Репин Максим Михайлович
E-mail: bmstu.iu8@gmail.com.
Тел.: 84992494429, факс: 84959331563.

Mikheev Viatcheslav Alexeevich
The deputy director of Joint-Stock Company «Radio Engineering Corporation «VEGA».
E-mail: mikheev@vega.su.
34, Kutuzov avenue, Moscow, 121170, Russia.
Phone: +74992490585, fax: +74959331563.

Repin Maxim Mixajlovich
E-mail: bmstu.iu8@gmail.com.
Phone: +74992494429; fax: +74959331563.

УДК 681.03.245

Л.К. Бабенко, Е.А. Ищукова

ДИФФЕРЕНЦИАЛЬНЫЙ КРИПТОАНАЛИЗ УПРОЩЕННОЙ ФУНКЦИИ ХЭШИРОВАНИЯ SHA*

Рассмотрены основные подходы к анализу современных функций хэширования с использованием метода дифференциального криптоанализа на примере упрощенных версий функции SHA. Подходы, рассмотренные для анализа функции SHA, могут быть легко использованы для анализа других современных функций хэширования.

Функция хэширования; дифференциальный криптоанализ; разность; вероятность.

L.K. Babenko, E.A. Ischukova

DIFFERENTIAL CRYPTANALYSIS OF SHA-LIKE HASH FUNCTIONS

In article highlights of hash functions differential cryptanalysis on an example of algorithm SHA are considered. The technique of carrying out of the differential analysis of SHA hash function and also other hash functions having a similar structure is offered.

Hash function; differential cryptanalysis; difference; probability.

Как известно, криптография призвана решать задачи обеспечения конфиденциальности, целостности, аутентификации, невозможности отказа от авторства, неотслеживаемости с использованием математических методов. Для решения ряда данных задач используются криптографические функции хэширования (hash-functions) [1]. Хэш-функции – это функции, предназначенные для сжатия произвольного сообщения или набора данных, записанного, как правило, в двоичном алфавите, в некоторую битовую комбинацию фиксированной длины, называемую сверткой. В криптографии хэш-функции применяются для решения двух основных задач:

- построения систем контроля целостности данных при их передаче или хранении;
- аутентификации данных.

В 1989 г. Р. Меркль (Ralph C. Merkle) и И. Дамгорд (Ivan Damgaard) [1] независимо предложили итеративный принцип построения криптографических функций хэширования. Данный принцип позволяет свести задачу построения хэш-функции на множестве сообщений различной длины к задаче построения отобра-

* Работа поддержана грантом РФФИ № 09-07-00245-а.

жения, действующего на множестве фиксированной конечной длины. По итеративному принципу построено абсолютное большинство хэш-функций, используемых в настоящее время на практике. Например, хэш-функции MD5, SHA-1, семейство хэш-функций SHA-2, отечественный стандарт на хэш-функцию ГОСТ Р 34.11-94.

В последние годы в научном мире наблюдается повышенный интерес к проектированию и анализу алгоритмов хэширования. Об этом свидетельствует большое количество статей, посвященных хэш-функциям, представляемым на мировых конференциях по криптографии CRYPTO и EUROCRYPT. Авторами этих статей часто являются люди, стоящие у истоков современной криптографии, такие как Эли Бихам (Eli Biham), Ади Шамир (Adi Shamir), Барт Пренил (Bart Preneel), Ларс Кнудсен (Lars R. Knudsen), Рональд Ривест (Ronald L. Rivest), Алекс Бирюков (Alex Biryukov), Орр Дункелман (Orr Dunkelman), Винсент Рижмен (Vincent Rijmen).

Наряду с анализом уже существующих функций хэширования, предлагаются новые, заявляемые авторами как более надежные. Кроме того, предлагаются новые методы анализа, которые, как правило, рассчитаны на довольно широкий класс алгоритмов хэширования. Подтверждением тому служит конкурс на принятие нового стандарта хэширования SHA-3, проводимый Национальным институтом стандартов и технологий США (НИСТ – National Institute of Standards and Technology (NIST)). В 2002 г. в США был принят стандарт Federal Information Processing Standard 180-2 (FIPS 180-2), определявший 5 основных функций хэширования SHA-1, SHA-224, SHA-256, SHA-384 и SHA-512. Появление серии работ японских ученых, направленных на анализ алгоритмов семейства SHA [2–4], позволило усомниться в стойкости данного стандарта. Так, в работе [4] заявлено, что для алгоритма SHA-1 возможно выполнить поиск коллизий. Несмотря на то, что работы [2–4] не содержат полных сведений о методах, предложенных для анализа, и на заявление сотрудника НИСТ Вильяма Бюрра (William E. Burr) о том, что метод поиска коллизий, предложенный в работе [4] до сих пор никем не подтвержден, в ноябре 2007 г. стартовал проект, направленный на поиск и принятие стандарта нового поколения SHA-3. При этом на сайте НИСТ были опубликованы сведения о том, что после 2010 года алгоритм SHA-1 не должен быть использован для электронной цифровой подписи (ЭЦП) и любых других приложений, требующих устойчивости к поиску коллизий (здесь и далее данные взяты с сайта Национального института стандартов и технологий США http://csrc.nist.gov).

На конкурс SHA-3 был представлен 51 алгоритм, о чем было объявлено в декабре 2008 г. В результате первого раунда конкурса, который завершился в июле 2009 г., было отобрано 14 претендентов. В настоящий момент продолжается второй этап конкурса, в результате которого все претенденты должны быть подвергнуты тщательному анализу. В конце 2010 г. будут объявлены финалисты, которые будут исследованы дальше в 2011–2012 гг.

Как и в случаях с алгоритмами шифрования, при криптоаналистических атаках функций хэширования пытаются обнаружить свойства алгоритма, позволяющие уменьшить объем вычислений в сравнении с объемом вычислений при полном переборе вариантов. Стойкость функции хэширования в отношении криптоанализа можно определить в сравнении с объемом усилий, необходимых для перебора всех вариантов. Одним из мощнейших методов анализа функций хэширования является метод дифференциального криптоанализа, который впервые был применен Э. Бихамом и А. Шамиром в начале 90-х годов XX века для анализа алгоритма DES и DES-подобных блочных шифров.

В данной статье авторы предлагают рассмотреть основные подходы, которые используются при анализе современных функций хэширования с использованием метода дифференциального криптоанализа. В качестве объектов анализа рассмотрим алгоритмы семейства SHA, которые являются одними из самых известных и распространенных на сегодняшний день.

Описание алгоритма SHA

В начале работы алгоритма сообщение дополняется с тем, чтобы его длина стала кратной 512 разрядам. При этом используется то же дополнение, что и в алгоритме MD5: в начале добавляется 1, а затем нули так, чтобы размер полученного сообщения был на 64 разряда меньше числа, кратного 512, а затем к полученному результату добавляется 64-битовое представление размера исходного сообщения (перед дополнением).

Далее инициализируются пять 32-разрядных переменных: A = 0x67452301; B = 0xefcdab89; C = 0x98badcfe; D = 0x10325476; E = 0xc3d2e1f0.

Затем начинается главный цикл обработки алгоритма. В этом цикле алгоритм SHA обрабатывает сообщение блоками размером 512 разрядов, и цикл продолжается, пока не исчерпаются все блоки сообщения.

Сначала пять переменных копируются в другие переменные: A в a, B в b, C в c, D в d и E в e. Главный цикл состоит из четырех этапов по 20 операций в каждом. Каждая операция представляет собой нелинейную функцию над тремя из a, b, c, d и e, а затем выполняет сдвиг и сложение аналогично MD5. Набор нелинейных функций, используемых в алгоритме SHA, а также набор констант приведены в табл. 1.

Таблица 1

Функции и константы, используемые в алгоритме SHA

Номер раунда i	Функция $f^{(i)}$		Константа $K^{(i)}$
	Название функции	Описание	
От 0 до 19	IF	$(X \wedge Y) \vee ((\neg X) \wedge Z)$	0x5a827999
От 20 до 39	XOR	$X \oplus Y \oplus Z$	0x6ed9eba1
От 40 до 59	MAJ	$(X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z)$	0x8flbbcdc
От 60 до 79	XOR	$X \oplus Y \oplus Z$	0xca62c1d6

Блок сообщения превращается из 16 32-битовых слов ($M^{(0)}$ по $M^{(15)}$) в 80 32-битовых слов ($W^{(0)}$ по $W^{(79)}$) с помощью следующего алгоритма:

$$W^{(i)} = M^{(i)}, \text{ для } i = 0 \text{ по } 15;$$

$$W^{(i)} = (W^{(i-3)} \oplus W^{(i-8)} \oplus W^{(i-14)} \oplus W^{(i-16)}) <<< 1, \text{ для } i = 16 \text{ по } 79. \quad (1)$$

Следует отметить, что в первой версии функции SHA-0 отсутствовал циклический сдвиг влево на 1 позицию и использовалась следующая формула:

$$W^{(i)} = W^{(i-3)} \oplus W^{(i-8)} \oplus W^{(i-14)} \oplus W^{(i-16)}, \text{ для } i = 16 \text{ по } 79. \quad (2)$$

Пусть i – это номер операции (от 1 до 80), $W^{(i)}$ представляет собой i-й подблок расширенного сообщения, $W_j^{(i)}$ – j-й бит i-го подблока сообщения (при этом j=0 означает младший значащий бит, а j=31 – старший значащий бит), а <<<s – это циклический сдвиг влево на s битов, тогда главный цикл преобразования выглядит следующим образом:

for i = 0 to 79
$a^{(i+1)} = (a^{(i)} <<< 5) + f^{(i)}(b^{(i)}, c^{(i)}, d^{(i)}) + e^{(i)} + W^{(i)} + K^{(i)}$;
$b^{(i+1)} = a^{(i)}$;
$c^{(i+1)} = (b^{(i)} <<< 30)$;
$d^{(i+1)} = c^{(i)}$;
$e^{(i+1)} = d^{(i)}$.

На рис. 1 показана одна операция функции SHA. Сдвиг переменных выполняет ту же функцию, которую в MD5 выполняет использование в различных местах различных переменных.

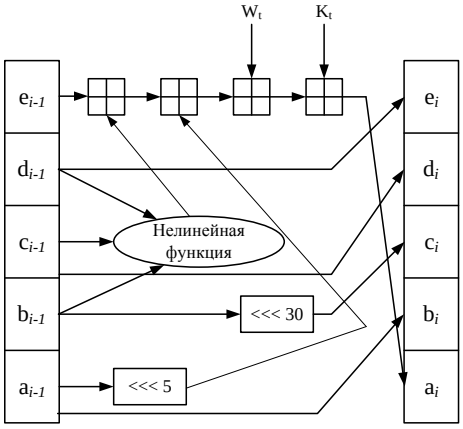

Рис. 1. Одна операция функции SHA

После всего этого a, b, c, d и e добавляются к A, B, C, D и E, соответственно, и алгоритм продолжается для следующего блока данных. Окончательным результатом служит объединение A, B, C, D и E.

Анализ функции SHA

Главной задачей при выполнении анализа функций хэширования является поиск коллизий за время меньшее, чем время полного перебора всех возможных вариантов. Таким образом, задача сводится к поиску двух таких сообщений $(M^{(0)} \ldots M^{(15)}) = (W^{(0)} \ldots W^{(15)})$ и $(M'^{(0)} \ldots M'^{(15)}) = (W'^{(0)} \ldots W'^{(15)})$, для которых будет выработано одно и то же хэш-значение $(a^{(80)}, b^{(80)}, c^{(80)}, d^{(80)}, e^{(80)})$ с использованием одних и тех же начальных значений $(a^{(0)}, b^{(0)}, c^{(0)}, d^{(0)}, e^{(0)})$.

Вначале рассмотрим, как будет изменяться разность в случае, если в функции SHA оставить только линейные элементы. Это необходимо для того, чтобы понять структуру функции хэширования, с одной стороны, и изучить критпографические примитивы функции, с другой. В семействе алгоритмов SHA присутствует два источника нелинейности – это функция $f^{(i)}$ и операция целочисленного сложения по модулю 2^{32}. Поэтому вначале рассмотрим вариант функции SHA, в котором используется формула (2) для выработки подблоков расширенного сообщения (как это сделано в алгоритме SHA-0), операция целочисленного сложения по модулю 2^{32} заменена на операцию сложения по модулю 2 и все функции $f^{(i)}$ используются в режиме XOR (см. табл. 1). В работе [5] вариант такой функции хэширования обозначен как функция SHI1.

Забудем на время о том, что подблоки W вырабатываются в процессе работы функции хэширования. Это, например, справедливо для первых шестнадцати блоков $(W^{(0)} \ldots W^{(15)})$, которые берутся из исходного сообщения. Рассмотрим, какое влияние окажет изменение одного бита в блоке $W^{(i)}$ на выработку последующих значений.

Например, изменим значение второго бита в блоке $W^{(i)}$, то есть обратим значение $W_1^{(i)}$. Для этого проследим 5 циклов преобразований так, как это сделано в табл. 2.

Таблица 2

Влияние изменения одного бита на выработку последующих значений

№ цикла	Преобразование	Измененные биты	Комментарий
1	$a^{(i+1)} = W^{(i)} \oplus (a^{(i)} <<< 5) \oplus b^{(i)} \oplus c^{(i)} \oplus d^{(i)} \oplus e^{(i)} \oplus K^{(i)}$;	$a_1^{(i+1)}$	Так как изменен бит $W_1^{(i)}$, то изменится значение бита $a_1^{(i+1)}$
	$b^{(i+1)} = a^{(i)}$;	–	Без изменений
	$c^{(i+1)} = (b^{(i)} <<< 30)$;	–	
	$d^{(i+1)} = c^{(i)}$;	–	
	$e^{(i+1)} = d^{(i)}$;	–	
2	$a^{(i+2)} = W^{(i+1)} \oplus (a^{(i+1)} <<< 5) \oplus b^{(i+1)} \oplus c^{(i+1)} \oplus d^{(i+1)} \oplus e^{(i+1)} \oplus K^{(i+1)}$;	$a_6^{(i+2)}$	Так как изменен бит $a_1^{(i+1)}$, то после сдвига влево на 5 позиций он окажет влияние на 7 бит блока $a^{(i+2)}$, то есть изменится бит $a_6^{(i+2)}$
	$b^{(i+2)} = a^{(i+1)}$;	$b_1^{(i+2)}$	Так как изменен бит $a_1^{(i+1)}$, то изменится бит $b_1^{(i+2)}$
	$c^{(i+2)} = (b^{(i+1)} <<< 30)$;	–	Без изменений
	$d^{(i+2)} = c^{(i+1)}$;	–	
	$e^{(i+2)} = d^{(i+1)}$;	–	
3	$a^{(i+3)} = W^{(i+2)} \oplus (a^{(i+2)} <<< 5) \oplus b^{(i+2)} \oplus c^{(i+2)} \oplus d^{(i+2)} \oplus e^{(i+2)} \oplus K^{(i+2)}$;	$a_1^{(i+3)}$ $a_{11}^{(i+3)}$	Так как изменен бит $a_6^{(i+2)}$, то после сдвига влево на 5 позиций он окажет влияние на 12 бит блока $a^{(i+3)}$, то есть изменится бит $a_{11}^{(i+3)}$. Кроме того, так как изменен бит $b_1^{(i+2)}$, то изменится бит $a_1^{(i+3)}$
	$b^{(i+3)} = a^{(i+2)}$;	$b_6^{(i+3)}$	Так как изменен бит $a_6^{(i+2)}$, то изменится бит $b_6^{(i+3)}$
	$c^{(i+3)} = (b^{(i+2)} <<< 30)$;	$c_{31}^{(i+3)}$	Так как изменен бит $b_1^{(i+2)}$, то после циклического сдвига влево на 30 позиций он окажет влияние на 32 бит блока $c^{(i+3)}$, то есть изменится бит $c_{31}^{(i+3)}$
	$d^{(i+3)} = c^{(i+2)}$;	–	Без изменений
	$e^{(i+3)} = d^{(i+2)}$;	–	
4	$a^{(i+4)} = W^{(i+3)} \oplus (a^{(i+3)} <<< 5) \oplus b^{(i+3)} \oplus c^{(i+3)} \oplus d^{(i+3)} \oplus e^{(i+3)} \oplus K^{(i+3)}$;	$a_{16}^{(i+4)}$ $a_{31}^{(i+4)}$	Так как изменены биты $a_1^{(i+3)}$ и $a_{11}^{(i+3)}$, то после сдвига влево на 5 позиций они окажут влияние соответственно на 7 и 17 биты блока $a^{(i+4)}$, то есть изменятся биты $a_6^{(i+4)}$ и $a_{16}^{(i+4)}$. Так как изменен бит $b_6^{(i+3)}$, то бит $a_6^{(i+4)}$ должен быть изменен еще раз. Таким образом, бит $a_6^{(i+4)}$ вернется в свое первоначальное значение. Кроме того, так как изменен бит $c_{31}^{(i+3)}$, то изменится бит $a_{31}^{(i+4)}$

Окончание табл. 2

№ цикла	Преобразование	Измененные биты	Комментарий
	$b^{(i+4)} = a^{(i+3)}$;	$b_1^{(i+4)}$ $b_{11}^{(i+4)}$	Так как изменены биты $a_1^{(i+3)}$ и $a_{11}^{(i+3)}$, то изменятся биты $b_1^{(i+4)}$ и $b_{11}^{(i+4)}$
	$c^{(i+4)} = (b^{(i+3)} <<< 30)$;	$c_4^{(i+4)}$	Так как изменен бит $b_6^{(i+3)}$, то после циклического сдвига влево на 30 позиций он окажет влияние на 5 бит блока $c^{(i+4)}$, то есть изменится бит $c_4^{(i+4)}$
	$d^{(i+4)} = c^{(i+3)}$;	$d_{31}^{(i+4)}$	Так как изменен бит $c_{31}^{(i+3)}$, то изменится бит $d_{31}^{(i+4)}$
	$e^{(i+4)} = d^{(i+3)}$;	–	Без изменений
5	$a^{(i+5)} = W^{(i+4)} \oplus (a^{(i+4)} <<< 5) \oplus b^{(i+4)} \oplus c^{(i+4)} \oplus d^{(i+4)} \oplus e^{(i+4)} \oplus K^{(i+4)}$;	$a_1^{(i+5)}$ $a_{11}^{(i+5)}$ $a_{21}^{(i+5)}$ $a_{31}^{(i+5)}$	Так как изменены биты $a_{16}^{(i+4)}$ и $a_{31}^{(i+4)}$, то после сдвига влево на 5 позиций они окажут влияние соответственно на 22 и 5 биты блока $a^{(i+5)}$, то есть изменятся биты $a_{21}^{(i+5)}$ и $a_4^{(i+5)}$. Так как изменены биты $b_1^{(i+4)}$ и $b_{11}^{(i+4)}$, то изменятся и биты $a_1^{(i+5)}$ и $a_{11}^{(i+5)}$. Так как изменен бит $c_4^{(i+4)}$, то бит $a_4^{(i+5)}$ должен быть изменен еще раз. Таким образом, бит $a_4^{(i+5)}$ вернется в свое первоначальное значение. Кроме того, так как изменен бит $d_{31}^{(i+4)}$, то изменится бит $a_{31}^{(i+5)}$
	$b^{(i+5)} = a^{(i+4)}$;	$b_{16}^{(i+5)}$ $b_{31}^{(i+5}$	Так как изменены биты $a_{16}^{(i+4)}$ и $a_{31}^{(i+4)}$, то изменятся биты $b_{16}^{(i+5)}$ и $b_{31}^{(i+5)}$
	$c^{(i+5)} = (b^{(i+4)} <<< 30)$;	$c_9^{(i+5)}$ $c_{31}^{(i+5)}$	Так как изменены биты $b_1^{(i+4)}$ и $b_{11}^{(i+4)}$, то после циклического сдвига влево на 30 позиций они окажут влияние соответственно на 32 и 10 биты блока $c^{(i+5)}$, то есть изменятся биты $c_{31}^{(i+5)}$ и $c_9^{(i+5)}$
	$d^{(i+5)} = c^{(i+4)}$;	$d_4^{(i+5)}$	Так как изменен бит $c_4^{(i+4)}$, то изменится бит $d_4^{(i+5)}$
	$e^{(i+5)} = d^{(i+4)}$;	$e_{31}^{(i+5)}$	Так как изменен бит $d_{31}^{(i+4)}$, то изменится бит $e_{31}^{(i+5)}$

Для того, чтобы предотвратить последующие изменения битов, необходимо провести коррекцию, изменив биты в блоках W: $W_6^{(i+1)}$, $W_1^{(i+2)}$, $W_{31}^{(i+3)}$, $W_{31}^{(i+4)}$ и $W_{31}^{(i+5)}$. Так, изменение бита $W_6^{(i+1)}$ позволит оставить неизменным бит $a_6^{(i+1)}$, что, в свою очередь, предотвратит изменение битов при дальнейших преобразованиях, которые были представлены в табл. 2. Преобразования битов с проведенной коррекцией представлены в табл. 3.

Таблица 3

Коррекция битов

№ цикла	Преобразование	Коррекция	Измененные биты	Комментарий
1	$a^{(i+1)} = W^{(i)} \oplus (a^{(i)} <<< 5) \oplus b^{(i)} \oplus c^{(i)} \oplus d^{(i)} \oplus e^{(i)} \oplus K^{(i)}$;	–	$a_1^{(i+1)}$	Так как изменен бит $W_1^{(i)}$, то изменится значение бита $a_1^{(i+1)}$
2	$a^{(i+2)} = W^{(i+1)} \oplus (a^{(i+1)} <<< 5) \oplus b^{(i+1)} \oplus c^{(i+1)} \oplus d^{(i+1)} \oplus e^{(i+1)} \oplus K^{(i+1)}$;	$W_6^{(i+1)}$	–	Так как должен измениться бит $a_6^{(i+1)}$, то после коррекции он останется неизменным
	$b^{(i+2)} = a^{(i+1)}$;	–	$b_1^{(i+2)}$	Так как изменен бит $a_1^{(i+1)}$, то изменится бит $b_1^{(i+2)}$
3	$a^{(i+3)} = W^{(i+2)} \oplus (a^{(i+2)} <<< 5) \oplus b^{(i+2)} \oplus c^{(i+2)} \oplus d^{(i+2)} \oplus e^{(i+2)} \oplus K^{(i+2)}$;	$W_1^{(i+2)}$	–	Так как изменен бит $b_1^{(i+2)}$, то должен измениться бит $a_1^{(i+3)}$, однако в результате коррекции этого не происходит
	$c^{(i+3)} = (b^{(i+2)} <<< 30)$;	–	$c_{31}^{(i+3)}$	Так как изменен бит $b_1^{(i+2)}$, то после циклического сдвига влево на 30 позиций он окажет влияние на 32 бит блока $c^{(i+3)}$, то есть изменится бит $c_{31}^{(i+3)}$
4	$a^{(i+4)} = W^{(i+3)} \oplus (a^{(i+3)} <<< 5) \oplus b^{(i+3)} \oplus c^{(i+3)} \oplus d^{(i+3)} \oplus e^{(i+3)} \oplus K^{(i+3)}$;	$W_{31}^{(i+3)}$	–	Так как изменен бит $c_{31}^{(i+3)}$, то должен измениться бит $a_{31}^{(i+4)}$, однако в результате коррекции этого не происходит
	$d^{(i+4)} = c^{(i+3)}$;	–	$d_{31}^{(i+4)}$	Так как изменен бит $c_{31}^{(i+3)}$, то изменится бит $d_{31}^{(i+4)}$
5	$a^{(i+5)} = W^{(i+4)} \oplus (a^{(i+4)} <<< 5) \oplus b^{(i+4)} \oplus c^{(i+4)} \oplus d^{(i+4)} \oplus e^{(i+4)} \oplus K^{(i+4)}$;	$W_{31}^{(i+4)}$	–	Так как изменен бит $d_{31}^{(i+4)}$, то должен измениться бит $a_{31}^{(i+5)}$, однако в результате коррекции этого не происходит
	$e^{(i+5)} = d^{(i+4)}$;	–	$e_{31}^{(i+5)}$	Так как изменен бит $d_{31}^{(i+4)}$, то изменится бит $e_{31}^{(i+5)}$.
6	$a^{(i+6)} = W^{(i+5)} \oplus (a^{(i+5)} <<< 5) \oplus b^{(i+5)} \oplus c^{(i+5)} \oplus d^{(i+5)} \oplus e^{(i+5)} \oplus K^{(i+5)}$;	$W_{31}^{(i+5)}$	–	Так как изменен бит $e_{31}^{(i+5)}$, то должен измениться бит $a_{31}^{(i+6)}$, однако в результате коррекции этого не происходит

Таким образом, из табл. 2 и 3 можно видеть, что изменение битов $W_6^{(i+1)}$, $W_1^{(i+2)}$, $W_{31}^{(i+3)}$, $W_{31}^{(i+4)}$ позволяет получить два различных преобразования от значений ($a^{(i)}$, $b^{(i)}$, $c^{(i)}$, $d^{(i)}$, $e^{(i)}$) к значениям ($a^{(i+6)}$, $b^{(i+6)}$, $c^{(i+6)}$, $d^{(i+6)}$, $e^{(i+6)}$), что ведет к локальной коллизии. Схематично это изображено на рис. 2.

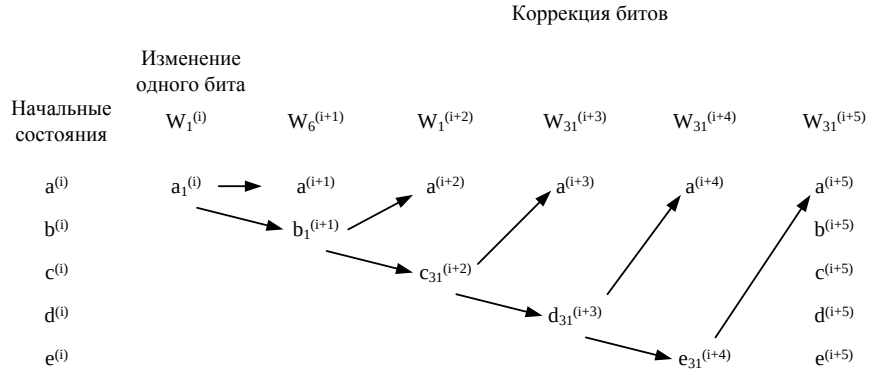

Рис. 2. Получение локальной коллизии

Важно отметить, что преобразования, рассмотренные для бита $W_1^{(i)}$, можно рассмотреть для любого бита блока $W^{(i)}$, однако, как отмечается в работе [5] выбор бита $W_1^{(i)}$ приводит к наилучшему результату.

Так как рассматриваемая версия функции линейна, то можно сымитировать столько локальных коллизий, сколько необходимо, и получить два различных преобразования от $(a^{(0)}, b^{(0)}, c^{(0)}, d^{(0)}, e^{(0)})$ до $(a^{(80)}, b^{(80)}, c^{(80)}, d^{(80)}, e^{(80)})$: первое преобразование с использованием исходного сообщения W и второе преобразование с использованием модифицированного сообщения W'. Остается выяснить, как определить такую локальную коллизию, которая бы учитывала тот факт, что подблоки $W^{(i)}$ и $W^{(i)'}$, начиная со значения i=16, вырабатываются в соответствии с формулой (2).

Определить локальную коллизию означает построить вектор m0, состоящий из 80 битов, пронумерованных от 0 до 79, со значением 1 в позиции i, если мы хотим обратить бит $W_1^{(i)}$. Однако нельзя обращать биты $W_1^{(i)}$ для значений $i \geq 75$, так как изменения в раунде под номером i не будут скорректированы до раунда номер (i+6), а нам необходимо, чтобы все изменения были скорректированы до 80 раунда.

Пусть $(m0^{(0)}, \ldots, m0^{(79)})$ будет одним из таких векторов. Исходя из этого, необходимо определить маску изменений для сообщения W. Для того, чтобы провести коррекцию, будем использовать маску L, состоящую из 80 32-разрядных блоков, то есть $L = (L'^{(0)} \ldots L'^{(80)})$. Для получения каждого блока L(i) введем шесть корректирующих масок L0, L1, L2, L3, L4, L5. Для первой корректирующей маски добавим еще пять нулевых блоков с i = –5 по i = –1 и заполним ее следующим образом:

$$L0^{(i)} = 0, \text{ для } -5 \leq i \leq 0;$$

$$L0_k^{(i)} = 0, \text{ для } 0 \leq i \leq 79 \text{ и для всех } k \neq 1;$$

$$L0_1^{(i)} = m_0^{(i)}, \text{ для } 0 \leq i \leq 79.$$

Для второй корректирующей маски L1 добавим еще четыре нулевых блока с i = – 4 по i = –1. Вторая корректирующая маска L1 вырабатывается из L0 с помощью циклического сдвига влево на 5 позиций. Сдвиг вытекает из описания преобразований функции SHA, что позволит использовать ее для коррекции бита k=6. Имеем:

$$L1^{(i)} = L0^{(i-1)} <<< 5, \text{ для } -4 \leq i \leq 79. \quad (3)$$

Следующая корректирующая маска L2 имеет три дополнительных нулевых блока и вырабатывается из L0 без использования циклического сдвига:

$$L2^{(i)} = L0^{(i-2)}, \text{ для } -3 \leq i \leq 79. \quad (4)$$

Аналогичным образом из L0 получаются корректирующие маски L3, L4, L5, имеющие соответственно два, один и ноль дополнительных нулевых блоков. Маски L3, L4, L5 вырабатываются с помощью циклического сдвига влево на 30 позиций и используются для коррекции бита номер k=31:

$$L3^{(i)} = L0^{(i-3)} <<< 30, \text{ для } -2 \le i \le 79; \quad (5)$$

$$L4^{(i)} = L0^{(i-4)} <<< 30, \text{ для } -1 \le i \le 79; \quad (6)$$

$$L5^{(i)} = L0^{(i-5)} <<< 30, \text{ для } 0 \le i \le 79. \quad (7)$$

Искомая маска L определяется путем сложения по модулю два соответствующих блоков из корректирующих масок L0, L1, L2, L3, L4, L5:

$$L^{(i)} = L0^{(i)} \oplus L1^{(i)} \oplus L2^{(i)} \oplus L3^{(i)} \oplus L4^{(i)} \oplus L5^{(i)}, \text{ для } 0 \le i \le 79. \quad (8)$$

Для того, чтобы более детально понять принцип нахождения маски L, рассмотрим нахождение ее первых пяти блоков в случае, если в маске m0 бит $m0^{(0)}$ равен единице, а остальные биты равны нулю, как это сделано в табл. 4. В табл. 4 нумерация блоков начинается со значения i = –5. Первые блоки до значения i = 0 в корректирующих масках L0, L1, L2, L3, L4, L5 заполняются нулями. В блок $L0^{(0)}$ в позицию 1 заносится значение $m0^{(0)}$. Остальные блоки L0 остаются равными 0. В соответствии с формулами (3) – (7) производится заполнение блоков для корректирующих масок L1, L2, L3, L4, L5 и по формуле (8) вычисление маски L. В результате можно видеть, что первый блок $L^{(0)}$ задаст обращение бита для $W'^{(0)}$, а блоки $L^{(1)}$, $L^{(2)}$, $L^{(3)}$, $L^{(4)}$, $L^{(5)}$ позволят провести коррекцию в соответствии со схемой, рассмотренной в табл. 2. Аналогичным образом можно произвести выработку корректирующей маски L для любого вектора m0.

Таблица 4

Построение корректирующей маски L

i	–5	–4	–3	–2	–1	0	1	2	3	4	5
L0	0	0	0	0	0	$L0_1^{(0)}=1$	0	0	0	0	0
L1	–	0	0	0	0	0	$L1_6^{(1)}=1$	0	0	0	0
L2	–	–	0	0	0	0	0	$L2_1^{(2)}=1$	0	0	0
L3	–	–	–	0	0	0	0	0	$L3_{31}^{(3)}=1$	0	0
L4	–	–	–	–	0	0	0	0	0	$L4_{31}^{(4)}=1$	0
L5	–	–	–	–	–	0	0	0	0	0	$L5_{31}^{(5)}=1$
L	–	–	–	–	–	$L_1^{(0)}=1$	$L_6^{(1)}=1$	$L_1^{(2)}=1$	$L_{31}^{(3)}=1$	$L_{31}^{(4)}=1$	$L_{31}^{(5)}=1$

При выборе вектора m0 необходимо помнить о том, что, начиная со значения i = 16, блоки $W^{(i)}$ вырабатываются в соответствии с формулой (2). Поэтому необходимо учитывать, что блоки корректирующего вектора L0, начиная с позиции 11, должны удовлетворять условию:

$$L0^{(i)} = L0^{(i-3)} \oplus L0^{(i-8)} \oplus L0^{(i-14)} \oplus L0^{(i-16)}, \text{ для } 11 \le i \le 80. \quad (9)$$

Позиция 11 обусловлена тем фактом, что, начиная с блока $W^{(16)}$, используется функция расширения (2). Следовательно, если в данном блоке необходимо провести коррекцию, то для этого нужно использовать блок корректирующей маски $L0^{(11)}$.

Так как преобразование (2) является полностью линейной операцией, которую можно проводить побитно, то можно представить результат работы данного преобразования в виде 32-х 80-разрядных блоков $V = (V^{(0)}, \ldots, V^{(32)})$. Каждый блок $V^{(i)}$ содержит 80 битов. Если обозначить индексом j номер бита в блоке $V^{(i)}$, то бит $V_j^{(i)}$ будет соответствовать биту $W_i^{(j)}$. При этом первые 16 битов (с 0 по 15) в блоке $V^{(i)}$ заполняются в соответствии с исходным значением $W = (W'^{(0)} \ldots W'^{(15)})$, а остальные могут быть получены с использованием формулы (2). Таким образом, до-

вольно легко осуществить перебор возможных значений для блока $V^{(i)}$. Из 2^{16} = 65536 возможных заполнений блока $V^{(i)}$ всего 128 вариантов удовлетворяют выражению (9) и содержат 5 нулей в позициях с 75 по 79, и таким образом, позволяют получить маску m0.

С использованием такой маски легко можно получить корректирующий вектор L. Зная преобразование (2), легко можно проделать обратные линейные вычисления и определить исходное 512-битовое сообщение μ, из которого получается маска L. Так как в рассматриваемой версии функции хэширования расширение блоков W выполняется с использованием одинаковых преобразований, то ясно, что $\mu = (L^{(0)}, \ldots, L^{(15)})$. Для всех входов $W = (W^{(0)}, \ldots, W^{(15)})$ и $W' = W \oplus \mu$ будут получены одинаковые выходные хэш-значения при использовании функции SHI1.

Теперь необходимо изучить появление коллизий при использовании функции $f^{(i)}$ в хэш-функциях семейства SHA. Рассмотрим еще один упрощенный вариант функции SHA, в котором функция расширения блоков работает в соответствии с преобразованием (2) и функция целочисленного сложения по модулю 2^{32} заменена на операцию XOR, выполняемую над пятью переменными. В работе [5] такая упрощенная версия функции SHA названа SHI2. Проще говоря, функция SHI2 – это функция SHI1 (рассмотренная нами ранее), в которую добавили нелинейную функцию $f^{(i)}$. Легко видеть, что в некоторых случаях (см. табл. 1) функция $f^{(i)}$ представляет собой линейное преобразование, заключающееся в сложении по модулю два трех операндов. Таким образом, рассмотренная ранее атака может работать. Остается выяснить, в каких случаях данная атака будет работать и какова вероятность успеха.

Для того, чтобы вычислить вероятность успеха, необходимо провести детальный анализ функции $f^{(i)}$, работающей в режимах IF и MAJ (см. табл. 1). Так как данные функции работают с 32 битами блока независимо, то необходимо всего лишь изучить, что произойдет при изменении одного бита. Рассмотрим поведение двух функций $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)})$ и $f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$ в случае, если имеются различия во входных блоках данных функций. Выделим четыре возможных варианта в соответствии с рассмотренной ранее схемой:

1. Входные блоки функций не имеют различий, то есть $B^{(i)} = B'^{(i)}$, $C^{(i)} = C'^{(i)}$, $D^{(i)} = D'^{(i)}$.
2. Имеется единственное различие в бите j=1 блока $B^{(i)}$, то есть $B'^{(i)} = B^{(i)} \oplus 2^1$.
3. Имеется единственное различие в бите j=31 блока $C^{(i)}$ или блока $D^{(i)}$.
4. Имеется два различия в бите j=31 блоков $C^{(i)}$ и $D^{(i)}$, то есть $C'^{(i)} = C^{(i)} \oplus 2^{31}$ и $D'^{(i)} = D^{(i)} \oplus 2^{31}$.

В первом случае оба варианта функции $f^{(i)}$ IF и MAJ будут давать одинаковые выходные значения, то есть в любом случае $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)}) = f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$, и значит функция $f^{(i)}$ оказывает воздействие на входные биты аналогично тому, как это было рассмотрено для функции $f^{(i)}$, работающей в режиме XOR.

Определим поведение функций для второго возможного варианта. Для этого построим таблицы истинности для функции $f^{(i)}$ в режиме IF (табл. 5) и в режиме MAJ (табл. 6). В табл. 5 и 6 последние два столбца представляют собой возможные выходные результаты для бита j = 1 соответственно функций $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)})$ и $f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$. При этом $B'^{(i)} = B^{(i)} \oplus 2^1$, $C^{(i)} = C'^{(i)}$, $D^{(i)} = D'^{(i)}$. Из них видно, что $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)}) \neq f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$ в том случае, если биты блоков C и D не равны, то есть, если $C_1^{(i)} \neq D_1^{(i)}$. И вероятность такого события равна ½. В этом случае выходы функций будут иметь различие в бите j=1, что будет скорректировано путем добавления бита W_1, согласно рассмотренной ранее схеме. Это справедливо как для функции $f^{(i)}$ в режиме IF, так и для функции $f^{(i)}$ в режиме MAJ.

Таблица 5

Таблица истинности для функции $f^{(i)}$ в режиме IF, если $B'^{(i)} = B^{(i)} \oplus 2^1$

B	¬B	B’	¬B’	C	D	B∧C=Y1	B’∧C=Y2	(¬B)∧D=Y3	(¬B’)∧D=Y4	Y1∨Y3	Y2∨Y4
0	1	1	0	0	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	1	0	1	0
0	1	1	0	1	0	0	1	0	0	0	1
0	1	1	0	1	1	0	1	1	0	1	1
1	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	1	0	1
1	0	0	1	1	0	1	0	0	0	1	0
1	0	0	1	1	1	1	0	0	1	1	1

Таблица 6

Таблица истинности для функции $f^{(i)}$ в режиме MAJ, если $B'^{(i)} = B^{(i)} \oplus 2^1$

B	B’	C	D	B∧C=Y1	B’∧C=Y2	B∧D=Y3	B’∧D=Y4	C∧D=Y5	Y1∨Y3=Y6	Y2∨Y4=Y7	Y6∨Y5	Y7∨Y5
0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	1	0	0	1	0	1
0	1	1	0	0	1	0	0	0	0	1	0	1
0	1	1	1	0	1	0	1	1	0	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	1	0	1	0
1	0	1	0	1	0	0	0	0	1	0	1	0
1	0	1	1	1	0	1	0	1	1	0	1	1

Рассмотрим третий возможный вариант, согласно которому должно иметься единственное различие в бите j=31 блока $C^{(i)}$ или блока $D^{(i)}$.

Построим таблицы истинности для функции $f^{(i)}$ в режиме IF соответственно для случаев, когда изменяется бит $C_{31}^{(i)}$ (табл. 7) или бит $D_{31}^{(i)}$ (табл. 8). В случае, если изменен бит $C_{31}^{(i)}$, то выполняются следующие условия: $B^{(i)} = B'^{(i)}$, $C'^{(i)} = C^{(i)} \oplus 2^{31}$, $D^{(i)} = D'^{(i)}$. В случае, если изменен бит $D_{31}^{(i)}$, то выполняются следующие условия: $B^{(i)} = B'^{(i)}$, $C^{(i)} = C'^{(i)}$, $D'^{(i)} = D^{(i)} \oplus 2^{31}$. В табл. 7 и 8 последние два столбца представляют собой возможные выходные результаты для бита j = 31 соответственно функций $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)})$ и $f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$. Из них видно, что $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)}) \neq f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$ в том случае, если бит $B_{31}^{(i)} = 1$ и при этом меняется бит блока C, то есть $C'^{(i)} = C^{(i)} \oplus 2^{31}$; или если бит $B_{31}^{(i)} = 0$ и при этом меняется бит блока D, то есть $D'^{(i)} = D^{(i)} \oplus 2^{31}$. Вероятность этих событий равна ½. Выходы функций будут иметь различие в бите j=31, что будет скорректировано путем добавления бита W_{31}, согласно рассмотренной ранее схеме.

Таблица 7

Таблица истинности для функции $f^{(i)}$ в режиме IF, если $C'^{(i)} = C^{(i)} \oplus 2^{31}$

C	C’	B	¬B	D	B∧C=Y1	B∧C’=Y2	(¬B)∧D=Y3	Y1∨Y3	Y2∨Y3
0	1	0	1	0	0	0	0	0	0
0	1	0	1	1	0	0	1	1	1
0	1	1	0	0	0	1	0	0	1
0	1	1	0	1	0	1	0	0	1
1	0	0	1	0	0	0	0	0	0
1	0	0	1	1	0	0	1	1	1
1	0	1	0	0	1	0	0	1	0
1	0	1	0	1	1	0	0	1	0

Таблица 8

Таблица истинности для функции $f^{(i)}$ в режиме IF, если $D'^{(i)} = D^{(i)} \oplus 2^{31}$

D	D’	B	¬B	C	B∧C=Y1	(¬B)∧D=Y2	(¬B)∧D’=Y3	Y1∨Y3	Y2∨Y3
0	1	0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	1	1	1
0	1	1	0	0	0	0	0	0	0
0	1	1	0	1	1	0	0	1	0
1	0	0	1	0	0	1	0	0	1
1	0	0	1	1	0	1	0	0	1
1	0	1	0	0	0	0	0	0	0
1	0	1	0	1	1	0	0	1	0

Аналогичным образом построим таблицы истинности для функции $f^{(i)}$ в режиме MAJ соответственно для случаев, когда изменяется бит $C_{31}^{(i)}$ (табл. 9) или бит $D_{31}^{(i)}$ (табл. 10). В случае, если изменен бит $C_{31}^{(i)}$, то выполняются следующие условия: $B^{(i)} = B'^{(i)}$, $C'^{(i)} = C^{(i)} \oplus 2^{31}$, $D^{(i)} = D'^{(i)}$. В случае, если изменен бит $D_{31}^{(i)}$, то выполняются следующие условия: $B^{(i)} = B'^{(i)}$, $C^{(i)} = C'^{(i)}$, $D'^{(i)} = D^{(i)} \oplus 2^{31}$. В табл. 9 и 10 последние два столбца представляют собой возможные выходные результаты для бита j = 31 соответственно функций $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)})$ и $f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$. Из них видно, что $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)}) \neq f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$ в том случае, если бит $B_{31}^{(i)} \neq D_{31}^{(i)}$ и при этом меняется бит блока C, то есть $C'^{(i)} = C^{(i)} \oplus 2^{31}$; или если бит $B_{31}^{(i)} \neq C_{31}^{(i)}$ 1 и при этом меняется бит блока D, то есть $D'^{(i)} = D^{(i)} \oplus 2^{31}$. Вероятность этих событий также равна ½. Выходы функций будут иметь различие в бите j=31, что будет скорректировано путем добавления бита W_{31}, согласно рассмотренной ранее схеме.

Таблица 9

Таблица истинности для функции $f^{(i)}$ в режиме MAJ, если $C'^{(i)} = C^{(i)} \oplus 2^{31}$

	’			B∧ C=Y1	B ∧C’=Y2	B ∧D=Y3	C ∧D=Y4	C’ ∧D=Y5	Y1∨ Y3∨Y4	Y2∨ Y3∨Y5
				0	0	0	0	0	0	0
				0	0	0	0	1	0	1
				0	1	0	0	0	0	1
				0	1	1	0	1	1	1
				0	0	0	0	0	0	0
				0	0	0	1	0	1	0
				1	0	0	0	0	1	0
				1	0	1	1	0	1	1

Таблица 10

Таблица истинности для функции $f^{(i)}$ в режиме MAJ, если $D'^{(i)} = D^{(i)} \oplus 2^{31}$

D	D’	B	C	B∧C=Y1	B∧D=Y2	B∧D’=Y3	C∧D=Y4	C∧D’=Y5	Y1∨Y2∨Y4	Y1∨Y3∨Y5
0	1	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	1	0	1
0	1	1	0	0	0	1	0	0	0	1
0	1	1	1	1	0	1	0	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	1	0
1	0	1	0	0	1	0	0	0	1	0
1	0	1	1	1	1	0	1	0	1	1

Наконец рассмотрим четвертый вариант, согласно которому должно иметься два различия в бите j=31 блоков $C^{(i)}$ и $D^{(i)}$, то есть $C'^{(i)} = C^{(i)} \oplus 2^{31}$ и $D'^{(i)} = D^{(i)} \oplus 2^{31}$. Построим таблицы истинности для функции $f^{(i)}$ в режиме IF (табл. 11) и для функции $f^{(i)}$ в режиме MAJ (табл. 12). При этом выполняются следующие условия: $B^{(i)} = B'^{(i)}$, $C'^{(i)} = C^{(i)} \oplus 2^{31}$, $D'^{(i)} = D^{(i)} \oplus 2^{31}$. Из табл. 11 видно, что функция $f^{(i)}$ в ре-

жиме IF никогда не даст на выходе два одинаковых бита при соответствующих начальных условиях, то есть вероятность того, что $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)}) \neq f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$ всегда равна единице. Так как во входных значениях имеется двойное различие в бите j=31, то в результате сложения по модулю два эти различия будут компенсировать друг друга. Таким образом, для корректной работы необходимо, чтобы на выходе функций $f^{(i)}$ различий не было, что невозможно в соответствии с таблицей истинности. Поэтому в первых 19 раундах необходимо использовать такие корректирующие маски, которые не допустят поступление на вход функции $f^{(i)}$ значений $C^{(i)}$ и $D^{(i)}$, имеющих одновременное различие в бите j=31. Из табл. 12 видно, что $f^{(i)}(B^{(i)}, C^{(i)}, D^{(i)}) = f^{(i)}(B'^{(i)}, C'^{(i)}, D'^{(i)})$ в том случае, если бит $C_{31}^{(i)}$ не равен биту $D_{31}^{(i)}$ ($C_{31}^{(i)} \neq D_{31}^{(i)}$). Вероятность такого события равна ½.

Таблица 11

Таблица истинности для функции $f^{(i)}$ в режиме IF, если $C'^{(i)} = C^{(i)} \oplus 2^{31}$ и $D'^{(i)} = D^{(i)} \oplus 2^{31}$

B	¬B	C	C'	D	D'	B∧C=Y1	(¬B)∧D=Y2	B∧C'=Y3	(¬B')∧D'=Y4	Y1∨Y2	Y3∨Y4
0	1	0	1	0	1	0	0	0	1	0	1
0	1	0	1	1	0	0	1	0	0	1	0
1	0	0	1	0	1	0	0	1	0	0	1
1	0	0	1	1	0	0	0	1	0	0	1
0	1	1	0	0	1	0	0	0	1	0	1
0	1	1	0	1	0	0	1	0	0	1	0
1	0	1	0	0	1	1	0	0	0	1	0
1	0	1	0	1	0	1	0	0	0	1	0

Таблица 12

Таблица истинности для функции $f^{(i)}$ в режиме MAJ, если $C'^{(i)} = C^{(i)} \oplus 2^{31}$ и $D'^{(i)} = D^{(i)} \oplus 2^{31}$

B	C	C'	D	D'	B∧C=Y1	B∧C'=Y2	B∧D=Y3	B∧D'=Y4	C∧D=Y5	C'∧D'=Y6	Y1∨Y3∨Y5	Y2∨Y4∨Y6
0	0	1	0	1	0	0	0	0	0	1	0	1
0	0	1	1	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1
1	0	1	1	0	0	1	1	0	0	0	1	1
0	1	0	0	1	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	1	0
1	1	0	0	1	1	0	0	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0

Основываясь на проведенном анализе функций $f^{(i)}$, выберем начальный вектор m0 такой, чтобы до раунда i=16 не встречалось двух измененных битов, стоящих рядом. Начиная с раунда i=16, такие изменения могут присутствовать, так как в данном случае одновременные изменения в битах $C_{31}^{(i)}$ и $D_{31}^{(i)}$ произойдут не раньше раунда i=20, в котором используется функция $f^{(i)}$ в режиме XOR. При этом необходимо помнить, что значения вектора m0 должны удовлетворять выражению (9). В качестве такого вектора используем:

m0 = (0,0,1,0,0,0,1,0,0,0,0,0,0,0,1,0,1,1,1,1,
0,1,1,0,0,0,1,1,1,0,0,0,0,0,0,1,0,1,0,0,
0,1,0,0,0,1,0,0,1,0,0,1,0,0,1,1,1,0,1,1,
0,0,1,1,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0,0).

Согласно оговоренным условиям, вектор m0 не содержит двух единиц, стоящих рядом в первых 16 позициях, а содержит пять нулей в последних позициях. Воспользовавшись формулами (3) – (8), легко можно получить корректирующий вектор L. Блоки вектора L представлены в табл. 13 в 7 столбце. Для того, чтобы

определить вероятность получения коллизии при использовании выбранного вектора m0, рассмотрим, какие значения будут поступать на вход функции $f^{(i)}$ в каждом раунде преобразования. При этом для функции $f^{(i)}$, работающей в режимах IF и MAJ, будем использовать соответственно обозначения IF(C) и MAJ(C), где C может принимать значения от 1 до 4 и указывает на номер одного из четырех рассмотренных вариантов различий во входных данных функции $f^{(i)}$. В табл. 13 представлено изменение значений разностей для значений A, B, C, D и E каждого раунда преобразования с тем, чтобы определить различия во входах функции $f^{(i)}$. При этом опущены раунды номер 0, 1, 12, 13, 34, 78 и 79, в которых значения переменных $A^{(i)}$, $B^{(i)}$, $C^{(i)}$, $D^{(i)}$, $E^{(i)}$ и $L^{(i)}$ равны нулю.

Таблица 13

Определение вероятности получения коллизии при использовании выбранного вектора m0

№	A	B	C	D	E	L	$f^{(i)}$	p
2	00000000	00000000	00000000	00000000	00000000	00000002	IF(1)	1
3	00000002	00000000	00000000	00000000	00000000	00000040	IF(1)	1
4	00000000	00000002	00000000	00000000	00000000	00000002	IF(2)	½
5	00000000	00000000	80000000	00000000	00000000	80000000	IF(3)	½
6	00000000	00000000	00000000	80000000	00000000	80000002	IF(3)	½
7	00000002	00000000	00000000	00000000	80000000	80000040	IF(1)	1
8	00000000	00000002	00000000	00000000	00000000	00000002	IF(2)	½
9	00000000	00000000	80000000	00000000	00000000	80000000	IF(3)	½
10	00000000	00000000	00000000	80000000	00000000	80000000	IF(3)	½
11	00000000	00000000	00000000	00000000	80000000	80000000	IF(1)	1
14	00000000	00000000	00000000	00000000	00000000	00000002	IF(1)	1
15	00000002	00000000	00000000	00000000	00000000	00000040	IF(1)	1
16	00000000	00000002	00000000	00000000	00000000	00000000	IF(2)	½
17	00000002	00000000	80000000	00000000	00000000	80000042	IF(3)	½
18	00000002	00000002	00000000	80000000	00000000	80000040	IF(2,3)	¼
19	00000002	00000002	80000000	00000000	80000000	00000040	IF(2,3)	¼
20	00000002	00000002	80000000	80000000	00000000	00000042	XOR	1
21	00000000	00000002	80000000	80000000	80000000	80000000	XOR	1
22	00000002	00000000	80000000	80000000	80000000	80000042	XOR	1
23	00000002	00000002	00000000	80000000	80000000	00000042	XOR	1
24	00000000	00000002	80000000	00000000	80000000	00000002	XOR	1
25	00000000	00000000	80000000	80000000	00000000	00000000	XOR	1
26	00000000	00000000	00000000	80000000	80000000	00000002	XOR	1
27	00000002	00000000	00000000	00000000	80000000	80000042	XOR	1
28	00000002	00000002	00000000	00000000	00000000	00000040	XOR	1
29	00000002	00000002	80000000	00000000	00000000	80000042	XOR	1
30	00000000	00000002	80000000	80000000	00000000	00000002	XOR	1
31	00000000	00000000	80000000	80000000	80000000	80000000	XOR	1
32	00000000	00000000	00000000	80000000	80000000	00000000	XOR	1
33	00000000	00000000	00000000	00000000	80000000	80000000	XOR	1
35	00000000	00000000	00000000	00000000	00000000	00000002	XOR	1
36	00000002	00000000	00000000	00000000	00000000	00000040	XOR	1
37	00000000	00000002	00000000	00000000	00000000	00000000	XOR	1
38	00000002	00000000	80000000	00000000	00000000	80000040	XOR	1
39	00000000	00000002	00000000	80000000	00000000	80000002	XOR	1
40	00000000	00000000	80000000	00000000	80000000	00000000	MAJ(3)	½
41	00000000	00000000	00000000	80000000	00000000	80000002	MAJ(3)	½
42	00000002	00000000	00000000	00000000	80000000	80000040	MAJ(1)	1
43	00000000	00000002	00000000	00000000	00000000	00000002	MAJ(2)	½
44	00000000	00000000	80000000	00000000	00000000	80000000	MAJ(3)	½
45	00000000	00000000	00000000	80000000	00000000	80000002	MAJ(3)	½
46	00000002	00000000	00000000	00000000	80000000	80000040	MAJ(1)	1
47	00000000	00000002	00000000	00000000	00000000	00000002	MAJ(2)	½
48	00000000	00000000	80000000	00000000	00000000	80000002	MAJ(3)	½

Окончание табл. 13

49	00000002	00000000	00000000	80000000	00000000	80000040	MAJ(3)	½
50	00000000	00000002	00000000	00000000	80000000	80000002	MAJ(2)	½
51	00000000	00000000	80000000	00000000	00000000	80000002	MAJ(3)	½
52	00000002	00000000	00000000	80000000	00000000	80000040	MAJ(3)	½
53	00000000	00000002	00000000	00000000	80000000	80000002	MAJ(2)	½
54	00000000	00000000	80000000	00000000	00000000	80000002	MAJ(3)	½
55	00000002	00000000	00000000	80000000	00000000	80000042	MAJ(3)	½
56	00000002	00000002	00000000	00000000	80000000	80000040	MAJ(2)	½
57	00000002	00000002	80000000	00000000	00000000	80000042	MAJ(2,3)	¼
58	00000000	00000002	80000000	80000000	00000000	00000000	MAJ(2,4)	¼
59	00000002	00000000	80000000	80000000	80000000	80000042	MAJ(4)	½
60	00000002	00000002	00000000	80000000	80000000	00000042	XOR	1
61	00000000	00000002	80000000	00000000	80000000	00000002	XOR	1
62	00000000	00000000	80000000	80000000	00000000	00000002	XOR	1
63	00000002	00000000	00000000	80000000	80000000	00000042	XOR	1
64	00000002	00000002	00000000	00000000	80000000	80000042	XOR	1
65	00000000	00000002	80000000	00000000	00000000	80000002	XOR	1
66	00000000	00000000	80000000	80000000	00000000	00000000	XOR	1
67	00000000	00000000	00000000	80000000	80000000	00000000	XOR	1
68	00000000	00000000	00000000	00000000	80000000	80000002	XOR	1
69	00000002	00000000	00000000	00000000	00000000	00000042	XOR	1
70	00000002	00000002	00000000	00000000	00000000	00000040	XOR	1
71	00000002	00000002	80000000	00000000	00000000	80000040	XOR	1
72	00000002	00000002	80000000	80000000	00000000	00000040	XOR	1
73	00000002	00000002	80000000	80000000	80000000	80000042	XOR	1
74	00000000	00000002	80000000	80000000	80000000	80000002	XOR	1
75	00000000	00000000	80000000	80000000	80000000	80000000	XOR	1
76	00000000	00000000	00000000	80000000	80000000	00000000	XOR	1
77	00000000	00000000	00000000	00000000	80000000	80000000	XOR	1
Итоговая вероятность:								2^{-32}

Таким образом, исходя из данных табл. 13, можно сделать вывод о том, что при использовании вектора m0 мы можем найти коллизию для функции SHI2 с вероятностью $p = 2^{-32}$. В работе [5] предлагается следующий подход к поиску двух сообщений W и W', для которых будет выработано одно и то же хэш-значение. Так как входное сообщение используется без изменений в первых раундах функции хэширования, то возможно разделить поиск сообщений на два этапа. Сначала необходимо найти первые 15 блоков сообщения W: $W^{(0)}$ … $W^{(14)}$ таких, чтобы при использовании парного сообщения W' = W ⊕ L функция $f^{(i)}$ имела различия в тех же позициях, в которых имелись различия у входных параметров функции. Согласно табл. 13 вероятность такого события равна 2^{-6}, следовательно, поиск не составит труда.

После того, как первые 15 блоков сообщения W будут найдены, необходимо перебрать достаточно много возможных значений для блока $W^{(15)}$ (естественно это число должно быть гораздо меньше 2^{32}, обычно около 10000 значений) с тем, чтобы оставшиеся раунды для сообщений W и W' выполнялись в соответствии с табл. 13. Согласно табл. 13, вероятность для нахождения такого сообщения $W^{(15)}$ равна 2^{-26}. Так как первая часть поиска можно проделать один раз, после чего подобрать множество значений $W^{(15)}$, то можно сказать, что сложность анализа равна 2^{26}.

Пример двух сообщений W и W', найденных в соответствии с вектором m0, удовлетворяющих данным табл. 13 и выдающих одинаковое хэш-значение **1334f224 21a3efc9 b667d2b2 2890013b 56013ca9** при использовании функции SHI2, приведен в табл. 14.

Таблица 14

Пример двух сообщений W и W', выдающих одинаковое хэш-значение

i	W	W'	L	i	W	W'	L
0	(1,a6191b0)	1a6191b0	00000000	8	2270fdbd	2270fdbf	00000002
1	3c4a331c	3c4a331c	00000000	9	2a8090f0	aa8090f0	80000000
2	1f228ea2	1f228ea0	00000002	10	4b12fd98	cb12fd98	80000000
3	403b7609	403b7649	00000040	11	473cc7a1	c73cc7a1	80000000
4	062ec496	062ec494	00000002	12	002831a9	002831a9	00000000
5	48611ca8	c8611ca8	80000000	13	50fe1535	50fe1535	00000000
6	583401bc	d83401be	80000002	14	61ac0d3d	61ac0d3f	00000002
7	399879d0	b9987990	80000040	15	f26700ec	f26700ac	00000040

Нам осталось рассмотреть еще одно криптографическое преобразование, входящее в состав функции хэширования SHA-0, А именно: целочисленное сложение по модулю 2^{32}. Сложность данной операции заключается в том, что разница в битах сообщений может привести к возникновению переноса, а следовательно число ненулевых битов разности может возрасти после применения данной операции. Если суметь предотвратить возникновение переноса, то дальнейший анализ можно будет провести по рассмотренным ранее схемам. На первый взгляд может показаться, что любой ненулевой бит разности может привести к возникновению переноса. Однако необходимо вспомнить, что при изменении бита $W_1^{(i)}$ необходимо провести коррекцию в битах $W_{31}^{(i+3)}$ $W_{31}^{(i+4)}$ $W_{31}^{(i+4)}$, то есть значение корректирующей разности будет иметь единицу в самом старшем бите, а все остальные биты разности будут равны нулю. В работах [6, 7] нами были рассмотрены свойства операции целочисленного сложения по модулю 2^n и выявлены ее основные свойства, которые сводятся к следующему:

1. Любое значение входной разности может отобразиться само в себя, то есть остаться неизменным. Вероятность такого отображения определяется следующим образом:

$$p = \frac{1}{2^k}, \text{ если входная разность } \Delta_{вх} < 2^{n-1}; \quad (10)$$

$$p = \frac{1}{2^{k-1}}, \text{ если входная разность } \Delta_{вх} \geq 2^{n-1}, \quad (11)$$

где k – число ненулевых позиций входной разности;

2. Для входной разности, равной Δвх=0, на выходе преобразования будет значение выходной разности Δвых = 0 с вероятностью p=1.

3. Для входной разности $\Delta вх = 2^{n-1}$ на выходе преобразования будет значение выходной разности $\Delta вых = 2^{n-1}$ с вероятностью p=1.

Таким образом, использование корректирующих разностей для блоков $W_{31}^{(i+3)}$ $W_{31}^{(i+4)}$ $W_{31}^{(i+4)}$ будет соответствовать третьему пункту правил преобразования разностей при целочисленном сложении, а значит, не будет оказывать влияния на изменение итоговой разности.

Предположим, что изначально бит $W_1^{(i)}$ равен 0, а бит $W'_1{}^{(i)}$ равен 1. Если при операции целочисленного сложения не возникнет переноса и разность останется неизмененной (что в соответствии с первым пунктом правил произойдет с вероятностью 1/2), тогда бит $a_1^{(i+1)}$ будет равен 0, и соответственно бит $a'_1{}^{(i+1)}$ будет равен 1. Рассматривая вычисление блока $a^{(i+2)}$, можно увидеть, что бит $W_6^{(i+1)}$ должен быть равен 1 (и соответственно бит $W'_6{}^{(i+1)}$ должен быть равен 0) для того, чтобы при

целочисленном сложении не возникло переноса и разность осталась неизменной. Если данное условие будет выполнено, то коррекция всегда будет проходить без возникновения переноса. Самым сложным моментом является коррекция при вычислении значения $a^{(i+3)}$. Для этого, как и раньше, примем, что бит $W_1^{(i+2)}$ равен 1, а бит $W'_1{}^{(i+2)}$ равен 0. В этом случае коррекция пройдет хорошо в том случае, если бит i=1 в выходном значении функции $f^{(i)}$ будет равен биту $b_1^{(i+2)}$ (или биту $a1^{(i+1)}$, так как они равны). Для этого необходимо, чтобы биты i=31 блоков $c^{(i+2)}$ и $d^{(i+2)}$ были равны друг другу ($c_{31}^{(i+2)} = d_{31}^{(i+2)}$), что может случиться с вероятностью ½.

В работе [5] отмечается, что если в блоке сообщения происходит изменение битов от 0 к 1, то оно должно быть скорректировано соответствующим изменением битов от 1 к 0. И наоборот, если происходит изменение битов от 1 к 0, то оно должно быть скорректировано соответствующим изменением битов от 0.

В работе [5] представлен пример построения вектора m0 и нахождения двух сообщений W и W', вырабатывающих одинаковые хэш-значения с использованием функции SHI3. Функция SHI3 представляет собой вариант функции SHI1, рассмотренной нами ранее, в котором элементы складываются не по модулю два, а по модулю 2^{32}, как это и должно быть в функции SHA.

Мы рассмотрели основные приемы дифференциального криптоанализа применительно к криптографическим примитивам, входящим в состав функций хэширования семейства SHA. Следующим этапом является проецирование данных приемов и способов на действительные функции хэширования SHA.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Основы: функции хэширования [Электронный ресурс], 03.06.2009. – http://www.cryptofaq.ru, свободный.
2. *Xiaoyun Wang, Hongbo Yu., Yiqun Lisa Yin*, Efficient Collision Search Attacks on SHA-0 [Электронный ресурс]. – http://citeseerx.ist.psu.edu, свободный.
3. *XiaoyunWang, Hongbo Yu.*, How to BreakMD5 and Other Hash Functions [Электронный ресурс]. – http://citeseerx.ist.psu.edu, свободный.
4. *XiaoyunWang, Yiqun Lisa Yin, Hongbo Yu.*, Finding Collisions in the Full SHA-1 [Электронный ресурс]. – http://people.csail.mit.edu/yiqun/pub.htm, свободный.
5. Florent Chabaund, Antoine Joux, Differential Collisions of SHA-0 [Электронный ресурс]. – 19 June, 2000. – http://fchabaud.free.fr /English/Publications/, свободный.
6. *Бабенко Л.К. Ищукова Е.А.* Применение рекурсивного алгоритма поиска в Б-деревьях для дифференциального криптоанализа алгоритма шифрования ГОСТ 28147-89 // Материалы IX Международной научно-практической конференции «Информационная безопасность». Часть 2. – Таганрог; Изд-во: ТТИ ЮФУ, 2007. – С. 92-97.
7. *Babenko L., Ischukova E.* Differential Analysis GOST Encryption Algorithm // Proceedings of the 3rd Internaternational Conference of Security of Information and Networks (SIN 2010), ACM, New York, 2010. – P.149-157.

Бабенко Людмила Климентьевна
Технологический институт федерального государственного автономного образовательного учреждения высшего профессионального образования «Южный федеральный университет» в г. Таганроге.
E-mail: blk@fib.tsure.ru.
347928, г. Таганрог, ул. Чехова, 2, корпус "И".
Тел.: 88634312018.

Ищукова Евгения Александровна
E-mail: jekky82@mail.ru.
Тел.: 88634371905.

Babenko Lyudmila Klimentevna
Taganrog Institute of Technology – Federal State-Owned Autonomy Educational Establishment of Higher Vocational Education "Southern Federal University".
E-mail: blk@fib.tsure.ru.
Block "I", 2, Chehov street, Taganrog, 347928, Russia.
Phone: +78634312018.

Ischukova Evgeniya Aleksandrovna
E-mail: jekky82@mail.ru.
Phone: +78634371905.

УДК 004.421.4

Л.К. Бабенко, И.Д. Сидоров, А.С. Кириллов

УСКОРЕНИЕ ВЫЧИСЛЕНИЙ ДИСКРЕТНОГО ЛОГАРИФМА С ПОМОЩЬЮ ТЕХНОЛОГИИ CUDA

Рассматриваются возможности дальнейшего ускорения реализации дискретного логарифмирования. Анализируется возможность применения технологии CUDA для ускорения вычислений на различных этапах. Рассматривается эффективная реализация необходимых арифметических операций. Приведены графики, построенные по результатам проведенных экспериментов.

Криптоанализ; дискретное логарифмирование; технология CUDA; параллельное программирование; вычислительно сложные задачи.

L.K. Babenko, I.D. Sidorov, A.S. Kirillov

SPEEDING UP DISCRETE LOG COMPUTATIONS USING CUDA TECHNOLOGY

Different possibilities for further discrete log performance increase are considered. The capabilities of CUDA technology for speeding up different parts of computation process are analyzed. Here we represent effective implementation of needful arithmetic operations. Some graphic materials illustrate results of experiments.

Cryptanalysis; discrete log problem; CUDA technology; parallel programming; computation intensive tasks.

Введение. Постановка задачи. В настоящее время широко распространены алгоритмы шифрования и цифровой подписи (такие, как Эль-Гамаль, DSA, ГОСТ), стойкость которых основана на сложности решения задачи дискретного логарифмирования в мультипликативной группе числового поля и в группе точек эллиптической кривой над конечным полем. Для произвольной циклической группы G эта задача формулируется следующим образом: по известным $a, b \in G$ найти такой логарифм x, что $a^b = x$.

Особенностью данной задачи является быстрый рост времени выполнения при увеличении размера задачи. Авторами был разработан ряд эффективных параллельных алгоритмов, предназначенных для ускорения решения задачи дискетного логарифмирования в различных группах с помощью распределённых многопроцессорных вычислений. Алгоритмы, предназначенные для мультипликативной группы числового поля, рассматриваются в работах [1,2], а алгоритмы для группы точек эллиптической кривой – в работе [3].