**Глава 1. Проект «eSTREAM»**

**1.1 Проект «NESSIE»**

Проект «NESSIE» (англ. New European Schemes for Signatures, Integrity and Ecryptions – Новые европейские алгоритмы для электронной подписи, целостности и шифрования) – это научно-исследовательский проект для определения новых шифровальных алгоритмов, на базе которых должны быть созданы новые криптографические стандарты Европы. Конкурс проходил с января 2000 года по февраль 2003 года. Участниками проекта могли стать любые организации и частные лица, приславшие свой криптографический алгоритм для анализа.

Проект насчитывал 5 категорий шифров:

1. блочное шифрование данных;
2. ассиметричное шифрование
3. хэширование;
4. электронная подпись;
5. потоковое шифрование

Отбор оптимальных алгоритмов, которые претендовали на победу, проходил в 3 этапа:

1. первичное изучение алгоритмов;
2. криптоаналитические исследования алгоритмов;
3. сравнение алгоритмов по нескольким критериям: быстродействие, минимальные требования к вычислительным ресурсам, производительность, гибкость и другие.

Всего на конкурс было заявлено 42 криптографических шифра, из которых 18 выбыло после 1-го этапа. В результате работы комиссии на 2-м и 3-м этапах победителями были объявлены 12 алгоритмов.

Победителями в соответствующих категориях стали:

1. блочные шифры: MISTY 1(Mitsubishi Electric Corp., Япония), Camellia (Nippon Telegraph and Telephone Corp., Япония и Mitsubishi Electric Corp., Япония), SHACAL-2 (Gemplus, Франция), AES (Винсент Рэймен и Йоан Даймен, Бельгия);
2. ассиметричные шифры: ACE Ecnrypt (научно-исследовательская лаборатория IBM, Швейцария), PSEC-KEM (Nippon Telegraph and Telephone Corp., Япония);
3. хэш-функции: Two-Track-MAC (K.U.Leuven, Бельгия и Debis AG, Германия), UMAC (Intel Corp., США, университет Невады, США, научно-исследовательская лаборатория IBM, США, Technion, Израиль и университет Калифорнии в Дэвисе, США), Whirlpool (Scopus Tecnologia S.A., Бразилия и K.U.Leuven, Бельгия);
4. алгоритмы для электронной подписи: ECD SA (Certicom Corp., США and Certicom Corp., Канада), RSA-PSS (лаборатории RSA, США), SFLASH (Schlumberger, Франция).

В категории потоковое шифрование было представлено 6 алгоритмов, которые не прошли все испытания и были признаны непригодными для использования.

Данный факт послужил началом для старта проекта по выявлению новых потоковых криптографических алгоритмов «eSTREAM».

**1.2 Проект «eSTREAM»**

Проект «eSTREAM» (англ. European Stream – Европейский поток) – научно-исследовательский проект по выявлению новых потоковых криптографических алгоритмов. Был начат в феврале 2004 года, после взлома всех 6-ти шифров проекта «NESSIE». Официально проект был завершен в мае 2008 года. Последняя публикация документации шифров-победителей датируется январем 2012 года. После чего изменения в криптографические алгоритмы не вносились.

Созданию проекта «eSTREAM» послужили результаты конференции, проходившей 14-15 октября 2004 года в городе Брюгге, «SASC – The State of the Art of Stream Ciphers» (Современное состояние потоковых шифров). Организаторы и слушатели конференции пришли к выводу, что необходимо организовать новый конкурс по выявлению потоковых криптографических алгоритмов.

В ноябре 2004 года началась приемная компания претендентов на участие в конкурсе. К участникам предъявлялись следующие основные требования: длина ключа максимум 128 бит; алгоритмы должны были работать быстрее, чем американский стандарт шифрования AES-128 (Advanced Encryption Standard) в режиме счетчика.

Все конкурсанты разделялись на 2 категории: «Программно-ориентированные алгоритмы» и «аппаратно-ориентированные алгоритмы».

В результате предварительного отбора, для участия в конкурсе были отобраны следующие участники: Frogbit, Fubuki, MAG, Mir-1, SSS, TRBDK3 YAEA, Phelix, Py, ABC, Achterbahn, DICING, Hermes8, NLS, Polar Bear, Pomaranch, SFINKS, TSC-3, VEST, WG, Yamb, ZK-Crypt, CryptMP, DECIM, Dragon, Edon80, LEX, MOSQUITO, Grain, HC-128, MICKEY, Rabbit, Salsa20, SOSEMANUK, Trivium, F-FC-SR.

На 1-м этапе проекта (29 апреля 2005 – 27 марта 2006) все участники подверглись общему анализу. Шифры были исследованы на надежность, производительность, корректность работы, простоту и гибкость реализации. Была изучена подробная документация по каждому алгоритму. Проверялась ее полнота и понятность, которая необходима при реализации сторонними разработчиками.

После 1-го этапа, в результате общего анализа, были отсеяны следующие алгоритмы: Frogbit, Fubuki, MAG, Mir-1, SSS, TRBDK3 YAEA.

2-й этап конкурса (2 августа 2006 – апрель 2007) включал в себя анализ алгоритмов производительности на различных платформах, операционных системах и архитектурах процессоров. В результате подробного анализа были выявлены слабые алгоритмы, которые выбыли из проекта: Phelix, Py, ABC, Achterbahn, DICING, Hermes8, NLS, Polar Bear, Pomaranch, SFINKS, TSC-3, VEST, WG, Yamb, ZKCrypt.

3-й этап проекта (апреля 2007 – май 2008) был ознаменован выпуском книги «New stream cipher designs» (Новые конструкции поточного шифра), в которую вошли история разработки и описание всех криптографических алгоритмов, вышедших в финальную часть.

Для выбора алгоритмов-победителей была сформирована комиссия, в которую вошли разработчики и исследователи крупнейших компаний и университетов Европы: С. Бэббидж (S. Babbage, Vodafone, Великобритания), К. Де Каннье (C. De Canniere, Katholieke Universiteit Leuven, Бельгия), А. Канто (A. Canteaut, INRIA, Франция), К. Сид (C. Cid, Royal Holloway, University of London, Великобритания), А. Жильбер (H. Gilbert, France Telecom R&D, Франция), Т. Юханссон (T. Johansson, Lund University, Швеция), К. Пар (C. Paar, Ruhr-University of Bochum, Германия), М. Паркер (M. Parker, University of Bergen, Норвегия), Б. Пренел (B. Preneel, Katholieke Universiteit Leuven, Бельгия), В. Реймен (V. Rijmen, Graz University of Technology, Австрия), М. Робшоу (M. Robshaw, France Telecom R&D, Франция), У Хунцзюнь (Hongjun Wu, Katholieke Universiteit Leuven, Бельгия).

Комиссия изучила всю имеющуюся документацию по алгоритмам, выслушала презентации авторов, и 15 апреля 2008 года, на основании тщательного анализа, были объявлены алгоритмы-победители проекта «eSTREAM» в обеих категориях.

В категории «Программно-ориентированные алгоритмы» победителями стали: HC128, Rabbit, Salsa20, Sosemanuk.

В категории «Аппаратно-ориентированные алгоритмы» победителями стали: Grain, MICKEY, Trivium, F-FCSR.

В сентябре 2008 года криптографический шифр F-FCSR был исключен из финальной публикации в виду криптографической слабости, обнаруженной после завершения проекта.

Следующие разделы будут посвящены алгоритмам шифрования, которые стали победителями проекта «eSTREAM».

**1.3 Алгоритм шифрования Salsa20**

Salsa20 – алгоритм поточного шифрования информации, разработанный Даниэлем Бернштейном. Алгоритм был впервые представлен на проекте «eSTREAM», где стал победителем в категории «Программно-ориентированные алгоритмы».

Шифр использует хэш-функцию с 20-ю циклами, в которых поочередно применяются операции побитового сложения и циклического сдвига 32-битных машинных слов, которая обеспечивает криптостойкость алгоритма.

Основные характеристики алгоритма:

1. длина ключа не должна превышать 256 бит. В зависимости от длины ключа (до 128 бит или 256 бит) алгоритм будет видоизменяться;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. в функции расширения ключа (в зависимости от длины ключа) используется константа: «expand 16-byte key» или «expand 32-byte key»;
5. на вход алгоритму подается уникальный 64-битный вектор инициализации.

Согласно официальной документации, в алгоритме Salsa20 определены несколько стандартных функций, определяющих ключевые преобразования: quarterround, rowround, columnround, doubleround, salsa20 hash function и salsa20 expansion function.

**1.3.1 Базовые преобразования**

Функция quarterround осуществляет преобразование над 4-мя 32-битными словами, при этом для каждого слова складываются 2 предыдущих. Затем полученная сумма сдвигается на определенное количество бит и полученный результат побитого суммируется с исходным словом. Математически функция quarterround(y) выглядит следующим образом:

z1 = y1 ⊕ ((y0 + y3) <<< 7);

z2 = y2 ⊕ ((z1 + y0) <<< 9);

z3 = y3 ⊕ ((z2 + z1) <<< 13);

z0 = y0 ⊕ ((z3 + z2) <<< 18),

где zi и yi 32-битные слова.

В функцию rowround передается 16 слов, которые представляются в виде матрицы 4х4. Затем каждый ряд этой матрицы передается в функцию quarterround(y). Слова из строки берутся по порядку, начиная с i-го для i-ой строки, где i = {0, 1, 2, 3}. Математически функция rowround(y) может быть представлена:

(z0, z1, z2, z3) = quarterround(y0, y1, y2, y3);

(z5, z6, z7, z4) = quarterround(y5, y6, y7, y4);

(z10, z11, z8, z9) = quarterround(y10, y11, y8, y9);

(z15,z12, z13, z14) = quarterround(y15, y12, y13, y14),

где y = (y0, …, y15) – 16-битные слова,

z = (z1, …, z15) – 16-битные слова, получаемые на выходе функции rowround(y).

На вход функции columnround(y) подаются 16 32-битных слов, которые представляются в виде матрицы 4х4, аналогично функции rowround. Затем каждый столбец этой матрицы подается в функцию quarterround(y). Слова из столбца берутся по порядку, начиная с j-го для j-го столбца, где j = {0, 1, 2, 3}. С помощью формул функцию columnround(y) можно представить следующим образом:

(y0, y4, y8, y12) = quarterround(x0, x4, x8, x12);

(y5, y9, y13, y1) = quarterround(x5, x9, x13, x1);

(y10, y14, y2, y6) = quarterround(x10, x14, x2, x6);

(y15, y3, y7, y11) = quarterround(x15, x3, x7, x11),

где x = (x1, …, x15) – 16-битные слова,

y = (y0, …, y15) – 16-битные слова, получаемые на выходе функции columnround(y).

Функция doubleround(y) является последовательным выполнением функций columnround(y) и rowround(y). В общем виде данное преобразование можно записать так:

doubleround(x) = rowround(columnround(y)),

где x и y последовательность из 16-ти 32-битных слов.

**1.3.2 Хэш-функция**

Хэш-функция предназначена для генерирования уникальной 64-битной ключевой последовательности.

На вход функции подается 64-байтная последовательность. Затем каждые 4 байта подвергаются операции приведения к обратному порядку байт (little endian). Математически эта операция выглядит так:

littleendian(b) = b0 + 28\*b1 + 216\*b2 + 224\*b3,

где b = (b0, b1, b2, b3) – 32-битовое слово, то есть упорядоченная последовательность из 4 байт (b0, b1, b2, b3).

На выходе преобразования littleendian получается последовательность, состоящая из 16-ти 32-битных слов, которая 20 раз подается на вход функции doubleround.

На выходе хэш-функции salsa20 получается 512-ти битовая ключевая последовательность, которая используется при шифровании/расшифровывании потоков данных.

**1.3.3 Расширение ключа**

На вход функции расширении ключа подается секретный ключ (до 256 бит) и 128-битный вектор инициализации.

В функции используются 2 константы: «expand 16-byte key» (если длина ключа до 128 бит) или «expand 32-byte key» (если длина ключа до 256 бит).

Расширение ключа происходит за счет последовательной записи секретного ключа, вектора инициализации и константы в 512-битовую последовательность, которая передается в хэш-функцию для генерации ключевой последовательности.

Схема расширения ключа выглядит так:

1. для ключа до 128 бит: key\_exp = (σ0, k, σ1, n, σ2, k, σ3);
2. для ключа до 256 бит: key\_exp = (σ0, k0, σ1, n, σ2, k1, σ3),

где k – секретный ключ, либо его i-часть,

σi – константа (определяется в зависимости от длины ключа),

n – вектор инициализации.

**1.3.4 Функция шифрования/расшифровывания**

В результате работы вышеперечисленных функций генерируется уникальная ключевая последовательность, которая используется для операции побитового сложения по модулю 2 с каждым байтом входного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

**1.2 Алгоритм шифрования Rabbit**

Rabbit – алгоритм поточного шифрования информации, впервые был представлен широкой публике в феврале 2003 года. В мае 2005 года Rabbit был заявлен для участия в проекте «eSTREAM», где стал победителем в категории «Программно-ориентированные алгоритмы».

Разработчиками алгоритма являются Мартин Боесгард (Martin Boesgaard), Мэтт Вестэрагер (Mette Vesterager), Томас Педерсен (Thomas Pedersen), Джаспер Кристиансен (Jesper Christiansen) и Ове Скавениус (Ove Scavenius).

Основные характеристики алгоритма:

1. длина секретного ключа до 128 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход алгоритму подается уникальный 64-битный вектор инициализации.

Согласно официальной документации в алгоритме Rabbit определены несколько стандартных функций, определяющих ключевые преобразования: key\_setup, iv\_setup, next\_state.

**1.2.1 Базовые преобразования**

Функция key\_setup предназначена для установления ключа в память компьютера по следующему алгоритму. Секретный ключ разбивается на 8 подключей k0, k1, …, k7, на основании которых генерируются 8 переменных состояний xj,0 и 8 счетчиков состояний сj,0, по следующей схеме:

xj,0 = k(j+1 mod 8) × kj, где j четно;

xj,0 = k(j+5 mod 8) × k(j+4 mod 8), где j нечетно;

cj,0 = k(j+4 mod 8) × k(j+5 mod 8), где j четно;

cj,0 = kj × k(j+1 mod 8), где j нечетно,

где операция × означает конкатенацию 2-х подключей,

j = {0, 1, …, 7}.

Вся схема проходит подобные преобразования 4 раза с помощью функции следующего шага (next\_state).

После чего все счетчики состояний переинициализируются согласно следующей формуле:

cj,4 = cj,4 ⊕ x(j+4 mod 8),4.

После функции key\_setup следует функция iv\_setup, которая предназначена для установления вектора инициализации в память компьютера по следующему алгоритму:

c0,4 = c0,4 ⊕ IV[31..0];

c2,4 = c2,4 ⊕ IV[63..32];

c4,4 = c4,4 ⊕ IV[31..0];

c6,4 = c6,4 ⊕ IV[63..32];

c1,4 = c1,4 ⊕ (IV[63..48] × IV[31..16]);

c3,4 = c3,4 ⊕ (IV[47..32] × IV[15..0]);

c5,4 = c5,4 ⊕ (IV[63..48] × IV[31..16]);

c7,4 = c7,4 ⊕ (IV[47..32] × IV[15..0]).

Вся схема проходит подобные преобразования 4 раза с помощью функции следующего шага, которая так же лежит в основе генерации ключевой последовательности.

Функция next\_state содержит следующие уравнения:

x0,i+1 = g0,i + (g7,i <<< 16) + (g6,i <<< 16);

x1,i+1 = g1,i + (g7,i <<< 16) + g7,i;

x2,i+1 = g2,i + (g1,i <<< 16) + (g0,i <<< 16);

x3,i+1 = g3,i + (g7,i <<< 16) + g1,i;

x4,i+1 = g4,i + (g3,i <<< 16) + (g2,i <<< 16);

x5,i+1 = g5,i + (g7,i <<< 16) + g3,i;

x6,i+1 = g6,i + (g5,i <<< 16) + (g4,i <<< 16);

x7,i+1 = g7,i + (g7,i <<< 16) + g5,i,

где xi,j – переменные состояния,

функция gj,i = ((xi,j + cj,i+1)2 ⊕ ((xi,j + cj,i+1)2 >> 32)) mod 232.

Система счетчиков так же подвержена изменением согласно функции следующего шага:

c0,i+1 = c0,i+1 + a0 + φ7,i mod 232;

c1,i+1 = c1,i+1 + a1 + φ0,i+1 mod 232;

c2,i+1 = c2,i+1 + a2 + φ1,i+1 mod 232;

c3,i+1 = c3,i+1 + a3 + φ2,i+1 mod 232;

c4,i+1 = c4,i+1 + a4 + φ3,i+1 mod 232;

c5,i+1 = c5,i+1 + a5 + φ4,i+1 mod 232;

c6,i+1 = c6,i+1 + a6 + φ5,i+1 mod 232;

c7,i+1 = c7,i+1 + a7 + φ6,i+1 mod 232,

где функция φj,i+1 определяется следующим образом:

φj,i+1 = 1, если c0,i + a0 + φ7,i  ≥ 232;

φj,i+1 = 1, если cj,i + aj + φj-1,i+1 ≥ 232;

φj,i+1 = 0, если другое.

Константы ai определены согласно официальной документации:

a0 = 0x4D34D34D; a1 = 0xD34D34D3;

a2 = 0x34D34D34; a3 = 0x4D34D34D;

a4 = 0xD34D34D3; a5 = 0x34D34D34;

a6 = 0x4D34D34D; a7 = 0xD34D34D3.

**1.2.2 Функция шифрования/расшифровывания**

В результате работы функций next\_state генерируется уникальная ключевая последовательность по следующей схеме:

si[15..0] = x0,i[15..0] ⊕ x5,i[31..16]; si[31..16] = x0,i[31..16] ⊕ x3,i[15..0];

si[47..32] = x2,i[15..0] ⊕ x7,i[31..16]; si[63..48] = x2,i[31..16] ⊕ x5,i[15..0];

si[15..0] = x4,i[15..0] ⊕ x1,i[31..16]; si[95..80] = x4,i[31..16] ⊕ x7,i[15..0];

si[15..0] = x6,i[15..0] ⊕ x3,i[31..16]; si[127..112] = x6,i[31..16] ⊕ x1,i[15..0],

где si – 16 бит ключевой последовательности.

Ключевая последовательность используется для операции побитового сложения по модулю 2 с каждым байтом входного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

**1.3 Алгоритм шифрования HC-128**

HC – алгоритм поточного шифрования информации, разработанный криптографом У Хунцзюн (Wu Hongjun). 128-битный вариант реализации алгоритма был заявлен для участия в проекте «eSTREAM», где стал победителем в категории «Программно-ориентированные алгоритмы».

Основные характеристики алгоритма:

1. длина ключа не должна превышать 128 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход алгоритму подается уникальный 128-битный вектор инициализации.

Согласно официальной документации в алгоритме HC-128 определены несколько стандартных функций, определяющих базовые преобразования: функция инициализации и функция генерирования ключевой последовательности.

**1.3.1 Базовые преобразования**

В алгоритме HC-128 используется 6 стандартных преобразований: f1, f2, g1, g2, h1,h2. На математическом языке данные преобразования можно представить так:

f1(x) = (x >>> 7) ⊕ (x >>> 18) ⊕ (x >> 3);

f2(x) = (x >>> 17) ⊕ (x >>> 19) ⊕ (x >> 10);

g1(x, y, z) = ((x >>> 10) ⊕ (z >>> 23)) + (y >>> 8);

g2(x, y, z) = ((x <<< 10) ⊕ (z <<< 23)) + (y <<< 8);

h1(x) = Q[x0] + Q[256+x2];

h2(x) = P[x0] + P[256+x2],

где x, y, z – 32-битные машинные слова,

P и Q – массивы 32-битных слов.

Функция инициализации предназначена для записи и расширения секретного ключа и вектора инициализации в массивы P и Q соответственно. Схематично данный процесс можно представить так:

Wi = Ki, при 0 ≤ i ≤ 7;

Wi = IVi, при 8 ≤ i ≤ 15;

Wi = f2(Wi-2) + Wi-7 + f1(Wi-15) + Wi-16 + i, при 16 ≤ i ≤ 1279;

P[i] = Wi+256, Q[i] = Wi+768, при 0 ≤ i ≤ 511;

Система начинает процесс шифрования 1024 раза без генерации ключевой последовательности:

P[i] = (P[i] + g1(P[i-3], P[i-10], P[i-511]) ⊕ h1(P[i-12])), при 0 ≤ i ≤ 511;

Q[i] = (Q[i] + g1(Q[i-3], Q[i-10], Q[i-511]) ⊕ h1(Q[i-12])) при 0 ≤ i ≤ 511,

где операция «-» – операция вычитания по модулю 512.

Функцию генерирования ключевой последовательности можно представить так:

при (i mod 1024) < 512

P[i] = (P[i] + g1(P[i-3], P[i-10], P[i-511]);

s = h1(P[i-12]) ⊕ P[i].

при (i mod 1024) ≥ 512

Q[i] = (Q[i] + g1(Q[i-3], Q[i-10], Q[i-511]);

s = h2(Q[i-12]) ⊕ Q[i],

где si – 32-битный элемент ключевой последовательности,

i – длина входного потока данных.

**1.3.2 Функция шифрования/расшифровывания**

В результате работы функции генерируется уникальная ключевая последовательность, которая используется для операции побитового сложения по модулю 2 с каждым байтом входного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

**1.4 Алгоритм шифрования Sosemanuk**

Sosemanuk – алгоритм поточного шифрования информации. Был заявлен для участия в проекте «eSTREAM», где стал победителем в категории «Программно-ориентированные алгоритмы».

Разработчиками алгоритма являются К. Бербайн (C. Berbain), О. Биллет (O. Billet), А. Кантейт (A. Canteaut), Н. Кортойс (N. Courtois), Х. Гилберт (H. Gilbert), Л. Губин (L. Goubin), А. Гойджет (A. Gouget),Л. Гранболан (A. Gouget), К. Люрадойкс (L. Granboulan), М. Миниер (M. Minier), Т. Порнин (T. Pornin) и Х. Сиберт (H. Sibert).

Основные характеристики алгоритма:

1. длина ключа не должна превышать 256 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход алгоритму подается уникальный 128-битный вектор инициализации.

Согласно официальной документации в алгоритме Sosemanuk определены несколько стандартных функций, определяющих ключевые преобразования: функции расширения ключа и вектора инициализации, функция генерирования ключевой последовательности.

**1.4.1 Базовые преобразования**

Функция расширения ключа основана на разбиении секретного ключа на 8 подключей по 32 бита каждый. Затем все подключи побитового складываются по модулю 2 и полученный результат циклически сдвигается на 11 бит. Затем массив подключей подается на вход 1 раунду блочного алгоритма Serpent. Алгоритм расширения ключа насчитывает 25 подобных преобразований, на выходе которых генерируется 100 32-битных слов. Схема расширения ключа приведена ниже:

w0 = K0, …, K31;

w1 = K32, …, K63;

.

.

.

w7 = K224, …, K255;

tt = wi ⊕ wi+3 ⊕ wi+5 ⊕ wi+7 при i ∍ {0, …, 7};

wi = tt <<< 11;

serpent(w),

где wi – i-й 32-х битный подключ, Ki – i-й бит ключа, serpent – расширение ключа блочного алгоритма Serpent (тут ссылка будет), tt – переменная для промежуточных вычислений.

Функция расширения вектора инициализации основана на использовании блочного шифра Serpent. Вектор инициализации разбивается на 4 32-х битных вектора, которые подаются на вход 1-му раунду шифра Serpent. Вся схема проходит 24 итерации, что позволяет достичь криптостойкости алгоритма. Расширение вектора инициализации можно представить так:

r0 = IV0, …, IV31;

r1 = IV32, …, IV63;

r2 = IV64, …, IV95;

r3 = IV96, …, IV127;

serpent24(r),

где ri – i-й 32-х битный вектор,

IVi – i-й бит вектора инициализации,

serpent24 – 24 раунда блочного алгоритма Serpent (тут ссылка будет).

Функция генерации ключевой последовательности осуществляет преобразования над массивами байт полученных при работе функций расширения ключа и вектора инициализации. Схему генерации ключевой последовательности можно представить так:

for i = 0 to 4

tt = XMUX(r1, sj, st);

or1 = r1;

r1 = r2 + tt;

tt = 0x54655307 \* r1;

r2 = tt <<< 7;

vi = st;

ui = (sj + r1) ⊕ r2;

end for,

z0 = (u2 ⊕ v0);

z1 = (u3 ⊕ v1);

z2 = (u1 ⊕ v2);

z3 = (u4 ⊕ v3),

где s – массив битов, полученный путем расширения секретного ключа и вектора инициализации,

z – поток ключевой последовательности,

ui, vi, r1, r2, tt, or1 – переменные для промежуточных вычислений,

i j – индексы генерируемые в зависимости от раунда алгоритма,

функция XMUX(r1, sj, st) возвращает sj если r1= 0, или st ⊕sj если r1 = 1.

**1.4.2 Функция шифрования/расшифровывания**

В результате работы функции генерируется уникальная ключевая последовательность, которая используется для операции побитового сложения по модулю 2 с каждым байтом входного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

**1.5 Алгоритм шифрования Grain**

Grain – алгоритм поточного шифрования информации, разработанный Т. Юханссоном (T. Johansson), А. Максимовым (A. Maximov) и М. Хеллом (M. Hell). Версия со 128-битным ключом была заявлена для участия в проекте «eSTREAM», где стала победителем в категории «Аппаратно-ориентированные алгоритмы».

Основные характеристики алгоритма:

1. длина ключа не должна превышать 128 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход алгоритму подается уникальный 96-битный вектор инициализации.

Согласно официальной документации в алгоритме Grain определены несколько стандартных функций, определяющих базовые преобразования: регистр сдвига с линейной обратной связью, регистр сдвига с нелинейной обратной связью и выходная функция.

**1.5.1 Базовые преобразования**

Секретный ключ разбивается на биты и загружается в регистр сдвига с нелинейной обратной связью (РССНОС), а вектор инициализации – в регистр сдвига с линейной обратной связью (РССЛОС).

РССЛОС, объемом 96 бит, предназначен для генерации нового бита на основе битов вектора инициализации. Формула для генерирования нового бита:

si+96 = s0 ⊕ s7 ⊕ s38 ⊕ s70 ⊕ s81 ⊕ s96,

где si - i-ый бит РССЛОС.

РССНОС, объемом 128 бит, предназначен для генерирования нового бита на основе битов секретного ключа и 1-го бита регистра с линейной обратной связью. Формула для генерации нового бита:

bi+128 = si ⊕ bi + b26 ⊕ b56 ⊕ b91⊕ b96 ⊕ (b3 & b67) ⊕ (b11 & b13) ⊕ (b17 & b18) ⊕ (b27 & b59) ⊕ (b40 & b48) ⊕ (b61 & b65) ⊕ (b68 & b84),

где bi – i-ый бит регистра РССНОС,

si - i-ый бит РССЛОС.

При старте работы алгоритма, генерация новых битов с помощью РССНОС и РССЛОС, повторяется 256 раз без генерации битов ключевого потока. Весь регистр сдвигается на 1-ну позицию влево, а новые сгенерированные биты записываются в последние позиции соответствующих регистров.

Выходом выходной функции является бит ключевой последовательности, который вычисляется по следующей формуле:

zi = j+i ⊕ h(x) ⊕ si+93 при A = {2, 15, 36, 45, 64, 73, 89},

где h(x) = (b12 & s8) ⊕ (s13 & s20) ⊕ (b95 & s42) ⊕ (s60 & s79) ⊕ (b12 & b95 & s95),

bi – i-ый бит регистра РССНОС,

si – i-ый бит РССЛОС

Таким образом, за одну итерацию алгоритма генерируется 1 бит ключевой последовательности.

**1.5.2 Функция шифрования/расшифровывания**

Так как за одну итерацию алгоритма генерируется 1 бит ключевой последовательности, то для получения 1 байта потребуется 8 итераций алгоритма.

Поэтому шифр за один проход (здесь это 8 итераций алгоритма) выполняет операцию побитового сложения по модулю 2 1 байта уникальной ключевой последовательности с 1 байтом исходного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

**1.6 Алгоритм шифрования Trivium**

Trivium – алгоритм поточного шифрования информации, разработанный К. де Канньером и Б. Пренелом. Шифр был заявлен для участия в проекте «eSTREAM», где стал победителем в категории «Аппаратно-ориентированные алгоритмы».

Основные характеристики алгоритма:

1. длина ключа не должна превышать 80 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход алгоритму подается уникальный 80-битный вектор инициализации.

Согласно официальной документации в алгоритме Trivium определены несколько стандартных функций, определяющих базовые преобразования: функция инициализации и функция генрации ключеовй последовательности.

**1.6.1 Базовые преобразования**

Функция инициализации представляет собой преобразование 80-ти битного секретного ключа и вектора инициализации в определенную последовательность бит и затем их трансформацию для дальнейшей генерации ключевого потока.

Схематично функцию инициализации можно представить так:

(s1, s2, …, s93) = (K1, …, K80, 0, …, 0);

(s94, s95, …, s177) = (IV1, …, IV80, 0, …, 0);

(s178, s179, …, s288) = (0, …, 0, 1, 1, 1);

for i = 1 to 4 \* 288 do

t1 = s66 ⊕ s91 & s92 ⊕ s93 ⊕ s171;

t2 = s161 ⊕ s175 & s176 ⊕ s177 ⊕ s264;

t3 = s243 ⊕ s286 & s287 ⊕ s288 ⊕ s69;

(s1, s2, …, s93) = (t3, s1, s2, …, s92);

(s94, s95, …, s177) = (t1, s94, s95, …., s176);

(s178, s179, …, s288) = (t2, s178, s179, …, s287);

end for,

где s – 288-ми битовый регистр,

Ki – i-й бит ключа,

IVi – i-й бит вектора инициализации,

t1, t2, t3 – биты для промежуточных вычислений.

После вышеописанного преобразования регистр s подается на вход функции генерации ключевой последовательности, алгоритм которой приведен ниже:

for i = 1 to N do

t1 = s66 ⊕ s93;

t2 = s162 ⊕ s177;

t3 = s243 ⊕ s288;

zi = t1 ⊕ t2 ⊕ t3;

t1 = t1 ⊕ s91 & s92 ⊕ s171;

t2 = t2 ⊕ s175 & s176 ⊕ s264;

t3 = t3 ⊕ s286 & s287 ⊕ s69;

(s1, s2, …, s93) = (t3, s1, s2, …, s92);

(s94, s95, …, s177) = (t1, s94, s95, …., s176);

(s178, s179, …, s288) = (t2, s178, s179, …, s287);

end for,

где N – длина входного потока в битах,

zi – бит ключевой последовательности.

**1.6.2 Функция шифрования/расшифровывания**

В результате работы алгоритма генерируется уникальная ключевая последовательность, которая используется для операции побитового сложения по модулю 2 с каждым байтом входного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

**1.7 Алгоритм шифрования MICKEY**

MICKEY – алгоритм поточного шифрования информации, разработанный С. Бэббиджем и М. Доддомю Шифр был заявлен для участия в проекте «eSTREAM», где стал победителем в категории «Аппаратно-ориентированные алгоритмы».

Основные характеристики алгоритма:

1. длина ключа не должна превышать 80 бит;
2. входной поток данных не ограничен по объему;
3. выходной поток данных такого же объема, как и входной;
4. на вход алгоритму подается уникальный 80-битный вектор инициализации.

Согласно официальной документации в алгоритме MICKEY определены несколько стандартных функций, определяющих базовые преобразования: сдвиг регистра R (CLOCK\_R), сдвиг регистра S (CLOCK\_S), функция управления алгоритмом (CLOCK\_KG), функция загрузки ключа и вектора инициализации и функция генерирования ключевой последовательности.

**1.7.1 Базовые преобразования**

Функция сдвига регистра R принимает несколько параметров: CLOCK\_R (R, input\_bit\_r, control\_bit\_r). Здесь R – это указатель на сам регистр, а input\_bit\_r и control\_bit\_r – специальные биты. Данное преобразование на псевдокоде выглядит следующим образом:

CLOCK\_R (R, input\_bit\_r, control\_bit\_r)

feedback\_bit = r99 ⊕ input\_bit\_r;

для 1 ≤ i ≤ 99, r'i = ri-1; r'0 = 0;

для 0 ≤ i ≤ 99, если i ∈ RTAPS, r'i = r'i ⊕ feedback\_bit;

если control\_bit\_r = 1, то

для 0 ≤ i ≤ 99, r'i = r'i ⊕ ri,

где r0, r1, …, r99 – биты регистра R до сдвига,

r'1, r'2, …, r'99 – биты регистра R после сдвига,

RTAPS – набор входов обратной связи.

Массив RTAPS определен следующим образом:

RTAPS = { 0, 1, 3, 4, 5, 6, 9, 12, 13, 16, 19, 20, 21, 22, 25, 28, 37, 38, 41, 42, 45, 46, 50, 52, 54, 56,58, 60, 61, 63, 64, 65, 66, 67, 71, 72, 79, 80, 81, 82, 87, 88, 89, 90, 91, 92, 94, 95, 96, 9}.

Функция сдвига регистра S так же принимает несколько параметров: CLOCK\_S(S, input\_bit\_s, control\_bit\_s). Здесь S – это указатель на регистр, а input\_bit\_s, control\_bit\_s – специальные биты. Сдвиг регистра S на псевдокоде можно выглядит следующим образом:

CLOCK\_S(S, input\_bit\_s, control\_bit\_s)

feedback\_bit = s99 ⊕ input\_bit\_s;

для 1 ≤ i ≤ 98,

ŝi = si-1 ⊕ ((si ⊕ COMP0i)&(si+1 ⊕ COMP1i)); ŝ0 = 0; ŝ99 = s98;

если control\_bit\_s = 0, то

для 0 ≤ i ≤ 99, s'i = ŝi ⊕ (FB0i & feedback\_bit);

если control\_bit\_s = 1, то

для 0 ≤ i ≤ 99, s'i = ŝi ⊕ (FB1i & feedback\_bit),

где s0, s1, …, s99 –биты регистра S до сдвига,

ŝ0, ŝ1, …, ŝ99 – промежуточные значения битов регистра S,

s'0, s'1, …, s'99 – биты регистра S после сдвига.

Массивы COMP0, COMP1, FB0 и FB1, в зависимости от раунда, определяются как показано в таблице 1.

Таблица 1. Массивы значений регистра S

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** | **16** | **17** | **18** | **19** | **20** | **21** | **22** | **23** | **24** |
| **COMP0i** |  | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| **COMP1i** |  | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| **FB0i** | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| **FB1i** | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **i** | **25** | **26** | **27** | **28** | **29** | **30** | **31** | **32** | **33** | **34** | **35** | **36** | **37** | **38** | **39** | **40** | **41** | **42** | **43** | **44** | **45** | **46** | **47** | **48** | **49** |
| **COMP0i** | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| **COMP1i** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| **FB0i** | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| **FB1i** | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **i** | **50** | **51** | **52** | **53** | **54** | **55** | **56** | **57** | **58** | **59** | **60** | **61** | **62** | **63** | **64** | **65** | **66** | **67** | **68** | **69** | **70** | **71** | **72** | **73** | **74** |
| **COMP0i** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| **COMP1i** | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| **FB0i** | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **FB1i** | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **i** | **75** | **76** | **77** | **78** | **79** | **80** | **81** | **82** | **83** | **84** | **85** | **86** | **87** | **88** | **89** | **90** | **91** | **92** | **93** | **94** | **95** | **96** | **97** | **98** | **99** |
| **COMP0i** | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |  |
| **COMP1i** | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |  |
| **FB0i** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| **FB1i** | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |

Порядок работы алгоритма регулируется функцией управления. Функция принимает 4 параметра: CLOCK\_KG(R, S, mixing, input\_bit). Здесь R и S указатели на регистры R и S соответственно, а mixing и input\_bit являются служебными битами. На псевдокоде данную функцию можно записать так:

CLOCK\_KG(R, S, mixing, input\_bit)

control\_bit\_r = s34 ⊕ r67;

control\_bit\_s = s67 ⊕ r33;

если mixing = TRUE, тогда input\_bit\_r = input\_bit ⊕ s50,

а если mixing = FALSE , тогда input\_bit\_r = input\_bit;

input\_bit\_s = input\_bit;

CLOCK\_R (R, input\_bit\_r, control\_bit\_r);

CLOCK\_S (S, input\_bit\_s, control\_bit\_s).

На основании вышеописанных преобразований при начале работы алгоритма происходит загрузка секретного ключа и вектора инициализации в регистры R и S соответственно. На псевдокоде данную функцию можно записать так:

(r0, r1, …, r99)= (0, …, 0);

(s0, s1, …, s99) = (0, …, 0);

для 0 ≤ i ≤ длина IV -1,

CLOCK\_KG (R, S, mixing = TRUE, input\_bit = IV i);

для 0 ≤ i ≤ 79,

CLOCK\_KG (R, S, mixing = TRUE, input\_bit = Ki);

для 0 ≤ i ≤ 99,

CLOCK\_KG (R, S, mixing =TRUE, input\_bit = 0),

где Ki – i-й бит секретного ключа,

IV i – i-й бит вектора инициализации.

Функция генерирования ключевой последовательности на псевдокоде выглядит следующим образом:

для 0 ≤ i ≤ L − 1,

zi = r0 ⊕ s0;

CLOCK\_KG (R, S, mixing = FALSE, input\_bit = 0),

где L – длина входного потока,

zi – i-й бит ключевой последовательности.

**1.7.2 Функция шифрования/расшифровывания**

В результате работы алгоритма генерируется уникальная ключевая последовательность, которая используется для операции побитового сложения по модулю 2 с каждым байтом входного потока. Вследствие чего на выходе получается зашифрованный поток данных.

Функция расшифровывания аналогична функции шифрования.

**Глава 2. Библиотека estream.h**

Последняя публикация алгоритмов-победителей проекта «eSTREAM» датируется январем 2012 года.

Организаторы представили результаты конкурса как разрозненные проекты, независящие друг от друга. За тем лишь исключением что функции, использующиеся в шифрах, называются одинаково. Но организаторами не было представлено единого механизма, который бы позволил использовать все разработки в рамках единого проекта. Сторонним разработчикам предлагается самим выбрать нужный алгоритм и присоединить к уже более большим проектам.

Поэтому было принято решение не только реализовать все 7 криптографических шифров, но и предоставить сторонним разработчикам программного обеспечения удобный интерфейс для облегчения интеграции алгоритмов шифрования в свои проекты и понимания логики работы разработанной библиотеки.

**2.1 Структура библиотеки estream.h**

Библиотека получила название estream.h в честь проекта, который подарил миру новые криптографические алгоритмы. Она была разработана на языке программирования СИ.

Структуру библиотеки можно посмотреть на рисунке 1.

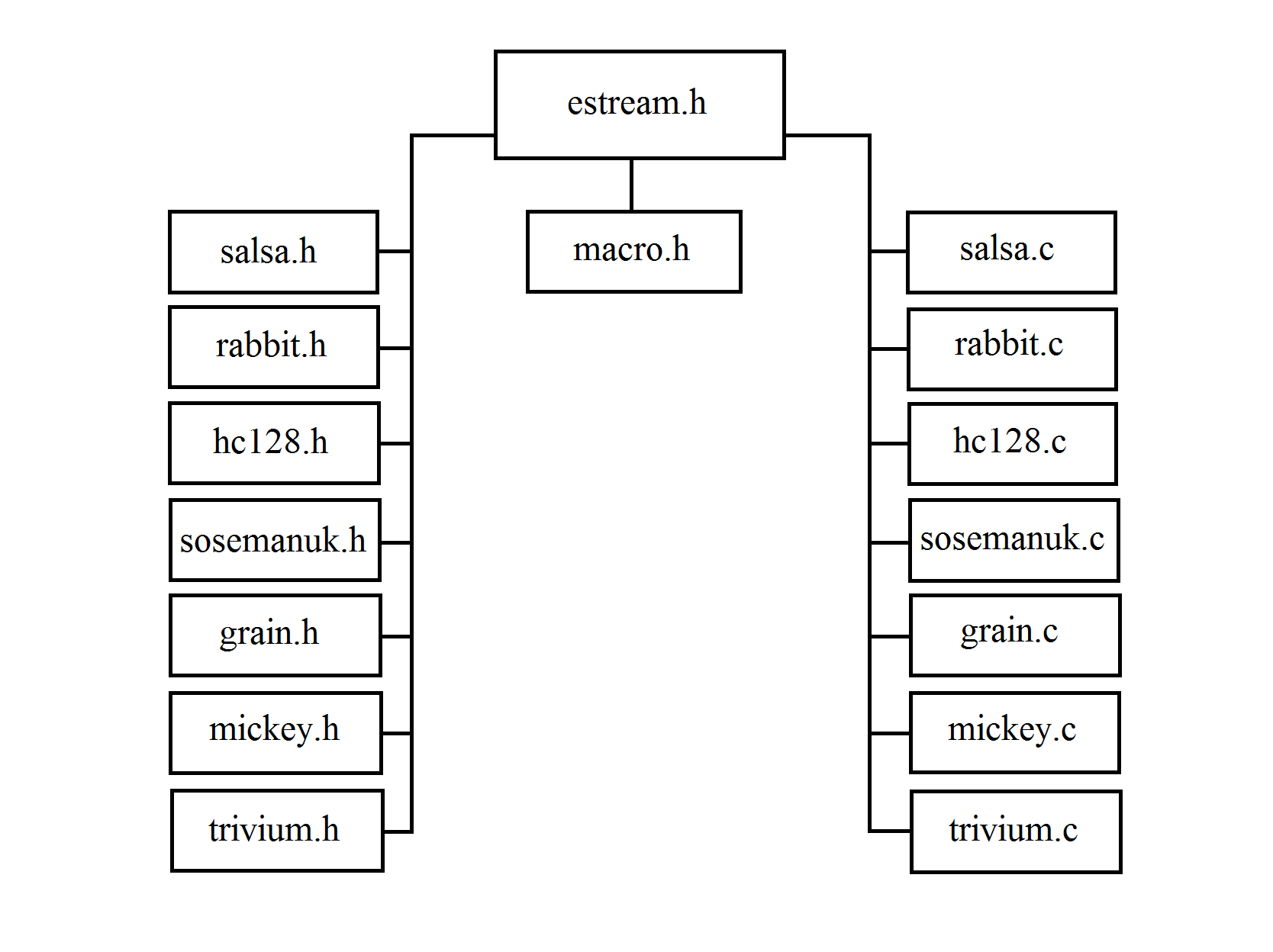


Рисунок 1 – Структура библиотеки estream.h

Как видно из рисунка 1, библиотека насчитывает 16 файлов. 9 заголовочных файлов: estream.h, macro.h, salsa.h, rabbit.h, hc128.h, sosemanuk.h, grain.h, mickey.h, trivium.h. И 7 исходных файлов: salsa.c, rabbit.c, hc128.c, sosemanuk.c, grain.c, mickey.c, trivium.c.

Основным файлом всего проекта является заголовочный файл estream.h, который необходимо разработчикам подключать к своей работе. Файл estream.h содержит инструкции, которые подгружают остальные заголовочные файлы. Исходный код заголовочного файла estream.h приведен в листинге 1.

#ifndef ESTREAM\_H

#define ESTREAM\_H

#include «salsa.h»

#include «rabbit.h»

#include «hc128.h»

#include «sosemanuk.h»

#include «grain.h»

#include «mickey.h»

#include «trivium.h»

#endif

Листинг 1 – Исходный код заголовочного файла estream.h

Подгружаемые в estream.h файлы содержат базовый набор функций доступных разработчикам и специально сформированные структуры данных. Все подгружаемые заголовочные файлы называются в зависимости от алгоритма шифрования, который они описывают. Например, файл salsa.h содержит исходный код, описывающий криптографический шифр Salsa.

Каждый заголовочный файл (header-файл) содержит описание структуры данных, которая облегчает взаимодействие разработчика с функциями алгоритмов. Структура данных представляет собой объединение нескольких переменных. В листинге 2 приведен исходный код структуры данных для алгоритма Salsa (header-файл salsa.h).

struct salsa\_context {

int keylen;

int ivlen;

uint8\_t key[32];

uint8\_t iv[16]

uint32\_t x[16];

}

Листинг 2 – Исходный код структуры данных salsa\_context

Для каждого алгоритма в соответствующем header-файле заведена своя структура данных. Причем количество полей в структуре данных в каждом алгоритме разное. Общими для всех шифров в структурах данных являются следующие поля структур:

1. keylen – длина секретного ключа в байтах;

2. ivlen – длина вектора инициализации в байтах;

3. key – массив, содержащий биты секретного ключа;

4. iv – массив, содержащий биты вектора инициализации.

Использование структур данных позволяет облегчать взаимодействие разработчика с библиотекой. Так как один раз заполнив структуру нужными значениями, разработчику нет необходимости заботиться о том, что происходит с этими значениями в процессе работы алгоритма.

Так же каждый header-файл содержит ссылки на три функции: \*\_set\_key\_and\_iv, \*\_crypt, \*\_test\_vectors (\* – обозначает один из семи алгоритмов шифрования).

Функция \*\_set\_key\_and\_iv(struct \*ctx, const uint8\_t \*key, const int keylen, const uint8\_t \*iv, const int ivlen) предназначена для заполнения структуры данных секретным ключом и вектором инициализации. Перед заполнением структуры происходит инициализация (заполнение нулями) структуры. Это необходимо для устранения попадания «мусора» в поля структуры.

Функция принимает 5 аргументов: указатель на структуру данных, указатель на секретный ключ, длину секретного ключа, указатель на вектор инициализации, длину вектора инициализации.

При успешном выполнении функция возвращает 0, в случае ошибки – -1.

Функция \*\_crypt(struct \*ctx, const uint8\_t \*buf, uint32\_t buflen, uint8\_t \*out) шифрует/расшифровывает входной поток данных. Функция принимает на вход 4 аргумента: указатель на структуру данных, указатель на входной поток данных, длину входного потока в байтах и указатель на выходной массив данных, в который будет помещен результат шифрования/расшифровывания. Данная функция ничего не возвращает.

Функция \*\_test\_vectors(struct \*ctx) предназначена для вывода ключевого потока, который генерируется за одну итерацию алгоритма. Длина ключевого потока зависит от алгоритма. На вход функции подается указатель на уже заполненную ключом и вектором инициализации структуру данных. Функция ничего не возвращает.

Заголовочный файл macro.h содержит описания некоторых математических преобразований, которые являются общими для нескольких криптографических алгоритмов. Подобный подход позволил уменьшить дублирование исходного кода. В header-файле macro.h содержатся макросы для обеспечения кроссплатформенности библиотеки и макросы циклических сдвигов машинных слов.

Библиотека estream.h поддерживает два основных порядка байтов: big endian и little endian. Big endian (порядок от старшего к младшему) – запись машинного слова начинается со старшего разряда и заканчивается младшим. Подобный порядок байтов поддерживается процессорами IBM 360/370/390, MIPS, SPARC и другими моделями процессоров.

Little endian (порядок от младшего к старшему) – запись машинного слова начинается с младшего разряда и заканчивается старшим. Подобный порядок байтов впервые стала использовать компания Intel, разработавшая процессоры с х86 архитектурой.

Если же в процессоре используется иной порядок байтов, что маловероятно, то библиотека выдаст ошибку «unsupported byte order» (порядок байт не поддерживается).

Наряду с заголовочными файлами библиотека содержит 7 файлов описания. Файлы описания содержат исходный программный код на языке программирования СИ, который описывает каждый криптографический алгоритм и функции доступные в header-файле. Файл описания носит такое же имя, как и заголовочный файл только с расширением .c. Например, пара: header-файл salsa.h и его файл описание salsa.c.

Разработанная библиотека estream.h предоставляет удобный интерфейс взаимодействия между разработчиком и криптографическими алгоритмами.

**2.2 Механизмы модернизации**

Главным достоинством потоковых криптографических алгоритмов является высокая скорость шифрования информации. Поэтому не достаточно просто реализовать шифр, а необходимо предложить такую реализацию, которая бы обеспечила высокую производительность алгоритма и смогла бы работать на различных архитектурах и платформах.

Подобный подход был взят за основу при разработке библиотеки. Данный раздел будет посвящен механизмам и способам модернизации алгоритмов для увеличения скоростных показателей, достижения кроссплатформенности при стабильной и корректной работе шифров.

**2.2.1 Использование макросов**

Макрос (в терминологии языка программирования СИ) – символьное имя, заменяемое при обработке препроцессором на последовательность программных инструкций.

Зачастую макросы используются для уменьшения дублирования программного кода, либо для увеличения понимания исходного кода, когда макросом заменяется какая-либо часть сложной составной инструкции.

Когда препроцессор, при обработке файла с исходным кодом, встречает символьное имя, которое определено с помощью директивы #define, то он подставляет значение макроса вместо символьного имени.

Подобное поведение препроцессора можно использовать для увеличения скорости работы алгоритма. Вместо многократного использования функций, где время тратится на вызов функции, можно использовать макрос. При обработке файла препроцессор сам подставит необходимый программный код. Такой подход позволяет получить выигрыш по времени, если набирается большое количество вызовов функции.

Например, в алгоритме HC128 (hc128.c) есть макрос GENERATE\_P, его исходный код приведен в листинге 3. Данный макрос производит генерацию ключевого потока.

#define GENERATE\_P(ctx, a, b, c, d, e, f, res) {

uint32\_T res1, res2;

G1(ctx->x[e], ctx->x[d], ctx->w[b], res1);

H1(ctx, ctx->x[f], res2);

ctx->w[a] += res1;

ctx->x[c] = ctx->w[a];

res = U32TO32((res2 ^ ctx->w[a]));

Листинг 3 – Исходный код макроса GENERATE\_P

При шифровании 512 байт информации данный макрос вызывается 32 раза. Использование макроса в данном случае дает значительный выигрыш в скорости, чем вызов аналогичной функции 32 раза.

При больших объемах данных выигрыш в скорости становится все больше и больше.

**2.2.2 Использование битовых операций**

Язык программирования СИ представляет удобный инструмент для работы с битами. В рамках разработки библиотеки использовались следующие битовые операции: левый битовый сдвиг, правый битовый сдвиг, побитовое отрицание, побитовое сложение и побитовое умножение.

Использование битовых операций позволяет увеличить производительность работы алгоритмов, так как все операции производятся непосредственно с битами машинного слова. Яркий пример комплексного использования битовых операций иллюстрируется в исходном коде макроса U32TO32 (описан в файле macro.h), который в зависимости от порядка байтов, выполняет заданное преобразование. Исходный код макроса приведен в листинге 4.

#if \_\_BYTE\_OORDER == \_\_BIG\_ENDIAN

#define U32TO32(x)

((x<<24) | ((x<<8) & (0xFF0000) | ((x>>8) & 0xFF00) | (x>>24))

#elif \_\_BYTE\_ORDER == \_\_LITTLE\_ENDIAN

#define U32TO32(x) (x)

#else

#error unsupported byte order

#endif

Листинг 4 – Исходный код макроса U32TO32

В данном примере используются битовые сдвиги, побитовые умножения и сложения, которые производят поворот машинного слова. Данная конструкция практически не затрачивает время на поворот машинного слова в виду того, что побитовые операции являются операциями низкого уровня.

2.3 Модернизация алгоритмов

В ходе изучения официальной документации по криптографическим алгоритмам и

Список используемых источников

<http://www.panasenko.ru/Articles/63/63.html>

http://www.itsec.ru/articles2/Oborandteh/estream\_ditya\_lohnesskogo\_chudovisha