Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
“Национальный исследовательский университет
Высшая школа экономики”
МИЭМ им. А.Н. Тихонова
Департамент прикладной математики 10.05.01
“Компьютерная безопасность”

ОТЧЕТ

к индивидуальному проекту

по дисциплине

“Программирование алгоритмов защиты информации”

Выполнил:
Студент гр. СКБ201
Магамедов А.М.

Проверил:
Преподаватель
Нестеренко А.Ю.

Москва, 2024

Принцип работы

В чем в общем случае заключается задание? Для повышения устойчивости генератора (что бы, например, противник не мог просто решить систему линейных уравнений основанную на нескольких этапах генерации, и не получил изначальное состоянии регистра, которое может быть ключом), мы хотим добавить в функцию обратной связи нелинейный компонент, но не каждый полином при добавлении его результата в функцию обратной связи будет сохранять максимальный период генератора (что мы очень хотим иметь), который обычно обеспечен неприводимым многочленом, лежащим в основе функции съема. Поэтому мы переберем все полиномы 7 степени, проверим, какие из них при применении в функции обратной связи будут давать максимальный период. В определенный момент, я задумался, насколько хорошо полиномы будут работать для другой функции съема? Поэтому добавил еще одну аналогичную первой функцию обратной связи где используется другой многочлен для функции съема.

Описание функций

В коде реализованы следующие функции:

- **applyNonlinearFunction**: применяет заданную нелинейную функцию к входным данным. Каждый моном в векторе проверяется на присутствие переменных следующим образом - разыменовывая итератор, обращаемся по циклу к каждому биту, и если он присутствует, то соответствующий регистр объединяем с оставльными, с помощью AND, а результат комбинируется с использованием XOR.

- **puFunction**: реализует основную функцию обратной связи. Она включает:
  - XOR константы, 15-го, 11-го, 2-го, и 0-го битов регистра, согласно выбранному неприводимому многочлену для 16 битового регистра: $x^{16} + x^{12} + x^3 + x + 1$.
  - Извлечение первых 7 бит регистра для подачи на нелинейную функцию.
  - Применение нелинейной функции и XOR её результата в новый бит.
  - Обновление регистра сдвигом влево и вставкой нового бита на место 0-го.
  - Так как нас не интересует работа непосредственно генерации, то мы можем просто забыть про вывод.
- **puSecondPolFunction**: реализует дополнительную функцию обратной связи. Она включает:
  - XOR константы, 15-го, 13-го, 12-го, и 10-го битов регистра, согласно второму выбранному неприводимому многочлену для 16 битового регистра: $x^{16} + x^{14} + x^{13} + x^{11} + 1$.
  - Извлечение первых 7 бит регистра для подачи на нелинейную функцию.
  - Применение нелинейной функции и XOR её результата в новый бит.
  - Обновление регистра сдвигом влево и вставкой нового бита на место 0-го.

– Так как нас не интересует работа непосредственно генерации, то мы можем просто забыть про вывод.

- **processNonlinearFunction**: тестирует заданную нелинейную функцию на её периодичность. Если период последовательности достигает максимального значения (65535), результаты сохраняются. Начинается с задания случайного начального значения регистра. Если мы имеем нужный нам максимальный период, то перебор в любом случае вернет нас к тому же значению (в нашем случае - 43767), так как максимальный период регистра обеспечит пербеор всех его значений. Далее задаем вектор state - текущее состояние регистра, представленное в виде битовой строке и период текущей нелинейной функции. Затем запускаем цикл до $2^{register\_length}$. В каждой итерации применяем к текущему состоянию регистра функцию обратной связи, присваеваем периоду текущее состояние итератора, проверяем, если состояния зациклились, то выходим из цикла, иначе переписываем текущее состояние регистра новым. Если период максимальный, то делаем вывод следующим образом: создаем итератор по вектору текущей нелинейной функции, начинаем собирать строку вывода, создаем цикл по количеству переменных максимальному, и затем, разыменовывая итератор, обращаемся по циклу к каждому биту, и если он присутствует, то соответствующий номер переменной добавляем к строке вывода. Дальше все просто, через критическую секцию что бы потоки друг друга не переписывали делаем вывод найденных полиномов. Далее в функцию добавлен новый этап проверок, со второй функцией обрятной связи, где используется другая функция съема.

- **generateAndTestNonlinearFunctions**: перебирает все возможные комбинации нелинейных функций, оптимизируя процесс с использованием многопоточности. Результаты с максимальным периодом сохраняются в файл. Во первых, тут я использую большие числа из одной из библиоткек GNU, так как максимальное количество итераций с учетом того, что у нас должен присутствовать всегда моном $x_1x_2x_3x_4x_5x_6x_7$ это 2^{126}. Затем реализуем параллелизацию, в довольно простом стиле, делим общее количество возможных нелинейных полиномов на количество логических потоков ЭВМ, и в зависимости от номера потока начинаем пербор с соответствующего места. Далее мы начинаем цикл генерации нелинейных полиномов, добавляем член $x_1x_2x_3x_4x_5x_6x_7$, затем в цикле проверяем какие биты текущего числа в нем есть через другой цикл, и если бит присутствует, то после проверки на размер доавляем этот бит как моном. Соираем таким образом нелинейный полином, и тестируем его, так же выводим прогресс

Описание констант и структуры

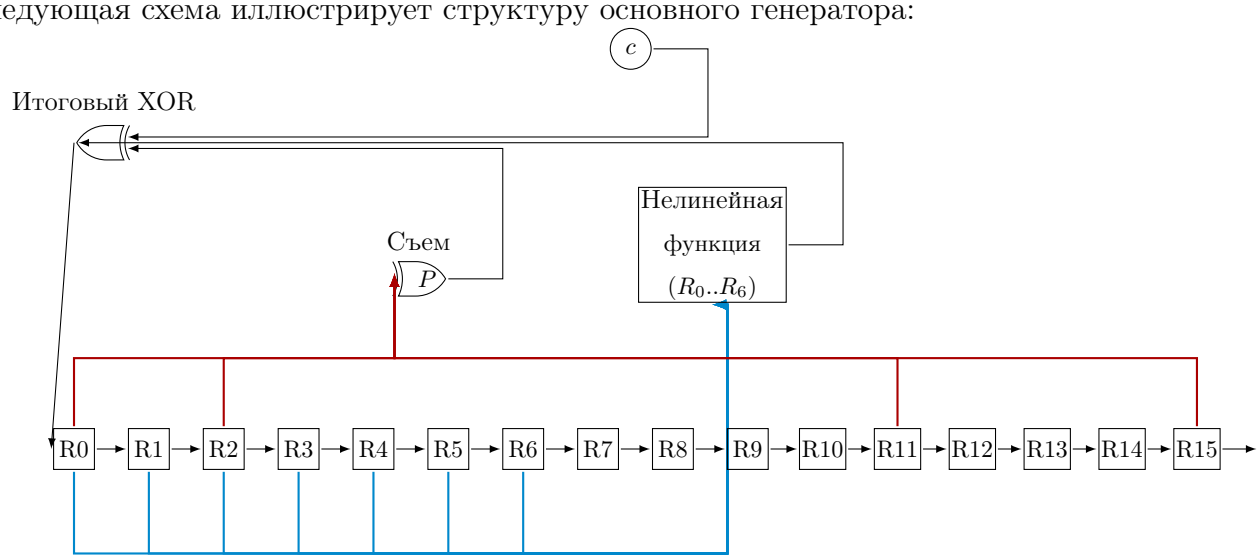
- **REGISTER_LENGTH**: задаёт длину регистра (16 бит).
- **VARIABLE_COUNT**: определяет максимальное количество переменных для нелинейной функции (7).
- **SIGMA**: константа.
- **NonlinearFunction**: структура(потому что в ранней реализации период указывался как поле этой структуры, позже было изменено, а структуру я оставил, в случае, если придется реализовать что

ниудь дополнительно), содержащая массив мономов(mons), представленных в виде битовых масок. Один вектор - один полином. Один элемент вектора - один моном в полиноме, представленный как 1, в месте где одна из 7 переменных присутствует, и 0 - если отсутствует.

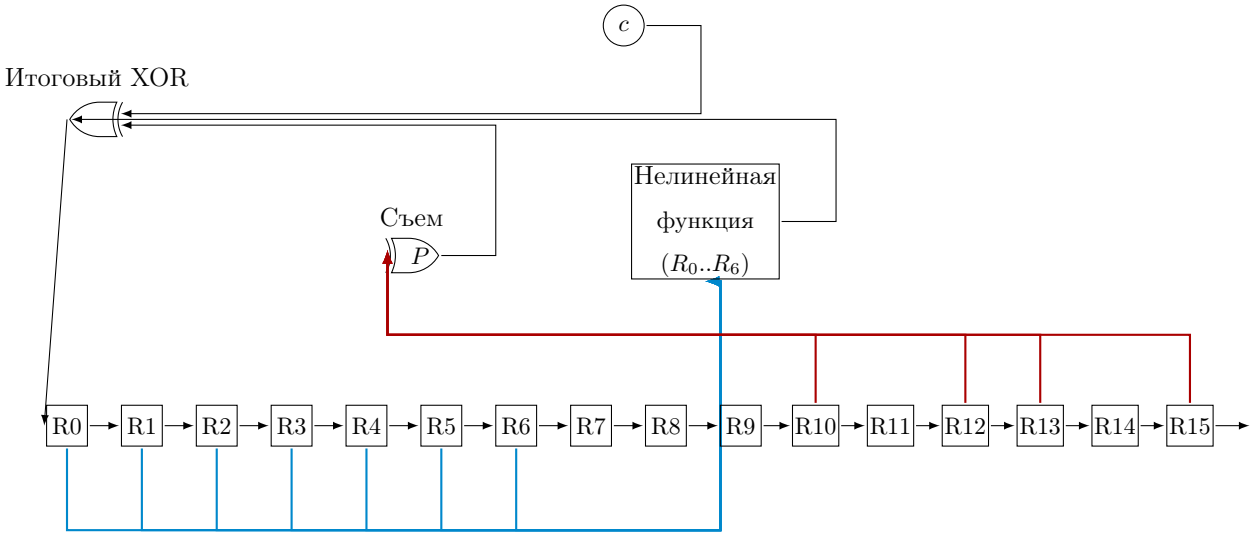
- **state**: текущее состояние регистра, представленное в виде битовой строки.

Визуализация

Следующая схема иллюстрирует структуру основного генератора:



Следующая схема иллюстрирует структуру дополнительного генератора:



Вывод

В отчете описаны функциональность каждой части кода и структура генератора. В процессе работы я сначала протестировал работу функции съема без нелинейного компонента, затем перешел к основной задаче. Так как количество полиномов для перебора просто огромно, я решил ускорить процесс и попросил товарища запустить пербеор на CHARISMe, оно там хорошо работало, но после 4 дней завершилось, не досчитав до конца(я забыл снять ограничение в 4 дня расчетов вообще). После этого я задумался, как уже писал выше, о том, будут ли найденные нелинейные полиномы так же эффективны для других функций съема и модифицировал код, для того что бы люой полином с периодом в 65535 проверялся еще и на второй функции съема. К сожалению, быстрая проверка на моей ЭВМ показала, что такие функции как минимум редки, и я зпустил подсчет уже модфифицированной программы на суперкомпьютер. Вообще, можно конечно попроовать другие варианты, например другие места съема для нелинейного полинома, но в связи с долгим подсчетом,

я решил пока что остановаться на этом.