11. Понятие надежности. Методология обоснования надежности криптографической защиты.

Надёжность = стойкость.

12. Автоматное определение шифра. Криптографические параметры узлов и блоков шифрующих автоматов.

13. Методы получения псевдослучайных последовательностей.

14. Генераторы псевдослучайных последовательностей и их свойства.

Лекуер предложил следующее обобщённое описание детерминированного ГПСЧ.

ГПСЧ описывается следующим набором параметров: (S, μ, f, U, g).

- S – множество состояний ГПСЧ;
- μ – распределение вероятностей выбора начального состояния s_0;
- f – функция перехода $f : S \to S, s_i = f(s_i - 1)$;
- U – множество выходных значений, как правило, $U = \{0, 1\}$;
- $g : S \to U$.

Периодом ГПСЧ называют число переходов, после которого состояния и, следовательно, генерируемые значения начинают повторяться.

Примеры ГПСЧ:

14.1. Линейный конгруэнтный метод

Линейный конгруэнтный метод – один из самых простых, но популярных. Каждое новое случайное число $X_i = (aX_i - 1 + c)\ mod\ m$, где a,

c и m – некоторые константы. Период не может быть больше m, но можно сделать его равным m при определённых условиях. Не используется в криптографии, т.к. значения предсказуемы. На основе этого генератора разрабатывались многие другие, например, полиномиальные конгруэнтные методы, но для них всех также было показано, что значения можно предсказать. Также стоит отметить, что биты генерируемых чисел зачастую не равновероятны, поэтому часто берут только некоторые битовые части получившегося числа. Однако их использование для моделирования вполне оправдано.

14.2. Запаздывающие генераторы Фибоначчи (аддитивные генераторы)

Это целое семейство генераторов, суть которых в том, что каждое следующее число зависит от нескольких уже сгенерированных сколько-то шагов назад. В общем виде аддитивные генераторы работают по следующей формуле: $X_i = (X_{i-a} + X_{i-b} + X_{i-c} + ... + X_{i-m})\ mod\ 2^n$. Соответственно, в качестве начального состояния необходимо задать массив из m значений. Если хорошо подобрать коэффициенты a, b, c и т.д., можно сделать так, чтобы период генератора был не меньше $2n - 1$. Например, распространённый фибоначчиев генератор работает по такой формуле:

$$X_k = \begin{cases} X_{k-a} - X_{k-b}, & \text{если } X_{k-a} \geq X_{k-b}, \\ X_{k-a} - X_{k-b} + 1, & \text{если } X_{k-a} < X_{k-b}, \end{cases}$$

В данном случае X_i – вещественные числа. Для работы генератору нужно держать в памяти $max(a, b)$ чисел, изначально они могут быть сгенерированы конгруэнтным методом. Период такого генератора может быть оценен как $T = (2^{max(a,b)} - 1) \cdot 2^l$, где l – число битов в мантиссе вещественного числа.

Фибоначчиевы генераторы создают последовательности, обладающие хорошими статистическими свойствами, причём для всех бит.

14.3. Регистры сдвига с линейной обратной связью (LFSR)

LFSR – сдвиговый регистр битовых слов, у которого значение входного (вдвигаемого) бита равно линейной булевой функции от значений остальных битов регистра до сдвига. Используется как ГПСЧ для генерации одиночных бит.

Булева функция, как правило, строится следующим образом. Каждый из L бит регистра умножается на некоторый коэффициент $c_i \in$

$\{0,1\}, i = \overline{1,L}$, причём $c_L = 1$. Дальше все результаты умножения складываются xor-ом. Таким образом, коэффициенты c_i формируют характеристический многочлен степени L над полем $GF(2)$ для данного регистра. Чтобы период генерации случайных чисел был максимальным $(2^L - 1)$, нужно, чтобы многочлен был примитивным.

Основные недостатки: медленно работает, многочлен можно определить, получив $2L$ бит с генератора.

14.4. Регистры сдвига с обобщённой обратной связью (GFSR)

GFSR – продолжение развития LFSR. Чтобы построить GFSR нужно:

1) Взять LFSR с примитивным характеристическим трёхчленом $x^p + x^{p-q} + 1$, где p и q – некоторые натуральные числа, $q < p$.

2) Сгенерировать неповторяющуюся последовательность максимальной длины $(2^p - 1)$, произвольно задав начальное состояние.

3) Составить таблицу из p строк, в которой каждая строка – сгенерированная на предыдущем шаге последовательность, циклически сдвинутая на некоторое фиксированное кол-во бит вправо.

4) Столбцы этой таблицы и есть значения нового генератора, их тоже $2^p - 1$ штук, но каждое из них уже представляет не бит, а p-битное слово.

14.5. Вихрь Мерсенна

Вихрь Мерсенна – очень популярная вариация GFSR. Используется во многих ЯП, например, в C++. Есть несколько вариаций, но самая распространённая – MT19937 ($2^{19937} - 1$ – период генератора).

Строится он похоже на GFSR:

1) Выстраивается таблица из 624 строк W_i, каждая по 32 бита.

2) Генерируется промежуточная строка tmp, равная первому (старшему) биту первой строки таблицы, конкатенированному с младшими 31 битами второй строки таблицы.

3) Полученная строка tmp объединяется xor-ом с 227-ой строкой таблицы и таким образом получается случайное 32-битное число.

4) Таблица сдвигается вверх удалением первой строки и добавлением полученного случайного числа в качестве последней строки.

Генератор получил такую популярность благодаря, во-первых, огромному периоду ($2^{19937} - 1$), во-вторых, прекрасным статистическим свойствам.

15. Блочные и поточные шифры.

Блочные шифры преобразуют открытый текст в шифртекст блоками по несколько байт (в зависимости от реализации шифра), поточные, как правило, преобразуют по одному байту.

Общее требование к блочным шифрам, сформированное Шенноном – принцип "перемешивания" – гласит, что незначительные изменения открытого текста должны приводить к значительным изменениям шифртекста.

Для выполнения этого требования блочные шифры строятся следующим образом. При зашифровании над каждым блоком открытого текста итерационно повторяются два типа преобразований: криптографически сложные преобразования частей блока и простые перестановки частей в пределах блока.

Примеры блочных шифров: AES, ГОСТ Р 34.12-2015.

16. Режимы использования блочных шифров.

Блочные шифры зашифровывают представленную информацию блоками фиксированной длины. Однако, если применять такие шифры непосредственно (что соответствует первому режиму далее), т.е. независимо шифровать каждый отдельный блок, шифр будет в какой-то степени уязвим к атаке "со словарём", т.к. блоки фиксированной длины могут повторяться в открытом тексте, и, набрав достаточное количество открытых и зашифрованных текстов, злоумышленник простым сопоставлением сможет раскрывать некоторые блоки. Чтобы такого не происходило, вводятся более сложные способы (режимы) использования блочных шифров.

Согласно ГОСТ Р 34.13-2015 "Режимы работы блочных шифров", существуют следующие режимы:

1) Режим простой замены (Electronic Codebook, ECB)

2) Режим гаммирования (Counter, CTR)

3) Режим гаммирования с обратной связью по выходу (Output Feedback, OFB)

4) Режим простой замены с зацеплением (Cipher Block Chaining, CBC)

5) Режим гаммирования с обратной связью по шифртексту (Cipher Feedback, CFB)

6) Режим выработки имитовставки (Message Authentication Code algorithm)

Для шифрования, очевидно, используются только первые 5 из них. Последний предназначен для выработки имитовставки (см. вопрос 17). При этом первый не рекомендуется к использованию по причинам, описанным в начале данного вопроса.

Если открытый текст по длине не кратен длине блока (или другому размеру, необходимому для работы соответствующего режима), он дополняется по одному из трёх алгоритмов, приведённых в том же ГОСТе.

Во всех режимах, кроме первого, используется синхропосылка IV (initialization vector, я полагаю) – двоичный вектор определённой длины. Она задаёт некоторые начальные значения для работы режимов и должна быть известна обеим сторонам для каждого шифруемого сообщения. При этом к ней предъявляются следующие требования:

Для режимов 4 и 5 синхропосылка должна выбираться случайным образом равновероятно из всех возможных двоичных векторов.

Для режима 2 синхропосылка должна быть разной для всех сообщений, зашифрованных на одном и том же ключе.

Для режима 3 значение синхропосылки должно быть либо случайным либо уникальным.

Режим простой замены работает тривиально. Для зашифрования применяем блочный алгоритм к каждому блоку открытого текста, для расшифрования – к шифртексту.

Для режима гаммирования сначала вырабатывается гамма – битовый вектор, равный длине открытого текста. Вырабатывается он с помощью шифрования блочным шифром значений счётчика, причём начальное значение задаётся синхропосылкой. Потом гамма накладывается на открытый текст с помощью xor. Расшифрование, соответственно, точно так же.

Для гаммирования с обратной связью по выходу используется регистр сдвига. Изначально он заполняется синхропосылкой. Для выработки гаммы левая часть регистра зашифровывается блочным шифром, из чего получается гамма для очередного блока открытого текста. Потом содержимое регистра сдвигается на длину этой гаммы, а сама гамма

записывается в регистр справа и процедура выработки повторяется для следующего блока. Гамма на блоки так же накладывается и "снимается" по xor.

Простая замена с зацеплением. Почти то же самое. Есть регистр сдвига длина больше, чем блок шифра. Сначала он заполняется синхропосылкой. Потом его старшая часть накладывается на открытый текст xorом, и результат зашифровывается. Регистр сдвигается влево на размер блока, а его младшая часть заполняется полученным зашифрованным блоком. Процедура повторяется.

То же самое, что и гаммирование с обратной связью по выходу, только в регистр справа дописывается не полученная на предыдущем шаге гамма, а результат её наложения на открытый текст, т.е. очередной блок шифртекста.

При выработке имитовставки в результате получается двоичный вектор фиксированной длины, в отличие от зашифрования. Для выработки имитовставки сначала из основного ключа генерируются два вспомогательных ключа, каждый по длине равен длине блока алгоритма шифрования. Для этого сначала на основном ключе зашифровывается нулевой вектор длиной как блок, потом с помощью сдвигов и xor с константой вырабатываются остальные. Дальше имитовставка вырабатывается просто: очередной блок зашифровывается блочным алгоритмом с основным ключом $\rightarrow$ результат зашифрования накладывается по xor на следующий блок открытого текста $\rightarrow$ полученный блок снова зашифровывается блочным алгоритмом с основным ключом. Исключением является последний блок. Для его вычисления берётся xor последнего блока открытого текста, предыдущего полученного блока имитовставки и один из вспомогательных ключей: первый, если блок полный, второй, если нет (блок дополняется). Результат этого xor также зашифровывается блочным алгоритмом на основном ключе.

17. Алгоритмы выработки имитовставки. Методы оценки имитозащищенности.

–

18. Режимы аутентифицированного шифрования. Современные стандартизированные решения.

–

19. Ключевые системы, методы распределения ключей.

–

20. Методы выработки производных ключей, принципы оценки качества производной ключевой информации.

–

21. Асимметричные криптографические схемы.

–

22. Гибридные схемы шифрования. Практические примеры реализации гибридных схем.

–

23. Электронная подпись, инфраструктура открытых ключей. Удостоверяющие центры. Методы обеспечения подлинности физических лиц.

–

24. **Атаки на криптографические алгоритмы: алгоритмические, алгебраические, статистические.**

–

25. **Свойства безопасности криптографических протоколов.**

–

26. **Методы и средства обеспечения заданных свойств безопасности криптографических протоколов.**

–

27. **Протоколы выработки общего ключа.**

–

28. **Протоколы распределения ключей.**

–

29. **Протоколы с разделением секрета.**

–

30. **Протоколы с подписью вслепую и протоколы электронного голосования.**

–

31. Протоколы семейства TLS, область их применения, методы оценки безопасности.

TLS (англ. transport layer security — Протокол защиты транспортного уровня) – криптографический протокол, обеспечивающий защищённую передачу данных между узлами в сети Интернет.

Использует асимметричное шифрование для аутентификации, симметричное шифрование для конфиденциальности и коды аутентичности сообщений для сохранения целостности сообщений.

Область применения: прежде всего, HTTPS, т.е. практически все веббраузеры и сайты соответственно. Однако также используется в электронной почте, чатах и IP-телефонии.

В общем все версии протокола действуют по следующему алгоритму:

1) Клиент подключается к серверу и отправляет ему поддерживаемую версию протокола TLS и поддерживаемые алгоритмы шифрования.

2) Сервер отвечает, какую версию и какое шифрование он готов использовать.

3) Сервер отправляет сертификат своего открытого ключа, если разделение общего ключа будет происходить через асимметричную криптографию.

4) Сервер отправляет свою часть Диффи-Хеллмана, если разделение общего ключа будет происходить через DH.

5) Клиент, в зависимости от способа разделения ключа, отправляет либо свою часть DH, либо зашифрованный на открытом ключе сервера секрет.

6) При необходимости клиент и сервер завершают выработку общего секрета, на основе которого будет осуществляться дальнейшее шифрование всего.

7) Клиент сообщает, что дальнейшая связь будет зашифрована и в зашифрованном виде отправляет хеш и имитовставку всех предыдущих сообщений.

8) Сервер всё проверяет и отправляет то же самое.

9) Клиент всё проверяет.

10) Если все проверки успешны, соединение считается установленным и дальше вся передача данных зашифровывается на основе разделённого секрета.

От версии к версии в протоколе меняются алгоритмы шифрования, цифровой подписи и выработки имитовставки, рекомендованные к использованию.

В частности, TLS 1.3 (последняя на данный момент версия) содержит алгоритмы из ГОСТ.

32. Протокол SSH, область его применения, реализуемые методы аутентифицированного удаленного доступа.

–

33. Протокол SESPAKE выработки общего ключа на основе пароля, область его применения, принципы обоснования сложности перебора паролей.

–

34. Протокол защищенного взаимодействия SPFIOT. Обоснование свойств безопасности, отличия от других протоколов.

–

35. Криптографические механизмы протокола IPSec. Обеспечиваемые им свойства безопасности.

–