Оглавление

Глава 1

Основы криптографии

1.1 История появления необходимости криптографии

Человечеству всегда было необходимо передавать информацию так, чтобы её могли получить лишь те, кому она назначена, и не попала в руки к тем, от кого её берегли. С изобретением письменности это стало намного актуальнее, и именно тогда появилось понятие тайнописи. Люди пробовали самые различные методы утаивания информации- от использования особых чернил, которые либо через время исчезали с бумаги, либо которые были видны только при контакте с особыми химическими реагентами, до скрытия основной информации в большом тексте. Криптография появилась именно как наука, изучающая способы шифрования данных, а именно обезопасить данные не сокрытием факта их передачи, а сделав содержимое недоступным посторонним, даже если они завладеют ими(данными). Для этого информацию нужно исказить таким образом, чтобы содержимое было понятно только корреспондентам, и никому более. В 1950-х, с появлением первых **ЭВМ** ситуация значительно изменилась- криптография обрела полноценный практический смысл, уйдя от понятия **“тайнопись”**. На сегодняшний день эта дисциплина включает в себя методы защиты информации и информационных взаимодействий различного характера, основываясь на преобразовании данных секретными алгоритмами, а также на алгоритмах, включающих в себя секретные параметры. Информационным взаимодействием называется процесс обмена, передачи и обработки информации между двумя и более субъектами. Основных, фундаментальных методов преобразования в современной криптографии не столь много, но комбинируя их можно получить полноценную прикладную систему. Информация является самым важным ресурсом современной жизни. Компьютерные сети и Интернет упростили доступ к данным о конкретных людях, так и о финансовых и государственных организациях. Очевидно, что этим могут воспользоваться злоумышленники. Как говорилось ранее, криптография является совокупностью способов преобразования (т.е. шифрования) информации, сделав её бесполезной в случае попадания данных в руки неправомерных пользователей. Сегодня, криптография и её методы применяются для идентификации пользователей и их аутентификации, защиты каналов связи и передачи данных, защита электронных документов от изменения и копирования.

1.2 Основные понятия криптографии и их реализации

Основной механизм защиты данных — это **шифрование**. **Шифрование** — это преобразование информации в нечитабельную форму, осуществляемое с помощью ключей шифрования. Методов шифрования имеется множество, среди которых есть элементарные, так и принципиально невозможные для вскрытия. Ниже представлена обобщенная схема криптосистемы, обеспечивающей шифрование данных.

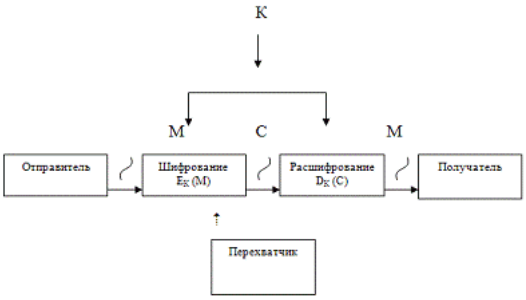


Рис. 1.1: Обобщенная схема криптосистемы

Отправитель составляет понятный текст первоначального сообщения M, которое будет передано по незащищённому каналу Получателю. Чтобы Перехватчик не смог воспользоваться содержимым сообщения даже после перехвата, сообщение M шифруется при помощи обратимого преобразования и получается криптограмма $= E(M)$. Получатель, приняв криптограмму, расшифровывает её при помощи обратного преобразования $D = {E_k}^{-1}$ и получает открытое содержимое сообщения
$M : D_k(C) = {E_k}^{-1}(E_k(M)) = M$.
Все преобразования выбираются из так называемого семейства криптографических преобразований, называемых **криптоалгоритмами.** Параметр, по которому выбирается конкретное криптографическое преобразование, называется криптографическим ключом . Ключ принадлежит определенному пользователю или группе пользователей и быть для них уникальным. Информация, зашифрованная с использованием определённого ключа может быть расшифрована лишь владельцем или владельцами ключа.
Говоря в целом, шифрование бывает симметричным или асимметричным в отношении преобразования расшифрования.
Это свойство разделяет практически все криптосистемы на две части:

- Симметричные криптосистемы (один ключ)
- Асимметричны криптосистемы (два ключа)

В случае с симметричной криптосистемой используются идентичные ключи в блоке шифрования и в блоке расшифрования. То есть, владея ключом шифрования, можно беспрепятственно расшифровать сообщение. Из-за этого симметричные криптосистемы так же называют криптосистемами с секретным ключом- потому что ключ шифрования должен быть доступен лишь тем, кому дозволена информация в сообщении. Следовательно, задача сохранения конфиденциальности сообщений и/или электронных документов сводится к сохранению конфиденциальности этого самого ключа шифрования. Одно из основных неудобств симметричного шифрования заключается в том, что перед обменом зашифрованными данными надо обменяться секретными ключами между адресатами. Нельзя допускать, чтобы этот обмен происходил в общедоступных каналах связях, ключ должен быть передан по защищённому специальному каналу распространения ключей. Симметричное шифрование хорошо подходит для шифрования с целью защитить архивные файлы, а также для шифрования логических и физических дисков. Принципиальное отличие ассиметричной системы шифрования от симметричной заключается в том, что для расшифрования используется ключ, отличающийся от ключа для шифрования. В таком случае, первый ключ является открытым, а второй- секретным.

Криптоанализ- наука о раскрытии первоначального текста зашифрованного сообщения без ключа. Успех такого анализа может раскрыть не только исходный текст, но и ключ. Также, можно выяснить слабые места криптосистемы и позже исправить их. Основополагающим правилом этой науки сформулировал голландец **А. Керхоффом**, заключив, что стойкость шифра должна определяться только секретностью ключа. Иначе говоря, это правило указывает на то, что весь алгоритм, кроме значения секретного ключа, известен **крипто-аналитику.** Это обусловлено тем, что любая криптосистема, реализующая семейство преобразований, рассматривается как открытая.

Глава 2

Хэширование

2.1 Разновидности хэширования

Как было упомянуто выше, криптография занимается не только конфиденциальностью данных, но и её целостностью. Контроль этой самой целостности осуществляется, в основном, с помощью расчета **"контрольной суммы"** данных. Для большинства приложений достаточно простой контрольной суммы, однако чтобы сделать криптографически стойкие контрольные суммы, необходимо использование хэш-функций.

Базовыми свойствами криптографически стойкой хэш-функции являются свойства рассеивания и стойкости к коллизиям. Коллизией хэш-функции H называется случай, в котором существуют два различных текста T1 и T2, но H(T1)=H(T2). Значение хэш-функции имеет фиксированную длину, а длина текста не ограничена. Следовательно, коллизии существуют. Стойкость к коллизиям означает то, что для текста T1 невозможно найти текст T2, вызывающий коллизию. Свойство рассеивания означает, что минимальное изменение текста будет вызывать максимальные изменения в значении хэш-функции. Основными алгоритмами хэш-функций, применяемым на сегодняшний день, являются **MD2, MD4, MD5, SHA** (а также его разновидность, SHA1). Типичные длины хэш-функций разнятся от 16 до 32 байт. Здесь стоит отметить важный аспект современного цифрового мира, объединяющий асимметричное шифрование и хэш-функции- электронная подпись. Электронная подпись документа- хэш сумма, зашифрованная секретным ключом. Если зашифровать сообщение секретным ключом, парным к открытому, то любой желающий сможет расшифровать содержимое. Проверка электронной подписи заключается в вычислению хэш-суммы документа, в вычислений хэш-суммы подписи и сравнения данных величин. Если значения равны, то подпись под документом считается верной.

2.2 SHA-1

Secure Hash Algorithm 1 — алгоритм криптографического хэширования. Входное сообщение генерируется в 160-битное хеш-значение. SHA-1 реализует хэш-функцию, основывающуюся на функции сжатия. Входами функции сжатия являются блок текста длиной 512 бит выход предыдущего блока сообщения. Выход является значением хэш-блоков до этого момента, то есть хэш блока M_i равен $H = f(M_i H_{i-1})$. Хэш значением всего сообщения является последний блох выхода. Хэш значением всего сообщения является последний блох выхода.

2.2.1 Коллизии SHA-1

В официальном сообщении авторы говорят, что эта находка стала результатом двухлетнего исследования, которая началась вскоре после публикации в 2013 году работы криптографа Марка Стивенса из Центра математики и информатики в Амстердаме о теоретическом подходе к созданию коллизии SHA-1. Он же в дальнейшем продолжил поиск практических методов взлома вместе с коллегами из Google.

Компания Google давно выразила своё недоверие SHA-1, особенно в качестве использования этой функции для подписи сертификатов TLS. Ещё в 2014 году, вскоре после публикации работы Стивенса, группа разработчиков Chrome объявила о постепенном отказе от использования SHA-1. Теперь они надеются, что практическая атака на SHA-1 увеличит понимание у сообщества информационной безопасности, так что многие ускорят отказ от SHA-1.

Чтобы представить число хешей, которые обсчитала Google во время брутфорса, можно упомянуть, что примерно такое же количество хешей SHA-256 обсчитывается в сети Bitcoin каждые три секунды, так

что в атаке нет ничего фантастического. Вполне можно предположить, что в криптографических отделах некоторых организаций с большими дата-центрами уже давно обсчитываются коллизии SHA-1.

Сейчас Марк Стивенс с соавторами опубликовали научную работу, в которой описывают общие принципы генерации документов с блоками сообщений, которые подвержены коллизии SHA-1.

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(1)}$	7f	46	dc	93	a6	b6	7e	01	3b	02	9a	aa	1d	b2	56	0b				
	45	ca	67	d6	88	c7	f8	4b	8c	4c	79	1f	e0	2b	3d	f6				
	14	f8	6d	b1	69	09	01	c5	6b	45	c1	53	0a	fe	df	b7				
	60	38	e9	72	72	2f	e7	ad	72	8f	0e	49	04	e0	46	c2				
$CV_1^{(1)}$	8d	64	d6	17	ff	ed	53	52	eb	c8	59	15	5e	c7	eb	34	f3	8a	5a	7b
$M_2^{(1)}$	30	57	0f	e9	d4	13	98	ab	e1	2e	f5	bc	94	2b	e3	35				
	42	a4	80	2d	98	b5	d7	0f	2a	33	2e	c3	7f	ac	35	14				
	e7	4d	dc	0f	2c	c1	a8	74	cd	0c	78	30	5a	21	56	64				
	61	30	97	89	60	6b	d0	bf	3f	98	cd	a8	04	46	29	a1				
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

CV_0	4e	a9	62	69	7c	87	6e	26	74	d1	07	f0	fe	c6	79	84	14	f5	bf	45
$M_1^{(2)}$	73	46	dc	91	66	b6	7e	11	8f	02	9a	b6	21	b2	56	0f				
	f9	ca	67	cc	a8	c7	f8	5b	a8	4c	79	03	0c	2b	3d	e2				
	18	f8	6d	b3	a9	09	01	d5	df	45	c1	4f	26	fe	df	b3				
	dc	38	e9	6a	c2	2f	e7	bd	72	8f	0e	45	bc	e0	46	d2				
$CV_1^{(2)}$	8d	64	c8	21	ff	ed	52	e2	eb	c8	59	15	5e	c7	eb	36	73	8a	5a	7b
$M_2^{(2)}$	3c	57	0f	eb	14	13	98	bb	55	2e	f5	a0	a8	2b	e3	31				
	fe	a4	80	37	b8	b5	d7	1f	0e	33	2e	df	93	ac	35	00				
	eb	4d	dc	0d	ec	c1	a8	64	79	0c	78	2c	76	21	56	60				
	dd	30	97	91	d0	6b	d0	af	3f	98	cd	a4	bc	46	29	b1				
CV_2	1e	ac	b2	5e	d5	97	0d	10	f1	73	69	63	57	71	bc	3a	17	b4	8a	c5

2.3 MD5

MD5 (англ. Message Digest 5) — 128-битный алгоритм хеширования, разработанный профессором Рональдом Л. Ривестом из Массачусетского технологического института (Massachusetts Institute of Technology, MIT) в 1991 году. Предназначен для создания **«отпечатков»** или **дайджестов сообщения произвольной длины и последующей проверки их подлинности.** Широко применялся для проверки целостности информации и хранения хешей паролей.

На вход алгоритма поступает входной поток данных, хеш которого необходимо найти. Длина сообщения измеряется в битах и может быть любой (в том числе нулевой). Запишем длину сообщения в L. Это число целое и неотрицательное. Кратность каким-либо числам необязательна. После поступления данных идёт процесс подготовки потока к вычислениям.

В данном алгоритме предполагается наличие 5 шагов, а именно:

1. Выравнивание потока
2. Добавление длины сообщения
3. Инициализация буфера
4. Вычисление в цикле
5. Результат вычислений

На первом шаге *“Выравнивание потока”* сначала дописывают единичный бит в конец потока, затем необходимое число нулевых бит. Входные данные выравниваются так, чтобы их новый размер был сравним с 448 по модулю 512. Выравнивание происходит, даже если длина уже сравнима с 448.

На втором шаге в оставшиеся 64 бита дописывают 64-битное представление длины данных до выравнивания. Сначала записывают младшие 4 байта. Если длина превосходит $2^{64} - 1$, то дописывают только младшие биты. После этого длина потока станет кратной 512. Вычисления будут основываться на представлении этого потока данных в виде массива слов по 512 бит.

На третьем для вычислений используются четыре переменные размером 32 бита и задаются начальные значения в 16-ричном виде. В этих переменных будут храниться результаты промежуточных вычислений.

Во время 4-го шага *“Вычисление в цикле”* происходит 4 раунда, в которых сохраняются значения, оставшиеся после операций над предыдущими блоками. После всех операций суммируются результаты

двух последних циклов. Раундов в MD5 стало 4 вместо 3 в MD4. Добавилась новая константа для того, чтобы свести к минимуму влияние входного сообщения. В каждом раунде на каждом шаге и каждый раз константа разная. Она суммируется с результатом и блоком данных. Результат каждого шага складывается с результатом предыдущего шага. Из-за этого происходит более быстрое изменение результата. Изменился порядок работы с входными словами в раундах **2 и 3.**

В итоге на 5-ом шаге мы получим результат вычислений, который находится в буфере -это и есть хеш. Если выводить побайтово, начиная с младшего байта первой переменной и закончив старшим байтом последней, то мы получим MD5-хеш.

2.3.1 Уязвимости MD5

Алгоритм MD5 уязвим к некоторым атакам. Например, возможно создание двух сообщений с одинаковой хеш-суммой.

На данный момент существуют несколько видов взлома хешей **MD5 — подбора сообщения с заданным хешем:**

- Перебор по словарю
- Brute-force
- RainbowCrack
- Коллизия хеш-функции

При этом методы перебора по словарю и **brute-force** могут использоваться для взлома хеша других хеш-функций (с небольшими изменениями алгоритма). **RainbowCrack** требует предварительной подготовки радужных таблиц, которые создаются для заранее определённой хеш-функции. Поиск коллизий специфичен для каждого алгоритма. Рассмотрим каждый вид **«взлома»** по отдельности.

2.3.2 Атаки переборного типа

В криптографии атака полного перебора или исчерпывающий поиск ключей -это стратегия, которая теоретически может быть использована против любых зашифрованных данных. Злоумышленник, который не может воспользоваться слабостью в системе шифрования, реализовывает атаку подобного типа. Она включает в себя систематическую проверку всех возможных ключей, пока не будет найден правильный. В худшем случае для взлома сообщения потребуется задействовать всю вычислительную мощность. Перебор по словарю — атака на систему защиты, применяющая метод полного перебора предполагаемых паролей, используемых для аутентификации, осуществляемого путём последовательного пересмотра всех слов (паролей в чистом виде) определённого вида и длины из словаря с целью последующего взлома системы и получения доступа к секретной информации. Как видно из определения, атаки по словарю являются атаками полного перебора. Единственное отличие состоит в том, что данные атаки обычно более эффективны так как становится не нужным перебирать все комбинации символов, чтобы добиться успеха. Злоумышленники используют обширные списки наиболее часто используемых паролей таких как, имена домашних животных, вымышленных персонажей или конкретно характерных слов из словаря – отсюда и название атаки. Однако если пароль действительно уникален (не является комбинацией слов), атака по словарю не сработает. В этом случае использование атаки полного перебора -единственный вариант. Для полного перебора или перебора по словарю можно использовать программы **PasswordsPro**, **MD5BFCPF**, **John the Ripper.** Для перебора по словарю существуют готовые словари.

2.3.3 RainbowCrack

Это ещё один метод взлома хеша. Он основан на генерировании большого количества хешей из набора символов, чтобы по получившейся базе вести поиск заданного хеша. Радужные таблицы состоят из хеш-цепочек и более эффективны, чем предыдущий упомянутый тип атак, поскольку они оптимизируют требования к хранению, хотя поиск выполняется немного медленнее. Радужные таблицы отличаются от хеш-таблиц тем, что они создаются с использованием как хеш-функций, так и функций редукции. **Цепочки хешей** — метод для уменьшения требования к объёму памяти. Главная идея — определение функции редукции R, которая сопоставляет значениям хеша значения из таблицы. Стоит отметить, что R не является обращением хеш-функции. Радужные таблицы являются развитием идеи таблицы хеш-цепочек. Функции редукции применяются по очереди, перемежаясь с функцией хеширования. Использование последовательностей функций редукции изменяет способ поиска по таблице. Поскольку хеш может быть найден в любом месте цепочки, необходимо сгенерировать несколько различных цепочек. Существует множество систем взлома паролей и веб-сайтов, которые используют подобные таблицы. Основная идея

данного метода — достижение компромисса между временем поиска по таблице и занимаемой памятью. Конечно, использование радужных таблиц не гарантирует 100% успеха взлома систем паролей. Но чем больше набор символов, используемый для создания радужной таблицы, и чем продолжительнее хеш-цепочки, тем больше будет шансов получить доступ к базе данных исходных паролей.

2.3.4 Коллизии MD5

Коллизия хеш-функции — это получение одинакового значения функции для разных сообщений и идентичного начального буфера. В отличие от коллизий, псевдоколлизии определяются как равные значения хеша для разных значений начального буфера, причём сами сообщения могут совпадать или отличаться. В 1996 году Ганс Доббертин нашёл псевдоколлизии в MD5, используя определённые инициализирующие векторы, отличные от стандартных. Оказалось, что можно для известного сообщения построить второе такое, что оно будет иметь такой же хеш, как и исходное. С точки зрения математики, это означает следующее: $MD5(I_V, L_1) = MD5(I_V, L_2)$, где I_V — **начальное значение буфера**, а L_1 и L_2 — **различные сообщения.** MD5 был тщательно изучен криптографическим сообществом с момента его первоначального выпуска и до 2004 года демонстрировал лишь незначительные недостатки. Однако летом 2004 года криптографы Ван Сяоюнь и Фэн Дэнго продемонстрировали алгоритм способный генерировать MD5-коллизии с использованием стандартного вектора инициализации. Позже данный алгоритм был усовершенствован, как следствие время поиска пары сообщений значительно уменьшилось, что позволило находить коллизии с приемлемой вычислительной сложностью. Как оказалось, в MD5 вопрос коллизий не решается.

Глава 3

Отечественные алгоритмы хэширования

3.1 Стрибог

Стрибог — это семейство хеш-функций, включающее в себя всего две функции. Функцию с длиной выходного значения в 256 бит и функцию с длиной выходного значения в 512 бит. Обе эти функции имеют одинаковую структуру и отличаются друг от друга только начальным внутренним состоянием. Входными данными для обеих функций является блок данных длиной 512 бит. В случае, если длина сообщения больше 512 бит, то происходит разбиение сообщения на блоки. В случае же, если длина меньше 512 бит, то производится дополнение сообщения.

Прежде чем приступать к описанию алгоритма, расскажем о преобразованиях, которые используются при вычислении хеш-функции.

3.1.1 Преобразования

1. **X-преобразование.** На вход функции X подаются две последовательности длиной 512 бит каждая, выходом функции является XOR этих последовательностей.

   $X[K] : V_{512} \to V_{512}, X[k](a) = k \oplus a, k, a \in V_{512}$

2. **S - преобразование.** Функция **S** является обычной функцией подстановки. Каждый байт из 512-битной входной последовательности заменяется соответствующим байтом из таблицы подстановок π

   $S : V_{512} \to V_{512}, S(a) = S(a_{63}||...||a_0) = \pi(a_{63}||...||\pi(a_0)$

   Таблица π является константой и может быть записана в виде массива

3. **P-преобразование.** Функция перестановки. Для каждой пары байт из входной последовательности происходит замена одного байта другим.

   $P : V_{512} \to V_{512}, P(a) = S(a_{63}||...||a_0) = a_{\tau(63)}||...||a_{\tau(0)}$

   Таблица перестановок τ также является константой

4. **L-преобразование.** Представляет собой умножение 64-битного входного вектора на бинарную матрицу **A** размерами 64x64.

   $L : V_{512} \to V_{512}, L(a) = S(a_7||...||a_0) = l(a_7)||...||l(a_0)$

3.1.2 Функция сжатия

Основным элементом любой хеш-функции является функция сжатия. Опишем используемую в новом стандарте функцию сжатия g_n в виде алгоритма. Пусть **h**, **N** и **m** — 512-битные последовательности. Для вычисления функции $g(N, m, h)$ необходимо проделать следующие шаги:

1. Вычислить значение $K = h \oplus N$
2. Присвоить значение $K = S(K)$
3. Присвоить значение $K = P(K)$
4. Присвоить значение $K = L(K)$
5. Вычислить $t = E(K, m)$

6. Присвоить значение $t = h \oplus t$
7. Вычислить значение $G = t \oplus m$
8. Вернуть G в качестве результата вычисления функции $g(N, m, h)$

$E(K, m)$, которая выполняет нижеприведенные действия:

1. Вычислить значение **state** $= K \oplus m$
2. Для i=0 по 11 выполнить:
   - Присвоить значение state $= S(state)$
   - Присвоить значение state $= P(state)$
   - Присвоить значение state $= L(state)$
   - Вычислить $K = KeySchedule(K, i)$
   - Присвоить значение state $= state \oplus K$
3. Вернуть Вернуть **state** в качестве результата.

3.1.3 Вычисление хеш-функции

Теперь опишем процедуру формирования хеш-значения. Для любого входного сообщения M:

1. Присвоить начальные значения внутренних переменных:
   - Для хеш-функции с длиной выхода 512 бит: $h = iv = 0x00^{64}$. Для хеш-функции с длиной выхода 256 бит: $h = iv = 0x01^{64}$.
   - $N = 0^{512}$
   - $\Sigma = 0^{512}$
2. Проверить следующее условие: длина сообщения M<512. Если условие выполняется перейти к пункту 3. В противном случае выполнить последовательность вычислений:
   - m — последние 512 бит сообщения M
   - $h = g(N, m, h)$
   - $N = (N + 512) \bmod 2^{512}$
   - $\Sigma = (\Sigma + m) \bmod 2^{512}$
   - Обрезать M, убрав последние 512 бит
   - Перейти к шагу 2
3. Произвести дополнение сообщения M до длины в 512 бит по следующему правилу: $m = 0^{511-|M|}||1||M$, $|M|$ — длина сообщения M в битах
4. Вычислить $h = g(N, m, h)$
5. Вычислить $N = (N + |M|) \bmod 2^{512}$
6. Вычислить $\Sigma = (\Sigma + m) \bmod 2^{512}$
7. Вычислить $h = g(0, h, N)$
8. Вычислить $h = g(0, h, \Sigma)$
9. Для хеш-функции с длиной выхода в 512 бит возвращаем h в качестве результата. Для функции с длиной выхода 256 бит возвращаем $MSB_{256}(h)$

3.2 Кузнечик

В отличие от **ГОСТ 28147-89** новый шифр представляет собой не сеть Фейстеля, а т.н. SP-сеть: преобразование, состоящее из нескольких одинаковых раундов, при этом каждый раунд состоит из нелинейного и линейного преобразований, а также операции наложения ключа. В отличие от сети Фейстеля, при использовании SP-сети преобразуется весь входной блок, а не его половина. Такая структура иногда также называется AES-like (похожей на AES), однако, в отличие от последнего у «Кузнечика» есть ряд своих «фишек»:

- линейное преобразование может быть реализовано в с помощью регистра сдвига
- ключевая развертка реализована с помощью сети Фейстеля, в которой в качестве функции используется раундовое преобразование исходного алгоритма.

Длина входного блока «Кузнечика» — 128 бит, ключа — 256 бит.

3.2.1 Преобразования

Шифрование основано на последовательном применении нескольких однотипных раундов, каждый из которых содержит три преобразования: сложение с раундовым ключом, преобразование блоком подстановок и линейное преобразование.

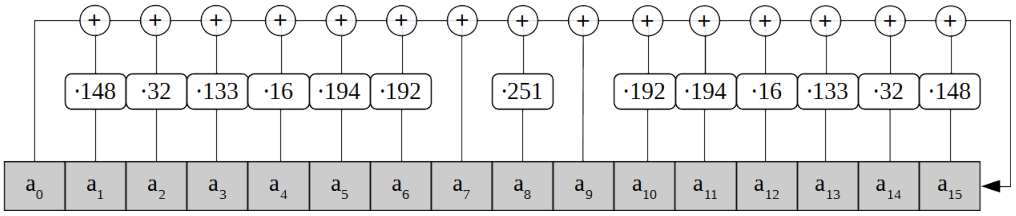
128-битный входной вектор очередного раунда складывается побитно с раундовым ключом:
$X[k] : V_{128} \to V_{128}$, $X[k](a) = k \oplus a$, где $k, a \in V_{128}$

Нелинейное преобразование представляет собой применение к каждому 8-битному подвектору 128-битного входного вектора фиксированной подстановки:
$S : V_{128} \to V_{128}$, $S(a) = S(a_{15}||...||a_0) = \pi(a_{15}||...||\pi(a_0)$

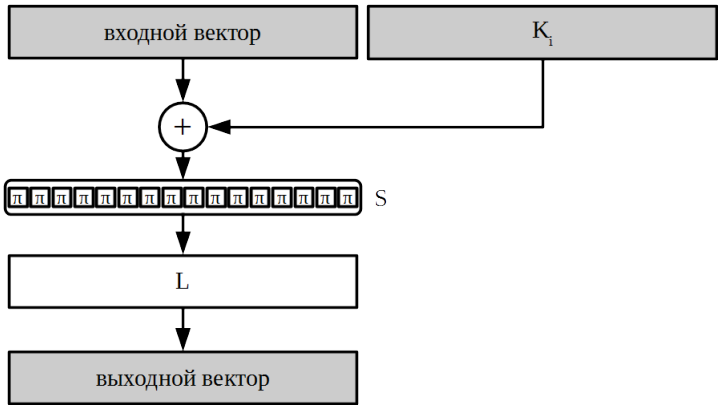
В «Кузнечике» используется та же подстановка, что и в хэш-функции «Стрибог».

Линейное преобразование, как я уже сказал, может быть реализовано не только как обычно в блочных шифрах — матрицей, но и с помощью РСЛОС — линейного регистра сдвига с обратной связью, который движется 16 раз.

Сам регистр реализуется над полем Галуа по модулю неприводимого многочлена степени 8:
$p(x) = x^8 + x^7 + x^6 + x + 1$:



Раундовое преобразование можно изобразить следующим образом:

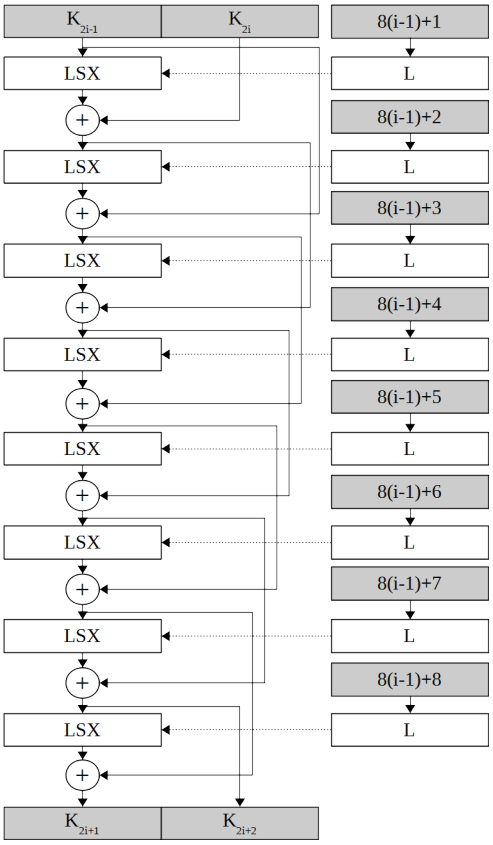


3.2.2 Выработка раундовых ключей

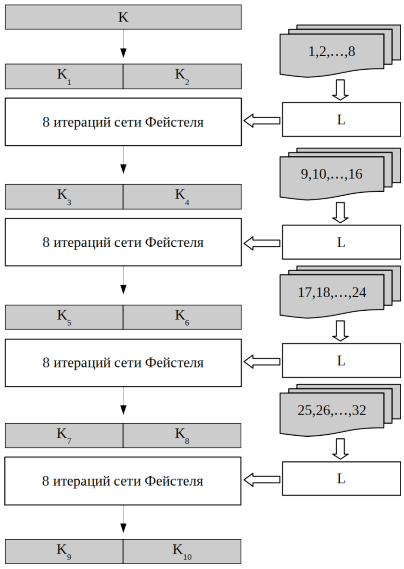
Рассмотрим теперь процедуру генерации раундовых ключей из мастер-ключа. Первые два получаются разбиением мастер-ключа пополам. Далее для выработки очередной пары раундовых ключей используется 8 итераций сети Фейстеля, где, в свою очередь, в качестве раундовых ключей используется счетчиковая последовательность, прошедшая через линейное преобразование алгоритма:

$F[k] : V_{128} \times V_{128} \to F[k](a_1, a_0) = (LSX[k](a_0, a_1))$

Раунд ключевой развертки можно представить следующим образом:



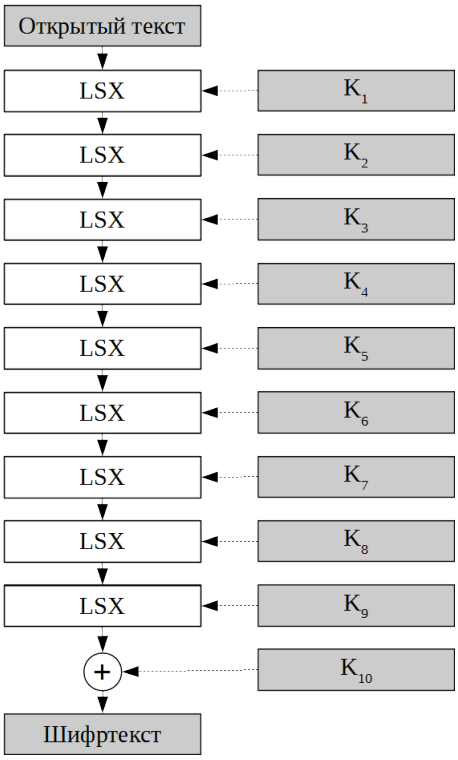
А всю процедуру выработки раундовых ключей так:



3.2.3 Шифрование и расшифрование

В результате, шифрование одного 128-битного входного блока описывается следующим уравнением:

$E_{K_1...,K10}(a) = X[K_{10}]LSX[K_2]LSX[K_1](a)$,
А в виде блок-схемы может быть представлено так:



Расшифрование реализуется обращением базовых преобразований и применением их в обратном порядке:
$D_{K_1...,K10}(a) = X[K_1]S^{-1}L^{-1}X[K_2]...S^{-1}L^{-1}X[K_9]S^{-1}L^{-1}X[K_{10}](a)$

Глава 4

Реализация и использование программы Hash Streebog

4.0.1 Возможности Hash Streebog

Hash Streebog представляет из себя приложение для Windows, для проверки целостности и подлинности файлов посредством вычисления контрольной суммы.

Hash Streebog использует алгоритм хеширования, **ГОСТ Р 34.11-2012 - Стрибог** с функцией длиной выходного значения в **256 бит** и функцию с дилной входного значения в **512 бит**. Обе эти функции имеют одинаковую структуру и отличаются друг от друга только начальным внутренним состоянием. Выходным данными для обеих функций является блок данных длиной **512 бит**. В случае, если длина сообщения больше **512 бит**, то происходит разбиение сообщения на блоки. В случае же, если длина меньше **512 бит**, то производится дополнение сообщения.

4.0.2 Использование Hash Streebog

Основное окно приложения **Hash Streebog** интуитивно понятно, для добавления файлов или же целых папок требуется перейти во вкладку **Add** - которая находится в **menuStrip**

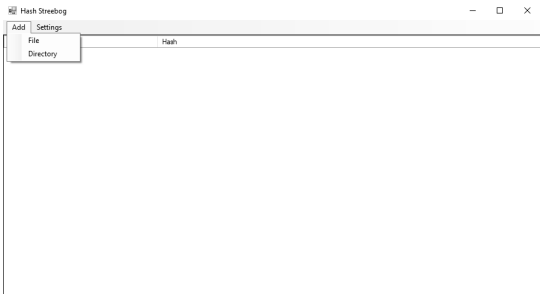


Рис. 4.1: Основное окно взаимодействия и добавления файлов для получения хэш-функции

Для изменения входного значения функции, требуется перейти во вкладку **Settings** - которая находится в **menuStrip**, в **Hash code size**, можно выбрать размер входной функции **256 бит**, которая стоит по стандарту, или же **512 бит**

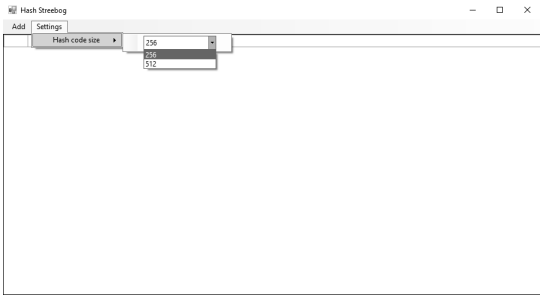


Рис. 4.2: Окно выбора размера входной функции

4.0.3 Литература и ссылки на скачивания

Статистическое тестирование российского стандарта функции хэширования ГОСТ 34.11-2012:
https://cyberleninka.ru/article/n/statisticheskoe-testirovanie-rossiyskogo-standarta-funktsii-heshirovaniya-gost-34-11-2012-stribog

Дифференциальный анализ шифра Кузнечик:
https://cyberleninka.ru/article/n/differentsialnyy-analiz-shifra-kuznechik

А.П. Алферов, А. Ю. Зубов, А. С. Кузьмин, А.В. Черемушкин Основы криптографии:
https://studfile.net/preview/6311470/

Контрольная сумма:
https://studfile.net/preview/6211065/page:7/

Некоторые методы анализа функций хэширования и их применение к алгоритму ГОСТ Р 34.11-94:
http://www.mathnet.ru/links/aef9a4396008a1c6d22235edbaa62714/mvk68.pdf

Hash Streebog:
https://github.com/danrom11/HashStreebog