Network Working Group
Request for Comments: 4086
BCP: 106
Obsoletes: 1750
Category: Best Current Practice

D. Eastlake, 3rd
Motorola Laboratories
J. Schiller
MIT
S. Crocker

June 2005

Требования к случайным значениям для безопасности

Randomness Requirements for Security

Статус документа

Этот документ служит для обмена опытом (Best Current Practices) в сообществе Internet и служит приглашением к дискуссии в целях дальнейшего развития. Документ может распространяться свободно.

Авторские права

Аннотация

Системы защиты строятся на основе сильных криптографических алгоритмов, которые препятствуют попыткам анализа повторяющихся последовательностей (pattern analysis). Однако защита этих систем зависит от генерации случайных величин для паролей, криптографических ключей и т. п. Использование псевдослучайных процессов для генерации таких значений может приводить к псевдозащите. Изощренным злоумышленникам может оказаться проще воспроизвести среду, использованную для генерации секретных величин, и искать в их в меньшем наборе возможных вариантов, нежели в полном пространстве значений.

Выбор случайных величин, которые поставят в тупик мотивированного и обладающего ресурсами злоумышленника, весьма сложен. В этом документе показано множество «подводных камней», связанных с использованием источников со слабой энтропией и традиционных методов генерации псевдослучайных значений. Документ рекомендует использование подходящих аппаратных средств для генерации случайных значений и показывает, как для этого могут применяться имеющиеся в разных системах компоненты. Документ включает также рекомендации по решению задачи в случаях отсутствия аппаратных решений и примеры оценки размера случайных величин для разных приложений.

Оглавление

1. Введение и обзор

Программная криптография применяется все более широко, но ещё не стала всеобъемлющей. Системы типа SSH, IPSEC, TLS, S/MIME, PGP, DNSSEC и Kerberos достаточно развиты и стали частью «сетевого ландшафта» [SSH] [IPSEC] [TLS] [S/MIME] [MAIL_PGP*] [DNSSEC*]. Для сравнения, на момент подготовки предыдущей версии этого документа [RFC1750] в 1994 г., единственной криптографической спецификацией Internet в рамках IETF был протокол Privacy Enhanced Mail [MAIL_PEM*].

Эти системы обеспечивают существенную защиту от подмены и отслеживания пакетов. Однако все такие системы имеют существенную особенность, связанную с генерацией секретных и непредсказуемых (т. е., случайных) чисел.

Отсутствие средств общего назначения для генерации таких случайных значений (т. е., действительно непредсказуемых источников) образует брешь в разработке криптографических программ. Для разработчиков, желающих создать процедуру генерации ключей или паролей, работающую на разных аппаратных платформах, это создаёт реальную проблему.

Отметим требование для данных иметь очень низкую вероятность их предсказания или определения злоумышленником. При использовании псевдослучайных данных, которые удовлетворяют только традиционным статистическим тестам или основаны на значениях с ограниченным диапазоном (типа часов) вероятность возникновения проблем весьма велика. Иногда такие псевдослучайные значения могут быть найдены злоумышленниками путём простого выбора из небольшого пространства возможных значений.

Этот документ из серии Best Current Practice описывает методы создания случайных величин, которые устойчивы к атакам. В новых системах рекомендуется включать аппаратную генерацию случайных чисел или обеспечивать доступ к имеющемуся оборудованию для этих целей. Предлагаются также методы для тех случаев, когда нужное оборудование не доступно и приведены оценки количества случайных битов, требуемого для некоторых приложений.

2. Основные требования

Сегодня требования хаотичности (randomness) предъявляются обычно при выборе пользовательских паролей, которые обычно являются символьными строками. Обычно предсказуемый пароль не обеспечивает безопасности. Для паролей неоднократного применения желательно делать пароли запоминающимися для пользователя. Для этого целесообразно использовать произносимые строки или фразы, состоящие из обычных слов. Однако это не делает пароли сложно предсказуемыми.

Из криптографии исходит и множество других требований. Криптографические методы могут применяться для обеспечения различных услуг, включая защиту конфиденциальности и контроль подлинности (аутентификацию). Такие услуги базируются на значениях, обычно называемых «ключами», которые не известны посторонним и не могут быть угаданы злоумышленниками.

Протокол TCP/IP также использует хаотические (случайные) значения для начальных порядковых номеров [RFC1948].

Вообще говоря, приведённые выше примеры иллюстрируют также два разных типа случайных величин, которые могут оказаться нужны. Для используемых людьми паролей единственным важным фактором является непредсказуемость пароля. Не имеет значения, что в паролях, содержащих символы ASCII, старший бит каждого байта будет нулевым. С другой стороны, для ключей фиксированного размера и подобных им элементов обычно желательны действительно хаотические значения, биты которых соответствуют статистическим тестам на хаотичность.

В некоторых случаях - например, использование симметричного шифрования с одноразовыми паролями или алгоритмов типа [AES[1]], стороны, желающие обеспечить конфиденциальность и/или аутентификацию обмена данными между собой, должны знать общий секретный ключ. В других ситуациях, когда применяются асимметричные методы шифрования с открытым ключом (public key), ключи являются парными. Один ключ в такой паре является секретным (private) и должен быть известен лишь одной стороне, а второй - открытым (public) и может быть опубликован для общего доступа. Определить секретный ключ по открытому вычислительными средствами невозможно и знание открытого ключа не поможет злоумышленнику [ASYMMETRIC]. Общие вопросы шифрования рассмотрены в работах [SCHNEIER, FERGUSON, KAUFMAN].

Требования к случайным величинам по частоте и объёму существенно отличаются в разных криптографических системах. Для чистого RSA случайные значения требуются только при генерации новой пары ключей, а после этого можно подписать любое число сообщений без дополнительных потребностей в случайных значениях. Для основанного

[1]US Advanced Encryption Standard - стандарт США для усовершенствованного шифрования.

на открытых ключах алгоритма DSA[1], разработанного в NIST[2], требуются качественные случайные значения при создании каждой подписи [DSS]. А шифрование с «одноразовым блокнотом» (one-time pad - в принципе, наиболее сильный из возможных методов шифрования) требует хаотичность равного объёма для всех обрабатываемых сообщений. Общие вопросы шифрования рассмотрены в работах [SCHNEIER, FERGUSON, KAUFMAN].

Во многих случаях злоумышленники могут попытаться определить секретный ключ методом проб и ошибок. Это возможно в тех случаях, когда размер ключа достаточно мал. Вероятность достижения успеха злоумышленником должна быть достаточно низкой (в зависимости от конкретного приложения). Размер пространства, в котором злоумышленник будет пытаться найти ключ, определяется теорией информации [SHANNON]. Это зависит от общего числа возможных секретных значений и вероятности каждого значения.

$$\text{Биты информации} = \sum_{i} - p_i * \log_2 (p_i)$$

где i меняется от 1 до числа возможных секретных значений, а p_i - вероятность значения с номером i (поскольку все значения p меньше 1, логарифм будет отрицательным, а все суммируемые значения - неотрицательными).

Если имеется 2^n различных равновероятных значений, злоумышленник должен будет подобрать значения n битов, а для этого потребуется перебрать в среднем половину возможных значений или $2^{(n-1)}$. Если вероятность различных значений разная, объем информации снижается и злоумышленнику в среднем потребуется меньше усилий. В частности, возможность исключения злоумышленником тех или иных значений, как невозможных или маловероятных, упрощает его работу, позволяя в первую очередь проверять наиболее вероятные значения.

Рассмотрим, например, криптосистему со 128-битовыми ключами. Если эти ключи создаются с использованием фиксированного генератора псевдослучайных чисел, который работает с 8-битовой затравкой (seed), нарушителю потребуется перебрать лишь 256 ключей (запуская генератор псевдослучайных чисел с каждым из возможных значений seed), а не 2^{128}, как могло показаться. В таких 128-битовых ключах содержится лишь 8 битов «информации».

Хотя приведённый выше анализ в общем случае корректен, он может вводить в заблуждение для некоторых случаев криптоанализа, которые могут играть важную роль в действиях нарушителей. В качестве примера предположим, что имеется генератор псевдослучайных чисел, создающий 128-битовые ключи (как в предыдущем примере), который первую половину времени генерирует 0, а во вторую выбирает случайное значение из оставшихся $2^{128} - 1$. Приведённое выше уравнение Шеннона говорит о том, что в таких ключах будет содержаться 64 бита информации, но злоумышленник может нарушить защиту для первой половины случаев, просто попробовав значение 0. Поэтому для криптографических целей полезно также принимать во внимание другие характеристики, типа минимальной энтропии

$$\text{min-entropy} = - \log (\text{maximum} (p_i))$$

где параметр i имеет описанный выше смысл. Используя это уравнение мы получим 1 бит min-entropy для предложенного выше распределения в отличие от 64 битов для классической энтропии Шеннона.

Был определён непрерывный спектр значений энтропии (иногда его называют энтропией Renyi), задаваемый параметром r. Случай r = 1 соответствует энтропии Шеннона, а бесконечное значение r - min-entropy. При r = 0 это просто log(n), где n указывает число отличных от нуля вероятностей. Энтропия Renyi представляет собой невозрастающую функцию r, поэтому min-entropy всегда является наиболее консервативной мерой энтропии и обычно лучше всего подходит для криптографической оценки [LUBY].

Статистически проверенная хаотичность в традиционном понимании **не** совпадает с требуемой для защиты непредсказуемостью.

Например, использование широко распространённых постоянных последовательностей типа стандартных математических таблиц CRC является очень слабой защитой от злоумышленников. Нарушители, знающие или догадывающиеся об этом, смогут легко разрушить любую защиту (прошлую или будущую), основанную на последовательностях [CRC]. Другим примером может быть использование AES с постоянным ключом для шифрования последовательности целых чисел (типа 1, 2, 3, …), которое на выходе будет давать превосходную статистически хаотичность, которая будет полностью предсказуемой. С другой стороны, последовательное бросание кубика с символом ASCII на каждой грани будет давать статистически слабую хаотичность, но достаточно высокую непредсказуемость. Поэтому подчеркнём, что результаты статистических тестов ничего не говорят о предсказуемости.

3. Источники энтропии

Как правило источники энтропии сильно зависят от реализации. После накопления достаточно объёма энтропии, она применяется в качестве затравки для создания нужного объёма криптостойких псевдослучайных значений, как описано в разделах 6 и 7, после чего выполняется нормализация (de-skewed[3]) или смешивание, как описано в разделах 4 и 5.

Возможно ли создание в будущем истинной, сильной и переносимой хаотичности? Возможно. Все, что для этого требуется, - физический источник непредсказуемых случайных чисел.

Тепловой шум (иногда называемый шумом Джонсона в микросхемах) или радиоактивный источник и высокочастотный автогенератор[4] (free-running oscillator) может обеспечить прямое решение задачи [GIFFORD]. Оно включает немного оборудования, которое легко включить в архитектуру компьютерной системы в качестве составной части. Подходит также большинство входных аудио- видеоустройств [TURBID]. Кроме того, любая система с вращающимся диском или кольцевым генератором (ring oscillator) и стабильными (кварцевыми) часами или чем-то похожим может служить подходящим источником хаотичности ([DAVIS] и параграф 3.3). Для этого нужно лишь признание со стороны производителей компьютерного оборудования целесообразности добавления в системы небольшого объёма оборудования и программ для доступа к нему.

---

[1] Digital Signature Algorithm - алгоритм цифровой подписи.
[2] US National Institute of Standards and Technology - Национальный институт стандартов и технологий США.
[3] Букв., устранение перекосов. *Прим. перев.*
[4] Генератор с положительной обратной связью без источника синхросигнала. *Прим. перев.*

Комитет ANSI X9 в настоящее время занимается разработкой стандарта, включающего раздел, посвящённый источникам энтропии (см. часть 2 [X9.82]).

3.1. Требуемый объем

Какой объем непредсказуемых данных нужен? Можно ли сформулировать количественные требования для этого (например, число непредсказуемых битов в секунду)?

Ответ на эти вопросы не так прост, как хотелось бы. Для AES ключ может иметь размер 128 битов и, как показано в примере раздела 8, даже для самых защищённых систем вряд ли потребуется больше 200 битов ключевого материала. Если нужна серия ключей, их можно создать из достаточно случайной затравки (стартовое значение), используя криптостойкую последовательность, как описано в параграфе 6.2. Нескольких сотен битов, генерируемых при запуске или однократно в течение дня, достаточно при использовании таких методов. Даже если случайные биты генерируются по одному в секунду и невозможно «распараллелить» процесс их генерации, ожидание в течение 200 секунд для требующих сильной защиты приложений не будет, скорей всего, критичным.

Получение этих значений является тривиальной задачей. Это может сделать человек, многократно подбрасывая монетку, а практически любое оборудование способно решить такую задачу гораздо быстрее.

3.2. Оборудование, которое можно применить

Как описано ниже, многие компьютеры включают в свой состав оборудование, которое с известными предосторожностями можно использовать для генерации действительно случайных значений.

3.2.1. Использование имеющихся видео-аудио входов

Во многих компьютерах имеются входы для приёма внешних аналоговых сигналов с целью их последующей оцифровки (например, от микрофонов или видеокамер). Входной поток от микрофонного порта, к которому не подключён микрофон, или от видеокамеры, объектив которой закрыт непрозрачной крышкой, определяется в основном тепловыми шумами. Если уровень усиления в системе достаточно высок для восприятия таких шумов, они могут обеспечить достаточно качественную генерацию случайных битов. Этот метод очень сильно зависит от используемого оборудования.

Например, в некоторых UNIX-системах можно считывать данные из устройства /dev/audio, не подключая микрофона или принимая через него лишь слабый фоновый шум. Принимаемые данные будут представлять собой случайный шум, хотя им следует доверять лишь после определённой проверки (оборудование может работать некорректно и вносить свои шумы) и нормализации.

Комбинируя эти данные с компрессией (для нормализации, см. раздел 4), можно создать большой объем случайных данных среднего качества с помощью команды

```
cat /dev/audio | compress - >random-bits-file
```

Подробное описание этого типа источников хаотичности приведено в [TURBID].

3.2.2. Использование имеющихся дисков

Дисковые накопители имеют небольшие случайные флуктуации скорости вращения, обусловленные хаотической турбулентностью в воздухе [DAVIS, Jakobsson]. Измерение времени поиска на диске (seek-time) на низком уровне позволяет получить последовательность случайных значений. Эти данные обычно имеют сильную корреляцию, поэтому требуется их существенная обработка, как описано ниже в параграфе 5.2. Тем не менее, многолетние эксперименты показывают, что при использовании такой обработки даже медленные диски легко позволяют получить 100 и более битов в минуту случайных данных превосходного качества.

При каждом увеличении скорости процессоров, вместе с которым повышалось разрешение при считывании перемещений головок дисковых устройств или возрастала скорость таких перемещений, что обеспечивало рост скорости генерации случайных значений таким способом. На момент подготовки этого документа типичные компьютеры и диски позволяли получать более 10 000 случайных битов в секунду. Эти методы используются в генераторах случайных чисел в составе библиотек разных операционных систем.

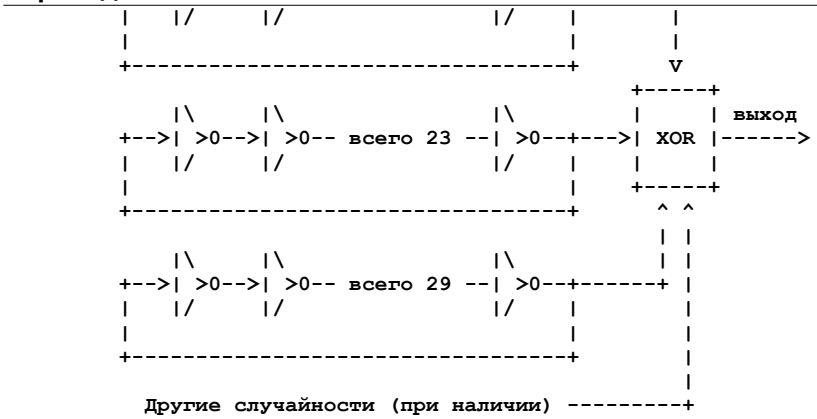
> Примечание. Добавление кэш-памяти в дисковые контроллеры не оказывает существенного влияния на этот метод, если игнорировать очень малое время поиска, которое просто соответствует выбору записи из кэша.

3.3. Кольцевые генераторы

При проектировании или программировании микросхемы можно соединить в кольцо нечётное количество логических элементов для создания автогенератора (free-running oscillator). Делая выборки в какой-либо точке кольца через одинаковые интервалы времени (например, синхронизируемые от внешнего стабильного генератора), можно получить некоторый объем хаотичности, обусловленной вариациями частоты колебаний автогенератора. Можно повысить скорость сбора энтропии, используя операцию XOR для выборок от нескольких кольцевых генераторов простым числом логических элементов. Иногда рекомендуют в таких случаях использовать нечётное число колец, чтобы даже при возникновении синхронизации между кольцами значения результирующей выборки менялись. Другим источником данных для выборки может служить шумящий диод.

Выборки из таких источников должны нормализоваться, как и для случая использования диска (см. раздел 4). Для определения объёма получаемой хаотичности в каждом случае потребуется специальное исследование. Однако в любом случае это может служить хорошим источником хаотичности, созданным на основе минимального количества современного оборудования.

Например, в стандарте IEEE 802.11i предлагается использование устройства, показанного на рисунке, с изоляцией каждого кольца для предотвращения нежелательной синхронизации и последующей дополнительной обработкой результата [IEEE_802.11i].



3.4. Часы и порядковые номера

Время по часам компьютера или аналогичные значения от операционной системы или оборудования обеспечивают значительно более низкий уровень непредсказуемости битов, нежели может казаться на основании их спецификаций.

Тестирование часов для разных операционных систем показало, что их поведение может меняться очень существенно и неожиданными способами. Конкретная версия операционной системы при использовании на том или ином наборе оборудования может обеспечивать микросекундное разрешение для часов, а на другом оборудовании будет давать всегда одни и те же значения в младших битах, что существенно снижает разрешение. Это означает, что при последовательном считывании показаний часов будут возвращаться идентичные значения даже при интервалах считывания, за которые при номинальном разрешении показания часов изменились бы. В некоторых случаях считывание показаний часов может давать искусственные последовательные значения, поскольку дополнительный код может увидеть, что показания часов не изменились и просто добавит 1 к возвращаемому значению. Разработка переносимого кода для генерации непредсказуемых значений на основе таких системных часов дополнительно осложняется тем, что разработчик не всегда знает свойства конкретных системных часов.

Использование аппаратных номеров (например, MAC-адресов Ethernet) обеспечивает значительно меньше уникальных битов, чем может казаться. Такие значения обычно хорошо структурированы и субполя имеют ограниченные диапазоны возможных значений, а значения номеров в целом могут быть легко предсказуемыми на основании даты выпуска или других данных. Например, очевидно, что компания, производящая компьютеры и адаптеры Ethernet, будет скорей всего использовать сетевые адаптеры собственного производства и это существенно ограничивает диапазон возможных адресов.

Описанные выше проблемы существенно осложняют разработку кода для генерации непредсказуемых значений, который сможет работать на разных платформах и операционных системах.

3.5. Синхронизация и значения от внешних событий

Можно фиксировать время и координаты перемещения мыши, факты нажатия клавиш и другие пользовательские события. Эта информация с некоторыми оговорками может служить источником непредсказуемых данных. На некоторых машинах информация о нажатии клавиш буферизуется. Хотя интервалы между нажатиями клавиш пользователем непредсказуемо меняются, получить доступ к значениям этих интервалов может оказаться не просто. Другая проблема заключается в отсутствии стандартных методов получения информации о времени событий. Это осложняет использование такого метода для создания стандартных программ, предназначенных для работы на разных классах машин.

Величину перемещения мыши и факты нажатия клавиш обычно получить гораздо проще, нежели информацию о времени этих событий, однако такая информация может оказаться значительно более предсказуемой, поскольку действия пользователя могут достаточно часто повторяться.

Могут применяться и другие внешние события (например, время прибытия пакетов из сети и их размер), но делать это следует осторожно. В частности, следует принимать во внимание возможность воздействия на параметры сетевого трафика со стороны нарушителей, а также потерю предшествующей информации при перезагрузке системы. Если такие данные могут быть изменены, их не следует рассматривать в качестве доверенного источника энтропии.

В принципе, почти любой сигнал от внешнего датчика (например, неразобранный сигнал от радиоприёмника или значение температуры от внешнего источника), подобающим образом подключённого к компьютеру, могут служить источником хаотичности. Но во всех случаях следует принимать во внимание возможность целенаправленного изменения данных нарушителями, а также обеспечиваемый объем хаотичности.

Описанные выше методы достаточно устойчивы к атакам, когда у нарушителя нет физического доступа к измерительному устройству (датчику). Например, эти методы подходят для систем, расположенных в местах с ограниченным доступом и фиксацией попыток доступа к датчикам. В любом случае повышение точности измерения времени или показателей внешнего датчика увеличивает скорость генерации хаотических данных.

3.6. Неаппаратные источники хаотичности

Лучшими источниками входной энтропии являются аппаратные источники случайных значений типа кольцевых генераторов, измерителей скорости доступа к дискам, источников теплового шума или счетчиков радиоактивного излучения. Однако при отсутствии таких источников существуют и другие возможности. Источниками хаотических данных могут служить системные часы, буферы ввода вывода или системные буферы, пользовательские, системные, аппаратные или сетевые порядковые номера, а также ввода данных пользователем. К сожалению, каждый из перечисленных источников в некоторых условиях может создавать очень ограниченный или предсказуемый набор значений.

Некоторые из упомянутых источников будут достаточно хороши для многопользовательских систем, в которых сами пользователи являются существенной частью хаотичности. Однако в небольших однопользовательских системах (особенно при старте) ситуация значительно хуже и нарушители зачастую могут оказаться способными предсказать собираемые значения.

Рекомендуется использовать множество входов со случайными значениями в комбинации с сильной функцией смешивания - это позволяет преодолеть слабость любого отдельного источника. Время и содержимое «случайных» нажатий клавиш пользователем может давать сотни случайных битов, но следует принять во внимание консервативные допущения. Например, одним из таких допущений может служить предположение от добавлении каждым интервалом между двумя нажатиями клавиш нескольких случайных битов при условии, что этот интервал отличается от всех предшествующих. Аналогичное допущение применимо и к вводимым при нажатиях клавиш символам. Таким образом, интервал и символ, которые повторяют предыдущие, рассматриваются, как не добавляющие новых случайных данных. Результаты смешивания времени нажатия клавиш с вводимыми при этом символами могут в дальнейшем комбинироваться с данными от часов и других источников.

Такая стратегия позволяет создать практичный переносимый код для генерации высококачественных случайных чисел, применяемых для защиты, даже в тех случаях, когда некоторые из применяемых источников достаточно слабы в той или иной среде применения. Однако это может не сберечь от атак на небольшие, однопользовательские или встраиваемые системы, особенно в тех случаях, когда нарушитель мог когда-либо видеть процесс генерации. Предпочтительными остаются аппаратные реализации случайных данных.

4. Нормализация

Есть ли какие-то конкретный требования к форме распределения значений, собираемых для энтропии с целью генерации случайных чисел? Хорошая новость заключается в том, что такое распределение не обязательно должно быть однородным. Требуется лишь консервативная оценка степени неоднородности этого распределения. Ниже описаны простые методы выравнивания (de-skew) битового потока, а в параграфе 5.2 - более сильные криптографические методы.

4.1. Использование чётности потока для нормализации

В качестве простого и непрактичного примера рассмотрим принятие битовой строки произвольного размера и её отображение в строку нулей и единиц. Отображение не даст совершенно однородного распределения, но оно может быть близким к желаемому. Одним из таких отображений является использование чётности строки. Преимуществом его является устойчивость ко всем возможным уровням неоднородности (перекоса - skew), вплоть до максимального, и простота аппаратной реализации.

Ниже приведён анализ числа битов, которые должны быть собраны.

Предположим, что отношение единиц и нулей в выборке составляет (0.5 + E) к (0.5 - E), где E имеет значение из диапазона от 0 до 0,5 и служит мерой «эксцентричности» распределения. Рассмотрим распределение функции чётности для выборки размером N битов. Вероятность того, что чётность будет иметь значение 1 или 0 определяется суммой нечётных или чётных компонент в биномиальном разложении $(p + q)^N$, где p = 0.5 + E указывает вероятность единицы, а q = 0.5 - E указывает вероятность нуля.

Эти суммы легко рассчитать по формулам

$$1/2 * ((p + q)^N + (p - q)^N)$$

и

$$1/2 * ((p + q)^N - (p - q)^N).$$

(какая из формул указывает вероятность того, что чётность будет иметь значение 1, зависит от чётности или нечётности значения N).

Поскольку p + q = 1 и p - q = 2E, эти выражения упрощаются до

$$1/2 * [1 + (2E)^N]$$

и

$$1/2 * [1 - (2E)^N].$$

Ни одно из этих выражений не будет в точности равно 0,5, пока E отлично от 0, но они могут быть сколь угодно близки к 0,5. Если мы хотим получить вероятность из диапазона d от значения 0,5, должно выполняться условие

$$(0.5 + (0.5 * (2E)^N)) < 0.5 + d.$$

Решение даёт для N значение N > log(2d)/log(2E). Отметим, что 2E < 1, поэтому логарифм будет отрицательным. Деление на отрицательное число обращает смысл неравенства.

В таблице показаны размеры строки выборки (N), требуемой для разных уровней неоднородности (перекоса) распределения из диапазона от 0,001 до 50/50.

Вероятность 1	***E***	***N***
0,5	0	1
0,6	0,10	4
0,7	0,20	7
0,8	0,30	13
0,9	0,40	28
0,95	0,45	59
0,99	0,49	308

Последняя строка показывает, что даже при перекосе 99% в пользу единиц чётность строки размером 308 битов будет попадать в диапазон распределения от 0,001 до 50/50. Однако, как будет показано в параграфе 5.2, есть более сильные методы получения большего объёма доступной энтропии.

4.2. Использование временных отображений для нормализации

Другим методом, изначально предложенным фон Нейманом (von Neumann) [VON_NEUMANN], является представление битового потока, как последовательности неперекрывающихся пар. Из выбранных пар отбрасываются все, имеющие значения 00 и 11, затем пары 01 интерпретируются, как 0, а пары 10 - как 1. Предположим, что вероятность 1 в битовом потоке составляет 0,5+E, а вероятность 0 - 0,5-E, где E показывает меру «эксцентричности» распределения, как описано в предыдущем параграфе. В этом случае вероятность каждой из возможных пар определяется выражениями, приведёнными в таблице.

Пара	*Вероятность*
00	$(0,5 - E)^2 = 0,25 - E + E^2$
01	$(0,5 - E)*(0.5 + E) = 0,25 - E^2$
10	$(0,5 + E)*(0.5 - E) = 0,25 - E^2$
11	$(0,5 + E)^2 = 0,25 + E + E^2$

Этот метод будет полностью устранять любое смещение, однако требует неопределённого числа входных битов для создания желаемого объёма данных на выходе. Вероятность отбрасывания любой пары составляет $0,5 + 2E^2$, поэтому ожидаемое число входных битов для создания на выходе X битов составит $X/(0,25 - E^2)$.

Этот метод предполагает получение входных битов из потока, где каждый бит имеет такую же вероятность значений 0 или 1, как и любой другой бит этого потока и биты декоррелированы (т. е., берутся из идентичных независимых распределений). Если биты берутся из связанных источников, приведённые выше рассуждения теряют силу.

Описанный выше метод является также иллюстрацией того, как простой статистический анализ может ввести в заблуждение, поскольку он не всегда позволяет увидеть характеристики, которыми могут воспользоваться злоумышленники. Если толковать алгоритм не вполне корректно и пользоваться перекрывающимися последовательными парами вместо неперекрывающихся, статистический анализ не покажет различий. Однако в этом случае вместо несмещенной, декоррелированной последовательности случайных нулей и единиц будет получена легко предсказуемая последовательность чередующихся значений битов 1 и 0.

4.3. Использование FFT для нормализации

Когда реальные данные содержат последовательности битов со значительной корреляцией, они все равно могут включать полезную энтропию. Для её извлечения могут применяться разные преобразования и наиболее эффективные из них описаны в параграфе 5.2.

Использование для данных преобразования Фурье или его оптимизированного варианта FFT представляет в основном теоретический интерес. Можно показать, что этот метод позволяет устранить сильную корреляцию данных. Если подходящие данные обработать и устранить из них остаточные корреляции, спектральные составляющие будут представлять собой статистически не зависимые случайные значения с нормальным распределением [BRILLINGER].

4.4. Использование компрессии для нормализации

Обратимые методы сжатия также обеспечивают выравнивание неоднородности битового потока. Это напрямую следует из определения обратимого сжатия и формулы в параграфе 2 для объёма информации в последовательности. Поскольку сжатие является обратимым, объем информации в сжатой и несжатой последовательности должен совпадать. В соответствии с уравнением Шеннона это возможно лишь в том случае, когда в среднем вероятности коротких последовательностей распределены более равномерно, нежели вероятности длинных последовательностей. Следовательно, более короткие последовательности должны выравниваться по сравнению с входными данными.

Однако многие методы сжатия добавляют в свой выходной поток легко предсказуемые префиксы, а также могут периодически помещать в этот поток похожие последовательности или создавать в нем иные регулярности. Поэтому методы сжатия следует рассматривать, как более грубые по сравнению с описанными в параграфе 5.2. По крайней мере начальную часть сжатой последовательности (префикс метода) следует пропускать, используя для приложений, требующих случайных битов, лишь последующую часть сжатого потока.

5. Смешивание

Какая стратегия обеспечивает получение непредсказуемых случайных значений при отсутствии сильного и надёжного аппаратного источника энтропии? Это получение входных данных из множества несвязанных источников и смешивание их с помощью специальной функции. Функция смешивания будет сохранять энтропию каждого из имеющихся источников даже если другие при будут давать фиксированные или легко предсказуемые значения (низкая энтропия). Такая модель применима и при наличии аппаратных источников, поскольку в них возможны отказы. Однако следует также принимать во внимание возможный рост отказа системы в целом в результате усложнения программных компонент.

После того, как найдены хорошие источники, описанные в разделе 3, и значения смешаны, как описано здесь, остаётся выбрать правильную «затравку» (seed). Это позволит создать большой объем криптостойкого материала, как описано в разделах 6 и 7.

Сильной функцией смешивания считается такая функция, которая принимает на входе множество потоков и даёт на выходе результат, каждый бит которого является результатом сложной, нелинейной функции от значений всех входных битов. В среднем такая функция будет менять примерно половину выходных битов при изменении единственного бита на входе. По причине сложной и нелинейной связи изменение конкретного выходного бита не гарантируется в результате изменения любого конкретного бита на входе.

Рассмотрим задачу преобразования битового потока, смещённого в сторону 0 или 1 и имеющего некоторую предсказуемость, в более короткий поток, который будет более случайным, как отмечено в разделе 4. Это будет просто ещё одним случаем, когда желательная сильная функция смешивания, принимающая входные биты и возвращающая меньшее число битов на выходе. Описанный в параграфе 4.1 метод на основе чётности множества битов является просто результатом многократного использования операции XOR. Этот метод обеспечивает простейшую функцию смешивания, описанную ниже. Использование более сильной функции смешивания для получения больше хаотичности в потоке со смещением описано в параграфе 5.2. См. также работу [NASLUND].

5.1. Тривиальная функция смешивания

В целях разъяснения рассмотрим тривиальный пример, когда однобитовые входные потоки смешиваются с помощью функции «исключительное-ИЛИ» (XOR[1]). Эта функция эквивалентна сложению без переноса, как можно видеть из таблицы. Это вырожденный случай, когда выход всегда изменяется при смене любого из входных битов. Однако, несмотря на простоту, она служит хорошей иллюстрацией.

Вход 1	*Вход 2*	*Выход*
0	0	0
0	1	1
1	0	1
1	1	0

Если несвязанные входы 1 и 2 скомбинировать таким способом, хаотичность результата будет лучше (меньшее смещение), чем на любом из входов. Если воспользоваться определённой в параграфе 4.1 «эксцентричностью» E, для результата функции будет выполняться уравнение

$$E_{output} = 2 * E_{input\ 1} * E_{input\ 2}$$

Поскольку E не может быть больше 1/2, эксцентричность всегда снижается за исключением случая полного перекоса хотя бы на одном из входов. Это хорошо видно из таблицы, где верхняя строка и левая колонка показывают входные значения, а в остальных ячейках указаны соответствующие выходные значения «эксцентричности.

E	**0,00**	**0,10**	**0,20**	**0,30**	**0,40**	**0,50**
0,00	0,00	0,00	0,00	0,00	0,00	0,00
0,10	0,00	0,02	0,04	0,06	0,08	0,10
0,20	0,00	0,04	0,08	0,12	0,16	0,20
0,30	0,00	0,06	0,12	0,18	0,24	0,30
0,40	0,00	0,08	0,16	0,24	0,32	0,40
0,50	0,00	0,10	0,20	0,30	0,40	0,50

Однако следует отметить, что при расчёте предполагалось отсутствие корреляции между входными значениями. Если в качестве входных данных использовать, например, значение чётности числа минут с полуночи на двух часах, согласованных с точностью до нескольких секунд, значения обоих входов будут представляться достаточно случайными при их отборе через случайные интервалы продолжительностью больше минуты. Однако при комбинировании этих данных с помощью операции XOR на выходе всегда будет 0.

5.2. Сильные функции смешивания

Стандарт US Government Advanced Encryption Standard [AES] служит примером использования сильной функции смешивания для множества битовых значений. В нем принимается до 384 битов на входе (128 битов данных и 256 битов ключа), а на выходе возвращается 128 битов, каждый из которых определяется сложной нелинейной функцией всех входных битов. Другие функции шифрования с такими же характеристиками (например, [DES]) тоже могут применяться для смешивания всех входных битов данных и ключа.

Другим хорошим классом функций смешивания являются функции создания «отпечатков» сообщений (message digest) или функции хэширования типа стандартов US Government Secure Hash Standard [SHA*] и серии MD4, MD5 [MD4, MD5]. Эти функции принимают на входе практически не ограниченное число битов и возвращают сравнительно короткую последовательность в результате смешивания всех входных битов. Функции MD* дают на выходе 128 битов, SHA-1 - 160 битов, а другие функции SHA - до 512 битов.

Хотя функции создания дайджестов сообщений рассчитаны на переменный размер входных данных, AES и другие функции шифрования также могут применяться для комбинирования произвольного числа входов. Если устраивает 128 битов на выходе, входные данные можно «упаковать» в 128-битовые блоки данных и «ключи» AES, дополняя при необходимости нулями. После этого блоки данных шифруются с помощью «ключей» при использовании алгоритма AES в режиме ECM[2]. Кроме того, входные данные могут быть «упакованы» в один 128-битовый ключ и множество блоков данных с последующим расчётом CBC-MAC [MODES].

Более сложное смешивание следует применять в тех случаях, когда на выходе нужно получить более 128 битов и хочется использовать AES (отметим, что абсолютно невозможно получить на выходе больше хаотичных битов, нежели было передано на вход). В качестве примера предположим, что входные данные упаковываются в три блока A, B и C. Один может использоваться алгоритмом AES для шифрования A с ключом B, а затем с ключом C для создания первой части выходных данных. Далее B шифруется с ключом C, а затем A для создания второго блока и при необходимости C шифруется с ключами A и B для создания третьего выходного блока. Если нужны ещё блоки на выходе, можно изменить указанный выше порядок использования ключей. То же самое можно сделать с помощью хэш-функций, хэшируя разные подмножества входных данных или разные их копии с разными префиксами и/или суффиксами для создания множества выходных блоков.

В качестве примера использования сильной функции смешивания рассмотрим строку из 308 битов со смещением 99% в сторону 0. Описанный в параграфе 4.1 метод использования чётности уменьшает объем данных до 1 бита, но отклонение также уменьшается до 1/1000 в любом из направлений. С учётом выравнивания, рассмотренного в разделе 2 эта 308-битовая смещённая последовательность содержит более 5 битов информации. Таким образом, хэширование последовательности с помощью функции SHA-1 и извлечение 5 «нижних» битов будет давать 5 несмещенных случайных битов, а не один, как при использовании чётности. Кроме того, для некоторых приложений можно использовать весь вывод хэш-функции для сохранения всех «пяти с лишним» битов энтропии в 160-битовом значении.

5.3. Использование S-блоков для смешивания

Многие современные функции блочного шифрования, включая DES и AES, содержат модули, называемые S-Box (S-блок) или блоками подстановки. Он создают небольшой объем выходной информации из большого объёма входных данных, используя сложную нелинейную функцию смешивания, что обеспечивает эффект концентрации ограниченной энтропии входных источников на выходе функции.

[1]Exclusive Or.
[2]Electronic Codebook Mode - режим простой замены.

S-блоки иногда включают специальные логические функции (bent Boolean function) от чётного числа битов, дающие на выходе один бит с максимальной нелинейностью. Выходные результаты для всех входных пар, различающихся в любой битовой позиции, будут делиться точно пополам. S-блок, в котором каждый выходной бит создаётся bent-функцией так, что любая линейная комбинация этих функций также является bent-функцией, называется «совершенным S-блоком» (perfect S-Box).

S-блоки и разные варианты повторяющегося или каскадированного применения могут служить для смешивания[SBOX1, SBOX2].

5.4. Diffie-Hellman в качестве функции смешивания

Метод обмена ключами Diffie-Hellman позволяет двум сторонам создать общий секрет. Посторонний не сможет рассчитать значение этого секрета, даже при возможности просматривать сообщения, передаваемые между двумя сторонами. Этот общий секрет представляет собой результат смешивания двух начальных значений, созданных каждой из сторон [D-H].

Если эти начальные значения случайны и не связаны между собой, общий секрет будет содержать в себе энтропию обоих, но, естественно, не может обеспечивать больше хаотических битов, нежели задаёт размер созданного общего секрета.

Сказанное верно для случая, когда расчёт Diffie-Hellman выполняется втайне, но злоумышленник может видеть оба открытых ключа и, зная используемый модуль, может ограничить поиск пространством секретного ключа, который позволит ему рассчитать общий секретный ключ [D-H]. Консервативный подход оценивает непредсказуемость значения Diffie-Hellman, как худшую из непредсказуемостей двух входных значений. С учётом этого, а также значительного объёма расчётов Diffie-Hellman, данный метод не рекомендуется применять в качестве функции смешивания.

5.5. Использование смешивания для «растягивания» случайных битов

Хотя от функция смешивания не требуется давать на выходе такое же или меньшее число битов по сравнению с поданным на вход, смешивание не может «растянуть» имеющуюся на входе непредсказуемую случайность битов. Таким образом, 4 входа по 32 бита в каждом, из которых только 12 достаточно непредсказуемы (например, 4096 равновероятных значений), не могут дать более 48 непредсказуемых битов на выходе. Выход можно растянуть на сотни и тысячи битов, смешивая, например, с последовательными целыми числами, но пространство поиска для злоумышленника по прежнему будет содержать 248 значений. Более того, снижение числа битов на выходе по сравнению с числом входных битов имеет тенденцию усиления хаотичности на выходе.

В последней таблице параграфа 5.1 показано, что смешивание случайного бита с постоянным м помощью операции XOR будет давать случайный бит. Хотя это верно, такой способ не обеспечивает «растягивания» одного случайного бита в несколько. Если смешивать, например, случайный бит сначала с 0, а затем с 1, это будет давать двухбитовую последовательность, которая всегда будет принимать значения 01 или 10. Поскольку число разных значений составляет лишь 2, объем непредсказуемости сохраняется равным одному биту.

5.6. Другие факторы при выборе функции смешивания

Для локального применения AES обеспечивает некоторые преимущества за счёт того, что алгоритм многократно проверен, достаточно эффективно реализуется в программах, хорошо документирован и реализован во множестве программных (включая открытые) и аппаратных систем по всему миру. Семейство алгоритмов SHA* не так хорошо изучено и требует оной раз больше процессорных циклов для расчётов, но нет никаких причин считать эти алгоритмы несовершенными. SHA* и MD5 были разработаны на основе общего предшественника - алгоритма MD4. Доступны реализации этих алгоритмов с открытым исходным кодом [SHA*, MD4, MD5]. Были обнаружены некоторые признаки слабости алгоритмов MD4 и MD5. В частности, MD4 использует только три раунда (цикла вычислений) и существует две независимых уязвимости для двух первых и двух последних раундов. В выходе MD5 были обнаружены коллизии.

Алгоритм AES был выбран в результате открытого международного конкурса. Вместе с алгоритмами SHA* он был выбран агентством NSA[1] на основе критериев, которые в основном остаются секретными, как и DES. Несмотря на множество слухов и спекуляций многолетние исследования алгоритма DES показали, что участие NSA в его изменении, начатом компанией IBM, свелось в основном к его усилению. Не было публикаций о наличии скрытых или специально сохранённых недостатков алгоритма DES. Вполне вероятно, что замена агентством NSA алгоритма MD4 на SHA также обусловлена большей защищенностью нового алгоритма (возможно от угроз, которые ещё не известны всему сообществу криптографов).

Если размер входных данных не предсказуем, алгоритмы хэширования использовать удобнее, чем алгоритмы шифрования, поскольку в общем случае эти алгоритмы рассчитаны на работу с входными данными любого размера. Алгоритмам блочного шифрования в общем случае нужен дополнительный алгоритм заполнения, служащий для выравнивания размера входных данных по размеру блока шифрования.

На момент публикации этого документе применительно к алгоритмам AES, DES, SHA*, MD4 и MD5 не было известно о каких-либо патентных претензиях, за исключением безотзывной лицензии на свободное их использование во всем мире. Тем не менее, могут существовать не известные авторам патенты, а также патенты на реализации и применения или иные патенты, которые могли или могут быть выданы.

6. Генераторы псевдослучайных чисел

Когда затравка имеет достаточную энтропию, из входных данных, описанных в разделе 3, с возможным выравниванием и и смешиванием, описанными в разделах 4 и 5, можно алгоритмически получить большой объем криптографически сильных случайных значений. Такие алгоритмы не зависят от платформы и могут одинаково работать на любом компьютере. Для обеспечения безопасности алгоритма его входные данные и внутренние операции должны быть защищены от доступа нарушителей.

[1]US National Security Agency - Агентство национальной безопасности США.

Устройство такого алгоритма генерации псевдослучайных чисел, подобно устройству симметричных алгоритмов шифрования, является задачей для профессионалов. В параграфе 6.1 рассмотрены некоторые плохие идеи, которые были использованы в неудачных алгоритмах. Если вы не хотите разбираться с такими неудачными идеями, пропустите параграф 6.1 и прочтите оставшуюся часть этого раздела и раздел 7, где описаны некоторые стандартные алгоритмы генерации псевдослучайных чисел со ссылками на документы. Дополнительную информацию можно найти в разделе 7 и части 3 документа [X9.82].

6.1. Несколько неудачных идей

В нескольких следующих параграфах описаны некоторые идеи, которые могут представляться привлекательными, но ведёт к небезопасной генерации случайных чисел.

6.1.1. Ошибочность сложных манипуляций

Одним из подходов, который может дать искажённое представление о непредсказуемости, может быть очень сложный алгоритм (или отличный традиционный генератор псевдослучайных чисел с хорошими статистическими свойствами) и рассчитывать криптографические ключи с использованием ограниченных входных данных типа компьютерных часов в качестве значения «затравки» (seed). Нарушитель, который представляет приблизительное время запуска генератора сможет ограничиться проверкой сравнительно небольшого числа значения затравки, поскольку знает, что это системное время. Может быть создано значительное число псевдослучайных битов, но пространство поиска, с которым требуется работать нарушителю, будет достаточно мало.

Таким образом, очень сильные или сложные манипуляции с данными не помогут, если нарушитель может узнать о характере этих манипуляций и в качестве затравки применяется значение с недостаточной энтропией. Обычно в таких случаях нарушителю бывает достаточно проверить незначительное число результатов с разными затравками для преодоления защиты.

Другой стратегической ошибкой является допущение о том, что очень сложный алгоритм генерации псевдослучайных значений будет давать качественные случайные числа даже без теоретического обоснования и анализа алгоритма. Есть отличный пример такой ошибки в начале раздела 3 работы [KNUTH], где автор описывает сложный алгоритм. Предполагается, что реализация этого алгоритма в машинном коде будет настолько сложна, что человек, пытающийся разобраться просто не сможет без комментариев понять, что делает программа. К сожалению, реальное применение этого алгоритма показало, что он почти сразу же сходится к одному повторяющемуся значению в одном случае и к короткому циклу повторяющихся значений - в другом.

Сложные манипуляции при ограниченном диапазоне затравочных значений не только не помогут, но и при слепом выборе таких сложных манипуляция могут даже снизить уровень энтропии хороших затравок!

6.1.2. Ошибочность выбора из большой базы данных

Другим подходом, который может давать искажённое представление о непредсказуемости является случайный выбор значения из большой базы данных и предположение о том, что большой общий размер базы данных обеспечивает должную непредсказуемость. Например, типичный сервер USENET обрабатывает каждый день много мегабайтов информации [USENET_1, USENET_2]. Предположим, что случайное значение определяется выборкой 32 последовательных байтов данных со случайной установкой стартовой позиции для выбора. Это не будет давать 32*8 = 256 битов непредсказуемых данных. Даже если это будет часть фразы на человеческом языке, содержащем 2 или 3 непредсказуемых бита на байт, суммарный объем непредсказуемости будет примерно 32*2 = 64 бита. Для злоумышленника, имеющего доступ к той же базе данных Usenet, непредсказуемым будет только значение стартовой позиции, что вряд ли составит более нескольких десятков непредсказуемых битов.

Такие же аргументы применимы к выборке последовательностей из публично доступных дисков CD/DVD или любой другой открытой базы данных большого размера. Если злоумышленник имеет доступ к той же базе данных, такой «выбор из большого объёма данных» мало что даст. Однако при выборе данных из недоступного злоумышленнику источника типа системных буферов в многопользовательской среде, результат может оказаться неплохим.

6.1.3. Традиционные псевдослучайные последовательности

В этом параграфе рассмотрены традиционные источники детерминистических или псевдослучайных чисел. Процесс обычно начинается с некого «затравочного» значения seed и использует простые числовые или логические операции для создания последовательности значений. Отметим, что ни один из рассмотренных в этом параграфе способов не подходит для криптографических приложений. Описание приведено лишь с информационными целями.

В [KNUTH] приведено классическое описание псевдослучайных чисел. Описаны их применения для моделирования природных явлений, отбора проб, численного анализа, тестирования компьютерных программ, принятия решений, игр. Ни одно из этих приложений не имеет таких же характеристик, как описываемые здесь защитные приложения. Только в двух последних случаях нарушители могут пытаться подобрать случайные значения. Однако и в этих случаях у нарушителя есть обычно только одна попытка воспользоваться предсказанным значением. В случаях же подбора пароля или взлома шифра нарушитель обычно имеет много (иногда неограниченно) попыток подобрать нужное значение. Иногда нарушители могут сохранить сообщения для взлома и многократно повторять атаки на них. Предполагается также, что нарушители могут воспользоваться помощью компьютера.

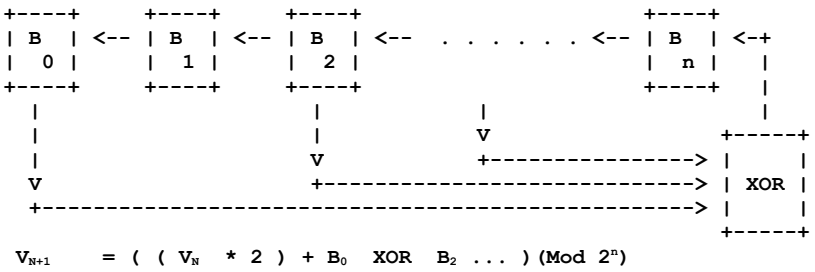
Для проверки «хаотичности» чисел Кнут предлагает различные методы, включая статистические и спектральные. Эти тесты проверяют разные аспекты типа наличия корреляций между разными частями «случайной» последовательности и распределения значений последовательности. Однако этим тестам удовлетворяет сохранённая случайная последовательность типа многократно изданных стандартных математических таблиц [CRC[1]]. Несмотря на прохождение всех предложенных Кнутом тестов, такая последовательность не подходит для криптографии, поскольку у нарушителя предполагается наличие копий всех опубликованных «случайных» последовательностей вкупе с возможностью определить источник данных и предсказать будущие значения.

[1]CRC Standard Mathematical Tables.

Типичным вариантом генерации псевдослучайных чисел является «линейный конгруэнтный генератор». Этот метод использует «модульную арифметику», где значение с номером N+1 рассчитывается на основе значения с номером N по формуле

$$V_{N+1} = (V_N * a + b) (Mod\ c)$$

Этот метод сильно связан с генерацией псевдослучайных чисел на основе линейного регистра сдвига, хорошо изученной с точки зрения криптографии [SHIFT*]. В таких генераторах группа битов подаётся с одной стороны регистра сдвига с использованием операции XOR (Исключающее-ИЛИ, двоичная сумма без переноса) для выбранных фиксированных отводов. Пример этого метода показан на рисунке.



$$V_{N+1} = ((V_N * 2) + B_0 \ XOR \ B_2 \ ...) (Mod\ 2^n)$$

Качество традиционных алгоритмов генерации псевдослучайных чисел измеряется статистическими тестами выдаваемых последовательностей. Аккуратно подобранные значения a, b, c и начальное значение V или обдуманный выбор отводов из регистра сдвига в описанном выше процессе позволяет получить отличную статистику на выходе.

Такие последовательности могут хорошо подходить для моделирования (например, методом Monte Carlo), пока последовательность ортогональная структуре исследуемого пространства. Но даже в таких случаях могут возникать проблемы, связанные с повторяющимися группами («узоры»). Однако такие последовательности явно не подходят для защитных приложений. Они полностью предсказуемы, если известно начальное состояние. В зависимости от типа генератора псевдослучайных чисел последовательность можно восстановить из наблюдения короткой её части [SCHNEIER, STERN]. Например, для показанного выше генератора можно определить V(n+1) по V(n). Фактически, было продемонстрировано, что в этих методах значение затравки (seed) можно определить даже по одному псевдослучайному биту на выходе.

Не только линейные конгруэнтные генераторы подвержены «взлому» - сейчас известны методы «взлома» полиномиальных конгруэнтных генераторов [KRAWCZYK].

6.2. Криптостойкие последовательности

В тех случаях, когда нужна генерация последовательности случайных значений, злоумышленник может узнать некоторые из них. В общем случае следует обеспечивать невозможность предсказания злоумышленником других значений на основании ставших ему известными случайных величин.

Корректным методом является использование качественной случайной «затравки» (strong random seed) для выполнения криптографически сильных операций на её основе [FERGUSON, SCHNEIER] без раскрытия полного состояния генератора в элементах последовательности. Если каждое значение в последовательности может быть рассчитано фиксированным способом на основе предыдущего значения, вся последовательность случайных значений будет скомпрометирована и любые будущие значения могут быть предсказаны. Такая ситуация может возникать, например, при создании каждого значения с помощью одной функции от предшествующих значений даже если эта функция является криптографически сильной, необратимой функцией создания отпечатка (дайджеста) сообщения.

> Отметим, что для достаточно быстрые методы генерации последовательности ключей могут служить тривиальной базой для систем защиты конфиденциальности. Если две стороны используют один метод генерации последовательности с одинаковой затравкой, они будут получать на выходе идентичные последовательности. Такая последовательность может применяться одной стороной для шифрования передаваемых данных (например, с помощью операции XOR), которые другая сторона легко расшифрует, используя свою последовательность значений и обратную операцию (в данном случае, снова XOR). Обычно это называют простым потоковым шифрованием.

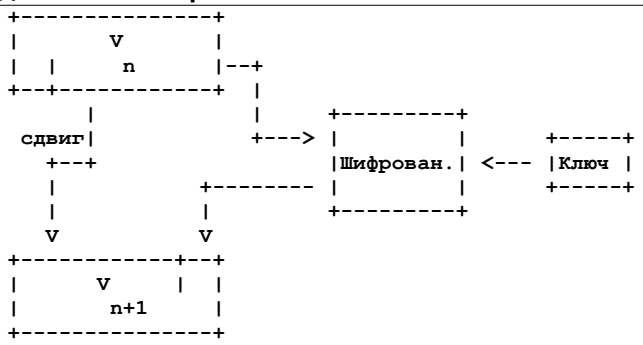
6.2.1. Последовательности OFB и CTR

Одним из вариантов генерации хороших последовательностей является хэширование конкатенации значения затравки с последовательными целыми числами или другой определённой последовательностью, а затем маскировать полученные значения с целью ограничения доступной злоумышленникам информации о состоянии генератора.

Может оказаться полезным использование алгоритма шифрования со случайным ключом и значением затравки применительно к последовательности целых чисел, как в режиме шифрования со счётчиком (CTR). Кроме того, можно на вход операции шифрования дополнительно подавать зашифрованный результат предыдущей итерации. Это является примером шифрования с обратной связью (OFB[1]) [MODES].

В показанном на рисунке примере используется сдвиг и маскирование для объединения выхода (обратная связь) с частью предыдущего входа. Такой частичной обратной связи следует избегать по причинам, рассмотренным ниже.

[1]Output feedback mode.



Отметим, что при использовании сдвига на один разряд этот метод похож на описанный в параграфе 6.1.3, но отличается в том, что обратная связь определяется сложной нелинейной функцией от всех битов вместо простой линейной или полиномиальной комбинации вывода из нескольких битовых позиций.

Donald W. Davies показал, что этот вариант частичной обратной связи со сдвигом существенно ослабляет алгоритм по сравнению с передачей на вход всех битов выхода. В частности, для алгоритма DES повторяющееся шифрования полного 64-битового значения будет давать ожидаемое повторение примерно через 2^{63} итерации. Возврат на вход меньшего числа битов (меньше 64, но больше 0) будет приводить к повтору через 2^{31} - 2^{32} итерации!

Предсказание значений последовательности по другим значениям, когда такая последовательность генерируется с помощью этих методов, эквивалентно взлому криптосистемы или обращению «необратимого» хэш-значения при доступности лишь части информации. Чем меньше информации раскрывается при каждой итерации, тем сложней будет для злоумышленника предсказание значений последовательности. Таким образом, лучшим вариантом будет использование единственного бита из каждого значения. Было показано, что в некоторых случаях это делает невозможным взлом системы даже при условии, что криптографическая компонента обратима и может быть взломана при доступности каждого сгенерированного значения.

6.2.2. Генератор Blum Blum Shub

На сегодняшний день наиболее сильное подтверждение качества имеет генератор Blum Blum Shub, названный по именам его изобретателей [BBS]. Этот генератор прост и работает на основе квадратичных вычетов. Единственным его недостатком является больший объем расчётов по сравнению с традиционными методами, описанными в параграфе6.1.3. Это не является существенной помехой, если генератор применяется не слишком часто (например, для создания сеансовых ключей).

Возьмём два больших простых числа (скажем, p и q), которые дают остаток 3 при делении на 4. Пусть n = p * q. Затем возьмём случайное число x, взаимно простое[1] с n. Ниже приведены уравнения для вычисления «затравки» (seed) и последующих значений.

$$s_0 = (x^2) \pmod n$$
$$s_{i+1} = (s_i^2) \pmod n$$

Будьте осторожны и берите небольшое число битов из каждого значения s. Наиболее безопасным будет использование единственного младшего бита. Если применяется не более

$$\log_2(\log_2(s_i))$$

младших битов, сложность верного предсказания последующих битов близка к сложности факторизации n.[2] Если начальное значение x хранится в секрете, значение n можно раскрыть.

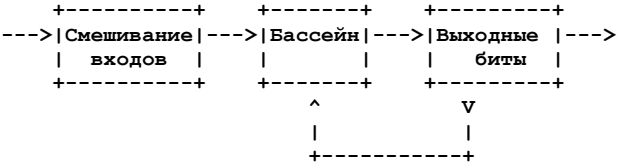
Интересной особенностью этого генератора является возможность прямого расчёта любого значения s. В частности

$$s_i = (s_0^{((2\hat{}i) \bmod ((p-1)*(q-1)))}) \pmod n$$

Это означает, что приложениям, в которых с помощью этого метода генерируется множество ключей, не требуется сохранять все такие ключи. Можно просто сохранить индексы ключей и восстановить нужный ключ по индексу, начальному значению s и значению n.

6.3. Методы сбора энтропии

Многие из современных источников псевдослучайных значений (типа описанных в параграфах 7.1.2 и 7.1.3) используют метод поддержки «бассейна» битов и предоставления операций для сильного смешивания входов с некоторой хаотичностью в этом бассейне, из которого потом извлекаются псевдослучайные биты, как показано на рисунке.



Биты для подачи в бассейн могут приходить от разного оборудования, из среды или от пользователя, как рассмотрено выше. При доступности хранилища состояние бассейна запоминается при перезагрузке системы и восстанавливается при старте.

Следует с осторожностью относиться к добавлению энтропии для обеспечения потребностей пользователя. См. также [RSA_BULL1] .

[1] Взаимно простые числа не имеют общих делителей, за исключением 1. *Прим. перев.*

[2] В https://www.rfc-editor.org/errata_search.php?eid=3105 отмечена ошибочность данного утверждения. *Прим. перев.*

7. Примеры и стандарты генерации хаотичности

В настоящее время имеется несколько общедоступных стандартов и множество примеров широкого развёртывания систем для генерации открытых ключей и создания других криптографических случайных величин. В некоторых (см. параграф 7.1) используются источники энтропии, в других (параграф 7.2) - генератор сильных псевдослучайных последовательностей, на входе которого предполагается случайная «затравка» (seed) или данные от источника энтропии.

7.1. Комплексные генераторы хаотичности

Ниже описаны три стандарта. Два наиболее старых стандарта используют алгоритм DES с размером блока 64 бита и ограниченным размером ключей. Оба варианта одинаково сильны и допускают замену функции смешивания на более сильную [DES]. Третий стандарт (более новый и сильный) основан на алгоритме SHA-1 [SHA*]. Кроме того, описаны широко распространённые современные генераторы случайных чисел операционных систем UNIX и Windows.

7.1.1. Рекомендации Министерства обороны США (US DoD) по выбору паролей

Министерство обороны США имеет свои рекомендации по выбору паролей [DoD]. Они предлагают использовать американский стандарт шифрования [DES] в режиме с обратной связью (Output Feedback Mode) [MODES], как описано ниже.

Использовать вектор инициализации, определяемый на основе комбинации

системных часов,

идентификатора системы,

идентификатора пользователя,

даты и времени.

Использовать ключ, определяемый на основе комбинации

регистров системных прерываний,

регистров состояния системы,

системных счётчиков.

В качестве открытых данных принимать созданное внешними средствами 64-битовое случайное значение в форме строки байтов ASCII из 8 символов, вводимых системным администратором.

На основе перечисленных выше данных пароль может быть рассчитан из 64 битов «шифротекста», создаваемого DES в 64-битовом режиме с обратной связью. Требуемое число битов может быть взято из этих 64 битов и «расширено» в произносимое слово, фразу или иной формат, если пароль должен запоминаться пользователем.

7.1.2. Устройство /dev/random

Некоторые версии операционной системы UNIX поддерживают встроенный в ядро генератор случайных числе. Часть таких генераторов использует события, собранные ядром в процессе его обычной работы.

Например, в некоторых версиях Linux генератор включает пул размером 512 байтов, организованных в 128 слов по 4 байта. Когда происходит событие (например, дисковое прерывание) время этого события вносится в пул с использованием операции XOR и пул перемешивается с помощью первообразного полинома степени 128. Сам пул трактуется, как кольцевой буфер и новые данные добавляются с помощью операции XOR (после полиномиального перемешивания).

Для каждых входных данных, добавляющих энтропию в пул, выполняется оценка содержащейся в них энтропии. Сам пул включает «аккумулятор», который оценивает общую энтропию пула.

Входные данные приходят из нескольких источников, перечисленных ниже. К сожалению, для серверных машин, где нет оператора, первый и третий источники данных не имеют смысла и темп добавления энтропии может быть более низким.

1. Клавиатурные прерывания. Моменты нажатия клавиш и скан-коды клавиатуры добавляются в пул. Реально добавляется энтропия за счёт изменения интервалов между нажатиями клавиш человеком.
2. Дисковые операции и другие прерывания. В системах, используемых людьми, дисковые операции обеспечивают трудно предсказуемую картину (однако не все дисковые устройства поддерживают достаточную точность информации о времени доступа).
3. Перемещение мыши. Временные метки и данные о перемещении мыши служат для добавления энтропии.

При возникновении потребности в случайных байтах содержимое пула хэшируется с помощью SHA-1 [SHA*] для получения выходного случайного значения. Если число требуемых данных превышает выходной размер SHA-1 (20 байтов), хэшированный вывод возвращается в пул и снова хэшируется для получения следующих 20 байтов. По мере выдачи байтов из пула оценка его общей энтропии соответствующим образом снижается.

Для обеспечения достаточной хаотичности при старте системы стандартные сценарии запуска и завершения работы обеспечивают запись имеющегося пула на диск при завершении работы и считывание этих данных при новом запуске системы.

Генератор имеет два пользовательских интерфейса. Устройство /dev/random возвращает байты из пула, но блокируется при достижении нулевого уровня энтропии в пуле. По мере добавления энтропии данные снова становятся доступными через /dev/random. Случайные значения из /dev/random подходят для генерации долговременных ключей, если в пуле имеется достаточно случайных битов или они могут быть добавлены в течение подходящего интервала времени.

Устройство /dev/urandom похоже на /dev/random, однако оно обеспечивает данные на выходе даже при нулевом уровне энтропии в пуле. Такие данные могут подходить для сеансовых ключей и других случаев, когда нет возможности ожидания достаточного уровня энтропии в пуле. Риск в результате продолжения использования данных при малом уровне энтропии в пуле невелик и заключается в том, что прошлые данные могут быть рассчитаны по текущим, если атакующий сможет обратить SHA-1. С учётом необратимости SHA-1 этот риск можно считать приемлемым.

Для получения случайных значений в Linux, Solaris и других системах семейства UNIX, где используется описанный выше код, приложения просто открывают устройство /dev/random или /dev/urandom и считывают из него требуемое число байтов.

(Код устройства Linux Random был написан Theodore Ts'o на основе генератора случайных чисел PGP 2.X и PGP 3.0 (он же PGP 5.0))

7.1.3. Windows CryptGenRandom

Компания Microsoft рекомендует пользователям широко распространённой операционной системы Windows применять вызовы генератора случайных чисел CryptGenRandom через интерфейс CryptAPI. При вызове задаётся библиотека криптопровайдера, указатель на буфер, в который вызывающее функцию приложение может поместить «энтропию», а криптопровайдер возвратит созданное случайное значение, и число желаемых случайных октетов.

Криптопровайдер Windows CryptAPI хранить переменную «затравки» (seed state variable) для каждого пользователя. При вызове CryptGenRandom значение этой переменной комбинируется с предоставленными при вызове случайными данными и пользовательскими данными (такими, как идентификаторы процесса и потока, системное время, системный счётчик, состояние памяти, данные о свободных кластерах на диске и хэшированный блок пользовательского окружения). Эти данные передаются функции SHA-1, а её результат служит «затравкой» потока RC4, который применяется для генерации запрошенных псевдослучайных данных и обновления «затравочной» переменной.

Пользователи Windows .NET могут предпочесть интерфейс метода RNGCryptoServiceProvider.GetBytes.

Дополнительную информацию можно найти в [WSC].

7.2. Генераторы, предполагающие источник энтропии

Описанные в трёх следующих параграфах генераторы псевдослучайных чисел предполагают предоставление им извне значения затравки с достаточным уровнем энтропии, с использованием которого генератор будет создавать сильную (см. параграф 6.2) последовательность значений.

7.2.1. Генерация псевдослучайных чисел X9.82

Комитет ANSI[1] X9F1 завершает подготовку стандарта для генерации случайных чисел, включающего как генераторы действительно случайных значений, так и генераторы псевдослучайных чисел. Стандарт включает множество генераторов, основанных на функциях хэширования, один из которых вероятно будет базироваться на конструкциях HMAC SHA [RFC2104]. Ниже описана предварительная версия такого генератора без рассмотрения множества необязательных функций [X9.82].

В последующих параграфах хэш-конструкция HMAC будет обозначаться просто HMAC и следует помнить, что для реализации должна быть выбрана конкретная стандартная функция SHA. В общем случае, если «сила» генерируемых псевдослучайных значений составляет N битов, выбранная функция SHA должна давать на выходе не менее N битов и требовать не менее N битов входной энтропии. Одна и та же хэш-функция должна использоваться на всех этапах процесса генерации случайных значений.

7.2.1.1. Обозначения

Ниже даны определения используемых в последующих параграфах обозначений.

hash_length
: выходной размер используемой хэш-функции.

input_entropy
: входная строка битов, служащая источником энтропии для генератора.

K
: битовая строка размера hash_length, являющаяся частью состояния генератора и обновляемая по крайней мере один раз за цикл генерации случайных битов.

V
: битовая строка размера hash_length, являющаяся частью состояния генератора и обновляемая при каждой генерации hash_length битов на выходе.

|
: указывает конкатенацию (склеивание).

7.2.1.2. Инициализация генератора

Установим во всех байтах V значение, содержащее нули во всех битах, кроме младшего.

Установим во всех байтах K нулевые значения, а затем

```
K = HMAC ( K, V | 0x00 | input_entropy )
V = HMAC ( K, V )
K = HMAC ( K, V | 0x01 | input_entropy )
V = HMAC ( K, V )
```

Примечание. Все алгоритмы SHA дают на выходе целое число байтов, поэтому размеры K и V в байтах будут целыми числами.

7.2.1.3. Генерация случайных битов

Когда нужно просто выходное значение, вызывается функция хэширования

[1]American National Standards Institute - Национальный институт стандартизации США.

```
V = HMAC ( K, V )
```

и берутся начальные биты V. Если число требуемых битов превышает размер V, создаётся пустая битовая строка temp и повторяется цикл

```
V = HMAC ( K, V )
temp = temp | V
```

пока размер temp не достигнет нужного значения. Далее требуемое число случайных битов берётся из начала переменной temp. Определение алгоритма запрещает запрашивать более 2^{35} битов.

После извлечения и сохранения выходных псевдослучайных битов до их возврата вызвавшему приложению нужно выполнить два дополнительных преобразования HMAC

```
K = HMAC ( K, V | 0x00 )
V = HMAC ( K, V )
```

7.2.2. Генерация ключей X9.17

ANSI задаёт для генерации последовательности ключей [X9.17] следующий метод:

s_0 - начальные 64 бита затравки;

g_n - последовательность сгенерированных 64-битовых ключей;

k - случайный ключ, зарезервированный для генерации данной последовательности ключей;

t - момент генерации ключа с максимально доступной точностью и разрешением (вплоть до 64 битов).

DES (K, Q) является результатом шифрования DES для значения Q с использованием ключа K.

Тогда

```
g_n   = DES ( k, DES ( k, t ) XOR s_n )
s_n+1 = DES ( k, DES ( k, t ) XOR  g_n )
```

Если g_n будет применяться в качестве ключа DES, каждый восьмой бит должен указывать чётность, но для расчёта следующего значения s нужно применять все 64 бита исходного значения g без их изменения.

7.2.3. Генерация псевдослучайных чисел DSS

Приложение 3 к стандарту NIST DSS[1] [DSS] определяет метод создания последовательности псевдослучайных 160-битовых значений для использования в качестве секретных ключей или похожих применений. Этот метод был изменён в [DSS_CN1] для создания псевдослучайных чисел общего назначения, как показано ниже.

```
t = 0x 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0
XKEY_0  = начальная затравка
For j = 0 to ...
    XVAL = ( XKEY_j  + необязательный ввод от пользователя ) (Mod 2^512)
    X_j  = G( t, XVAL )
    XKEY_j+1 = ( 1 + XKEY_j  + X_j ) (Mod 2^512)
```

Создаваемые таким способом псевдослучайные значения X являются 160-битовыми. В качестве указанной выше функции G могут применяться две функции, каждая из которых имеет два аргумента размером 160 и 512 битов.

Первая функция основана на SHA-1 и работает путём установки для 5 связующихся переменных, обозначенных H с нижним индексом в спецификации SHA-1, значений из первого параметра, поделённого на 5 частей. Затем выполняются этапы (a) - (e) из раздела 7 спецификации NIST SHA-1 для второго аргумента, как 512-битового блока данных. После этого значения связующихся переменных объединяются с помощью конкатенации в выходное значение G [SHA*].

В качестве дополнительного метода NIST определяет функцию G на основе многократного применения функции шифрования DES [DSS].

8. Примеры требуемой хаотичности

Ниже приведены два примера приблизительных расчётов хаотичности, требуемой для защиты. В первом рассматриваются умеренно защищённые пароли, а во втором - криптографический ключ с высоким уровнем стойкости.

В дополнение к этому отметим работы [ORMAN] и [RSA_BULL13], где приведена информация о размерах открытых ключей, которыми следует пользоваться при обмене симметричными ключами.

8.1. Создание пароля

Предположим, что пользовательские пароли меняются один раз в год и желательно обеспечить вероятность угадывания нарушителем пароля для той или иной учётной записи пользователя не более 1:1000. Предположим также, что передача пароля в систему является единственным способом проверить его корректность. В таком случае основным вопросом становится скорость передачи нарушителем подбираемых паролей в систему. Предположим, что система не позволяет вводить пароль чаше одного раза в течение 6 секунд. Это позволит нарушителю проверить 600 паролей в час, около 15 000 в сутки и около 5 миллионов в год. При наличии хоть какого-то контроля и мониторинга у нарушителя просто не будет возможности непрерывно подбирать пароли в течение года. Даже при однократной проверке журнальных файлов в течение месяца попытка подбора 500 000 паролей будет замечена и приведёт к смене пользовательских паролей с целью усложнения их подбора.

При вероятности угадывания пароля 1:1000 для того, чтобы 500 000 попыток не дали результата, пространство паролей должно иметь размер не менее 500 миллионов (приблизительно 2^{29}). Следовательно, требуется 29 битов непредсказуемости. Этого можно достигнуть, воспользовавшись рекомендациями US DoD по выбору входных данных для генерации паролей, предлагающими использовать 8 входных байтов со средней хаотичностью 5 битов в каждом (см. параграф 7.1). Используя список из 1000 слов, пароли можно задавать фразами из 3 слов, что даст 1000000000

[1]Digital Signature Standard - стандарт цифровой подписи.

вариантов. При использовании строчных и прописных букв, а также цифр пароли размером 6 символов вполне подойдут ($(26+10)^6$ = 2176782336 вариантов).

Для более защищённых паролей число битов требуется увеличить. Для снижения вероятности подбора в 1000 раз требуется расширить пространство паролей во столько же раз и для этого нужно добавить около 10 битов. Таким образом, для снижения вероятности угадывания пароля до одной миллионной в описанном выше сценарии будет требоваться 39 битов непредсказуемости и пароли, представляющие собой фразы в 4 слова из списка в 1000 слов или 8 алфавитно-цифровых символов. Для снижения вероятности угадывания до 10^{-9} потребуется 49 случайных битов, что может быть реализовано с использованием паролей в пять слов или 10 алфавитно-цифровых символов.

В реальных системах, естественно, имеется множество других факторов. Например, при удлинении и усложнении паролей их будет труднее запомнить и вырастет вероятность их записи пользователями, что создаёт дополнительный риск.

8.2. Стойкий криптографический ключ

Предположим, что требуется надёжно защищённый ключ для операций симметричного шифрования/дешифрования между двумя сторонами. Предположим также, что нарушителю доступен канал связи между сторонами и известен применяемый ими алгоритм. В рамках поля случайных возможностей нарушитель может пробовать разные ключи в надежде подобрать верный. Также будем считать, что перебор паролей (brute force) является единственным способом, доступным злоумышленнику.

8.2.1. Цена одной попытки

Сколько усилий потребуется на одну попытку подбора ключа? Для приложений с высоким уровнем защиты лучше предполагать незначительный объем усилий в расчёте на один ключ. Даже если представляется очевидным, что на проверку одного ключа затрачиваются десятки тысяч машинных тактов, может найтись тот или иной способ, позволяющий проверять разом очень большие блоки ключей, что существенно снизит затраты в расчёте на один ключ. По этой причине лучше полагаться на то, что для проверки одного ключа нужно не более нескольких сотен машинных тактов (нижняя граница этой оценки не очевидна, поскольку «параллельное» использование компьютеров для множества битов и недостатки алгоритма шифрования могут позволить проверять параллельно множество ключей и даже групп ключей; однако мы должны принять некое значение и будем надеяться, что достаточно для нашей гипотетической задачи выбран достаточно сильный алгоритм).

Если злоумышленнику доступен многопотоковый процессор или большая сеть рабочих станций, на сегодняшний день следует считать, что он способен выполнять не менее 10^{11} операций (машинных тактов) в секунду. Заглядывая на несколько лет вперёд, это значение следует увеличить по крайней мере на порядок. Таким образом, можно предположить у злоумышленника возможность проверки 10^{10} ключей в секунду, $3{,}6*10^{12}$ в час, $6*10^{14}$ в день или $2{,}4*10^{15}$ в месяц[1]. С учётом этого ключи должны содержать не менее 63 битов непредсказуемости для того, чтобы устоять против подбора в течение месяца. Но даже в этом случае можно предположить, что через несколько лет мотивированный и обладающий ресурсами злоумышленник сможет подобрать ключ в течение 2 недель - в среднем для этого нужно перебрать только половину из возможных ключей.

Эти вопросы подробно рассматриваются в работе «Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security: A Report by an Ad Hoc Group of Cryptographers and Computer Scientists» [KeyStudy], спонсированной Business Software Alliance. Там делается заключение, что в 1995 году для ключей с высоким уровнем защиты следовало использовать от 75 до 90 битов и, поскольку «стоимость» криптографии не зависит от размера ключа, рекомендовалось использовать 90 битов. Для приведения этих рекомендаций в соответствие с современностью просто добавьте 2/3 на каждый год в соответствии с законом Мура [MOORE]. Это показывает, что в 2004 году разумным размером ключей является 81 - 96 битов. Фактически, сегодня повсеместно расширяется использование ключей размером более 96 битов, типа 128-битовых (или более длинных) ключей AES, а также ключей triple-DES с эффективным размером 112 битов.

8.2.2. Атаки meet-in-the-middle

При одновременной доступности зашифрованных и открытых данных становятся возможными атаки типа meet-in-the-middle (встреча посередине), если структура алгоритма шифрования позволяет их организовать. При атаках с известными открытыми данными злоумышленник знает все или часть (возможно, некоторые стандартные поля заголовков или трейлеров) данных из шифруемых сообщений. При атаках по выбранным данным злоумышленник может спровоцировать шифрование тех или иных выбранных им данных (возможно, имитировав утечку «важных» данных, которые в результате будут переданы по шифрованному каналу).

В сильно упрощённом представлении атака meet-in-the-middle выглядит так - злоумышленник может полу-шифровать (half-encrypt) известные или выбранные данные со всеми возможными первыми полу-ключами, отсортировать результат, а затем полу-расшифровать (half-decrypt) их со всеми вторыми полу-ключами. Если будет найдено соответствие, можно собрать полный ключ из двух половинок и применять его для расшифровки других частей данного сообщения или других сообщений. В лучшем случае при такой атаке нарушитель сможет вдвое снизить показатель степени объёма работы, хотя при этом добавится очень большой, но примерно постоянный коэффициент. Таким образом, если такая атака считается возможной, для уровня 2004 года потребуется удвоение объёма хаотичности криптостойкого ключа как минимум до 192 битов (96*2), в соответствии с анализом [KeyStudy].

Этот объем хаотичности значительно превосходит рекомендации US DoD для генерации паролей и может потребовать продолжительного сбора информации от ввода данных пользователем, применения аппаратного источника случайных значений или иного источника хаотичности.

Атака meet-in-the-middle предполагает возможность разделения алгоритма шифрования на две части. Есть надежда, что современные алгоритмы не обладают таким недостатком, но возможны случаи, когда нет уверенности или даже не известен применяемый алгоритм. Даже если базовый алгоритм не подвержен атакам meet-in-the-middle, попытка создания более сильного алгоритма за счёт повторного его применения (или последовательного использования двух

[1] В числовых значениях и порядках (показателях степени) тут явные ошибки. Оставим их на совести авторов. *Прим. перев.*

разных алгоритмов) с другим ключом не повысит уровень защиты ожидаемым образом. Такие композитные алгоритмы будут уязвимы для атак meet-in-the-middle.

Для организации атаки meet-in-the-middle могут потребоваться аномально большие ресурсы, но они вполне могут оказаться доступными для спецслужб больших стран. В частности, все страны пытаются отслеживать трафик других стран.

8.2.3. Другие вопросы
В [KeyStudy] рассматриваются возможности взлома шифров с использованием специального оборудования а также оценки адекватности уровня защиты.

Отметим, что приведённые выше расчёты размеров ключей противоречивы и зависят от различных предположений о применяемых алгоритмах шифрования. В некоторых случаях профессионалы с хорошим пониманием методов взлома алгоритмов, знающие уровень криптостойкости используемого алгоритма могут обойтись ключами, размером менее половины от выведенного выше размера 192 бита.

Дополнительные примеры консервативных принципов проектирования можно найти в работе [FERGUSON].

9. Заключение
Генерация непредсказуемых «случайных» секретных значений для защиты является важной и сложной задачей.

Аппаратные методы сбора требуемой энтропии сравнительно просты. В частности, для случаев, когда требования к объёму и качеству не слишком высоки, вполне подойдут обычные компьютеры с использованием аудио-входов или времени выполнения дисковых операций.

Доступные методы расчётов позволяют принимать случайные значения низкого качества из многих источников или большой объем данных с низким качеством от одного источника для создания небольшого объёма высококачественного ключевого материала. При отсутствии аппаратных источников хаотичности может применяться множество источников хаотичности от программ и действий пользователя, но это следует делать с осторожностью. Однако в большинстве современных систем присутствуют аппаратные компоненты (типа дисковых устройств или аудио-входов), которые можно использовать для получения случайных значений с высоким качеством.

После сбора достаточного объёма (несколько сотен битов) высококачественного ключевого материала может использоваться множество доступных расчётных методов для генерации не предсказуемых расчётным путём значений на основе такой «затравки».

10. Вопросы безопасности
Весь документ посвящён рассмотрению методов и рекомендациям по генерации непредсказуемых «случайных» величин, используемых в качестве пароле, криптографических ключей, векторной инициализации, порядковых номеров и других компонент защиты.

11. Благодарности
Большое спасибо Paul Hoffman и John Kelsey за активное комментирование и Peter Gutmann за разрешение использовать материал из его статьи «Software Generation of Practically Strong Random Numbers».

Ниже в алфавитном порядке перечислены люди, внёсшие существенный вклад в разработку этого документа.

Steve Bellovin, Daniel Brown, Don Davis, Peter Gutmann, Tony Hansen, Sandy Harris, Paul Hoffman, Scott Hollenback, Russ Housley, Christian Huitema, John Kelsey, Mats Naslund, Damir Rajnovic.

Ниже в алфавитном порядке перечислены участники разработки RFC 1750 - предшественника этого документа.

David M. Balenson, Don T. Davis, Carl Ellison, Marc Horowitz, Christian Huitema, Charlie Kaufman, Steve Kent, Hal Murray, Neil Haller, Richard Pitkin, Tim Redmond, Doug Tygar.

Приложение A. Отличия от RFC 1750
1. Добавлены благодарности.
2. Добавлен параграф 5.3, посвящённый смешиванию блоков подстановки.
3. Добавлен параграф 3.3 и описанием кольцевого генератора в качестве источника хаотичности.
4. Добавлен алгоритм AES и серия алгоритмов SHA, дающие на выходе более 160 битов. Применение AES было рекомендовано взамен DES.
5. Добавлен параграф 6.3 с описанием методов на основе пула энтропии.
6. Добавлен параграф 7.2.3 с описанием методов генерации псевдослучайных чисел, приведённых в FIPS 186-2 (с изменением 1), параграф 7.2.1 с описанием методов из параграфа 7.12 X9.82, используемых в устройствах /dev/random операционных систем Linux и других UNIX-систем, а также параграф 7.1.3 с описанием методов генерации случайных чисел в Windows.
7. Добавлена ссылка на исследование «Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security», результаты которого опубликованы в январе 1996 года [KeyStudy], и [RFC1948].
8. Добавлены предостережения в части использования Diffie-Hellman в качестве функции смешивания.
9. Добавлены ссылки на работы X9.82 и статьи [TURBID] и [NASLUND].
10. Расширено рассмотрение min-entropy и Renyi entropy, а также приведена ссылка на книгу [LUBY].
11. Существенная реструктуризация, незначительное редактирование и обновление множества ссылок.

Литература

[AES] "Specification of the Advanced Encryption Standard (AES)", United States of America, US National Institute of Standards and Technology, FIPS 197, November 2001.

[ASYMMETRIC] Simmons, G., Ed., "Secure Communications and Asymmetric Cryptosystems", AAAS Selected Symposium 69, ISBN 0-86531-338-5, Westview Press, 1982.

[BBS] Blum, L., Blum, M., and M. Shub, "A Simple Unpredictable Pseudo-Random Number Generator", SIAM Journal on Computing, v. 15, n. 2, 1986.

[BRILLINGER] Brillinger, D., "Time Series: Data Analysis and Theory", Holden-Day, 1981.

[CRC] "C.R.C. Standard Mathematical Tables", Chemical Rubber Publishing Company.

[DAVIS] Davis, D., Ihaka, R., and P. Fenstermacher, "Cryptographic Randomness from Air Turbulence in Disk Drives", Advances in Cryptology - Crypto '94, Springer-Verlag Lecture Notes in Computer Science #839, 1984.

[DES] "Data Encryption Standard", US National Institute of Standards and Technology, FIPS 46-3, October 1999. Also, "Data Encryption Algorithm", American National Standards Institute, ANSI X3.92-1981. See also FIPS 112, "Password Usage", which includes FORTRAN code for performing DES.

[D-H] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[DNSSEC1] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[DNSSEC2] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[DNSSEC3] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

[DoD] "Password Management Guideline", United States of America, Department of Defense, Computer Security Center, CSC-STD-002-85, April 1885.

(см. также "Password Usage", FIPS 112, включающий CSC-STD-002-85 в качестве jlyjuj bp приложениq. FIPS 112 доступен по ссылке http://www.idl.nist.gov/fipspubs/fip112.htm).

[DSS] "Digital Signature Standard (DSS)", US National Institute of Standards and Technology, FIPS 186-2, January 2000.

[DSS_CN1] "Digital Signature Standard Change Notice 1", US National Institute of Standards and Technology, FIPS 186-2 Change Notice 1, 5, October 2001.

[FERGUSON] Ferguson, N. and B. Schneier, "Practical Cryptography", Wiley Publishing Inc., ISBN 047122894X, April 2003.

[GIFFORD] Gifford, D., "Natural Random Number", MIT/LCS/TM-371, September 1988.

[IEEE_802.11i] "Amendment to Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN Specific Requirements - Part 11: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: Medium Access Control (MAC) Security Enhancements", IEEE, January 2004.

[IPSEC] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

[Jakobsson] Jakobsson, M., Shriver, E., Hillyer, B., and A. Juels, "A practical secure random bit generator", Proceedings of the Fifth ACM Conference on Computer and Communications Security, 1998.

[KAUFMAN] Kaufman, C., Perlman, R., and M. Speciner, "Network Security: Private Communication in a Public World", Prentis Hall PTR, ISBN 0-13-046019-2, 2nd Edition 2002.

[KeyStudy] Blaze, M., Diffie, W., Riverst, R., Schneier, B. Shimomura, T., Thompson, E., and M. Weiner, "Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Commercial Security: A Report by an Ad Hoc Group of Cryptographers and Computer Scientists", January 1996. Доступен по ссылкам http://www.crypto.com/papers/keylength.txt и http://www.securitydocs.com/library/441.

[KNUTH] Knuth, D., "The Art of Computer Programming", Volume 2: Seminumerical Algorithms, Chapter 3: Random Numbers, Addison-Wesley Publishing Company, 3rd Edition, November 1997.

[KRAWCZYK] Krawczyk, H., "How to Predict Congruential Generators", Journal of Algorithms, V. 13, N. 4, December 1992.

[LUBY] Luby, M., "Pseudorandomness and Cryptographic Applications", Princeton University Press, ISBN 0691025460, 8 January 1996.

[MAIL_PEM1] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures", RFC 1421, February 1993.

[MAIL_PEM2] Kent, S., "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management", RFC 1422, February 1993.

[MAIL_PEM3] Balenson, D., "Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers", RFC 1423, February 1993.

[MAIL_PEM4] Kaliski, B., "Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services", RFC 1424, February 1993.

[MAIL_PGP1] Callas, J., Donnerhacke, L., Finney, H., and R. Thayer, "OpenPGP Message Format", RFC 2440, November 1998.

[MAIL_PGP2] Elkins, M., Del Torto, D., Levien, R., and T. Roessler, "MIME Security with OpenPGP", RFC 3156, August 2001.

[S/MIME] RFC 2632 - 2634

Ramsdell, B., "S/MIME Version 3 Certificate Handling", RFC 2632, June 1999.

Ramsdell, B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

Hoffman, P., "Enhanced Security Services for S/MIME", RFC 2634, June 1999.

[MD4] Rivest, R., "The MD4 Message-Digest Algorithm", RFC 1320, April 1992.

[MD5] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[MODES] "DES Modes of Operation", US National Institute of Standards and Technology, FIPS 81, December 1980. Also: "Data Encryption Algorithm - Modes of Operation", American National Standards Institute, ANSI X3.106-1983.

[MOORE] Закон Мура - экспоненциальный рост логической плотности кремниевых микросхем. В первоначальной формулировке Gordon Moore (1964) плотность возрастала каждый год, начиная с 1962, вдвое, но в конце 1970-х скорость роста упала до удвоения в течение каждый 18 месяцей и оставалась такой на момент публикации этого документа. См. "The New Hacker's Dictionary", Third Edition, MIT Press, ISBN 0-262-18178-9, Eric S. Raymond, 1996.

[NASLUND] Naslund, M. and A. Russell, "Extraction of Optimally Unbiased Bits from a Biased Source", IEEE Transactions on Information Theory. 46(3), May 2000.

[ORMAN] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.

[RFC1750] Eastlake 3rd, D., Crocker, S., and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.

[RFC1948] Bellovin, S., "Defending Against Sequence Number Attacks", RFC 1948, May 1996.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RSA_BULL1] "Suggestions for Random Number Generation in Software", RSA Laboratories Bulletin #1, January 1996.

[RSA_BULL13] Silverman, R., "A Cost-Based Security Analysis of Symmetric and Asymmetric Key Lengths", RSA Laboratories Bulletin #13, April 2000 (revised November 2001).

[SBOX1] Mister, S. and C. Adams, "Practical S-box Design", Selected Areas in Cryptography, 1996.

[SBOX2] Nyberg, K., "Perfect Non-linear S-boxes", Advances in Cryptography, Eurocrypt '91 Proceedings, Springer-Verland, 1991.

[SCHNEIER] Schneier, B., "Applied Cryptography: Protocols, Algorithms, and Source Code in C", 2nd Edition, John Wiley & Sons, 1996.

[SHANNON] Shannon, C., "The Mathematical Theory of Communication", University of Illinois Press, 1963. Originally from: Bell System Technical Journal, July and October, 1948.

[SHIFT1] Golub, S., "Shift Register Sequences", Aegean Park Press, Revised Edition, 1982.

[SHIFT2] Barker, W., "Cryptanalysis of Shift-Register Generated Stream Cypher Systems", Aegean Park Press, 1984.

[SHA] "Secure Hash Standard", US National Institute of Science and Technology, FIPS 180-2, 1 August 2002.

[SHA_RFC] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[SSH] Products of the SECSH Working Group, Works in Progress, 2005.

[STERN] Stern, J., "Secret Linear Congruential Generators are not Cryptographically Secure", Proc. IEEE STOC, 1987.

[TLS] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[TURBID] Denker, J., "High Entropy Symbol Generator", <http://www.av8n.com/turbid/paper/turbid.htm>, 2003.

[USENET_1] Kantor, B. and P. Lapsley, "Network News Transfer Protocol", RFC 977, February 1986.

[USENET_2] Barber, S., "Common NNTP Extensions", RFC 2980, October 2000.

[VON_NEUMANN] Von Nuemann, J., "Various techniques used in connection with random digits", Von Neumann's Collected Works, Vol. 5, Pergamon Press, 1963.

[WSC] Howard, M. and D. LeBlanc, "Writing Secure Code, Second Edition", Microsoft Press, ISBN 0735617228, December 2002.

[X9.17] "American National Standard for Financial Institution Key Management (Wholesale)", American Bankers Association, 1985.

[X9.82] "Random Number Generation", American National Standards Institute, ANSI X9F1, Work in Progress.

Part 1 - Overview and General Principles.

Part 2 - Non-Deterministic Random Bit Generators

Part 3 - Deterministic Random Bit Generators

Адреса авторов

Donald E. Eastlake 3rd

Motorola Laboratories

155 Beaver Street

Milford, MA 01757 USA

Phone: +1 508-786-7554 (w)

+1 508-634-2066 (h)

EMail: Donald.Eastlake@motorola.com

Jeffrey I. Schiller

MIT, Room E40-311

77 Massachusetts Avenue

Cambridge, MA 02139-4307 USA

Phone: +1 617-253-0161

EMail: jis@mit.edu

Steve Crocker

EMail: steve@stevecrocker.com

Перевод на русский язык

Николай Малых

nmalykh@protokols.ru

Полное заявление авторских прав

Интеллектуальная собственность

IETF не принимает какой-либо позиции в отношении действительности или объема каких-либо прав интеллектуальной собственности (Intellectual Property Rights или IPR) или иных прав, которые, как может быть заявлено, относятся к реализации или использованию описанной в этом документе технологии, или степени, в которой любая лицензия, по которой права могут или не могут быть доступны, не заявляется также применение каких-либо усилий для определения таких прав. Сведения о процедурах IETF в отношении прав в документах RFC можно найти в BCP 78 и BCP 79.

Копии раскрытия IPR, предоставленные секретариату IETF, и любые гарантии доступности лицензий, а также результаты попыток получить общую лицензию или право на использование таких прав собственности разработчиками или пользователями этой спецификации, можно получить из сетевого репозитория IETF IPR по ссылке http://www.ietf.org/ipr.

IETF предлагает любой заинтересованной стороне обратить внимание на авторские права, патенты или использование патентов, а также иные права собственности, которые могут потребоваться для реализации этого стандарта. Информацию следует направлять в IETF по адресу ietf-ipr@ietf.org.

Подтверждение

Финансирование функций RFC Editor обеспечено Internet Society.