее на 33% медленнее, чем MD4. Содержит четыре цикла по 16 шагов каждый. Обеспечивает контроль целостности данных. Первые успешные попытки взлома данной хэш-функции датируются 1993: исследователи Берт ден Боер и Антон Боссиларис показали, что в алгоритме возможны псевдоколлизии. В 1996 году Ганс Доббертин показал наличие возможности коллизий и теоретически описал алгоритм взлома. 24 августа 2004 года четыре независимых исследователя — Ван Сяоюнь, Фэн Дэнгуо, Лай Сюэцзя и Юй Хунбо — обнаружили уязвимость алгоритма, позволяющую найти коллизии аналитическим методом за более-менее приемлемое время. В 2005 году Властимил Клима опубликовал алгоритм, позволяющий обнаруживать коллизии за несколько часов. Восемнадцатого марта 2006 года исследователь обнародовал алгоритм, находящий коллизии за одну минуту, который позднее получил название «туннелирование». На сегодняшний день MD5 не рекомендована для использования в реальных приложениях.
SHA-1	В 1993 году NSA совместно с NIST разработали алгоритм безопасного хэширования (сейчас известный как SHA-0) (опубликован в документе FIPS PUB 180) для стандарта безопасного хэширования. Однако вскоре NSA отозвало данную версию, сославшись на обнаруженную ими ошибку, которая так и не была раскрыта. И заменило его исправленной версией, опубликованной в 1995 году в документе FIPS PUB 180-1. Эта версия и считается тем, что называют SHA-1. Позже, на конференции CRYPTO в 1998 году два французских исследователя представили атаку на алгоритм SHA-0, которая не работала на алгоритме SHA-1. Возможно, это и была ошибка, открытая NSA. SHA-1 создает 160-битное значение, называемое также дайджестом сообщения. Содержит четыре этапа. И MD5, и SHA-1 являются, по сути, улучшенными продолжениями MD4. Различия: В SHA-1 на четвертом этапе используется та же функция, что и на втором этапе. В MD5 в каждом действии используется уникальная прибавляемая константа. В SHA-1 константы используются повторно для каждой из четырех групп.

Тип	Описание
	В SHA-1 добавлена пятая переменная. SHA-1 использует циклический код исправления ошибок. В MD5 четыре различных элементарных логических функции, в SHA-1 - три. В MD5 длина дайджеста составляет 128 бит, в SHA-1 — 160 бит. SHA-1 содержит больше раундов (80 вместо 64) и выполняется на 160-битном буфере по сравнению со 128-битным буфером MD5. Таким образом, SHA-1 должен выполняться приблизительно на 25 % медленнее, чем MD5 на той же аппаратуре.
SHA-2	Семейство криптографических алгоритмов — хэш-функций, включающее в себя алгоритмы SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/256 и SHA-512/224. В 2003 году Гилберт и Хандшух провели исследование SHA-2, но не нашли каких-либо уязвимостей. Однако в марте 2008 года индийские исследователи Сомитра Кумар Санадия и Палаш Саркар опубликовали найденные ими коллизии для 22 итераций SHA-256 и SHA-512. В сентябре того же года они представили метод конструирования коллизий для усечённых вариантов SHA-2 (21 итерация). Как показали исследования, алгоритмы SHA-2 работают в 2—3 раза медленнее хэш-алгоритмов MD5, SHA-1.
SHA-3 (*Keccak)*	Хэш-функция SHA-3 (также называемая Keccak) является функцией переменной разрядности, разработанная группой авторов во главе с Йоаном Дайменом. 2 октября 2012 года Keccak стала победителем конкурса криптографических алгоритмов, проводимым Национальным институтом стандартов и технологий США. 5 августа 2015 года алгоритм функции был утверждён и опубликован в качестве стандарта FIPS 202. Алгоритм функции SHA-3 построен по принципу криптографической губки.

2.6 Требования к алгоритмам шифрования, используемых для хэширования

Рассмотрев основные способы применения алгоритмов хэширования с открытым ключом, изучим требования, которым должен, по мнению основоположников теории шифрования с открытым ключом Диффи и Хеллмана, удовлетворять алгоритм шифрования с открытым ключом. Эти требования следующие:

1. Вычислительно легко создавать пару (открытый ключ, закрытый ключ).
2. Вычислительно легко зашифровать сообщение открытым ключом.
3. Вычислительно легко расшифровать сообщение, используя закрытый ключ.
4. Вычислительно невозможно, зная открытый ключ, определить соответствующий закрытый ключ.
5. Вычислительно невозможно, зная только открытый ключ и зашифрованное сообщение, восстановить исходное сообщение.

Из этих общих требований видно, что реализация конкретного алгоритма с открытым ключом зависит от соответствующей односторонней функции.

Математиками и криптографами предложено большое количество алгоритмов шифрования с открытым ключом, основанных на различных односторонних функциях. Некоторые алгоритмы можно задействовать тремя, рассмотренными ранее в данной лекции способами, в то время как другие могут использоваться только одним или двумя способами. Мы рассмотрим четыре алгоритма с открытым ключом, три из которых достаточно давно применяются на практике, а четвертый вид алгоритмов совсем недавно начал применяться в системах защиты информации. Эти алгоритмы используются обычно для различных целей, что отражено в следующей таблице 2.2.

Таблица 2.2 – Описание основных алгоритмов хэш-функций MD и SHA

Название алгоритма	**Возможность использования**		
	Шифрование / расшифрование данных	Цифровая подпись	Согласование или формирование ключа
RSA	Да	Да	Да
Алгоритм Диффи-Хеллмана	Нет	Нет	Да
Алгоритм Эль-Гамаля	Да	Да	Да
Алгоритмы с использованием эллиптических кривых	Да	Да	Да

Чтобы убедиться, что сообщение отправил конкретный отправитель, вместе с сообщением передаётся так называемая электронная подпись. Получатель проверяет, действительно ли электронная подпись (ЭП) относится к данному сообщению.

В связи с тем, что использование криптографии с открытыми ключами (подписывание, проверка подписей и т. д.), процесс очень медленный, более того, если подписывать всё сообщение целиком, то размеры этой подписи будут сопоставимы с размером сообщения; подписывают не сообщение, а хэш-функцию от сообщения. И далее получатель, когда расшифровывает подпись, получает хэш-функцию. Далее он сравнивает хэш-функцию от того сообщения, которое он получил, и хэш-функцию, которая была получена в результате расшифровки. За счет того, что хэш-функция имеет фиксированную длину, она меньше, чем само сообщение. Это позволяет быстро вычислить электронную цифровую подпись. Размер этой подписи будет мал по сравнению с размером сообщения.

Существует несколько схем построения цифровой подписи:

На основе алгоритмов симметричного шифрования. Данная схема предусматривает наличие в системе третьего лица — арбитра, пользующегося доверием обеих сторон. Авторизацией документа является сам факт зашифрования его секретным ключом и передача его арбитру.

На основе алгоритмов асимметричного шифрования. На данный момент такие схемы ЭП наиболее распространены и находят широкое применение.

Кроме этого, существуют другие разновидности цифровых подписей (групповая подпись, неоспоримая подпись, доверенная подпись), которые являются модификациями описанных выше схем. Их появление обусловлено разнообразием задач, решаемых с помощью ЭП.

Далее была выполнена программная реализация различных алгоритмов, используемых для получения хэш-функций для аутентификации.

ГЛАВА 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ МЕТОДОВ ФОРМИРОВАНИЯ ХЭШ-ФУНКЦИЙ

3.1 Программная реализация алгоритма ГОСТ Р 34.11-2012

ГОСТ Р – это стандарт хэширования в России. Он обрабатывает блоки сообщений размером 512 бит и вычисляет 512- или 256-битные хэш-значения. *l*-битное сообщение сначала дополняется до размера, кратного 512 битам. Единичный бит присоединяется к концу сообщения; после него следуют 512 - 1 - (l mod 512) нулевых бит. Пусть $M = M_t \,||\, M_{t-1} \,||\ldots||\, M_1$ – это сообщение, состоящее из t блоков после дополнения, представленное в big-endian форме. [14] Тогда, как показано на рисунке 3.1, вычисление $H(M)$ можно описать следующим образом:

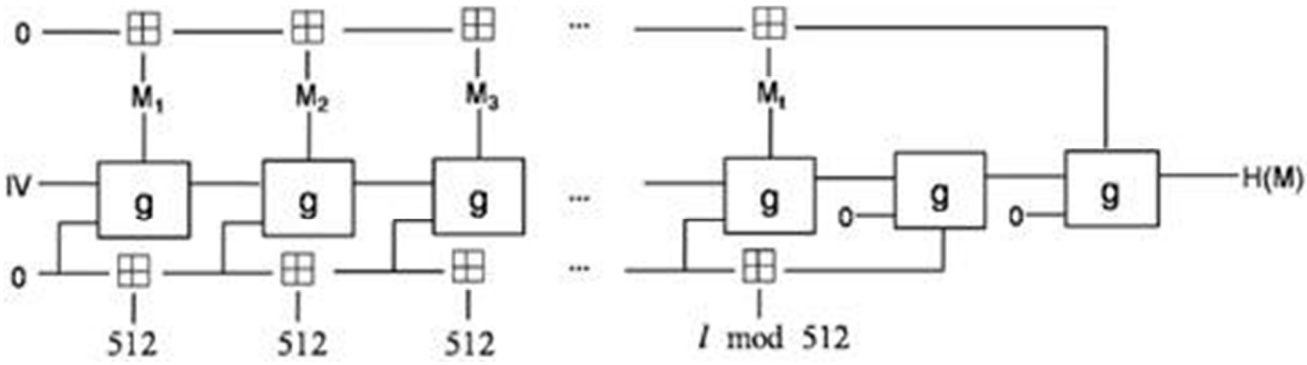


Рисунок 3.1 – Схема хэш-функции ГОСТ-Р 34.11-2012 Стрибог

$$h_0 = IV, N = 0, \Sigma = 0 \tag{3.1}$$

$$h_j = g_N\left(h_{j-1}, M_j\right), N = N \otimes 512, \Sigma = \Sigma \otimes M_j \text{ для } 0 < j < t \tag{3.2}$$

$$h_t = g_N\left(h_{t-1}, M_t\right), N = N \otimes \left(l \bmod 512\right), \Sigma = \Sigma \otimes M_t \tag{3.3}$$

$$h_{t+1} = g_0\left(h_t, N\right) \tag{3.4}$$

$$H\left(M\right) = g_0\left(h_{t+1}, \Sigma\right) \tag{3.5}$$

На рисунке 3.1 IV – предопределенное начальное значение, а $\otimes$ обозначает операцию сложения в кольце $Z_{2^{512}}$. $g_N(h, m)$ – это функция сжатия алгоритма ГОСТ Р 34.11-2012 [14], которая основана на 512-битном блочном шифре и вычисляется так:

$$g_N(h,m) = E\left(L \circ P \circ S\left(h \oplus N\right), m\right) \oplus h \oplus m \qquad (3.6)$$

Используемый в ГОСТ Р 34.11-2012 [14] блочный шифр E представляет собой вариант AES, который обновляет 64-байтное состояние (представленное как массив 8 x 8 – состояние можно представить также как 64 x 64 в поле GF(2)) и раундовый ключ, выполняя 12 раундов.

В основу хэш-функции положена итерационная конструкция Меркла-Дамгарда с использованием MD-усиления. Под MD-усилением понимается дополнение неполного блока при вычислении хэш-функции до полного путём добавления вектора (0 … 01) такой длины, чтобы получился полный блок. Из дополнительных элементов нужно отметить следующие:

- завершающее преобразование, которое заключается в том, что функция сжатия применяется к контрольной сумме всех блоков сообщения по модулю 2^{512};
- при вычислении хэш-кода на каждой итерации применяются разные функции сжатия. Можно сказать, что функция сжатия зависит от номера итерации.

Описанные выше решения позволяют противостоять многим известным атакам.

Кратко описание хэш-функции ГОСТ Р 34.11-2012 можно представить следующим образом. На вход хэш-функции подается сообщение произвольного размера. Далее сообщение разбивается на блоки по 512 бит, если размер сообщения не кратен 512, то оно дополняется необходимым количеством бит. Потом итерационно используется функции сжатия, в результате действия которой обновляется внутреннее состояние хэш-функции [14]. Также вычисляется контрольная сумма блоков и число обработанных бит. Когда обработаны все блоки исходного сообщения,

производятся еще два вычисления, которые завершают вычисление хэш-функции:

- обработка функцией сжатия блока с общей длиной сообщения.
- обработка функцией сжатия блока с контрольной суммой.

В рамках выпускной квалификационной работы была выполнена программная реализация алгоритма ГОСТ Р 34.11-2012 Стрибог на базе пакета прикладных программ MATLAB, которая представлена в листинге 3.1.

Листинг 3.1 – Программная реализация алгоритма хэширования ГОСТ Р 34.11-2012 на языке Matlab

```
% Гост 34.11-2012
function h = gost(M)
if ischar(M) % Если шестнадцатеричная строка перегоняем
   M = hex2uint8(M);   % в массив из 8 битных чисел
end
%% 1 Инициализация данных
% S-преобразование. Функция S является обычной функцией подстановки.
% Каждый байт из 512-битной входной последовательности заменяется соответствующим байтом из таблицы подстановок ?
p = [252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250, 218, 35, 197, 4, 77, 233, 119, 240,219, 147, 46, 153, 186, 23, 54, 241, 187, 20, 205, 95, 193, 249, 24, 101, 90, 226, 92, 239,33, 129, 28, 60, 66, 139, 1, 142, 79, 5, 132, 2, 174, 227, 106, 143, 160, 6, 11, 237, 152, 127,212, 211, 31, 235, 52, 44, 81, 234, 200, 72, 171, 242, 42, 104, 162, 253, 58, 206, 204, 181, 112, 14, 86, 8, 12, 118, 18, 191, 114, 19, 71, 156, 183, 93, 135, 21, 161, 150, 41, 16, 123, 154, 199, 243, 145, 120, 111, 157, 158, 178, 177, 50, 117, 25, 61, 255, 53, 138, 126, 109, 84, 198, 128, 195, 189, 13, 87, 223, 245, 36, 169, 62, 168, 67, 201, 215, 121, 214, 246, 124, 34, 185, 3, 224, 15, 236, 222, 122, 148, 176, 188, 220, 232, 40, 80, 78, 51, 10, 74, 167, 151, 96, 115, 30, 0, 98, 68, 26, 184, 56, 130, 100, 159, 38, 65, 173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165, 125, 105, 213, 149, 59, 7, 88, 179, 64, 134, 172, 29, 247, 48, 55, 107, 228, 136, 217, 231, 137, 225, 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144, 202, 216, 133, 97, 32, 113, 103, 164, 45, 43, 9, 91, 203, 155, 37, 208, 190, 229, 108, 82, 89, 166, 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57, 75, 99, 182];
p = uint8(p);
%  P-преобразование. Функция перестановки. Для каждой пары байт из входной
%  последовательности происходит замена одного байта другим.
Tau = [0, 8, 16, 24, 32, 40, 48, 56, 1, 9, 17, 25, 33, 41, 49, 57, 2, 10, 18, 26, 34, 42, 50, 58, 3, 11, 19, 27, 35, 43, 51, 59, 4, 12, 20, 28, 36, 44, 52, 60, 5, 13, 21, 29, 37, 45, 53, 61, 6, 14, 22, 30, 38, 46, 54, 62, 7, 15, 23, 31, 39, 47, 55, 63];
Tau = uint8(Tau+1);
% L-преобразование. Представляет собой умножение 64-битного входного вектора
```

% на бинарную матрицу A размерами 64x64. Матрицу A можно представить как массив 64-битных слов:

```
A = {'8e20faa72ba0b470' '47107ddd9b505a38' 'ad08b0e0c3282d1c' 'd8045870ef14980e',
    '6c022c38f90a4c07' '3601161cf205268d' '1b8e0b0e798c13c8' '83478b07b2468764',
```

Продолжение листинг 3.1

'a011d380818e8f40' '5086e740ce47c920' '2843fd2067adea10' '14aff010bdd87508',
'0ad97808d06cb404' '05e23c0468365a02' '8c711e02341b2d01' '46b60f011a83988e',
'90dab52a387ae76f' '486dd4151c3dfdb9' '24b86a840e90f0d2' '125c354207487869',
'092e94218d243cba' '8a174a9ec8121e5d' '4585254f64090fa0' 'accc9ca9328a8950',
'9d4df05d5f661451' 'c0a878a0a1330aa6' '60543c50de970553' '302a1e286fc58ca7',
'18150f14b9ec46dd' '0c84890ad27623e0' '0642ca05693b9f70' '0321658cba93c138',
'86275df09ce8aaa8' '439da0784e745554' 'afc0503c273aa42a' 'd960281e9d1d5215',
'e230140fc0802984' '71180a8960409a42' 'b60c05ca30204d21' '5b068c651810a89e',
'456c34887a3805b9' 'ac361a443d1c8cd2' '561b0d22900e4669' '2b838811480723ba',
'9bcf4486248d9f5d' 'c3e9224312c8c1a0' 'effa11af0964ee50' 'f97d86d98a327728',
'e4fa2054a80b329c' '727d102a548b194e' '39b008152acb8227' '9258048415eb419d',
'492c024284fbaec0' 'aa16012142f35760' '550b8e9e21f7a530' 'a48b474f9ef5dc18',
'70a6a56e2440598e' '3853dc371220a247' '1ca76e95091051ad' '0edd37c48a08a6d8',
'07e095624504536c' '8d70c431ac02a736' 'c83862965601dd1b' '641c314b2b8ee083'};

A = hexA2dec(A);

% Здесь C — это набор 512-битных значений. Значение C является постоянным C = {'b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc4b7ce09192676901a2422a08a460d31505767436cc744d23dd806559f2a64507',...

'b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc4b7ce09192676901a2422a08a460d31505767436cc744d23dd806559f2a64507',...

'f574dcac2bce2fc70a39fc286a3d843506f15e5f529c1f8bf2ea7514b1297b7bd3e20fe490359eb1c1c93a376062db09c2b6f443867adb31991e96f50aba0ab2',...

'ef1fdfb3e81566d2f948e1a05d71e4dd488e857e335c3c7d9d721cad685e353fa9d72c82ed03d675d8b71333935203be3453eaa193e837f1220cbebc84e3d12e',...

'4bea6bacad4747999a3f410c6ca923637f151c1f1686104a359e35d7800fffbdbfcd1747253af5a3dfff00b723271a167a56a27ea9ea63f5601758fd7c6cfe57',...

'ae4faeae1d3ad3d96fa4c33b7a3039c02d66c4f95142a46c187f9ab49af08ec6cffaa6b71c9ab7b40af21f66c2bec6b6bf71c57236904f35fa68407a46647d6e',...

'f4c70e16eeaac5ec51ac86febf240954399ec6c7e6bf87c9d3473e33197a93c90992abc52d822c3706476983284a05043517454ca23c4af38886564d3a14d493',...

'9b1f5b424d93c9a703e7aa020c6e41414eb7f8719c36de1e89b4443b4ddbc49af4892bcb929b069069d18d2bd1a5c42f36acc2355951a8d9a47f0dd4bf02e71e',...

'378f5a541631229b944c9ad8ec165fde3a7d3a1b258942243cd955b7e00d0984800a440bdbb2ceb17b2b8a9aa6079c540e38dc92cb1f2a607261445183235adb',...

'abbedea680056f52382ae548b2e4f3f38941e71cff8a78db1fffe18a1b3361039fe76702af69334b7a1e6c303b7652f43698fad1153bb6c374b4c7fb98459ced',...

'7bcd9ed0efc889fb3002c6cd635afe94d8fa6bbbebab07612001802114846679 8a1d71efea48b9caefbacd1d7d476e98dea2594ac06fd85d6bcaa4cd81f32d1b',...

Продолжение листинг 3.1

```
'378ee767f11631bad21380b00449b17acda43c32bcdf1d77f82012d430219f9b5d80ef9d1891cc86
e71da4aa88e12852faf417d5d9b21b9948bc924af11bd720'};
        C = hexC2dec(C);
        % 256
        % IV = uint8(zeros(1,64)); % 0^512
        % IV(64) = 111111;
        % 512
        IV = uint8(zeros(1,64)); % 0^512
        h=IV;
        N = uint8(zeros(1,64)); % 0^512
        Si = uint8(zeros(1,64)); % 0^512 сигма
        %% 2 Длина сообщения < 512 ?
        fl = 1; % Флаг продолжения
        while fl
          if length(M)<64 % в M восьмерки % Если условие выполняется перейти к пункту
3.
% 3 Произвести дополнение сообщения M до длины в 512 бит по следующему правилу:
            % m = 0^511-|M|||1||M, где |M| — длина сообщения M в битах.
            m = [zeros(1,63-length(M)) 1 M];
            % 4 Вычислить h = g(N, m, h)
            h = g(N, m, h);
            % 5 Вычислить N = (N + |M|) mod 2^512
            N = mod512add(N,length(M));
            % 6 Вычислить ? = (? + m) mod 2^512
            Si = mod512add(Si,m);
            % 7 Вычислить h = g(0, h, N)
            h = g(uint8(zeros(1,64)), h, N);
            % 8 Вычислить h = g(0, h, ?)
            h = g(uint8(zeros(1,64)), h, Si);
            % 9 Для хэш-функции с длиной выхода в 512 бит возвращаем h в качестве
результата. Для функции с длиной выхода 256 бит возвращаем MSB256(h).
            fl=0; % выход с цикла: флаг=0
          else % В противном случае выполнить последовательность вычислений:
            %    m — последние 512 бит сообщения M
            m = M((end-63):end);% 64 числа по 8 бит
            %    h = g(N, m, h)
            h = g(N, m, h);
            %    N = (N + 512) mod 2^512
            %    ? = (? + m) mod 2512
            %    Обрезать M, убрав последние 512 бит
            %    Перейти к шагу 2
          end
        end
        %% Функции
        % X-преобразование. На вход функции X подаются две последовательности
длиной
          % 512 бит каждая, выходом функции является XOR этих последовательностей.
          function z = X(z,varargin)
            for i=1:(nargin-1)
```

```
        z = bitxor(z,varargin{i});
    end
```

Продолжение листинга 3.1

```
end
% S-преобразование.
function y = S(x)
    y = p(x+1);
end
% P -преобразование.
function y = P(x)
   y(Tau) = x;
end
% Тогда оба вместе P(S(x))
function y = PS(x)
    y(Tau) = p(x+1);
end
% L преобразование
function  y = L(x)
    y = uint8(zeros(1,64));
    for i=1:8 % 512 x делим на 8 по 64 бита (8*8)
      block = dec2bin(x((1:8)+(i-1)*8),8)'; % в биты
      block = block(:); % в 1 столбец
      block = block == '1'; % из char в бинар
      bu = uint8(zeros(1,8)); % Нулевая строка
      for j = 1:64 % Идем по строкам
         if block(j) % там где бит 1 складываем строки по мод2
            bu = bitxor(bu,A(j,:));
         end
      end
      y((1:8)+(i-1)*8) = bu;
    end
end
% Объединение
function z = LPSX(x,y)
    z = L(PS(X(x,y)));
end
% Сложение по модулю 2^512 (вход массив по 8 бит)
function z = mod512add(x,y)
    cp = 0;                          % буфер переноса переполнения
    z = uint8(zeros(1,64));              % инициализация массива
    for i=1:64                        % 64*8 = 512
       cp = uint16(x) + uint16(y) + cp; % складываем с переносом
       ds = dec2bin(cp,16);             % в биты
       cp = bin2dec(ds(1:8));           % переполенение в cp
       z(i) = uint8(bin2dec(ds(9:16))); % остаток в z
    end
end
% Функция сжатия
function z = g(N,m,h)
    z = X(E(LPSX(h,N),m),h,m);
end
```

```
    % Подфункция для функции сжатия
    function m = E(K,m)
```

Окончание листинга 3.1

```
        for i = 1:12
            m = LPSX(K,m);
            K = LPSX(K,C(i,:));
        end
        m = X(K,m);
    end
    % Матрицу A в uint
    function  Z = hexA2dec(A)
        Z = uint8(zeros(64,8));
        for i=1:64 % строки
            m8 = hex2uint8(A{i}); % Разобьем на 8
            Z(i,:) = m8;
        end
    end
    % Матрицу C в uint
    function  Z = hexC2dec(C)
        Z = uint8(zeros(12,64));
        for i=1:12 % строки
              Z(i,:) = hex2uint8(C{i});
        end
    end
end
```

Результат вычисления хэш-функции с помощью разработанной программной реализации для сообщения представлены на рисунке 3.2.

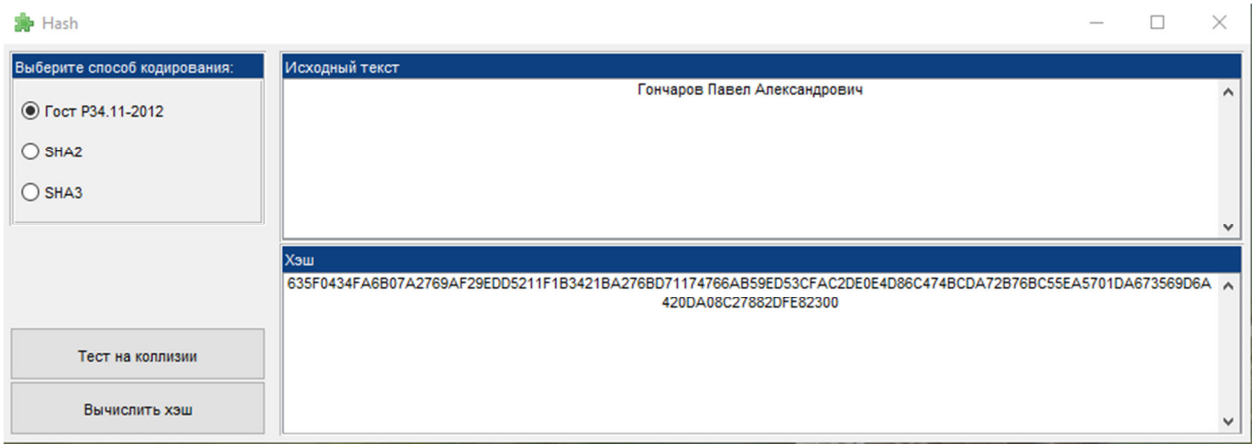


Рисунок 3.2 – Результат вычисления хэша ГОСТ для сообщения

Криптоанализ старого стандарта выявил некоторые его слабые стороны с теоретической точки зрения. Так в одной из работ, посвящённых криптоанализу ГОСТ Р 34.11-94, было выявлено, что сложность алгоритма

построения прообраза оценивается в 2192 вычислений функций сжатия, коллизии 2105, что меньше «универсальных» оценок, которые для ГОСТ Р 34.11-94 равны 2256 и 2128. Хотя по состоянию на 2013 год нет большого числа работ, посвящённых криптостойкости новой хэш-функции, исходя из конструкции новой хэш-функции, можно сделать некоторые выводы о её криптостойкости и предположить, что её криптостойкость будет выше, чем у ГОСТ Р 34.11-94:

В разделе «Описание» из схемы ГОСТ Р 34.11-2012 видно, что все блоки сообщения суммируются по модулю 2^{512} и уже результат суммирования всех блоков подается на вход завершающего этапа (stage3). Благодаря тому, что здесь суммирование — это не побитовое сложение, получается защита от следующих атак:

- построение мультиколлизий;
- удлинение прообраза;
- дифференциальный криптоанализ;

В функции сжатия используется конструкция Миагучи-Пренели, это обеспечивает защиту от атаки, основанную на фиксированных точках, так как для конструкции Миагучи-Пренели не найдено способов (легких) для поиска фиксированных точек;

На каждой итерации при вычислении хэш-кода используются различные константы. Это затрудняет атаки на основе связанных и разностных связанных ключей, атаки скольжения и отражения.

Для изучения криптостойкости новой хэш-функции компания «ИнфоТеКС» в ноябре 2013 года объявила о старте конкурса[6]; он завершился в мае 2015 года. Победителем стала работа «The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function», в которой авторы представили атаку нахождения второго прообраза для хэш-функции «Стрибог-512», требующую 2266 вызовов функции сжатия для сообщений длиннее 2259 блоков.

На конференции Crypto-2015 Алекс Бирюков, Лео Перрин и Алексей Удовенко представили доклад, в котором говорится о том, что значения S-блока шифра «Кузнечик» и хэш-функции Стрибог не являются (псевдо)случайными числами, а сгенерированы на основе скрытого алгоритма, который докладчикам удалось восстановить методами обратного проектирования.

В ходе работы было проведено исследование хэш-функции на лавинный эффект. Это исследование заключается в изменение одного бита в хэшируемом сообщении (таблица 3.1) и подсчёте доли изменившихся бит в выходном хэше (таблица 3.2). Чем больше доля изменений, тем лучше криптостойкость хэш функции.

Таблица 3.1 – Хэши сообщений полученные по ГОСТ Р 34.11-2012

ID	**Хэш сообщения полученный по ГОСТ Р 34.11-2012**	**Хэшируемое сообщение**
hash 1	'A7A537053B41EAB09AE0AFDDC3AFA84DEBDD8F0470D3598862ABED9131D190B162321E930A1D1B6659609221D91D8C75B58D326D6CEF7510FDF2CF105EC6A93C'	Гончаров Павел Александрович1
hash 2	'58429C1599DAAE2808EB53F9B680FD2B2F425BD70BC9F597322FF2B8C029CA5A8A7F211DE39D2DC68C83D7011E0C9E323602D19FA5BE3E9F3F85CFED2737862B'	Гончаров Павел Александрович2
hash 3	'E355383804A34FCC26077369E1DF3FAC4B0C86A5E6B4BA865F754FB474B537822ECC2C2A484F90994F2588FBDCCAB1F6622F41A9139338FCB643726D76D5D338'	Гончаров Павел Александрович3
hash 4	'4C03EF74C2339248C2075D77E61ADC5933135E95ED468D7888BBAD620A35D1A0E8A03C6CEA15C9017545149A7327C06F58BC55040BE68FF6C36C05C3F276FBD7'	Гончаров Павел Александрович4
hash 5	'C7CE075D455AD94706B21AB04DA14601D7D683ED512931DEC3B5CFF831DDCD5E3DE07056A9796513F2B76B17AC1F8A35F036B2C4EEE69AE7FB4E9B37465AA141'	Гончаров Павел Александрович5
hash 6	'DD942620E08A1D04CD2D34221649E543EA968BC70C00280AF0312529F6A2EF797BB9E6766F7A66BADBD4AD2F2B3B61C0A36D77C9787A680C17BC94114F595042'	Гончаров Павел Александрович6
hash 7	'BAC61FA50220AAEA5427CE8C7A34E71533FF931C9CF87FDDB17510F7F00962DE9A03585BDA2C2F6D36F5A4CE48E32A45C8B7F42B6EC374F48B7AFEB24720C778'	Гончаров Павел Александрович7
hash 8	'CD9270C78A7C471930D519317519CABD52A9C829BC6B59F2DA57B7DA88797AB108D096B3C9A4EB8BAD418BD46A067122F4878B2619472C8805064389113AF523'	Гончаров Павел Александрович8
hash 9	'3117128E44122FA12B7087661FA7D06C8DA2C51B91B5549D58B966A65B837F7C49B9B930AAF6FE4F9D5B090D3915BFB4ABA5FA3C51ADD994B7789A80AD69C1AB'	Гончаров Павел Александрович9

Таблица 3.2 – Результат исследования лавинного эффекта хэша смежных хэшей сообщений, полученных по ГОСТ Р 34.11-2012

hash 1 vs hash 2	hash 2 vs hash 3	hash 3 vs hash 4	hash 4 vs hash 5	hash 5 vs hash 6	hash 6 vs hash 7	hash 7 vs hash 8	hash 8 vs 9 gost
0,506	0,518	0,502	0,498	0,465	0,512	0,512	0,496
				Среднее значение			0,503

3.2 Программная реализация алгоритма SHA-2

Хэш-функции SHA-2 разработаны Агентством национальной безопасности США и опубликованы Национальным институтом стандартов и технологий в федеральном стандарте обработки информации FIPS PUB 180-2 в августе 2002 года. В этот стандарт также вошла хэш-функция SHA-1, разработанная в 1995 году. В феврале 2004 года в FIPS PUB 180-2 была добавлена SHA-224. В октябре 2008 года вышла новая редакция стандарта — FIPS PUB 180-3. В марте 2012 года вышла последняя на данный момент редакция FIPS PUB 180-4, в которой были добавлены функции SHA-512/256 и SHA-512/224, основанные на SHA-512 (поскольку на 64-битных архитектурах SHA-512 работает быстрее, чем SHA-256). Схема формирования хэш функции SHA-2 представлена на рисунке 3.3.

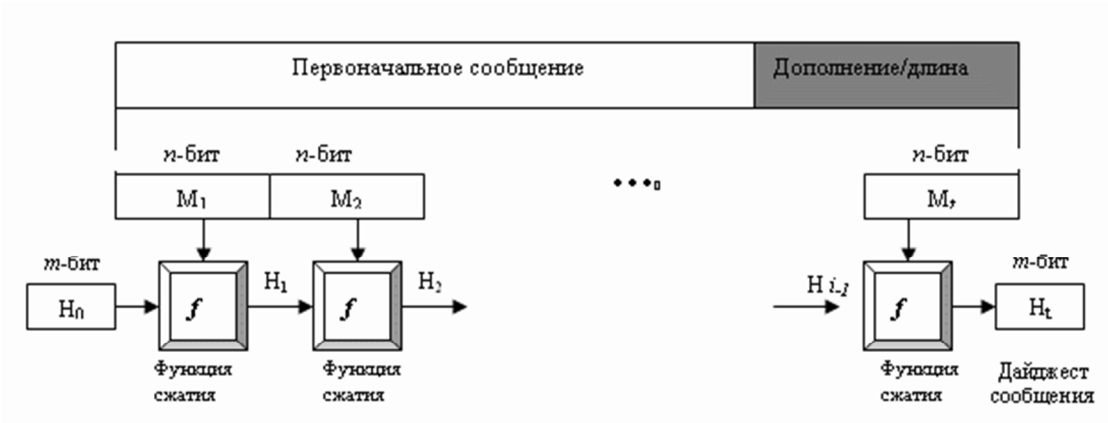


Рисунок 3.2 – Схема хэш-функции SHA-2

В основе криптографических хэш-функций SHA-2 лежит метод Меркла – Дамгарда. Исходная информация разбивается на части одинакового размера, каждая из которых подвергается обработке односторонней функцией сжатия. После такой операции длина входящего сообщения уменьшается.

Сформированный хэш-функцией код имеет фиксированную длину, независимо от размера входной информации. Размер полученных образов варьируется в диапазоне от 30 до 512 бит.

Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64 или 80 итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока складываются, сумма является значением хэш-функции. Тем не менее, инициализация внутреннего состояния производится результатом обработки предыдущего блока. Поэтому независимо обрабатывать блоки и складывать результаты нельзя

Алгоритм использует следующие битовые операции:

|| — конкатенация,

+ — сложение,

and — побитовое «И»,

xor — исключающее «ИЛИ»,

shr (shift right) — логический сдвиг вправо,

rotr (rotate right) — циклический сдвиг вправо.

В данной работе реализован алгоритм SHA-256, имеющий следующие параметры:

Показатель размера блока (байт) – 64.

Предельно допустимая длина сообщения (байт) – 33.

Характеристика размера дайджеста сообщения (байт) – 32.

Стандартный размер слова (байт) – 4.

Параметр длины внутреннего положения (байт) – 32.

Число итераций в одном цикле – всего 64.

Достигаемая протоколом скорость (MiB/s) – примерно 140.

В рамках выпускной квалификационной работы была выполнена программная реализация алгоритма SHA-2 (SHA-256) на базе пакета прикладных программ MATLAB, которая представлена в листинге 3.2.

Листинг 3.2 - Программная реализация алгоритма хэширования SHA-2 (SHA-256)

```
% Функция SHA2-256
function s = sha2_256(message)
% Все переменные беззнаковые, имеют размер 32 бита и при вычислениях суммируются по модулю 232
%  message — исходное двоичное сообщение
%  m — преобразованное сообщение
%  Инициализация переменных
%  (первые 32 бита дробных частей квадратных корней первых восьми простых чисел [от 2 до 19]):
hx={'6A09E667','BB67AE85','3C6EF372','A54FF53A','510E527F','9B05688C','1F83D9AB',
'5BE0CD19'};
% Таблица констант
% (первые 32 бита дробных частей кубических корней первых 64 простых чисел [от 2 до 311]):
kx={'428A2F98', '71374491', 'B5C0FBCF', 'E9B5DBA5', '3956C25B', '59F111F1',
'923F82A4', 'AB1C5ED5', 'D807AA98', '12835B01', '243185BE', '550C7DC3', '72BE5D74',
'80DEB1FE', '9BDC06A7', 'C19BF174', 'E49B69C1', 'EFBE4786', '0FC19DC6', '240CA1CC',
'2DE92C6F', '4A7484AA', '5CB0A9DC', '76F988DA', '983E5152', 'A831C66D', 'B00327C8',
'BF597FC7', 'C6E00BF3', 'D5A79147', '06CA6351', '14292967', '27B70A85', '2E1B2138',
'4D2C6DFC', '53380D13', '650A7354', '766A0ABB', '81C2C92E', '92722C85', 'A2BFE8A1',
'A81A664B', 'C24B8B70', 'C76C51A3', 'D192E819', 'D6990624', 'F40E3585', '106AA070',
'19A4C116', '1E376C08', '2748774C', '34B0BCB5', '391C0CB3', '4ED8AA4A', '5B9CCA4F',
'682E6FF3','748F82EE', '78A5636F', '84C87814', '8CC70208', '90BEFFFA', 'A4506CEB',
'BEF9A3F7', 'C67178F2'};
hs = hex2uint32(hx);
k = hex2uint32(kx);
clear 'kx','hx';
% Предварительная кодировка текста в целые числа по 32 бита:
m = text2uint32(message);
% + Длина(message) — длина исходного сообщения в битах в виде 64-битного числа
ds = dec2bin(uint64(length(char(message))*8),64); % 64 бит длина
m(end+1) = bin2dec(ds(1:32));  % делим пополам
m(end+1) = bin2dec(ds(33:64));
clear 'ds';
% Далее сообщение обрабатывается последовательными порциями по 512 бит: на 16 слов длиной 32 бита
```

```
n  = length(m)/16;
```
Продолжение листинга 3.2

```
for i=1:n
  one_window(m((1:16)+(i-1)*16));
end
% Получить итоговое значение хэша:
s = '';
for i=1:8
   s = [s dec2bin(hs(i),32)];
end
s = binaryVectorToHex(str2logic(s));
  % один участок данных
   function one_window(m)
      w = uint32(zeros(1,64));
      w(1:16) = m;
      % обработка куска
      % Сгенерировать дополнительные 48 слов:
      for j=17:64
         s0 = bitxor( bitxor( bitrot(w(j-15),-7) , bitrot(w(j-15),-18) ) , bitshift(w(j-15),-3) );
         s1 = bitxor( bitxor( bitrot(w(j-2),-17) , bitrot(w(j-2),-19) ) , bitshift(w(j-2),-10) );
         w(j) = mod32add( w(j-16), s0 ,w(j-7) , s1);
      end
      clear 's0','s1';
      % Инициализация вспомогательных переменных:
      a = hs(1);
      b = hs(2);
      c = hs(3);
      d = hs(4);
      e = hs(5);
      f = hs(6);
      g = hs(7);
      h = hs(8);
      % Основной цикл:
      for j = 1:64
          [a,b,c,d,e,f,g,h] = one_round(a,b,c,d,e,f,g,h,k(j),w(j));
      end
      % Добавляем к получившимуся
      hs(1) =  mod32add(hs(1),a);
      hs(2) =  mod32add(hs(2),b);
      hs(3) =  mod32add(hs(3),c);
      hs(4) =  mod32add(hs(4),d);
      hs(5) =  mod32add(hs(5),e);
      hs(6) =  mod32add(hs(6),f);
      hs(7) =  mod32add(hs(7),g);
      hs(8) =  mod32add(hs(8),h);
   end
   % Один раунд
   function [a,b,c,d,e,f,g,h] = one_round(a,b,c,d,e,f,g,h,k,w)
         s0 = bitxor( bitxor( bitrot(a,-2), bitrot(a,-13) ), bitrot(a,-22) );
         Ma = bitxor( bitxor( bitand(a,b), bitand(a,c) ), bitand(b,c));
         t2 = mod32add(s0,Ma);
```

```
            s1 = bitxor( bitxor( bitrot(e,-6), bitrot(e,-11) ), bitrot(e,-25) );
```

Окончание листинга 3.2

```
            Ch = bitxor( bitand(e,f), bitand(bitnot(e),g) );
            t1 = mod32add(h,s1,Ch,k,w);
            h = g;
            g = f;
            f = e;
            e = mod32add(d,t1);
            d = c;
            c = b;
            b = a;
            a = mod32add(t1,t2);
        end
        function m = text2uint32(message)
            % Предварительная обработка текста:
            m = uint8(char(message));
            % + [единичный бит] и 7 нулей (128)
            m(end+1) =  128;
            % + [k нулевых бит], где k — наименьшее неотрицательное число, такое что (L
+ 1 + K) mod 512 = 448
            mo = mod(length(m),64);
            if mo < 56 % тогда просто дополним до 56
                m = [m zeros(1,56-mo)];
            else % тогда до 64 и +56
                m = [m zeros(1,(64-mo)+56)];
            end
            % групируем 8 в 32 по 4
            m = group8to32(m);
        end
    end
```

Результат вычисления хэш-функции SHA2 с помощью разработанной программной реализации для сообщения “Гончаров Павел Александрович” представлены на рисунке 3.4.

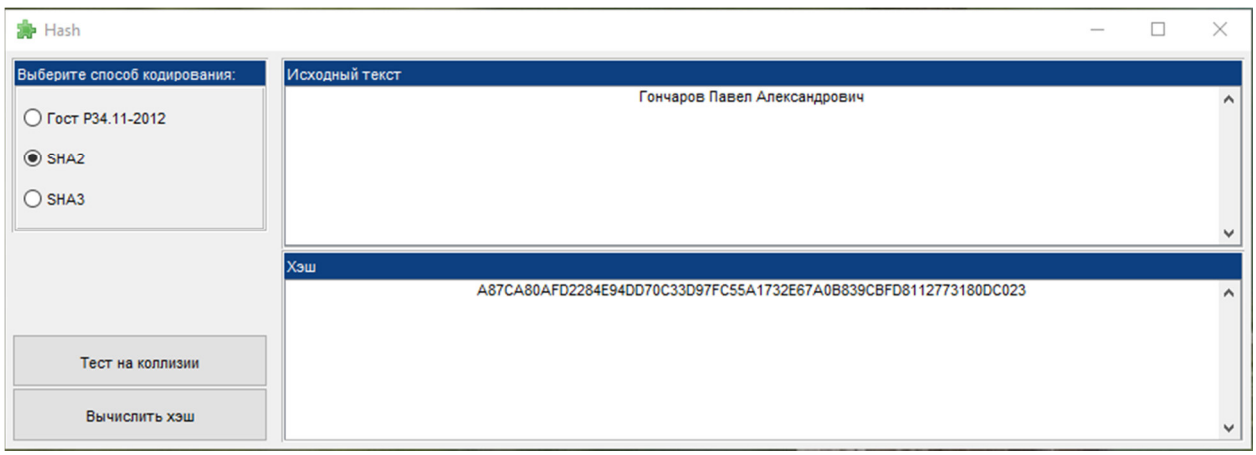


Рисунок 3.4 – Результат вычисления хэша ГОСТ для сообщения

В ходе работы было проведено исследование хэш-функции на лавинный эффект. Это исследование заключается в изменение одного бита в хэшируемом сообщении (таблица 3.3) и подсчёте доли изменившихся бит в выходном хэше (таблица 3.4). Чем больше доля изменений, тем лучше криптостойкость хэш функции.

Таблица 3.3 – Хэши сообщений полученные по SHA-2 (SHA-256)

ID	**Хэш сообщения полученный по SHA-2 (SHA-256)**	**Хэшируемое сообщение**
hash 1	'46767ABED473A08A20FD6FEC1F952114D7E97ECE1E734750252AD1B285271F86'	Гончаров Павел Александрович1
hash 2	'ADC0AF4241DDC339C04442E48F1112BF2B5EF205B159E0C8F8B86573F1513EC9'	Гончаров Павел Александрович2
hash 3	'8A9FE5A44563D98AD884DE865B3EF4D9074CFCFC7575076AF9AA4C6B3B5956AC'	Гончаров Павел Александрович3
hash 4	'7ECAE0A068D4801F774329F402015122F30EB055814FCB72CA66B7C751ACFBAF'	Гончаров Павел Александрович4
hash 5	'F1162ED1DBBD518B4A709AD301387E2293935778C1EA3A00CE084AAC1960B6BA'	Гончаров Павел Александрович5
hash 6	'E59D96BB3281B6A001F3DBC89F4EF04199FD99BFA3A434D940EEAA142492D77E'	Гончаров Павел Александрович6
hash 7	'47320AE07B82245E25F202AADCCDA616058319FF693C14E3F1133A0A3CC2D275'	Гончаров Павел Александрович7
hash 8	'B224E2C71817E464AEC5389BF9959B6B3524B206D45A6EC14448548674E287B5'	Гончаров Павел Александрович8
hash 9	'39DAC98A7460D5A87C30723F27ED551C38BBE68FB854B9E3056F41AB492286EE'	Гончаров Павел Александрович9

Таблица 3.4 – Результат исследования лавинного эффекта хэша смежных хэшей сообщений, полученных по SHA-2 (SHA-256)

hash 1 vs hash 2	hash 2 vs hash 3	hash 3 vs hash 4	hash 4 vs hash 5	hash 5 vs hash 6	hash 6 vs hash 7	hash 7 vs hash 8	hash 8 vs hash 9
0,521	0,484	0,486	0,517	0,509	0,486	0,507	0,511
				Среднее значение			0,501

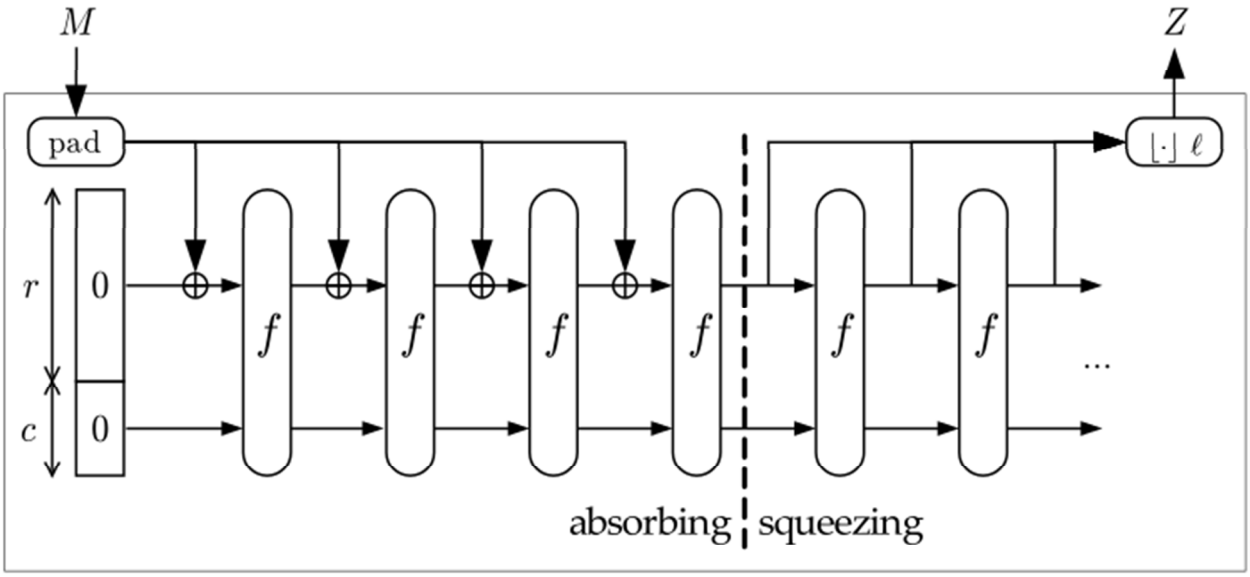
3.3 Программная реализация алгоритма Keccak

Традиционный дизайн хэш-функций основан на использовании функции сжатия. Эта функция отображает значение $m \rightarrow n$ $(m > n)$ псевдослучайным образом. При этом значение n должно быть до 512 бит, а m порядка 2n. Повторяя эту функцию в течении нескольких раундов с различными константами, достигают нужного значения стойкости. Дополняя и сцепляя между собой блоки от разных фрагментов исходного текста, получают возможность вычислить хэш от сообщения произвольной длины. При этом возникает существенная проблема: сконструировать функцию сжатия трудно. Многораундовые повторения сгладят её дефектность, но заведомое наличие быстрой возможности найти частичную коллизию в исходной функции сжатия ставит под вопрос стойкость всей конструкции. Авторы алгоритма Keccak (Guido Bertoni, Joan Daemen, Michaël Peeters и Gilles Van Assche) утверждают, что сконструировать надёжную функцию сжатия вида $m \rightarrow n$ $(m > n)$ как однораундовый блок криптопримитива, крайне сложно (или невозможно вообще).

Авторы алгоритма Skein (и Whirlpool, который не участвует в конкурсе, но вошёл в некоторые промежуточные стандарты) считают, что в качестве функции сжатия можно использовать блочный шифр. Действительно, в отличие от псевдослучайных функций (Pseudo Random Function — PRF), псевдослучайные перестановки (Pseudo Random Permutation — PRP), создавать проще. А блочный шифр является такой перестановкой, зависимой от ключа. Достаточно сконструировать шифр с размером блока и размером ключа 512 бит и можно будет получить функцию сжатия $m \rightarrow n$ $(m > n)$, подавая на вход такого шифра эти значения. Проблема однако в том, что хотя блочные шифры и считаются самыми доверяемыми в плане стойкости симметричными криптопримитивами, они часто имеют облегчённое или слабое ключевое расписание (функцию разворачивания подключей раунда из основного ключа). Это приводит к атакам со связанными ключами, которые

хотя и не представляют практической угрозы в большинстве протоколов, но ставят под удар функцию сжатия на основе блочного шифра. Более того — идеальное ключевое расписание для идеального блочного шифра само по себе должно обладать свойствами идеальной псевдослучайной функции. То есть для создания идеальной хэш-функции нужно использовать … идеальную хэш-функцию. Получается замкнутый круг, который лишь отчасти можно победить некоторыми конструкторскими уловками и некоторыми натяжками в доказательствах. Так, можно использовать фиксированный ключ, но изменяющийся твик-вектор или использовать лишь часть выхода шифра.

Авторы хэш-функции Keccak (произносится как "Кечак") приняли ряд радикальных решений. Они решили не использовать функцию сжатия в виде отдельного строительного блока, а в качестве стойкого криптопреобразования решили сконструировать бесключевую PRP. Всё это упаковано в очень простую конструкцию Sponge ("Губка"), что проиллюстрировано на рисунке 3.5. Авторами этой конструкции являются непосредственные создатели алгоритма Keccak, которые впервые представили её на симпозиуме ECRYPT Hash function – 2007 в качестве альтернативы традиционному дизайну Дамгарда-Меркла.



sponge

Рисунок 3.5 – Фазы поглощения и впитывания базового алгоритма Keccak (SHA-3)

В фазе абсорбции ("поглощения", "впитывания" губкой) сначала задаётся исходное состояние из нулевого вектора размером до 1600 бит. Затем производится операция xor фрагмента исходного сообщения p_0 с фрагментом исходного состояния размером r, оставшаяся часть состояния ёмкостью с остаётся незатронутой. Результат обрабатывается функцией f — многораудовой бесключевой PRP и так повторяется до исчерпания блоков исходного сообщения. Затем наступает фаза сжатия или отжатия (как если бы отжимали губку), при которой можно вывести хэш произвольной длины.

Атаки на нахождение коллизий (двух произвольных сообщений, дающих одинаковый хэш) и вторых прообразов (сообщения, дающего такой же хэш, что и имеющееся) имеют важное практическое и теоретическое значение, но наряду с неинвертируемостью не обеспечивают полной оценки стойкости хэш-функции. Существует целый ряд экзотических атак — на удлинение сообщения, атаки на частичные коллизии с подобранным префиксом и др. Чтобы доказать стойкость конструкции Sponge ко всем возможным атакам, авторы приняли ещё одно радикальное решение: считать единственным и достаточным общим критерием стойкости неразличимость хэш-функции от случайного оракула.

Случайный оракул (Random Oracle) — это идеализированная функция, описывающая работу машины с практически бесконечным объёмом памяти, которая на любой запрос может выдать идеально случайное число и запомнить пару "запрос-ответ". Если такой же запрос будет когда-либо повторён, то ответ будет не генерироваться заново, а взят из запомненного списка. Если перестановка f, лежащая в основе Sponge идеальна, то и хэш-функция доказуемо неразличима от RO, значит никакие из возможных атак действовать не будут. Конечно, есть оговорка на наличие универсального тривиального различителя для всех возможных хэш-функций: построение случайного оракула на алгоритме с конечным числом состояний невозможно.

Например коллизия внутреннего состояния приведёт к неслучайному выходу, что было бы невозможно в RO, у которого никаких внутренних состояний нет. Однако, авторы доказывают, что такая конструкция обеспечивает стойкость для N запросов к функции f или f^{-1}, равную $N^2 \cdot 2^{-(c+1)}$, что при обычной длине выхода укладывается в $2^{c/2}$ и невозможно без наличия различителя в используемой PRP.

Именно эти теоретические результаты, основанные на одном из самых строгих доказательств неразличимости от случайного оракула в конкурсе SHA-3, дают удивительные практические возможности для простого использования хэш-функции Keccak в качестве практически универсального криптопримитива.

Простое добавление секретного ключа на вход хэш-функции Keccak/Sponge превращает её в код аутентификации сообщений – рисунок 3.6. Это было невозможно в обычных хэш-функциях SHA-1 или SHA-2 и требовало громоздкой конструкции HMAC, что проиллюстрировано на рисунке

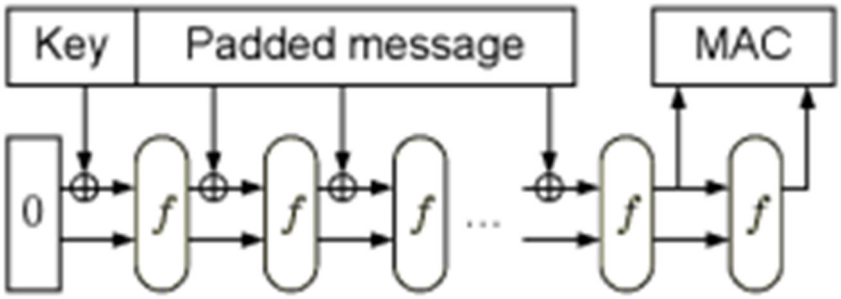


Рисунок 3.6 – Использование хэш-функции Keccak для аутентификации

Ещё одной интересной особенностью конструкции Sponge является лёгкость персонализации данных. Предположим, один и тот же пароль используется как для генерации симметричного ключа, так и для ключа MAC — для предотвращения изменения сообщения. Нежелательно, чтобы они были одинаковыми. Например, в хэш-функции Skein для этого предусмотрен

ограниченный набор битов для переключения режимов использования. А в конструкции Sponge хэш-функции Keccak это не требуется. Любая персонализация данных может быть добавлена на вход функции, например, по рекомендации авторов, в XML-формате и кодировке UTF, что делает число возможных вариантов персонализации и режимов использования практически неограниченным.

Кроме конструкции Sponge, интересна также и сама функция перестановки хэш-функции Keccak. Текущее внутреннее состояние представлено в ней в виде набора битов, сгруппированных в виде виртуального объекта в трёхмерном пространстве (трёхмерного массива). Сам объект можно разбить на плоскости или точнее слои вдоль трёх осей координат, а элементы каждого слоя — на фрагменты в виде столбцов или векторов.

При этом для обработки внутренних состояний не используются S-блоки или операции, параметры которых менялись бы в зависимости от данных. Для создания нелинейных преобразований используется набор простейших логических операций (XOR, AND, NOT), которые существуют в любом процессоре и минимальный набор констант. А трёхмерное представление текущего внутреннего состояния помогает применить эти операции оптимальным образом.

В рамках выпускной квалификационной работы была выполнена программная реализация алгоритма SHA-3 (Keccak) на базе пакета прикладных программ MATLAB, которая представлена в листинге 3.3.

Листинг 3.3 - Программная реализация алгоритма хэширования Keccak (SHA-3)

```
function Z = SHA3(message)
%  Инициализация
r = 576; % 512 версия sha3
c = 1024;
d = 512/4;
S = uint64(zeros(5,5));     % нулевое начальное состояние 5x5 64 бит = 1600 бит
```

```
P = text2blocks(message,r); % текст впревращаем в числа 64 бит и дополняем если надо блок до кратному r
k = (length(P)*64)/r; % кол-во блоков
```

Продолжение листинга 3.3

```
% Постоянные
b=1600;     % стандартное рекомедуемое число для всех версий sha3
rc =   ['0000000000000001';
        '0000000000008082';
        '800000000000808A';
        '8000000080008000';
        '000000000000808B';
        '0000000080000001';
        '8000000080008081';
        '8000000000008009';
        '000000000000008A';
        '0000000000000088';
        '0000000080008009';
        '000000008000000A';
        '000000008000808B';
        '800000000000008B';
        '8000000000008089';
        '8000000000008003';
        '8000000000008002';
        '8000000000000080';
        '000000000000800A';
        '800000008000000A';
        '8000000080008081';
        '8000000000008080';
        '0000000080000001';
        '8000000080008008'];
for  m = 1:size(rc,1) % В 64 бит числа
   RC(m) = hex2uint64(rc(m,:));
end
clear rc;
% Таблица сдвигов
sd = [ 0, 36, 3, 41, 18;
       1, 44, 10, 45, 2;
      62, 6, 43, 15, 61;
      28, 55, 25, 21, 56;
      27, 20, 39, 8, 14];
% Подсчет раундов
w = b / 25; % n=12+2*l, где 2^l=(b/25). тут =64
l = log2(w); % =6
n = 12 + 2 * l; % Так для b=1600, Количество раундов равно 24.
dl = r/w; % Длина блока в 64 числах
% Фаза накопления
for m=1:k % перебор блоков
  S(1:dl) =  X(S(1:dl), P((1:dl)+(m-1)*dl)); % складываем начало S с блоком из P
  S=f(S); % Функция f
end
% Фаза сжатия
```

```
Z = '';
fl=1;
while fl
```

Продолжение листинга 3.3

```
    for ii=1:5
        for jj=1:5
            if fl
                Z = [Z uint642hex(S(ii,jj))];
                if length(Z) >= d % остановка
                    fl=0;
                end
            end
        end
    end
    S=f(S); % Функция f
end
    function A = f(A)
        % основная функция хэширования
        for i=1:n % 1:24
            A = round(A, RC(i)); % запуск раунда
        end
    end
    function A = round(A,RC)
        % Раунд
        C = uint64(zeros(1,5));
        D = C;
        B = uint64(zeros(5,5));
        % шаг ?
        for i=1:5
          C(i) = X(A(i,1),A(i,2),A(i,3),A(i,4),A(i,5));
        End
        for i=1:5
          D(i) = X( C( mod(i-2,5)+1 ),  bitrot64( C( mod(i,5)+1 ), -1) );
        end
        for i=1:5
            for j=1:5
                A(i, j) = X(A(i, j),D(i));
            end
        end
        % шаги ? и ?
        for i=1:5
            for j=1:5
                B(j,mod((2*(i-1)+3*(j-1)),5)+1) = bitrot64(A(i, j), -sd(i, j));
            end
        end
        % шаг ?
        for i=1:5
            for j=1:5
                 A(i, j) = X( B(i, j) , bitand(bitnot64(B(mod(i,5)+1, j)) , B(mod((i+1),5)+1,j)) );
            end
        end
```

```
            % шаг ?
            A(1,1) = X(A(1,1),RC);
        end
```

Окончание листинга 3.3

```
        function z = X(z,varargin)
            % XOR с любым кол-вом операндов
            for i=1:(nargin-1)
                z = bitxor(z,varargin{i});
            end
        end
        function me = text2blocks(message,r)
            if isempty(message)
                me = uint8(zeros(1,r/8));
            else
                % Предварительная обработка текста:
                me = uint8(char(message));
                if mod(length(me)*8,r)~=0 % если блок не полон добавляем до полного
                    u = (r - mod(length(me)*8,r))/8; % сколько нужно дополнить
                    if u==1 % если необходимо дополнить всего один байт, то достаточно добавить лишь '81.
                        me(end+1) = 129;
                    else % В виде формулы их можно изобразить следующим образом: M=M||'01||'00||..||'00||'80.
                        me(end+1) = 128;   % к сообщению дописывается единичный байт
                        if u-2>0         % нули если >2 требуется добавить
                            me(end+(1:(u-2))) = uint8(zeros(1,u-2));
                        end
                        me(end+1) = 1; %128 и этот ансамбль завершает байт со значением '80.
                    end
                end
            end
            % Теперь объединим по 8 байт в числа 64 битные
            me = group8to64(me);
        end
    end
```

Результат вычисления хэш-функции Keccak (SHA-3) с помощью разработанной программной реализации для сообщения “Гончаров Павел Александрович” представлены на рисунке 3.7.

В ходе работы было проведено исследование хэш-функции Keccak на лавинный эффект. Это исследование заключается в изменение одного бита в хэшируемом сообщении (таблица 3.5) и подсчёте доли изменившихся бит в выходном хэше (таблица 3.6). Чем больше доля изменений, тем лучше криптостойкость хэш функции Keccak.

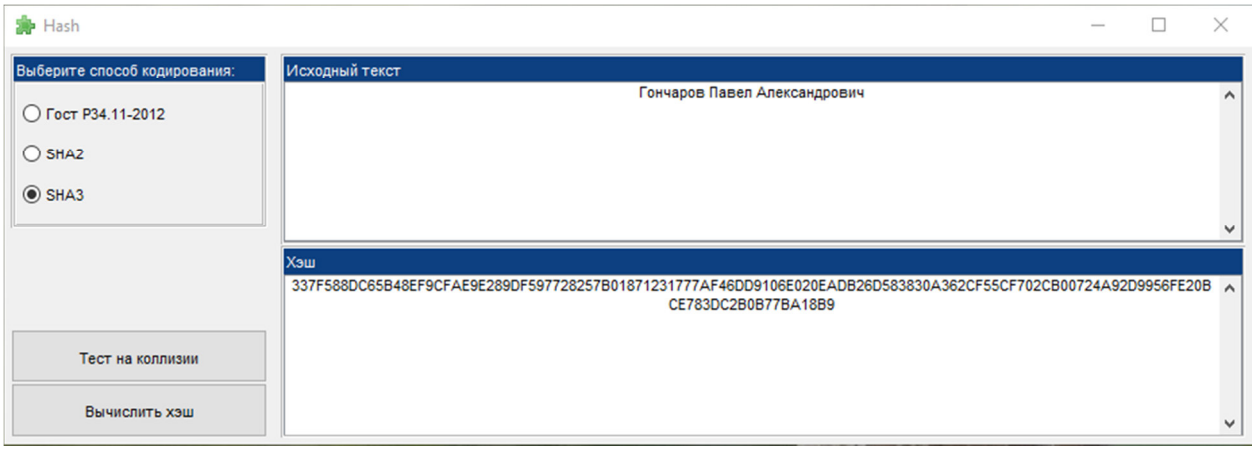


Рисунок 3.7 – Результат вычисления хэша Keccak для сообщения

Таблица 3.5 – Хэши сообщений полученные по sha3

ID	**Хэш сообщения полученный по алгоритму Keccak**	**Хэшируемое сообщение**
hash 1	'209E60AAA8DC3B572B55F25077F219A891A02AECD346B49F56AA4E2A7A00EC2BC1099316B157C48B44A085BA4ECE2D4631DEFE5BD062E619840C24C4A83789C8'	Гончаров Павел Александрович1
hash 2	'FA5CAE95AE77F2D3E6E12FE21A1A4DDF07315F4A3455CA30C21D144D9942D95C7CC9607B169CB2B09AB7A0B92FC7C9DEBC80586A07F21DDAB42A3426D8347EDB'	Гончаров Павел Александрович2
hash 3	'8A8C2DE4D8F20B6B557F982E77CDAB8EC058589F1E5D80BC95258C05129DA46C2325CCFEFE51A430AADC0109A9B96F31EC278B03453DC90DAC28FC3F9ED757DA'	Гончаров Павел Александрович3
hash 4	'5B933F31348AC7FE26B31232839589CC04E45C70196AF0EA4AC3FC2DF3FD14CABBCD2D69CDC961C284885060C2FC06ADB27AB59E483711E2DD428B3F173C54F5'	Гончаров Павел Александрович4
hash 5	'AF83E3AD0F13BB6837D18536168A37B2FED7475EDC14CAA570F1F37E76307F4A8BB9E3B8082ED8685CD20821753201B0AEA850B552F15C95F4D3E6C7A8F53ACC'	Гончаров Павел Александрович5
hash 6	'DB8CEB936FD3E430879CDAC98CE5972914A876A1280478FF9F4629DD8A271D90976BC4FA9CC3B2A46C49E364501771E001F51E191836F680B6C9FC5258231999'	Гончаров Павел Александрович6
hash 7	'8CE0EDE8866151D1C7C37FB902BC466B9E1F9F69AA59414CE02293D211D6CBF1C98D745A9D794FE66BFE846252F4C5E618BFD5DAD6903509C6F3882B7327B81A'	Гончаров Павел Александрович7
hash 8	'6A072C4BFE6F392EBE2AE67000DFD475B0328DC3F3FABBD0E1F191C3EE2019F69458390D1791C1A4314AA5660BE89A946EC00E6C88C218BFE17A6ACC3B74660F'	Гончаров Павел Александрович8
hash 9	'0CBA4EDEBD7964AA769CB8BC3002454CD2C620256468BCAE22C52DCFABC634719C2C07A1F703750AC3E1C9E3E83E23071CB6D0F47ACDD2E617BBA868B4CFDA28'	Гончаров Павел Александрович9

Таблица 3.6 – Результат исследования лавинного эффекта хэша смежных хэшей сообщений, полученных по алгоритму Keccak

hash 1 vs hash 2	hash 2 vs hash 3	hash 3 vs hash 4	hash 4 vs hash 5	hash 5 vs hash 6	hash 6 vs hash 7	hash 7 vs hash 8	hash 8 vs hash 9
0,527	0,433	0,539	0,492	0,5	0,425	0,492	0,503
				Среднее значение			0,489

3.4 Коллизии и свойства хэш-функций

Коллизии – это одинаковые образы, возникающие из-за обработки одинаковых входных блоков данных. Для борьбы с коллизиями разработаны специальные методы. Например, при работе с хэш-таблицами применяются методы открытой адресации либо прямого связывания. Если же стоит задача защитить от фальсификации пароли и электронные подписи, применяется метод «криптографической соли», когда в создании хэша участвует свободно изменяемая последовательность битов. Он используется в том числе и в криптовалютах.

Важно, чтобы содержимое образа менялось вслед за оригиналом. Это является первым и главным требованием к хэш-функциям любых семейств. Если изменения не будут отслеживаться, то образ перестанет соответствовать оригиналу, и работа с данными окажется невозможной.

Второе требование – уникальность каждого образа. Конечно, существует вероятность того, что образы двух файлов или баз данных совпадут, но она чрезвычайно мала. Чем выше надежность алгоритма, тем эта вероятность меньше. К счастью, хэш-функции семейства SHA-2 достаточно защищены от коллизий, поэтому сбои в работе систем практически исключены.

Третье требование –однонаправленность функции. Алгоритм работает только в одну сторону, то есть сжимает, перемешивает и рассеивает информацию, но восстановить ее на основании полученного результата он не

способен без наличия ключей. Это служит дополнительной защитой при шифровании данных.

Четвертое требование – невозможность подделки: необходимо, чтобы подбор сообщения с правильным значением образа обладал высокой сложностью. Если значение одного из сообщений все же станет достоверно известным, то подбор остальных сообщений должен быть максимально сложен.

Для того, чтобы хэш-функция H считалась криптографически стойкой, она должна удовлетворять трём основным требованиям, на которых основано большинство применений хэш-функций в криптографии:

Необратимость: для заданного значения хэш-функции m должно быть практически невозможно найти блок данных *X*, для которого *H(X)=m*.

Стойкость к коллизиям первого рода: для заданного сообщения *M* должно быть практически невозможно подобрать другое сообщение *N*, для которого *H(N)=H(M)*.

Стойкость к коллизиям второго рода: должно быть практически невозможно подобрать пару сообщений (*M,M'*), имеющих одинаковый хэш.

В таблице 3.7 представлены значения критериев для исследованных хэш-функций: ГОСТ Р 34.11-2012, SHA-2, Keccak.

Таблица 3.7 – Критерии криптостойкости хэш-функций

Хэш-функция	Критерии криптостойкости			
	Изменчивость	Уникальность	Однонаправленность	Стойкость
ГОСТ Р 34.11-2012	Очень высокая	Очень высокая	-	Высокая
SHA-2	Высокая	Высокая	-	Средняя
Keccak	Высокая	Высокая	-	Высокая

3.5 Разработка и исследование программы для сравнения криптостойкости хэш-функций

В данной выпускной квалификационной работе был реализован механизм проверки криптостойкости хэш-функций с помощью поиска коллизий. Метод поиска коллизий реализован на базе атаки «дней рождения».

С помощью этой атаки отыскание коллизии для хэш-функции разрядности n битов потребует в среднем около $2^{n/2}$ операций. Поэтому n-битная хэш-функция считается криптостойкой, если вычислительная сложность нахождения коллизий для неё близка к $2^{n/2}$.

Атака «дней рождения» основана на известном парадоксе дней рождений, который заключается в том, что в группе, состоящей из 23 или более человек, вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышает 50 %. Для 60 и более человек вероятность такого совпадения превышает 99 %, хотя 100 % она достигает, согласно принципу Дирихле, только тогда, когда в группе не менее 367 человек.

Для заданной хэш-функции h целью атаки является поиск коллизии второго рода. Для этого вычисляются значения h для случайно выбранных блоков входных данных до тех пор, пока не будут найдены два блока, имеющие один и тот же хэш.

Пусть h — хэш-функция. Атака «дней рождения» успешна если найдется пара $x_1 \neq x_2$ такая, что $x_1 \neq x_2$, но $h(x_1) = h(x_2)$. Таким образом, если функция $h(x)$ дает любой из N разных выходов с равной вероятностью и N достаточно велико, то мы ожидаем получить пару различных аргументов $x_1 \neq x_2$, с $h(x_1) = h(x_2)$ после оценки функции около $1,25 \cdot \sqrt{N}$ различных аргументов в среднем. Оценку числа операций хэширования для поиска коллизии идеальной криптографической хэш-функции с размером выхода n бит часто записывают как $2^{n/2}$, а не 2^n.

Пусть *P(n)* – вероятность того, что в группе из *n* человек $(n < N = 365)$ хотя бы двое имеют одинаковый день рождения.

$$P(n) = 1 - 1 \cdot (1 - 1/N) \cdot (1 - 2/N) \ldots (1 - (n-1)/N) = 1 - \prod_{i=0}^{n-1} (1 - i/N)$$

Следуя из ряда Тейлора функции $e^x = 1 + x/1! + x/2! + x/3! + \ldots$, $e^x \geq 1 + x$.

$$P(n) \approx 1 - \prod_{i=0}^{n-1} \left(e^{i/N}\right) = 1 - e^{-n(n-1)/2N}$$

Найдем число n, такое чтобы $P(n) \geq 1/2$.

$1/2 \geq \exp\left(-n^2/2N\right)$

Отсюда следует, что $n \geq 1{,}177\sqrt{N} \approx 23$.

Например, если используется 64-битный хэш, существует примерно $1{,}8 \times 10^{19}$ различных выходов. Если все они одинаково вероятны (лучший случай), то для создания коллизии с использованием грубой силы потребуется около 5 миллиардов попыток (5.38×10^9). Примеры для исследуемых хэш-функций показаны в таблице 3.8.

Таблица 3.8– Оценка количество переборов для взлома хэша длинной 256 и 512 бит

Биты	**Возможные выходы (N)**	**Желаемая вероятность случайной коллизии (P)**									
		10^{-18}	10^{-15}	10^{-12}	10^{-9}	10^{-6}	0,001	0,01	0,25	0,50	0,75
256	2^{256} (~1.2×10^{77})	4.8×10^{29}	1.5×10^{31}	4.8×10^{32}	1.5×10^{34}	4.8×10^{35}	1.5×10^{37}	4.8×10^{37}	2.6×10^{38}	4.0×10^{38}	5.7×10^{38}
512	2^{512} (~1.3×10^{154})	1.6×10^{68}	5.2×10^{69}	1.6×10^{71}	5.2×10^{72}	1.6×10^{74}	5.2×10^{75}	1.6×10^{76}	8.8×10^{76}	1.4×10^{77}	1.9×10^{77}

В разработанной программе (рисунок 3.8) желаемая вероятность появлений случайной коллизии ограничена значением 0,001 в виду высоких требований к объемам оперативной памяти (свыше 32 Гбайт). В таблице 3.9 представлены результаты расчёта для реализованных алгоритмов.

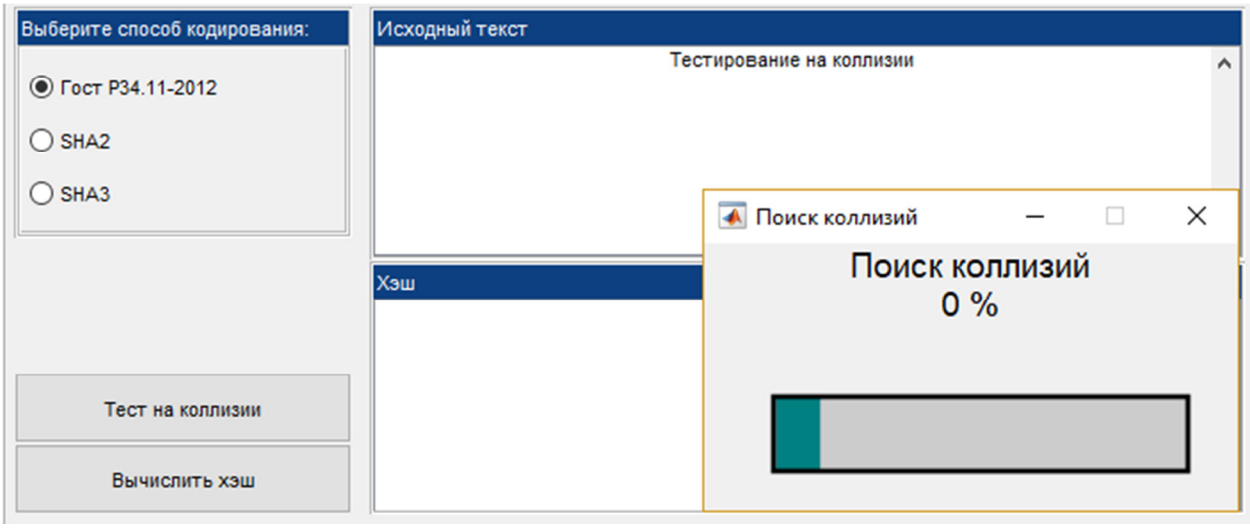


Рисунок 3.8 – Интерфейс программы для подсчёта коллизий (старт)

После перебора, программа выдает значение времени (рисунок 3.9), потраченное на тест, а также достигнута ли вероятность появления случайной коллизии - 0.001. В случае обнаружения коллизии до заданной вероятности программа выводит информационное сообщение.

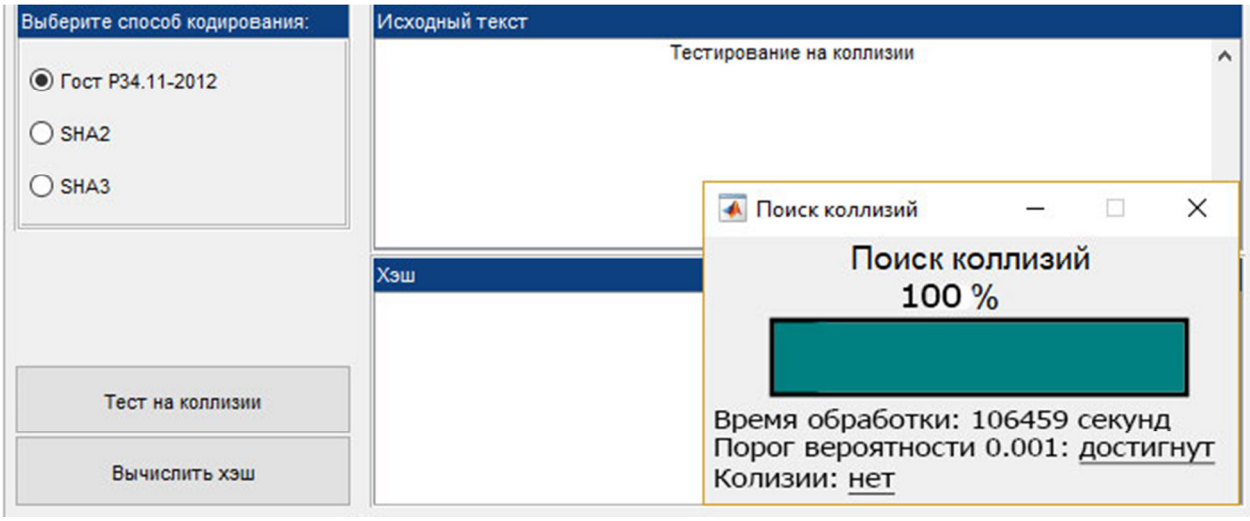


Рисунок 3.9 – Интерфейс программы для подсчёта коллизий (финиш)

Таблица 3.9 – Время расчёта коллизий для исследованных алгоритмов

Алгоритм и его вариация		**Время поиска коллизий, секунд**
SHA-2	*SHA-256*	130156
	SHA-512	198256
Keccak	*SHA3-256*	110783
	SHA3-512	172322
ГОСТ Р 34.11-2012	*Стрибог 256*	106459
	Стрибог 512	165861

В таблице 3.10 представлены обобщенные результаты исследования алгоритмов хэширования ГОСТ Р 34.11-2012, SHA-2, Keccak.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы магистра было выполнено исследование криптостойкости методов и алгоритмов аутентификации данных построенных на базе следующих хэш-функций: ГОСТ Р 34.11-2012, SHA-2, Keccak.

Для достижения данной цели были сформулированы и решены следующие задачи:

1) Проведен анализ существующих методов и подходов к аутентификации данных (простая аутентификация, строгая аутентификация, в том числе одно-, дву- и трехнаправленая);

2) Выполнена классификация существующих подходов к аутентификации данных;

3) Выполнена программную реализацию хэш-функций: SHA-2, Keccak, ГОСТ Р 34.11-2012, используемых для аутентификации данных;

4) Проведен сравнительные исследования реализованных хэш-функций для задач аутентификации данных;

5) Разработаны рекомендации по использованию хэш-функций в задачах аутентификации данных.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Аграновский А.В. Основы компьютерной стеганографии: Учебное пособие / А. В. Аграновский, П.Н. Девянин, Р. А. Хади, А.В. Черемушкин. М.: Радио и Связь, 2003. - 154 с.

2. Анин Б.Ю. Защита компьютерной информации / Б.Ю. Анин. СПб.: БХВ-Петербург, 2000. - 384 с.

3. Астахов А. Анализ защищенности корпоративных систем / А. Астахов // Открытые системы, 2002. - №07-08. - С. 44-49.

4. Баричев С.С. Основы современной криптографии / С.С. Баричев, В.В. Гончаров, Р.Е. Серов. М.: Мир, 1997. - 176 с.

5. Барсуков В.С. Компьютерная стеганография вчера, сегодня, завтра / В.С. Барсуков, А.П. Романцов //Специальная техника. 1998. - №4-5.

6. Варновский Н.П. О теоретико-сложностном подходе к определению стойкости стеганографических систем / Н.П. Варновский // Сборник трудов IV международной конференции «Дискретные модели в теории управляющих систем» М.: «МАКС Пресс». - С. 15-16.

7. Варфоломеев А.А. Блочные криптосистемы. Основные свойства и методы анализа стойкости. / А.А. Варфоломеев, А.Е. Жуков, А.Б. Мельников, Д.Д. Устюжанин. М.: МИФИ, 1998. 200 с.

8. Вихорев С.В. Классификация угроз информационной безопасности /С.В. Вихорев. Сетевые атаки и системы информационной безопасности. - 2001. - №9. [Электронный ресурс] - Режим доступа: http://www.cnews.m/comments/security/

9. Генне О. В. Основные положения стеганографии /О.В. Генне. Защита информации. Конфидент, 2000. №3. [Электронный ресурс] - Режим доступа: http://www.confident.ru/magazine/

10. Городецкий В.И. Стеганография на основе цифровых изображений /В.И. Городецкий, В.В. Самойлов. // Информационные технологии и вычислительные системы, 2001. №2/3, с. 51-64.

11. ГОСТ Р 50922-96. Защита информации. Основные термины и определения.

12. ГОСТ Р ИСО/МЭК 15408-3-2002. Информационная технология. Методы и средства обеспечения безопасности. Критерии оценки безопасности информационных технологий.

13. ГОСТ Р 34.10-94. Информационная технология. Криптографическая защита информации. Электронная цифровая подпись. М.: Госстандарт РФ.

14. ГОСТ Р 34.11-2012. Информационная технология. Криптографическая защита информации. Электронная цифровая подпись. М.: Госстандарт РФ.

15. Грушко А.А. Теоретические основы защиты информации / А.А. Грушко, Е.Е. Тимонина. М.: Яхтсмен, 1996. 31 с.

16. Дерявин П.Н. Теоретические основы компьютерной безопасности: Учебное пособие для вузов / П.Н. Дерявин и др. М.: Радио и связь, 2000. – 192с.

17. Домашев А. В. Программирование алгоритмов защиты информации / А.В. Домашев, В.О. Попов, Д.И. Правиков, И.В. Прокофьев, А.Ю. Щербаков. М.: Нолидж, 2000. 288 с.

18. Елманова Н.З. Введение в Borland C++ Builder /Н.З. Елманова, С.П. Кошель. М.: Диалог-МИФИ, 1998. 675 с.

19. Зегжда Д.П. Основы безопасности информационных систем / Д.П. Зегжда, А.М. Ивашко. М.: Горячая линия - Телеком, 2000. 452 с.

20. Зегжда Д.П. Создание систем обработки закрытой информации на основе защищенной ОС и распространенных приложений / Д.П. Зегжда // Проблемы информационной безопасности. Компьютерные системы. 1999.-№1.-С.106-109.

21. Зегжда П.Д. Безопасные информационные системы на основе защищенной ОС / П.Д. Зегжда // Проблемы информационной безопасности. Компьютерные системы. 1999. -№1. С.96-99.

22. Зима В.М. Безопасность глобальных сетевых технологий /В,М. Зима, А.А. Молдовян, Н.А. Молдовян. Спб.: БХВ-Петербург, 2000. - 320 с.

23. Кустов В. Н. Методы встраивания скрытых сообщений / В.Н. Кустов, А. А. Федчук.//Защита информации. Конфидент, 2000. № 3.- С. 34-36.

24. Мельников Ю.Н. Электронная цифровая подпись. Возможности защиты /Ю.Н. Мельников // Конфидент. 1995. - №4(6). С. 35-47.

25. Оков И. Н. Криптографические системы защиты информации /И.Н. Оков. СПб.: Типография ВУС, 2001. - 236 с.

26. Оков И.Н. О требуемой пропускной способности каналов передачи аутентифицированных сообщений в безусловно стойких системах / И.Н. Оков // Проблемы информационной безопасности. Компьютерные системы. 2000. -№ 3(7). С.78-64.

27. Оков И.Н. Электронные водяные знаки как средство аутентификации передаваемых сообщений / И.Н. Оков, Р.М. Ковалев. // Конфидент, -2001. №3. [Электронный ресурс] - Режим доступа: http://www.confident.ru/maRazine/new/64.html

28. Преступления в сфере компьютерной информации: квалификация и доказывание: Учебное пособие. / Под редакцией Ю.В.Гаврилина. Серия: Высшая школа. М: ЮИ МВД РФ, 2003. - 240 с.

29. Проблемы безопасности программного обеспечения. / Под ред. Зегжда П.Д. СПб.: СПбГТУ, 1995. - 192 с.

30. Пярин В.А. Безопасность электронного бизнеса / В.А. Пярин, А.С. Кузьмин, С.Н. Смирнов.; Под ред. действительного члена РАЕН д.т.н., проф. В.А. Минаева; М.: Гелиос АРВ, 2002. - 432 с.

31. Симонов С.В. Технологии аудита информационной безопасности /С.В. Семенов // Защита информации. Конфидент, 2002. -№2. - С.38-43.

32. Соколов А.В. Защита от компьютерного терроризма: Справочное пособие /А.В. Соколов, О.М. Степанюк. БВХ-Петербург: Арлит, 2002. -496 с.

33. Трубачев А.П. Общие критерии оценки безопасности информационных технологий / А.П. Трубачев, И. В. Егоркин, М.Т. Кобзарь, А.А. Сидак. // Конфидент. Защита информации. 2002. - №2. - С.54-60.

34. Шелупанов А.А. Основы информационной безопасности: Учебное пособие / А.А. Шелупанов, В.П. Лось, Р.В. Мещеряков, Е.Б. Белов. М.: Горячая линия телеком, 2006. - 544 с.

35. Шелупанов А.А. Специальные вопросы информационной безопасности / А. А. Шелупанов, Р.В. Мещеряков // Монография. Томск: Институт оптики атмосферы СО РАН, 2003. - 244 с.

36. Шокарев А.В. Использование цифровых водяных знаков для аутентификации передаваемых сообщений /А.В. Шокарев //Вестник СибГАУ «Системная интеграция и безопасность». Красноярск, 2006. -Спец. выпуск. С.123-127.

37. Шокарев А.В. Теоретико-информационный и Теоретико-сложностный подходя для оценки стойкости стеганографических систем /

38. Шелупанов, А.В. Шокарев //Вестник СибГАУ «Системная интеграция и безопасность». Красноярск, 2006. - Спец. выпуск. С. 121-123.

39. Шокарев А.В. Цифровые водяные знаки. Защита авторских прав / А.В. Шокарев // Научное творчество молодежи: Материалы X Всероссийской научно-практической конференции. 4.1. Томск: Изд-во ТГУ, 2006. С -98-100.

40. Иванов А.И. Биометрическая идентификация личности по динамике подсознательных движений / А.И. Иванов // Электронный ресурс. -Режим доступа: http://beda.stup.ac.ru/biometry.

41. Thorpe J. Graphical dictionaries and the memory space of graphical passwords /J. Thorpe, P. van Oorschot // 13th Usenix Security Symposium, 2004.- P. 135-150.

42. Adams A. Privacy in multimedia communications: protecting users not just data /A. Adams, M.A. Sasse. //Proceedings of IMH HCI'01, 2001. P. 49 -64.

43. Adams A. Privacy issues in ubiquitous multimedia environments: Wake sleeping dogs, or let them lie? / A. Adams, M.A. Sasse. //Proceedings of INTERACT'99, Edinburgh, 1999. P. 214-221.

44. Коновалов Д. Н. Технология защиты информации на основе идентификации голоса / Д.Н. Коновалов, А. Г. Бояров // Электронный ресурс. Режим доступа: http://www.fact.ru/archive/07/voice.shtml

45. Anderson R. Stretching the limits of steganography / R. Anderson // Proceedings 1st Intern. Workshop on Inform Hiding, vol. 1174. 1996, P.39-48.

46. Anderson R. On the limits of steganography / R. Anderson, F .Petitcolas. // IEEE J. on Selected Areas in Communications, vol. 16, № 4. 1998. P. 474481.

47. Barak B On the (im)possibility of obfuscating programs. / B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, Ke Yang. // Advances in Cryptology — Crypto'Ol. LNCS, vol. 2139. 2001. P.1-18.

48. Besnard D. Computer security impaired by legitimate users / D. Besnard, B. Arief. // Computers and Security, 23(3), 2004. P. 253-264.

49. Blonder G. Graphical passwords, United States Patent 5559961,1996.

50. Brostoff, S. Are Passfaces more usable than passwords: A field trial investigation / S. Brostoff, , M.A. Sasse, // People and Computers XIV -Usability or Else, Proceedings of HCI, 2000. P. 405-424.

51. Brown, A.S. Generating and remembering passwords / A.S. Brown, E. Bracken, S. Zoccoli, K. Douglas // Applied Cognitive Psychology, 2001. -P.641-651.

52. Bas P. Image watermarking: an evolution to content based approaches / P. Bas, J.-M. Chassery, B. Macq // Pattern Recognition vol. 35. 2002. P. 545561.

53. Cachin C. An information-theoretic model for steganography /C. Cachin // Proceedings 2nd Intern. Workshop on Inform. Hiding, vol.1525. 1998. -P.306-318.

54. Chandramouli R. Analysis of lsb Based Image Steganography Techniques / R. Chandramouli, N. Memon // Proceedings IEEE International Conf. On Image Processing, vol.3. 2001. P. 1019-1022.

55. Charbon E. On Intellectual Property Protection. In proceedings of Custom Integrated Circuits / E. Charbon, I.H. Torunoglu //Conference, 2000. P. 517523.

56. Christian C. An information-theoretic model for steganography / C. Christian Электронный ресурс. Режим доступа: http://algolist.manual.ru/defence/hide/

57. Cox I.J. Digital Watermarking. /I.J. Cox, M.L. Miller, J.A. Bloom //Morgan Kaufmann Publisher. Academic Press. 2002. 542 p.

58. Craver S. On public-key steganography in the presence of an active warden / S. Craver // Proceedings 2nd Intern. Workshop on Inform. Hiding, vol. 1525. 1996. P.355-368.