МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Систем сбора и обработки данных

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Соснова Максима Евгеньевича
(фамилия, имя, отчество автора)

Разработка клиентского программного обеспечения для обмена данными на основе ассиметричного шифрования.

Направление подготовки ***230200 Информационные системы***

Руководитель

Воронов В.В.
(фамилия, И.О.)

Старший преподаватель кафедры ССОД
(уч. степень, уч. звание)

(подпись, дата)

Автор

Соснов М.Е.
(фамилия, И.О.)

АВТ, АТ-03
(факультет, группа)

(подпись, дата)

Новосибирск, 2014 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Систем сбора и обработки данных

Утверждаю

Зав. кафедрой _____
(подпись)
Белик Д.В.
(фамилия, инициалы)

«___»_____ _____г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студенту ***Соснову Максиму Евгеньевичу***
(фамилия, имя, отчество студента)

Направление подготовки ***230200 Информационные системы***

Факультет автоматики и вычислительной техники

Тема ***Разработка клиентского программного обеспечения для обмена данными на основе ассиметричного шифрования.***

Исходные данные (или цель работы):

Разработка клиентского программного обеспечения, позволяющего производить обмен данными по защищенному протоколу через незащищенный канал.

Структурные части работы:

Изучение предметной области. Выбор параметров будущей системы. Определение основных принципов. Создание протокола с учетом существующих методов шифрования и атак на них. Выбор инструментов для создания клиентского приложения. Создание клиентского приложения. Тестирование приложения и его компонентов. Описание полученной технологии. Создание ВКР.

План-график выполнения работы

№ п/п	Наименование этапа	Планируемые сроки выполнения
1	Анализ существующих алгоритмов шифрования, их преимуществ/недостатков а также известных атак. Анализ способов защиты от известных атак.	12.02.2014 – 01.04.2014
2	Выбор параметров будущей системы. Определение основных принципов.	05.04.2014 – 15.05.2014
3	Создание протокола с учетом существующих методов шифрования и атак на них.	20.05.2014 – 26.05.2014
4	Выбор инструментов для создания клиентского приложения.	26.05.2014 – 27.05.2014
5	Создание клиентского приложения. Тестирование приложения и его компонентов.	27.05.2014 – 09.06.2014
6	Описание полученной технологии. Создание ВКР.	10.06.2014 – 20.06.2014

Задание согласовано и принято к исполнению.

Руководитель	**Автор**
Воронов В.В.	***Соснов М.Е.***
(фамилия, И.О.)	(фамилия, И.О.)
кафедры ССОД	***АВТ, АТ-03***
(уч. степень, уч. звание)	(факультет, группа)
.....	.....
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № ***6550/2*** от « ***4*** » ***декабря*** ***2013*** г.

.....
(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

.....
(фамилия, инициалы секретаря экзаменационной комиссии по защите ВКР)

Аннотация

Содержание

Введение

С появлением сети интернет открылась возможность передавать данные с одного конца мира в другой за короткое время. Однако сеть интернет не является защищенным каналом передачи данных исходя из его строения. При передаче данных пакет, следуя от отправителя к получателю, проходит множество узлов, каждый из которых при должном желании может посмотреть содержимое пакета.

Для конфиденциальности передачи данных используются различные алгоритмы, основанные на шифровании данных и электронных подписях. Например, существуют алгоритмы SSL и TLS, использующие асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Эти алгоритмы повсеместно используются в различных приложениях, использующих сеть Интернет (Веб-браузеры, приложения для работы с электронной почтой, мессенджеры, IP-телефония и тд). Существует множество бесплатных библиотек с открытым исходным кодом для установления безопасного соединения в открытом канале передачи данных по технологиям SSL/TLS, но, как показывают события последних месяцев библиотеки с открытым исходным кодом не всегда справляются со своей работой. Например, в апреле 2014го года была найдена уязвимость в библиотеке OpenSSL, названной “Heartbleed”, которая позволяла получить стороннему человеку расшифрованные переданные данные. Спустя 2 месяца была найдена еще одна уязвимость. Надо заметить, что эти уязвимости были внесены с функционалом в начале 2012-го года. Таким образом, начиная с 2012го года информация, передаваемая с помощью OpenSSL могла быть получена злоумышленниками, знающими об уязвимости. Например, Агентство Национальной Безопасности США использовало эту уязвимость в своих целях 2 года.

Крупным информационным системам, которым нужно обеспечивать конфиденциальность передачи данных, желательно использовать качественное

программное обеспечение для этих целей. Например, крупный банк может позволить себе потратиться на разработку своего решения создания защищенного канала передачи данных в среде интернет.

Целями бакалаврской работы являются:

1. Создание протокола передачи данных по защищенному каналу по типу SSL/TLS, используя RSA, AES шифрования и обмен ключами по алгоритму Diffie-Hellman.
2. Создание клиентского приложения, способного установить защищенное соединение с сервером, по созданному протоколу.

Глава 1: Описание предметной области

1.1. Исследование существующих методов шифрования, их достоинств и недостатков

RSA.

Определение

RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) — асимметричный криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптографические системы с открытым ключом используют так называемые односторонние функции, которые обладают следующим свойством:

Если известно x, то $f(x)$ вычислить относительно просто

Если известно $y = f(x)$, то для вычисления x нет простого (эффективного) пути.

Под односторонностью понимается не теоретическая однонаправленность, а практическая невозможность вычислить обратное значение, используя современные вычислительные средства, за обозримый интервал времени.

В основу криптографической системы с открытым ключом RSA положена сложность задачи факторизации произведения двух больших простых чисел. Для шифрования используется операция возведения в степень по модулю большого числа. Для дешифрования за разумное время (обратной операции) необходимо уметь вычислять функцию Эйлера от данного большого числа, для чего необходимо знать разложения числа на простые множители.

Описание алгоритма

Создание ключей

RSA-ключи генерируются следующим образом:

1.1. Выбираются два различных случайных простых числа p и q заданного размера (например, 1024 бита каждое).

1.2. Вычисляется их произведение $n = p \cdot q$, которое называется *модулем*.

1.3. Вычисляется значение функции Эйлера от числа n: $\varphi(n) = (p-1)(q-1)$.

1.4. Выбирается целое число e ($1 < e < \varphi(n)$),взаимно простое со значением функции $\varphi(n)$. Обычно, в качестве e берут простые числа, содержащие небольшое количество единичных бит в двоичной записи, например, простые числа Ферма 17, 257 или 65537.

1.5. Число e называется открытой экспонентой (англ. *public exponent*)

Время, необходимое для шифрования с использованием быстрого возведения в степень, пропорционально числу единичных бит в e. Слишком малые значения e, например 3, потенциально могут ослабить безопасность схемы RSA.

1.6. Вычисляется число d, мультипликативно обратное к числу e по модулю $\varphi(n)$, то есть число, удовлетворяющее условию: $d \cdot e \equiv 1 \mod \varphi(n)$.

1.7. Число d называется *секретной экспонентой*. Обычно, оно вычисляется при помощи расширенного алгоритма Евклида.

Пара $\{e, n\}$ публикуется в качестве *открытого ключа RSA* (англ. *RSA public key*). Пара $\{d, n\}$ играет роль *закрытого ключа RSA* (англ. *RSA private key*) и держится в секрете.

Шифрование и дешифрование

Шифрование сообщения m открытым ключом: $c \equiv m^e \pmod n$, при этом $0 \leq m < n$

Дешифрование сообщения с закрытым ключом: $m \equiv c^d \pmod n$

Пример алгоритма:

1.1. Выбрать простые числа p и q

1.2. Вычислить n = p * q

1.3. Вычислить m = (p - 1) * (q - 1)

1.4. Выбрать число d взаимно простое с m

1.5. Выбрать число e так, чтобы e * d = 1 (mod m)

1.6. Открытый ключ = (n,e). Закрытый ключ = (n,d)

1.7. Шифрование: $b = a^e \pmod n$

1.8. Дешифровка: $a = b^d \pmod n$

Практический пример:

1.1. Выбрали числа: p=61, q=53

1.2. Вычисляем n=3233

1.3. Вычисляем m`=3120

1.4. Выбираем d=17

1.5. Выбираем e=2753

1.6. Выбираем сообщение m = 65

1.7. Шифруем сообщение m $c = 65^{17} \mod 3233 = 2790$

1.8. Дешифруем сообщение $m = 2790^{2753} \mod 3233 = 65$

Elliptic Curve

Определение

Эллиптическая криптография — раздел криптографии, который изучает асимметричные криптосистемы, основанные на эллиптических кривых над конечными полями. Основное преимущество эллиптической криптографии заключается в том, что на сегодняшний день неизвестно существование субэкспоненциальных алгоритмов решения задачи дискретного логарифмирования.

При использовании алгоритмов на эллиптических кривых полагается, что не существует субэкспоненциальных алгоритмов для решения задачи дискретного логарифмирования в группах их точек. При этом порядок группы точек эллиптической кривой определяет сложность задачи. Считается, что для достижения такого же уровня безопасности как и в RSA требуются группы меньших порядков, что уменьшает затраты на хранение и передачу информации.

Алгоритм

Параметры:

Пусть P - точка эллиптической кривой E над полем G(p), имеющая порядок n. Тогда циклическая подгруппа E порожденная точкой P будет состоять из точек {O, P,2P,3P, . . .,(n?1)P}. Характеристика поля p, уравнение эллиптической кривой E, точка P и ее порядок n являются параметрами кривой. Секретным ключом является число d, которое выбирается случайно из интервала [1, n-1], а открытым ключом является точка Q = dP. Задача вычисления d по известным параметрам кривой и точке Q называется проблемой дискретного логарифма в группе точек эллиптической кривой (ECDLP).

Генерация ключей:

1. Вход: Параметры кривой (p, E, P, n).
2. Выход: Открытый ключ Q и секретный ключ d.
3. Алгоритм:
    a. Выбрать d из [1,n-1].
    b. Вычислить Q = dP.
    c. Вернуть: (Q,d)

Шифрование и дешифрование

Открытый текст m представляется в виде точки *M*, а затем шифруется путем сложения с точкой *kQ*. Отправитель передает точку*C1 = kP* и *C2 = M + kQ* получателю, который использует свой секретный ключ *d* для вычисления *dC1 = d(kP) = kQ* и затем восстанавливает *M = C2 - k*q. Злоумышленник, желающий восстановить *M*, должен вычислить *kQ*. Проблема вычисления *kQ* при знании параметров кривой, *Q* и *C1 = kP*, является аналогом проблемы Диффи-Хеллмана для эллиптических кривых. Ниже предоставлен алгоритм шифрования.

Шифрование

1. Вход: Параметры кривой (p, E, P,n), открытый ключ Q, текст m.
2. Вывод: Криптограмма (C1,C2).
3. Алгоритм:
    a. Представить сообщение m в виде точки M кривой E(Fp).
    b. Выбрать k из R [1,n - 1].
    c. . Вычислить C1 = kP.
    d. . Вычислить C2 = M +kQ.
    e. Вернуть: (C1,C2).

Дешифрование

1. **Вход**: Параметры кривой *(p, E, P,n)*, секретный ключ *d*, криптограмма *(C1,C2)*.
2. **Выход**: Исходный текст *m*.
3. **Алгоритм**:
    a. Вычислить *M = C2 - dC1*,
    b. Вычислить *m* из *M*.
4. **Вернуть**: (*m*).

AES

Описание

Advanced Encryption Standard (**AES**), также известный как Rijndael (произносится [rɛinda:l] (Рэндал)) — симметричный алгоритм блочного шифрования (размер блока 128 бит, ключ 128/192/256 бит), принятый в качестве стандарта шифрования правительством США по результатам конкурс. Этот алгоритм хорошо проанализирован и сейчас широко используется. Национальный институт стандартов и технологий США (англ. National Institute of Standards and Technology, NIST) опубликовал спецификацию AES 26 ноября 2001 года после пятилетнего периода, в ходе которого были созданы и оценены 15 кандидатур. 26 мая 2002 года AES был объявлен стандартом шифрования. По состоянию на 2009 год AES является одним из самых распространённых алгоритмов симметричного шифрования. Поддержка AES (и только его) введена фирмой Intel в семейство процессоров x86 начиная с Intel Core i7-980X Extreme Edition, а затем на процессорах Sandy Bridge.

Алгоритм шифрования

Определения

1. State — промежуточный результат шифрования, который может быть представлен как прямоугольный массив байтов имеющий 4 строки и Nb колонок. Каждая ячейка State содержит значение размером в 1 байт
2. Nb — число столбцов (32-х битных слов), составляющих State. Для стандарта регламентировано Nb = 4
3. Nk — длина ключа в 32-х битных словах. Для AES, Nk = 4, 6, 8.
4. Nr — количество раундов шифрования. В зависимости от длины ключа, Nr = 10, 12 или 14

Схема

Алгоритм имеет четыре трансформации, каждая из которых своим образом влияет на состояние State и в конечном итоге приводит к результату: *SubBytes()*, *ShiftRows()*, *MixColumns()* и *AddRoundKey()*. Общую схему шифрования можно представить как:

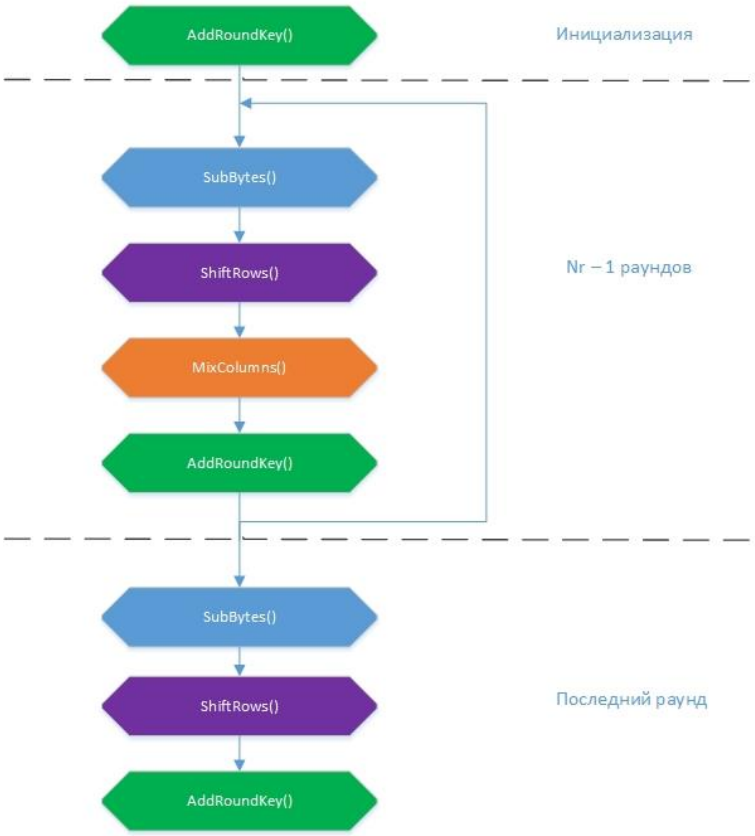


Рисунок 1.1Схема шифрования AES

Подготовка данных

В начале заполняется массив State входными значениями по формуле State[r][c] = input[r + 4c], r = 0,1...4; c = 0,1..Nb. То есть по колонкам. За раз шифруется блок размером 16 байт.

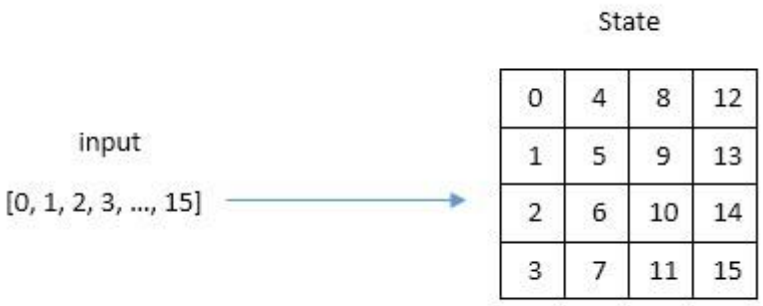


Рис.2. Заполнение State.

Алгоритм оперирует байтами, считая их элементами конечного поля или поля Галуа $GF(2^8)$. Число в скобках — это количество элементов поля или его мощность. Элементами поля $GF(2^8)$ являются многочлены степени не более 7, которые могут быть заданы строкой своих коэффициентов. Байт очень легко

представить в виде многочлена. Например, байту {1,1,1,0,0,0,1,1} соответствует элемент поля $1x^7 + 1x^6 + 1x^5 + 0x^4 + 0x^3 + 0x^2 + 1x^1 + 1x^0 = 1x^7 + 1x^6 + 1x^5 + x + 1$. То, что мы работаем с элементами поля, очень важно потому, что это меняет правила операций сложения и умножения. Этого мы коснемся немного позже.

SubButes()

Преобразование представляет собой замену каждого байта из State на соответствующий ему из константной таблицы Sbox. Значения элементов Sbox представлены в шестнадцатеричной системе счисления. Сама же таблица получена посредством преобразований уже известного нам поля GF(28)

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Рис.3. Sbox AES.

Каждый байт из State можно представить как {xy} в шестнадцатеричной системе счисления. Тогда следует заменять его на элемент, стоящий на пересечении строки x и столбца y. Например, {6e} заменится на {9f}, а {15} на {59}

ShiftRows()

Простая трансформация. Она выполняет циклический сдвиг влево на 1 элемент для первой строки, на 2 для второй и на 3 для третьей. Нулевая строка не сдвигается.

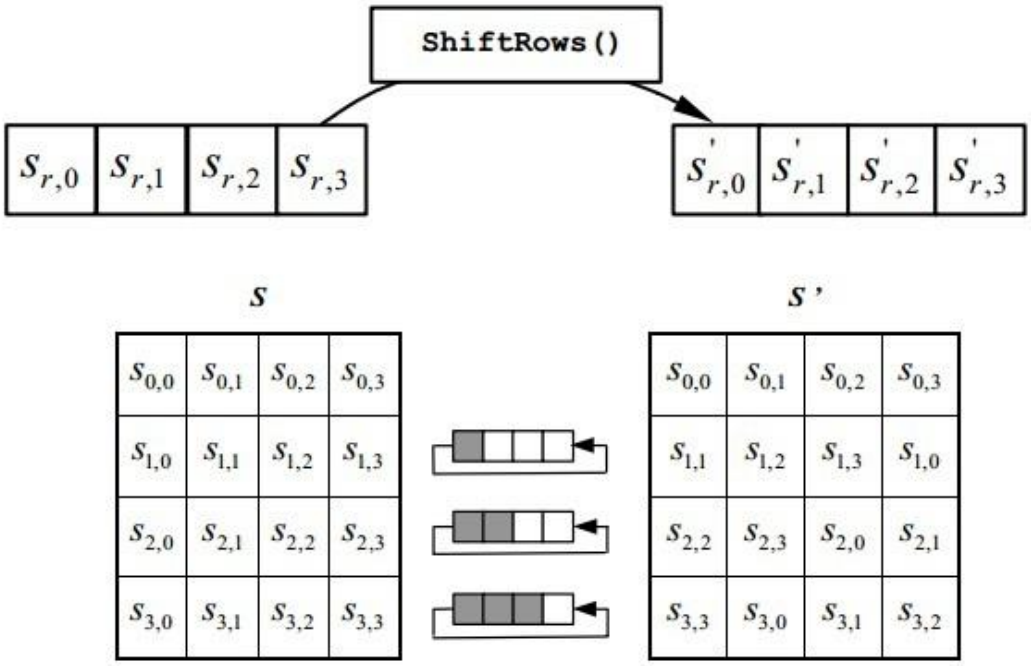


Рис.4. Shift Rows AES.

MixColumns()

В рамках этой трансформации каждая колонка в State представляется в виде многочлена и перемножается в поле GF(2^8) по модулю $x^4 + 1$ с фиксированным многочленом $3x^3 + x^2 + x + 2$. Звучит вроде просто, но малопонятно, как это сделать. Картина становится проще, если посмотреть на эквивалентную матричную запись, предоставленную в официальном документе стандарта:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb.$$

Рис. 5. Mix Columns AES.

При умножении матриц, значение a_{ij} получается как сумма произведений соответствующих элементов i-ой строки первой матрицы и j-ого столбца второй, т. е.

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Рис. 6. Mix Columns AES.

Здесь нужно вспомнить о неработоспособности обычных правил сложения и умножения.

Новые правила:

1. Сложение в поле GF(2^8) эквивалентно операции XOR
2. Умножение на {01} не меняет умножаемое
3. Умножение на {02} производится по правилу: если умножаемое значение меньше {80}, оно сдвигается влево на 1 бит. Если же умножаемое значение больше или равно {80}, оно сначала сдвигается влево на 1 бит, а затем к результату сдвига применяется операция XOR со значением {1b}. Результат может перескочить за значение {ff}, то есть за границы одного байта. В этом случае нужно вернуть остаток от деления результата на {100}.
4. Умножение на другие константы можно выразить через предыдущие

Естественно, это не общие правила арифметики в конечном поле, но в рамках алгоритма придется умножать на три константы при шифровании и на четыре при дешифровке. MixColumns() вместе с ShiftRows() добавляют диффузию в шифр.

AddRoundKey()

Трансформация производит побитовый XOR каждого элемента из State с соответствующим элементом из RoundKey. RoundKey — массив такого же размера, как и State, который строится для каждого раунда на основе секретного ключа функцией KeyExpansion(), которую и рассмотрим далее.

KeyExpansion()

Эта вспомогательная трансформация формирует набор раундовых ключей — KeySchedule. KeySchedule представляет собой длинную таблицу, состоящую из Nb*(Nr + 1) столбцов или (Nr + 1) блоков, каждый из которых равен по размеру State. Первый раундовый ключ заполняется на основе секретного ключа, который вы придумаете, по формуле KeySchedule[r][c] = SecretKey[r + 4c], r = 0,1...4; c = 0,1..Nk.

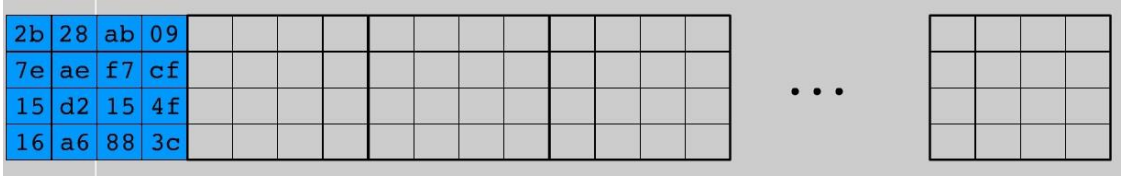


Рис. 6. KeySchedule для AES 128.

На рисунке 6 изображен макет KeySchedule для AES-128: 11 блоков по 4 колонки. Для других вариаций алгоритма будет соответственно (Nr + 1) блоков по Nb колонок. Теперь нам предстоит заполнить пустые места. Для преобразований опять определена константная таблица — Rcon — значения которой в шестнадцатеричной системе счисления.

01	02	04	08	10	20	40	80	1b	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Rcon

Рис. 7. Rcon AES.

Алгоритм дозаполнения KeySchedule:

На каждой итерации работаем с колонкой таблицы. Колонки 0,..,(Nk — 1) уже предварительно заполнены значениями из секретного слова. Начинаем с колонки под номером Nk (в нашем случае с четвертой)

Если номер W_i колонки кратен Nk (в нашем случае каждая четвертая), то берем колонку W_{i-1}, выполняем над ней циклический сдвиг влево на один элемент, затем все байты колонки заменяем соответствующими из таблицы Sbox, как делали это в SubBytes(). Далее выполняем операцию XOR между колонкой

W_{i-Nk}, измененной W_{i-1} и колонкой $Rcon_{i/Nk-1}$. Результат записывается в колонку W_i. Чтобы было немного понагляднее, иллюстрация для i = 4.

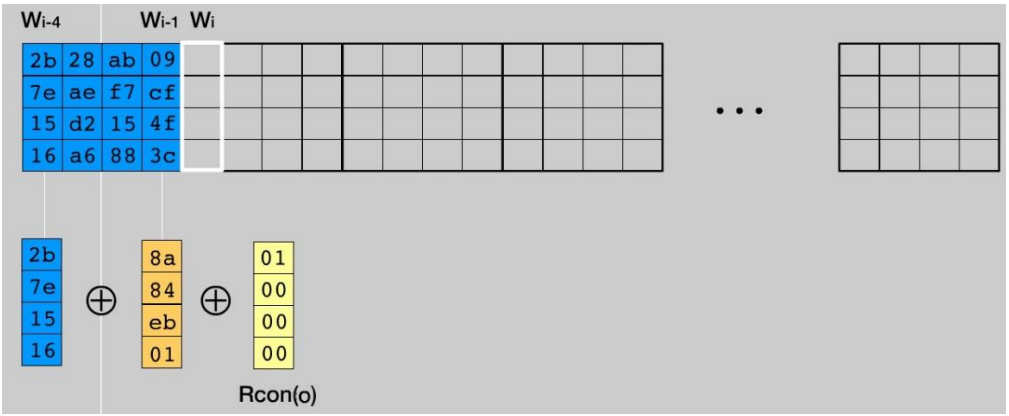


Рис. 8. Алгоритм дозаполнения KeySchedule.

Для остальных колонок выполняем XOR между W_{i-Nk} и W_{i-1}. Результат записываем в W_i

Эта вспомогательная трансформация является самой объемной по написанию и, наверное, самой сложной, после осознания математики в MixColumns(), в алгоритме. Шифроключ обязан состоять из 4*Nk элементов (в нашем случае 16). Но так как мы делаем все это для домашнего применения, то вполне вероятно, что придумывать ключ в 16 символов и запоминать его не каждый будет. Поэтому, если на вход поступила строка длиной менее 16, я в KeySchedule дозаношу значения {01} до нормы.

Как было сказано ранее, KeySchedule используется в трансформации AddRoundKey(). В раунде инициализации раундовым ключом будут колонки с номерами 0,..,3, в первом — с номерами 4,..,7 и тд. Вся суть AddRoundKey() — произвести XOR State и раундового ключа.

Это, собственно, все, что касается процесса шифрования. Выходной массив зашифрованных байтов составляется из State по формуле **output[r + 4c] = State[r][c], r = 0,1...4; c = 0,1..Nb**.

Алгоритм дешифрования

Схема

Идея здесь проста: если с тем же ключевым словом выполнить последовательность трансформаций, инверсных трансформациям шифрования,

то получится исходное сообщение. Такими инверсными трансформациями являются InvSubBytes(), InvShiftRows(), InvMixColumns() и AddRoundKey(). Общая схема алгоритма расшифровки:

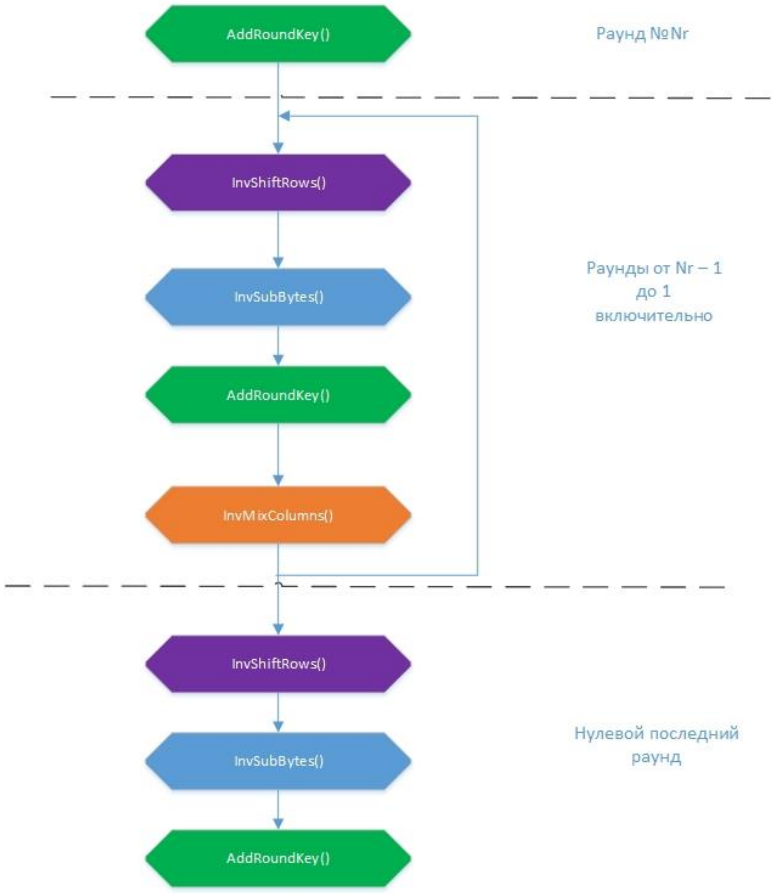


Рис. 9. Схема дешифрования AES.

Стоит отметить, что последовательность добавления раундовых ключей в AddRoundKey() должна быть обратной: от Nr + 1 до 0. Изначально, как и при шифровании, из массива входных байтов формируется таблица State. Затем над ней в каждом раунде производятся преобразования, в конце которых должно получиться расшифрованный файл. Порядок трансформаций немного изменился. Что будет первым, InvSubBytes() или InvShiftRows(), на самом деле не важно, потому что одна из них работает со значениями байтов, а вторая переставляет байты, этих самых значений не меняя, но я придерживался последовательности преобразований в псевдокоде стандарта.

InvSubBytes()

Работает точно так же, как и SubBytes(), за исключением того, что замены делаются из константной таблицы InvSbox.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Рис. 10. InvSbox AES.

Оставшиеся обратные трансформации тоже будут очень похожи на свои прямые аналоги, поэтому в коде не выделяем под них отдельных функций. Каждая функция, описывающая трансформацию, будет иметь входную переменную inv. Если она равна False, то функция будет работать в обычном или прямом режиме(шифрование), если True — в инверсном(дешифровка).

InvShiftRows()

Трансформация производит циклический сдвиг вправо на 1 элемент для первой строки State, на 2 для второй и на 3 для третьей. Нулевая строка не поворачивается.

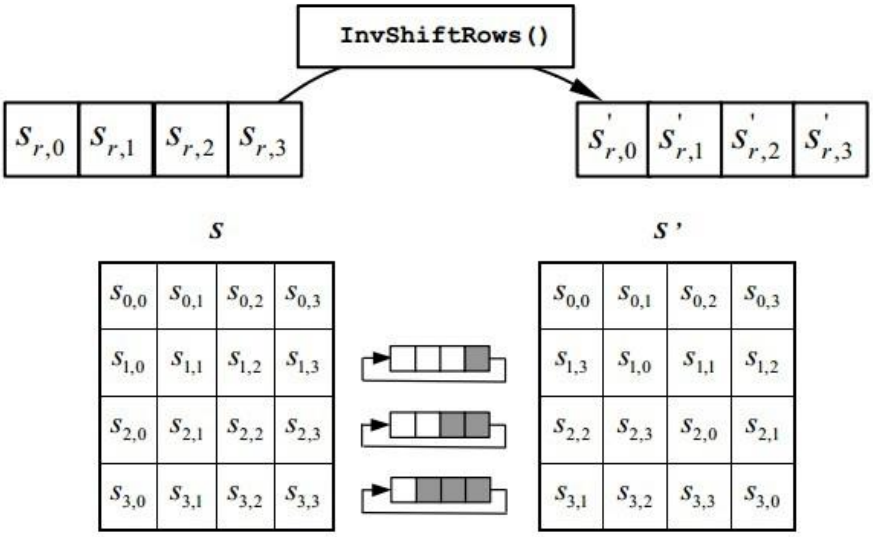


Рис. 11. InvShiftRows AES.

InvMixColumns()

Операции те же, но каждая колонка State перемножается с другим многочленом {0b}x3 + {0d}x2 + {09}x + {0e}. В матричной форме это выглядит так:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < \mathbf{Nb}.$$

Рис. 12. InvMixColumns AES.

Или готовые формулы. Все значения в шестнадцатеричной системе счисления.

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

Рис. 13. InvMixColumns AES

AddRoundKey()

Эта трансформация обратна сама себе в силу свойства операции XOR: *(a XOR b) XOR b = a*

Поэтому никаких изменений в нее вносить не нужно. Набор раундовых ключей формируется таким же образом, как и для шифрования с помощью функции KeyExpansion(), но раундовые ключи необходимо подставлять в обратном порядке.

XOR

XOR шифрование основано на свойстве исключающего ИЛИ. Алгоритм:

1. a XOR 0 = a
2. a XOR a = 0
3. a XOR b = b XOR a
4. (a XOR b) XOR b = a

Таким образом, мы получаем самое быстрое симметричное шифрование. При этом в данной записи a – сообщение, b –ключ.

Обмен ключами Diffie-Hellman

Протокол Ди́ффи — Хе́ллмана (англ. *Diffie-Hellman*, *DH*) — криптографический протокол, позволяющий двум и более сторонам получить общий секретный ключ, используя незащищенный от прослушивания канал связи. Полученный ключ используется для шифрования дальнейшего обмена с помощью алгоритмов симметричного шифрования.

Алгоритм

Схема

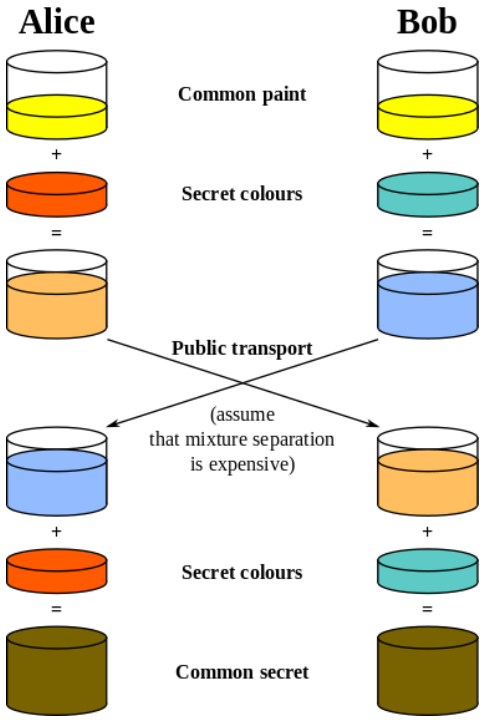


Рис. 14. Схема обмена ключами по протоколу Diffie-Hellman.

Пример:

Пусть Алена и Борис хотят обменяться ключами.

1. Алена и Борис соглашаются на том, чтобы использовать p = 23 и g = 5
2. Алена выбирает секретное число a=6 и посылает Борису $A = g^a \bmod p$
   a. $A = 5^6 \bmod 23 = 15{,}625 \bmod 23 = 8$
3. Борис выбирает секретное число b=15 и посылает Алене $B = g^b \bmod p$
   a. $B = 5^{15} \bmod 23 = 30{,}517{,}578{,}125 \bmod 23 = 19$
4. Алена вычисляет общий секретный ключ $s = B^a \bmod p$
   a. $s = 19^6 \bmod 23 = 47{,}045{,}881 \bmod 23 = 2$

5. Борис вычисляет общий секретный ключ $s = A_b \bmod p$
    a. $s = 8^{15} \bmod 23 = 35{,}184{,}372{,}088{,}832 \bmod 23 = 2$

В результате получаем общий секрет S. который неизвестен никому, кроме Алены и Бориса.

Примечание:

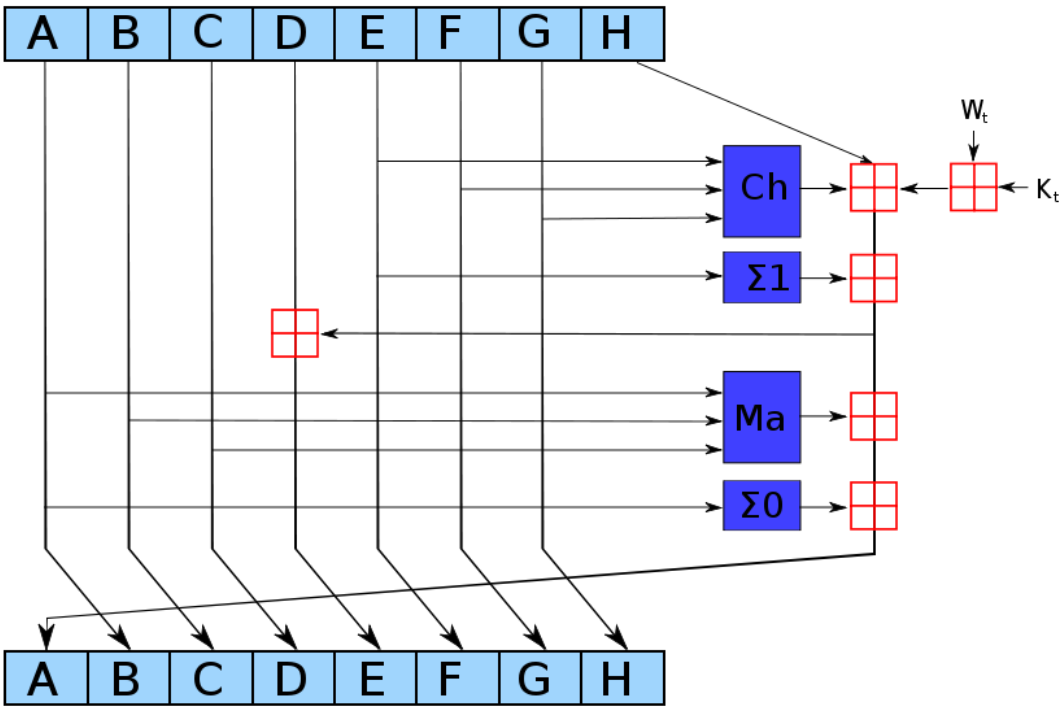
1. p - простое число
2. a,b - натуральные числа, такие, что a<p, b<p
3. g - первообразный корень по модулю p

SHA-2

SHA-2 (англ. Secure Hash Algorithm Version 2 — безопасный алгоритм хеширования, версия 2) — семейство криптографических алгоритмов — однонаправленных хеш-функций, включающее в себя алгоритмы SHA-224, SHA-256, SHA-384 и SHA-512. Хеш-функции предназначены для создания «отпечатков» или «дайджестов» сообщений произвольной битовой длины. Применяются в различных приложениях или компонентах, связанных с защитой информации.

Хеш-функции семейства SHA-2 построены на основе структуры Меркла — Дамгарда.

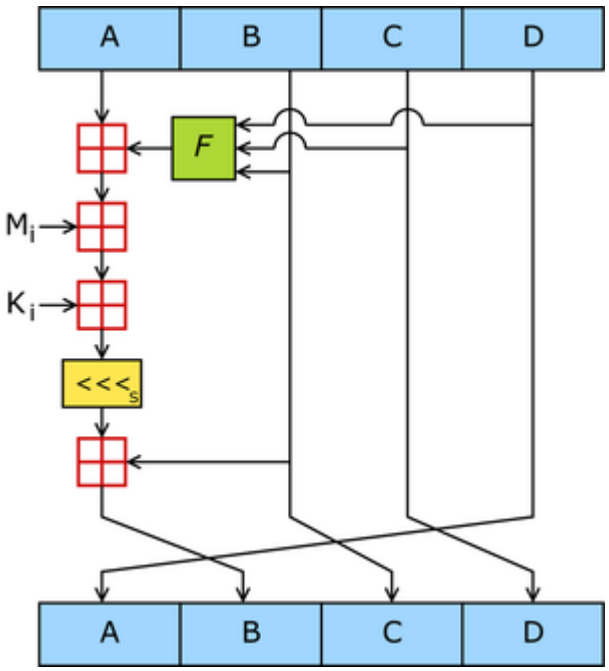
Исходное сообщение после дополнения разбивается на блоки, каждый блок — на 16 слов. Алгоритм пропускает каждый блок сообщения через цикл с 64-мя или 80-ю итерациями (раундами). На каждой итерации 2 слова преобразуются, функцию преобразования задают остальные слова. Результаты обработки каждого блока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



MD5

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

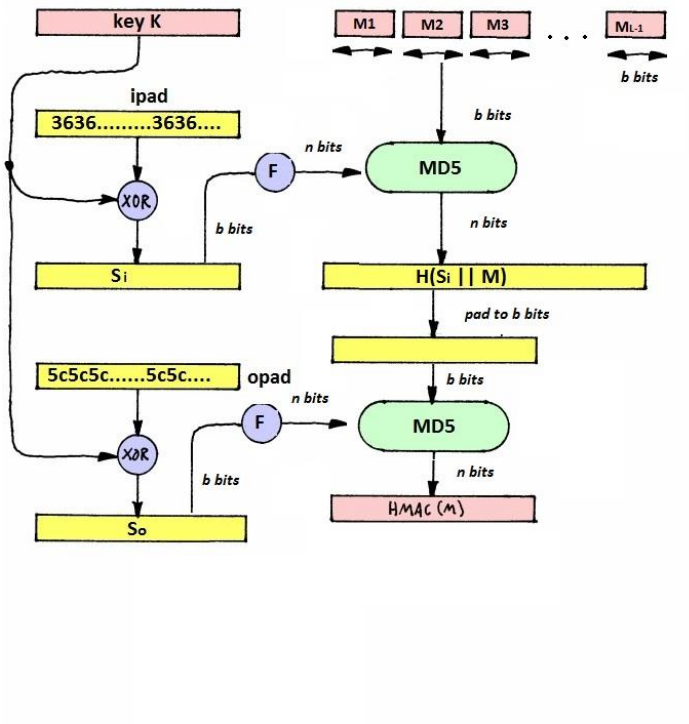
Алгоритм MD5 уязвим к некоторым атакам, например возможно создание двух сообщений с одинаковой хеш-суммой, поэтому его использованиене рекомендуется.



HMAC

HMAC (сокращение от англ. hash-based message authentication code, хеш-код аутентификации сообщений). Наличие способа проверить целостность информации, передаваемой или хранящийся в ненадежной среде является неотъемлемой и необходимой частью мира открытых вычислений и коммуникаций. Механизмы, которые предоставляют такие проверки целостности на основе секретного ключа, обычно называют кодом аутентичности сообщения (MAC). Как правило, MAC используется между двумя сторонами, которые разделяют секретный ключ для проверки подлинности информации, передаваемой между этими сторонами. Этот стандарт определяет

MAC. Механизм, который использует криптографические хеш-функции в сочетании с секретным ключом называется HMAC.



Полученный код аутентичности позволяет убедиться в том, что данные не изменялись каким бы то ни было способом с тех пор как они были созданы, переданы или сохранены доверенным источником. Для такого рода проверки необходимо, чтобы, например, две доверяющие друг другу стороны заранее договорились об использовании секретного ключа, который известен только им. Тем самым гарантируется аутентичность источника и сообщения. Недостаток такого подхода очевиден — необходимо наличие двух доверяющих друг другу сторон.

1.2. Исследование существующих решений.

SSL

SSL (англ. Secure Sockets Layer — уровень защищённых сокетов) — криптографический протокол, который обеспечивает безопасность связи. Он использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений. Протокол широко используется для обмена мгновенными сообщениями и передачи голоса через IP

(англ. Voice over IP — VoIP), в таких приложениях, как электронная почта, Интернет-факс и др.

Протокол SSL позволяет общаться клиенту с сервером в сети, предотвращая перехват или фальсификацию. Так как протоколы могут работать либо без SSL либо поверх SSL, то для клиента необходимо указать серверу, хочет ли он установить соединение SSL или нет. Есть две возможности сделать это. Одним из вариантов является использование различных номеров портов для соединения SSL (например, порт 443 для HTTPS). Другой заключается в использовании регулярного номера порта, сервер установит соединение с клиентом, используя протокол конкретного механизма (например, STARTTLS для почты и новостных протоколов). После того как клиент и сервер решили использовать SSL, они ведут переговоры, отслеживая состояние соединения с помощью процедуры рукопожатия[2]. Во время этого рукопожатия клиент и сервер соглашаются на различные параметры, используемые для установки безопасного соединения. После завершения процедуры рукопожатия начинается защищенное соединение. Клиент и сервер используют сеансовые ключи для шифрования и расшифрования данных, которые они посылают друг другу. Это нормальный алгоритм работы по защищенному каналу. В любое время, в связи с внутренним или внешним раздражителем (автоматическое вмешательство или вмешательство пользователя), любая из сторон может пересмотреть сеанс связи. В этом случае, весь процесс повторяется. SSL работает модульным способом.

В протоколе SSL все данные передаются в виде записей-объектов, состоящих из заголовка и передаваемых данных. Передача начинается с заголовка. Заголовок содержит либо два, либо три байта кода длины. Причём, если старший бит в первом байте кода равен единице, то полная длина заголовка равна двум байтам, иначе — длина заголовка равна трём байтам. Код длины записи не включает в себя число байт заголовка.

Аутентификация и обмен ключами

SSL поддерживает 3 типа аутентификации:

1. аутентификация обеих сторон (клиент — сервер),

2. аутентификация сервера с неаутентифицированным клиентом,
3. полная анонимность.

Если сервер аутентифицирован, то его сообщение о сертификации должно обеспечить верную сертификационную цепочку, ведущую к приемлемому центру сертификации. Проще говоря, аутентифицированный клиент должен предоставить допустимый сертификат серверу. Каждая сторона отвечает за проверку того, что сертификат другой стороны ещё не истек и не был отменен. Всякий раз, когда сервер аутентифицируется, канал устойчив (безопасен) к попытке перехвата данных между веб-сервером и браузером, но полностью анонимная сессия по своей сути уязвима к такой атаке. Анонимный сервер не может аутентифицировать клиента. Главная цель процесса обмена ключами — это создание секрета клиента (pre_master_secret), известного только клиенту и серверу. Секрет (pre_master_secret) используется для создания общего секрета (master_secret). Общий секрет необходим для того чтобы создать сообщение для проверки сертификата, ключей шифрования, секрета MAC (message authentication code) и сообщения «finished». Отсылая сообщение «finished», стороны указывают, что они знают верный секрет (pre_master_secret).

Анонимный обмен ключами

Полностью анонимная сессия может быть установлена при использовании алгоритма RSA или Диффи-Хеллмана для создания ключей обмена. В случае использования RSA клиент шифрует секрет (pre_master_secret) с помощью открытого ключа несертифицированного сервера. Открытый ключ клиент узнает из сообщения обмена ключами от сервера. Результат посылается в сообщении обмена ключами от клиента. Поскольку перехватчик не знает закрытого ключа сервера, то ему будет невозможно расшифровать секрет (pre_master_secret). При использовании алгоритма Диффи-Хеллмана открытые параметры сервера содержатся в сообщении обмена ключами от сервера, и клиенту посылают в сообщении обмена ключами. Перехватчик, который не знает приватных значений, не сможет найти секрет (pre_master_secret).

Аутентификация и обмен ключами при использовании RSA

В этом случае обмен ключами и аутентификация сервера может быть скомбинирована. Открытый ключ также может содержаться в сертификате сервера или может быть использован временный ключ RSA, который посылается в сообщении обмена ключами от сервера. Когда используется временный ключ RSA, сообщения обмена подписываются server's RSA или сертификат DSS. Сигнатура включает текущее значение сообщения Client_Hello.random, таким образом старые сигнатуры и старые временные ключи не могут повторяться. Сервер может использовать временный ключ RSA только однажды для создания сессии. После проверки сертификата сервера клиент шифрует секрет (pre_master_secret) при помощи открытого ключа сервера. После успешного декодирования секрета (pre_master_secret) создается сообщение «finished», тем самым сервер демонстрирует, что он знает частный ключ, соответствующий сертификату сервера.

Когда RSA используется для обмена ключами, для аутентификации клиента используется сообщение проверки сертификата клиента. Клиент подписывается значением, вычисленным из master_secret и всех предшествующих сообщений протокола рукопожатия. Эти сообщения рукопожатия включают сертификат сервера, который ставит в соответствие сигнатуре сервера, сообщение Server_Hello.random, которому ставит в соответствие сигнатуру текущему сообщению рукопожатия.

Аутентификация и обмен ключами при использовании Diffie-Hellman

В этом случае сервер может также поддерживать содержащий конкретные параметры алгоритм Диффи-Хеллмана или может использовать сообщения обмена ключами от сервера для посылки набора временных параметров, подписанных сертификатами DSS или RSA. Временные параметры хэшируются с сообщением hello.random перед подписанием для того, чтобы злоумышленник не смог совершить повтор старых параметров. В любом случае клиент может проверить сертификат или сигнатуру, для уверенности, что параметры принадлежат серверу.

Если клиент имеет сертификат, содержащий параметры алгоритма Diffie-Hellman, то сертификат также содержит информацию, требуемую для того, чтобы завершить обмен ключами. Заметим, что в этом случае клиент и сервер должны будут сгенерировать те же Diffie-Hellman результаты (pre_master_secret), каждый раз, когда они устанавливают соединение. Для того чтобы предотвратить остановку секрета (pre_master_secret) в памяти компьютера на время дольше, чем необходимо, секрет должен быть переведен в общий секрет (master_secret) настолько быстро, на сколько это возможно. Параметры клиента должны быть совместимы с теми, которые поддерживает сервер для того, чтобы работал обмен ключами.

Протокол записи

Протокол записи (Record Layer) — это уровневый протокол. На каждом уровне сообщения включают поля для длины, описания и проверки. Протокол записи принимает сообщения, которые нужно передать, фрагментирует данные в управляемые блоки, разумно сжимает данные, применяя MAC (message authentication code), шифрует и передаёт результат. Полученные данные он расшифровывает, проверяет, распаковывает, собирает и доставляет к более верхним уровням клиента.

Существует четыре протокола записи:

1. Протокол рукопожатия (handshake protocol);
2. Протокол тревоги (alert protocol);
3. Протокол изменения шифра (the change cipher spec protocol);
4. Протокол приложения (application data protocol);

Если SSL реализация получает тип записи, который ей неизвестен, то эта запись просто игнорируется. Любой протокол, созданный для использования совместно с SSL, должен быть хорошо продуман, так как будет иметь дело с атаками на него. Заметим, что из-за типа и длины записи, протокол не защищен шифрованием. Внимание следует уделить тому, чтобы минимизировать трафик.

Протокол рукопожатия

SSL клиент и сервер договариваются об установлении связи с помощью процедуры рукопожатия. Во время рукопожатия клиент и сервер договариваются о различных параметрах, которые будут использованы, чтобы обеспечить безопасность соединения:

1. Клиент посылает серверу номер версии SSL клиента, зашифрованные параметры, специфичные данные для сеанса и другую информацию, которая нужна серверу, чтобы общаться с клиентом, используя SSL.
2. Сервер посылает клиенту номер версии SSL сервера, зашифрованные параметры, специфичные данные для сеанса и другую информацию, которая нужна серверу, чтобы общаться с клиентом по протоколу SSL. Сервер также посылает свой сертификат, который требует проверки подлинности клиента. После идентификации сервер запрашивает сертификат клиента.
3. Клиент использует информацию, переданную сервером для проверки подлинности. Если сервер не может быть проверен, пользователь получает предупреждение о проблеме и о том, что шифрование и аутентификация соединения не может быть установлена. Если сервер успешно прошел проверку, то клиент переходит к следующему шагу.
4. Используя все данные, полученные до сих пор от процедуры рукопожатия, клиент (в сотрудничестве с сервером) создает предварительный секрет сессии, в зависимости от используемого шифра от сервера, шифрует его с помощью открытого ключа сервера (полученного из сертификата сервера, отправленного на 2-м шаге), а затем отправляет его на сервер.
5. Если сервер запросил аутентификацию клиента (необязательный шаг рукопожатия), клиент также подписывает ещё один кусок данных, который является уникальным для этого рукопожатия и известным как для клиента, так и сервера. В этом случае, клиент отправляет все подписанные данные и собственный сертификат клиента на сервер вместе с предварительно зашифрованным секретом.

6. Сервер пытается аутентифицировать клиента. Если клиент не может пройти проверку подлинности, сеанс заканчивается. Если клиент может быть успешно аутентифицирован, сервер использует свой закрытый ключ для расшифровки предварительного секрета, а затем выполняет ряд шагов (которые клиент также выполняет), чтобы создать главный секрет.
7. И клиент, и сервер используют секрет для генерации ключей сеансов, которые являются симметричными ключами, использующимися для шифрования и расшифрования информации, которой обмениваются во время SSL сессии. Происходит проверка целостности (то есть для обнаружения любых изменений в данных между временем, когда он был послан, и временем его получения на SSL-соединении).
8. Клиент посылает сообщение серверу, информируя его, что будущие сообщения от клиента будут зашифрованы с помощью ключа сеанса. Затем он отправляет отдельное, зашифрованное сообщение о том, что часть рукопожатие закончена.
9. И в заключение, сервер посылает сообщение клиенту, информируя его, что будущие сообщения от сервера будут зашифрованы с помощью ключа сеанса. Затем он отправляет отдельное, зашифрованное сообщение о том, что часть рукопожатие закончена.

На этом рукопожатие завершается, и начинается защищенное соединение, которое зашифровывается и расшифровывается с помощью ключевых данных. Если любое из перечисленных выше действий не удается, то рукопожатие SSL не удалось, и соединение не создается.

Протокол изменения параметров шифрования

Протокол изменения параметров шифрования существует для сигнализации перехода в режим шифрования. Протокол содержит единственное сообщение, которое зашифровано и сжато при текущем установленном соединении. Сообщение состоит только из одного бита со значением 1.

Сообщение изменения шифра посылается клиентом и сервером для извещения принимающей стороны, что последующие записи будут защищены в

соответствии с новым договоренным CipherSpec и ключами. Принятие этого сообщения заставляет получателя отдать приказ уровню записи незамедлительно копировать состояние отложенного чтения в состояние текущего чтения. Сразу после послания этого сообщения, тот кто послал должен отдать приказ уровню записи перевести режим отложенной записи в режим текущей записи. Сообщение изменения шифра посылается во время рукопожатия, после того как параметры защиты были переданы, но перед тем как будет послано сообщение «finished».

Протокол тревоги

Один из типов проверки, поддерживаемых в протоколе SSL записи, — это протокол тревоги. Сообщение тревоги передаёт трудности, возникшие в сообщении, и описание тревоги. Сообщение тревоги с критическим уровнем незамедлительно прерывает соединение. В этом случае другие соединения, соответствующие сессии, могут быть продолжены, но идентификатор сессии должен быть признан недействительным. Как и другие сообщения, сообщение тревоги зашифровано и сжато, как только указано текущее состояние соединения.

Протокол приложения

Сообщение приложения данных работает на уровне записи. Он фрагментируется, сжимается и шифруется на основе текущего состояния соединения. Сообщения считаются прозрачными для уровня записи.

TLS

TLS (англ. Transport Layer Security) — безопасность транспортного уровня, как и его предшественник SSL (англ. Secure Socket Layers — уровень защищённых сокетов) — криптографические протоколы, обеспечивающие защищённую передачу данных между узлами в сети Интернет[1]. TLS и SSL используют асимметричную криптографию для аутентификации, симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений.

Описание

TLS даёт возможность клиент-серверным приложениям осуществлять связь в сети таким образом, чтобы предотвратить прослушивание и несанкционированный доступ.

Так как большинство протоколов связи могут быть использованы как с, так и без TLS (или SSL), при установке соединения необходимо явно указать серверу, хочет ли клиент устанавливать TLS. Это может быть достигнуто либо с помощью использования унифицированного номера порта, по которому соединение всегда устанавливается с использованием TLS (как например порт 443 для HTTPS), либо с использованием произвольного порта и специальной команды серверу со стороны клиента на переключение соединения на TLS с использованием специальных механизмов протокола (как например STARTTLS для протоколов электронной почты). Как только клиент и сервер договорились об использовании TLS, им необходимо установить защищённое соединение. Это делается с помощью процедуры подтверждения связи. Во время этого процесса клиент и сервер принимают соглашение относительно различных параметров, необходимых для установки безопасного соединения.

Основные шаги процедуры создания защищённого сеанса связи:

- клиент подключается к серверу, поддерживающему TLS, и запрашивает защищённое соединение;
- клиент предоставляет список поддерживаемых алгоритмов шифрования и хеш-функций;
- сервер выбирает из списка, предоставленного клиентом, наиболее надёжные алгоритмы среди тех, которые поддерживаются сервером, и сообщает о своём выборе клиенту;
- сервер отправляет клиенту цифровой сертификат для собственной аутентификации. Обычно цифровой сертификат содержит имя сервера, имя удостоверяющего центра сертификации и открытый ключ сервера;

- клиент может связаться с сервером доверенного центра сертификации и подтвердить аутентичность переданного сертификата до начала передачи данных;
- для генерации сеансового ключа для защищённого соединения, клиент шифрует случайно сгенерированную цифровую последовательность открытым ключом сервера и посылает результат на сервер. Учитывая специфику алгоритма асимметричного шифрования, используемого для установления соединения, только сервер может расшифровать полученную последовательность, используя свой закрытый ключ.

На этом заканчивается процедура подтверждения связи. Между клиентом и сервером установлено безопасное соединение, данные, передаваемые по нему, шифруются и расшифровываются с использованием ключа шифрования до тех пор, пока соединение не будет завершено.

При возникновении ошибки на любом из вышеуказанных шагов подтверждение связи завершится с ошибкой и соединение не будет установлено.

Процедура подтверждения связи в TLS в деталях

Согласно протоколу TLS приложения обмениваются записями, инкапсулирующими (хранящими внутри себя) информацию, которая должна быть передана. Каждая из записей может быть сжата, дополнена, зашифрована или идентифицирована MAC (код аутентификации сообщения) в зависимости от текущего состояния соединения (состояния протокола). Каждая запись в TLS содержит следующие поля: content type (определяет тип содержимого записи), поле, указывающее длину пакета, и поле, указывающее версию протокола TLS.

Когда соединение только устанавливается, взаимодействие идёт по протоколу TLS handshake, *content type* которого 22.

Простое подтверждение связи в TLS

Далее показан простой пример установления соединения, при котором сервер (но не клиент) проходит аутентификацию по его сертификату.

1. Фаза переговоров:

a. Клиент посылает сообщение **ClientHello**, указывая последнюю версию поддерживаемого TLS протокола, случайное число и список поддерживаемых методов шифрования и сжатия, подходящих для работы с TLS;
b. Сервер отвечает сообщением **ServerHello**, содержащим: выбранную сервером версию протокола, случайное число, посланное клиентом, подходящий алгоритм шифрования и сжатия из списка предоставленного клиентом;
c. Сервер посылает сообщение **Certificate**, которое содержит цифровой сертификат сервера (в зависимости от алгоритма шифрования этот этап может быть пропущен);
d. Сервер отсылает сообщение **ServerHelloDone**, идентифицирующее окончание подтверждения связи;
e. Клиент отвечает сообщением **ClientKeyExchange**, которое содержит открытый ключ PreMasterSecret(Этот PreMasterSecret шифруется с помощью открытого ключа сертификата сервера.) или ничего (опять же зависит от алгоритма шифрования);
f. Клиент и сервер, используя ключ **PreMasterSecret** и случайно сгенерированные числа, вычисляют общий секретный ключ. Вся остальная информация о ключе будет получена из общего секретного ключа (и сгенерированных клиентом и сервером случайных значений);

2. Клиент посылает сообщение **ChangeCipherSpec**, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи алгоритмом, используя общий секретный ключ. Это сообщения уровня записей и поэтому имеет тип 20, а не 22;

a. Клиент посылает сообщение **Finished**, которое содержит хеш и MAC, сгенерированные на основе предыдущих сообщений процедуры подтверждения связи;
b. Сервер пытается расшифровать Finished-сообщение клиента и проверить хеш и MAC. Если процесс расшифровки или проверки не удаётся, подтверждение связи считается неудавшимся, и соединение должно быть оборвано;

3. Сервер посылает **ChangeCipherSpec** и зашифрованное сообщение Finished, и в свою очередь клиент тоже выполняет расшифровку и проверку.

С этого момента подтверждение связи считается завершённым, протокол установленным. Всё последующее содержимое пакетов идет с типом 23, а все данные будут зашифрованы.

Подтверждение связи с аутентификацией клиента

В данном примере показана полная аутентификация клиента (в дополнение к аутентификации сервера, как в предыдущем примере) с помощью обмена сертификатами между сервером и клиентом.

1. Фаза переговоров:
   a. Клиент посылает сообщение **ClientHello**, указывая последнюю версию поддерживаемого TLS протокола, случайное число и список поддерживаемых методов шифрования и сжатия, подходящих для работы с TLS;
   b. Сервер отвечает сообщением **ServerHello**, содержащим: выбранную сервером версию протокола, случайное число, посланное клиентом, подходящий алгоритм шифрования и сжатия из списка предоставленного клиентом;
   c. Сервер посылает сообщение **Certificate**, которое содержит цифровой сертификат сервера (в зависимости от алгоритма шифрования этот этап может быть пропущен);

d. Сервер посылает сообщение **CertificateRequest**, которое содержит запрос сертификата клиента для взаимной проверки подлинности;

e. *[Не хватает пункта получения и проверки сертификата клиента]*;

f. Сервер отсылает сообщение **ServerHelloDone**, идентифицирующее окончание подтверждения связи;

g. Клиент отвечает сообщением **ClientKeyExchange**, которое содержит открытый ключ PreMasterSecret или ничего (опять же зависит от алгоритма шифрования);

h. Клиент и сервер, используя ключ **PreMasterSecret** и случайно сгенерированные числа, вычисляют общий секретный ключ. Вся остальная информация о ключе будет получена из общего секретного ключа (и сгенерированных клиентом и сервером случайных значений);

2. Клиент посылает сообщение **ChangeCipherSpec**, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи алгоритмом, используя общий секретный ключ. Это сообщения уровня записей и поэтому имеет тип 20, а не 22;

   a. Клиент посылает сообщение **Finished**, которое содержит хеш и MAC, сгенерированные на основе предыдущих сообщений процедуры подтверждения связи;

   b. Сервер пытается расшифровать Finished-сообщение клиента и проверить хеш и MAC. Если процесс расшифровки или проверки не удаётся, подтверждение связи считается неудавшимся, и соединение должно быть оборвано.

3. Сервер посылает **ChangeCipherSpec** и зашифрованное сообщение Finished, и в свою очередь клиент тоже выполняет расшифровку и проверку.

С этого момента подтверждение связи считается завершённым, протокол установленным. Всё последующее содержимое пакетов идёт с типом 23, а все данные будут зашифрованы.

Возобновление TLS-соединения

Алгоритмы асимметричного шифрования, использующиеся при генерации сеансового ключа, обычно являются дорогими с точки зрения вычислительных мощностей. Для того чтобы избежать их повторения при возобновлении соединения, TLS создаёт специальный ярлык при подтверждении связи, использующийся для возобновления соединения. При этом при обычном подтверждении связи клиент добавляет в сообщение **ClientHello** идентификатор предыдущей сессии *session id*. Клиент связывает идентификатор *session id* с IP-адресом сервера и TCP-портом так, чтобы при соединении к серверу можно было использовать все параметры предыдущего соединения. Сервер сопоставляет идентификатор предыдущей сессии *session id* с параметрами соединения, такими как использованный алгоритм шифрования и master secret. Обе стороны должны иметь одинаковый master secret, иначе соединение не будет установлено. Это предотвращает использование *session id* криптоаналитиком для получения несанкцианированного доступа. Случайные цифровые последовательности в сообщениях **ClientHello** и **ServerHello**позволяют гарантировать, что сгенерированный сеансовый ключ будет отличаться от сеансового ключа при предыдущем соединении. В RFC такой тип подтверждения связи называется*сокращённым*.

1. Фаза переговоров:
    a. Клиент посылает сообщение **ClientHello**, указывая последнюю версию поддерживаемого TLS протокола, случайное число и список поддерживаемых методов шифрования и сжатия, подходящих для работы с TLS; Также в сообщение добавляется идентификатор предыдущего соединения **session id**.
    b. Сервер отвечает сообщением **ServerHello**, содержащим: выбранную сервером версию протокола, случайное число,

посланное клиентом, подходящий алгоритм шифрования и сжатия из списка предоставленного клиентом. Если сервер узнал идентификатор сессии **session id**, то он добавляет в сообщение **ServerHello** тот же самый идентификатор **session id**. Это является сигналом для клиента о том, что можно использовать возобновление предыдущей сессии. Если сервер не узнал идентификатор сессии **session id**, то он добавляет в сообщение **ServerHello**другое значение вместо **session id**. Для клиента это означает, что использовать возобновлённое соединение нельзя. Таким образом, сервер и клиент должны иметь одинаковый master secret и случайные числа для генерации сеансового ключа;

2. Сервер посылает сообщение **ChangeCipherSpec**, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи алгоритмом, используя общий секретный ключ. Это сообщения уровня записей и поэтому имеет тип 20, а не 22;
    a. Сервер посылает зашифрованное сообщение **Finished**, которое содержит хеш и MAC, сгенерированные на основе предыдущих сообщений процедуры подтверждения связи;
    b. Клиент пытается расшифровать **Finished** сообщение сервера и проверить хеш и MAC. Если процесс расшифровки или проверки не удаётся, подтверждение связи считается неудавшимся, и соединение должно быть оборвано;
3. Клиент посылает сообщение **ChangeCipherSpec**, которое указывает на то, что вся последующая информация будет зашифрована установленным в процессе подтверждения связи алгоритмом, используя общий секретный ключ.
    a. Клиент посылает своё зашифрованное сообщение **Finished**;

b. Сервер схожим образом пытается расшифровать **Finished** сообщение клиента и проверить хеш и MAC;

4. С этого момента подтверждение связи считается завершённым, протокол установленным. Всё последующее содержимое пакетов идёт с типом 23, а все данные будут зашифрованы.

Кроме преимуществ с точки зрения производительности, алгоритм возобновления соединения может быть использован для реализации единого входа, поскольку гарантируется, что исходной сессия, как и любая возобновлённая сессия, инициирована тем же самым клиентом (RFC 5077). Это имеет особенно важное значение для реализации FTPS протокола, который в противном случае был бы уязвим к атаке типа человек посередине, при которой злоумышленник мог бы перехватить содержание данных при установлении повторного соединения.

1.3. Выводы по результатам проработки предметной области

Глава 2: Описание нового протокола

2.1. Описание новых методов шифрования и защиты данных

2.2. Создание нового протокола

2.3. Анализ нового протокола с учетом известных атак на существующие системы

Глава 3: Разработка серверного приложения

3.1. Описание будущего проекта

3.2. Создание серверного приложения, описание дизайна

3.3. Оценка получившегося решения

Заключение

1. Общая оценка работы

2. Полнота решения поставленных задач

3. Экономическая и научная значимость работы

Список использованных источников

Приложения