G. K. Sedov, The security of GOST R 34.11-2012 against preimage and collision attacks,
Mat. Vopr. Kriptogr., 2015, Volume 6, Issue 2, 79–98

https://www.mathnet.ru/eng/mvk147

МАТЕМАТИЧЕСКИЕ ВОПРОСЫ КРИПТОГРАФИИ
2015 Т. 6 № 2 С. 79–98

УДК: 519.719.2

Стойкость ГОСТ Р 34.11-2012 к атаке поиска прообраза и к атаке поиска коллизий

Г. К. Седов

Московский государственный университет имени М. В. Ломоносова, Москва

Получено 18.XI.2014

В январе 2013 года национальный стандарт Российской Федерации, ГОСТ Р 34.11-94 определяющий алгоритм и процедуру вычисления хэш-функции, был заменен на новый ГОСТ Р 34.11-2012. В качестве нового стандарта было утверждено семейство хэш-функций Стрибог. В данной работе семейство хэш-функций Стрибог рассмотрено с позиций математической криптографии; доказана его стойкость к атаке поиска прообраза и к атаке поиска коллизий.

Ключевые слова: хэш-функции, математическая криптография, Стрибог, ГОСТ Р 34.11-2012

The security of GOST R 34.11-2012 against preimage and collision attacks

G. K. Sedov

Lomonosov Moscow State University, Moscow

Abstract. In January 2013 the National standard of the Russian Federation GOST R 34.11-94 defining the algorithm and computational procedure for hash function was replaced by GOST R 34.11-2012. A family of hash functions Streebog was approved as a new standard. We analyse the family Streebog from the mathematical cryptography viewpoint and prove that it is secure against preimage and collision attacks.

Key words: hash functions, mathematical cryptography, Streebog, GOST R 34.11-2012

Citation: *Mathematical Aspects of Cryptography*, 2015, vol. 6, no. 2, pp. 79–98 (Russian).

1. Введение

ГОСТ Р 34.11-2012 [1], или Стрибог, — хэш-функции, принятые 1 января 2013 года в качестве национального стандарта Российской Федерации, определяющего алгоритм и процедуру вычисления хэш-функции. Старый стандарт был заменен после успешной атаки Ф. Менделя (F. Mendel) и др., понизившей сложность построения коллизий в 2^{23} раз [2].

Новый стандарт вызвал повышенное внимание криптографического научного сообщества, и в 2013 году появилось несколько работ, посвященных криптоанализу Стрибога. В частности, в работе Р. АльТави (R. AlTawy) и др. [3] описывается процедура построения коллизий и почти коллизий со свободным стартом для сокращенного до 4.5, 5.5 и 7.75 раундов внутреннего шифра функции сжатия, используемой в хэш-функции Стрибог. В этой работе внутренний шифр рассматривался как функция сжатия.

Несколько позже появилась статья З. Ванг (Z. Wang) и др. [4], связанная с построением коллизий для сокращенной до 9.5 раундов функции сжатия со сложностью по времени 2^{176} и необходимым количеством памяти 2^{128} байт. Также в этой работе представлен метод построения k-коллизий для 512-битной версии функции Стрибог.

В дальнейшем Р. АльТави (R. AlTawy) и Амр М. Юссуф (Amr M. Youssef) опубликовали статьи по исследованию стойкости функции Стрибог к нахождению прообраза [7] и по исследованию злонамеренного (malicious) хэширования с помощью функции Стрибог [8]. В [7] был получен псевдопрообраз для сокращенной до 5 раундов функции сжатия со сложностью 2^{448} по времени и с необходимым количеством памяти 2^{64} бит. В работе [8] авторы исследуют функцию Стрибог с измененными раундовыми константами и числом раундов функции сжатия, равным 12, и для такого варианта функции Стрибог описывают двухблоковую коллизию.

Также известна работа Бинкге Ма (Bingke Ma) и др. [9], в которой авторы предлагают атаки, связанные с нахождением прообраза и построением коллизий, на сокращенные версии хэш-функции Стрибог. В этой работе авторы предлагают улучшенный алгоритм построения прообраза для сокращенной до 6 раундов функции Стрибог, а также улучшенный алгоритм построения коллизий для сокращенной до 7,5 раундов функции сжатия Стрибог.

Указанные выше работы по исследованию хэш-функции Стрибог посвящены построению криптографических атак на нее, и, насколько нам известно, в открытой печати нет работ, посвященных обоснованию криптографической стойкости данной хэш-функции.

В январе 2009 года Е. Додис (E. Dodis), Т. Ристенпарт (T. Ristenpart) и Т. Шримптон (T. Shrimpton) [5] вводят для функций общего вида понятие стойкости к атаке на прообраз (preimage awareness) и доказывают стойкость в этом смысле некоторых функций сжатия, а также итеративной схемы Меркля–Дамгарда, используя модель идеального шифра.

В нашей работе рассматривается семейство хэш-функций Стрибог в модели идеального шифра и доказывается его стойкость к атаке поиска коллизий и стойкость к атаке поиска прообраза в определениях работы [5].

2. Определения, обозначения и известные результаты

2.1. Определения

Обозначения. Для двух строк x и y будем обозначать через $x||y$ их конкатенацию, а записью x^n будем обозначать конкатенацию n экземпляров вектора x. Для некоторого множества X запись $X \leftarrow x$ обозначает добавление элемента x в множество X, а запись $y \xleftarrow{\$} X$ означает, что в переменную y следует занести случайно выбранное значение из множества X. Если множество X конечно, то будем рассматривать равномерное распределение, а если $X = \{0,1\}^\infty$, то будем использовать такое распределение, что для любого подмножества

$$Y_{i_1,i_2,\ldots,i_m}^{\sigma_1,\sigma_2,\ldots,\sigma_m} = \{y = (y_1, y_2, \ldots, y_n, \ldots) \in \{0,1\}^\infty \mid y_{i_k} = \sigma_k,\ \sigma_k \in \{0,1\}, k = 1, \ldots, m\}$$

вероятность того, что $Y_{i_1,i_2,\ldots,i_m}^{\sigma_1,\sigma_2,\ldots,\sigma_m} \ni \alpha \xleftarrow{\$} \{0,1\}^\infty$, равна $\frac{1}{2^m}$. Для строки x будем обозначать через $|x|$ ее длину. Записью $m_1, m_2, \ldots, m_k \xleftarrow{n} M$ будем обозначать разбиение строки M длины $k \cdot n$ на подстроки длины n.

В дальнейшем при построении моделей защищенности нам потребуется также определение достаточно малой функции. Функция $\mu : \mathbb{N} \to \mathbb{R}$ называется *достаточно малой*, если для любого полинома $p(\cdot)$ существуют такие $N \in \mathbb{N}$ и $c \in \mathbb{R}$, что для любого $n > N$

$$|\mu(n)| < \frac{c}{p(n)}.$$

Вероятностные машины Тьюринга (МТ). Определим *вероятностную машину Тьюринга* следующим образом. Пусть помимо обычной ленты с входным словом x, доступной машине Тьюринга M и на чтение, и на запись, имеется дополнительная лента, содержащая строку $r \xleftarrow{\$} \{0,1\}^\infty$

и доступная M только на чтение. Читающая головка данной ленты в начальный момент времени стоит на первом символе этой строки.

Для машины Тьюринга M кроме основного алфавита входа M_A определим некоторый дополнительный алфавит M_I, который будем называть *алфавитом интерфейсов*. Будем считать, что для каждого $i_k \in M_I$ машина M на слове $i_k||a$, где $a \in M_A$, моделирует работу некоторой машины Тьюринга M_{i_k} на слове a. Каждый символ i из алфавита M_I, а также МТ, моделируемую M на слове a, где $a \in M_A$, будем называть *интерфейсом*.

Взаимодействие МТ A и МТ B будем описывать следующим образом. Будем считать, что помимо своих лент у A и B есть общая лента и каждая из МТ имеет к ней доступ на чтение и на запись. Машина A производит вычисления на своем входном слове x_A, затем записывает некоторое сообщение m_1 на общую ленту и переходит в специальное состояние ожидания, запуская тем самым машину B. Машина B производит некоторые вычисления над своим входным словом x_B и сообщением m_1, записывает на общую ленту сообщение m_2 и переходит в состояние ожидания, запуская МТ A. Если на входных словах x_A, x_B МТ A и B останавливаются и выдают сообщения y_A и y_B соответственно, то пара y_A, y_B называется результатом взаимодействия машин A и B.

Если во взаимодействии машин A и B необходимо выделить машину A как основную, а машину B как зависимую, взаимодействие машин A и B называется *доступом машины A к машине B*, или считается, что МТ *A может делать запросы* к МТ B. Иногда в таком случае машина B называется *оракулом*. Для МТ A формула $x \Leftarrow A$ обозначает запись вывода A в переменную x, а формула $x \overset{\$}{\Leftarrow} A$ — запись вывода в переменную при условии, что МТ A может быть вероятностной. Возможность доступа некоторой МТ A к МТ $P_1, P_2, \ldots, P_n, \ldots$ обозначается $A^{P_1,P_2,\ldots,P_n,\ldots}$. Будем писать $A \Rightarrow a$ для обозначения события, при котором МТ A вывела значение a. Под словами *МТ B запускает МТ A, перенаправляя запросы МТ A к оракулу P_A на вход оракулу P_B*, подразумевается следующий процесс: МТ B имеет доступ к машине A, у которой общая лента с машиной P_A заменена на общую ленту с машиной P_B. *Эффективным по параметру k* алгоритмом называется алгоритм, выполняющийся за полиномиальное по параметру k время. Под словами «алгоритм» и «процедура» в дальнейшем понимается некоторая машина Тьюринга, возможно вероятностная.

Для алгоритма f, получающего на вход строку из некоторого непустого множества $Dom \subseteq \{0,1\}^*$, определяется $Time(f, m)$ как максимальное время работы алгоритма $f(x)$ для любого такого входа $x \in Dom$, что $|x| \leqslant m$. Ес-

ли второй аргумент опускается, то $Time(f)$ рассматривается как максимальное время работы для всех возможных входов $x \in Dom$. Под обозначением $Time(f(x))$ понимается время работы f на строке $x \in Dom$. При подсчете времени работы МТ A время ожидания ответа от оракула B будем считать одним тактом работы МТ A.

Идеальные примитивы. Под *идеальным примитивом* подразумевается некоторая (возможно, вероятностная) машина Тьюринга. Данное определение используется, чтобы подчеркнуть использование этой машины Тьюринга в построении более сложных конструкций (см. [5, с. 7]).

Для фиксированных множеств Dom, Rng *случайным оракулом* $\mathcal{F}_{Dom,Rng}$ называется идеальный примитив, действующий следующим образом. При первом запросе у оракула $\mathcal{F}_{Dom,Rng}$ некоторого значения $x_i \in Dom$ он сопоставляет ему некоторое случайно выбранное значение $y_i \in Rng$ и возвращает y_i. При повторном запросе x_i возвращается соответствующий ему y_i. Если $Dom = \{0,1\}^n$ или множество $Rng = \{0,1\}^k$, то соответствующие обозначения в индексе заменяются на n и k соответственно, т. е. вместо $\mathcal{F}_{Dom,Rng}$ пишется $\mathcal{F}_{n,k}$.

Модель идеального шифра. Для любых натуральных n и k обозначим

$$BC(k,n) = \{E\colon \{0,1\}^n \times \{0,1\}^k \to \{0,1\}^n \colon \forall K \in \{0,1\}^k \\ E(\cdot,K) \text{ является биективным на множестве } \{0,1\}^n\}.$$

Идеальным шифром называется идеальный примитив $C_{k,n} = (E,D)$ с двумя интерфейсами, реализующими соответственно преобразование $E(x,k) = y$, случайно выбранное из $BC(k,n)$, и обратное ему преобразование $D(y,k) = x$ (см. [5, с. 7]).

Хэш-функции. Пусть $Dom \subseteq \{0,1\}^*$ — непустое множество строк, а Rng — некоторое непустое множество (обычно $\{0,1\}^n$). *Хэш-функцией* называется алгоритм, который вычисляет отображение $H : Dom \to Rng$. В данной статье мы будем рассматривать хэш-функции, использующие некоторый идеальный примитив P. Чтобы подчеркнуть эту зависимость, будем писать H^P. Для некоторых $n, d > 0$ хэш-функцию $f^P : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^n$ будем называть *функцией сжатия*.

Семейством хэш-функций будем называть некоторое нумерованное множество хэш-функций $\mathcal{H}$, в котором для любого $n \in \mathbb{N}$ определена хэш-функция $H_n \in \mathcal{H}$, осуществляющая отображение $H_n : Dom \to \{0,1\}^n$, где $Dom \subseteq \{0,1\}^*$.

О противнике. Будем называть *противником* для семейства хэш-функций $\mathcal{H} = \{H_n \mid H_n : Dom \to \{0,1\}^n\}$ эффективную по параметру n вероятностную машину Тьюринга A, на вход которой подается параметр n. При рассмотрении работы противника время обращения к другим машинам Тьюринга будем считать равным одному такту работы противника. В зависимости от задачи вывод противника A может различаться.

Напомним некоторые определения, которые Е. Додис, Т. Ристенпарт и Т. Шримптон ввели в работе [5].

Устойчивость к коллизиям (CR). Зафиксируем множество $Dom \subseteq \subseteq \{0,1\}^*$, и пусть A — противник, выводящий две строки: $x, x^* \in Dom$ (CR-противник). Пусть P — идеальный примитив. Для семейства хэш-функций $\mathcal{H}^P = \{H_n^P \mid H_n^P : Dom \to \{0,1\}^n\}$ и противника A *CR-преимущество* определяется как

$$Adv^{cr}_{H_n^P,P}(A) = Pr\left[(x, x^*) \xleftarrow{\$} A^P : H_n^P(x) = H_n^P(x^*) \wedge x \neq x^*\right].$$

Если $Adv^{cr}_{H_n^P,P}(A)$ как функция от n достаточно мало для любого противника A, то семейство функций $\mathcal{H}^P$ называется *CR-защищенным*, или *устойчивым к коллизиям*. В понимании данного определения может помочь рисунок 1.

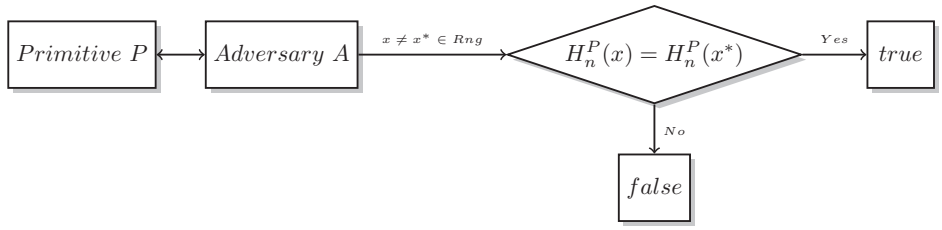

Рис. 1. Устойчивость к коллизиям.

Неотличимость. Пусть $\hat{\mathcal{H}}^P$ — семейство хэш-функций, которое использует идеальный примитив P и отображает множество Dom в множество $\{0,1\}^n$, а $\mathcal{H}^Q$ — другое семейство хэш-функций, которое использует идеальный примитив Q и также отображает множество Dom в множество $\{0,1\}^n$. Семейство $\hat{\mathcal{H}}^P$ называется *неотличимым* (*indifferentiable*) от $\mathcal{H}^Q$, если существует такой симулятор (МТ) S, симулирующий поведение P, который может делать запросы к Q, что для любого противника A, выводящего 1 или 0

(PRO-противника), *indiff-преимущество*

$$Adv^{indiff}_{\hat{H}^P_n,H^Q_n,S}(A) = \left|Pr\left[A^{\hat{H}^P_n,P} \Rightarrow 1\right] - Pr\left[A^{H^Q_n,S} \Rightarrow 1\right]\right|$$

как функция от n достаточно мало для любого противника A. Если преимущество $Adv^{indiff}_{\hat{H}^P_n,H^Q_n,S}(A)$ достаточно мало, то можно использовать $\hat{\mathcal{H}}^P$ вместо $\mathcal{H}^Q$ с точностью до оценки неотличимости.

PRO-защищенность (неотличимость от случайного оракула) семейства хэш-функций $\mathcal{H}^P = \{H^P_n \mid H^P_n : Dom \to \{0,1\}^n\}$ определяется следующим образом: заменим в приведенном выше определении $\mathcal{H}^Q$ семейством случайных оракулов $\mathcal{F}_{Dom,n}$, т. е. положим $H^Q_n = Q = \mathcal{F}_{Dom,n}$. Определим *PRO-преимущество* как $Adv^{PRO}_{H^P_n,S}(A) = Adv^{indif}_{H^P_n,\mathcal{F}_{Dom,n},S}(A)$. Иначе говоря,

$$Adv^{PRO}_{H^P_n,S}(A) = \left|Pr\left[A^{H^P_n,P} \Rightarrow 1\right] - Pr\left[A^{\mathcal{F}_{Dom,n},S} \Rightarrow 1\right]\right|,$$

где симулятор S может делать запросы к $\mathcal{F}_{Dom,n}$. Если для любого противника A преимущество $Adv^{PRO}_{H^P_n,S}(A)$ достаточно мало как функция от n, будем называть $\mathcal{H}^P$ *неотличимым от случайного оракула*, или *PRO-защищенным*.

PGV-схемы первого типа. В своей работе Мартин Стэм (Martijn Stam) [6] предложил классификацию функций сжатия, основанных на использовании блочных шифров. Он заметил, что эти функции, получая на вход переменную сцепления $v \in \{0,1\}^n$ и блок сообщения $m \in \{0,1\}^d$, используют их следующим образом:

$$\boxed{f^E(v,m):}$$
$$(k,x) \leftarrow C^{PRE}(v,m);$$
$$y \leftarrow E(k,x);$$
$$Ret\ w \leftarrow C^{POST}(v,m,y),$$

где $C^{PRE} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^d \times \{0,1\}^n$ — функция предобработки, $E(k,x)$ — блоковый шифр, а $C^{POST} : \{0,1\}^n \times \{0,1\}^d \times \{0,1\}^n \to \{0,1\}^n$ — функция постобработки. Стэм определил также вспомогательную функцию постобработки $C^{AUX}(k,x,y) : \{0,1\}^d \times \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, которая выполняет преобразование $C^{AUX}(k,x,y) = C^{POST}(C^{-PRE}(k,x),y)$, где C^{-PRE} обозначает обратную к C^{PRE} функцию, если C^{PRE} является биективным отображением. В данных обозначениях предполагается существование эффективных по n алгоритмов, реализующих функции C^{PRE}, C^{-PRE} и C^{POST}.

Стэм назвал функцию сжатия *PGV-схемой типа 1*, если:

1) C^{PRE} является биективным отображением;
2) для любых v, m отображение $C^{POST}(v, m, \cdot)$ является биективным;
3) для любых k, y отображение $C^{AUX}(k, \cdot, y)$ является биективным.

Когда выполняются сформулируемые условия, через $C^{-POST}(v, m, \cdot)$ и $C^{-AUX}(k, \cdot, y)$ обозначим функции, обратные к $C^{POST}(v, m, \cdot)$ и $C^{AUX}(k, \cdot, y)$ соответственно.

Стойкость к атаке поиска прообраза. Будем использовать следующее понятие стойкости к атаке поиска прообраза [5]. Зафиксируем множество $Dom \subseteq \{0,1\}^*$, и пусть A — противник, выводящий строку $x \in Dom$. Рассмотрим $Pr\left[\mathbf{Exp}^{pra}_{H,P,\mathcal{E},A} \Rightarrow true\right]$, где процедура $\mathbf{Exp}^{pra}_{H,P,\mathcal{E},A}$ описана следующей последовательностью действий:

$\mathbf{Exp}^{pra}_{H,P,\mathcal{E},A}$:	**oracle** $\mathrm{P}(m)$:	**oracle** $\mathrm{Ex}(z)$:
$x \xleftarrow{\$} A^{\mathrm{P,Ex}}$	$c \leftarrow P(m)$	$Q[z] \leftarrow 1$
$z \leftarrow H^P(x)$	$\alpha \leftarrow \alpha \| (m, c)$	$V[z] \leftarrow \mathcal{E}(z, \alpha)$
$Ret\ ((x \neq V[z])\ \&(\ Q[z] = 1))$	$Ret\ c$	$Ret\ V[z]$

Противник A (PrA-противник) имеет доступ к двум оракулам. Во-первых, к оракулу P, который предоставляет доступ к идеальному примитиву P и, кроме того, сохраняет все запросы и ответы на запросы в строку состояния α. Если P предоставляет доступ к нескольким примитивам, то из строки состояния понятно, к какому именно примитиву был сделан запрос. Во-вторых, к оракулу Ex, который предоставляет доступ к экстрактору $\mathcal{E}$. *Экстрактором* $\mathcal{E}$ называется детерминированный алгоритм, который получает на вход точку $z \in Rng$ и строку состояния α и возвращает точку $x \in Dom \cup \{\perp\}$, где $\perp \notin Dom$ — некоторый символ. Оракул Ex использует два массива $Q[z]$ и $V[z]$, которые в начальный момент времени для любого $z \in Rng$ равны $\perp$. Массив $Q[z]$ определяет уже запрошенные у Ex точки из Rng, а $V[z]$ — результат, который вернул экстрактор $\mathcal{E}$. В дальнейшем мы будем рассматривать «честные» экстракторы, то есть такие, что если x является выводом экстрактора на некотором наборе $(z \in Rng, \alpha)$, то либо $x = \perp$, либо $H^P(x) = z$. Схематически взаимодействие машин Тьюринга можно представить в виде следующего рис. 2.

Для семейства хэш-функций $\mathcal{H}^P = \{H^P_n \mid H^P_n : Dom \to \{0,1\}^n\}$, противника A и экстрактора $\mathcal{E}$ авторы статьи [5] определяют *PRA-преимущество*

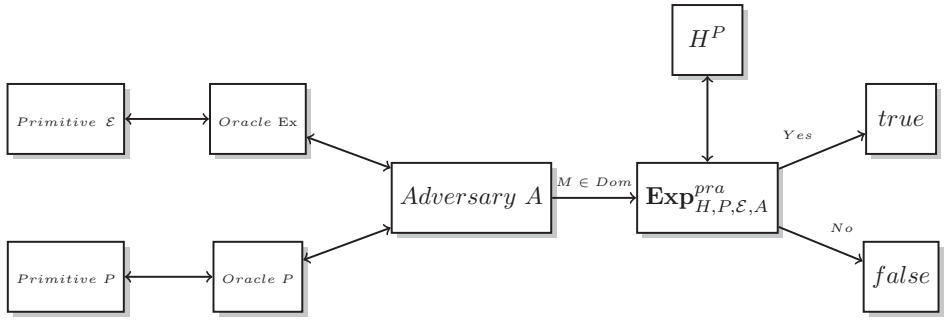

Рис. 2. Стойкость к атаке поиска прообраза.

как

$$Adv^{PrA}_{H_n,P,\mathcal{E}}(A) = Pr\left[\mathbf{Exp}^{pra}_{H_n,P,\mathcal{E},A} \Rightarrow true\right].$$

Предполагается, что противник никогда не делает запрос за пределы области определения соответствующего оракула. Если существует такой эффективный по n экстрактор $\mathcal{E}$, что для любого возможного противника A преимущество $Adv^{PrA}_{H_n,P,\mathcal{E}}(A)$ достаточно мало как функция от n, то семейство хэш-функций $\mathcal{H}$ называется *стойким к атаке поиска прообраза*, или *PrA-защищенным*.

При рассмотрении данной конструкции будем называть *победой* PRA-противника A событие, в результате которого $\mathbf{Exp}^{pra}_{H,P,\mathcal{E},A} \Rightarrow true$.

2.2. Используемые теоремы

Свойство стойкости к атаке на прообраз является более сильным, чем свойство усточивости к коллизиям, но более слабым, чем свойство неотличимости от случайного оракула. В работе [5] показано, что, во-первых, устойчивость к коллизиям следует из стойкости к атаке поиска прообраза, а во-вторых, что случайный оракул также является стойким к атаке поиска прообраза. Более формально: имеют место следующие 4 леммы.

Лемма 1 (теорема 3.1 из [5] (PrA $\Rightarrow$ CR)). *Пусть P — идеальный примитив и $\mathcal{H} = \{H_n \mid H_n : Dom \to \{0,1\}^n\}$ — семейство хэш-функций. Пусть $\mathcal{E}$ — некоторый экстрактор. Пусть A — CR-противник против $\mathcal{H}$, делающий*

всего q_P запросов к P и выводящий сообщения длиной не более $l_{\max}$. Тогда существует такой PrA-противник B, что

$$Adv^{cr}_{H_n,P}(A) \leqslant Adv^{PrA}_{H_n,P,\mathcal{E}}(B).$$

При этом $Time(B) = Time(A) + \mathcal{O}(q_P) + Time(H_n, l_{\max})$, B делает не более q_P запросов к P и один запрос к экстрактору.

Лемма 2 (теорема 3.2 из [5] (случайный оракул $\Rightarrow$ PrA)). *Пусть $Dom \subseteq \{0,1\}^*$ и $n > 0$. Пусть $P_n = \mathcal{F}_{Dom,n}$. Тогда семейство хэш-функций $\mathcal{H} = \{P_n\}$ является стойким к атаке поиска прообраза. В частности, существует такой экстрактор $\mathcal{E}$, что для любого противника A, делающего q_P запросов к P и q_e запросов к экстрактору $\mathcal{E}$, выполняется неравенство*

$$Adv^{PrA}_{H_n,\mathcal{E}}(A) \leqslant \frac{q_e q_P}{2^n} + \frac{q_P^2}{2^n}.$$

При этом $Time(\mathcal{E}) = \mathcal{O}(q_e + q_P)$.

Лемма 3 (теорема 5.1 из [5] (PGV-схема типа 1 $\Rightarrow$ PrA)). *Зафиксируем $k, n > 0$, пусть $C_{k,n} = (E, D)$ — идеальный шифр и пусть $\mathcal{H} = \{H^{C_{k,n}}\}$ — семейство PGV-схем типа 1, основанных на блоковом шифре. Тогда существует такой экстрактор $\mathcal{E}$, что для любого противника A, делающего не более q_P запросов к $C_{k,n}$ и q_e запросов к экстрактору $\mathcal{E}$, будет выполняться следующее неравенство:*

$$Adv^{PrA}_{H_n,C_{k,n},\mathcal{E}}(A) \leqslant \frac{q_e q_P}{2^n - q_P} + \frac{q_P(q_P+1)}{2(2^n - q_P)}.$$

При этом $Time(\mathcal{E}) = \mathcal{O}(q_P(Time(C^{-PRE}) + Time(C^{POST})))$.

Лемма 4 (теорема 4.1 из [5]). *Пусть P — идеальный примитив и $\mathcal{H}^P$ — семейство хэш-функций. Пусть R — идеальный примитив с двумя интерфейсами, которые предоставляют выполнение P и $\mathcal{R} = \mathcal{F}_{Rng,Rng}$. Определим $F_R(M)_n = \mathcal{R}(H^P_n(M))$. Пусть $\mathcal{E}$ — произвольный экстрактор для $\mathcal{H}^P$. Тогда существует такой симулятор (MT) $S = (S_1, S_2)$, что для каждого PRO-противника A, делающего не более (q_0, q_1, q_2) запросов к оракулам (F_R, R, S), существует такой PrA-противник B, что*

$$Adv^{PRO}_{F,R,S}(A) \leqslant Adv^{PrA}_{\mathcal{H},P,\mathcal{E}}(B).$$

При этом $Time(S) = \mathcal{O}(q_1 + q_2 \cdot Time(\mathcal{E}))$. Пусть $l_{\max}$ — максимальная длина (в битах) запроса, сделанного A к своему первому оракулу F_R. Противник B делает $q_1 + q_0 \cdot NumQueries(H, l_{\max})$ запросов к идеальному примитиву P, q_2 запросов к экстрактору $\mathcal{E}$ и выводит строку длины не более $l_{\max}$.

При этом $Time(B) = Time(A) + \mathcal{O}(q_0 \cdot Time(H_n, l_{\max}) + q_1 + q_2)$. *Здесь под* $NumQueries(H, l_{\max})$ *понимается максимальное количество обращений функции* H *к примитиву* P *на входе длины не более* $l_{\max}$.

3. ГОСТ Р 34.11-2012

3.1. Определение

Приведем формальное описание хэш-функции ГОСТ Р 34.11-2012 из [1]. Рассмотрим функцию сжатия

$$g_N(v, m) = E(L \circ P \circ S(v \oplus N), m) \oplus v \oplus m \mid \{0,1\}^{512} \times \{0,1\}^{512} \to \{0,1\}^{512}.$$

Здесь L — умножение слева на некоторую матрицу над полем $GF(2)$, P — некоторая перестановка байт, S — нелинейное биективное отображение байт.

Блочный шифр E определяется следующим образом:

$$E(k, m) = X[K_{13}] \circ \prod_{i=1}^{12} L \circ P \circ S \circ X[K_i](m),$$

где $X[K](m) = m \oplus K$, а раундовые ключи вычисляются по формуле

$$K_i = L \circ P \circ S(C_{i-1} \oplus K_{i-1}),$$

$K_0 = k$, C_i — некоторые константы, описанные в [1].

Процедуру вычисления хэш-значения сообщения M хэш-функцией ГОСТ Р 34.11-2012 можно описать с использованием следующего алгоритма GOST:

GOST
$M_1 \leftarrow M||1||0^{511-(Len \bmod 512)}||Len||Sum$
$m_1, m_2, \ldots, m_k \xleftarrow{512} M_1$
$N = 0; v_1 = IV;$
$for\ i = 1\ to\ k-2\ do$
  $v_{i+1} = g_N(v_i, m_i)$
  $N = N + 512$
$v_k = g_0(v_{k-1}, m_{k-1})$
$v_{k+1} = g_0(v_k, m_k)$
$return\ v_{k+1}$

Здесь через $Len \in \{0,1\}^{512}$ обозначена длина сообщения M, а через $Sum \in \{0,1\}^{512}$ — сумма блоков длины 512 и оставшегося блока длиной $|M| - (k-3)\cdot 512$ сообщения M по модулю 2^{512}, при этом используется естественное взаимно-однозначное соответствие элементов множества $\{0,1\}^{512}$ и кольца вычетов $\mathbb{Z}_{2^{512}}$.

Данная процедура при значении $IV = 0^{512}$ реализует алгоритм хэш-функции ГОСТ Р 34.11-2012 с длиной хэш-кода 512 бит. Для получения длины хэш-кода 256 бит используется значение инициализационного вектора $IV = (00000001)^{64}$, а вместо значения v_{k+1} выдаются его старшие 256 бит.

3.2. Построение семейства хэш-функций $\mathcal{GOST}$

Определим семейство хэш-функций $\mathcal{GOST}$, построенное на основе функции ГОСТ Р 34.11-2012. Для функций $GOST_{512} \in \mathcal{GOST}$ и $GOST_{256} \in \mathcal{GOST}$ положим их равными процедуре GOST с соответствующим размером выходного сообщения и значением IV, при условии замены функции E идеальным шифром. Для остальных натуральных n для построения функций $GOST_n \in \mathcal{GOST}$ рассмотрим семейство функций сжатия

$$\{g_N(v,m)_n\} = \{E(L_n \circ P_n \circ S_n(v \oplus N), m) \oplus v \oplus m \mid \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n\},$$

где L_n, P_n, S_n — некоторые биективные преобразования на $\{0,1\}^n$, а E — функция, реализуемая первым интерфейсом идеального шифра, отображающая $\{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. На основе этого построим алгоритм, реализующий функцию $GOST_n : \{0,1\}^* \to \{0,1\}^n$:

$\boxed{GOST_n}$
$M_1 \leftarrow M||1||0^{n-1-(Len \bmod n)}||Len||Sum$
$m_1, m_2, \ldots, m_k \xleftarrow{n} M_1$
$N = 0; v_1 = IV;$
$for\ i = 1\ to\ k-2\ do$
  $v_{i+1} = g_N(v_i, m_i)_n$
  $N = N + n$
$v_k = g_0(v_{k-1}, m_{k-1})_n$
$v_{k+1} = g_0(v_k, m_k)_n$
$return\ v_{k+1}$

Здесь через $Len \in \{0,1\}^n$ обозначена длина сообщения M, а через $Sum \in \{0,1\}^n$ — сумма блоков длины n и оставшегося блока длиной $|M| - (k-3)\cdot n$ сообщения M по модулю 2^n, при этом используется естественное

взаимно-однозначное соответствие элементов множества $\{0,1\}^n$ и кольца вычетов $\mathbb{Z}_{2^{512}}$. Сообщение M_1 в дальнейшем будем называть дополнением сообщения M. Схематически процесс вычисления функции $GOST_n$ можно описать рисунком 3.

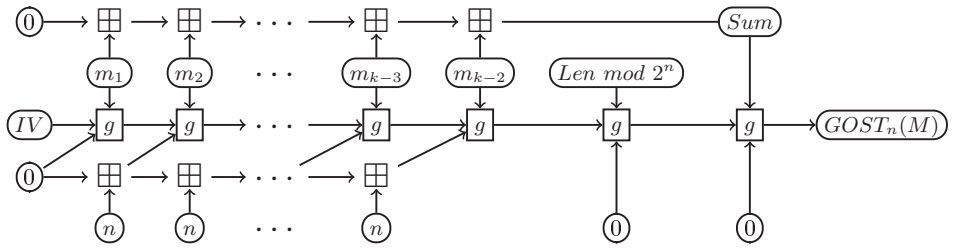

Рис. 3. Вычисление функции $GOST_n$.

3.3. Стойкость $\mathcal{GOST}$ к атаке поиска прообраза и атаке поиска коллизий

Покажем, что семейство хэш-функций $\mathcal{GOST}$ является стойким к атаке поиска прообраза.

Функция $g_0(v,m)_n$ удовлетворяет следующим условиям:

- $C^{PRE}(v,m) = (L_n \circ P_n \circ S_n(v), m)$ является биективным отображением;
- $C^{POST}(v,m,y) = y \oplus v \oplus m$ является биективным отображением относительно переменной y;
- $C^{AUX}(k,x,y) = y \oplus x \oplus S_n^{-1} \circ P_n^{-1} \circ L_n^{-1}(k)$ является биективным отображением относительно переменной x.

Следовательно, функция сжатия $g_0(v,m)_n$ является PGV-схемой типа 1, а значит, является стойкой к атаке поиска прообраза согласно лемме 3.

В дальнейшем нам также понадобится следующее свойство функции $g_N(\cdot,\cdot)_n$: $g_N(v,m)_n = g_0(v \oplus N, m)_n \oplus N$.

Теорема 3.1. *Для любого эффективного по n экстрактора $\mathcal{E}_g$ для семейства функций $\{g_0(v,m)_n\}$ существует такой эффективный по n экстрактор $\mathcal{E}_G$ для семейства функций $\mathcal{GOST}$, что для любого PRA-противника A, делающего не более q_p запросов к идеальному примитиву E*

и q_e запросов к экстрактору $\mathcal{E}_G$ и при любом n выводящего сообщение длиной не более $l_{\max}(n) \cdot n \leqslant 2^n$ символов, где $l_{\max}(n)$ — некоторая полиномиальная по n функция, существует такой PRA-противник B, что

$$Adv^{PrA}_{GOST_n,E,\mathcal{E}_G}(A) \leqslant Adv^{PrA}_{g_0(v,m)_n,E,\mathcal{E}_g}(B).$$

При этом противник B делает не более $q_p + l_{\max}(n) + 2$ запросов к идеальному примитиву E и не более $q_e \cdot (l_{\max}(n) + 2)$ запросов к экстрактору $\mathcal{E}_g$.

Доказательство. Построим на основе экстрактора $\mathcal{E}_g$ экстрактор $\mathcal{E}_G$ для семейства функций $\mathcal{GOST}$:

$\boxed{\mathcal{E}_G(z,\alpha)}$
$(v_1, m_1) \leftarrow \mathcal{E}_g(z, \alpha)$;
if $(v_1, m_1) = \perp$ *then Ret* $\perp$;
$(v_2, m_2) \leftarrow \mathcal{E}_g(v_1, \alpha)$;
if $(v_2, m_2) = \perp$ *then Ret* $\perp$;
$i = 3$; $N = m_2 - (m_2 \ mod \ n)$;
if $N > n \cdot (l_{\max}(n) + 2)$ *then Ret* $\perp$;
while $N \geqslant 0$ *do*
  $(v_i, m_i) \leftarrow \mathcal{E}_g(v_{i-1} \oplus N, \alpha)$;
  if $(v_i, m_i) = \perp$ *then Ret* $\perp$;
  $i = i + 1$;
  $v_i = v_i \oplus N$;
  $N = N - n$;
if $v_i \neq IV$ *then Ret* $\perp$;
$M \leftarrow unpad(m_i || m_{i-1} || \ldots || m_1)$;
Ret M;

Рассмотрим данный псевдокод подробнее. Экстрактор $\mathcal{E}_G$ принимает на вход строку $z \in \{0,1\}^n$ и строку состояния α. Затем он вычисляет $(v_1, m_1) \leftarrow \leftarrow \mathcal{E}_g(z, \alpha)$. Если $\mathcal{E}_g$ возвращает $\perp$, то $\mathcal{E}_G$ выдает $\perp$ и завершает свою работу. Если нет, то вычисляется следующее значение $(v_2, m_2) \leftarrow \mathcal{E}_g(v_1, \alpha)$. Заметим, что в случае удачной работы экстрактора $\mathcal{E}_G$ символ m_1 будет обозначать сумму блоков выводимого сообщения M, а m_2 — длину сообщения M. Если $\mathcal{E}_g$ возвращает $\perp$, то $\mathcal{E}_G$ выдает $\perp$ и завершает свою работу. Иначе $\mathcal{E}_G$ вычисляет $N = m_2 - (m_2 \ mod \ n)$ и запускает цикл, в котором последовательно вычисляет (v_i, m_i) с помощью свойства $g_N(h, m) = g_0(h \oplus N, m) \oplus N$ и экстрактора $\mathcal{E}_g$ функции сжатия g_0. Если в какой-то момент $\mathcal{E}_g$ возвращает $\perp$, то $\mathcal{E}_G$ выдает $\perp$ и завершает свою работу. В противном случае $\mathcal{E}_G$ по окончании цикла запускает функцию $unpad(m_i || m_{i-1} || \ldots || m_1)$, которая проверяет,

могло ли получиться сообщение $\hat{M} = (m_i||m_{i-1}||\ldots||m_1)$ из некоторого сообщения M в результате процедуры дополнения (padding) входного сообщения, то есть $\hat{M} = (M||1||0^{n-1-(Len\ mod\ n)}||Len||Sum)$, и возвращает M, если могло, и $\perp$, если нет. По завершении работы функции $unpad(\hat{M})$ экстрактор $\mathcal{E}_G$ возвращает результат ее работы. Экстрактор $\mathcal{E}_G$ является эффективным по n алгоритмом, так как производит не более $l_{\max}(n) + 2$ полиномиальных по n действий в цикле и полиномиален по n вне цикла. Первое утверждение следует из ограничения $N \geqslant n \cdot (l_{\max}(n) + 2)$ (в противном случае не будет выполняться условие входа в цикл), второе очевидно из построения $\mathcal{E}_G$.

Теперь построим противника B (псевдокод 2), используя противника A и процедуру $Sim\mathcal{E}_G$ (псевдокод 1). В дальнейшем оракулы Ex_A и Ex_B — это оракулы экстракторов, возникающие в экспериментах $\mathbf{Exp}^{pra}_{GOST_n,E,\mathcal{E}_G,A}$ и $\mathbf{Exp}^{pra}_{g_0,E,\mathcal{E}_g,B}$ соответственно.

$\boxed{Sim\mathcal{E}_G(z,\alpha)}$
$(v_1, m_1) \leftarrow \mathrm{Ex}_B(z, \alpha)$;
$\hat{Q}[z] = 1; \hat{V}[z] = (v_1, m_1)$;
$if\ (v_1, m_1) = \perp\ then\ Ret\ \perp$;
$(v_2, m_2) \leftarrow \mathrm{Ex}_B(v_1, \alpha)$;
$\hat{Q}[v_1] = 1; \hat{V}[v_1] = (v_2, m_2)$;
$if\ (v_2, m_2) = \perp\ then\ Ret\ \perp$;
$i = 3;\ \ N := m_2 - (m_2\ mod\ n)$;
$if\ N > n \cdot (l_{\max}(n) + 2)\ then\ Ret\ \perp$;
$while\ N \geqslant 0\ do$
  $(v_i, m_i) \leftarrow \mathrm{Ex}_B(v_{i-1} \oplus N, \alpha)$;
  $\hat{Q}[v_{i-1} \oplus N] = 1; \hat{V}[v_{i-1} \oplus N] = (v_i, m_i)$;
  $if\ (v_i, m_i) = \perp\ then\ Ret\ \perp$;
  $v_i = v_i \oplus N; i = i + 1$;
  $N = N - n$;
$if\ v_i \neq IV\ then\ Ret\ \perp$;
$M \leftarrow unpad(m_i||m_{i-1}||\ldots||m_1)$;
$Ret\ M$;

Псевдокод 1. Процедура $Sim\mathcal{E}_G$

Рассмотрим подробнее устройство противника B и процедуры $Sim\mathcal{E}_G$. Сначала B запускает противника A, перенаправляя его запросы к оракулу экстрактора Ex_A на вход процедуре $Sim\mathcal{E}_G$. Процедура $Sim\mathcal{E}_G$ действует так же, как и экстрактор $\mathcal{E}_G$, за исключением того, что все запросы к экстрактору $\mathcal{E}_g$

$\boxed{Adversary\ B^{\mathrm{E},\mathrm{Ex}_B}(\mathcal{E})}$
$M^* \xleftarrow{\$} A^{\mathrm{E},Sim\mathcal{E}_G}$
$\hat{M}^* \leftarrow M^*||1||0^{n-1-(|M^*| \bmod n)}|| \ |M^*| \ ||Sum$
$m_k^*, m_{k-1}^*, \ldots, m_1^* \xleftarrow{n} \hat{M}^*$
$N = 0; v_k = IV;$
$for\ i = k\ downto\ 3\ do$
  $v_{i-1}^* = g_0(v_i^* \oplus N, m_i^*) \oplus N$
  $if\ \hat{Q}[v_{i-1}^* \oplus N] = 1\ and\ \hat{V}[v_{i-1}^* \oplus N] \neq (v_i^* \oplus N, m_i^*)$
    $then\ Ret\ (v_i^* \oplus N, m_i^*);$
  $N = N + n;$
$v_1 = g_0(v_2, m_2);$
$if\ \hat{Q}[v_1^*] = 1\ and\ \hat{V}[v_1^*] \neq (v_2^*, m_2^*)$
  $then\ Ret\ (v_2^*, m_2^*);$
$v_0 = g_0(v_1, m_1);$
$if\ \hat{Q}[v_0^*] = 1\ and\ \hat{V}[v_0^*] \neq (v_1^*, m_1^*)$
  $then\ Ret\ (v_1^*, m_1^*);$
$Ret\ res \xleftarrow{\$} Dom;$

Псевдокод 2. Противник B

заменяются запросами к оракулу Ex_B, а результат каждого запроса к этому оракулу сохраняется в массивах $\hat{Q}[\,]$ и $\hat{V}[\,]$. Затем B производит хэширование выведенного противником A сообщения, после каждого обращения к функции сжатия проверяя истинность вывода процедуры $\mathbf{Exp}^{pra}_{g_0,E,\mathcal{E}_g,B}$. Докажем теперь, что $Adv^{PrA}_{GOST_n,E,\mathcal{E}_G}(A) \leqslant Adv^{PrA}_{g_0(v,m)_n,E,\mathcal{E}_g}(B)$.

Обозначим через Ω событие, при котором $A^{E,\mathcal{E}_G}$ выведет сообщение, приводящее к его победе, то есть $\Omega = \{\mathbf{Exp}^{pra}_{GOST_n,E,\mathcal{E}_G,A} \Rightarrow true\}$. Из построения процедуры $Sim\mathcal{E}_G$ и определения стойкости к атаке поиска прообраза следует, что

$$Pr[\Omega] = Pr[\mathbf{Exp}^{pra}_{GOST_n,E,\mathcal{E}_G,A} \Rightarrow true] = \\ = Pr[\mathbf{Exp}^{pra}_{GOST_n,E,Sim\mathcal{E}_G,A} \Rightarrow true] = Adv^{PrA}_{GOST_n,E,\mathcal{E}_G}(A).$$

Второе равенство в цепочке следует из того, что экстрактор $\mathcal{E}_g$ выдает такой же ответ, как и оракул Ex_B.

Покажем, что $Pr[\Omega] \leqslant Adv^{PrA}_{g_0(v,m)_n,E,\mathcal{E}_g}(B)$. Рассмотрим элементарное событие $\omega \in \Omega$. Так как $\omega \in \Omega$, то противник A выведет некоторое сообще-

ние M^*, приводящее к его победе, в процессе работы МТ $\mathbf{Exp}^{pra}_{GOST_n,E,Sim\mathcal{E}_G,A}$. Из условия $\mathbf{Exp}^{pra}_{GOST_n,E,Sim\mathcal{E}_G,A} \Rightarrow true$ следует, что $GOST(M^*)$ запрашивалось у $Sim\mathcal{E}_G$ и было возвращено такое сообщение M, что $M \neq M^*$. Тогда рассмотрим сообщения $\hat{M} = (m_k||m_{k-1}||\ldots||m_1)$ и $\hat{M}^* = (m_l^*||m_{l-1}^*||\ldots||m_1^*)$ — дополнения сообщений M и M^* соответственно. Из определения семейства хэш-функций $\mathcal{GOST}$ следует, что $k > 2$ и $l > 2$. Докажем, что существует такое $s \leqslant min\{k,l\}$, что $m_s^* \neq m_s$. Пусть данное условие не выполняется. Тогда $m_1^* = m_1$ и $m_2^* = m_2$, т. е. длины сообщений M и M^* равны, следовательно, $min\{k,l\} = k = l$. Так как по предположению для каждого $i \leqslant min\{k,l\}$ имеет место равенство $m_i = m_i^*$, то сообщения M и M^* совпадают, что противоречит событию ω.

Так как из неравенства $m_s^* \neq m_s$ следует, что для любого $v \in \{0,1\}^n$ имеет место неравенство $(v, m_s^*) \neq (v, m_s)$, то существует такое $1 \leqslant s < 3$, что $(v_s^*, m_s^*) \neq (v_s, m_s)$, или существует такое $3 \leqslant s \leqslant min\{k,l\}$, что $(v_s^* \oplus N, m_s^*) \neq (v_s, m_s)$. Пусть r — максимальное из таких значений s. Если $v_{r-1}^* = v_{r-1}$, то возврат пары $(v_r^* \oplus N, m_r^*)$ обеспечивает истинность вывода процедуры $\mathbf{Exp}^{pra}_{g_0,E,\mathcal{E}_g,B}$, потому что v_{r-1} запрашивалось у Ex_B и было возвращено значение (v_r, m_r), а $(v_r^* \oplus N, m_r^*) \neq (v_r, m_r)$, хотя $g_0(v_r, m_r)_n = g_0(v_r^* \oplus N, m_r^*)_n$. Если же нет, то рассмотрим следующую пару с $r = s - 1$. Тогда $(v_{s-1}^* \oplus N, m_{s-1}^*) \neq (v_{s-1}, m_{s-1})$. Если $v_{s-2}^* = v_{s-2}$, то возврат пары $(v_{s-1}^* \oplus N, m_{s-1}^*)$ снова обеспечивает победу B. В противном случае продолжаем уменьшать индекс r и рассматривать пары (v_r, m_r). В какой-то момент $v_r^* = v_r$ (в противном случае $GOST(M) \neq GOST(M^*)$, так как $v_0^* = GOST(M^*)$, а $v_0 = GOST(M)$). Тогда возврат пары $(v_{r+1}^* \oplus N, m_{r+1}^*)$ для $r > 1$ или пары (v_{r+1}^*, m_{r+1}^*) для $r \leqslant 1$ обеспечивает победу B. Изменения в случае $r \leqslant 1$ связаны с тем, что последние два блока соответствующих сообщений обрабатываются функцией сжатия $g_0(v,m)_n$, а не $g_N(v,m)_n$. Отсюда получаем, что

$$Adv^{PrA}_{g,E,\mathcal{E}_g}(B) \geqslant Pr[\omega] = Adv^{PrA}_{G,E,\mathcal{E}_G}(A).$$

Теперь докажем эффективность по n построенной нами МТ B. Противник B состоит из двух частей. Первая — это запуск противника A. Так как A по определению противника является эффективным по n алгоритмом, то эта часть эффективна по n. В ходе запуска противника A происходит не более q_p запросов к идеальному примитиву E и не более $q_e \cdot (l_{\max}(n)+2)$ запросов к экстрактору $\mathcal{E}_g$, так как при каждом запросе противника A к процедуре $Sim\mathcal{E}_G$ происходит не более $(l_{\max}(n)+2)$ запросов к Ex_B. Иначе противник A мог бы возвращать сообщение длиной более $l_{\max}(n) \cdot n$, что противоречит условию теоремы. Во второй части МТ B производит хэширование полученного на

первом этапе сообщения M^* длины не более $l_{\max}(n) \cdot n$. Для преобразования с помощью функции сжатия одного блока сообщения и одного вспомогательного блока требуется один запрос к идеальному примитиву E. Так как длина сообщения M^* не больше $l_{\max}(n) \cdot n$, то хэширование потребует не более $(l_{\max}(n) + 2)$ запросов к идеальному примитиву E. Так как противник A может выводить сообщения не более чем полиномиальной по n длины, то и вторая часть B полиномиальна по n. Отсюда получаем полиномиальность противника B по n, что и завершает доказательство.

Теорема 3.2 (ГОСТ является PrA-защищенным). *Семейство хэш-функций $\mathcal{GOST}$ является стойким к атаке поиска прообраза и к атаке поиска коллизий.*

Доказательство. Согласно определению PrA-защищенности требуется показать, что существует такой эффективный экстрактор $\mathcal{E}$, что для любого возможного противника A преимущество $Adv^{PrA}_{GOST_n,E,\mathcal{E}}(A)$ достаточно мало как функция от n.

Так как функция $g_0(v, m)_n$ является PGV-схемой типа 1, а следовательно, и PrA-защищенной, то для нее существует такой экстрактор $\mathcal{E}_g$, что для любого возможного противника B преимущество $Adv^{PrA}_{g_0(v,m)_n,E,\mathcal{E}_g}(B)$ достаточно мало как функция от n. Зафиксируем данный экстрактор $\mathcal{E}_g$. Согласно теореме 3.1 существует такой экстрактор $\mathcal{E}_G$ для функции $GOST_n$, что для любого противника A, делающего не более q_p запросов к идеальному примитиву E и q_e запросов к экстрактору $\mathcal{E}_G$ и выводящего сообщение длины не более $l_{\max}(n) \cdot n \leqslant 2^n$ символов, где $l_{\max}(n)$ — некоторая полиномиальная по n функция, существует такой противник B, что

$$Adv^{PrA}_{GOST_n,E,\mathcal{E}_G}(A) \leqslant Adv^{PrA}_{g_0(v,m)_n,E,\mathcal{E}_g}(B).$$

Причем противник B делает не более $q_p + l_{\max}(n) + 2$ запросов к идеальному примитиву E и не более $q_e \cdot (l_{\max} + 2)$ запросов к экстрактору $\mathcal{E}_g$.

По лемме 3 преимущество $Adv^{PrA}_{g_0(v,m)_n,E,\mathcal{E}_g}(B)$ может быть ограничено сверху в терминах числа запросов противника A к экстрактору $\mathcal{E}_G$ из теоремы 3.1 и идеального примитива E, то есть

$$Adv^{PrA}_{g_0(v,m)_n,E,\mathcal{E}_g}(B) \leqslant \frac{q_e(l_{\max}+2)(q_p + l_{\max}(n) + 2)}{2^n - (q_p + l_{\max}(n) + 2)} + \\ + \frac{(q_p + l_{\max}(n) + 2)(q_p + l_{\max}(n) + 3)}{2(2^n - (q_p + l_{\max}(n) + 2))}.$$

Отсюда следует, что существует такой экстрактор $\mathcal{E}_G$ для функции $GOST_n$, что для любого противника A, делающего максимум q_p запросов к идеальному примитиву E и q_e запросов к экстрактору $\mathcal{E}_G$ и выводящего сообщение длиной не более $l_{\max}(n) \cdot n \leqslant 2^n$ символов, где $l_{\max}(n)$ — некоторая полиномиальная от n функция, выполняется

$$Adv^{PrA}_{GOST_n,E,\mathcal{E}_G}(A) \leqslant \frac{q_e(l_{\max}+2)(q_p+l_{\max}(n)+2)}{2^n-(q_p+l_{\max}(n)+2)} + \\ + \frac{(q_p+l_{\max}(n)+2)(q_p+l_{\max}(n)+3)}{2(2^n-(q_p+l_{\max}(n)+2))}.$$

Следовательно, преимущество $Adv^{PrA}_{GOST_n,E,\mathcal{E}_G}(A)$ достаточно мало как функция от n, а значит, семейство хэш-функций $\mathcal{GOST}$ является стойким к атаке поиска прообраза. Стойкость к атаке поиска коллизий следует из утверждения леммы 1, которое ограничивает сверху CR-преимущество PrA-преимуществом.

Применяя лемму 4 к теореме 3.2, получаем, что семейство хэш-функций $\{\mathcal{F}_{n,n}(GOST_n(M))\}$ является неотличимым от случайного оракула произвольной длины.

4. Заключение

В работе на основе семейства хэш-функций ГОСТ Р 34.11-2012 описано такое семейство хэш-функций $\mathcal{GOST}$, что при одинаковой длине хэш-кода функции семейства $\mathcal{GOST}$ получаются из функций семейства ГОСТ Р 34.11-2012 заменой внутреннего шифра на идеальный. Для остальных функций семейства $\mathcal{GOST}$ использована схожая конструкция. Для построенного семейства $\mathcal{GOST}$ доказаны стойкость к атаке поиска прообраза и устойчивость данного семейства к коллизиям. Данные свойства семейства $\mathcal{GOST}$ косвенно подтверждают стойкость использованной в ГОСТ Р 34.11-2012 конструкции. В дальнейшем планируется исследование семейства хэш-функций $\mathcal{GOST}$ на соответствие свойству неотличимости от случайного оракула, а также, возможно, исследование на удовлетворение понятиям стойкости к атаке поиска прообраза и устойчивости к коллизиям семейства хэш-функций, лучше отображающего свойства функции сжатия, используемой в ГОСТ Р 34.11-2012.

Благодарности

В заключение автор выражает глубокую благодарность своему научному руководителю Карпунину Г. А. за постановку задачи и внимание к работе.

Также автор признателен рецензенту И. В. Лаврикову за важные замечания, способствовавшие улучшению изложения результатов.

Список литературы

[1] *ГОСТ Р 34.11-2012, Национальный стандарт Российской Федерации. Информационная технология. Криптографическая защита информации. Функция хэширования.*, Стандартинформ, Москва, 2012.

[2] Mendel F., Pramstaller N., Rechberger C., Kontak M., Szmidt J., “Cryptanalysis of the GOST hash function CRYPTO’2008”, *Lect. Notes Comput. Sci*, **5157** (2008), 162–178.

[3] AlTawy R., Kircanski A., Youssef A. M., “Rebound attacks on Stribog”, *IACR Cryptology ePrint Archive*, 2013, 539.

[4] Wang Z., Yu H., Wang X., “Cryptanalysis of GOST R hash function”, *IACR Cryptology ePrint Archive*, 2013, 584.

[5] Dodis Y., Ristenpart T., Shrimpton T., “Salvaging Merkle-Damgård for practical applications”, *IACR Cryptology ePrint Archive*, 2009, 177.

[6] Stam M., “Blockcipher based hashing revisited”, *IACR Cryptology ePrint Archive*, 2008, 71.

[7] AlTawy R., Youssef A. M., “Preimage attacks on reduced-round Stribog”, *IACR Cryptology ePrint Archive*, 2014, 319.

[8] AlTawy R., Youssef A. M., “Watch your constants: malicious Streebog”, *IACR Cryptology ePrint Archive*, 2014, 879.

[9] Ma B., Li B., Hao R., Li X., “Improved cryptanalysis on reduced-round GOST and Whirlpool hash function (Full version)”, *IACR Cryptology ePrint Archive*, 2014, 375.