и примитивного элемента a поля Галуа $GF(2^m)$. При $m=1$ код C_t называют примитивным, при $m>1$ не примитивным кодом [1].

Примитивные коды, как правило, имеют конструктивное расстояние $d=2t+1$ и исправляют как минимум t- кратные случайные ошибки. Не примитивные БЧХ-коды изучены слабо так как их параметры ведут себя достаточно хаотично.

В докладе приводятся результаты систематического исследования не примитивных БЧХ-кодов на примере кода C_5 длиной n =39 и m =12. Этот код имеет минимальное расстояние d =9, что больше конструктивного и, следовательно, способного исправлять четырехкратные ошибки. Приведен вариант схемы декодера адаптивной для работы с БЧХ-кодом данной длины. Близким по длине коду n =39 из примитивных кодов будет код n =31. Проведенный сравнительный анализ показал, что схема декодирования для n =31 адаптированная для работы с тройными ошибками будет сложнее, а алгоритм работы более медленным, чем для n =39. Но при этом примитивный код длины n =31 будет оставаться более высокоскоростным чем n =39. Найденное реальное значение d =9 и относительно простая схема декодирования, позволяет отнести данный БЧХ-код к классу перспективных в практическом плане кодов.

Литература

1. Липницкий, В.А. Норменное декодирование помехоустойчивых кодов и алгебраические уравнения / В.А. Липницкий, В.К. Конопелько. – Мн.: Издат. Центр БГУ, 2007. – 216 с.
2. MacWilliams, F.J. The Theory of Error-Correcting Codes / F.J. MacWilliams, N.J.A. Sloane // North-Holland Mathematical Library. – 1977. – Vol.16. – 762 p.

ИСПОЛЬЗОВАНИЕ ФИЛЬТРА БЛУМА ДЛЯ ОБНАРУЖЕНИЯ ПОВТОРОВ СИНХРОПОСЫЛОК

Семенов В. И.

НИИ ППМИ, Минск, Беларусь, e-mail: Semenov.vlad88@gmail.com

В настоящее время для защиты информации в современных информационных системах широко применяются поточные шифры и блочные шифры в режиме гаммирования. Необходимым условием стойкости является однократное использование синхропосылки при заданном ключе. Например, повтор синхропосыли встречается в MS Office версий до 2003 [1] и протоколе WEP [2].

Для предотвращения повтора синхропосылки предлагается использовать фильтр Блума [3]. Фильтр Блума *f* представляет собой хеш-таблицу *t* размера *m* с хеш функцией *h(s)* и позволяет проверить принадлежность элемента заданному множеству. Фильтр позволяет выполнять две операции на множестве *S={0,1}**:

add(s) – добавить элемент *s*: *t[h(s)]:=true*;

have(s) – принадлежность элемента *s*: *return t[h(s)]*.

Если *h(s)=h(s')*, то после добавления элемента *s*, функция *have(s')* ошибочно вернет положительный ответ. Хеш-функцию следует выбирать таким образом, чтобы

минимизировать вероятность ложноположительного ответа. Процедуру обнаружения повторов синхропосылки при шифровании сообщений можно представить в виде:

1. Для каждого нового ключа *k* построить новый фильтр *f(t,h)*;
2. Для каждого нового сообщения:
    a. Создать синхропосылку *s*;
    b. Если *f.have(s)*, то перейти к шагу 2.a;
    c. *f.add(s)*;
    d. Зашифровать сообщение, используя ключ *k* и синхропосылку *s*.

Данная процедура имеет несколько особенностей:

1. Фильтр строится отдельно для каждого ключа.
2. При длительном сеансе (многократном использовании ключа шифрования) либо при использовании неподходящей хеш-функции функция *have* будет всегда выдавать ложноположительные ответы;
3. Использование фильтра сокращает пространство синхропосылок, поэтому ключ шифрования может использоваться меньшее число раз.

Предложенная процедура исключает повтор синхропосылки. Параметрами фильтра являются размер таблицы *m* и вид хеш-функции *h*. Значения *h(s)* хеш-функции должны быть равномерно распределены на множестве индексов таблицы. Если для получения синхропосылки используется инкремент счетчика, то можно использовать хеш-функцию следующего вида: $h(s_1||\ldots||s_k) = s_1+\ldots+s_k$.

Литература

1. Hongjun, Wu. The Misuse of RC4 in Microsoft Word and Excel. / Wu. Hongjun // Institute for Infocomm Research – Singapore, 2005.
2. Walker J., Unsafe at any key size; An analysis of the WEP encapsulation. / J. Walker // Intel Corporation, 2000.
3. Bloom, Burton H. Space/time trade-offs in hash coding with allowable errors / Burton H. Bloom // Communications of the ACM – 1970 – 422–426 с.

ОБЗОР МЕТОДОВ ФАЗЗИНГА ДЛЯ ПОИСКА УЯЗВИМОСТЕЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Семенюк А. А.

БГУ, Минск, Беларусь, e-mail: gtv.blame@gmail.com

Фаззинг — метод поиска уязвимостей и ошибок программного обеспечения, основанный на передаче на вход системы заведомо некорректных данных. В процессе разработки программ, в частности на этапе отладки, её работа, как правило, проверяется на корректных входных данных, а работе с неправильными входными данными уделяется недостаточно внимания. Это может служить источником возникновения уязвимостей и угроз безопасности. Фаззинг позволяет автоматизировать поиск уязвимостей. Он подразделяется на несколько этапов:

- Генерация входных данных случайным образом либо при помощи модификации некоторого экземпляра входных данных.
- Передача данных на вход программы либо отправка их по сети.
- Отслеживание состояния программы, перехват исключительных ситуаций, отладка выполнения.