Цель работы: найти реализацию алгоритма хэширования ГОСТ Р 34.11-2012. Найти второй прообраз и коллизии, а также построить графики зависимости среднего значения сложности второго прообраза и коллизии от количества взятых бит. Оценить полученные графики.

1 Описание алгоритма

1.1 Преобразования

1.1.1 X-преобразование

На вход функции X подаются две последовательности длиной 512 бит каждая, выходом функции является XOR этих последовательностей:

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

1.1.2 S-преобразование

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

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

Значения подстановки записаны ниже в виде массива $\pi' = (\pi(0), \dots, \pi(255))$:

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).

1.1.3 P-преобразование

Функция перестановки. Для каждой пары байт из входной последовательности происходит замена одного байта другим.

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

Значения перестановки τ записаны ниже в виде массива $\tau = (\tau(0), \dots, \tau(63))$:

τ = (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).

1.1.4 L-преобразование

Представляет собой умножение 64-битного входного вектора на бинарную матрицу A размерами 64×64:

8e20faa72ba0b470	47107ddd9b505a38	ad08b0e0c3282d1c	d8045870ef14980e
6c022c38f90a4c07	3601161cf205268d	1b8e0b0e798c13c8	83478b07b2468764
a011d380818e8f40	5086e740ce47c920	2843fd2067adea10	14aff010bdd87508
0ad97808d06cb404	05e23c0468365a02	8c711e02341b2d01	46b60f011a83988e
90dab52a387ae76f	486dd4151c3dfdb9	24b86a840e90f0d2	125c354207487869
092e94218d243cba	8a174a9ec8121e5d	4585254f64090fa0	accc9ca9328a8950
9d4df05d5f661451	c0a878a0a1330aa6	60543c50de970553	302a1e286fc58ca7
18150f14b9ec46dd	0c84890ad27623e0	0642ca05693b9f70	0321658cba93c138
86275df09ce8aaa8	439da0784e745554	afc0503c273aa42a	d960281e9d1d5215
e230140fc0802984	71180a8960409a42	b60c05ca30204d21	5b068c651810a89e
456c34887a3805b9	ac361a443d1c8cd2	561b0d22900e4669	2b838811480723ba
9bcf4486248d9f5d	c3e9224312c8c1a0	effa11af0964ee50	f97d86d98a327728
e4fa2054a80b329c	727d102a548b194e	39b008152acb8227	9258048415eb419d
492c024284fbaec0	aa16012142f35760	550b8e9e21f7a530	a48b474f9ef5dc18
70a6a56e2440598e	3853dc371220a247	1ca76e95091051ad	0edd37c48a08a6d8
07e095624504536c	8d70c431ac02a736	c83862965601dd1b	641c314b2b8ee083

1.2 Функция сжатия

Основным элементом любой хэш-функции является функция сжатия. Функция сжатия g_n в виде алгоритма описывается следующим образом:

Пусть h, N и m – 512-битные последовательности. Для вычисления $g(N, m, h)$ необходимо:

$$K = h \oplus N$$

$$K = S(K)$$

$$K = P(K)$$

$$K = L(K)$$

$$t = E(K, m)$$

$$t = h \oplus t$$

$$G = t \oplus m$$

В качестве результата $g(N, m, h)$ возвращается G.

Вычисление $E(K, m)$ приведено ниже и включает следующие действия:

$$s = K \oplus m$$

Для $i = 0$ до 11 выполнить:

$$s = S(s)$$

$$s = P(s)$$

$$s = L(s)$$

$$K = KeySchedule(K, i)$$

$$s = K \oplus s$$

В качестве результата $E(K, m)$ возвращается s.

Формирование временного ключа K на каждом раунде i функции $E(K, m)$ осуществляется функцией $KeySchedule(K, i)$ и включает следующие вычисления:

$$K = K \oplus C[i]$$

$$K = S(K)$$

$$K = P(K)$$

$$K = L(K)$$

В качестве результата $KeySchedule(K, i)$ возвращается K.

C_1 = b1085bda1ecadae9ebcb2f81c0657c1f2f6a76432e45d016714eb88d7585c4fc
4b7ce09192676901a2422a08a460d31505767436cc744d23dd806559f2a64507;

C_2 = 6fa3b58aa99d2f1a4fe39d460f70b5d7f3feea720a232b9861d55e0f16b50131
9ab5176b12d699585cb561c2db0aa7ca55dda21bd7cbcd56e679047021b19bb7;

C_3 = f574dcac2bce2fc70a39fc286a3d843506f15e5f529c1f8bf2ea7514b1297b7b
d3e20fe490359eb1c1c93a376062db09c2b6f443867adb31991e96f50aba0ab2;

C_4 = ef1fdfb3e81566d2f948e1a05d71e4dd488e857e335c3c7d9d721cad685e353f
a9d72c82ed03d675d8b71333935203be3453eaa193e837f1220cbebc84e3d12e;

C_5 = 4bea6bacad4747999a3f410c6ca923637f151c1f1686104a359e35d7800fffbd
bfcd1747253af5a3dfff00b723271a167a56a27ea9ea63f5601758fd7c6cfe57;

C_6 = ae4faeae1d3ad3d96fa4c33b7a3039c02d66c4f95142a46c187f9ab49af08ec6
cffaa6b71c9ab7b40af21f66c2bec6b6bf71c57236904f35fa68407a46647d6e;

C_7 = f4c70e16eeaac5ec51ac86febf240954399ec6c7e6bf87c9d3473e33197a93c9
0992abc52d822c3706476983284a05043517454ca23c4af38886564d3a14d493;

C_8 = 9b1f5b424d93c9a703e7aa020c6e41414eb7f8719c36de1e89b4443b4ddbc49a
f4892bcb929b069069d18d2bd1a5c42f36acc2355951a8d9a47f0dd4bf02e71e;

C_9 = 378f5a541631229b944c9ad8ec165fde3a7d3a1b258942243cd955b7e00d0984
800a440bdbb2ceb17b2b8a9aa6079c540e38dc92cb1f2a607261445183235adb;

C_{10} = abbedea680056f52382ae548b2e4f3f38941e71cff8a78db1fffe18a1b336103
9fe76702af69334b7a1e6c303b7652f43698fad1153bb6c374b4c7fb98459ced;

C_{11} = 7bcd9ed0efc889fb3002c6cd635afe94d8fa6bbbebab076120018021148466798
a1d71efea48b9caefbacd1d7d476e98dea2594ac06fd85d6bcaa4cd81f32d1b;

C_{12} = 378ee767f11631bad21380b00449b17acda43c32bcdf1d77f82012d430219f9b
5d80ef9d1891cc86e71da4aa88e12852faf417d5d9b21b9948bc924af11bd720.

1.3 Вычисление хэш-функции

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

1) Инициализация значений переменных:

$$h = 0^{64} \text{ для 512 бит}$$

$$h = (00000001)^{64} \text{ для 256 бит}$$

$$N = 0^{512}$$

$$\sum = 0^{512}$$

2) Проверка размера $M < 512$. Если условие выполняется, то переход к пункту 3. Иначе выполнить следующие шаги:

$m -$ последние 512 бит сообщения M

$$h = g(N, m, h)$$

$$N = (N + 512)\ mod\ 2^{512}$$

$$\sum = \left(\sum + m\right)\ mod\ 2^{512}$$

$M -$ без последних 512 бит

Этот шаг повторяется до тех пор, пока $M \geq 512$.

3) Далее сообщение M дополняется до длины 512 по следующему правилу: $m = 0^{511-|M|} \| 1 \| M$, где $|M| -$ длина сообщения в битах.

4) $h = g(N, m, h)$

5) $N = (N + |M|) mod\ 2^{512}$

6) $\sum = (\sum + m)\ mod\ 2^{512}$

7) $h = g(0, h, N)$

8) $h = g(0, h, \sum)$

Для хэш-функции с длиной хэш-кода 512 бит возвращается h в качестве результата. Для хэш-кода 256 бит возвращается $MSB_{256}(h)$.

2 Описание реализации

Необходимо придумать слово-пароль, которое является исходным сообщением x, найти от него $h(x)$, взять от полученного хэша первые 8 бит, обозначив последовательность y_0. Далее случайным образом сгенерировать N сообщений, найти от каждого хэш и взять от каждого хэша первые 8 бит, получив последовательность $y_1, y_2, \dots, y_{N-1}$.

1) Нахождение второго прообраза. Необходимо найти такой y_i, что $y_i = y_0$. Посчитать количество шагов, которое потребовалось, чтобы найти $y_i -$ это сложность второго прообраза.
2) Нахождение коллизий. Необходимо найти в полученной последовательности такие $y_i = y_j$. Посчитать количество шагов, которое потребовалось, чтобы найти эту пару – это сложность коллизии.

Проделать оба шага для последовательности хэшей размером 4, 6, 8, 10 и 12 бит. Необходимо построить графики зависимости среднего значения сложности второго прообраза и коллизии от количества взятых бит.

3 Источник кода

За основу был взят код https://github.com/sftp/gost34.11-2012_stribog.

4 Примеры работы программы

Для начала необходимо проверить, что хэш-код, вычисляемый программой, совпадает с хэш-кодом из стандарта.

В стандарте приведен следующий пример:

Необходимо вычислить хэш-код сообщения

M_1 = 323130393837363534333231303938373635343332313039383736353433323130393837363534333231303938373635343332313039383736353433323130.

Был получен следующий хэш-код длиной 512:

$H(M_1)$ = 486f64c1917879417fef082b3381a4e211c324f074654c38823a7b76f830ad00fa1fbae42b1285c0352f227524bc9ab16254288dd6863dccd5b9f54a1ad0541b.

Был получен следующий хэш-код длиной 256:

Хэш-кодом сообщения M_1 является значение:

$H(M_1)$ = 00557be5e584fd52a449b16b0251d05d27f94ab76cbaa6da890b59d8ef1e159d.

В результате работы программы были получены:

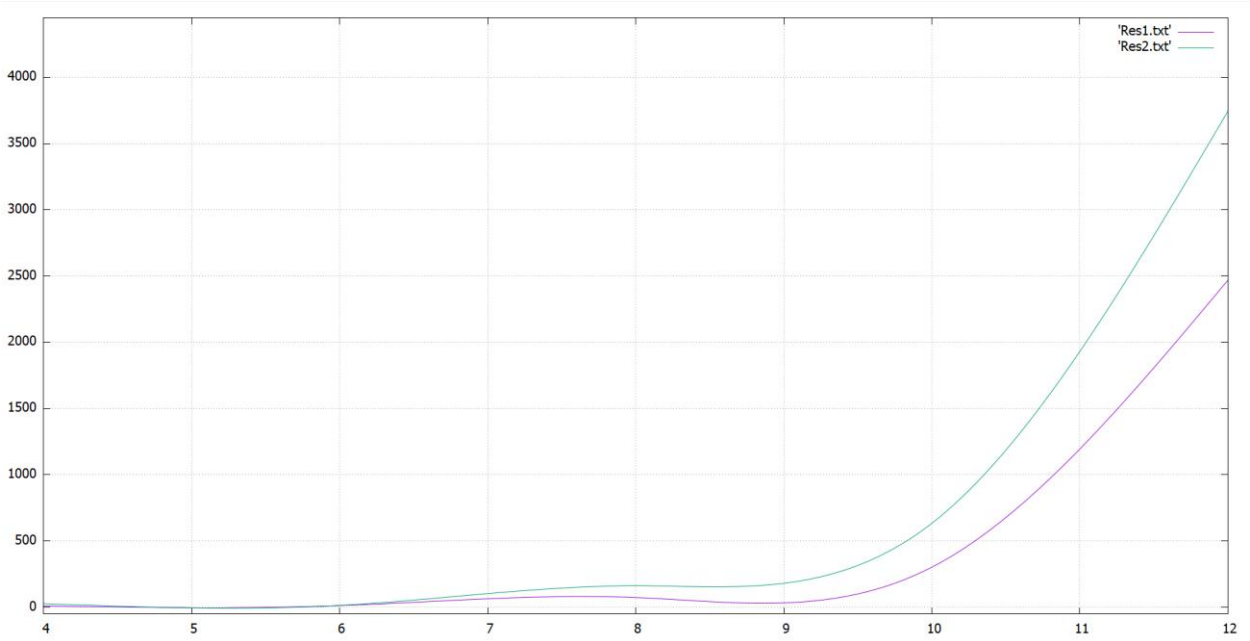
```
H^{512}: 486f64c1917879417fef082b3381a4e211c324f074654c38823a7b76f830ad00fa1fbae42b1285c0352f227524bc9ab16254288dd6863dccd5b9f54a1ad0541b
H^{256}: 00557be5e584fd52a449b16b0251d05d27f94ab76cbaa6da890b59d8ef1e159d
```

Как видно, хэш-коды совпадают.

Далее необходимо построить графики. За слово пароль взято *hello*. Хэш-код длины 256 равен:

```
hello
H^{256}: 6d6dd4bdc59c42e8faa2be16d5dea65b13f28bf964a73b41416ece410a70b03f
```

В результате шагов, описанных ранее шагов были получены следующие графики:



Исходя из графиков, можно сделать вывод, что зависимости сложностей от бит растут прямо пропорционально. Для 12 бит средняя сложность коллизии уже достаточно велика и далее также разрыв будет увеличиваться.

5 Выводы

Настоящий стандарт определяет алгоритм и процедуру вычисления хеш-функции для любой последовательности двоичных символов, которые применяются в криптографических методах обработки и защиты информации, в том числе для реализации процедур обеспечения целостности, аутентичности, электронной цифровой подписи (ЭЦП) при передаче, обработке и хранении информации в автоматизированных системах.

Определенная в настоящем стандарте функция хэширования используется при реализации систем электронной цифровой подписи на базе ассиметричного криптографического алгоритма по ГОСТ Р 34.10–2012.

Список используемых источников:

1) Стандарт ГОСТ Р 34.11 – 2011 [Электронный ресурс] URL: https://yztm.ru/wp-content/uploads/2018/03/gost-34.11-2012.pdf (дата обращения: 20.03.2022)

2) Хэш-функция Стрибог [Электронный ресурс] URL: https://habr.com/ru/post/188152/ (дата обращения: 20.03.2022)