Обращение криптографической функции A5/1 на платформе GPU с применением альтернативных схем вычисления значений сдвиговых регистров*

В.Г. Булавинцев, А.А.Семенов

ИДСТУ СО РАН

Исследуется возможность обращения криптографической функции A5/1 с применением вычислительных ресурсов графических ускорителей общего назначения (GPU). Применение «атаки Андерсона» в реализации криптоанализа A5/1 методом прямого перебора позволяет снизить мощность пространства поиска с 2^{64} до 2^{53} . Одним из дальнейших путей ускорения атаки является увеличение эффективности алгоритма шифрования A5/1. В настоящей работе мы сравниваем реализацию генератора A5/1, основанную на полном предвычислении значений входящих в него регистров сдвига с линейной обратной связью с реализацией, основанной на технике "bitslice". Проведенное нами сравнение CPU и GPU версий данных алгоритмов показывает существенное преимущество GPU-bitslice версии.

Ключевые слова: криптоанализ, генератор A5/1, GSM, GPU, bitslice.

1. Введение

Генератор ключевого потока A5/1 является основным средством шифрования, используемым в стандарте мобильной связи GSM. Всемирное распространение сетей GSM делает задачу криптоанализа генератора A5/1 популярной среди исследователей сетевой безопасности. Дополнительным фактором привлекательности A5/1 для криптоаналитиков является его сравнительно короткая длина ключа (64 бита). За четверть века существования протокола GSM исследователями по всему миру были опубликованы десятки работ, описывающие различные атаки на A5/1. Для эффективного применения многих таких атак принципиально важна скорость программной реализации алгоритма генератора A5/1. В настоящей работе мы проводим сравнение двух эффективных реализаций этого генератора. Первая реализация основана на полном предвычислении последовательности состояний входящих в A5/1 сдвиговых регистров. Впервые данная идея была представлена в [1]. Вторая реализация, основанная на применении техники bitslice («разрядно-модульный подход») [2], использовалась для генерации rainbow-таблиц в проекте [3]. Мы используем эти реализации для построения атаки сокращенного перебора («атака Андерсона» [4]) на A5/1. Мы показываем, что данная атака может быть осуществлена за реальное время на кластерах, использующих современные GPU.

1.1 Генератор A5/1

Генератор A5/1 (рис. 1) состоит из трех регистров сдвига с линейной обратной связью (РСЛОС) (англ. LFSR от Linear Feedback Shift Register), длиной 19, 22 и 23 разряда соответственно (в сумме 64 разряда), заданных следующими полиномами обратной связи:

$$x^{19} + x^{18} + x^{17} + x^{14} + 1;$$
$$x^{22} + x^{21} + 1;$$
$$x^{23} + x^{22} + x^{21} + x^{8} + 1.$$

Важной особенностью A5/1 является схема тактирования РСЛОС, используемая в нем. В начале каждого такта значения самых старших бит регистров снимаются и складываются по модулю 2 (операция XOR). Полученное таким образом значение становится следующим битом ключевого потока.

*Работа была частично поддержана РФФИ (№ 14-07-00403-а, 15-07-07891-а и 16-07-00155-а).

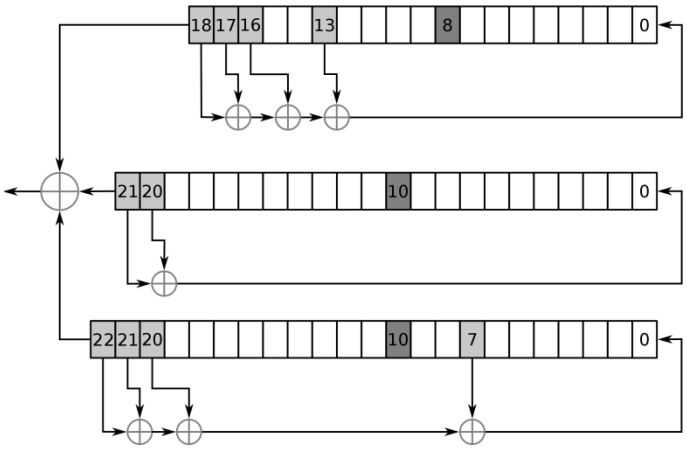

Рис. 1. Схема работы генератора A5/1.

РСЛОС в A5/1 сдвигаются несинхронно. В каждом такте t для регистра с номером $j \in \{1,2,3\}$ проверяется равенство

$$b_j^t = maj(b_1^t, b_2^t, b_3^t) \qquad (*)$$

где b_j^t – значение серединного разряда регистра с номером j на такте с номером t (серединными называются разряды РСЛОС 1-3 с номерами 9, 11, 11), а maj – функция большинства (majority function), то есть булева функция, задаваемая следующей формулой:

$$maj(A, B, C) = A \wedge B \vee A \wedge C \vee B \wedge C.$$

Если равенство (*) имеет место, то РСЛОС с номером j сдвигается в такте с номером t. В противном случае сдвиг не производится.

В нашей работе мы не рассматриваем процесс инициализации A5/1, предусмотренный спецификацией стандарта GSM. Хорошо известно [3], что слабости протокола GSM позволяют получить несколько первых берстов (англ. «burst», 1 берст равен 114 битам) ключевого потока. Зная этот фрагмент, достаточно найти лишь состояние регистров, которое его порождает, после чего мы можем дешифровать шифртекст любого объема. Из [5] известно, что для корректного восстановления неизвестного состояния регистров достаточно проанализировать 64 бита порожденного этим состоянием ключевого потока.

1.2 Обзор литературы и предшествующих работ

Первая атака, улучшающая оценку сложности по сравнению с полным перебором, была предложена в 1994 году в [4] (т. н. «атака Андерсона»). Она основана на угадывании заполнений 2-х регистров и вычислении значений 3-го. Данная атака получила развитие, например, в [6].

В дальнейшем были предложены атаки, использующие линеаризацию систем уравнений, описывающих генерацию ключевого потока [5], а также атаки, основанные на принципах пространственно-временного компромисса [1, 7]. В [8] была описана атака на A5/1, основанная на алгоритмах решения задачи о булевой выполнимости (SAT). Данная атака была реализована летом 2009 г. в распределенной вычислительной среде BNB-Grid. Следует отметить, что первые оценки времени, подтверждающие принципиальную возможность этой атаки, были получены годом ранее в [9]. Также в [8] было вычислительно подтверждено наличие т.н. «коллизий» для функции шифрования A5/1: то есть различных секретных ключей, порождающих один и тот же ключевой поток произвольной длины. Результаты работ [8, 9] были опубликованы в англоязычном издании только в 2011 г. [10]. В дальнейшем эта технология была существенно развита, и на ее основе был проведен криптоанализ A5/1 в проекте добровольных распределенных вычислений SAT@home [10].

Осенью 2009 года появились первые rainbow-таблицы для криптоанализа A5/1, построенные группой «A5/1 Cracking Project» [3]. На сегодняшний день данный метод криптоанализа A5/1 можно считать самым эффективным в плане временных затрат. Правда, строго говоря, этот метод не обеспечивает полностью достоверных результатов криптоанализа,

поскольку вероятность найти им секретный ключ, анализируя 8 берстов ключевого потока (912 бит), составляет приблизительно 88%.

1.3 «Атака Андерсона»

В данной работе мы реализовали вариант «атаки Андерсона», в применении к регистру R_2. Конкретно, по известному ключевому потоку, известному заполнению регистров R_1 и R_3 и известному заполнению ячеек регистра R_2, лежащих правее серединного бита, можно вычислить значения оставшихся ячеек регистра R_2.

Алгоритм «атаки Андерсона» можно описать следующим образом [4]:

1) угадать заполнение регистров R_1 и R_3 полностью, и заполнение регистра R_2 справа от серединного бита (рис. 1);
2) по известному фрагменту ключевого потока, тактируя генератор, вычислить заполнение регистра R_3 слева от серединного бита;
3) проверить полученное таким образом начальное состояние генератора, тактируя его и сравнивая с известным фрагментом ключевого потока.

По сути, этот алгоритм является модификацией метода прямого перебора, в котором за счет эксплуатации особенностей конструкции шифра удается значительно уменьшить пространство поиска. В данном конкретном случае пространство поиска сокращается с 2^{64} до 2^{53} ключей.

2. Быстрая программная реализация генератора A5/1

Эффективность метода прямого перебора напрямую зависит от 2-х факторов: масштабов пространства поиска и скорости проверки ключей-кандидатов. В случае генератора A5/1 малая длина ключа и применение «атаки Андерсона» позволяют добиться разумного масштаба пространства поиска (2^{53}). Скорость проверки можно увеличить, задействовав вместо «наивной» программной реализации генератора A5/1 альтернативные способы вычисления ключевого потока. Далее мы приводим описание двух наиболее эффективных из них: техники bitslice (наиболее близкий русскоязычный термин «разрядно-модульный подход») и метода предвычисления РСЛОС. Данные по сравнению производительности получившихся реализаций приведены в разделе 3 в таблице 1 («атака Андерсона») и таблице 2 (обыкновенный прямой перебор).

2.1 Техника «bitslice»

Современные универсальные вычислительные архитектуры далеко не оптимальны для реализации криптографических алгоритмов. Последние работают с битами, тогда как современные процессоры проектируются для работы со словами длиной 32-256 бит. В итоге «наивная» реализация криптографической функции стандартными методами программирования может не задействовать вычислительные возможности рассматриваемой платформы полностью. К примеру, на CPU при необходимости провести операцию «сложение по модулю 2» (XOR) над двумя булевыми аргументами, их значения будут записаны в младшие биты двух 32-х битных регистров общего назначения (РОН). Результат также будет записан в один из доступных 32-х битных регистров. Фактически, 31 из 32-х бит регистра будет простаивать в бездействии. Эту проблему можно решить с применением побитовых булевых операций, которые действуют над РОН поразрядно – как над булевыми векторами. Зачастую криптографический алгоритм может быть представлен в виде схемы из логических вентилей так, что появляется возможность эффективно обрабатывать сразу столько наборов данных, сколько позволяет разрядность РОН вычислительной платформы. В англоязычной литературе этот прием иногда называют «SIMD within a register», т. е. дословно «ОКМД[1] на одном регистре», что весьма точно передает его суть. Еще одно часто используемое название –

[1] Вычислительная архитектура типа «одна команда, много данных».

«bitslice», т.е. «разрядно-модульный подход» (по аналогии с «разрядно-модульными процессорами»). Опишем основную идею данного подхода.

Рассмотрим произвольную булеву функцию

$$f: \{0,1\}^n \to \{0,1\}.$$

Данную функцию всегда можно представить в виде суперпозиции булевых функций арности не более 2, образующих некоторый полный базис B. Простейший пример такого представления нам дают КНФ или ДНФ рассматриваемой функции (в этом случае $B = \{\wedge, \vee, \neg\}$). Значение произвольной функции из базиса B в вычислительном устройстве получается в результате применения соответствующей данной функции инструкции к одной или двум (в зависимости от арности функции) ячейкам памяти устройства, содержащим по одному биту входа. Современным вычислительным устройствам доступны т. н. «побитовые» инструкции, которые могут применяться к целому регистру или паре регистров (например, инструкция «побитовое сложение по модулю 2»). Результатом выполнения такой инструкции является одновременное независимое вычисление множества значений некоторой базисной функции на соответствующем наборе входов. Обычно размер регистра (число ячеек или разрядов) есть 2^d (для большинства современных архитектур $5 \le d \le 7$).

Пусть теперь рассматривается задача вычисления произвольной функции f (описанного выше вида) на всех 2^n возможных входах. Предположим, что f представлена в виде суперпозиции m базисных функций из B, и пусть размер регистра вычислительного устройства есть 2^d. Таким образом, число инструкций, необходимых для вычисления f на всех входах из $\{0,1\}^n$, есть $m \cdot 2^{n-d}$.

Фактически, разрядно-модульный подход может быть использован в любых мультиразрядных арифметико-логических устройствах (АЛУ), в которых доступны инструкции, реализующие побитовые операции над РОН.

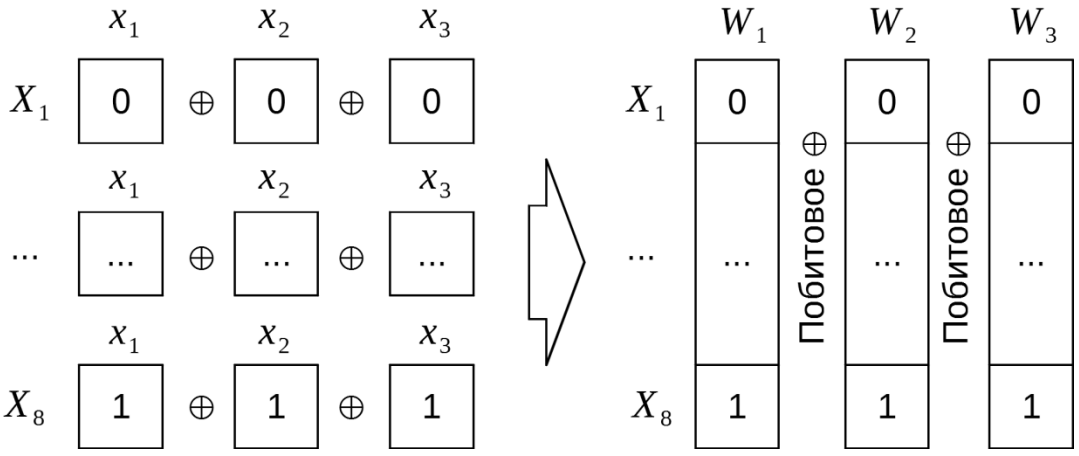

Рис. 2. Разрядно-модульный подход.

Пример 1. Пусть требуется вычислить значения функции

$$f_{\oplus}: \{0,1\}^3 \to \{0,1\},$$

заданной формулой $x_1 \oplus x_2 \oplus x_3$, на всех возможных входах (их 8). При последовательном вычислении для каждого слова вида $(x_1, x_2 x_3)$ используется $m = 2$ инструкций, поэтому общее число инструкций для данного способа вычислений равно $m \cdot 2^n = 16$. Теперь предположим, что рассматривается вычислительное устройство с регистрами, состоящими из 8 разрядов, и на этом устройстве доступна инструкция «побитовое $\oplus$» для пары регистров. Тогда мы можем рассмотреть массив, в котором записаны все возможные слова из $\{0,1\}^3$, т.е. 8 слов $(\mathrm{X}_1, \ldots, \mathrm{X}_8)$, состоящих из 3 бит, как три слова $\mathrm{W}_1, \mathrm{W}_2, \mathrm{W}_3$, каждое из которых состоит из 8 бит (см. Рис. 2). Складывая слова W_1 и W_2 при помощи одной инструкции «побитовое $\oplus$» для 8 битных регистров, мы получаем вектор значений суммы первых двух битов для всех слов из $\{0,1\}^3$. Вычисляя аналогичным образом значение $(\mathrm{W}_1 \oplus \mathrm{W}_2) \oplus \mathrm{W}_3$, имеем значения функции $f_{\oplus}$ для всех слов из $\{0,1\}^3$, записанные в 8-битном результирующем регистре. При таком способе

значения функции $f_{\oplus}$ на всех входах из $\{0,1\}^3$ вычисляются за две инструкции «побитовое $\oplus$» для 8-битных регистров.

Приведем далее описание реализации генератора A5/1 в этой технике. Сопоставим каждому из $n \in \{1,\ldots,64\}$ разрядов РСЛОС R_1, R_2, R_3 слово $W_n \in \{0,1\}^k$, где k – разрядность вычислительной платформы:

$$R_1: W_1, \ldots, W_{19}$$
$$R_2: W_{20}, \ldots, W_{42}$$
$$R_3: W_{43}, \ldots, W_{64}$$

Сдвигу регистра R_1 при его реализации в bitslice-технике будут соответствовать следующие операции над словами длины k:

$$W_1' = W_{19} \oplus W_{18} \oplus W_{17} \oplus W_{14},$$
$$W_n' = W_{n-1}, n \in \{2,\ldots,19\},$$

где $\oplus$ – операция побитового сложения векторов длины K по модулю 2. Вычисление очередного бита ключевого потока будет выглядеть так:

$$W_{out} = W_{19} \oplus W_{20} \oplus W_{43}.$$

Несколько сложнее реализовать в bitslice-схеме условное тактирование. Для выяснения того, требуется ли тактирование регистра, необходимо вычислить флаги тактирования $F_{R_1}, F_{R_2}, F_{R_3}$ с использованием функции большинства:

$$W_{maj} = maj(W_9, W_{33}, W_{54}) = (W_9 \wedge W_{33}) \vee (W_9 \wedge W_{54}) \vee (W_{33} \wedge W_{54});$$
$$F_{R_1} = W_9 \oplus \overline{W_{maj}};$$
$$F_{R_2} = W_{33} \oplus \overline{W_{maj}};$$
$$F_{R_3} = W_{54} \oplus \overline{W_{maj}}.$$

(все операции побитовые над булевыми векторами соответствующей длины).

Поскольку сдвиг РСЛОС зависит от их текущего состояния, возникает необходимость реализовать независимый условный сдвиг отдельных «потоков», содержащихся в разрядах переменных W_n. Для этого можно воспользоваться побитовым вариантом функции арности 3 «выбор бита» («bitselect»):

$$x, y, z \in \{0,1\};$$
$$BS(x,y,z) = \begin{cases} y, x = 1 \\ z, x = 0 \end{cases}.$$

Если аппаратная реализация этой функции недоступна на вычислительном устройстве, ее можно реализовать через стандартные побитовые функции ($\wedge, \vee, \neg$).:

$$BS(x,y,z) = (x \wedge y) \vee (\neg x \wedge z)$$

Тактирование регистра R_1 с применением функции $BS(x,y,z)$ и флагов тактирования $F_{R_1}, F_{R_2}, F_{R_3}$ выглядит следующим образом:

$$W_1' = BS\big(F_{R_1}, (W_{19} \oplus W_{18} \oplus W_{17} \oplus W_{14}), W_1\big);$$
$$W_n' = BS\big(F_{R_1}, W_{n-1}, W_n\big), n \in \{2, \ldots, 19\}.$$

2.2 Предвычисление последовательностей РСЛОС

Используемая в данном пункте схема представления состояний регистров A5/1 в целом заимствована из [1]. Полиномы, описывающие РСЛОС генератора A5/1, являются примитивными и, как следствие, порождают двоичные рекуррентные последовательности с длинами 2^{19}-1, 2^{22}-1, 2^{23}-1 бит для 1, 2 и 3 регистров соответственно. Вычисленные полностью и записанные в виде циклических битовых массивов A_1, A_2, A_3, они в сумме занимают около 1,5 Мбайт памяти вычислительного устройства. Вместо того чтобы каждый раз при тактировании вычислять новое состояние регистров достаточно увеличивать на 1 (или 0) индексы смещения i, j, k в этих массивах (для регистров R_1, R_2, R_3 соответственно, см. рис. 3). Пусть S_t – бит ключевого потока с номером t. Тогда вычисление S_{t+1} будет выглядеть следующим образом:

$$S_t = A_1(i_t) \oplus A_2(j_t) \oplus A_3(k_t);$$
$$m_t^1 = A_1(i_t - 11);$$
$$m_t^2 = A_2(j_t - 12);$$
$$m_t^3 = A_3(k_t - 13);$$
$$M_t = maj(m_t^1, m_t^2, m_t^3);$$
$$i_{t+1} = i_t + m_t^1 \oplus \neg M_t;$$
$$j_{t+1} = j_t + m_t^2 \oplus \neg M_t;$$
$$k_{t+1} = k_t + m_t^3 \oplus \neg M_t;$$
$$S_{t+1} = A_1(i_{t+1}) \oplus A_2(j_{t+1}) \oplus A_3(k_{t+1}).$$

Здесь m_t^1, m_t^2, m_t^3, M_t – значения серединных бит регистров R_1, R_2, R_3 и функции большинства от них на шаге с номером t.

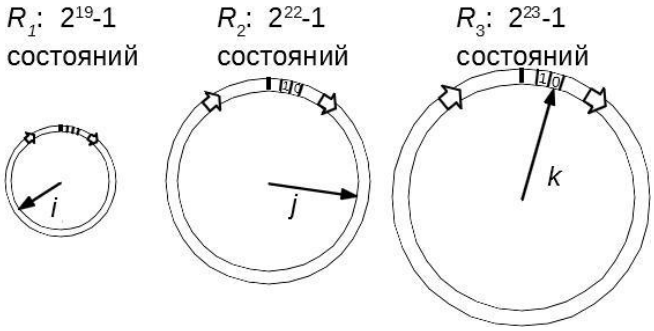

Рис. 3. Представление состояний РСЛОС A5/1 в виде циклических массивов.

Для того чтобы реализовать полный перебор всех возможных начальных состояний генератора A5/1 необходимо проверить все возможные сочетания троек i, j, k:

$$0 \le i \le (2^{19} - 1),$$
$$0 \le j \le (2^{22} - 1),$$
$$0 \le k \le (2^{23} - 1).$$

Как и bitslice, такой подход устраняет условные переходы в программном коде генератора, что повышает эффективность его выполнения на современных CPU и GPU.

Применение «атаки Андерсона» с такой реализацией генератора требует введения дополнительного массива I_2, который используется для того, чтобы по известному заполнению регистра R_2 найти номер k данного заполнения в циклическом массиве A_2.

3. Результаты экспериментов

Мы реализовали «атаку Андерсона» на GPU и CPU с применением описанных выше техник bitslice и предвычисления РСЛОС. Результаты приведены в таблице 1. В таблице 2 для сравнения приведены данные по реализации обыкновенного метода прямого перебора, реализованного с использованием тех же техник.

В экспериментах на CPU использовался процессор Intel Core i7 930 2,8 ГГц. Приложение запускалось в 1 поток, для bitslice использовались 32-разрядные типы данных. Оптимизация приложения на уровне ассемблера и специальных SIMD-инструкций процессора (SSE2, AVX и т.д.) не производилась. Компилятор g++ 4.9.2 вызывался с флагом оптимизации -O5. Эксперименты с GPU проводились на NVIDIA GeForce GTX 750 Ti с компилятором nvcc 6.5.

Таблица 1. Скорость «атаки Андерсона» (пространство перебора 2^{53}) в различных реализациях генератора A5/1 на GPU и CPU, в миллионах ключей в секунду.

Платформа	bitslice	Предвычисление РСЛОС
CPU	37,06	6,83
GPU	9544,3	490,8

Таблица 2. Скорость прямого перебора (пространство перебора 2^{64}) в различных реализациях генератора A5/1 на GPU и CPU, в миллионах ключей в секунду.

Платформа	bitslice	Предвычисление РСЛОС
CPU	135,5	35,1
GPU	20211	1470

4. Заключение

Приведенные в таблице 1 результаты показывают, что «атака Андерсона» на одной далеко не самой производительной видеокарте имеет вполне реалистичное время (порядка 270 часов). Реализация данной атаки на современных кластерах из GPU позволит осуществлять криптоанализ A5/1 за минуты. Преимущество данной атаки перед Rainbow-методом [3] состоит в гарантированном восстановлении секретного ключа. Преимущество перед методом прямого перебора – в значительном меньшем пространстве поиска.

Литература

1. Biryukov A., Shamir A., Wagner D. Real Time Cryptanalysis of A5/1 on a PC // Fast Software Encryption. 2000. LNCS. Vol. 1978. Springer Berlin Heidelberg. pp. 1–18.

2. Biham E. A fast new DES implementation in software // Fast Software Encryption. 1997. LNCS. Vol. 1267. Springer Berlin Heidelberg. pp. 260-272.

3. Nohl K. Attacking phone privacy // Black Hat USA. 2010. URL: https://srlabs.de/blog/wp-content/uploads/2010/07/Attacking.Phone_.Privacy_Karsten.Nohl_1.pdf (дата обращения 27.11.2015).

4. Anderson R., Roe M. A5 The GSM Encryption Algorithm // sci. crypt. 1994. URL: http://jya.com/crack-a5.htm (дата обращения 27.11.2015).

5. Golic J. D. Cryptanalysis of alleged A5 stream cipher // Advances in Cryptology—EUROCRYPT'97 LNCS. Vol. 1233. Springer Berlin Heidelberg. pp. 239–255.

6. Shah J., Mahalanobis A. A new guess-and-determine attack on the A5/1 stream cipher // arXiv preprint 2012 arXiv:1204.4535.

7. Barkan E., Biham E., Keller N. Instant ciphertext-only cryptanalysis of GSM encrypted communication // Journal of Cryptology. 2008. Vol. 21, No. 3. Springer Berlin Heidelberg. pp. 392-429.

8. Посыпкин М.А., Заикин О.С., Беспалов Д.В., Семенов А.А. Решение задач криптоанализа поточных шифров в распределенных вычислительных средах // Труды ИСА РАН. 2009. №46. С. 119-137.

9. Семенов А.А., Заикин О.С., Беспалов Д.В., Буров П.С., Хмельнов А.Е. Решение задач обращения дискретных функций на многопроцессорных вычислительных системах // Тр. IV Междунар. конф. «Параллельные вычисления и задачи управления» (PACO'2008). М., 2008. С. 152–176.

10. Semenov A., Zaikin O., Bespalov D., Posypkin M. Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system // Parallel Computing Technologies. 2011. LNCS Vol. 6873. Springer Berlin Heidelberg. pp. 473-483.

11. Semenov A., Zaikin O. Using Monte Carlo method for searching partitionings of hard variants of Boolean satisfiability problem // Parallel Computing Technologies. 2015. LNCS. Vol. 9251 Springer International Publishing. pp. 222-230.

Inverting A5/1 cryptographic function on a GPU with alternative A5/1 algorithm software implementations*

V.G.Bulavintsev, A.A.Semenov

ISDCT SB RAS

In this paper we study possibilities for speeding up A5/1 cryptographic function inversion procedure with the help of a GPU. By employing "the Anderson's attack" for the A5/1 cryptographic function we reduce search space power from 2^{64} to 2^{53}. To enhance this attack's speed one needs an effective software implementation of the A5/1 generator. We describe two such implementations. The first one is based on a precomputation of A5/1's shift registers output sequences. The second one uses "bitslice" technique. We compare CPU and GPU performance of these implementations. Experimental data shows that GPU version of bitslice implementation is the fastest. We conclude that a modern GPU-based computing cluster is able to invert the A5/1 cryptographic function in several minutes.

Keywords: cryptanalysis, A5/1 generator , GSM, GPU, bitslice.

References

1. Biryukov A., Shamir A., Wagner D. Real Time Cryptanalysis of A5/1 on a PC // Fast Software Encryption. 2000. LNCS. Vol. 1978. Springer Berlin Heidelberg. pp. 1–18.

2. Biham E. A fast new DES implementation in software // Fast Software Encryption. 1997. LNCS. Vol. 1267. Springer Berlin Heidelberg. pp. 260-272.

3. Nohl K. Attacking phone privacy // Black Hat USA. 2010. URL: https://srlabs.de/blog/wp-content/uploads/2010/07/Attacking.Phone_.Privacy_Karsten.Nohl_1.pdf (дата обращения 27.11.2015).

4. Anderson R., Roe M. A5 The GSM Encryption Algorithm // sci. crypt. 1994. URL: http://jya.com/crack-a5.htm (дата обращения 27.11.2015).

5. Golic J. D. Cryptanalysis of alleged A5 stream cipher // Advances in Cryptology—EUROCRYPT'97 LNCS. Vol. 1233. Springer Berlin Heidelberg. pp. 239–255.

6. Shah J., Mahalanobis A. A new guess-and-determine attack on the A5/1 stream cipher // arXiv preprint 2012. arXiv:1204.4535.

7. Barkan E., Biham E., Keller N. Instant ciphertext-only cryptanalysis of GSM encrypted communication // Journal of Cryptology. 2008. Vol. 21, No. 3. Springer Berlin Heidelberg. pp. 392-429.

8. Posypkin M., Zaikin O., Bespalov D., Semenov A. Solving the Cryptanalysis Problems of Stream Ciphers in Distributed Computing Environments // Trudy Instituta Systemnogo Analiza 2009. No. 46. pp. 119-137.

9. Semenov A., Zaikin O., Bespalov D., Burov P., Hmelnov A. Solving of Discrete Functions Inversion Problems on Multiprocessor Computer Systems // Parallel Computations and Control Problems, Proceedings of the 4th International Conference PACO'2008. pp. 152-176.

10. Semenov A., Zaikin O., Bespalov D., Posypkin M. Parallel logical cryptanalysis of the generator A5/1 in BNB-Grid system // Parallel Computing Technologies. 2011. LNCS Vol. 6873. Springer Berlin Heidelberg. pp. 473-483.

*This work was partially funded by RFBR (grants no.14-07-00403-a, 15-07-07891-a and 16-07-00155-a).

11. Semenov A., Zaikin O. Using Monte Carlo method for searching partitionings of hard variants of Boolean satisfiability problem // Parallel Computing Technologies. 2015. LNCS. Vol. 9251 Springer International Publishing. pp. 222-230.