ГОСУДАРСТВЕННЫЙ СТАНДАРТ
РЕСПУБЛИКИ БЕЛАРУСЬ

СТБ 34.101.77-2020

Информационные технологии и безопасность

КРИПТОГРАФИЧЕСКИЕ АЛГОРИТМЫ НА ОСНОВЕ SPONGE-ФУНКЦИИ

Інфармацыйныя тэхналогіі і бяспека

КРЫПТАГРАФІЧНЫЯ АЛГАРЫТМЫ НА АСНОВЕ SPONGE-ФУНКЦЫІ

Госстандарт
Минск

УДК 004.056.55(083.74)(476) МКС 35.240.40

Ключевые слова: криптографический алгоритм, sponge-функция, хэширование, шифрование, имитозащита, аутентифицированное шифрование

Предисловие

Цели, основные принципы, положения по государственному регулированию и управлению в области технического нормирования и стандартизации установлены Законом Республики Беларусь «О техническом нормировании и стандартизации».

1 РАЗРАБОТАН учреждением Белорусского государственного университета «Научно-исследовательский институт прикладных проблем математики и информатики»

ВНЕСЕН Комитетом государственной безопасности Республики Беларусь

2 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ постановлением Государственного комитета по стандартизации Республики Беларусь от 05.11.2020 г. № 64

3 ВЗАМЕН СТБ 34.101.77-2016

Издан на русском языке

Содержание

ГОСУДАРСТВЕННЫЙ СТАНДАРТ РЕСПУБЛИКИ БЕЛАРУСЬ

Информационные технологии и безопасность КРИПТОГРАФИЧЕСКИЕ АЛГОРИТМЫ НА ОСНОВЕ SPONGE-ФУНКЦИИ

Інфармацыйныя тэхналогіі і бяспека КРЫПТАГРАФІЧНЫЯ АЛГАРЫТМЫ НА АСНОВЕ SPONGE-ФУНКЦЫІ

Information technology and security
Sponge-based cryptographic algorithms

Дата введения 2021-09-01

1 Область применения

Настоящий стандарт устанавливает криптографические алгоритмы на основе sponge-функции — криптографической функции, которая задает сложное преобразование двоичных слов большой длины. Устанавливаемые алгоритмы предназначены для контроля целостности и подлинности, обеспечения конфиденциальности и других способов защиты информации при ее хранении, передаче и обработке.

Настоящий стандарт применяется при разработке средств криптографической защиты информации.

2 Нормативные ссылки

В настоящем стандарте использованы ссылки на следующие технические нормативные правовые акты в области технического нормирования и стандартизации (далее — ТНПА):

СТБ 34.101.31-2020 Информационные технологии и безопасность. Алгоритмы шифрования и контроля целостности

СТБ 34.101.45-2013 Информационные технологии и безопасность. Алгоритмы электронной цифровой подписи и транспорта ключа на основе эллиптических кривых

ГОСТ 34.973-91 (ИСО 8824-87) Информационная технология. Взаимосвязь открытых систем. Спецификация абстрактно-синтаксической нотации версии 1 (АСН.1)

Примечание — При пользовании настоящим стандартом целесообразно проверить действие ТНПА по каталогу, составленному по состоянию на 1 января текущего года, и по соответствующим информационным указателям, опубликованным в текущем году.

Если ссылочные ТНПА заменены (изменены), то при пользовании настоящим стандартом следует руководствоваться действующими взамен ТНПА. Если ссылочные ТНПА отменены без замены, то положение, в котором дана ссылка на них, применяется в части, не затрагивающей эту ссылку.

3 Термины и определения

В настоящем стандарте применяют следующие термины с соответствующими определениями:

3.1 анонс: Входные данные криптографического алгоритма, которые задают определенный контекст его использования и могут включать синхропосылку.

3.2 аутентифицированное шифрование: Одновременные шифрование и имитозащита.

3.3 зашифрование: Преобразование сообщения, направленное на обеспечение его конфиденциальности, которое выполняется с использованием ключа.

3.4 имитовставка: Двоичное слово, которое определяется по сообщению с использованием ключа и служит для контроля целостности и подлинности сообщения.

3.5 имитозащита: Контроль целостности и подлинности сообщений, который реализуется путем выработки и проверки имитовставок.

3.6 команда: Алгоритм, который связан с автоматом, принимая в качестве дополнительного входа его текущее состояние и возвращая дополнительно обновленное состояние.

3.7 конфиденциальность: Гарантия того, что сообщения доступны для понимания или использования только тем сторонам, которым они предназначены.

3.8 (криптографический) автомат: Логическое устройство, способное находиться в одном из нескольких состояний, выполнять хэширование, шифрование, имитозащиту и другие криптографические операции с помощью команд, изменяющих состояние.

3.9 октет: Двоичное слово длины 8.

3.10 подлинность: Гарантия того, что сторона действительно является владельцем, создателем или отправителем определенного сообщения.

3.11 псевдослучайные числа: Последовательность элементов, полученная в результате выполнения некоторого алгоритма и используемая в конкретном случае вместо последовательности случайных чисел.

3.12 расшифрование: Преобразование, обратное зашифрованию.

3.13 (секретный) ключ: Параметр, который управляет операциями шифрования и имитозащиты и который известен только определенным сторонам.

3.14 синхропосылка: Открытые входные данные криптографического алгоритма, которые обеспечивают уникальность результатов криптографического преобразования на фиксированном ключе.

3.15 случайные числа: Последовательность элементов, каждый из которых не может быть предсказан (вычислен) только на основе знания предшествующих ему элементов данной последовательности.

3.16 снятие защиты: Расшифрование и проверка имитовставок.

3.17 сообщение: Двоичное слово конечной длины.

3.18 установка защиты: Зашифрование и вычисление имитовставок.

3.19 хэш-значение: Двоичное слово фиксированной длины, которое определяется по сообщению без использования ключа и служит для контроля целостности сообщения и для представления сообщения в (необратимо) сжатой форме.

3.20 хэширование: Выработка хэш-значений.

3.21 целостность: Гарантия того, что сообщение не изменено при хранении или передаче.

3.22 шифрование: Зашифрование или расшифрование.

4 Обозначения и сокращения

4.1 Список обозначений и сокращений

$\perp$	специальный объект или ситуация: пустое слово, игнорируемая переменная, ошибка;
$\{0,1\}^n$	множество всех слов длины n в алфавите $\{0,1\}$;
$\{0,1\}^*$	множество всех слов конечной длины в алфавите $\{0,1\}$ (включая пустое слово длины 0);
$\lvert u\rvert$	длина слова $u \in \{0,1\}^*$;

$\{0,1\}^{n*}$	множество всех слов из $\{0,1\}^*$, длина которых кратна n;
α^n	для $\alpha \in \{0,1\}$ слово из n экземпляров α ($\alpha^0 = \perp$);
$u[i]$	для $u \in \{0,1\}^n$ и $0 \leqslant i < n$ символ u с номером i (нумерация от 0);
$u[\dots m)$	для $u \in \{0,1\}^n$ и $0 < m \leqslant n$ слово $u[0] \dots u[m-1]$;
$u[m \dots)$	для $u \in \{0,1\}^n$ и $0 \leqslant m < n$, слово $u[m]u[m+1] \dots u[n-1]$;
$u[m_1 \dots m_2)$	для $u \in \{0,1\}^n$ и $0 \leqslant m_1 < m_2 \leqslant n$ слово $u[m_1]u[m_1+1] \dots u[m_2-1]$;
$u \parallel v$	для $u \in \{0,1\}^n$ и $v \in \{0,1\}^m$ слово $w \in \{0,1\}^{n+m}$ такое, что $w[\dots n) = u$ и $w[n \dots) = v$ (конкатенация);
$\texttt{Split}(u, m)$	для $u \in \{0,1\}^*$ и натурального m представление u в виде набора $(u_1, u_2, \dots, u_n)$ фрагментов $u_i \in \{0,1\}^*$ таких, что $u = u_1 \parallel u_2 \parallel \dots \parallel u_n$, $\lvert u_1\rvert = \lvert u_2\rvert = \dots = \lvert u_{n-1}\rvert = m$ и $0 < \lvert u_n\rvert \leqslant m$, причем набор пуст ($n = 0$), если $u = \perp$;
$\texttt{01234...}_{16}$	представление $u \in \{0,1\}^{4*}$ шестнадцатеричным словом, при котором последовательным четырем символам u соответствует один шестнадцатеричный символ (например, $10100010 = \texttt{A2}_{16}$);
$U \bmod m$	для целого U и натурального m остаток от деления U на m;
$u \oplus v$	для $u, v \in \{0,1\}^n$ слово $w \in \{0,1\}^n$ из символов $w[i] = (u[i] + v[i]) \bmod 2$ (посимвольное исключающее ИЛИ);
$\neg u$	для $u \in \{0,1\}^n$ слово $u \oplus 1^n$ (посимвольное НЕ);
$u \wedge v$	для $u, v \in \{0,1\}^n$ слово $w \in \{0,1\}^n$ из символов $w[i] = u[i] * v[i]$ (посимвольное И);
$u \vee v$	для $u, v \in \{0,1\}^n$ слово $w \in \{0,1\}^n$ из символов $w[i] = (u[i] * v[i] + u[i] + v[i]) \bmod 2$ (посимвольное ИЛИ);
$\lfloor u \rceil$	а) для октета $u \in \{0,1\}^8$ число $2^7u[0] + 2^6u[1] + \dots + u[7]$, б) для $u = u_0 \parallel u_1 \parallel \dots \parallel u_{n-1}$, $u_i \in \{0,1\}^8$, число $\lfloor u_0 \rceil + 2^8 \lfloor u_1 \rceil + \dots + 2^{8(n-1)} \lfloor u_{n-1} \rceil$;
$\langle U \rangle_{8n}$	для неотрицательного целого U и натурального n слово $u \in \{0,1\}^{8n}$ такое, что $\lfloor u \rceil = U \bmod 2^{8n}$;
$\lfloor z \rfloor$	для вещественного z максимальное целое, не превосходящее z;
$\texttt{ShLo}(u)$	для $u \in \{0,1\}^{8n}$ слово $\langle \lfloor \lfloor u \rceil / 2 \rfloor \rangle_{8n}$;
$\texttt{ShHi}(u)$	для $u \in \{0,1\}^{8n}$ слово $\langle 2 \lfloor u \rceil \rangle_{8n}$;
$\varphi^r(u)$	для слова u и преобразования φ результат r-кратного действия φ на u (например, $\texttt{ShLo}^r(u)$ — результат r-кратного действия $\texttt{ShLo}$);
$\texttt{RotHi}(u)$	для $u \in \{0,1\}^{8n}$ слово $\texttt{ShHi}(u) \oplus \texttt{ShLo}^{8n-1}(u)$;
$\texttt{alg}(u_1, u_2, \dots)$	вызов алгоритма alg с входными данными $u_1, u_2, \dots$;
$\texttt{alg}[p_1, p_2, \dots]$	алгоритм alg с параметрами $p_1, p_2, \dots$;
$a \leftarrow u$	присвоение переменной a значения u;
$\min(u, v)$	минимальное из чисел u и v;
ЭЦП	электронная цифровая подпись.

4.2 Пояснения к обозначениям

4.2.1 Слова

Входными и выходными данными алгоритмов настоящего стандарта являются двоичные слова — последовательности символов алфавита $\{0, 1\}$ (битов). Биты нумеруются слева направо от нуля.

В настоящем подразделе в качестве примера рассматривается слово

$$w = 0000000100100011010001010110011111000100110101011110011010111101111.$$

В этом слове первые семь битов нулевые, затем — 1, затем — два 0 и т. д.

Двоичное слово разбивается на тетрады из четверок последовательных битов. Тетрады кодируются шестнадцатеричными символами по правилам, заданным в таблице 1.

Таблица 1

Тетрада	Символ	Тетрада	Символ	Тетрада	Символ	Тетрада	Символ
0000	$\mathtt{0}_{16}$	0001	$\mathtt{1}_{16}$	0010	$\mathtt{2}_{16}$	0011	$\mathtt{3}_{16}$
0100	$\mathtt{4}_{16}$	0101	$\mathtt{5}_{16}$	0110	$\mathtt{6}_{16}$	0111	$\mathtt{7}_{16}$
1000	$\mathtt{8}_{16}$	1001	$\mathtt{9}_{16}$	1010	$\mathtt{A}_{16}$	1011	$\mathtt{B}_{16}$
1100	$\mathtt{C}_{16}$	1101	$\mathtt{D}_{16}$	1110	$\mathtt{E}_{16}$	1111	$\mathtt{F}_{16}$

Например, слово w кодируется следующим образом:

$$\mathtt{0123456789ABCDEF}_{16}.$$

Пары последовательных тетрад образуют октеты. Последовательные октеты слова w имеют вид:

$$00000001 = \mathtt{01}_{16},\ 00100011 = \mathtt{23}_{16},\ 01000101 = \mathtt{45}_{16},\ 01100111 = \mathtt{67}_{16},$$
$$10001001 = \mathtt{89}_{16},\ 10101011 = \mathtt{AB}_{16},\ 11001101 = \mathtt{CD}_{16},\ 11101111 = \mathtt{EF}_{16}.$$

4.2.2 Слова как числа

Октету $u = u[0]u[1]\ldots u[7]$ ставится в соответствие байт — число $\lfloor u \rceil = 2^7u[0] + 2^6u[1] + \ldots + u[7]$. Например, октетам w соответствуют байты

$$1,\ 35 = 2^5 + 2^1 + 1,\ 69 = 2^6 + 2^2 + 1,\ 103 = 2^6 + 2^5 + 2^2 + 2^1 + 1,$$
$$137 = 2^7 + 2^3 + 1,\ 171 = 2^7 + 2^5 + 2^3 + 2^1 + 1,\ 205 = 2^7 + 2^6 + 2^3 + 2^2 + 1,$$
$$239 = 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^1 + 1.$$

Число ставится в соответствие не только октету, но и любому другому двоичному слову, длина которого кратна 8. При этом используется распространенное для многих современных процессоров соглашение «от младших к старшим» (little-endian): считается, что первый байт является младшим, последний — старшим. Например, слову w соответствует число

$$\lfloor w \rceil = 1 + 2^8 \cdot 35 + 2^{16} \cdot 69 + 2^{24} \cdot 103 + 2^{32} \cdot 137 + 2^{40} \cdot 171 + 2^{48} \cdot 205 + 2^{56} \cdot 239 =$$
$$= 17279655951921914625.$$

При отождествлении слов с числами удобно представить себе гипотетический регистр, разрядность которого совпадает с длиной слова. В самый правый октет регистра загружается первый октет слова, во второй справа октет регистра — второй октет слова и

т. д., пока, наконец, в самый левый октет регистра не загружается последний октет слова. Например, для w содержимое регистра имеет вид:

$$\texttt{EFCDAB8967452301}_{16} = 111011111100\ldots001100000001.$$

При таком представлении операции ShLo, ShHi, RotHi состоят в сдвигах содержимого регистра: ShLo — вправо (в сторону младших разрядов), ShHi — влево (в сторону старших разрядов) и RotHi — циклически влево, причем при сдвигах ShLo и ShHi в освободившиеся разряды регистров записываются нули. Например, предыдущий регистр изменяется при сдвигах следующим образом:

$$\begin{aligned} \texttt{ShLo}: &\ \texttt{77E6D5C4B3A29180}_{16} = 01110111\ldots10000000,\\ \texttt{ShHi}: &\ \texttt{DF9B5712CE8A4602}_{16} = 11011111\ldots00000010,\\ \texttt{RotHi}: &\ \texttt{DF9B5712CE8A4603}_{16} = 11011111\ldots00000011. \end{aligned}$$

Выгружая из регистра октеты слева направо, получаем следующие результаты:

$$\begin{aligned} \texttt{ShLo}(w) = &\ \texttt{8091A2B3C4D5E677}_{16},\\ \texttt{ShHi}(w) = &\ \texttt{02468ACE12579BDF}_{16},\\ \texttt{RotHi}(w) = &\ \texttt{03468ACE12579BDF}_{16}. \end{aligned}$$

Перестановки октетов при загрузке слова в регистр и при выгрузке из регистра в современных процессорах выполняются неявно.

4.3 Запись перечислений

При записи последовательности $u_1, u_2, \ldots, u_n$ допускается, если не оговорено иное, выполнение неравенства $n < 2$. При $n = 0$ идет речь о пустой последовательности, а при $n = 1$ — об одноэлементной последовательности u_1.

Аналогичные соглашения распространяются на запись конкатенации нескольких слов, итератора цикла. Например:

– слово $u_1 \parallel u_2 \parallel \ldots \parallel u_n$ является пустым при $n = 0$ и состоит из единственного фрагмента u_1 при $n = 1$;

– тело цикла «для $i = 1, 2, \ldots, n$ выполнить $\ldots$» не выполняется ни разу, если $n = 0$, и выполняется один раз, если $n = 1$.

5 Общие положения

5.1 Назначение

Настоящий стандарт определяет семейство криптографических алгоритмов, построенных по схеме sponge (губка). Схема предложена в [1]. Ядром схемы является sponge-функция, которая определяет сложное биективное преобразование двоичных слов большой длины.

Криптографические алгоритмы делятся на две группы:

1) алгоритмы хэширования (см. раздел 7);

2) программируемые алгоритмы (см. раздел 8).

Алгоритм хэширования по сообщению произвольной длины строит хэш-значение — слово фиксированной длины. Стороны могут организовать контроль целостности сообщений путем сравнения их хэш-значений с достоверными контрольными хэш-значениями. Изменение сообщения с высокой вероятностью приводит к изменению соответствующего хэш-значения, и поэтому хэш-значения могут использоваться вместо самих сообщений, например в системах ЭЦП.

Алгоритмы хэширования могут дополнительно использоваться при построении самих систем ЭЦП, генераторов случайных и псевдослучайных чисел, протоколов аутентификации, криптографических аккумуляторов, доказательств вычислительной работы и др.

Программируемые алгоритмы представляют собой последовательности команд криптографического автомата. Разные последовательности дают алгоритмы разного назначения: шифрования, имитозащиты, генерации псевдослучайных чисел, того же хэширования. В настоящем стандарте объявляются правила безопасного программирования (см. 8.11), явно определяются программируемые алгоритмы хэширования (см. 8.12) и аутентифицированного шифрования (см. 8.13).

Программируемые алгоритмы хэширования обладают большей гибкостью, чем базовые: с их помощью можно вычислять хэш-значения произвольной длины, а при определенной настройке можно повысить скорость хэширования за счет уравнивания гарантий безопасности относительно различных атак. Кроме этого, программируемый алгоритм дополнительно принимает на вход анонс, с помощью которого можно уточнить контекст хэширования.

Алгоритмы аутентифицированного шифрования предназначены для обеспечения конфиденциальности, контроля целостности и подлинности данных. Стороны, располагающие общим ключом, могут организовать конфиденциальный обмен сообщениями путем их зашифрования перед отправкой и расшифрования после получения. Кроме этого, стороны могут организовать контроль целостности при обмене сообщениями путем добавления к ним имитовставок при отправке и проверки имитовставок при получении. Проверка имитовставок дополнительно позволяет стороне-получателю убедиться в том, что сторона-отправитель знает ключ, т. е. позволяет проверить подлинность сообщений.

Алгоритмы аутентифицированного шифрования обеспечивают гибкую функциональность. Во-первых, вместе с сообщением контролируется целостность и подлинность ассоциированных открытых данных, которые не зашифровываются. Во-вторых, части сообщения и ассоциированных данных могут чередоваться. В-третьих, имитовставки могут выдаваться несколько раз по мере обработки частей. В-четвертых, в процессе обработки данных могут обновляться ключи.

Примечание — Схожие по назначению алгоритмы установлены в СТБ 34.101.31. Алгоритмы настоящего стандарта дают дополнительные функциональные возможности. Например, алгоритмы хэширования поддерживают все три уровня стойкости алгоритмов ЭЦП, определенных в СТБ 34.101.45, в то время как алгоритм хэширования СТБ 34.101.31 поддерживает только первый уровень. Переход от алгоритмов СТБ 34.101.31 к алгоритмам настоящего стандарта позволит увеличить скорость обработки данных по крайней мере на паритетном уровне стойкости на 64-разрядных аппаратных платформах.

Примеры выполнения алгоритмов стандарта приведены в приложении А. Примеры можно использовать для проверки корректности реализаций алгоритмов.

Модуль абстрактно-синтаксической нотации версии 1 (АСН.1), определенной в ГОСТ 34.973, приведен в приложении Б. Модуль задает идентификаторы алгоритмов стандарта, в том числе в их связках с алгоритмами СТБ 34.101.45. Рекомендуется использовать модуль при встраивании алгоритмов в информационные системы, в которых также используется АСН.1.

5.2 Sponge-функция

Sponge-функция настоящего стандарта действует на двоичные слова длины 1536. Действие задается алгоритмом `bash-f`, определенным в 6.2.

Sponge-функция `bash-f` лежит в основе криптографического автомата, с помощью которого определяются программируемые алгоритмы.

Sponge-функция может использоваться за пределами настоящего стандарта для построения других криптографических алгоритмов.

5.3 Уровень стойкости

Алгоритмы настоящего стандарта отличаются уровнем стойкости ℓ.

В алгоритмах хэширования это натуральное число, кратное 16 и не превосходящее 256. Уровни $\ell = 128$, $\ell = 192$ и $\ell = 256$ являются стандартными, им следует отдавать предпочтение.

В программируемых алгоритмах уровень стойкости ℓ может принимать только стандартные значения.

С ростом ℓ повышается стойкость алгоритмов и одновременно снижается их быстродействие. В частности, хэширование на уровне $\ell = 256$ выполняется примерно в 2 раза медленнее, чем при $\ell = 128$.

5.4 Емкость

В программируемых алгоритмах уровень стойкости ℓ сопровождается емкостью $d \in \{1, 2\}$. С увеличением емкости повышаются гарантии безопасности, но снижается производительность. Выбор $(\ell, d) = (256, 2)$ обеспечивает максимальные гарантии. Выбор $(\ell, d) = (128, 1)$ дает максимальную производительность.

В алгоритмах аутентифицированного шифрования речь идет о гарантиях, связанных с объемом данных, которые разрешается обработать без смены или обновления ключа. При $d = 1$ суммарная длина входных и выходных данных не должна превышать $2^{\ell/2}$. При $d = 2$ ограничений на объем нет.

В программируемых алгоритмах хэширования в зависимости от емкости меняется соотношение между трудоемкостями основных атак (см. таблицу 2). При этом минимальная трудоемкость остается без изменений.

5.5 Хэш-значение

Базовый алгоритм хэширования уровня ℓ вычисляет хэш-значения — двоичные слова длины 2ℓ. Если при выбранном ℓ требуются не все, а $n < 2\ell$ символов хэш-значения, то должны использоваться первые n символов.

Программируемый алгоритм хэширования вычисляет хэш-значения произвольной заданной длины n.

Таблица 2 — Трудоемкость атак на алгоритмы хэширования

Алгоритм	Обращение	Второй прообраз	Коллизия
Базовый ($n \leqslant 2\ell$)	n	n	$n/2$
Программируемый ($d = 1$)	$\min(n, \ell)$	$\min(n, \ell)$	$\min(n/2, \ell)$
Программируемый ($d = 2$)	$\min(n, 2\ell)$	$\min(n, 2\ell)$	$\min(n/2, \ell)$

При выборе (ℓ, n) следует учитывать данные таблицы 2. В таблице приводится трудоемкость атак по обращению (найти сообщение с заданным хэш-значением), построению второго прообраза (найти другое сообщение с тем же хэш-значением) и построению коллизии (найти различные сообщения с одинаковыми хэш-значениями). Трудоемкость представлена в таблице в логарифмической форме: для осуществления атаки трудоемкости u требуется выполнить порядка 2^u операций.

5.6 Ключ

В программируемых алгоритмах может использоваться ключ. Это двоичное слово, длина которого кратна 32, не меньше уровня стойкости ℓ и не превосходит 480.

Ключ длины ℓ должен вырабатываться без возможности предсказания, распространяться с соблюдением мер конфиденциальности и храниться в секрете. Ключ большей длины может быть избыточным кодом ℓ-битового ключа, например, дополнительно содержать контрольные биты.

Сразу после загрузки ключа автомат программируемого алгоритма переходит в ключевой режим. В этом режиме автомат может выполнять шифрование, аутентифицированное шифрование, имитозащиту, генерировать псевдослучайные числа для создания секретных объектов.

Пока ключ не введен, автомат остается в бесключевом режиме. Здесь можно только хэшировать данные.

5.7 Имитовставка

С помощью программируемых алгоритмов можно вычислять имитовставки. Это данные, которые выдает автомат программируемого алгоритма, причем автомат переведен в ключевой режим.

В алгоритмах аутентифицированного шифрования длина имитовставки совпадает с уровнем стойкости ℓ.

В других алгоритмах могут вычисляться имитовставки сколь угодно большой длины или даже последовательности имитовставок. Имитовставки могут интерпретироваться как псевдослучайные числа (биты), а автомат — как их генератор.

Псевдослучайные числа являются материалом для создания синхропосылок, ключей, других криптографических объектов. Ключи и другие секретные объекты должны строиться по непересекающимся фрагментам последовательностей псевдослучайных чисел. Если генерируемый ключ используется для управления другим автоматом, то уровень стойкости целевого автомата не должен быть выше уровня стойкости генерирующего.

5.8 Анонс

С помощью анонса можно уточнить контекст использования программируемого алгоритма — сузить назначение, локализовать, задать настройки. Например, один анонс может описывать исходящий трафик сети A, второй — внутренний трафик сети B. Алгоритмы разных контекстов, то есть с разными анонсами, по-разному обрабатывают одни и те же данные.

Анонс представляет собой двоичное слово, длина которого кратна 32 и не превосходит 480. Допускаются пустые анонсы.

Анонс указывается в командах инициализации и повторной инициализации автомата программируемого алгоритма.

5.9 Синхропосылка

В алгоритмах аутентифицированного шифрования анонс может содержать синхропосылку.

Синхропосылки, которые используются с одним и тем же ключом для шифрования различных сообщений, должны быть уникальны: на уровне ℓ они могут повторяться только с вероятностью, близкой к $2^{-\ell}$. Например, если синхропосылка строится как случайное

или псевдослучайное число и с одним ключом планируется использовать q синхропосылок длины m, то должно выполняться ограничение $q^2 \cdot 2^\ell \leqslant 2^{m+1}$.

Синхропосылки могут быть неявными: в их качестве могут выступать волатильные открытые данные, которые обрабатываются до шифрования сообщений.

Более того, синхропосылки могут вообще не использоваться, если для автомата алгоритма аутентифицированного шифрования соблюдается принцип: последовательность команд, выполненная до шифрования, уникальна — любые две последовательности отличаются либо командами, либо их входными данными. Принцип заведомо соблюден, если в автомат загружается одноразовый ключ.

Синхропосылка необязательна, если шифрование не предполагается, а речь идет только об имитозащите.

При генерации псевдослучайных чисел, предназначенных для построения ключей и других секретных объектов, действуют те же требования к синхропосылкам, что и при шифровании.

5.10 Интерфейсы

Определение алгоритма или группы алгоритмов начинается с назначения им коротких имен, описания параметров и соглашений о входных и выходных данных. В совокупности такая вводная информация называется интерфейсом. Например, алгоритму хэширования, определенному в разделе 7, назначается имя `bash-hash`, объявляется, что его параметром является уровень стойкости ℓ. Алгоритм, полученный из основного определенной настройкой параметров, считается самостоятельным алгоритмом. Так, самостоятельным является алгоритм `bash-hash`$[\ell]$. В интерфейсе объявляется, что его входными данными является хэшируемое слово $X \in \{0,1\}^*$, выходными — хэш-значение $Y \in \{0,1\}^{2\ell}$.

С помощью интерфейсов можно лаконично и однозначно описывать вызов одних алгоритмов в других. Например, вызов алгоритма `bash-hash` уровня ℓ записывается как $Y \leftarrow$ `bash-hash`$[\ell](X)$.

Алгоритм может вызываться в других алгоритмах настоящего стандарта или в алгоритмах других стандартов. И наоборот, алгоритм может вызывать другие алгоритмы. В последнем случае интерфейс содержит перечень задействованных алгоритмов.

5.11 Переменные

В настоящем стандарте переменные алгоритма, которые явно объявляются перед определением его шагов, могут содержать критические данные, например, фрагмент ключа или промежуточный результат вычислений, который упрощает определение этого фрагмента. Речь идет в том числе о переменных алгоритма хэширования, который может использоваться для обработки критических данных.

При реализации алгоритма объявленные переменные следует очищать после использования. Очистке подлежит в том числе автомат (точнее, его состояние), задействованный в программируемом алгоритме. При организации очистки необходимо учитывать особенности реализации, например ситуации, когда переменная алгоритма представляется несколькими переменными реализации.

Очистка переменных алгоритма может не выполняться, если алгоритм все-таки не обрабатывает и не возвращает критические данные. Переменные могут также не очищаться,

если алгоритм выполняется в защищенной среде, и доступ к переменным блокируется аппаратными или другими способами.

6 Sponge-функция

6.1 Алгоритм bash-s

6.1.1 Интерфейс

Значения sponge-функции вычисляются с помощью алгоритма bash-s.

Входными данными bash-s являются слова $W_0, W_1, W_2 \in \{0,1\}^{64}$ и числа $m_1, n_1, m_2, n_2 \in \{1, 2, \ldots, 63\}$.

Выходными данными являются преобразованные слова W_0, W_1, W_2.

6.1.2 Переменные

Используются переменные $T_0, T_1, T_2 \in \{0,1\}^{64}$.

6.1.3 Алгоритм

Вычисление $\texttt{bash-s}(W_0, W_1, W_2, m_1, n_1, m_2, n_2)$ выполняется следующим образом:

1 Установить $T_0 \leftarrow \texttt{RotHi}^{m_1}(W_0)$.
2 Установить $W_0 \leftarrow W_0 \oplus W_1 \oplus W_2$.
3 Установить $T_1 \leftarrow W_1 \oplus \texttt{RotHi}^{n_1}(W_0)$.
4 Установить $W_1 \leftarrow T_0 \oplus T_1$.
5 Установить $W_2 \leftarrow W_2 \oplus \texttt{RotHi}^{m_2}(W_2) \oplus \texttt{RotHi}^{n_2}(T_1)$.
6 Установить $T_0 \leftarrow \neg W_2$.
7 Установить $T_1 \leftarrow W_0 \vee W_2$.
8 Установить $T_2 \leftarrow W_0 \wedge W_1$.
9 Установить $T_0 \leftarrow T_0 \vee W_1$.
10 Установить $W_1 \leftarrow W_1 \oplus T_1$.
11 Установить $W_2 \leftarrow W_2 \oplus T_2$.
12 Установить $W_0 \leftarrow W_0 \oplus T_0$.
13 Возвратить (W_0, W_1, W_2).

6.2 Алгоритм bash-f

6.2.1 Интерфейс

Sponge-функция задается алгоритмом bash-f.

Входными данными bash-f является слово $S \in \{0,1\}^{1536}$, аргумент sponge-функции.

Выходными данными является преобразованное слово S, значение sponge-функции.

Используется алгоритм bash-s, определенный в 6.1.

6.2.2 Алгоритм

Вычисление $\texttt{bash-f}(S)$ выполняется следующим образом:

1 Определить $(S_0, S_1, \ldots, S_{23}) = \texttt{Split}(S, 64)$.
2 Установить $C \leftarrow \texttt{B194BAC80A08F53B}_{16}$.
3 Для $i = 1, 2, \ldots, 24$ выполнить:
  1) $(m_1, n_1, m_2, n_2) \leftarrow (8, 53, 14, 1)$;
  2) для $j = 0, 1, \ldots, 7$:
    (a) $(S_j, S_{8+j}, S_{16+j}) \leftarrow \texttt{bash-s}(S_j, S_{8+j}, S_{16+j}, m_1, n_1, m_2, n_2)$;
    (b) $(m_1, n_1, m_2, n_2) \leftarrow (7\,m_1 \bmod 64, 7\,n_1 \bmod 64, 7\,m_2 \bmod 64, 7\,n_2 \bmod 64)$;

3) $S \leftarrow S_{15} \| S_{10} \| S_9 \| S_{12} \| S_{11} \| S_{14} \| S_{13} \| S_8 \| S_{17} \| S_{16} \| S_{19} \| S_{18} \| S_{21} \| S_{20} \| S_{23} \| S_{22} \| S_6 \| S_3 \| S_0 \| S_5 \| S_2 \| S_7 \| S_4 \| S_1$;
4) $S_{23} \leftarrow S_{23} \oplus C$;
5) если $\lfloor C \rceil$ — четное, то $C \leftarrow \mathtt{ShLo}(C)$;
иначе $C \leftarrow \mathtt{ShLo}(C) \oplus \mathtt{AED8E07F99E12BDC}_{16}$.

4 Возвратить S.

7 Алгоритмы хэширования

7.1 Интерфейс

Хэширование задается алгоритмом `bash-hash`. Параметром алгоритма является уровень стойкости $\ell \in \{16, 32, \ldots, 256\}$ (см. 5.3).

Входными данными $\mathtt{bash\text{-}hash}[\ell]$ является хэшируемое сообщение $X \in \{0,1\}^*$.

Выходными данными является хэш-значение $Y \in \{0,1\}^{2\ell}$.

Используется алгоритм `bash-f`, определенный в 6.2.

7.2 Вспомогательные переменные

Используется переменная $S \in \{0,1\}^{1536}$.

7.3 Алгоритм

Хэширование $\mathtt{bash\text{-}hash}[\ell](X)$ выполняется следующим образом:

1 Определить $(X_1, X_2, \ldots, X_n) = \mathtt{Split}(X \| 01, 1536 - 4\ell)$.
2 Установить $X_n \leftarrow X_n \| 0^{1536-4\ell-|X_n|}$.
3 Установить $S \leftarrow 0^{1472} \| \langle \ell/4 \rangle_{64}$.
4 Для $i = 1, 2, \ldots, n$ выполнить:
1) $S[\ldots 1536 - 4\ell) \leftarrow X_i$;
2) $S \leftarrow \mathtt{bash\text{-}f}(S)$.
5 Установить $Y \leftarrow S[\ldots 2\ell)$.
6 Возвратить Y.

8 Программируемые алгоритмы

8.1 Автомат

Автомат, который лежит в основе программируемых алгоритмов, характеризуется состоянием и управляется командами. Основные команды состоят в загрузке и (или) выгрузке данных. Дополнительные служебные команды управляют состоянием автомата — инициализируют его или необратимо меняют.

На вход автомата могут поступать анонсы, ключи, открытые данные или данные, которые требуется зашифровать или расшифровать. Выгружаться могут зашифрованные или расшифрованные данные, хэш-значения, имитовставки, псевдослучайные числа. Выходные данные могут использоваться в качестве входных для другого автомата.

В таблице 3 описаны допустимые типы данных. Типам назначены 6-битовые коды.

Перечень допустимых команд автомата:

– `start` (инициализировать);
– `restart` (повторно инициализировать);
– `absorb` (загрузить данные);
– `squeeze` (выгрузить данные);
– `encrypt` (зашифровать);

Таблица 3 — Типы данных автомата

Тип	Описание	Код
`NULL`	Служебные данные	000000
`KEY`	Ключ	000001
`DATA`	Открытые входные данные	000010
`TEXT`	Сообщения для зашифрования или расшифрования	000011
`OUT`	Выходные данные	000100

– `decrypt` (расшифровать);
– `ratchet` (необратимо изменить состояние).

Команды выполняются в отложенной манере: выполнение текущей команды завершается в начале выполнения следующей. Для завершения используется внутренняя команда `commit`, которую можно вызывать только из других команд.

Команды определяются в 8.3–8.10. Каждая команда имеет полный доступ к состоянию автомата. Состояние исключается из входных и выходных данных команд.

8.2 Состояние

Состояние автомата представляет собой слово $S \in \{0,1\}^{1536}$.

Оно сопровождается следующими параметрами:
– $\ell \in \{128, 192, 256\}$ — уровень стойкости;
– $d \in \{1, 2\}$ — емкость;
– $r \in \{1536 - 2d\ell, 1536 - \ell - d\ell/2\}$ — длина буфера (см. далее);
– $pos \in \{0, 1, \ldots, r-1\}$ — текущее смещение в буфере.

Эти параметры устанавливаются или используются в командах, хотя и не указываются в качестве входных и выходных данных. Параметры ℓ и d устанавливаются при инициализации автомата и после этого не меняются.

Состояние S разбивается на 2 части:

1 Буфер $S[\ldots r)$. В эту часть записываются блоки загружаемых данных и по ней определяются блоки выгружаемых.

2 Память $S[r\ldots)$. Биты памяти никогда не выгружаются из состояния и меняются только путем применения sponge-функции к предыдущему состоянию. Бит $S[r]$ имеет особый статус: его дополнительно инвертирует команда `commit`, фиксируя завершение выполнения предыдущей команды.

8.3 Команда `start`

8.3.1 Интерфейс

Параметрами команды `start` являются уровень стойкости $\ell \in \{128, 192, 256\}$ и емкость $d \in \{1, 2\}$.

Входными данными `start`$[\ell, d]$ являются анонс $A \in \{0,1\}^{32*}$ и ключ $K \in \{0,1\}^{32*}$. Длины A и K не должны превосходить 480. Если $K \neq \perp$, то $|K| \geqslant \ell$.

Выходные данные отсутствуют.

8.3.2 Алгоритм

Команда `start`$[\ell, d](A, K)$ выполняется следующим образом:

1 Если $K \neq \perp$, то $r \leftarrow 1536 - \ell - d\ell/2$.

2 Иначе, если $K = \perp$, то $r \leftarrow 1536 - 2d\ell$.

3 Установить $pos \leftarrow 8 + |A| + |K|$.

4 Установить $S[\dots pos) \leftarrow \langle |A|/2 + |K|/32 \rangle_8 \parallel A \parallel K$.
5 Установить $S[pos \dots 1472) \leftarrow 0^{1472-pos}$.
6 Установить $S[1472 \dots) \leftarrow \langle \ell/4 + d \rangle_{64}$.
7 Сохранить параметры (ℓ, d, r, pos) вместе с состоянием S.

8.4 Команда `commit`

8.4.1 Интерфейс

Входными данными команды `commit` является слово $t \in \{0,1\}^6$, код из последнего столбца таблицы 3.

Выходные данные отсутствуют.

Используется алгоритм `bash-f`, определенный в 6.2.

8.4.2 Алгоритм

Команда `commit`(t) выполняется следующим образом:
1 Установить $S[pos \dots pos + 8) \leftarrow S[pos \dots pos + 8) \oplus (t \parallel 01)$.
2 Установить $S[r] \leftarrow S[r] \oplus 1$.
3 Установить $S \leftarrow$ `bash-f`(S).
4 Установить $pos \leftarrow 0$.

8.5 Команда `restart`

8.5.1 Интерфейс

Входными данными команды `restart` являются анонс $A \in \{0,1\}^{32*}$ и ключ $K \in \{0,1\}^{32*}$. Длины A и K не должны превосходить 480. Если $K \neq \perp$, то $|K| \geqslant \ell$.

Выходные данные отсутствуют.

8.5.2 Алгоритм

Команда `restart`(A, K) выполняется следующим образом:
1 Если $K \neq \perp$:
  1) `commit`(`KEY`);
  2) $r \leftarrow 1536 - \ell - d\ell/2$.
2 Иначе, если $K = \perp$:
  1) `commit`(`NULL`).
3 Установить $pos \leftarrow 8 + |A| + |K|$.
4 Установить $S[\dots pos) \leftarrow S[\dots pos) \oplus (\langle |A|/2 + |K|/32 \rangle_8 \parallel A \parallel K)$.

8.6 Команда `absorb`

8.6.1 Интерфейс

Входными данными команды `absorb` является слово $X \in \{0,1\}^{8*}$.

Выходные данные отсутствуют.

Используется алгоритм `bash-f`, определенный в 6.2.

8.6.2 Алгоритм

Команда `absorb`(X) выполняется следующим образом:
1 Выполнить `commit`(`DATA`).
2 Определить $(X_1, X_2, \dots, X_n) =$ `Split`(X, r).
3 Для $i = 1, 2, \dots, n$:
  1) $pos \leftarrow |X_i|$;

2) $S[\ldots pos) \leftarrow S[\ldots pos) \oplus X_i$;
3) если $pos = r$, то $S \leftarrow \texttt{bash-f}(S)$ и $pos \leftarrow 0$.

8.7 Команда squeeze

8.7.1 Интерфейс

Входными данными команды `squeeze` является длина выхода n. Число n должно быть неотрицательным целым, кратным 8.

Выходными данными является слово $Y \in \{0,1\}^n$.

Используется алгоритм `bash-f`, определенный в 6.2.

8.7.2 Алгоритм

Команда $\texttt{squeeze}(n)$ выполняется следующим образом:

1 Выполнить $\texttt{commit}(\texttt{OUT})$.
2 Установить $Y \leftarrow \perp$.
3 Пока $|Y| + r \leqslant n$:
  1) $Y \leftarrow Y \parallel S[\ldots r)$;
  2) $S \leftarrow \texttt{bash-f}(S)$.
4 Установить $pos \leftarrow n - |Y|$.
5 Установить $Y \leftarrow Y \parallel S[\ldots pos)$.
6 Возвратить Y.

8.8 Команда encrypt

8.8.1 Интерфейс

Входными данными команды `encrypt` является сообщение $X \in \{0,1\}^{8*}$.

Выходными данными является зашифрованное сообщение $Y \in \{0,1\}^{|X|}$.

Используется алгоритм `bash-f`, определенный в 6.2.

8.8.2 Алгоритм

Команда $\texttt{encrypt}(X)$ выполняется следующим образом:

1 Выполнить $\texttt{commit}(\texttt{TEXT})$.
2 Определить $(X_1, X_2, \ldots, X_n) = \texttt{Split}(X, r)$.
3 Установить $Y \leftarrow \perp$.
4 Для $i = 1, 2, \ldots, n$:
  1) $pos \leftarrow |X_i|$;
  2) $S[\ldots pos) \leftarrow S[\ldots pos) \oplus X_i$;
  3) $Y \leftarrow Y \parallel S[\ldots pos)$;
  4) если $pos = r$, то $S \leftarrow \texttt{bash-f}(S)$ и $pos \leftarrow 0$.
5 Возвратить Y.

8.9 Команда decrypt

8.9.1 Интерфейс

Входными данными команды `decrypt` является зашифрованное сообщение $Y \in \{0,1\}^{8*}$.

Выходными данными является расшифрованное сообщение $X \in \{0,1\}^{|Y|}$.

Используется алгоритм `bash-f`, определенный в 6.2.

8.9.2 Алгоритм

Команда `decrypt`(Y) выполняется следующим образом:

1 Выполнить `commit`(`TEXT`).
2 Определить $(Y_1, Y_2, \ldots, Y_n) = \texttt{Split}(Y, r)$.
3 Установить $X \leftarrow \perp$.
4 Для $i = 1, 2, \ldots, n$:
   1) $pos \leftarrow |Y_i|$;
   2) $X \leftarrow X \parallel \big(S[\ldots pos) \oplus Y_i\big)$;
   3) $S[\ldots pos) \leftarrow Y_i$;
   4) если $pos = r$, то $S \leftarrow \texttt{bash-f}(S)$ и $pos \leftarrow 0$.
5 Возвратить X.

8.10 Команда `ratchet`

8.10.1 Интерфейс

Команда `ratchet` изменяет состояние автомата, не принимая и не возвращая данные.

8.10.2 Переменные

Используется переменная $T \in \{0,1\}^{1536}$.

8.10.3 Алгоритм

Команда `ratchet`$(\perp)$ выполняется следующим образом:

1 Установить $T \leftarrow S$.
2 Выполнить `commit`(`NULL`).
3 Установить $S \leftarrow S \oplus T$.

8.11 Программирование

Работа автомата начинается с вызова команды `start`, через которую загружаются анонс A и ключ K. Позже с помощью команды `restart` можно загрузить новые пары (A, K).

Если в `start` и во всех вызовах `restart` ключ был пустым словом, то есть не загружался, то автомат находится в бесключевом режиме. В этом режиме:

– запрещено вызывать команду `encrypt` (и соответственно `decrypt`);

– данные, возвращаемые `squeeze`, могут использоваться только в качестве хэш-значений.

В ключевом режиме синхропосылка загружается явно, как часть анонса, с помощью `start` / `restart` или неявно, как волатильные открытые данные, с помощью `absorb`.

Команда `ratchet` вызывается для того, чтобы затруднить возврат к предыдущим состояниям автомата. Даже если состояние автомата после вызова `ratchet` раскрыто, криптографические операции до вызова сохраняют надежность. В частности, затруднено расшифрование ранее зашифрованных данных, то есть обеспечивается защита от «чтения назад».

В записи программируемого алгоритма фигурирует один или несколько автоматов. Автоматам назначаются произвольные имена. Команда автомата указывается после имени автомата и точки.

Пример программы с автоматами α, β и γ:

1 Выполнить α.`start`$[256, 2](\perp, K)$.
2 Выполнить α.`absorb`(I).

3 Выполнить α.`ratchet`($\perp$).
4 Вычислить $K_1 \leftarrow \alpha$.`squeeze`(128).
5 Выполнить β.`start`$[128, 1](A_1, K_1)$.
6 Установить $\gamma \leftarrow \beta$.
7 Выполнить γ.`restart`$(A_2, \perp)$.
8 Вычислить $Y_1 \leftarrow \beta$.`encrypt`(X).
9 Вычислить $Y_2 \leftarrow \gamma$.`encrypt`(X).

Присваивание $\gamma \leftarrow \beta$ означает, что в γ переписываются состояние и сопровождающие параметры β. После присваивания автоматы становятся эквивалентными. Эквивалентные автоматы функционируют одинаково, в частности, одинаково зашифровывают данные. Это недопустимо по соображениям безопасности. В примере эквивалентность разрушается вызовом команды `restart`.

8.12 Хэширование

8.12.1 Интерфейс

Хэширование задается алгоритмом `bash-prg-hash`. Параметрами алгоритма являются уровень стойкости $\ell \in \{128, 192, 256\}$ и емкость $d \in \{1, 2\}$.

Входными данными `bash-prg-hash`$[\ell, d]$ являются анонс $A \in \{0,1\}^{32*}$, хэшируемое сообщение $X \in \{0,1\}^{8*}$ и длина хэш-значения n. Длина A не должна превосходить 480.

Выходными данными является хэш-значение $Y \in \{0,1\}^n$.

8.12.2 Переменные

Используется автомат α.

8.12.3 Алгоритм

Хэширование `bash-prg-hash`$[\ell, d](A, X, n)$ выполняется следующим образом:
1 Выполнить α.`start`$[\ell, d](A, \perp)$.
2 Выполнить α.`absorb`(X).
3 Вычислить $Y \leftarrow \alpha$.`squeeze`(n).
4 Возвратить Y.

8.13 Аутентифицированное шифрование

8.13.1 Интерфейс

Аутентифицированное шифрование задается алгоритмом установки защиты `bash-prg-ae` и алгоритмом снятия защиты `bash-prg-ae`$^{-1}$. Параметрами алгоритмов являются уровень стойкости $\ell \in \{128, 192, 256\}$ и емкость $d \in \{1, 2\}$.

Входными данными `bash-prg-ae`$[\ell, d]$ являются анонс $A \in \{0,1\}^{32*}$, сообщение $X \in \{0,1\}^{8*}$, ассоциированные данные $I \in \{0,1\}^{8*}$ и ключ $K \in \{0,1\}^{32*}$. Длины A и K должны быть не больше 480, длина K должна быть не меньше ℓ.

Выходными данными `bash-prg-ae`$[\ell, d]$ являются зашифрованное сообщение $Y \in \{0,1\}^{|X|}$ и имитовставка $T \in \{0,1\}^{\ell}$.

Входными данными `bash-prg-ae`$^{-1}[\ell, d]$ являются анонс $A \in \{0,1\}^{32*}$, зашифрованное сообщение $Y \in \{0,1\}^{8*}$, ассоциированные данные $I \in \{0,1\}^{8*}$, имитовставка $T \in \{0,1\}^{\ell}$ и ключ $K \in \{0,1\}^{32*}$. Длины A и K должны быть не больше 480, длина K должна быть не меньше ℓ.

Выходными данными $\texttt{bash-prg-ae}^{-1}[\ell, d]$ являются либо признак ошибки $\perp$, либо расшифрованное сообщение $X \in \{0,1\}^{|Y|}$. Возврат $\perp$ означает нарушение целостности входных данных.

8.13.2 Переменные

В обоих алгоритмах используется автомат α.

В алгоритме снятия защиты расшифрованное сообщение X считается переменной вплоть до проверки имитовставки (шаг 4). Если имитовставка признана некорректной, то переменная должна быть очищена в соответствии с правилами 5.11.

8.13.3 Алгоритм установки защиты

Установка защиты $\texttt{bash-prg-ae}[\ell, d](A, X, I, K)$ выполняется следующим образом:

1 Выполнить $\alpha.\texttt{start}[\ell, d](A, K)$.

2 Обработать (X, I):

   1) $\alpha.\texttt{absorb}(I)$;

   2) $Y \leftarrow \alpha.\texttt{encrypt}(X)$;

   3) $T \leftarrow \alpha.\texttt{squeeze}(\ell)$;

   4) возвратить (Y, T).

Примечание 1 — Если $I = \perp$, то шаг 2.1 может быть пропущен. Если $X = \perp$, то шаг 2.2 может быть упрощен: $Y \leftarrow \perp$. Пропуск и (или) упрощение шагов должны зеркально повторяться в алгоритме снятия защиты (см. примечание 5).

Примечание 2 — После возврата (Y, T) работу с автоматом можно продолжить и обработать новую пару (X, I), повторяя шаг 2.

Примечание 3 — Перед обработкой новой пары (X, I) можно ввести новые анонс A и ключ K с помощью команды $\alpha.\texttt{restart}(A, K)$.

Примечание 4 — После обработки (X, I) можно вызвать команду $\alpha.\texttt{ratchet}(\perp)$. Вызов блокирует определение X по (Y, I) даже после раскрытия состояния автомата.

8.13.4 Алгоритм снятия защиты

Снятие защиты $\texttt{bash-prg-ae}^{-1}[\ell, d](A, Y, I, T, K)$ выполняется следующим образом:

1 Выполнить $\alpha.\texttt{start}[\ell, d](A, K)$.

2 Обработать (Y, I, T):

   1) $\alpha.\texttt{absorb}(I)$;

   2) $X \leftarrow \alpha.\texttt{decrypt}(Y)$;

   3) если $T \neq \alpha.\texttt{squeeze}(\ell)$, то возвратить $\perp$;

   4) возвратить X.

Примечание 5 — Расширения алгоритма установки защиты в соответствии с примечаниями 1–4 зеркально переносятся на алгоритм снятия защиты с одним уточнением: после возврата $\perp$ обработка новых троек (Y, I, T) должна быть прекращена.

Приложение А
(справочное)
Проверочные примеры

А.1 Алгоритм bash-s

В таблице А.1 представлен пример выполнения алгоритма `bash-s` с входными параметрами $(m_1, n_1, m_2, n_2) = (8, 53, 14, 1)$.

Таблица А.1 — Алгоритм `bash-s`

Шаг	Слово	Вычисляется как	Значение
	W_0		$\texttt{B194BAC80A08F53B}_{16}$
	W_1		$\texttt{E12BDC1AE28257EC}_{16}$
	W_2		$\texttt{E9DEE72C8F0C0FA6}_{16}$
1	T_0	$\texttt{RotHi}^{m_1}(W_0)$	$\texttt{3BB194BAC80A08F5}_{16}$
2	W_0	$W_0 \oplus W_1 \oplus W_2$	$\texttt{B96181FE6786AD71}_{16}$
3	T_1	$W_1 \oplus \texttt{RotHi}^{n_1}(W_0)$	$\texttt{CDFB23D652B779DB}_{16}$
4	W_1	$T_0 \oplus T_1$	$\texttt{F64AB76C9ABD712E}_{16}$
5	W_2	$W_2 \oplus \texttt{RotHi}^{m_2}(W_2) \oplus \texttt{RotHi}^{n_2}(T_1)$	$\texttt{F1401A7713A9DFD3}_{16}$
6	T_0	$\neg W_2$	$\texttt{0EBFE588EC56202C}_{16}$
7	T_1	$W_0 \vee W_2$	$\texttt{F9619BFF77AFFFF3}_{16}$
8	T_2	$W_0 \wedge W_1$	$\texttt{B040816C02842120}_{16}$
9	T_0	$T_0 \vee W_1$	$\texttt{FEFFF7ECFEFF712E}_{16}$
10	W_1	$W_1 \oplus T_1$	$\texttt{0F2B2C93ED128EDD}_{16}$
11	W_2	$W_2 \oplus T_2$	$\texttt{41009B1B112DFEF3}_{16}$
12	W_0	$W_0 \oplus T_0$	$\texttt{479E76129979DC5F}_{16}$

А.2 Sponge-функция

В таблице А.2 представлен пример выполнения алгоритма `bash-f`. Приводится результат выполнения $i = 1, 2, 3$ итераций шага 2 алгоритма, а также окончательное выходное значение.

Таблица А.2 — Шаговая функция

S	B194BAC80A08F53B 366D008E584A5DE4 8504FA9D1BB6C7AC 252E72C202FDCE0D 5BE3D61217B96181 FE6786AD716B890B 5CB0C0FF33C356B8 35C405AED8E07F99 E12BDC1AE28257EC 703FCCF095EE8DF1 C1AB76389FE678CA F7C6F860D5BB9C4F F33C657B637C306A DD4EA7799EB23D31 3E98B56E27D3BCCF 591E181F4C5AB793 E9DEE72C8F0C0FA6 2DDB49F46F739647 06075316ED247A37 39CBA38303A98BF6 92BD9B1CE5D14101 5445FBC95E4D0EF2 682080AA227D642F 2687F93490405511$_{16}$
S $(i = 1)$	E2B6A7F6F035D3F2 39480309210BEE8D DED2F39B17FE7C73 4ECA319DCCB1FF76 7BC40A127CF4877A E7FB536FE9390C54 99F34A34D10940B3 0F2B2C93ED128EDD EEB12106DC4F0DFD 41009B1B112DFEF3 BC6D797961DEC912 60E31EF060BE55EB C45AFC52E748DC91 2CAFCA63316F4885 51293EE80CC2D263 22368797C4123CC4 D7C509C309827DE3 2C98DECE4BC4A759 479E76129979DC5F 08C16DF28F6305A6 9D17224CB6817E27 F5823D9AFB05B086 C917D78B6ECAA711 EB72E1BF436E40E7$_{16}$
S $(i = 2)$	BA9659361A0C4CEE 4E3D7DBEA2105A0F C013BAF75A0D25A7 B75E9FD11911F45D FC69D759AECDE7C3 03EF0B29E992C6F8 8B9DE3850D8DFE0C 1BDDCE12F8D6FA9A AF72F482DF11C7CD E5BF7296886D1FAB 4752419560C91DB8 5FB21DB9B8FDE868 C6DC94B8011B4EA1 AE7B7EAD5C8259EA 22DAD6B09B827CD1 D93F3E3D9AB7A83D FF1D5681C46C4A06 9D57FC71FC5A5544 25A032DE52434699 43B5DBE9DDA545A8 94EE1EB7B0B6DEC9 1B02E5748F9141C1 7B2C3572CAC28A7B DDAFB4BA42799C9C$_{16}$
S $(i = 3)$	DFCF8BEE927CFE37 5D9C4D5CAF40D3CB B9D88D53C69035BB 5731D745CC819EBA E2997B65309B248A 84D02D7449D95208 0B501107F1758917 D088BB8CB4CC72C1 EB04E3084DA79297 E636CC72732EFD58 1F31744F59995332 28C3061400E0C34B 9EAE60469BB4F1B6 1E37FA5B319F90FF D4B7D3F007592688 6EBB6B818BB9BAC4 2904D6B8AAABE559 56B7D63B932FA660 D5068CCACE824E9A 43696F09544AA03A 559E325797384232 3435388ADDBF17C4 479570E8E01E18EE 1BE353ABA3EA17EC$_{16}$
bash-f(S)	8FE727775EA7F140 B95BB6A200CBB28C 7F0809C0C0BC68B7 DC5AEDC841BD94E4 03630C301FC255DF 5B67DB53EF65E376 E8A4D797A6172F22 71BA48093173D329 C3502AC946767326 A2891971392D3F70 89959F5D61621238 655975E00E2132A0 D5018CEEDB17731C CD88FC50151D37C0 D4A3359506AEDC2E 6109511E7703AFBB 014642348D8568AA 1A5D9868C4C7E6DF A756B1690C7C2608 A2DC136F5997AB8F BB3F4D9F033C87CA 6070E117F099C409 4972ACD9D976214B 7CED8E3F8B6E058E$_{16}$

А.3 Хэширование (базовые алгоритмы)

В таблице А.3 представлены примеры хэширования с помощью алгоритмов bash-hash$[\ell]$. В таблице для различных уровней стойкости ℓ приводятся хэш-значения сообщения $S[\dots 8m)$, где S — первое слово в таблице А.2.

Таблица А.3 — Хэширование (базовые алгоритмы)

m	Хэш-значение
	$\ell = 128$
0	114C3DFAE373D9BC BC3602D6386F2D6A 2059BA1BF9048DBA A5146A6CB775709D$_{16}$
127	3D7F4EFA00E9BA33 FEED259986567DCF 5C6D12D51057A968 F14F06CC0F905961$_{16}$
128	D7F428311254B8B2 D00F7F9EEFBD8F30 25FA87C4BABD1BDD BE87E35B7AC80DD6$_{16}$
135	1393FA1B65172F2D 18946AEAE576FA1C F54FDD354A0CB297 4A997DC4865D3100$_{16}$
	$\ell = 192$
95	64334AF830D33F63 E9ACDFA184E32522 103FFF5C6860110A 2CD369EDBC04387C 501D8F92F749AE4D E15A8305C353D64D$_{16}$
96	D06EFBC16FD6C088 0CBFC6A4E3D65AB1 01FA82826934190F AABEBFBFFEDE93B2 2B85EA72A7FB3147 A133A5A8FEBD8320$_{16}$
108	FF763296571E2377 E71A1538070CC0DE 88888606F32EEE6B 082788D246686B00 FC05A17405C55176 99DA44B7EF5F55AB$_{16}$
	$\ell = 256$
63	2A66C87C189C12E2 55239406123BDEDB F19955EAF0808B2A D705E249220845E2 0F4786FB6765D0B5 C48984B1B16556EF 19EA8192B985E423 3D9C09508D6339E7$_{16}$
64	07ABBF8580E7E5A3 21E9B940F667AE20 9E2952CEF557978A E743DB086BAB4885 B708233C3F5541DF 8AAFC3611482FDE4 98E58B3379A6622D AC2664C9C118A162$_{16}$
127	526073918F97928E 9D15508385F42F03 ADE3211A23900A30 131F8A1E3E1EE21C C09D13CFF6981101 235D895746A4643F 0AA62B0A7BC98A26 9E4507A257F0D4EE$_{16}$
192	8724C7FF8A2A83F2 2E38CB9763777B96 A70ABA3444F214C7 63D93CD6D19FCFDE 6C3D3931857C4FF6 CCCD49BD99852FE9 EAA7495ECCDD96B5 71E0EDCF47F89768$_{16}$

А.4 Программа для трех автоматов

В таблице А.4 представлен пример выполнения программы для трех автоматов из 8.11.

Таблица А.4 — Программа для трех автоматов

K	B194BAC80A08F53B 366D008E584A5DE4 8504FA9D1BB6C7AC 252E72C202FDCE0D$_{16}$
I	5BE3D61217B96181 FE6786AD716B890B 5CB0C0FF33C356B8 35C405AED8E07F99 E12BDC1AE28257EC 703FCCF095EE8DF1 C1AB76389FE678CA F7C6F860D5BB9C4F F33C657B637C306A DD4EA7799EB23D31 3E98B56E27D3BCCF 591E181F4C5AB7$_{16}$
A_1	E9DEE72C8F0C0FA6 2DDB49F46F739647$_{16}$
A_2	06075316$_{16}$
X	92BD9B1CE5D14101 5445FBC95E4D0EF2 682080AA227D64$_{16}$
K_1	71CC358A0D508217 3DE04803F7E905CB$_{16}$
Y_1	51ED3B28D345FFD1 AD22815B86ECC17C 278C8FE8920214$_{16}$
Y_2	28FE0998BFC010F1 3B260685A27AFB36 CCF580F753521B$_{16}$

А.5 Хэширование (программируемые алгоритмы)

В таблице А.5 представлены примеры хэширования с помощью алгоритмов `bash-prg-hash`$[\ell, d]$. В таблице для различных пар (ℓ, d) приводятся хэш-значения сообщения $S[\dots 8m)$, где S — первое слово в таблице А.2. При хэшировании анонс $A = \perp$, длина хэш-значения $n = 2\ell$.

Таблица А.5 — Хэширование (программируемые алгоритмы)

m	Хэш-значение
$(\ell, d) = (128, 2)$	
0	36FA075EC15721F2 50B9A641A8CB99A3 33A9EE7BA8586D06 46CBAC3686C03DF3$_{16}$
127	C930FF427307420D A6E4182969AA1FFC 3310179B8A0EDB3E 20BEC285B568BA17$_{16}$
128	92AD1402C2007191 F2F7CFAD6A2F8807 BB0C50F73DFF95EF 1B8AF08504D54007$_{16}$
150	48DB61832CA10090 03BC0D8BDE67893A 9DC683C48A5BC23A C884EB4613B480A6$_{16}$
$(\ell, d) = (192, 1)$	
143	6166032D6713D401 A6BC687CCFFF2E60 3287143A84C78D2C 62C71551E0E2FB2A F6B799EE33B5DECD 7F62F190B1FBB052$_{16}$
144	8D84C82ECD0AB646 8CC451CFC5EEB3B2 98DFD381D200DA69 FBED5AE67D26BAD5 C727E2652A225BF4 65993043039E338B$_{16}$
150	47529F9D499AB6AB 8AD72B1754C90C39 E7DA237BEB16CDFC 00FE87934F5AFC11 01862DFA50560F06 2A4DAC859CC13DBC$_{16}$

A.6 Аутентифицированное шифрование

В таблице А.6 представлен пример аутентифицированного шифрования с помощью алгоритма `bash-prg-ae`$[256, 1]$.

Таблица А.6 — Аутентифицированное шифрование

A	B194BAC80A08F53B 366D008E584A5DE4$_{16}$
K	5BE3D61217B96181 FE6786AD716B890B 5CB0C0FF33C356B8 35C405AED8E07F99$_{16}$
X	0^{1536}
I	E12BDC1AE28257EC 703FCCF095EE8DF1 C1AB76389FE678CA F7C6F860D5BB9C4F F33C657B637C306A DD4EA7799EB23D31 3E$_{16}$
Y	690673766C3E848C AC7C05169FFB7B77 51E52A011040E560 2573FAF991044A00 4329EEF7BED8E687 5830A91854D1BD2E DC6FC2FF37851DBA C249DF400A0549EA 2E0C811D499E1FF1 E5E32FAE7F0532FA 4051D0F9E300D9B1 DBF119AC8CFFC48D D3CBF1CA0DBA5DD9 7481C88DF0BE4127 85E40988B3158553 7948B80F5A9C49E0 8DD684A7DCA871C3 80DFDC4C4DFBE61F 50D2D0FBD24D8B9D 32974A347247D001 BAD5B16844002569 3967E77394DC088B 0ECCFA8D291BA13D 44F60B06E2EDB351$_{16}$
T	CDE5AF6EF9A14B7D 0C191B869A6343ED 6A4E9AAB4EE00A57 9E9E682D0EC051E3$_{16}$

Приложение Б
(рекомендуемое)
Модуль ACH.1

В модуле ACH.1 определяются идентификаторы следующих алгоритмов:

`bashNNN`	Алгоритм хэширования `bash-hash`$[\ell]$.
`bash-prg-hashNNND`	Алгоритм хэширования `bash-prg-hash`$[\ell, d]$.
`bash-prg-aeLLLD`	Алгоритм аутентифицированного шифрования `bash-prg-ae`$[\ell, d]$.
`bash-f`	Алгоритм вычисления значений sponge-функции (см. 6.2).

Здесь `NNN` — десятичный код числа 2ℓ, `LLL` — десятичный код числа ℓ, `D` — десятичный код числа d.

Алгоритм `bash-hash`$[\ell]$ определяет функцию хэширования $h\colon \{0,1\}^* \to \{0,1\}^{2\ell}$. Эта функция может использоваться в алгоритмах ЭЦП СТБ 34.101.45, уточняя их. Правила использования h определены в СТБ 34.101.45 (пункт 5.5). Уточненным алгоритмам ЭЦП присваиваются следующие идентификаторы:

`bign-with-bashNNN`	Алгоритмы ЭЦП СТБ 34.101.45 (пункт 7.1) с функцией хэширования, заданной алгоритмом `bash-hash`$[\ell]$.
`bign-ibs-with-bashNNN`	Алгоритмы идентификационной ЭЦП СТБ 34.101.45 с функцией хэширования, заданной алгоритмом `bash-hash`$[\ell]$.

Здесь, как и прежде, `NNN` — десятичный код числа 2ℓ.

Модуль ACH.1 имеет следующий вид:

```
Bash-module-v2 {iso(1) member-body(2) by(112) 0 2 0 34 101 77 module(1) ver2(2)}
DEFINITIONS ::=
BEGIN
  IMPORTS
    bign
      FROM Bign-module-v2 {iso(1) member-body(2) by(112) 0 2 0 34 101 45
        module(1) ver2(2)};

  bash OBJECT IDENTIFIER ::= {iso(1) member-body(2) by(112) 0 2 0 34 101 77}

  bash256 OBJECT IDENTIFIER ::= {bash 11}
  bash384 OBJECT IDENTIFIER ::= {bash 12}
  bash512 OBJECT IDENTIFIER ::= {bash 13}
  bash-prg-hash2561 OBJECT IDENTIFIER ::= {bash 21}
  bash-prg-hash2562 OBJECT IDENTIFIER ::= {bash 22}
  bash-prg-hash3841 OBJECT IDENTIFIER ::= {bash 23}
  bash-prg-hash3842 OBJECT IDENTIFIER ::= {bash 24}
  bash-prg-hash5121 OBJECT IDENTIFIER ::= {bash 25}
  bash-prg-hash5122 OBJECT IDENTIFIER ::= {bash 26}
  bash-prg-ae1281 OBJECT IDENTIFIER ::= {bash 31}
  bash-prg-ae1282 OBJECT IDENTIFIER ::= {bash 32}
```

```
  bash-prg-ae1921 OBJECT IDENTIFIER ::= {bash 33}
  bash-prg-ae1922 OBJECT IDENTIFIER ::= {bash 34}
  bash-prg-ae2561 OBJECT IDENTIFIER ::= {bash 35}
  bash-prg-ae2562 OBJECT IDENTIFIER ::= {bash 36}
  bash-f OBJECT IDENTIFIER ::= {bash 101}

  bign-with-bash256 OBJECT IDENTIFIER ::= {bign 13}
  bign-with-bash384 OBJECT IDENTIFIER ::= {bign 14}
  bign-with-bash512 OBJECT IDENTIFIER ::= {bign 15}
  bign-ibs-with-bash256 OBJECT IDENTIFIER ::= {bign 73}
  bign-ibs-with-bash384 OBJECT IDENTIFIER ::= {bign 74}
  bign-ibs-with-bash512 OBJECT IDENTIFIER ::= {bign 75}
END
```

Библиография

[1] Bertoni G., Daemen J., Peeters M., Van Assche G. Cryptographic sponge functions
Avail. at `http://sponge.noekeon.org/CSF-0.1.pdf`, 2011
(Бертони Г., Дэмен Дж., Питерс М., ван Аш Г. Криптографические sponge-функции)

Поправка к официальной редакции

В каком месте	Напечатано	Должно быть
Приложение Б, определение идентификатора `bign-with-bashNNN`	Алгоритмы ЭЦП СТБ 34.101.45 (пункт 7.1) с функцией хэширования, заданной алгоритмом `bash-hash`$[2\ell]$.	Алгоритмы ЭЦП СТБ 34.101.45 (пункт 7.1) с функцией хэширования, заданной алгоритмом `bash-hash`$[\ell]$.
Приложение Б, определение идентификатора `bign-ibs-with-bashNNN`	Алгоритмы идентификационной ЭЦП СТБ 34.101.45 с функцией хэширования, заданной алгоритмом `bash-hash`$[2\ell]$.	Алгоритмы идентификационной ЭЦП СТБ 34.101.45 с функцией хэширования, заданной алгоритмом `bash-hash`$[\ell]$.

Синим цветом выделены корректировки, пока не принятые официально.