Разработка функции генерации ключа на основе пароля в качестве приложения генератора Blum-Blum-Shub

Ю.Д. Выборнова

Самарский национальный исследовательский университет имени академика С.П. Королева, 443086, Московское шоссе, 34, Самара, Россия

Аннотация

Цель работы — анализ практической применимости криптографически стойкого программного генератора псевдослучайных чисел Blum-Blum-Shub для решения задач аутентификации и шифрования. В статье показано, что рассматриваемый генератор псевдослучайных последовательностей, обладающий высокой вычислительной сложностью, может эффективно применяться в тех криптографических задачах, в которых требуется низкая скорость генерации ключа. Предложен новый способ реализации алгоритма Password-Based Key Derivation Function, основанный на применении генератора Blum-Blum-Shub в качестве псевдослучайной функции. Предложенный алгоритм позволяет замедлить атаки со словарем и атаки перебором. По результатам экспериментального исследования доказано, что разработанный алгоритм позволяет адаптивно подстраивать гарантированное минимальное время выработки криптографического ключа под конкретные задачи аутентификации и шифрования.

Ключевые слова: генерация ключа на основе пароля; Password-based key derivation function; PBKDF; криптографический ключ; псевдослучайные последовательности; генератор псевдослучайных последовательностей; криптографически стойкий генератор; генератор Blum-Blum-Shub; защита данных; информационная безопасность

1. Введение

В криптографии генератор псевдослучайных последовательностей — такой алгоритм, который быстро порождает из небольших значений инициализации длинные последовательности битов. Генератор должен создавать такое впечатление, словно его результат определяется последовательным подбрасыванием абсолютно ровной монеты. Конечно, идея быстрого детерминированного механизма, выдающего такое недетерминированное поведение кажется противоречивой: наблюдая за выходом генератора в течение длительного времени, мы могли бы, в принципе, постепенно выявить детерминизм и воспроизвести такой генератор заново [1].

Обычно для решения этой проблемы для таких генераторов выдвигают только одно требование: чтобы сгенерированные последовательности проходили определенные стандартные статистические тесты (например, чтобы в достаточно длинной выходной последовательности частота появляющихся нулей и единиц была одинаковой, и чтобы нули и единицы были тщательно «перемешаны»).

Однако, для задач аутентификации и шифрования прохождения стандартных статистических тестов недостаточно. Псевдослучайные последовательности должны быть непредсказуемы для реальных компьютеров. Имеется в виду, что генератор псевдослучайных последовательностей имеет полиномиальную оценку непредсказуемости (как вправо, так и влево), тогда и только тогда, когда для каждого конечного начального фрагмента последовательности, который был создан этим генератором, при удалении любого элемента (крайнего правого, либо крайнего левого) вероятностная машина Тьюринга не сможет предсказать за полиномиальное время значение пропавшего элемента с вероятностью большей, чем $0,5$. Предсказуемость генератора означает, что при заданном небольшом фрагменте последовательности можно быстро сделать предположение о значениях инициализации и рационально продолжить данный фрагмент в обе стороны.

Свойство непредсказуемости вправо и влево позволяет защититься от статистических атак, которые делятся на два подкласса:

1) метод криптоанализа статистических свойств шифрующей гаммы, который направлен на изучение выходной последовательности криптосистемы: криптоаналитик пытается установить значение следующего бита последовательности с вероятностью выше вероятности случайного выбора с помощью различных статистических тестов [2];

2) метод криптоанализа сложности последовательности: криптоаналитик пытается найти способ генерировать последовательность, аналогичную гамме, но более просто реализуемым способом.

Однако для современных генераторов псевдослучайных последовательностей еще одним важным требованием является высокая скорость генерации битов. Генератор, исследуемый в данной работе, не обладает высокой скоростью по причине высокой вычислительной сложности, но в ряде задач данный недостаток может стать преимуществом.

2. Генератор Blum-Blum-Shub

Генератор $x^2 \bmod N$ из небольших значений инициализации быстро генерирует длинные хорошо распределенные последовательности. В своей основе он имеет серьезную задачу — задачу о квадратичных вычетах.

Пусть $\mathbf{N} = \{$ *целые* $N \mid N = P \cdot Q$, *такие что* P,Q — *различные простые числа равной длины* $(|P| = |Q|)$, $P \equiv 3 \pmod 4$ *и* $Q \equiv 3 \pmod 4 \}$ — множество значений параметра.

Пусть $Z_N^* = \{$ *целые* $x \mid 0 < x < N$ *и НОД* $(x, N) = 1\}$. Для $N \in \mathbf{N}$ положим, что $X_N = \{x^2 \bmod N \mid x \in Z_N^*\}$ — множество квадратичных вычетов по модулю N. Пусть дизъюнктное объединение $X = \mathop{\mathbf{T}}\limits_{N \in \mathbf{N}} X_N$ — область определения значений инициализации.

Для $(N, x) \in X^n$ пусть $\mu_n(N, x) = u_n(N) \cdot v_n(X)$, где u_n — равномерное распределение вероятности $\{N \in \mathbf{N} \mid |N| = n\}$, а v_n — равномерное распределение X_n. Тогда $\{\mu_n\}$ — достижимое распределение вероятности X, так как:

1) асимптотически, $1/(k \ln 2)$ всех k-битных чисел — простые числа, и половина простых чисел любой длины сравнимы с 3 по модулю 4 (по теореме Валле Пуссена о простых числах);

2) задача проверки простоты разрешима за полиномиальное время с помощью вероятностных алгоритмов (Монте-Карло), например, тест Соловея-Штрассена, тест Рабина или, исходя из расширенной гипотезы Римана, с помощью детерминированного полиномиального теста Миллера;

3) наибольшие общие делители вычислимы за полиномиальное время и

$$|Z_N^*| / |Z_N| \to 1 \text{ при } n \to \infty.$$

Пусть преобразование $T : X \to X$ определяется как $T(x) = x^2 \bmod N$ при $x \in X_N$. T — перестановка X_N и вычисляется за полиномиальное время. Положим, $B : X \to \{0,1\}$ определяется как $B(x) =$ *бит четности* (x). B вычисляется за полиномиальное время. Тогда (X, T, B) определяет генератор псевдослучайных последовательностей (по основанию 2), называемый генератором $x^2 \bmod N$. Таким образом, при подаче на вход (N, X_0) на выходе генератора $x^2 \bmod N$ образуется псевдослучайная последовательность бит b_0, $b_1, \ldots$, полученная при установлении $x_{i+1} = x^2 \bmod N$ и извлечении бита b_i, равного *биту четности* (x_i).

Такие последовательности периодичны с периодом, обычно равным $\lambda(\lambda(N))$, где $\lambda(n)$ — функция Кармайкла. Дополнительное привлекательное свойство этого генератора заключается в том, что он обеспечивает прямое определение тех отдельных битов, которые он вырабатывает, для всех, кто знает разложение N на множители [3]. Равенство

$$x_i = x_0^{2i} \bmod N = x_0^{2i \bmod \lambda(N)} \bmod N$$

позволяет рационально вычислить i-тый элемент последовательности, если даны x_0, N и $\lambda(N)$, при $i > 0$. При $i < 0$ необходимо использовать формулу

$$x_i = x_{i \bmod \lambda(\lambda(N))}.$$

Пример. Пусть $N = 7 \cdot 19 = 133$ и $x_0 = 4$. Тогда последовательность x_0, $x_1 = x_0^2 \bmod 133, \ldots$ имеет период 6, где $x_0, x_1, \ldots, x_5, \ldots = 4, 16, 123, 100, 25, 93, \ldots$ Таким образом, $b_0, b_1, \ldots, b_5, \ldots = 0\ 0\ 1\ 0\ 1\ 1 \ldots$ — псевдослучайная последовательность, порожденная генератором $x^2 \bmod N$, на входе которого $(133,4)$. Здесь $\lambda(N) = 18$ и $\lambda(\lambda(N)) = 6$.

Основные результаты по поводу непредсказуемости и криптографической стойкости следуют из предположений касательно неразрешимости некоторых задач теории чисел посредством вероятностных полиномиальных алгоритмов.

В основе генератора $x^2 \bmod N$ лежит предположение о квадратичных вычетах. Пусть N — произведение двух различных нечетных простых чисел. Ровно половина элементов из Z_N^* имеют символ Якоби, равный $+1$, остальная половина имеет символ Якоби -1. Обозначим их как $Z_N^*(+1)$ и $Z_N^*(-1)$ соответственно. Ни один элемент из $Z_N^*(-1)$ не является квадратичным вычетом. Ровно половина элементов из $Z_N^*(+1)$ — квадратичные вычеты. Задача о квадратичных вычетах с параметрами N и x состоит в том, чтобы решить, является ли x из $Z_N^*(+1)$ квадратичным вычетом или нет. Предполагается, что любой эффективный алгоритм для определения, является ли входной элемент квадратичным вычетом, будет неверным по входным данным.

Самым быстрым алгоритмом факторизации больших целых чисел на сегодняшний день является общий метод решета числового поля [4]. В предположении о том, что факторизация N трудна, генератор $x^2 \bmod N$ имеет полиномиальную оценку непредсказуемости влево.

Из теоремы Яо [5] следует, что последовательности, порожденные генератором $x^2 \bmod N$, основанном на задаче о квадратичных вычетах по модулю, проходят все вероятностные полиномиальные статистические тесты (то есть ни один статистический тест не сможет отличить такие последовательности от последовательностей, порожденных в результате последовательного «подбрасывания монеты» за время $poly(n)$ (с преимуществом больше бесконечно малого)).

В 1986 году Ленор Блюм, Мануель Блюм и Майкл Шуб привели математическое обоснование криптографической стойкости генератора $x^2 \bmod N$ и опубликовали алгоритм, который впоследствии получил название генератора Blum-Blum-Shub.

Данный алгоритм [6] генерирует псевдослучайную последовательность бит $z_1, z_2, \ldots, z_l$ длины l:

1) Возьмем два достаточно больших секретных случайных (и различных) простых числа P и Q, каждое из которых сравнимо с 3 по модулю 4.

2) Вычислим $N = P \cdot Q$.

3) Выберем случайное целое число s (значение инициализации) из отрезка $[1, N-1]$, такое что $НОД(s, N) = 1$.

4) Вычислим $x_0 \leftarrow s^2 \bmod N$.

5) Для i от 1 до l выполняем следующее:

   i. $x_i \leftarrow x_{i-1}^2 \bmod N$;

   ii. $z_i \leftarrow$ *наименее значимый бит* x_i.

6) Формируем выходную последовательность $z_1, z_2, \ldots, z_l$.

Приведем пример выполнения данного алгоритма.

1) Выберем два простых числа $P = 47$ и $Q = 67$. Оба сравнимы с 3 по модулю 4 в том смысле, что $47 \bmod 4 = 3$ и $67 \bmod 4 = 3$.

2) Теперь $N = 47 \cdot 67 = 3149$.

3) Выберем начальное значение $s = 7$, $НОД(7, 3149) = 1$.

4) Теперь $x_0 = 7^2 \bmod 3149 = 49$.

5) Запускаем цикл от 1 до 16 и оставшиеся числа вычисляем согласно алгоритму.

6) Получаем последовательность длиной 16 бит: 1110101011010010.

В статье [7] приведено исследование средней скорости различных реализаций генератора Blum-Blum-Shub и сделан вывод о том, что данный генератор имеет ограниченное применение по причине своей низкой скорости. Средняя скорость работы генератора Blum-Blum-Shub во много раз ниже средней скорости современных симметричных алгоритмов блочного шифрования. Так, например, средняя скорость алгоритма AES равна 2,5 ГБ/с, алгоритма Twofish — 415 МБ/с, алгоритма Serpent — 255 МБ/с, а средняя скорость одной из самых быстрых реализаций генератора Blum-Blum-Shub — 23 КБ/с. Из этого следует, что исследуемый генератор неприменим для генерации гаммы в шифрах гаммирования, но может использоваться для генерации данных в системах и протоколах, в которых требуются непредсказуемость и криптографическая стойкость, а скорость генерации не играет роли. А именно: для генерации ключей в асимметричных системах, таких как RSA, в качестве генератора имитовставки, генератора ключей для имитовставки, а также в некоторых реализациях криптографических протоколов аутентификации и шифрования. Однако, стоит заметить, что низкая скорость не всегда является недостатком. В криптографии существуют задачи, в которых специально используются медленные алгоритмы. Одной из таких задач является генерация ключа на основе пароля.

3. Функция генерации ключа на основе пароля

3.1. Password-Based Key Derivation Function

Пароль или парольная фраза — это строка символов, которая используется для аутентификации пользователя и предоставления ему доступа к системе. Чаще всего пароли выбираются пользователями. Поскольку большинство самостоятельно выбранных пользователями паролей обладают низкой энтропией и недостаточной случайностью, такие пароли не рекомендуется использовать непосредственно в качестве криптографических ключей. Однако, во многих системах при защите данных в устройствах хранения пароль может быть единственным способом защиты. Следовательно, во избежание атак на такие системы необходимо применение дополнительных мер по их защите.

Функция генерации ключа на основе пароля (Password-Based Key Derivation Function, PBKDF) — это детерминированный алгоритм, который применяется для получения криптографической ключевой информации из какого-либо секретного значения [8]. Таким образом, такая функция позволяет из легко запоминающегося пароля пользователя получить криптографически стойкий ключ заданной длины, называемый мастер-ключом.

Мастер-ключ используется для генерации ключей защиты данных. Ключ защиты данных (Data Protection Key, DPK) — ключ или набор ключей, используемый для защиты или восстановления данных, проверки подлинности или целостности защищаемых данных или для защиты закрытого ключа, используемого для формирования цифровых подписей. Существует два способа получить ключ защиты данных из мастер-ключа: непосредственное использование мастер-ключа (или фрагмента мастер-ключа) в качестве ключа защиты данных и генерация ключа защиты данных из мастер-ключа с использованием функции генерации ключа. Функция генерации ключа (Key Derivation Function, KDF) — функция, которая генерирует ключевую информацию (т.е. бинарную строку, у которой любые непересекающиеся фрагменты требуемой длины могут быть использованы в качестве симметричных криптографических ключей). Кроме того, мастер-ключ может применяться для генерации промежуточных ключей, обеспечивающих защиту одного или нескольких существующих ключей защиты данных (которые, в свою очередь, могут быть сгенерированы из мастер-ключа с использованием функции KDF).

В основе алгоритма PBKDF лежит псевдослучайная функция. Обозначим ее PRF. На вход алгоритма подается фиксированное число C, обозначающее, сколько раз следует запустить псевдослучайную функцию. Назовем это число счетчиком итераций. Также входными данными алгоритма являются: пароль P, соль S — несекретная строка символов, уникальная для каждого пароля, и желаемая длина мастер-ключа MK_Length. Рекомендуемая длина мастер-ключа: не менее 112 бит. Рекомендуемая длина соли: не менее 128 бит. Обозначим мастер-ключ MK и обобщим вышесказанное формулой:

$$MK = PBKDF_{(PRF,C)}(P, S, MK\_Length).$$

Общая схема алгоритма PBKDF представлена на рисунке 1.

Как уже говорилось выше, пароль или парольная фраза — это секретная строка символов, которая используется для доступа к защищенному ресурсу. Когда пароли используются для генерации криптографических ключей, предполагается, что злоумышленник имеет возможность провести поисковую оффлайн атаку на процесс генерации ключа, выбрав систему, которая может быть гораздо мощнее той, которую использует легальный пользователь. Более

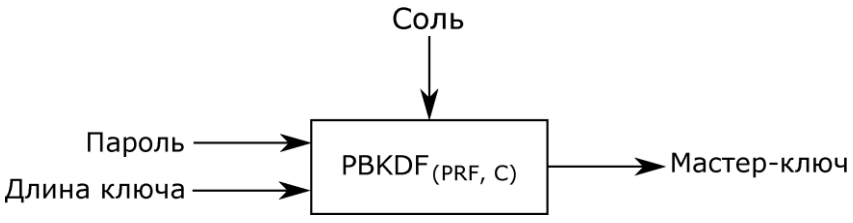

Рис. 1. Общая схема алгоритма генерации ключа на основе пароля.

того существует возможность распараллеливания таких атак. Следовательно, энтропия пароля (среднее число попыток, необходимых злоумышленнику для определения пароля) является критическим показателем.

Случайно сгенерированные пароли, как правило, имеют более высокую энтропию, чем выбранные пользователем пароли той же длины. Однако из-за сложности запоминания случайно сгенерированных паролей во многих приложениях и сервисах пароли выбираются пользователями самостоятельно. Для безопасности данных, хранящихся в электронном виде, пароли должны быть подобраны таким образом, чтобы злоумышленник не смог произвести атаку перебором. Стойкость пароля зависит от его длины и от того, насколько он случайный. Пароли короче 10 символов в большинстве случаев считаются слабыми. Существует много других свойств, по которым можно судить о слабости пароля. Например, не рекомендуется использовать последовательности чисел или последовательности букв в качестве

пароля. Легко доступная личная информация, такая как имя пользователя, телефон или дата рождения, не должны использоваться непосредственно в качестве пароля.

Парольные фразы часто состоят исключительно из букв, но они обладают хорошей энтропией за счет своей длины, которая обычно составляет от 20 до 30 символов.

Атаки со словарем нацелены на получение доступа путем перебора наиболее популярных паролей, которые хранятся в специально созданных словарях. Атаки такого типа имеют меньше шансов на успешное осуществление при использовании паролей, которые содержат случайные комбинации верхнего и нижнего регистра букв, а также числовые значения. Такие пароли могут быть подобраны только с помощью полного перебора всех возможных паролей. Данный метод считается самым неэффективным.

Основная идея PBKDF заключается в том, чтобы замедлить атаки со словарем и атаки перебором путем повышения времени, затрачиваемого на проверку каждого пароля. Злоумышленник со списком вероятных паролей может оценить PBKDF, если известны счетчик итераций и соль. Поскольку злоумышленнику придется потратить на каждую попытку значительное количество вычислительного времени, реализация атак со словарем и атак перебором будет сильно затруднена. Кроме того, использование PBKDF с применением соли делает неосуществимой атаку по предвычисленным (радужным) таблицам.

Применение уникальной соли в данном алгоритме является необходимым, несмотря на то что ее значения не хранятся в секрете. Допустим, ключи были сгенерированы только из паролей пользователей без добавления уникальной случайной соли. Допустим также, что злоумышленник получил доступ к алгоритму генерации ключа (оценил его) и сформировал таблицу результирующих ключей (радужную таблицу), подавая на вход наиболее вероятные пароли. Тогда, получив доступ к базе результирующих ключей, он сможет легко вычислить пароли пользователей по радужной таблице.

Благодаря применению уникальной соли для каждого пользователя, даже если несколько пользователей при регистрации ввели одинаковый пароль, значение ключа в базе будет разным, и злоумышленник не сможет догадаться, у каких пользователей пароли совпадают.

Рассмотрим еще один случай: злоумышленник не имеет доступа к базе результирующих ключей и пытается получить пароли пользователей путем полного перебора. Конечно, если злоумышленник получит доступ к значениям соли, то, скорее всего, ему удастся путем перебора подобрать пароль конкретного пользователя. Однако, вычислить пароли всех пользователей будет намного сложнее, поскольку для каждого пользователя соль уникальна, и, соответственно, перебор осуществлять придется для каждого пользователя. Таким образом, соль не защищает отдельный пароль от атаки перебором, а используется для борьбы с восстановлением всех паролей пользователей за один проход полного перебора.

Таким образом, основное предназначение соли: сделать возможной генерацию большого набора ключей для каждого пароля при фиксированном значении счетчика итераций. Для каждого пароля число возможных ключей составит, в среднем, 2^{Salt_Length}, где $Salt\_Length$ — длина соли в битах. Следовательно, в случае атаки со словарем применение соли значительно затруднит попытки злоумышленника сгенерировать такой словарь даже для небольшого набора наиболее вероятных паролей, а в случае атаки по предвычисленным таблицам применение соли сделает невозможной генерацию таблицы результирующих ключей.

Основное предназначение счетчика итераций: увеличить число вычислений, необходимых для генерации ключа на основе пароля, таким образом значительно увеличивая сложность осуществления атак. Так, например, если в PBKDF выполнится C итераций для получения ключа, то вычислительная сложность атаки со словарем на пароль с энтропией t бит возрастет с 2^t операций до $C \cdot 2^t$ операций.

Однако, стоит заметить, что количество вычислений, необходимых для формирования ключа, с увеличением числа итераций возрастет и для легального пользователя. Пользователь может заметить увеличение времени, требуемого, например, для аутентификации или доступа к защищенному хранилищу данных. Таким образом, при увеличении счетчика итераций уменьшится вероятность успешной атаки, но вместе с тем уменьшится производительность легального пользователя. Поэтому число итераций должно быть достаточно большим, но при этом не сильно влиять на производительность легальной системы.

Приемлемая производительность зависит от задачи, в которой будет применяться PBKDF. Так, например, при запуске системы, приемлемое время генерации ключа составит около секунды. Если же рассматриваемая функция запускается каждый раз при доступе к файлу, то время ее выполнения не должно превышать четверти секунды, поскольку пользователь может обращаться к файлам несколько раз в час. Если же речь идет о доступе к серверу, то приемлемое время работы PBKDF может достигать нескольких секунд.

Наиболее очевидным вариантом применения предложенного алгоритма PBKDF является генерация ключей шифрования.

С помощью ключа защиты данных (сгенерированного из мастер-ключа) можно, например, зашифровать раздел жесткого диска или съемного носителя информации. Наиболее известными программами для создания зашифрованных логических дисков являются TrueCrypt и BitLocker.

Для того чтобы зашифровать данные, пользователю сначала необходимо ввести пароль необходимой длины. Затем с помощью алгоритма PBKDF генерируется ключ шифрования. После этого осуществляется непосредственно шифрование. В случае, когда за одним компьютером работает несколько пользователей, каждый из них создает виртуальный диск или отводит себе раздел на жестком диске и шифрует его.

Для того чтобы получить доступ к зашифрованным данным, пользователь вводит пароль, указанный при шифровании. После этого с помощью алгоритма PBKDF генерируется ключ дешифрования, который совпадает с ключом шифрования. Если пароль был указан верно, то данные расшифруются корректно и пользователь сможет с ними работать. Если пароль был введен неверно, то данные расшифруются некорректно.

Если к данным на носителе применяется только шифрование и размер данных велик, то в целях обнаружения ввода некорректного пароля без дешифрования всех данных на носителе информации перед шифрованием к открытому тексту может быть добавлена приставка. При использовании такого метода пароль будет проверяться путем дешифрования фрагмента шифротекста, содержащего приставку, и сравнения результата с ожидаемым значением приставки.

Смена пароля приводит к изменению связанных ключей защиты данных. Таким образом, всякий раз, когда пароль изменяется, любые данные, защищенные старым паролем, должны быть восстановлены (например, дешифрованы) с использованием соответствующего ключа защиты данных, связанного со старым паролем, и затем защищены заново (например, зашифрованы) с использованием соответствующего ключа защиты данных, связанного с новым паролем.

3.2. Разработка Password-Based Key Derivation Function на основе алгоритма Blum-Blum-Shub

В качестве демонстрации того, что низкая скорость исследуемого в данной работе генератора псевдослучайных последовательностей может оказаться полезной, была разработана PBKDF, которая в качестве псевдослучайной функции использует алгоритм Blum-Blum-Shub, в дальнейшем обозначаемый BBS.

Входные и выходные данные разработанного алгоритма приведены в таблице 1.

Таблица 1. Входные и выходные данные алгоритма

НА ВХОДЕ:	
P	Пароль
S	Соль
MK_Length	Требуемая длина мастер-ключа
$Dummy\_Bits\_Number$	Число холостых бит на выходе BBS
$True\_Bits\_Number$	Число полезных бит на выходе BBS
НА ВЫХОДЕ:	
MK	Мастер-ключ

Разработанный алгоритм генерации ключа на основе пароля имеет следующий вид:

$a = \lceil MK\_Length / True\_Bits\_Number \rceil;$
$r = MK\_Length - (a-1) \cdot True\_Bits\_Number;$
for $(i = 1,\ i < a,\ i{+}{+})$
{
  $T_i = 0;$
  $U_0 = S \,||\, Binary(i);$
  for $(j = 1,\ j < C,\ j{+}{+})$
  {
    $U_j = BBS(P, U_{j-1});$
    $T_i = T_i \oplus U_j;$
  }
}

return $MK = T_1 \| T_2 \| \ldots \| T_a < 0 \ldots r-1 >$.

Холостыми назовем первые $Dummy\_Bits\_Number$ бит, порожденных генератором BBS, которые не будут учитываться при создании мастер-ключа. Полезными назовем те биты, порожденные генератором BBS, которые будут принимать участие в генерации ключа. Мастер-ключ будем генерировать блоками. Общее количество блоков обозначим a. Чтобы получить необходимое количество бит ключа, независимо от того, сколько полезных бит выдает генератор BBS на каждой итерации, представим мастер-ключ в виде:

$$MK\_Length = (a-1) \cdot True\_Bits\_Number + r,$$

где $a-1$ — число блоков, в которых в мастер-ключ войдет ровно $True\_Bits\_Number$ бит, а r — число бит последнего блока, которые войдут в мастер-ключ.

Создадим два временных массива: T и U и для каждого блока i выполним следующее: обнулим ячейку массива T_i, в ячейку U_0 запишем результат конкатенации соли S и номера блока i, переведенного в бинарный вид функцией $Binary()$. Затем будем запускать BBS генератор и складывать результат со значением T_i до тех пор, пока не обнулится счетчик итераций. Мастер-ключ является результатом конкатенации всех блоков T. Последний блок T_a заменяется на первые r бит этого блока $T_a < 0 \ldots r-1 >$.

3.3. Программная реализация Password-Based Key Derivation Function на основе алгоритма Blum-Blum-Shub

Алгоритм генерации ключа на основе пароля, являющийся модификацией алгоритма Blum-Blum-Shub и описанный в предыдущем пункте, был реализован программно в среде Microsoft Visual Studio 2012 на языке программирования C++. Данная реализация представляет собой консольное приложение, предназначенное для запуска на компьютерах под управлением операционных систем Windows Vista, 7, 8, 10. В качестве генератора Blum-Blum-Shub используется реализованный ранее и описанный в [7] программный генератор, использующий теоретико-числовую библиотеку MPIR, а, следовательно, для запуска данного приложения необходимо наличие динамической библиотеки mpir.dll.

Пример запуска программы с параметрами $\{P$, S, C, MK_Length, $Dummy\_Bits\_Number$, $True\_Bits\_Number$, показан на рисунке 2.

Здесь Out_File_Name — имя файла, в который будет записан мастер-ключ. Значение остальных параметров совпадает с описанными в пункте 3.2. В данном примере программе передаются параметры $\{f9QhsF3pd1V1jfBdn8hNesd7ALBskjU2PXsd40u$, $BaDrjq6D2m1OSJ3vF2QvIerB480gmX7h11HVwLbsoiDRct8W39bd0PhfsE$, 10, 512, 1000, 10, $D:\backslash key.txt\}$. После того как ключ будет сгенерирован, программа оповестит об этом фразой «Key has been successfully generated».

На рисунке 3 представлен пример файла содержащего 512-битный мастер-ключ.

В таблице 2 представлено среднее время работы реализованной PBKDF, которая формирует 512-битный ключ. При этом $Dummy\_Bits\_Number = 1000$, $True\_Bits\_Number = 10$.

В таблице 3 представлено среднее время формирования 512-битного ключа. При этом $Dummy\_Bits\_Number = 10$, $True\_Bits\_Number = 1000$.

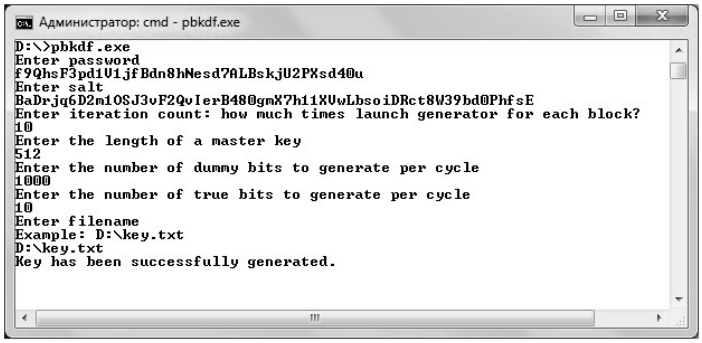

Рис. 2. Пример работы PBKDF на основе алгоритма Blum-Blum-Shub.

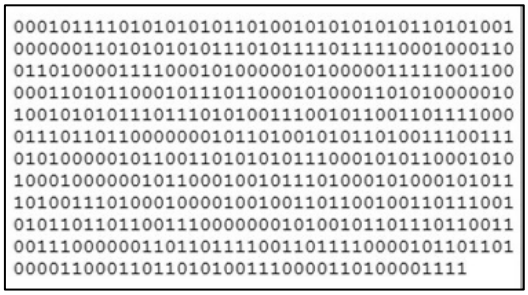

Рис. 3. Пример файла, содержащего мастер-ключ.

Таблица 2. Время формирования ключа

Количество итераций	Время, сек
2	20,971
4	45,211
6	65,966
8	84,857
10	106,745

Таблица 3. Время формирования ключа

Количество итераций	Время, сек
2	0,483
4	1,072
6	1,335
8	2,145
10	2,907

Из таблиц 2 и 3 следует, что, присваивая различные значения *Dummy _ Bits _ Number* , *True _ Bits _ Number* и *C* , мы можем добиться необходимого времени генерации ключа требуемой длины.

4. Заключение

В статье предложен альтернативный способ реализации алгоритма Password-Based Key Derivation Function, основанный на применении криптографически стойкого генератора псевдослучайных последовательностей Blum-Blum-Shub. Таким образом, получен алгоритм генерации криптографически стойких ключей из легко запоминаемых паролей, который в силу высокой вычислительной сложности генератора Blum-Blum-Shub замедляет процесс аутентификации/идентификации пользователя или процесс дешифрования данных, что, в свою очередь, значительно затрудняет реализацию атак со словарем, атак перебором, а также атак по предвычисленным таблицам. В результате экспериментов выявлено, что изменяя входные параметры алгоритма, можно регулировать время выработки криптографического ключа и тем самым подстраивать предложенный алгоритм под конкретные задачи аутентификации и шифрования.

Литература

[1] Blum, L. A simple unpredictable pseudo-random number generator/ L. Blum, M. Blum, M.Shub // SIAM Journal on Computing. – 1986. – Vol. 15(2). – P. 364-383.

[2] Стасев, Ю. В., Потий, А.В., Избенко, Ю.А. Исследование методов криптоанализа поточных шифров [Электронный ресурс]. – Режим доступа: http://www.nrjetix.com/fileadmin/doc/publications/articles/stasev_potiy_izbenko_ru.pdf (10.11.2016).

[3] Брассар, Дж. Современная криптология / Дж. Брассар. – Москва: Полимед, 1999. – 107 с.

[4] Sidorenko, A. Concrete security of the Blum-Blum-Shub pseudo-random generator / A. Sidorenko, B. Schoenmakers // Lecture notes in computer science. – Berlin: Springer-Verlag, 2005. – P. 355-375.

[5] Junod, P. Cryptographic secure pseudo-random bits generation: the Blum-Blum-Shub generator [Electronic resource]. — Access mode: http://crypto.junod.info/bbs.pdf (22.11.2016)

[6] Sewak, K. FPGA implementation of 16 bit BBS and LFSR PN sequence generator: a comparative study / K. Sewak, P. Rajput, A. K.Panda // IEEE Students' Conference on Electrical, Electronics and Computer Science. – Bhopal: IEEE, 2012. – P. 1-3.

[7] Выборнова, Ю.Д. Реализация генератора Blum-Blum-Shub и исследование его основных характеристик / Ю.Д. Выборнова // IN SITU. – 2015. – Т. 4, № 4. – С. 36-38.

[8] Turan, M. Recommendation for password-based key derivation / M. Turan, E. Barker, W. Burr, L. Chen – NIST, 2010. – 18 p.