Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ

по лабораторной работе

на тему

Хэш-функции

Выполнил

Студент гр. 053502

Юрьев В. А.

Проверил

Ассистент кафедры информатики

Лещенко Е. А.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc148537787)

[2 Теоретические сведения 3](#_Toc148537788)

[3 Блок-схема алгоритма 4](#_Toc148537789)

[4 Демонстрация работы 5](#_Toc148537790)

[5 Вывод 5](#_Toc148537791)

[Приложение А 6](#_Toc148537792)

1. **ПОСТАНОВКА ЗАДАЧИ**

Реализовать программное средство контроля целостности сообщений с помощью вычисления хэш-функции и алгоритма ГОСТ 34.11. Необходимо выполнить 2 реализации хэш-функций: ГОСТ34.11 и MD5.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Описывается в ГОСТ 34.11-2018 «Информационная технология. Криптографическая защита информации. Функция хэширования» — действующем межгосударственном криптографическом стандарте.

Разработан Центром защиты информации и специальной связи ФСБ России с участием ОАО «ИнфоТеКС» на основе национального стандарта Российской Федерации ГОСТ Р 34.11-2012 и введен в действие с 1 июня 2019 года приказом Росстандарта № 1060-ст от 4 декабря 2018 года.

Стандарт ГОСТ Р 34.11-2012 разработан и введён в качестве замены устаревшему стандарту ГОСТ Р 34.11-94.

Необходимость разработки вызвана потребностью в создании хэш-функции, соответствующей современным требованиям к криптографической стойкости и требованиям стандарта ГОСТ Р 34.10-2012 на электронную цифровую подпись.

В основу хеш-функции положена итерационная конструкция Меркла — Дамгора с использованием MD-усиления. Под MD-усилением понимается дополнение неполного блока при вычислении хеш-функции до полного путём добавления вектора (0 … 01) такой длины, чтобы получился полный блок. Из дополнительных элементов нужно отметить следующие: завершающее преобразование, которое заключается в том, что функция сжатия применяется к контрольной сумме всех блоков сообщения по модулю 2512; при вычислении хеш-кода на каждой итерации применяются разные функции сжатия. Можно сказать, что функция сжатия зависит от номера итерации.

Описанные выше решения позволяют противостоять многим известным атакам. Кратко описание хеш-функции ГОСТ Р 34.11-2012 можно представить следующим образом. На вход хеш-функции подается сообщение произвольного размера. Далее сообщение разбивается на блоки по 512 бит, если размер сообщения не кратен 512, то оно дополняется необходимым количеством бит. Потом итерационно используется функция сжатия, в результате действия которой обновляется внутреннее состояние хеш-функции. Также вычисляется контрольная сумма блоков и число обработанных бит. Когда обработаны все блоки исходного сообщения, производятся ещё два вычисления, которые завершают вычисление хеш-функции: обработка функцией сжатия блока с общей длиной сообщения; обработка функцией сжатия блока с контрольной суммой.

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

1. **БЛОК-СХЕМА АЛГОРИТМА**

Схема алгоритма ГОСТ 34.11 приведена на рисунке 3.1.

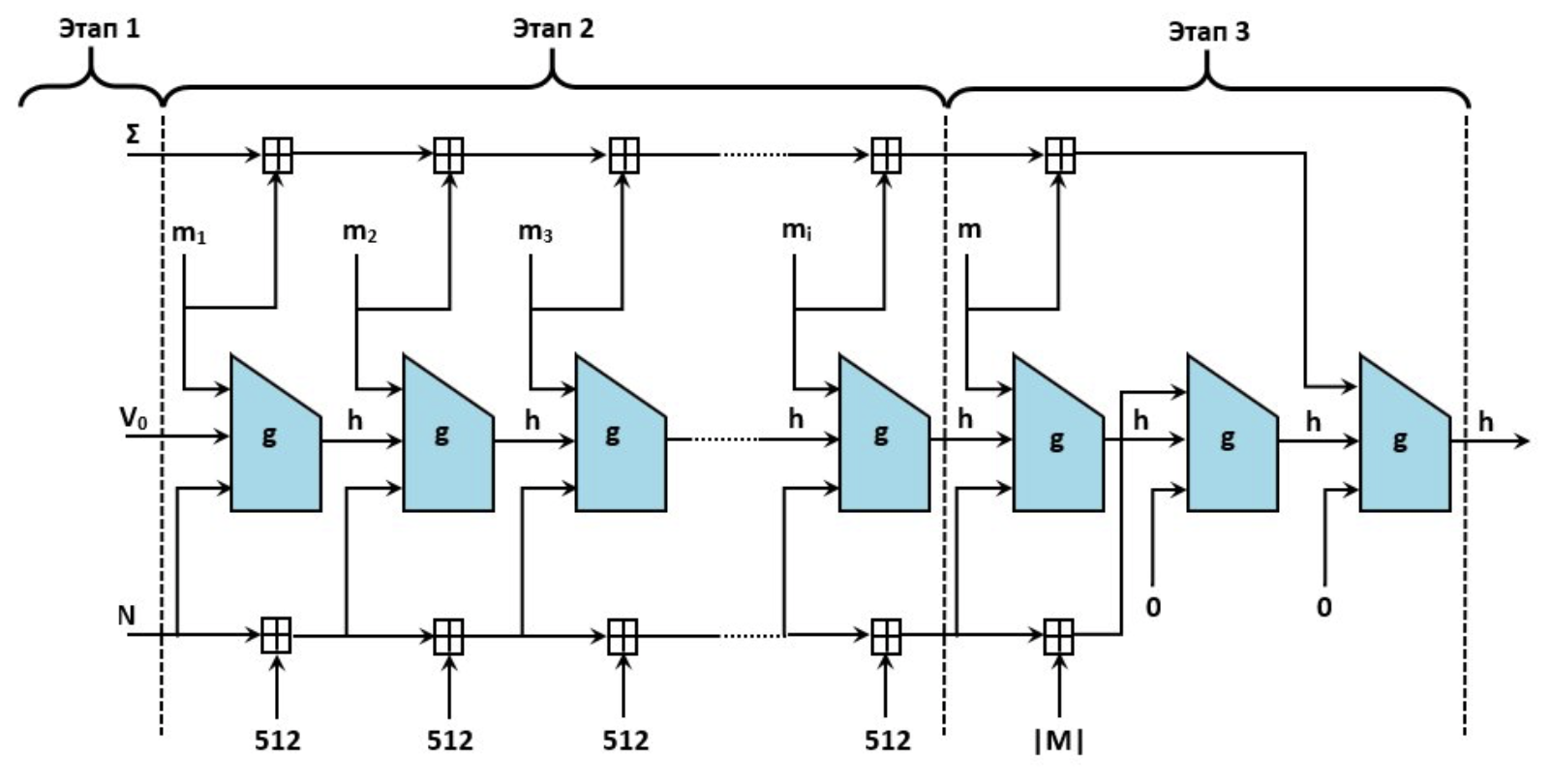


Рисунок 3.1 – Схема алгоритма ГОСТ 34.11

Схема алгоритма MD5 приведена на рисунке 3.2.

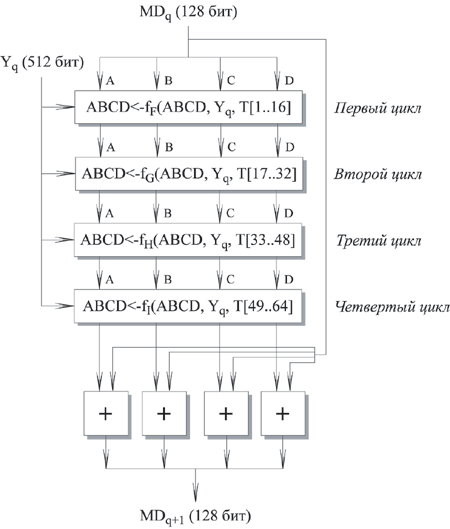


Рисунок 3.2 – Схема алгоритма MD5

1. **ДЕМОНСТРАЦИЯ РАБОТЫ**

Пример работы программы для сообщения “Hello world!” показан на рисунке 4.1.

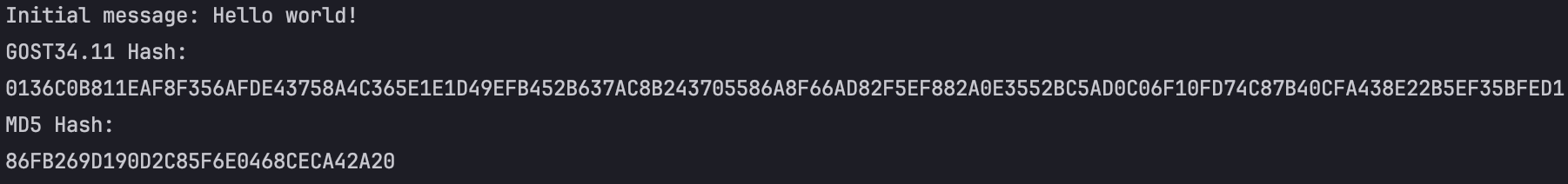


Рисунок 4.1 – Пример работы программы

1. **ВЫВОД**

В ходе выполнения данной лабораторной работы было реализовано программное средство контроля целостности сообщений с помощью вычисления хэш-функции с использованием алгоритмов ГОСТ 34.11 и MD5.

# **ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Код программы**

public static class GOST3411

{

private static byte[] Pi =

{

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

};

private static byte[] t =

{

0, 8, 16, 24, 32, 40, 48, 56,

1, 9, 17, 25, 33, 41, 49, 57,

2, 10, 18, 26, 34, 42, 50, 58,

3, 11, 19, 27, 35, 43, 51, 59,

4, 12, 20, 28, 36, 44, 52, 60,

5, 13, 21, 29, 37, 45, 53, 61,

6, 14, 22, 30, 38, 46, 54, 62,

7, 15, 23, 31, 39, 47, 55, 63

};

private static byte[] l =

{

0x8e, 0x20, 0xfa, 0xa7, 0x2b, 0xa0, 0xb4, 0x70,

0x47, 0x10, 0x7d, 0xdd, 0x9b, 0x50, 0x5a, 0x38,

0xad, 0x08, 0xb0, 0xe0, 0xc3, 0x28, 0x2d, 0x1c,

0xd8, 0x04, 0x58, 0x70, 0xef, 0x14, 0x98, 0x0e,

0x6c, 0x02, 0x2c, 0x38, 0xf9, 0x0a, 0x4c, 0x07,

0x36, 0x01, 0x16, 0x1c, 0xf2, 0x05, 0x26, 0x8d,

0x1b, 0x8e, 0x0b, 0x0e, 0x79, 0x8c, 0x13, 0xc8,

0x83, 0x47, 0x8b, 0x07, 0xb2, 0x46, 0x87, 0x64,

0xa0, 0x11, 0xd3, 0x80, 0x81, 0x8e, 0x8f, 0x40,

0x50, 0x86, 0xe7, 0x40, 0xce, 0x47, 0xc9, 0x20,

0x28, 0x43, 0xfd, 0x20, 0x67, 0xad, 0xea, 0x10,

0x14, 0xaf, 0xf0, 0x10, 0xbd, 0xd8, 0x75, 0x08,

0x0a, 0xd9, 0x78, 0x08, 0xd0, 0x6c, 0xb4, 0x04,

0x05, 0xe2, 0x3c, 0x04, 0x68, 0x36, 0x5a, 0x02,

0x8c, 0x71, 0x1e, 0x02, 0x34, 0x1b, 0x2d, 0x01,

0x46, 0xb6, 0x0f, 0x01, 0x1a, 0x83, 0x98, 0x8e,

0x90, 0xda, 0xb5, 0x2a, 0x38, 0x7a, 0xe7, 0x6f,

0x48, 0x6d, 0xd4, 0x15, 0x1c, 0x3d, 0xfd, 0xb9,

0x24, 0xb8, 0x6a, 0x84, 0x0e, 0x90, 0xf0, 0xd2,

0x12, 0x5c, 0x35, 0x42, 0x07, 0x48, 0x78, 0x69,

0x09, 0x2e, 0x94, 0x21, 0x8d, 0x24, 0x3c, 0xba,

0x8a, 0x17, 0x4a, 0x9e, 0xc8, 0x12, 0x1e, 0x5d,

0x45, 0x85, 0x25, 0x4f, 0x64, 0x09, 0x0f, 0xa0,

0xac, 0xcc, 0x9c, 0xa9, 0x32, 0x8a, 0x89, 0x50,

0x9d, 0x4d, 0xf0, 0x5d, 0x5f, 0x66, 0x14, 0x51,

0xc0, 0xa8, 0x78, 0xa0, 0xa1, 0x33, 0x0a, 0xa6,

0x60, 0x54, 0x3c, 0x50, 0xde, 0x97, 0x05, 0x53,

0x30, 0x2a, 0x1e, 0x28, 0x6f, 0xc5, 0x8c, 0xa7,

0x18, 0x15, 0x0f, 0x14, 0xb9, 0xec, 0x46, 0xdd,

0x0c, 0x84, 0x89, 0x0a, 0xd2, 0x76, 0x23, 0xe0,

0x06, 0x42, 0xca, 0x05, 0x69, 0x3b, 0x9f, 0x70,

0x03, 0x21, 0x65, 0x8c, 0xba, 0x93, 0xc1, 0x38,

0x86, 0x27, 0x5d, 0xf0, 0x9c, 0xe8, 0xaa, 0xa8,

0x43, 0x9d, 0xa0, 0x78, 0x4e, 0x74, 0x55, 0x54,

0xaf, 0xc0, 0x50, 0x3c, 0x27, 0x3a, 0xa4, 0x2a,

0xd9, 0x60, 0x28, 0x1e, 0x9d, 0x1d, 0x52, 0x15,

0xe2, 0x30, 0x14, 0x0f, 0xc0, 0x80, 0x29, 0x84,

0x71, 0x18, 0x0a, 0x89, 0x60, 0x40, 0x9a, 0x42,

0xb6, 0x0c, 0x05, 0xca, 0x30, 0x20, 0x4d, 0x21,

0x5b, 0x06, 0x8c, 0x65, 0x18, 0x10, 0xa8, 0x9e,

0x45, 0x6c, 0x34, 0x88, 0x7a, 0x38, 0x05, 0xb9,

0xac, 0x36, 0x1a, 0x44, 0x3d, 0x1c, 0x8c, 0xd2,

0x56, 0x1b, 0x0d, 0x22, 0x90, 0x0e, 0x46, 0x69,

0x2b, 0x83, 0x88, 0x11, 0x48, 0x07, 0x23, 0xba,

0x9b, 0xcf, 0x44, 0x86, 0x24, 0x8d, 0x9f, 0x5d,

0xc3, 0xe9, 0x22, 0x43, 0x12, 0xc8, 0xc1, 0xa0,

0xef, 0xfa, 0x11, 0xaf, 0x09, 0x64, 0xee, 0x50,

0xf9, 0x7d, 0x86, 0xd9, 0x8a, 0x32, 0x77, 0x28,

0xe4, 0xfa, 0x20, 0x54, 0xa8, 0x0b, 0x32, 0x9c,

0x72, 0x7d, 0x10, 0x2a, 0x54, 0x8b, 0x19, 0x4e,

0x39, 0xb0, 0x08, 0x15, 0x2a, 0xcb, 0x82, 0x27,

0x92, 0x58, 0x04, 0x84, 0x15, 0xeb, 0x41, 0x9d,

0x49, 0x2c, 0x02, 0x42, 0x84, 0xfb, 0xae, 0xc0,

0xaa, 0x16, 0x01, 0x21, 0x42, 0xf3, 0x57, 0x60,

0x55, 0x0b, 0x8e, 0x9e, 0x21, 0xf7, 0xa5, 0x30,

0xa4, 0x8b, 0x47, 0x4f, 0x9e, 0xf5, 0xdc, 0x18,

0x70, 0xa6, 0xa5, 0x6e, 0x24, 0x40, 0x59, 0x8e,

0x38, 0x53, 0xdc, 0x37, 0x12, 0x20, 0xa2, 0x47,

0x1c, 0xa7, 0x6e, 0x95, 0x09, 0x10, 0x51, 0xad,

0x0e, 0xdd, 0x37, 0xc4, 0x8a, 0x08, 0xa6, 0xd8,

0x07, 0xe0, 0x95, 0x62, 0x45, 0x04, 0x53, 0x6c,

0x8d, 0x70, 0xc4, 0x31, 0xac, 0x02, 0xa7, 0x36,

0xc8, 0x38, 0x62, 0x96, 0x56, 0x01, 0xdd, 0x1b,

0x64, 0x1c, 0x31, 0x4b, 0x2b, 0x8e, 0xe0, 0x83

};

private static byte[] C =

{

0xb1, 0x08, 0x5b, 0xda, 0x1e, 0xca, 0xda, 0xe9, 0xeb, 0xcb, 0x2f, 0x81, 0xc0, 0x65, 0x7c, 0x1f,

0x2f, 0x6a, 0x76, 0x43, 0x2e, 0x45, 0xd0, 0x16, 0x71, 0x4e, 0xb8, 0x8d, 0x75, 0x85, 0xc4, 0xfc,

0x4b, 0x7c, 0xe0, 0x91, 0x92, 0x67, 0x69, 0x01, 0xa2, 0x42, 0x2a, 0x08, 0xa4, 0x60, 0xd3, 0x15,

0x05, 0x76, 0x74, 0x36, 0xcc, 0x74, 0x4d, 0x23, 0xdd, 0x80, 0x65, 0x59, 0xf2, 0xa6, 0x45, 0x07,

0x6f, 0xa3, 0xb5, 0x8a, 0xa9, 0x9d, 0x2f, 0x1a, 0x4f, 0xe3, 0x9d, 0x46, 0x0f, 0x70, 0xb5, 0xd7,

0xf3, 0xfe, 0xea, 0x72, 0x0a, 0x23, 0x2b, 0x98, 0x61, 0xd5, 0x5e, 0x0f, 0x16, 0xb5, 0x01, 0x31,

0x9a, 0xb5, 0x17, 0x6b, 0x12, 0xd6, 0x99, 0x58, 0x5c, 0xb5, 0x61, 0xc2, 0xdb, 0x0a, 0xa7, 0xca,

0x55, 0xdd, 0xa2, 0x1b, 0xd7, 0xcb, 0xcd, 0x56, 0xe6, 0x79, 0x04, 0x70, 0x21, 0xb1, 0x9b, 0xb7,

0xf5, 0x74, 0xdc, 0xac, 0x2b, 0xce, 0x2f, 0xc7, 0x0a, 0x39, 0xfc, 0x28, 0x6a, 0x3d, 0x84, 0x35,

0x06, 0xf1, 0x5e, 0x5f, 0x52, 0x9c, 0x1f, 0x8b, 0xf2, 0xea, 0x75, 0x14, 0xb1, 0x29, 0x7b, 0x7b,

0xd3, 0xe2, 0x0f, 0xe4, 0x90, 0x35, 0x9e, 0xb1, 0xc1, 0xc9, 0x3a, 0x37, 0x60, 0x62, 0xdb, 0x09,

0xc2, 0xb6, 0xf4, 0x43, 0x86, 0x7a, 0xdb, 0x31, 0x99, 0x1e, 0x96, 0xf5, 0x0a, 0xba, 0x0a, 0xb2,

0xef, 0x1f, 0xdf, 0xb3, 0xe8, 0x15, 0x66, 0xd2, 0xf9, 0x48, 0xe1, 0xa0, 0x5d, 0x71, 0xe4, 0xdd,

0x48, 0x8e, 0x85, 0x7e, 0x33, 0x5c, 0x3c, 0x7d, 0x9d, 0x72, 0x1c, 0xad, 0x68, 0x5e, 0x35, 0x3f,

0xa9, 0xd7, 0x2c, 0x82, 0xed, 0x03, 0xd6, 0x75, 0xd8, 0xb7, 0x13, 0x33, 0x93, 0x52, 0x03, 0xbe,

0x34, 0x53, 0xea, 0xa1, 0x93, 0xe8, 0x37, 0xf1, 0x22, 0x0c, 0xbe, 0xbc, 0x84, 0xe3, 0xd1, 0x2e,

0x4b, 0xea, 0x6b, 0xac, 0xad, 0x47, 0x47, 0x99, 0x9a, 0x3f, 0x41, 0x0c, 0x6c, 0xa9, 0x23, 0x63,

0x7f, 0x15, 0x1c, 0x1f, 0x16, 0x86, 0x10, 0x4a, 0x35, 0x9e, 0x35, 0xd7, 0x80, 0x0f, 0xff, 0xbd,

0xbf, 0xcd, 0x17, 0x47, 0x25, 0x3a, 0xf5, 0xa3, 0xdf, 0xff, 0x00, 0xb7, 0x23, 0x27, 0x1a, 0x16,

0x7a, 0x56, 0xa2, 0x7e, 0xa9, 0xea, 0x63, 0xf5, 0x60, 0x17, 0x58, 0xfd, 0x7c, 0x6c, 0xfe, 0x57,

0xae, 0x4f, 0xae, 0xae, 0x1d, 0x3a, 0xd3, 0xd9, 0x6f, 0xa4, 0xc3, 0x3b, 0x7a, 0x30, 0x39, 0xc0,

0x2d, 0x66, 0xc4, 0xf9, 0x51, 0x42, 0xa4, 0x6c, 0x18, 0x7f, 0x9a, 0xb4, 0x9a, 0xf0, 0x8e, 0xc6,

0xcf, 0xfa, 0xa6, 0xb7, 0x1c, 0x9a, 0xb7, 0xb4, 0x0a, 0xf2, 0x1f, 0x66, 0xc2, 0xbe, 0xc6, 0xb6,

0xbf, 0x71, 0xc5, 0x72, 0x36, 0x90, 0x4f, 0x35, 0xfa, 0x68, 0x40, 0x7a, 0x46, 0x64, 0x7d, 0x6e,

0xf4, 0xc7, 0x0e, 0x16, 0xee, 0xaa, 0xc5, 0xec, 0x51, 0xac, 0x86, 0xfe, 0xbf, 0x24, 0x09, 0x54,

0x39, 0x9e, 0xc6, 0xc7, 0xe6, 0xbf, 0x87, 0xc9, 0xd3, 0x47, 0x3e, 0x33, 0x19, 0x7a, 0x93, 0xc9,

0x09, 0x92, 0xab, 0xc5, 0x2d, 0x82, 0x2c, 0x37, 0x06, 0x47, 0x69, 0x83, 0x28, 0x4a, 0x05, 0x04,

0x35, 0x17, 0x45, 0x4c, 0xa2, 0x3c, 0x4a, 0xf3, 0x88, 0x86, 0x56, 0x4d, 0x3a, 0x14, 0xd4, 0x93,

0x9b, 0x1f, 0x5b, 0x42, 0x4d, 0x93, 0xc9, 0xa7, 0x03, 0xe7, 0xaa, 0x02, 0x0c, 0x6e, 0x41, 0x41,

0x4e, 0xb7, 0xf8, 0x71, 0x9c, 0x36, 0xde, 0x1e, 0x89, 0xb4, 0x44, 0x3b, 0x4d, 0xdb, 0xc4, 0x9a,

0xf4, 0x89, 0x2b, 0xcb, 0x92, 0x9b, 0x06, 0x90, 0x69, 0xd1, 0x8d, 0x2b, 0xd1, 0xa5, 0xc4, 0x2f,

0x36, 0xac, 0xc2, 0x35, 0x59, 0x51, 0xa8, 0xd9, 0xa4, 0x7f, 0x0d, 0xd4, 0xbf, 0x02, 0xe7, 0x1e,

0x37, 0x8f, 0x5a, 0x54, 0x16, 0x31, 0x22, 0x9b, 0x94, 0x4c, 0x9a, 0xd8, 0xec, 0x16, 0x5f, 0xde,

0x3a, 0x7d, 0x3a, 0x1b, 0x25, 0x89, 0x42, 0x24, 0x3c, 0xd9, 0x55, 0xb7, 0xe0, 0x0d, 0x09, 0x84,

0x80, 0x0a, 0x44, 0x0b, 0xdb, 0xb2, 0xce, 0xb1, 0x7b, 0x2b, 0x8a, 0x9a, 0xa6, 0x07, 0x9c, 0x54,

0x0e, 0x38, 0xdc, 0x92, 0xcb, 0x1f, 0x2a, 0x60, 0x72, 0x61, 0x44, 0x51, 0x83, 0x23, 0x5a, 0xdb,

0xab, 0xbe, 0xde, 0xa6, 0x80, 0x05, 0x6f, 0x52, 0x38, 0x2a, 0xe5, 0x48, 0xb2, 0xe4, 0xf3, 0xf3,

0x89, 0x41, 0xe7, 0x1c, 0xff, 0x8a, 0x78, 0xdb, 0x1f, 0xff, 0xe1, 0x8a, 0x1b, 0x33, 0x61, 0x03,

0x9f, 0xe7, 0x67, 0x02, 0xaf, 0x69, 0x33, 0x4b, 0x7a, 0x1e, 0x6c, 0x30, 0x3b, 0x76, 0x52, 0xf4,

0x36, 0x98, 0xfa, 0xd1, 0x15, 0x3b, 0xb6, 0xc3, 0x74, 0xb4, 0xc7, 0xfb, 0x98, 0x45, 0x9c, 0xed,

0x7b, 0xcd, 0x9e, 0xd0, 0xef, 0xc8, 0x89, 0xfb, 0x30, 0x02, 0xc6, 0xcd, 0x63, 0x5a, 0xfe, 0x94,

0xd8, 0xfa, 0x6b, 0xbb, 0xeb, 0xab, 0x07, 0x61, 0x20, 0x01, 0x80, 0x21, 0x14, 0x84, 0x66, 0x79,

0x8a, 0x1d, 0x71, 0xef, 0xea, 0x48, 0xb9, 0xca, 0xef, 0xba, 0xcd, 0x1d, 0x7d, 0x47, 0x6e, 0x98,

0xde, 0xa2, 0x59, 0x4a, 0xc0, 0x6f, 0xd8, 0x5d, 0x6b, 0xca, 0xa4, 0xcd, 0x81, 0xf3, 0x2d, 0x1b,

0x37, 0x8e, 0xe7, 0x67, 0xf1, 0x16, 0x31, 0xba, 0xd2, 0x13, 0x80, 0xb0, 0x04, 0x49, 0xb1, 0x7a,

0xcd, 0xa4, 0x3c, 0x32, 0xbc, 0xdf, 0x1d, 0x77, 0xf8, 0x20, 0x12, 0xd4, 0x30, 0x21, 0x9f, 0x9b,

0x5d, 0x80, 0xef, 0x9d, 0x18, 0x91, 0xcc, 0x86, 0xe7, 0x1d, 0xa4, 0xaa, 0x88, 0xe1, 0x28, 0x52,

0xfa, 0xf4, 0x17, 0xd5, 0xd9, 0xb2, 0x1b, 0x99, 0x48, 0xbc, 0x92, 0x4a, 0xf1, 0x1b, 0xd7, 0x20

};

private static byte[] ModularAddition(byte[] a, byte[] b)

{

var result = new byte[64];

var sum = 0;

for (var i = 0; i < 64; i++)

{

sum = a[i] + b[i] + (sum >> 8);

result[i] = (byte)(sum & 0xff);

}

return result;

}

private static byte[] XOR(byte[] a, byte[] b)

{

var result = new byte[64];

for (var i = 0; i < 64; i++)

{

result[i] = (byte)(a[i] ^ b[i]);

}

return result;

}

private static byte[] S(byte[] a)

{

return a.Reverse().Select(v => Pi[v]).ToArray();

}

private static byte[] P(byte[] a)

{

return t.Reverse().Select(elem => a[elem]).ToArray();

}

private static byte[] L(byte[] a)

{

byte[] result = new byte[64];

int i, j, k, n, p;

for (i = 7; i >= 0; i--)

{

for (n = 0; n < 8; n++)

{

p = 63;

for (j = 7; j >= 0; j--)

{

for (k = 0; k < 8; k++)

{

if (((a[i \* 8 + j] >> k) & 1) != 0)

{

result[i \* 8 + n] ^= l[p \* 8 + n];

}

p--;

}

}

}

}

return result;

}

private static byte[] LPS(byte[] a)

{

var result = S(a);

result = P(result);

result = L(result);

return result;

}

private static byte[] E(byte[] K, byte[] m)

{

var result = XOR(K, m);

for (var i = 0; i < 12; i++)

{

result = LPS(result);

K = XOR(K, C.Skip(i \* 64).Take(64).ToArray());

K = LPS(K);

result = XOR(K, result);

}

return result;

}

private static byte[] gN(byte[] N, byte[] h, byte[] m)

{

var result = XOR(h, N);

result = LPS(result);

result = E(result, m);

result = XOR(result, h);

result = XOR(result, m);

return result;

}

public static byte[] Hash(byte[] M)

{

var h = new byte[64];

var N = new byte[64];

var E = new byte[64];

var v\_512 = new byte[64];

v\_512[1] = 0x02;

byte[] m;

while (M.Length >= 64)

{

m = M.Skip(M.Length - 64).Take(64).ToArray();

h = gN(N, h, m);

N = ModularAddition(N, v\_512);

E = ModularAddition(E, m);

M = M.SkipLast(64).ToArray();

}

m = new byte[64 - M.Length];

m[^1] = 0x01;

m = m.Concat(M).ToArray();

h = gN(N, h, m);

var v\_60 = new byte[60];

N = ModularAddition(N, v\_60.Concat(BitConverter.GetBytes(M.Length)).Reverse().ToArray());

E = ModularAddition(E, m);

h = gN(new byte[64], h, N);

h = gN(new byte[64], h, E);

return h;

}

}

public static class GOST3411

{

private static byte[] Pi =

{

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

};

private static byte[] t =

{

0, 8, 16, 24, 32, 40, 48, 56,

1, 9, 17, 25, 33, 41, 49, 57,

2, 10, 18, 26, 34, 42, 50, 58,

3, 11, 19, 27, 35, 43, 51, 59,

4, 12, 20, 28, 36, 44, 52, 60,

5, 13, 21, 29, 37, 45, 53, 61,

6, 14, 22, 30, 38, 46, 54, 62,

7, 15, 23, 31, 39, 47, 55, 63

};

private static byte[] l =

{

0x8e, 0x20, 0xfa, 0xa7, 0x2b, 0xa0, 0xb4, 0x70,

0x47, 0x10, 0x7d, 0xdd, 0x9b, 0x50, 0x5a, 0x38,

0xad, 0x08, 0xb0, 0xe0, 0xc3, 0x28, 0x2d, 0x1c,

0xd8, 0x04, 0x58, 0x70, 0xef, 0x14, 0x98, 0x0e,

0x6c, 0x02, 0x2c, 0x38, 0xf9, 0x0a, 0x4c, 0x07,

0x36, 0x01, 0x16, 0x1c, 0xf2, 0x05, 0x26, 0x8d,

0x1b, 0x8e, 0x0b, 0x0e, 0x79, 0x8c, 0x13, 0xc8,

0x83, 0x47, 0x8b, 0x07, 0xb2, 0x46, 0x87, 0x64,

0xa0, 0x11, 0xd3, 0x80, 0x81, 0x8e, 0x8f, 0x40,

0x50, 0x86, 0xe7, 0x40, 0xce, 0x47, 0xc9, 0x20,

0x28, 0x43, 0xfd, 0x20, 0x67, 0xad, 0xea, 0x10,

0x14, 0xaf, 0xf0, 0x10, 0xbd, 0xd8, 0x75, 0x08,

0x0a, 0xd9, 0x78, 0x08, 0xd0, 0x6c, 0xb4, 0x04,

0x05, 0xe2, 0x3c, 0x04, 0x68, 0x36, 0x5a, 0x02,

0x8c, 0x71, 0x1e, 0x02, 0x34, 0x1b, 0x2d, 0x01,

0x46, 0xb6, 0x0f, 0x01, 0x1a, 0x83, 0x98, 0x8e,

0x90, 0xda, 0xb5, 0x2a, 0x38, 0x7a, 0xe7, 0x6f,

0x48, 0x6d, 0xd4, 0x15, 0x1c, 0x3d, 0xfd, 0xb9,

0x24, 0xb8, 0x6a, 0x84, 0x0e, 0x90, 0xf0, 0xd2,

0x12, 0x5c, 0x35, 0x42, 0x07, 0x48, 0x78, 0x69,

0x09, 0x2e, 0x94, 0x21, 0x8d, 0x24, 0x3c, 0xba,

0x8a, 0x17, 0x4a, 0x9e, 0xc8, 0x12, 0x1e, 0x5d,

0x45, 0x85, 0x25, 0x4f, 0x64, 0x09, 0x0f, 0xa0,

0xac, 0xcc, 0x9c, 0xa9, 0x32, 0x8a, 0x89, 0x50,

0x9d, 0x4d, 0xf0, 0x5d, 0x5f, 0x66, 0x14, 0x51,

0xc0, 0xa8, 0x78, 0xa0, 0xa1, 0x33, 0x0a, 0xa6,

0x60, 0x54, 0x3c, 0x50, 0xde, 0x97, 0x05, 0x53,

0x30, 0x2a, 0x1e, 0x28, 0x6f, 0xc5, 0x8c, 0xa7,

0x18, 0x15, 0x0f, 0x14, 0xb9, 0xec, 0x46, 0xdd,

0x0c, 0x84, 0x89, 0x0a, 0xd2, 0x76, 0x23, 0xe0,

0x06, 0x42, 0xca, 0x05, 0x69, 0x3b, 0x9f, 0x70,

0x03, 0x21, 0x65, 0x8c, 0xba, 0x93, 0xc1, 0x38,

0x86, 0x27, 0x5d, 0xf0, 0x9c, 0xe8, 0xaa, 0xa8,

0x43, 0x9d, 0xa0, 0x78, 0x4e, 0x74, 0x55, 0x54,

0xaf, 0xc0, 0x50, 0x3c, 0x27, 0x3a, 0xa4, 0x2a,

0xd9, 0x60, 0x28, 0x1e, 0x9d, 0x1d, 0x52, 0x15,

0xe2, 0x30, 0x14, 0x0f, 0xc0, 0x80, 0x29, 0x84,

0x71, 0x18, 0x0a, 0x89, 0x60, 0x40, 0x9a, 0x42,

0xb6, 0x0c, 0x05, 0xca, 0x30, 0x20, 0x4d, 0x21,

0x5b, 0x06, 0x8c, 0x65, 0x18, 0x10, 0xa8, 0x9e,

0x45, 0x6c, 0x34, 0x88, 0x7a, 0x38, 0x05, 0xb9,

0xac, 0x36, 0x1a, 0x44, 0x3d, 0x1c, 0x8c, 0xd2,

0x56, 0x1b, 0x0d, 0x22, 0x90, 0x0e, 0x46, 0x69,

0x2b, 0x83, 0x88, 0x11, 0x48, 0x07, 0x23, 0xba,

0x9b, 0xcf, 0x44, 0x86, 0x24, 0x8d, 0x9f, 0x5d,

0xc3, 0xe9, 0x22, 0x43, 0x12, 0xc8, 0xc1, 0xa0,

0xef, 0xfa, 0x11, 0xaf, 0x09, 0x64, 0xee, 0x50,

0xf9, 0x7d, 0x86, 0xd9, 0x8a, 0x32, 0x77, 0x28,

0xe4, 0xfa, 0x20, 0x54, 0xa8, 0x0b, 0x32, 0x9c,

0x72, 0x7d, 0x10, 0x2a, 0x54, 0x8b, 0x19, 0x4e,

0x39, 0xb0, 0x08, 0x15, 0x2a, 0xcb, 0x82, 0x27,

0x92, 0x58, 0x04, 0x84, 0x15, 0xeb, 0x41, 0x9d,

0x49, 0x2c, 0x02, 0x42, 0x84, 0xfb, 0xae, 0xc0,

0xaa, 0x16, 0x01, 0x21, 0x42, 0xf3, 0x57, 0x60,

0x55, 0x0b, 0x8e, 0x9e, 0x21, 0xf7, 0xa5, 0x30,

0xa4, 0x8b, 0x47, 0x4f, 0x9e, 0xf5, 0xdc, 0x18,

0x70, 0xa6, 0xa5, 0x6e, 0x24, 0x40, 0x59, 0x8e,

0x38, 0x53, 0xdc, 0x37, 0x12, 0x20, 0xa2, 0x47,

0x1c, 0xa7, 0x6e, 0x95, 0x09, 0x10, 0x51, 0xad,

0x0e, 0xdd, 0x37, 0xc4, 0x8a, 0x08, 0xa6, 0xd8,

0x07, 0xe0, 0x95, 0x62, 0x45, 0x04, 0x53, 0x6c,

0x8d, 0x70, 0xc4, 0x31, 0xac, 0x02, 0xa7, 0x36,

0xc8, 0x38, 0x62, 0x96, 0x56, 0x01, 0xdd, 0x1b,

0x64, 0x1c, 0x31, 0x4b, 0x2b, 0x8e, 0xe0, 0x83

};

private static byte[] C =

{

0xb1, 0x08, 0x5b, 0xda, 0x1e, 0xca, 0xda, 0xe9, 0xeb, 0xcb, 0x2f, 0x81, 0xc0, 0x65, 0x7c, 0x1f,

0x2f, 0x6a, 0x76, 0x43, 0x2e, 0x45, 0xd0, 0x16, 0x71, 0x4e, 0xb8, 0x8d, 0x75, 0x85, 0xc4, 0xfc,

0x4b, 0x7c, 0xe0, 0x91, 0x92, 0x67, 0x69, 0x01, 0xa2, 0x42, 0x2a, 0x08, 0xa4, 0x60, 0xd3, 0x15,

0x05, 0x76, 0x74, 0x36, 0xcc, 0x74, 0x4d, 0x23, 0xdd, 0x80, 0x65, 0x59, 0xf2, 0xa6, 0x45, 0x07,

0x6f, 0xa3, 0xb5, 0x8a, 0xa9, 0x9d, 0x2f, 0x1a, 0x4f, 0xe3, 0x9d, 0x46, 0x0f, 0x70, 0xb5, 0xd7,

0xf3, 0xfe, 0xea, 0x72, 0x0a, 0x23, 0x2b, 0x98, 0x61, 0xd5, 0x5e, 0x0f, 0x16, 0xb5, 0x01, 0x31,

0x9a, 0xb5, 0x17, 0x6b, 0x12, 0xd6, 0x99, 0x58, 0x5c, 0xb5, 0x61, 0xc2, 0xdb, 0x0a, 0xa7, 0xca,

0x55, 0xdd, 0xa2, 0x1b, 0xd7, 0xcb, 0xcd, 0x56, 0xe6, 0x79, 0x04, 0x70, 0x21, 0xb1, 0x9b, 0xb7,

0xf5, 0x74, 0xdc, 0xac, 0x2b, 0xce, 0x2f, 0xc7, 0x0a, 0x39, 0xfc, 0x28, 0x6a, 0x3d, 0x84, 0x35,

0x06, 0xf1, 0x5e, 0x5f, 0x52, 0x9c, 0x1f, 0x8b, 0xf2, 0xea, 0x75, 0x14, 0xb1, 0x29, 0x7b, 0x7b,

0xd3, 0xe2, 0x0f, 0xe4, 0x90, 0x35, 0x9e, 0xb1, 0xc1, 0xc9, 0x3a, 0x37, 0x60, 0x62, 0xdb, 0x09,

0xc2, 0xb6, 0xf4, 0x43, 0x86, 0x7a, 0xdb, 0x31, 0x99, 0x1e, 0x96, 0xf5, 0x0a, 0xba, 0x0a, 0xb2,

0xef, 0x1f, 0xdf, 0xb3, 0xe8, 0x15, 0x66, 0xd2, 0xf9, 0x48, 0xe1, 0xa0, 0x5d, 0x71, 0xe4, 0xdd,

0x48, 0x8e, 0x85, 0x7e, 0x33, 0x5c, 0x3c, 0x7d, 0x9d, 0x72, 0x1c, 0xad, 0x68, 0x5e, 0x35, 0x3f,

0xa9, 0xd7, 0x2c, 0x82, 0xed, 0x03, 0xd6, 0x75, 0xd8, 0xb7, 0x13, 0x33, 0x93, 0x52, 0x03, 0xbe,

0x34, 0x53, 0xea, 0xa1, 0x93, 0xe8, 0x37, 0xf1, 0x22, 0x0c, 0xbe, 0xbc, 0x84, 0xe3, 0xd1, 0x2e,

0x4b, 0xea, 0x6b, 0xac, 0xad, 0x47, 0x47, 0x99, 0x9a, 0x3f, 0x41, 0x0c, 0x6c, 0xa9, 0x23, 0x63,

0x7f, 0x15, 0x1c, 0x1f, 0x16, 0x86, 0x10, 0x4a, 0x35, 0x9e, 0x35, 0xd7, 0x80, 0x0f, 0xff, 0xbd,

0xbf, 0xcd, 0x17, 0x47, 0x25, 0x3a, 0xf5, 0xa3, 0xdf, 0xff, 0x00, 0xb7, 0x23, 0x27, 0x1a, 0x16,

0x7a, 0x56, 0xa2, 0x7e, 0xa9, 0xea, 0x63, 0xf5, 0x60, 0x17, 0x58, 0xfd, 0x7c, 0x6c, 0xfe, 0x57,

0xae, 0x4f, 0xae, 0xae, 0x1d, 0x3a, 0xd3, 0xd9, 0x6f, 0xa4, 0xc3, 0x3b, 0x7a, 0x30, 0x39, 0xc0,

0x2d, 0x66, 0xc4, 0xf9, 0x51, 0x42, 0xa4, 0x6c, 0x18, 0x7f, 0x9a, 0xb4, 0x9a, 0xf0, 0x8e, 0xc6,

0xcf, 0xfa, 0xa6, 0xb7, 0x1c, 0x9a, 0xb7, 0xb4, 0x0a, 0xf2, 0x1f, 0x66, 0xc2, 0xbe, 0xc6, 0xb6,

0xbf, 0x71, 0xc5, 0x72, 0x36, 0x90, 0x4f, 0x35, 0xfa, 0x68, 0x40, 0x7a, 0x46, 0x64, 0x7d, 0x6e,

0xf4, 0xc7, 0x0e, 0x16, 0xee, 0xaa, 0xc5, 0xec, 0x51, 0xac, 0x86, 0xfe, 0xbf, 0x24, 0x09, 0x54,

0x39, 0x9e, 0xc6, 0xc7, 0xe6, 0xbf, 0x87, 0xc9, 0xd3, 0x47, 0x3e, 0x33, 0x19, 0x7a, 0x93, 0xc9,

0x09, 0x92, 0xab, 0xc5, 0x2d, 0x82, 0x2c, 0x37, 0x06, 0x47, 0x69, 0x83, 0x28, 0x4a, 0x05, 0x04,

0x35, 0x17, 0x45, 0x4c, 0xa2, 0x3c, 0x4a, 0xf3, 0x88, 0x86, 0x56, 0x4d, 0x3a, 0x14, 0xd4, 0x93,

0x9b, 0x1f, 0x5b, 0x42, 0x4d, 0x93, 0xc9, 0xa7, 0x03, 0xe7, 0xaa, 0x02, 0x0c, 0x6e, 0x41, 0x41,

0x4e, 0xb7, 0xf8, 0x71, 0x9c, 0x36, 0xde, 0x1e, 0x89, 0xb4, 0x44, 0x3b, 0x4d, 0xdb, 0xc4, 0x9a,

0xf4, 0x89, 0x2b, 0xcb, 0x92, 0x9b, 0x06, 0x90, 0x69, 0xd1, 0x8d, 0x2b, 0xd1, 0xa5, 0xc4, 0x2f,

0x36, 0xac, 0xc2, 0x35, 0x59, 0x51, 0xa8, 0xd9, 0xa4, 0x7f, 0x0d, 0xd4, 0xbf, 0x02, 0xe7, 0x1e,

0x37, 0x8f, 0x5a, 0x54, 0x16, 0x31, 0x22, 0x9b, 0x94, 0x4c, 0x9a, 0xd8, 0xec, 0x16, 0x5f, 0xde,

0x3a, 0x7d, 0x3a, 0x1b, 0x25, 0x89, 0x42, 0x24, 0x3c, 0xd9, 0x55, 0xb7, 0xe0, 0x0d, 0x09, 0x84,

0x80, 0x0a, 0x44, 0x0b, 0xdb, 0xb2, 0xce, 0xb1, 0x7b, 0x2b, 0x8a, 0x9a, 0xa6, 0x07, 0x9c, 0x54,

0x0e, 0x38, 0xdc, 0x92, 0xcb, 0x1f, 0x2a, 0x60, 0x72, 0x61, 0x44, 0x51, 0x83, 0x23, 0x5a, 0xdb,

0xab, 0xbe, 0xde, 0xa6, 0x80, 0x05, 0x6f, 0x52, 0x38, 0x2a, 0xe5, 0x48, 0xb2, 0xe4, 0xf3, 0xf3,

0x89, 0x41, 0xe7, 0x1c, 0xff, 0x8a, 0x78, 0xdb, 0x1f, 0xff, 0xe1, 0x8a, 0x1b, 0x33, 0x61, 0x03,

0x9f, 0xe7, 0x67, 0x02, 0xaf, 0x69, 0x33, 0x4b, 0x7a, 0x1e, 0x6c, 0x30, 0x3b, 0x76, 0x52, 0xf4,

0x36, 0x98, 0xfa, 0xd1, 0x15, 0x3b, 0xb6, 0xc3, 0x74, 0xb4, 0xc7, 0xfb, 0x98, 0x45, 0x9c, 0xed,

0x7b, 0xcd, 0x9e, 0xd0, 0xef, 0xc8, 0x89, 0xfb, 0x30, 0x02, 0xc6, 0xcd, 0x63, 0x5a, 0xfe, 0x94,

0xd8, 0xfa, 0x6b, 0xbb, 0xeb, 0xab, 0x07, 0x61, 0x20, 0x01, 0x80, 0x21, 0x14, 0x84, 0x66, 0x79,

0x8a, 0x1d, 0x71, 0xef, 0xea, 0x48, 0xb9, 0xca, 0xef, 0xba, 0xcd, 0x1d, 0x7d, 0x47, 0x6e, 0x98,

0xde, 0xa2, 0x59, 0x4a, 0xc0, 0x6f, 0xd8, 0x5d, 0x6b, 0xca, 0xa4, 0xcd, 0x81, 0xf3, 0x2d, 0x1b,

0x37, 0x8e, 0xe7, 0x67, 0xf1, 0x16, 0x31, 0xba, 0xd2, 0x13, 0x80, 0xb0, 0x04, 0x49, 0xb1, 0x7a,

0xcd, 0xa4, 0x3c, 0x32, 0xbc, 0xdf, 0x1d, 0x77, 0xf8, 0x20, 0x12, 0xd4, 0x30, 0x21, 0x9f, 0x9b,

0x5d, 0x80, 0xef, 0x9d, 0x18, 0x91, 0xcc, 0x86, 0xe7, 0x1d, 0xa4, 0xaa, 0x88, 0xe1, 0x28, 0x52,

0xfa, 0xf4, 0x17, 0xd5, 0xd9, 0xb2, 0x1b, 0x99, 0x48, 0xbc, 0x92, 0x4a, 0xf1, 0x1b, 0xd7, 0x20

};

private static byte[] ModularAddition(byte[] a, byte[] b)

{

var result = new byte[64];

var sum = 0;

for (var i = 0; i < 64; i++)

{

sum = a[i] + b[i] + (sum >> 8);

result[i] = (byte)(sum & 0xff);

}

return result;

}

private static byte[] XOR(byte[] a, byte[] b)

{

var result = new byte[64];

for (var i = 0; i < 64; i++)

{

result[i] = (byte)(a[i] ^ b[i]);

}

return result;

}

private static byte[] S(byte[] a)

{

return a.Reverse().Select(v => Pi[v]).ToArray();

}

private static byte[] P(byte[] a)

{

return t.Reverse().Select(elem => a[elem]).ToArray();

}

private static byte[] L(byte[] a)

{

byte[] result = new byte[64];

int i, j, k, n, p;

for (i = 7; i >= 0; i--)

{

for (n = 0; n < 8; n++)

{

p = 63;

for (j = 7; j >= 0; j--)

{

for (k = 0; k < 8; k++)

{

if (((a[i \* 8 + j] >> k) & 1) != 0)

{

result[i \* 8 + n] ^= l[p \* 8 + n];

}

p--;

}

}

}

}

return result;

}

private static byte[] LPS(byte[] a)

{

var result = S(a);

result = P(result);

result = L(result);

return result;

}

private static byte[] E(byte[] K, byte[] m)

{

var result = XOR(K, m);

for (var i = 0; i < 12; i++)

{

result = LPS(result);

K = XOR(K, C.Skip(i \* 64).Take(64).ToArray());

K = LPS(K);

result = XOR(K, result);

}

return result;

}

private static byte[] gN(byte[] N, byte[] h, byte[] m)

{

var result = XOR(h, N);

result = LPS(result);

result = E(result, m);

result = XOR(result, h);

result = XOR(result, m);

return result;

}

public static byte[] Hash(byte[] M)

{

var h = new byte[64];

var N = new byte[64];

var E = new byte[64];

var v\_512 = new byte[64];

v\_512[1] = 0x02;

byte[] m;

while (M.Length >= 64)

{

m = M.Skip(M.Length - 64).Take(64).ToArray();

h = gN(N, h, m);

N = ModularAddition(N, v\_512);

E = ModularAddition(E, m);

M = M.SkipLast(64).ToArray();

}

m = new byte[64 - M.Length];

m[^1] = 0x01;

m = m.Concat(M).ToArray();

h = gN(N, h, m);

var v\_60 = new byte[60];

N xaa= ModularAddition(N, v\_60.Concat(BitConverter.GetBytes(M.Length)).Reverse().ToArray());

E = ModularAddition(E, m);

h = gN(new byte[64], h, N);

h = gN(new byte[64], h, E);

return h;

}

}