I have an IDEA

IDEA. Начало

IDEA - (англ. International Data Encryption Algorithm, международный алгоритм шифрования данных) — симметричный блочный алгоритм шифрования данных.

Ранее известный как PES(proposed encryption standard), IPES(improved PES), после чего переименованный в IDEA

Создатель: Ascom, Xuejia Lai, James Massey

Создан: 1991 год

Опубликован: 1991 год

Размер ключа: 128 бит

Размер блока: 64 бит

Число раундов: 8.5

Тип: модификация сети Фейстеля

IDEA. Введение

Будут приведены дизайны в двух архитектурах *bit-parallel*(параллельно-двоичная) и *bit-serial*(последовательно-двоичная) их отличие заключается в способе передачи данных в отдельные моменты времени. *Bit-serial* передаёт один бит данных по одному каналу, *bit-parallel* сразу все значения по группе проводов.

На Xilinx Virtex XCV300-6 FPGA параллельная реализация обеспечивает шифрование равное **1166** Мб./сек. с частотой **82** МГц, в то время, как у последовательной реализации эти значения равны **600** Мб./сек. и **150** МГц. Оба варианта подходят для приложений в реальном времени(RTA — real-time applications) и высокоскоростных соединений(online high-speed networks). Также пропускная способность может быть увеличена за счёт повышения производительности вычислительной системы.

На плате XCV1000-6 характеристики у обоих подходов: **5,25** Gb/sec. и **2,4** Gb/sec. соответственно.

IDEA. Проблемс?

Хоть алгоритм и включает в себя только 16ти битные операции, его программные реализации до сих пор не могут достичь показатель шифрования требуемый для высокоскоростных сетей. Компания **Ascom** смогла добиться **0,37 * 10^6** шифрования за секунду или **23,53** Мб/сек. на *Пентиуме II 450* МГц.

Реализация алгоритма с использованием мультимедийных инструкций *Intel MMX* предложенных **Хелгером** достигла скорости **0,51 * 10^6** шифрования за секунду или **32,9** Мб/сек. на *Пентиуме II 233* МГц.

Ещё одна, приведённая ниже реализация достигает **2,3 * 10^6** шифр./сек. или **147,13** Мб/сек. реализованная на *Sun Enterprise E4500* с *12ю 400* МГц *Ultra-lii* процессором.

Все они не могут быть применены к ATM(Asynchronous Transfer Mode) сетям со скоростью 155 Мб/сек.

Короче, всё это плохо, но проблема решается.

IDEA. Решение

Проблемы со скоростью решаются аппаратной реализацией алгоритма и использования распараллеливания между операторами. В совокупности они дешевле, меньше жрут энергию и меньше занимают место.

Менсером был предложен макет(paper design) процессора **IDEA** достигающий скорости **528** Мб/сек. на 4ёх *XC4020XL* устройствах. Ништяк, да? Дальше больше...

Первая реализация **IDEA** на **VLSI** была предложена и разработана *Бонненбергом* в 1992 году и использовала *1,5* мкм *CMOS* технологию. Короче достигала она скорости шифрования порядка **44** Мб/сек.

Сюригер, 1994 г. **VINCI 177** Мб/сек. **VLSI** реализация на *1,2* мкм *CMOS*.

Вольтер, 1995 г. **355** Мб/сек. на *0,8* мкм CMOS.

Саломао, 1995 г. **424** Мб/сек. на одночипном *0,7* мкм *CMOS*.

IDEA. Решение

*Леонг, **500** Мб/сек. с последовательной реализацией алгоритма **IDEA** на **Xilinx Virtex XCV300-6 FPGA**,* который можно улучшить за счёт дополнительных ресурсов.

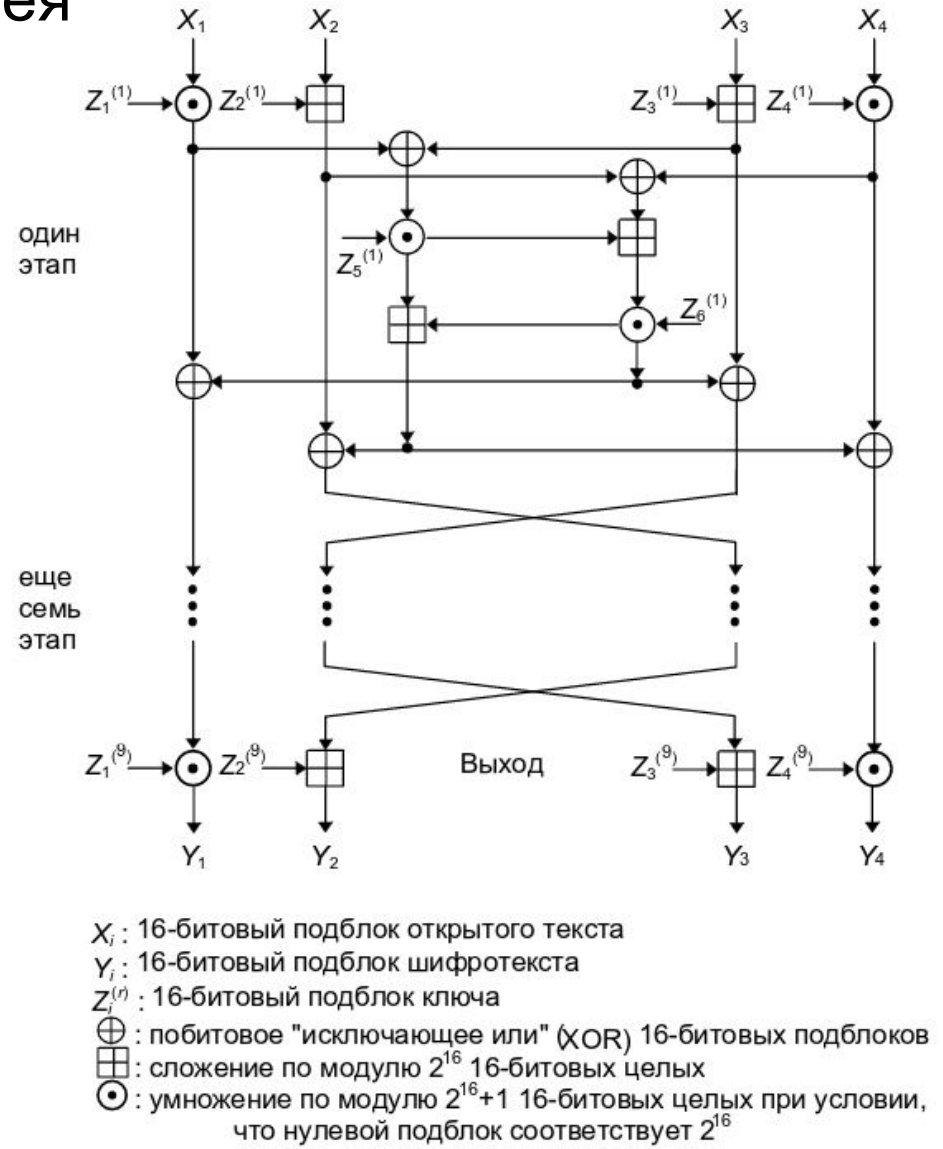
Голдштейн, какой то год, **1013** Мб/сек. на **PipeRench FPGA**.

Коммерческие реализации алгоритма носят название IDEACrypt kernel, разработанные **Ascom** со скоростью **720** Мб/сек. на технологии 0,25 мкм и последовавший за ним сопроцессор **IDEACrypt coprocessor** с пропускной способностью 300 Мб/сек.

IDEA. Идея

IDEA принадлежит классу криптосистем с закрытым ключом. На входе алгоритма 64 битный текст на выходе 64 битная криптограмма с применением 128 битного ключа.

В философии алгоритма — смешивание операций из самых разных алгебраических групп (XOR, сумма по модулю 2^{16}, и умножение по модулю простого числа Ферма $2^{16} + 1$) все они работают на 16 битных подблоках.



X_i : 16-битовый подблок открытого текста
Y_i : 16-битовый подблок шифротекста
$Z_i^{(r)}$: 16-битовый подблок ключа
⊕ : побитовое "исключающее или" (XOR) 16-битовых подблоков
⊞ : сложение по модулю 2^{16} 16-битовых целых
⊙ : умножение по модулю $2^{16}+1$ 16-битовых целых при условии, что нулевой подблок соответствует 2^{16}

Блок шифрования содержит нисходящую структуру из 8 идентичных блоков, называемых раундами, сопровождающимися полураундом или конечным преобразованием. На каждом раунде присутствуют операции XOR, сложение и умножение по модулю.

IDEA. Идея

IDEA позиционируется, как крипоустойчивый алгоритм потому что:

- В нём заложены примитивные операции трёх отдельных алгебраических групп 2^{16} элементов
- Умножение по модулю $2^{16} + 1$ даёт эффект статистической независимости между текстом и криптограммой
- Свойство раундов усложняет дифференциальные атаки
- Три операции($\oplus$, +, *) несовместимы в плане:
  никакие две из них не удовлетворяют дистрибутивному закону:
  $a * (b + c) \neq (a * b) + (a * c)$
  никакие две из них не удовлетворяют ассоциативному закону:
  $a + (b \oplus c) \neq (a + b) \oplus c$
- запутывание — шифрование зависит от ключа сложным и запутанным образом
- рассеяние — каждый бит незашифрованного текста влияет на каждый бит зашифрованного текста

IDEA. Алгоритм. Обзор алгоритма

Процесс шифрования:

- 64 битный текст делится на 4 подблока(X_1, …, X_4) по 16 бит каждый
- Далее каждый блок превращается в криптограммы по 16 бит(Y_1, …, Y_4)
- Вычисляются 52 16 битных подключа $Z_i^{(r)}$ из 128 битного секретного ключа, i и r — номера подключа и раунда, соответственно.
- Каждый раунд использует 6 подключей, а оставшиеся 4 используются в конечном преобразовании.

Процесс расшифрования схож с шифрованием за исключением, что ключи вычисляются иначе

IDEA. Алгоритм. Получение ключей. Шифрование

Процесс вычисления ключей(подключей) для шифрования (key-schedule):

- Упорядочим 52 подключа таким образом: $Z_1^{(1)}$, …, $Z_6^{(1)}$, …, $Z_1^{(8)}$, …, $Z_6^{(8)}$, $Z_1^{(9)}$, …, $Z_4^{(9)}$
- Процедура начинается с деления 128 битного секретного ключа Z на 8 16 битных блоков и назначения их прямо в первые 8 подключей.
- Далее Z поворачивается налево на 25 бит, разбитых на 8 блоков по 16 бит и снова назначаются следующим 8 подключам.
- Процесс продолжается, пока все 52 подключа не будут назначены.

Получается 128ми битный ключ разбивается на 8 подключей таким образом: $Z_1^{(1)}$, $Z_2^{(1)}$, …, $Z_6^{(1)}$, $Z_1^{(2)}$, $Z_2^{(2)}$ и соответствует следующему представлению:
1: $Z_1^{(1)}$, $Z_2^{(1)}$, $Z_3^{(1)}$, $Z_4^{(1)}$, $Z_5^{(1)}$, $Z_6^{(1)}$
2: $Z_1^{(2)}$, $Z_2^{(2)}$
После чего ключ Z циклически сдвигается влево на 25 бит и процедура деления повторяется, откуда получается уже такое представление:
1: $Z_1^{(1)}$, $Z_2^{(1)}$, $Z_3^{(1)}$, $Z_4^{(1)}$, $Z_5^{(1)}$, $Z_6^{(1)}$
2: $Z_1^{(2)}$, $Z_2^{(2)}$, $Z_3^{(2)}$, $Z_4^{(2)}$, $Z_5^{(2)}$, $Z_6^{(2)}$
3: $Z_1^{(3)}$, $Z_2^{(3)}$, $Z_3^{(3)}$, $Z_4^{(3)}$
и так далее, пока не заполнятся все 9 раундов и 52 ключа

IDEA. Алгоритм. Получение ключей. Расшифрование

Процесс вычисления ключей(подключей) для расшифрования иллюстрирует следующая таблица:

Табл. 13-4.
Подключи шифрования и дешифрирования IDEA

Этап	Подключи шифрования						Подключи дешифрирования					
1	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_4^{(1)}$	$Z_5^{(1)}$	$Z_6^{(1)}$	$Z_1^{(9)-1}$	$-Z_2^{(9)}$	$-Z_3^{(9)}$	$Z_4^{(9)-1}$	$Z_5^{(8)}$	$Z_6^{(8)}$
2	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$	$Z_4^{(2)}$	$Z_5^{(2)}$	$Z_6^{(2)}$	$Z_1^{(8)-1}$	$-Z_2^{(8)}$	$-Z_3^{(8)}$	$Z_4^{(8)-1}$	$Z_5^{(7)}$	$Z_6^{(7)}$
3	$Z_1^{(3)}$	$Z_2^{(3)}$	$Z_3^{(3)}$	$Z_4^{(3)}$	$Z_5^{(3)}$	$Z_6^{(3)}$	$Z_1^{(7)-1}$	$-Z_2^{(7)}$	$-Z_3^{(7)}$	$Z_4^{(7)-1}$	$Z_5^{(6)}$	$Z_6^{(6)}$
4	$Z_1^{(4)}$	$Z_2^{(4)}$	$Z_3^{(4)}$	$Z_4^{(4)}$	$Z_5^{(4)}$	$Z_6^{(4)}$	$Z_1^{(6)-1}$	$-Z_2^{(6)}$	$-Z_3^{(6)}$	$Z_4^{(6)-1}$	$Z_5^{(5)}$	$Z_6^{(5)}$
5	$Z_1^{(5)}$	$Z_2^{(5)}$	$Z_3^{(5)}$	$Z_4^{(5)}$	$Z_5^{(5)}$	$Z_6^{(5)}$	$Z_1^{(5)-1}$	$-Z_2^{(5)}$	$-Z_3^{(5)}$	$Z_4^{(5)-1}$	$Z_5^{(4)}$	$Z_6^{(4)}$
6	$Z_1^{(6)}$	$Z_2^{(6)}$	$Z_3^{(6)}$	$Z_4^{(6)}$	$Z_5^{(6)}$	$Z_6^{(6)}$	$Z_1^{(4)-1}$	$-Z_2^{(4)}$	$-Z_3^{(4)}$	$Z_4^{(4)-1}$	$Z_5^{(3)}$	$Z_6^{(3)}$
7	$Z_1^{(7)}$	$Z_2^{(7)}$	$Z_3^{(7)}$	$Z_4^{(7)}$	$Z_5^{(7)}$	$Z_6^{(7)}$	$Z_1^{(3)-1}$	$-Z_2^{(3)}$	$-Z_3^{(3)}$	$Z_4^{(3)-1}$	$Z_5^{(2)}$	$Z_6^{(2)}$
8	$Z_1^{(8)}$	$Z_2^{(8)}$	$Z_3^{(8)}$	$Z_4^{(8)}$	$Z_5^{(8)}$	$Z_6^{(8)}$	$Z_1^{(2)-1}$	$-Z_2^{(2)}$	$-Z_3^{(2)}$	$Z_4^{(2)-1}$	$Z_5^{(1)}$	$Z_6^{(1)}$
заключительное преобразование	$Z_1^{(9)}$	$Z_2^{(9)}$	$Z_3^{(9)}$	$Z_4^{(9)}$			$Z_1^{(1)-1}$	$-Z_2^{(1)}$	$-Z_3^{(1)}$	$Z_4^{(1)-1}$		

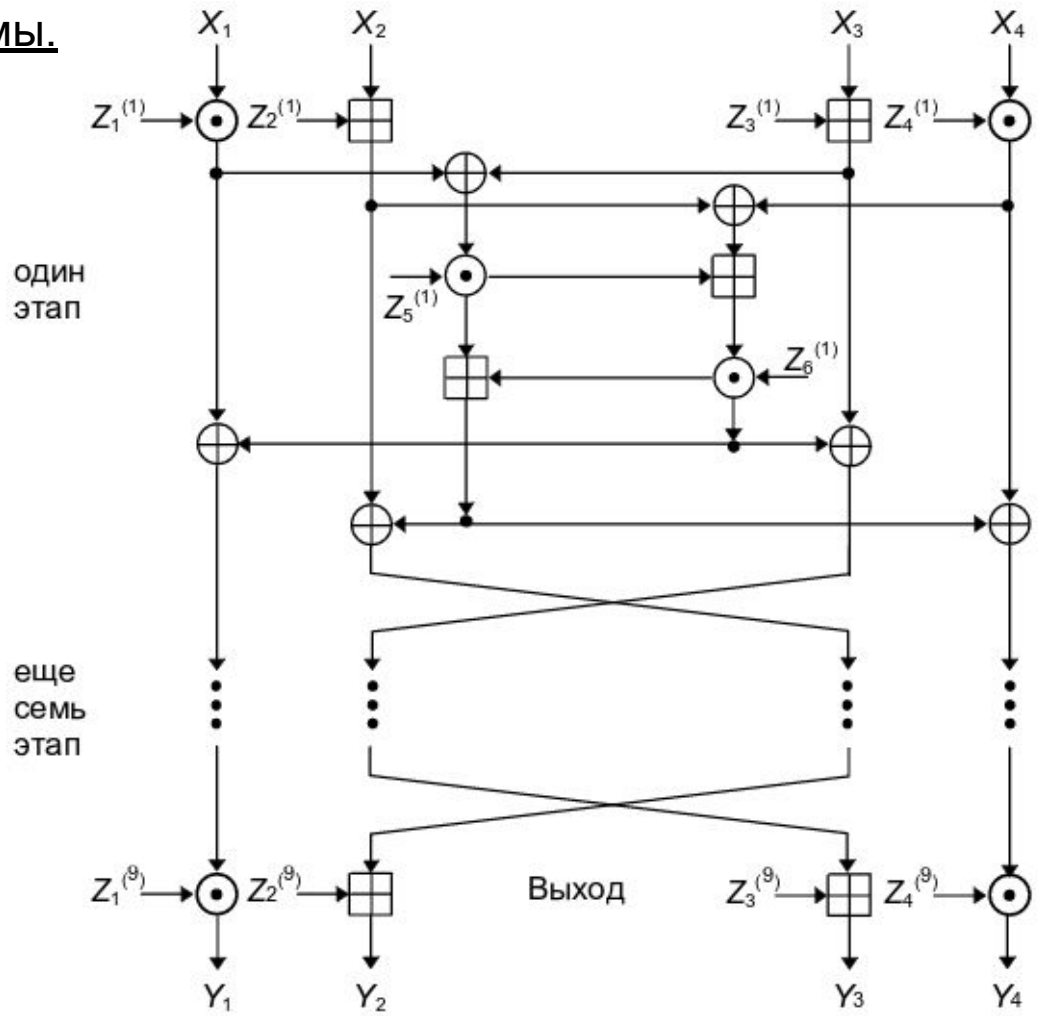
$Z_i^{(r)-1} * Z_i^{(r)} = 1 \pmod{2^{16}+1}$; $(-Z_i^{(r)}) + Z_i^{(r)} = 0 \pmod{2^{16}}$;

IDEA. Алгоритм. Шифрование

Процесс превращения текста в криптограммы.

Один раунд: $Y_1^{(1)} = X_1, \ldots, Y_4^{(1)} = X_4$

1) $Y_1^{(r)} = Y_1^{(r)} * Z_1^{(r)}$
2) $Y_2^{(r)} = Y_2^{(r)} + Z_2^{(r)}$
3) $Y_3^{(r)} = Y_3^{(r)} + Z_3^{(r)}$
4) $Y_4^{(r)} = Y_4^{(r)} * Z_4^{(r)}$
5) $A_1^{(r)} = Y_1^{(r)} \oplus Y_3^{(r)}$
6) $A_2^{(r)} = Y_2^{(r)} \oplus Y_4^{(r)}$
7) $A_1^{(r)} = A_1^{(r)} * Z_5^{(r)}$
8) $A_2^{(r)} = A_1^{(r)} + A_2^{(r)}$
9) $A_2^{(r)} = A_2^{(r)} * Z_6^{(r)}$
10) $A_1^{(r)} = A_1^{(r)} * A_2^{(r)}$
11) $Y_1^{(r+1)} = Y_1^{(r)} \oplus A_2^{(r)}$, $Y_2^{(r+1)} = Y_3^{(r)} \oplus A_2^{(r)}$
12) $Y_3^{(r+1)} = Y_2^{(r)} \oplus A_1^{(r)}$, $Y_4^{(r+1)} = Y_4^{(r)} \oplus A_1^{(r)}$
13) Следующий раунд

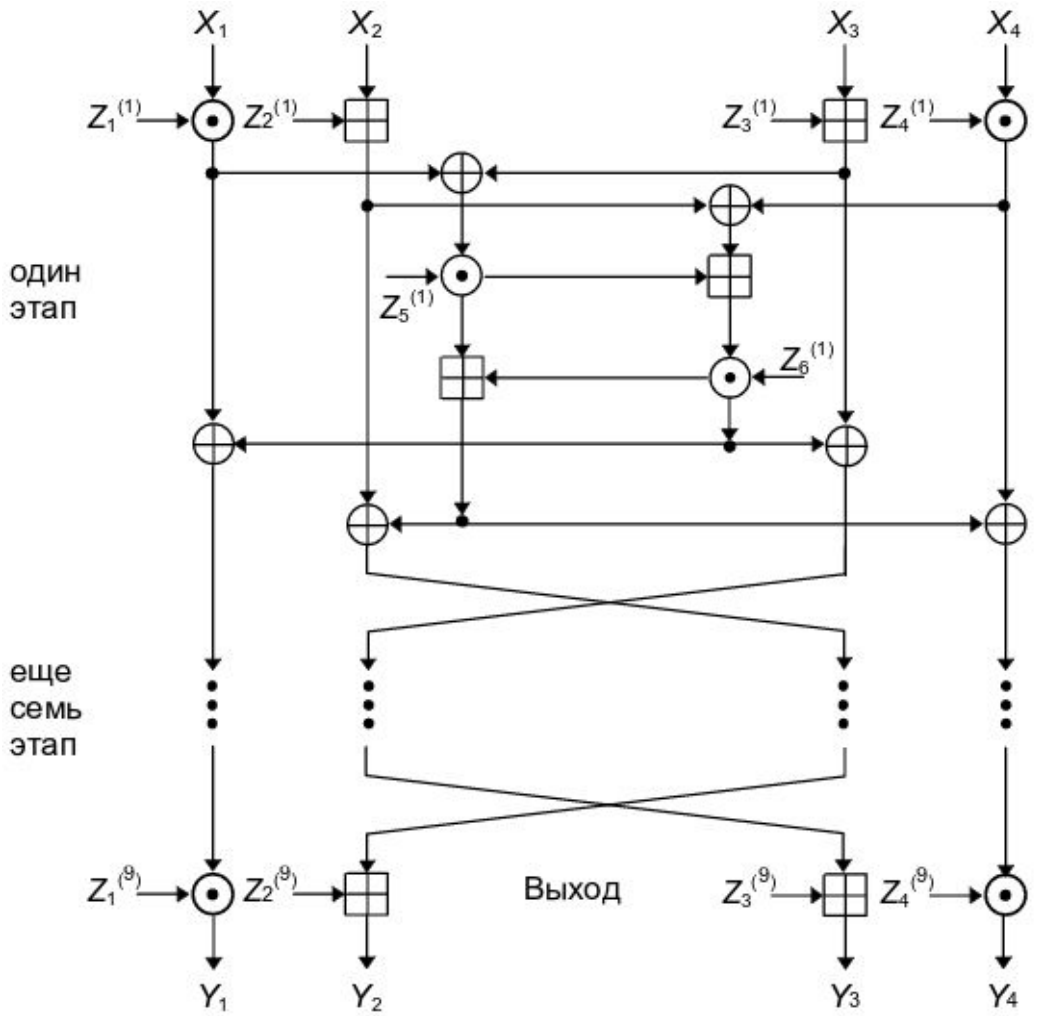


X_i : 16-битовый подблок открытого текста
Y_i : 16-битовый подблок шифротекста
$Z_i^{(r)}$: 16-битовый подблок ключа
$\oplus$: побитовое "исключающее или" (XOR) 16-битовых подблоков
$\boxplus$: сложение по модулю 2^{16} 16-битовых целых
$\odot$: умножение по модулю $2^{16}+1$ 16-битовых целых при условии, что нулевой подблок соответствует 2^{16}

IDEA. Алгоритм. Шифрование

Процесс превращения текста в криптограммы. Последний (9ый) раунд:

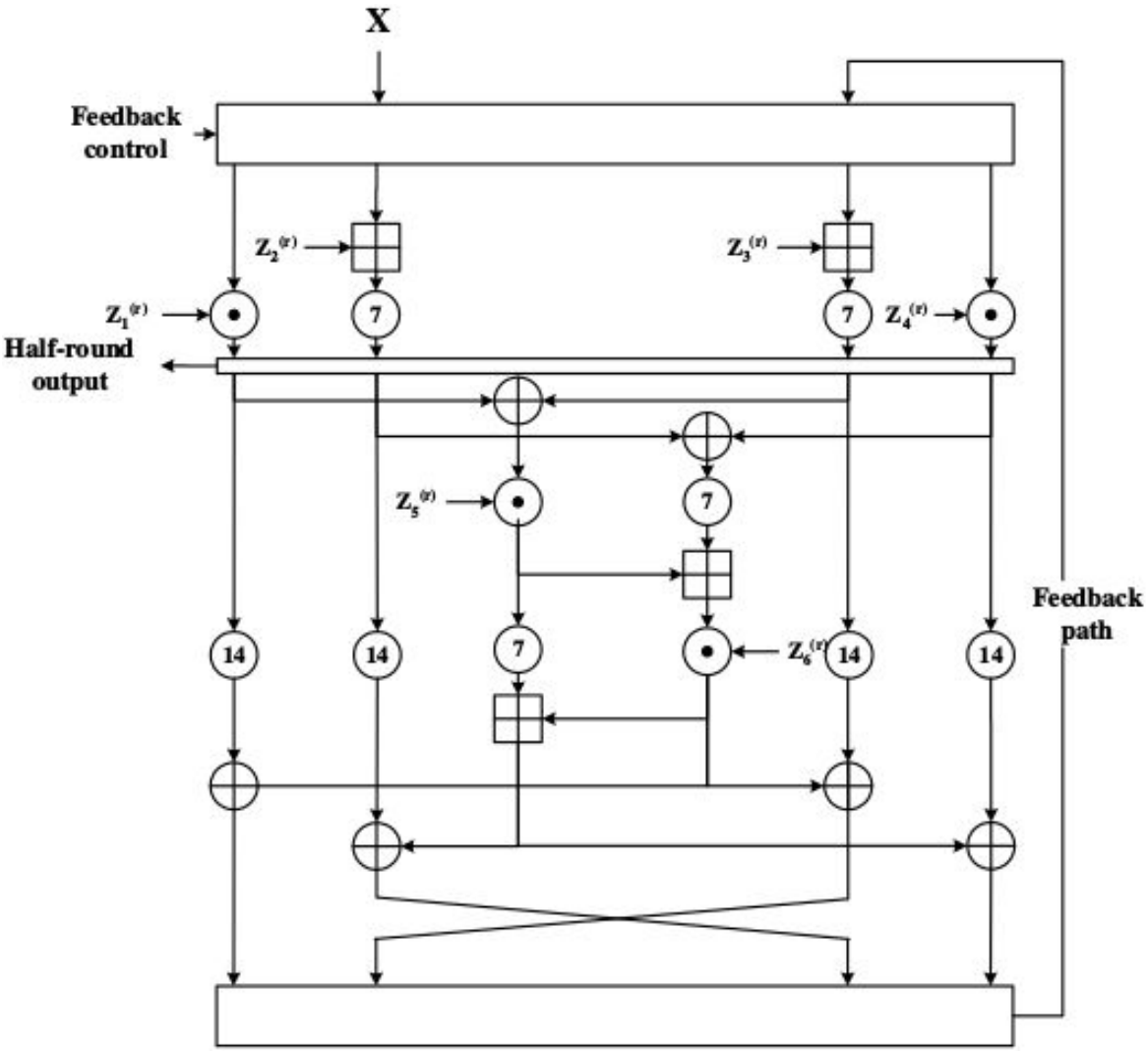
1) $Y_1^{(9)} = Y_1^{(9)} * Z_1^{(9)}$
2) $Y_2^{(9)} = Y_2^{(9)} + Z_2^{(9)}$
3) $Y_3^{(9)} = Y_3^{(9)} + Z_3^{(9)}$
4) $Y_4^{(9)} = Y_4^{(9)} * Z_4^{(9)}$
5) Конец.



X_i : 16-битовый подблок открытого текста
Y_i : 16-битовый подблок шифротекста
$Z_i^{(r)}$: 16-битовый подблок ключа
⊕ : побитовое "исключающее или" (XOR) 16-битовых подблоков
⊞ : сложение по модулю 2^{16} 16-битовых целых
⊙ : умножение по модулю $2^{16}+1$ 16-битовых целых при условии, что нулевой подблок соответствует 2^{16}

Bit-parallel. Параллельно-двоичная архитектура

Умножение по модулю $2^{16} + 1$ — это камень преткновения этого алгоритма. В каждом раунде используется по 4 таких операции, из-за этого правильно и хорошо реализованное умножение это ключевой момент, потому что это будет иметь последствия, как в размерах платы, так и в пропускной способности.



Bit-parallel. Параллельно-двоичная архитектура. Оператор умножение по модулю $2^{16} + 1$

В этом алгоритме передаются два аргумента x и y, y — это подключ, так как все ключи уже заранее меньше, то одно вычитание лишнее, убираем его.

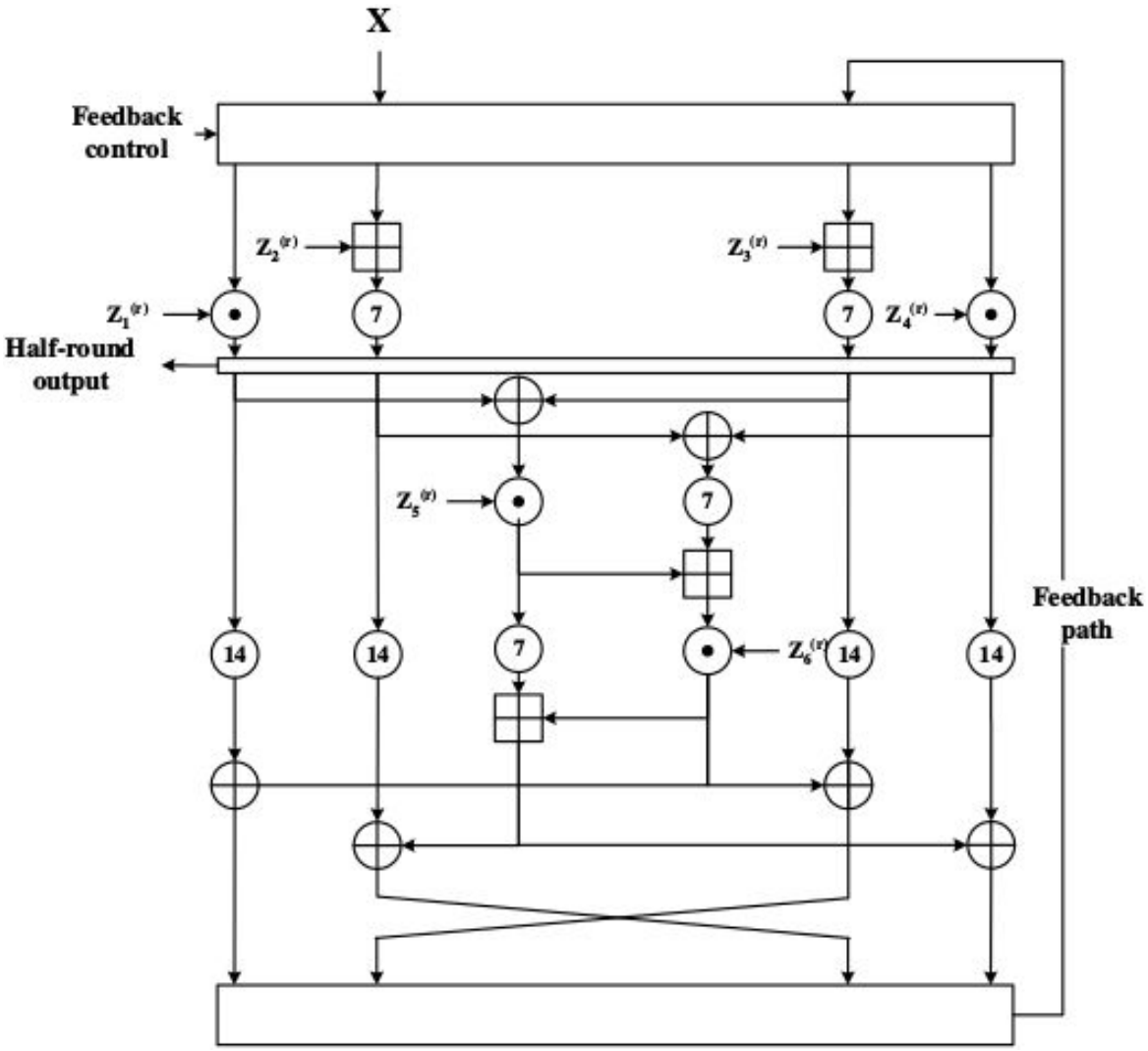
Далее, чтобы увеличить пропускную способность этого алгоритма добавляются дополонительные дорожки (pipeline stages?). В приведённом алгоритме строку 7 выполняет **Xilinx CORE generator** у которого задержка 4 цикла. И у умножения по модулю $2^{16} + 1$ задержка 7 циклов.

Реализация умножения по модулю $2^n + 1$

```
1   uint16 mulmod(uint16 x, uint16 y)
2   {
3       uint16 xd, yd, th, tl;
4       uint32 t;
5       xd = (x - 1) & 0xFFFF;
6       yd = (y - 1) & 0xFFFF;
7       t = (uint32) xd * yd + xd + yd + 1;
8       tl = t & 0xFFFF;
9       th = t >> 16;
10      return (tl - th) + (tl <= th);
11  }
```

Bit-parallel. Параллельно-двоичная архитектура

В связи с ограниченными аппаратными возможностями, каждый раунд выполняется на одном устройстве, но с разными ключами. Последнее преобразование, будучи не полным раундом, тоже выполняется на этом устройстве. Ключи хранятся внутри постоянной памяти (ПЗУ). Для большей эффективности свойство замков на этапах для временного выравнивания выполняет **Virtex SRL16E**.



Bit-parallel. Параллельно-двоичная архитектура

В одном раунде используется 3 умножения по модулю с задержкой в 7 циклов, в общей сложности задержка раунда получается 7 * 3 = 21, количество раундов 8,5 => 21 * 8 = 168, на конечном раунде, в преобразовании потребуется одно умножение по модулю => 168 + 7 = 175, в итоге такая реализация будет давать задержку в 175 циклов. Плата хавает 21 64 битных текстов(plaintext) за 21 * 9 = 189 циклов, что означает: (21/189) * 64 * f Мб/сек. с частотой f МГц.

Для примера дано 89 МГц, тогда такая реализация будет приносить в скорости шифрования 583 Мб/сек. с задержкой 2,134 мкс.

Bit-serial. Последовательно-двоичная архитектура

Представленная здесь модель, это слегонца улучшенная реализация классической модели путём перемещения и копирования регистров. Такое представление даёт пропускную способность шифрования равную 600 Мб/сек., что на 20% больше.

Характеризуется свойством, что операторы выполняют вычисления в побитовом режиме и их взаимодействие мультиплексированно во времени по одному проводу. Поток данных начинается либо с менее значимого бита, либо с более значимого, но последнее используется чаще из-за совместимости с двоичным дополнением (дополнительным кодом).

В типичном последовательно-двоичном представлении каждая переменная ассоциируется с контрольным сигналом, который выставляется высоким, только когда первый бит передаётся через ассоциированную шину данных. Для уменьшения размеров, доступ к контрольным сигналам может быть распределён среди переменных. Так как операторы в этой реализации обычно требуют первые биты их операндов, чтобы войти в оператор, в тот же момент времени цикла, соответствующие замки на этапах должны быть закрыты для выравнивания времени.

Xor и сумма по модулю 2^{16} могут быть запросто реализованы для последовательно-двоичной архитектуры. У этих двух операторов задержка равна 1 и они могут принимать последовательно последовательно-двоичные операнды. У умножения задержка равна 35 временным циклам, как и в параллельно-двоичной реализации замки на этапах и постоянные(const) реализует **SRL16E**. У постоянных соединёны поток ввода с потоком вывода для возможности циклического сдвига.

Bit-serial. Последовательно-двоичная архитектура. Умножение по модулю

Описанный алгоритм в параллельно-двоичной архитектуре в последовательно-двоичной будет мало эффективен, поэтому используется не изменённый $2^n + 1$

При умножении *N* * *N* бит получается 2*N* бит на выходе и требуется 2*N* циклов, чтобы получить результат. Пропускная способность операторов умножения в этой архитектуре ограничена, потому что минимальный диапазон между последовательными умножителями должен быть хотя бы 2*N* циклов. В алгоритме **IDEA** один из сомножителей каждого умножения по модулю это подключ, а он всегда одного размера, можно принять его за постоянную величину.

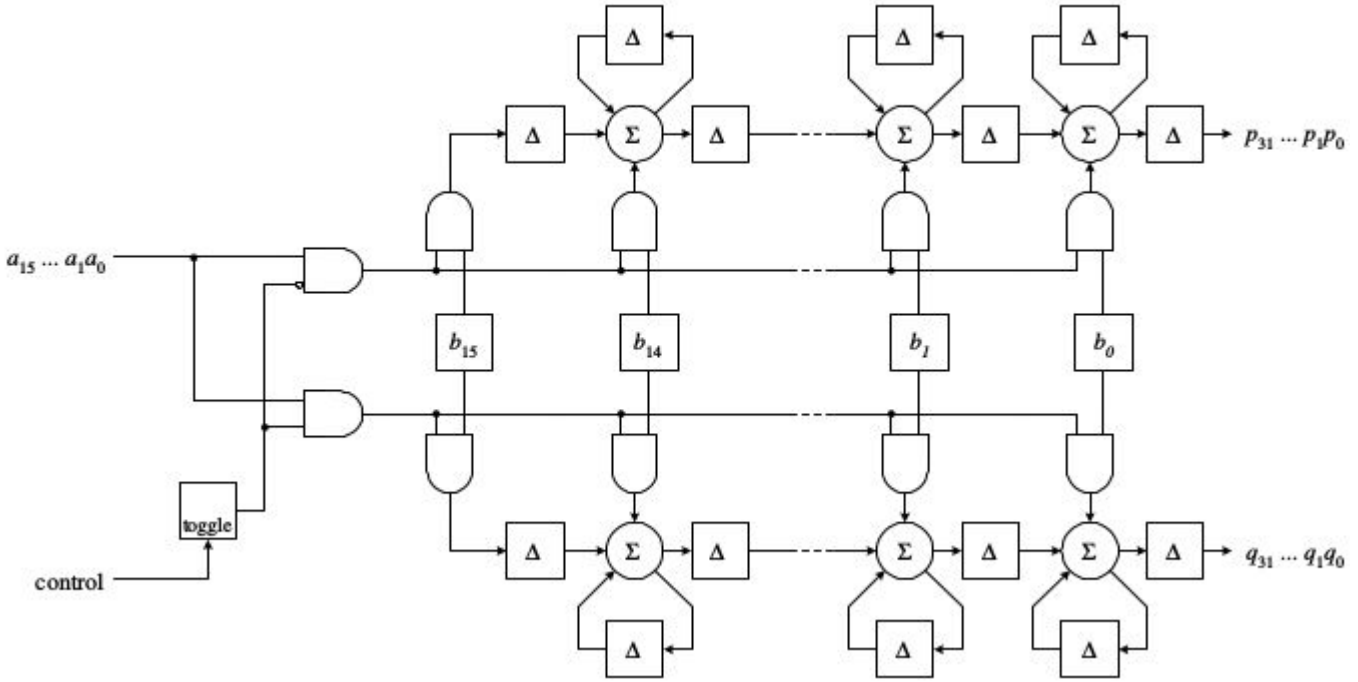
```
1   uint16 mulmod(uint16 x, uint16 y)
2   {
3       uint16 xd, yd, th, tl;
4       uint32 t;
5       xd = (x - 1) & 0xFFFF;
6       yd = (y - 1) & 0xFFFF;
7       t = (uint32) xd * yd + xd + yd + 1;
8       tl = t & 0xFFFF;
9       th = t >> 16;
10      return (tl - th) + (tl <= th);
11  }
```

Bit-serial. Последовательно-двоичная архитектура. Умножение по модулю

```
1   uint16 mulmod(uint16 x, uint16 y)
2   {
3       uint16 xd, yd, th, tl;
4       uint32 t;
5       xd = (x - 1) & 0xFFFF;
6       yd = (y - 1) & 0xFFFF;
7       t = (uint32) xd * yd + xd + yd + 1;
8       tl = t & 0xFFFF;
9       th = t >> 16;
10      return (tl - th) + (tl <= th);
11  }
```

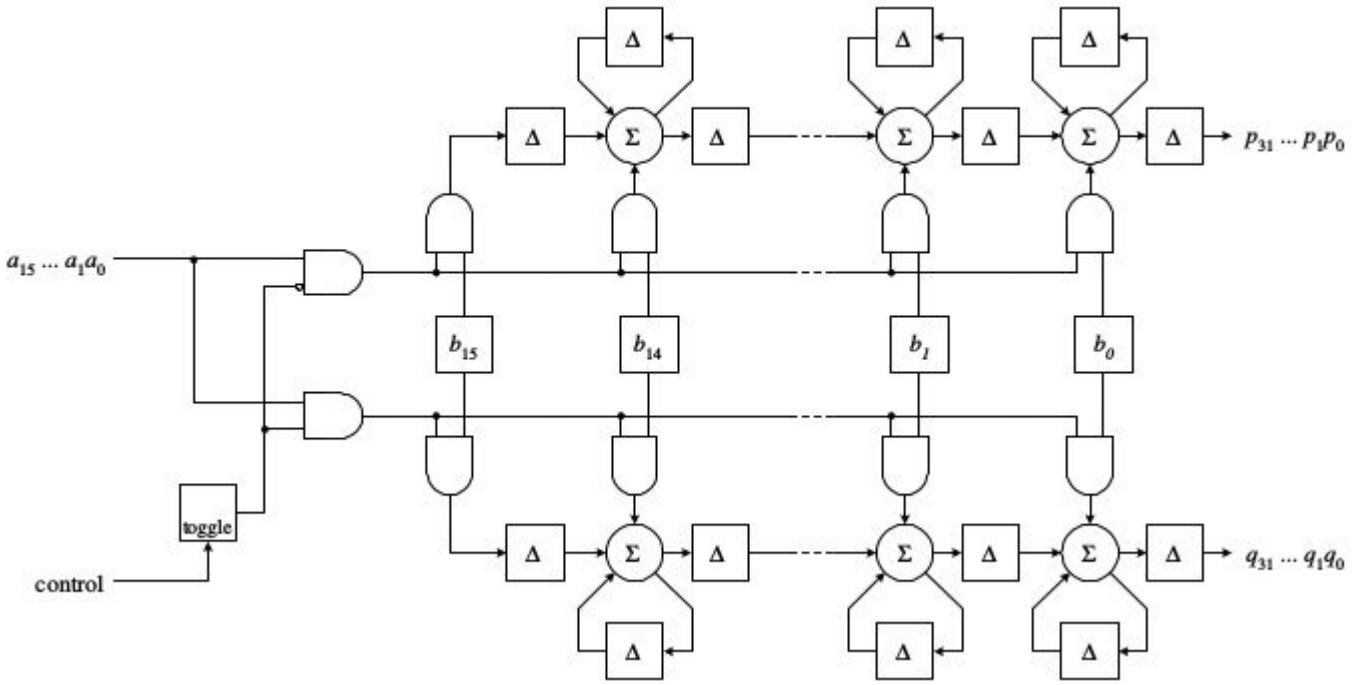
Представим, что промежуточное значение *t* в умножении разделено на 2 части(строки 7-9), это старшая и младшая части этого числа, они понадобятся нам далее, желательно, чтобы дизайн позволял считать старшие и младшие части независимо, не запрещяя чтобы все входные, выходные и промежуточные переменные оператора были 16 бит в длину. Используя эту схему и отразив аппаратное воплощение, как будет показано далее, пропускная способность оператора умножения может быть удвоена.

Bit-serial. Последовательно-двоичная архитектура. Умножение по модулю



Была разработана изменённая схема последовательного умножителя Лиона которая решила эту проблему. Для того, чтобы сгенерировать 2 16 битных результата за 16 циклов пропускная способность должна быть удвоена. Это достигается путём дублирования устройства умножения, как показано на картинке выше. Регистры в которых находятся константы доступны для двух частей оператора. Выходные значения *p* и *q* соответствуют двум последовательным умножениям, где у двух 32ух битных переменных разница во времени 16 циклов. [Контрольный сигнал, который был выше одного цикла, до того, как наименее значимый бит вошёл в модуль, захватывает контроль над регистром?]. Вектор из входных значений $a_{n-1} \ldots a_0$ из-за этого попеременно перенаправляется на два провода, ведущих в умножители.

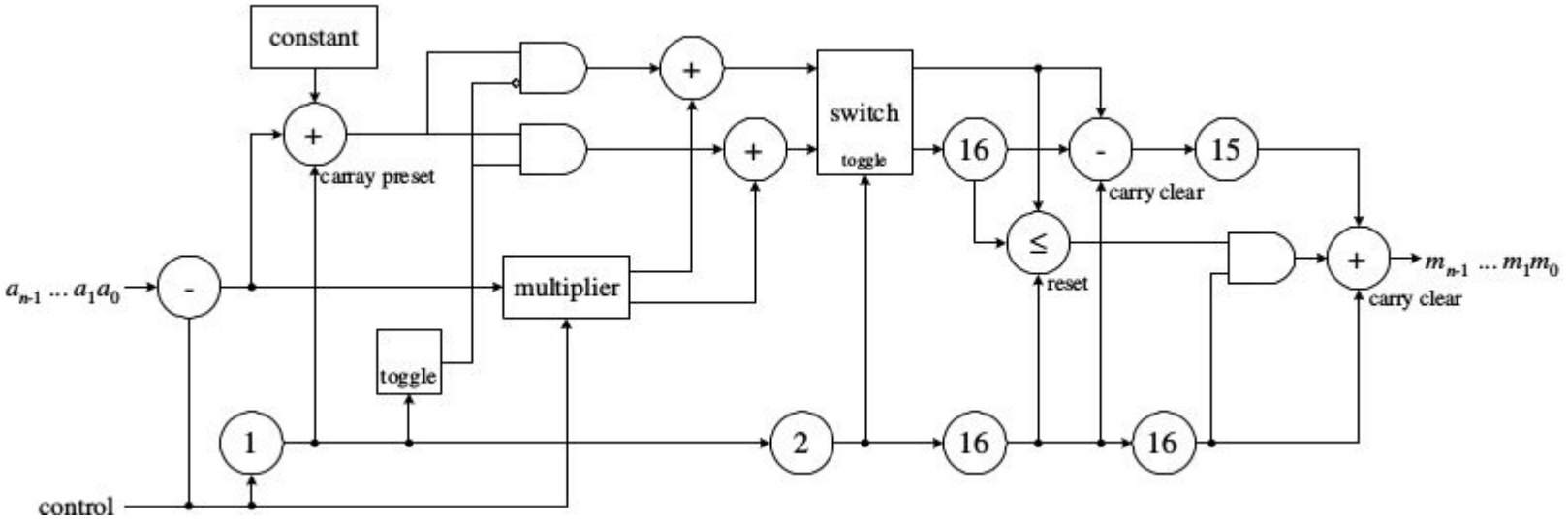
Bit-serial. Последовательно-двоичная архитектура. Умножение по модулю



Так как вектор был перенаправлен на одну из дорожек, логический ноль выбрал другую дорожку, чтобы тот нёс дополнительные нули.

Чтобы в итоге старшие и младшие значения *t* получились выровненными во времени, требуется сдвиг на 16 позиций. Входящие и исходящие значения регистра сдвига это старшие и младшие значения переменной *t*, соответственно, 16 циклов после *t* действительны. Сам регистр сдвига реализуется через **SRL16E**.

Bit-serial. Последовательно-двоичная архитектура. Умножение по модулю

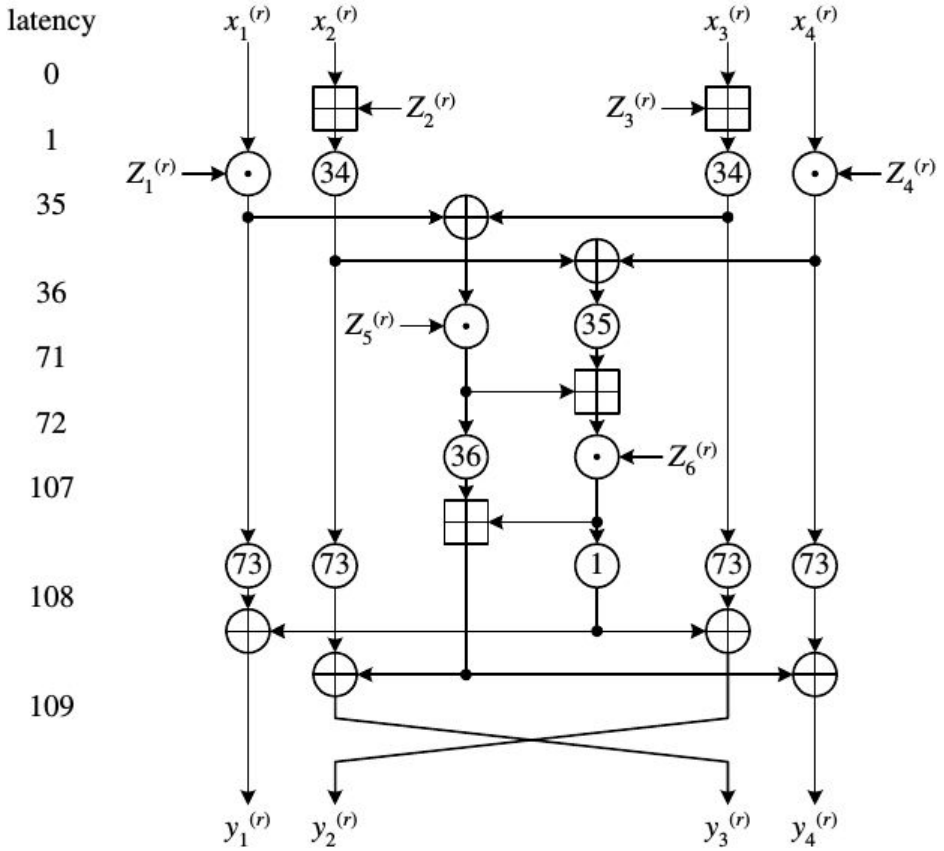


Здесь представлена конечная архитектура для оператора умножения по модулю. После инициализации подключ, ассоциируемый с оператором, проходит на него двоично-последовательно. Заранее уменьшенный подключ движется на регистры умножителя и в то же время сохраняется в **SRL16E**, который должен их хранить.

Используя идею множества дорожек(каналов) умножение достигает 16 циклов и даже считается 32ух битный промежуточный результат. Такая схема увеличивает вдвое пропускную способность, но так как происходит общее использование регистров *b* цена устройства меньше чем double.

Bit-serial. Последовательно-двоичная архитектура

Плата с 8,5 раундами реализованных нисходящей структурой



Плата хавает 1 64ёх битный простой текст каждые 16 циклов, выдавая показатель шифрования: f * 64/16 Мб/сек. на системе с частотой процессора f МГц.

Пример: для 150 МГц это будет 600 Мб/сек.

У каждого раунда задержка равна 109 циклам, задержка конечного преобразования 35 циклов. У каждого преобразователя с последовательной в параллельную архитектуру на выходе задержка получается 16 циклов.

Следовательно в среднем у платы IDEA задержка: 109 * 8 + 35 + 16 = 923 циклов, с чистотой 150 МГц задержка 6,153 мкс.

Слабые ключи

Существуют большие классы слабых ключей. Слабые они в том смысле, что существуют процедуры, позволяющие определить, относится ли ключ к данному классу, а затем и сам ключ. В настоящее время известны следующие:

- $2^{23} + 2^{35} + 2^{51}$ слабых к дифференциальному криптоанализу ключей. Принадлежность к классу 2^{51} можно вычислить за 2^{12} операций с помощью подобранного открытого текста. Авторы данной атаки предложили модификацию алгоритма IDEA. Данная модификация заключается в замене подключей $Z_i^{(r)}$ на соответствующие $Z'^{(r)}_i = a \oplus Z_i^{(r)}$, где r — номер раунда шифрования. Точное значение *a* не критично. Например при *a* = 0xdae данные слабые ключи исключаются[стойкость 6].
- 2^{63} слабых к линейному дифференциальному криптоанализу ключей[стойкость 7]. Принадлежность к данному классу выясняется с помощью теста на связанных ключах.
- $2^{53} + 2^{56} + 2^{64}$ слабых ключей было найдено с использованием метода бумеранга (boomerang attack), предложенного Дэвидом Вагнером(David Wagner)[стойкость 8]. Тест на принадлежность к данному классу выполняется за 2^{16} операций и потребует 2^{16} ячеек памяти [стойкость 9].

Существование столь больших классов слабых ключей не влияет на практическую криптостойкость алгоритма IDEA, так как полное число всех возможных ключей равно 2^{128}.

Преимущества

Недостатки

hh

Один из наиболее известных в мире криптологов **Брюс Шнайер** в своей книге «*Прикладная криптография*», 1996 г. заметил:

«…удивительно, как такие незначительные изменения могут привести к столь большим различиям» сравнивая *PES* и *IDEA*.

«Мне кажется, это самый лучший и надежный блочный алгоритм, опубликованный до настоящего времени».

Источники

* https://google.ru

Источники

* http://www.quadibloc.com/crypto/co040302.htm

* http://htrd.su/wiki/_media/zhurnal/2012/03/23/todo_prikladnaja_kriptografija/cryptoshn.pdf

* https://en.wikipedia.org/wiki/International_Data_Encryption_Algorithm

* https://ru.wikipedia.org/wiki/IDEA

* http://www.ti.com/lit/an/slyt264/slyt264.pdf

* https://books.google.ru/books?id=UmUd3aIuk3IC&pg=SA2-PA17&lpg=SA2-PA17&dq=latency+n+cycles&source=bl&ots=TXbar0nRWX&sig=yAVhb1jN1nT1UnvHpwlRQVEdhhY&hl=ru&sa=X&ved=0ahUKEwjiuIaC6aPQAhVIhywKHXKmDT4Q6AEINDAD#v=onepage&q=latency%20n%20cycles&f=false

A latency cycle is a latency sequence which repeats that means if I have a sequence like this say 5, 2, 5, 2, 5, 2 then this 5, 2 these becomes a latency cycle or I can say 2, 5 is a latency cycle because that is a sequence which repeats.

A latency cycle is a possibly repeating sequence of issue latencies.

Cycle latency is the number of complete data cycles between the conversion initiation and the availability of the corresponding output data

Pipeline - Канал, дорожка, поток