Таблица 2

Результаты применения набора тестов SMHasher

Тест	CRC32	MD5	SHA2-256	SHA3-256	Стрибог	*AG-S*-Стрибог
Sanity	+	+	+	+	+	+
Avalanche	−	+	+	+	+	+
Chi2	−	+	+	+	+	+
Differential	1/2	2/2	2/2	2/2	2/2	2/2
Collisions	6/41	36/41	41/41	41/41	41/41	25/41

Выводы

1. Алгоритм *AG-S* от 1,93 до 6,44 раз превышает по производительности известные функции хэширования (бесключевые и ключевые) и близок по производительности к CRC32.

2. По результатам тестов Avalanche, Chi2 и Differential алгоритм *AG-S*-Стрибог не уступает известным функциям хэширования и значительно превосходит алгоритм CRC32. Для усиления свойств алгоритма *AG-S* следует продолжить исследование коллизий.

ЛИТЕРАТУРА

1. *Stigge M., Plotz H., Muller W., and Redlich J.-P.* Reversing CRC — Theory and Practice. HU Berlin Public Report, 2006.
2. *Фомичев В. М., Коренева А. М., Набиев Т. Р.* О новом алгоритме контроля целостности данных // Конференция Рускрипто'20, Московская область, 2020. `https://www.ruscrypto.ru/resource/archive/rc2020/files/02_koreneva_fomichev.pdf`
3. *Фомичев В. М., Коренева А. М., Набиев Т. Р.* Характеристики алгоритма контроля целостности данных на основе аддитивных генераторов и s-боксов // Прикладная дискретная математика. Приложение. 2020. № 13. С. 62–66.
4. `https://github.com/rurban/smhasher`.
5. Хэш-функция t1ha. `ttps://github.com/PositiveTechnologies/t1ha`.

УДК 004.056 DOI 10.17223/2226308X/14/16

ОБ АЛГОРИТМЕ ДОПОЛНЕНИЯ БЛОКОВ БОЛЬШОГО РАЗМЕРА В СИСТЕМАХ КОНТРОЛЯ ЦЕЛОСТНОСТИ

Д. А. Бобровский, Т. Р. Набиев, В. М. Фомичёв

В алгоритмах контроля целостности при расчёте контрольной суммы файла требуется, чтобы его длина была кратна заданной величине (l бит). При защите файла произвольной длины, как правило, выполняется его дополнение до требуемой длины. Представлена вычислительно простая и эффективная схема дополнения, предназначенная для систем контроля целостности, обрабатывающих большие блоки (порядка 1 кбайт). Схема построена на основе выходов линейного конгруэнтного генератора. Начальное состояние генератора формируется с помощью данных дополняемого блока и исходной длины файла. Результаты анализа криптографических свойств алгоритма контроля целостности и экспериментов по оценке производительности показали преимущества предложенной схемы по сравнению с известными стандартными схемами дополнения.

Ключевые слова: *алгоритм дополнения, широкий блок, линейный конгруэнт-*

ный генератор, характеристики процедур дополнения, контроль целостности, AG-S, SMHasher.

Введение

При работе с большими блоками (порядка 1 кбайт), например, как в алгоритме контроля целостности данных на основе аддитивных генераторов и s-боксов [1, 2], определение корректной и экономной процедуры дополнения блоков данных до подходящих размеров является одной из важнейших задач, так как для достижения хорошего перемешивания биты дополнения должны быть реализацией псевдослучайной функции.

Для сравнения проанализированы известные схемы дополнения: дополнение битами (00...0, 00...01), двухступенчатое дополнение (SHA, MD5). Оценена производительность и свойства предложенного алгоритма.

1. Описание схемы дополнения

Процедура дополнения файлов произвольной длины (в битах) до длины, кратной l, использует линейный конгруэнтный генератор (ЛКГ) над кольцом вычетов $\mathbb{Z}_m$, то есть линейное преобразование g кольца $\mathbb{Z}_m$ вида

$$g(X) = (aX + c) \bmod m,$$

где $X, a, c \in \mathbb{Z}_m$; $a \neq 0$; числа a и c суть множитель и сдвиг соответственно.

Обозначим $X_0 \in \mathbb{Z}_m$ — начальное состояние ЛКГ; $X_{n+1} = g(X_n)$, $n \geqslant 0$. ЛКГ порождает периодическую последовательность с начальным состоянием X_0 и длиной периода m, если сдвиг нечётный и $a \equiv 1 \pmod 4$.

Пусть $D \in V^*$ — исходный файл, представимый как битовая последовательность, где V^* — множество всех двоичных строк конечной длины; D' — последний неполный блок файла D. Если длина p блока D' меньше l, то он дополняется до длины l блоком L длины $l - p$, и дополненный блок имеет вид $D' \parallel L$ ($\parallel$ означает присоединение).

Блок L длины $l - p$ вырабатывается следующим образом:

1. Формируется начальное значение ЛКГ X_0:
   а) строке P присваивается значение D', т. е. $P = D'$;
   б) строка P дополняется нулями до длины, кратной $r = 64$, т. е. $P^* = (P||0^{r-(p \bmod r)})$;
   в) строка P^* разбивается на $\lceil p/r \rceil$ блоков длины r: $P^* = (P_1 \parallel \ldots \parallel P_{\lceil p/r \rceil})$, где $P_1, \ldots, P_{\lceil p/r \rceil} \in V_r$;
   г) $X_0 = (P_1 \boxplus \ldots \boxplus P_{\lceil p/r \rceil}) \lll ((p/8) \bmod 64)$, где $a \lll b$ — операция циклического сдвига элементов строки a на b позиций; $\boxplus$ — сложение по модулю 2^r.
2. С помощью ЛКГ [3] с начальным значением X_0 и параметрами

   $$a = 6364136223846793005,\ c = 1442695040888963407,\ m = 2^{64}$$

   порождается последовательность $\lceil (l-p)/r \rceil$ элементов $\{X_1, \ldots, X_{\lceil (l-p)/r \rceil}\}$, из которых формируется строка L^*:

   $$L^* = (X_1|| \ldots ||X_{\lceil (l-p)/r \rceil}).$$
3. Дополнение L образуют старшие $l - p$ бит строки L^*.

2. Экспериментальное сравнение характеристик процедур дополнения

С помощью экспериментов исследованы следующие процедуры дополнения:

1) дополнение нулями;
2) дополнение битовым вектором $(100\ldots000)$;
3) дополнение битовым вектором $(100\ldots000)||\lambda$, где λ — запись длины дополняемой строки (для записи требуется 2 байта);
4) дополнение в соответствии с процедурой, описанной в п. 1.

Обозначим данные процедуры дополнения как Padding 1, ..., Padding 4 соответственно.

2.1. Продолжительность выполнения операции дополнения

Проведено сравнение производительности процедур дополнения входных векторов. В ходе эксперимента сгенерировано по 20000 случайных входных строк длины L, которые дополнялись до длины 1024 байта, длина L изменялась от 100 до 1000 с шагом 100 байтов.

Программная реализация процедур дополнения выполнена на языке программирования C++. Эксперименты проведены на ПЭВМ с процессором Intel(R) Core(TM) i5-8600 с постоянной тактовой частотой $U = 4{,}1$ ГГц, архитектура операционной системы 64-битная (x64). Оптимизация программного кода — /O2.

Время t, затраченное на формирование дополненного блока длины 1024 байт, измерялось с помощью системных часов реального времени стандартной библиотеки std::chrono.

По формуле tU/L рассчитано среднее время, затраченное на дополнение, а также независимая от частоты процессора характеристика производительности процедуры — среднее количество тактов на байт (CpB). Результаты экспериментов приведены в табл. 1 и на рис. 1.

Таблица 1

Результаты замера скорости процедуры дополнения

Характеристика	Padding 1	Padding 2	Padding 3	Padding 4
Среднее время дополнения, нс	148	189	204	320
Среднее число тактов на байт, CpB	1,475	2,266	2,399	3,452

По результатам эксперимента длительность дополнения с помощью ЛКГ больше, чем у других процедур, от 1,5 до 2 раз. Однако важно, что в системах контроля целостности файлов произвольной длины, использующих алгоритмы с большим блоком, например алгоритм *AG-S* [1], время генерации кода контроля целостности без учёта дополнения составляет 7100 нс. Следовательно, суммарное время с процедурой дополнения нулями составляет 7248 нс, а с предложенной процедурой дополнения — 7420 нс, т. е. общее замедление незначительно и не превышает 2,37 %. Значит, с учётом общей продолжительности генерации кода процедура дополнения на основе ЛКГ другим процедурам существенно не уступает.

2.2. Криптографические характеристики процедур дополнения

При анализе алгоритмов генерации кодов контроля целостности важные криптографические характеристики связаны с равномерностью распределения хэш-значений, оценкой лавинного эффекта, поиском коллизий и производительностью хэш-

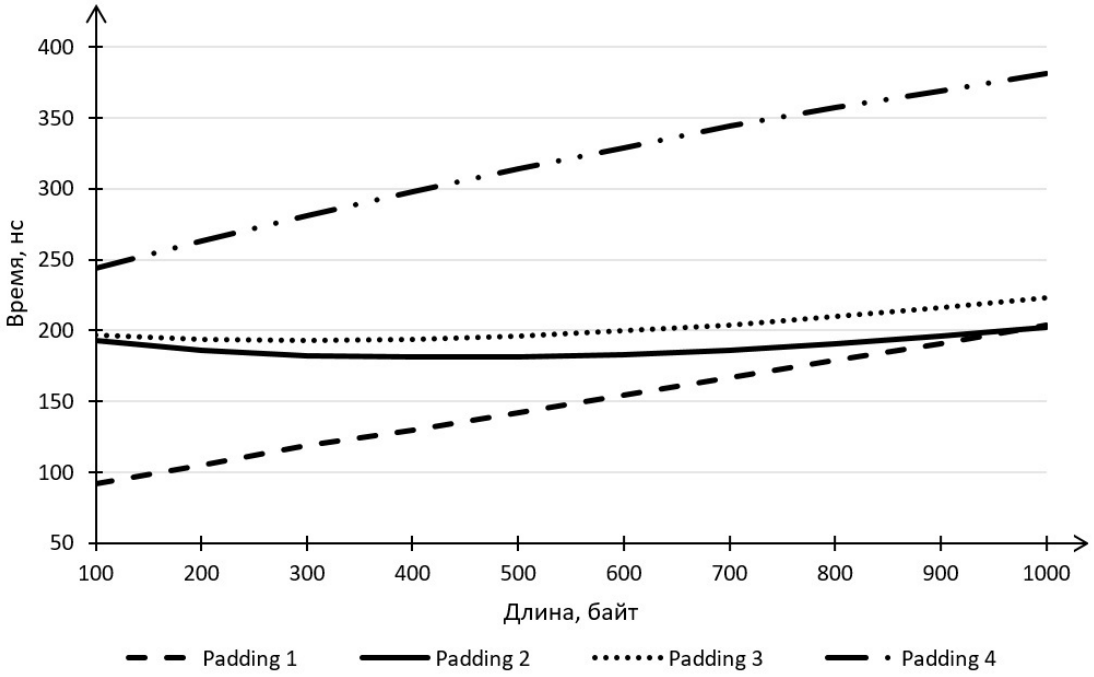

Рис. 1. Зависимость времени дополнения от размера входного блока

функций. Статистические испытания были выполнены с помощью хорошо зарекомендовавшего себя на практике тестового пакета SMHasher [3]. В 2017 г. компания «Positive Technologies» использовала данный пакет тестов для тестирования разработанной функции хэширования [4]. Этот набор тестов (open-source продукт) предназначен для тестирования указанных криптографических характеристик.

При тестировании за основу был взят алгоритм *AG-S*-Стрибог [2]. Были подготовлены четыре реализации, где входы дополнялись процедурами дополнения Padding 1–Padding 4. Сравнительные результаты тестирования данных реализаций с помощью тестов SMHasher представлены в табл. 2–4.

Криптографические свойства реализаций в связи с процедурой дополнения были протестированы пакетом SMHasher, а именно группами тестов: «Avalanche» — на выполнение строгого лавинного критерия; «Sparse» — на поиск коллизий среди входов с большим количеством нулевых битов; «Permutation» — на поиск коллизий среди входов, составленных из повторяющихся блоков.

Таблица 2

Общие результаты тестирования (количество пройденных тестов)

Группа тестов	Padding 1	Padding 2	Padding 3	Padding 4	Всего тестов
«Avalanche»	6	6	12	12	12
«Sparse»	7	7	7	9	11
«Permutation»	0	2	2	2	15
Общее число пройденных тестов	13	15	21	23	38

Суммарное количество пройденных тестов реализации с процедурой дополнения на основе выходов ЛКГ больше, чем при других процедурах дополнения. Группа тестов «Avalanche» полностью пройдена при двухступенчатом дополнении и при дополнении на основе выходов ЛКГ.

Результаты по группам тестов «Sparse» и «Permutation» представлены в табл. 3 и 4.

Таблица 3

Количество коллизий при различных длинах входа в группе тестов «Sparse»

Размер входного вектора, бит	Padding 1	Padding 2	Padding 3	Padding 4	Всего входных векторов
72	241644 (0,016 %)	259209 (0,017 %)	109595 (0,007 %)	0	15082603
96	46809 (0,013 %)	48817 (0,014 %)	28666 (0,008 %)	0	3469497
160	719293 (0,026 %)	724939 (0,024 %)	468548 (0,017 %)	77968 (0,003 %)	26977161
256	179226 (0,064 %)	170800 (0,061 %)	131358 (0,047 %)	5465 (0,002 %)	2796417
Всего коллизий	1186972	1203765	738167	83433	48325678
Доля коллизий среди всех входов	0,024562	0,024909	0,015275	0,001726	

Применение процедуры дополнения на основе ЛКГ в группе тестов «Sparse» снижает количество коллизий примерно от 9 до 14 раз, что положительно характеризует её криптографические свойства.

Таблица 4

Количество коллизий в группе тестов «Permutation»

Параметр	Padding 1	Padding 2	Padding 3	Padding 4	Всего входных векторов
Общее число коллизий по всем тестам	8217021	6622245	6089853	4256492	20143432
Доля коллизий среди всех входов	0,407926	0,328755	0,302324	0,211309	

Применение процедуры дополнения на основе ЛКГ в группе тестов «Permutation» снижает количество коллизий примерно от 1,5 до 2 раз.

Выводы

1. Процедура дополнения на основе ЛКГ продолжительнее стандартных процедур дополнения от 1,5 до 2 раз, однако при этом общая продолжительность генерации кода другим процедурам существенно не уступает, замедление составляет не более 2,37 %.

2. На примере алгоритма *AG-S*-Стрибог экспериментально показано, что процедура дополнения на основе ЛКГ может обеспечить преимущества перед другими известными процедурами по ряду важных криптографических свойств.

ЛИТЕРАТУРА

1. *Фомичев В. М., Коренева А. М., Набиев Т. Р.* Характеристики алгоритма контроля целостности данных на основе аддитивных генераторов и s-боксов // Прикладная дискретная математика. Приложение. 2020. № 13. С. 62–66.

2. *Бобровский Д. А., Задорожный Д. И., Коренева А. М. и др.* О контроле целостности данных с использованием хэширования // Рускрипто'21, Московская область, 2021. https://www.ruscrypto.ru/resource/archive/rc2021/files/02_bobrovskiy_zadorozhniy_koreneva_nabiyev_fomichev.pdf.

3. https://github.com/rurban/smhasher.
4. Хэш-функция t1ha. https://github.com/PositiveTechnologies/t1ha.

УДК 004.056.55 DOI 10.17223/2226308X/14/17

ПОРОГОВАЯ СХЕМА ПРОТОКОЛА ДИФФИ — ХЕЛЛМАНА

Д. Н. Колегов, Ю. Р. Халниязова

Предлагается пороговая схема протокола Диффи — Хеллмана на эллиптических кривых, которая позволяет создавать и хранить закрытый ключ участника протокола распределённым образом без необходимости восстановления ключа для выполнения криптографических операций на этом ключе.

Ключевые слова: *пороговая криптография, протокол Диффи — Хеллмана, эллиптические кривые.*

Пусть sk — закрытый ключ участника протокола Диффи — Хеллмана. Будем называть *функцией Диффи — Хеллмана* функцию $DH(sk, Q)$, которая принимает на вход закрытый ключ sk (скаляр) и точку Q на эллиптической кривой и возвращает точку $sk \cdot Q$. Под протоколом Диффи — Хеллмана обычно понимают следующую последовательность вычислений (G — образующий элемент подгруппы простого порядка q группы точек эллиптической кривой E над конечным полем):

1) Алиса генерирует случайное число $a \in \mathbb{Z}_q$, вычисляет значение $A = DH(a, G)$ и отправляет его Бобу;
2) Боб генерирует случайное число $b \in \mathbb{Z}_q$, вычисляет значение $B = DH(b, G)$ и отправляет его Алисе;
3) Алиса вычисляет общий секрет как $K = DH(a, B)$, а Боб — как $K = DH(b, A)$.

Идея пороговой схемы Диффи — Хеллмана заключается в том, чтобы сгенерировать закрытый ключ участника протокола x с помощью распределённого алгоритма некоторыми сущностями, а затем делегировать им вычисление значения функции $DH(x, Q)$, используя методы пороговой криптографии. Будем называть таких сущностей *агентами*, а *участником* протокола Диффи — Хеллмана будем называть группу агентов, которая представляет одну из сторон протокола Диффи — Хеллмана и выполняет установленные протоколом шаги.

Если в классическом протоколе Диффи — Хеллмана участником является атомарная сущность (человек, процесс и т. д.), то теперь участник протокола — это группа сущностей — агентов (людей, процессов, ...), которые взаимодействуют между собой посредством разработанных протоколов так, что для внешних сущностей (другого участника протокола, сторонних наблюдателей и т. д.) группа агентов неотличима от обычного участника протокола Диффи — Хеллмана. При этом по результатам выполнения этих протоколов любой из агентов знает открытый ключ группы и может представлять группу при взаимодействии с другим участником протокола. Так как группа агентов неотличима для внешних сущностей от обычного участника, то для простоты изложения далее будем считать, что только один из участников протокола Диффи — Хеллмана представлен группой агентов. При этом неважно, какой именно из участников протокола состоит из агентов, так как вычисления, выполняемые участниками, симметричны.

Первым этапом предлагаемой схемы является генерация долей закрытого ключа, а также вычисление открытого ключа соответствующего участника протокола Диффи — Хеллмана. Вычисленный открытый ключ известен каждому агенту из группы,