Министерство образования Республики Беларусь Учреждение образования
Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

ОТЧЕТ

по лабораторной работе №2

Симметричная криптография. СТБ 34.101.31-2011

Выполнил:
студент гр. 653503
Лисковец Б.Н.

Проверил:
Артемьев В.С.

Минск 2019

Введение

СТБ 34.101.31 – блочный шифр с 256-битным ключом и 8 циклами криптопреобразований, оперирующий с 128-битными словами. Криптографические алгоритмы стандарта построены на основе базовых режимов шифрования блоков данных. Все алгоритмы стандарта делятся на 8 групп:

- алгоритмы шифрования в режиме простой замены;
- алгоритмы шифрования в режиме сцепления блоков;
- алгоритмы шифрования в режиме гаммирования с обратной связью;
- алгоритмы шифрования в режиме счётчика;
- алгоритм выработки имитовставки;
- алгоритмы одновременного шифрования и имитозащиты данных;
- алгоритмы одновременного шифрования и имитозащиты ключей;
- алгоритм хэширования;

Первые четыре группы предназначены для обеспечения безопасного обмена сообщениями. Каждая группа включает алгоритм зашифрования и алгоритм расшифрования на секретном ключе. Стороны, располагающие общим ключом, могут организовать обмен сообщениями путём их зашифрования перед отправкой и расшифрования после получения. В режимах простой замены и сцепления блоков шифруются сообщения, которые содержат хотя бы один блок, а в режимах гаммирования с обратной связью и счётчика — сообщения произвольной длины.

Задание: Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма СТБ 34.101.31-2011 в различных режимах.

Алгоритм

Шифрование блока

Входные данные и выходные данные

Входными данными алгоритмов зашифрования и расшифрования являются блок $X \in \{0,1\}^{128}$ и ключ $\theta \in \{0,1\}^{256}$.

Выходными данными является блок $Y \in \{0,1\}^{128}$ — результат зашифрования либо расшифрования слова X на ключе $\theta : Y = F_\theta(X)$ либо $Y = F_\theta^{-1}(X)$.

Входные данные для шифрования подготавливаются следующим образом:

- Слово X записывается в виде $X = X_1 \| X_2 \| X_3 \| X_4, X_i \in \{0,1\}^{32}$.
- Ключ θ записывается в виде $\theta = \theta_1 \| \theta_2 \| \theta_3 \| \theta_4 \| \theta_5 \| \theta_6 \| \theta_7 \| \theta_8, \theta_i \in \{0,1\}^{32}$ и определяются тактовые ключи
$K_1 = \theta_1, K_2 = \theta_2, K_3 = \theta_3, K_4 = \theta_4, K_5 = \theta_5, K_6 = \theta_6, K_7 = \theta_7, K_8 = \theta_8, K_9 = \theta_1, \ldots\ldots, K_{56} = \theta_8.$

Обозначения и вспомогательные преобразования

Преобразование $G_r : \{0,1\}^{32} \to \{0,1\}^{32}$ ставит в соответствие слову $u = u_1 \| u_1 \| u_2 \| u_3 \| u_4, u_i \in \{0,1\}^8$, слово

$G_r(u) = RotHi^r(H(u_1) \| H(u_2) \| H(u_3) \| H(u_4)).$

$RotHi^r$ циклический сдвиг влево на r бит.

$H(u)$ операция замены 8-битной входной строки подстановкой с рисунка 1.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	B1	94	BA	C8	0A	08	F5	3B	36	6D	00	8E	58	4A	5D	E4
1	85	04	FA	9D	1B	B6	C7	AC	25	2E	72	C2	02	FD	CE	0D
2	5B	E3	D6	12	17	B9	61	81	FE	67	86	AD	71	6B	89	0B
3	5C	B0	C0	FF	33	C3	56	B8	35	C4	05	AE	D8	E0	7F	99
4	E1	2B	DC	1A	E2	82	57	EC	70	3F	CC	F0	95	EE	8D	F1
5	C1	AB	76	38	9F	E6	78	CA	F7	C6	F8	60	D5	BB	9C	4F
6	F3	3C	65	7B	63	7C	30	6A	DD	4E	A7	79	9E	B2	3D	31
7	3E	98	B5	6E	27	D3	BC	CF	59	1E	18	1F	4C	5A	B7	93
8	E9	DE	E7	2C	8F	0C	0F	A6	2D	DB	49	F4	6F	73	96	47
9	06	07	53	16	ED	24	7A	37	39	CB	A3	83	03	A9	8B	F6
A	92	BD	9B	1C	E5	D1	41	01	54	45	FB	C9	5E	4D	0E	F2
B	68	20	80	AA	22	7D	64	2F	26	87	F9	34	90	40	55	11
C	BE	32	97	13	43	FC	9A	48	A0	2A	88	5F	19	4B	09	A1
D	7E	CD	A4	D0	15	44	AF	8C	A5	84	50	BF	66	D2	E8	8A
E	A2	D7	46	52	42	A8	DF	B3	69	74	C5	51	EB	23	29	21
F	D4	EF	D9	B4	3A	62	28	75	91	14	10	EA	77	6C	DA	1D

Рисунок 1 – Преобразование H

Подстановка $H : \{0,1\}^8 \to \{0,1\}^8$ задается фиксированной таблицей. В таблице используется шестнадцатеричное представление слов $u \in \{0,1\}^8$

$\boxplus$ и $\boxminus$ операции сложения и вычитания по модулю 2^{32}

Зашифрование

Для зашифрования блока X на ключе θ выполняются следующие шаги:

1. Установить $a \leftarrow X_1, b \leftarrow X_2, c \leftarrow X_3, d \leftarrow X_4$.
2. Для i = 1,2,… ,8 выполнить:

   1) $b \leftarrow b \oplus G_5(a \boxplus K_{7i-6})$;
   2) $c \leftarrow c \oplus G_{21}(d \boxplus K_{7i-5})$;
   3) $a \leftarrow a \boxminus G_{13}(b \boxplus K_{7i-4})$;
   4) $e \leftarrow G_{21}(b \boxplus c \boxplus K_{7i-3}) \oplus \langle i \rangle_{32}$;
   5) $b \leftarrow b \boxplus e$;
   6) $c \leftarrow c \boxminus e$;
   7) $d \leftarrow d \boxplus G_{13}(c \boxplus K_{7i-2})$;
   8) $b \leftarrow b \oplus G_{21}(a \boxplus K_{7i-1})$;
   9) $c \leftarrow c \oplus G_5(d \boxplus K_{7i})$;
   10) $a \leftrightarrow b$;
   11) $c \leftrightarrow d$;
   12) $b \leftrightarrow c$.
3. Установить $Y \leftarrow b \| d \| a \| c$.
4. Возвратить Y.

Расшифрование

Для расшифрования блока X на ключе θ выполняются следующие шаги:

1. Установить $a \leftarrow X_1, b \leftarrow X_2, c \leftarrow X_3, d \leftarrow X_4$.
2. Для i = 8,7,… ,1 выполнить:

   1) $b \leftarrow b \oplus G_5(a \boxplus K_{7i})$;
   2) $c \leftarrow c \oplus G_{21}(d \boxplus K_{7i-1})$;
   3) $a \leftarrow a \boxminus G_{13}(b \boxplus K_{7i-2})$;

4) $e \leftarrow G_{21}(b \boxplus c \boxplus K_{7i-3}) \oplus \langle i \rangle_{32}$;
5) $b \leftarrow b \boxplus e$;
6) $c \leftarrow c \boxminus e$;
7) $d \leftarrow d \boxplus G_{13}(c \boxplus K_{7i-4})$;
8) $b \leftarrow b \oplus G_{21}(a \boxplus K_{7i-5})$;
9) $c \leftarrow c \oplus G_5(d \boxplus K_{7i-6})$;
10) $a \leftrightarrow b$;
11) $c \leftrightarrow d$;
12) $a \leftrightarrow d$.

3. Установить $Y \leftarrow c \parallel a \parallel d \parallel b$.
4. Возвратить Y.

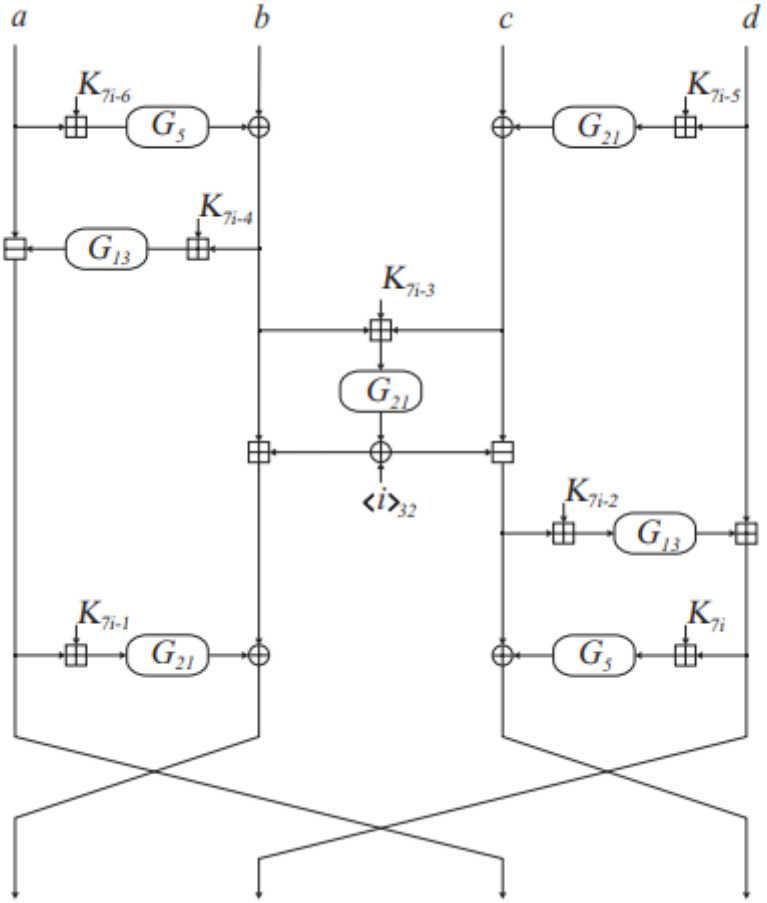

Рисунок 2 – Вычисления на i такте шифрования

Выработка имитовставки

Входные данные

- Исходное сообщение произвольной длины, представленное в виде битовой последовательности $X \in \{0,1\}^*$. Если X - непустое слово, то записать его в виде:
  $X = X_1 \| X_2 \| \ldots \| X_n, |X_1| = |X_2| = \ldots = |X_{n-1}| = 128, 0 < |X_n| \leq 128$.
  Если же X - пустое, то $n = 1$ и $|X_1| = 0$.
- Ключ $\theta \in \{0,1\}^{256}$ - битовая последовательность длины 256.

Вспомогательные преобразования и переменные

- Преобразования $\phi_1, \phi_2 : \{0,1\}^{128} \to \{0,1\}^{128}$, которые действуют на слово $u = u_1 || u_2 || u_3 || u_4, u_i \in \{0,1\}^{32}$ - битовая последовательность длины 32 . При этом:

$$\phi_1(u) = u_2 || u_3 || u_4 || (u_1 \oplus u_2),$$

$$\phi_2(u) = (u_1 \oplus u_4) || u_1 || u_2 || u_3.$$

Отображение ψ, которое ставит в соответствие битовой последовательности длины меньше 128, слово длиной 128. Действует по правилу:

$$\psi(u) = u || 1 || 0^{127-|u|}.$$

- Вспомогательные переменные $r, s \in \{0,1\}^{256}$ - битовые последовательности длины 128.

Алгоритм выработки имитовставки

1. Заполнить вспомогательную переменную s нулями: $s \leftarrow 0^{128}$ и установить результат шифрования s на данном ключе $\theta \in \{0,1\}^{256}$ в r: $r \leftarrow F_\theta(s)$.
2. Для каждого блока входного сообщения $i = 1, 2, \ldots, n-1$ выполнить: $s \leftarrow F_\theta(s \oplus X_i)$.
3. Если $|X_n| = 128$, то выполняем $s \leftarrow s \oplus X_n \oplus \phi_1(r)$, иначе $s \leftarrow s \oplus \psi(X_n) \oplus \phi_2(r)$.
4. Записать в T первые 64 бита слова $F_\theta(s)$: $T \leftarrow L_{64}(F_\theta(s))$.
5. Возвратить T.

Практическая часть

Зададим 128-битную синхропосылку и 256 битный ключ (рис. 3).

```
std::string key = "aijdbvkjabfnkjsnbd";
std::string sync = "some_sync_message_to_be_encrypted_for";
```

Рисунок 3 – Синхропосылка и ключ шифрования

Создадим файл с начальным текстом (рис. 4).

init_text.txt

```
Anyone who reads Old and Middle English literary texts will be familiar
with the mid-brown volumes of the EETS, with the symbol of Alfred's
jewel embossed on the front cover. Most of the works attributed to
King Alfred or to Aelfric, along with some of those by bishop Wulfstan
and much anonymous prose and verse from the pre-Conquest period, are
to be found within the Society's three series; all of the surviving
medieval drama, most of the Middle English romances, much religious
and secular prose and verse including the English works of John Gower,
Thomas Hoccleve and most of Caxton's prints all find their place in the
publications. Without EETS editions, study of medieval English texts
would hardly be possible.

```

Рисунок 4 – Начальный текст

Далее запустим программу и получим 2 файла: с зашифрованным и расшифрованным сообщениями (рис. 5 и рис. 6).

encrypted_text.txt

```
"dъ8ЦY=Ii€RS%T7y,Pw:Э+Ї.“SYNLэ°OIбESCЧCDC2ЬSUBЃ’»рзиЭМ<oMzKrETXDC4bJяюиѓ@nш/ETXЬлD
ЋГ+EACK1+BSTXШ-(cACKкESCTxлK›ETBч”VTЭSTXACK5k-§iSIVTDLET EЭkЃ8 JKшe¶~-SO09DC3a 3
ЄsinIsPAGSoBS±^BHY-ьГ_a@GSй;КЊмьЁOE!8ЛESCБЄыїЇ^a®P·DC1ENO-Dg—њaDo=dEЦO1c)ц+ENO E
```

Рисунок 5 – Зашифрованный текст

decrypted_text.txt

```
Anyone who reads Old and Middle English literary texts will be familiar
with the mid-brown volumes of the EETS, with the symbol of Alfred's
jewel embossed on the front cover. Most of the works attributed to
King Alfred or to Aelfric, along with some of those by bishop Wulfstan
and much anonymous prose and verse from the pre-Conquest period, are
to be found within the Society's three series; all of the surviving
medieval drama, most of the Middle English romances, much religious
and secular prose and verse including the English works of John Gower,
Thomas Hoccleve and most of Caxton's prints all find their place in the
publications. Without EETS editions, study of medieval English texts
would hardly be possible.

```

Рисунок 6 –Расшифрованный текст

Приложение А. Текст программы

belt.cpp

```
#include "belt.h"

void belt_init(uint8_t *ks, const uint8_t *k, std::size_t klen) {
 std::size_t i;
 switch (klen) {
 case 16:
  for (i = 0; i < 16; ++i) {
   ks[i + 0] = k[i];
   ks[i + 16] = k[i];
  }
  break;

 case 24:
  for (i = 0; i < 24; ++i) {
   ks[i] = k[i];
  }
  store32(ks + 24, load32(k + 0) ^ load32(k + 4) ^ load32(k + 8));
  store32(ks + 28, load32(k + 12) ^ load32(k + 16) ^ load32(k + 20));
  break;

 case 32:
  for (i = 0; i < 32; ++i) {
   ks[i] = k[i];
  }
  break;
 }
}

void belt_encrypt(uint8_t *out, const uint8_t *in, const uint8_t *ks) {
 uint32_t a = load32(in + 0);
 uint32_t b = load32(in + 4);
 uint32_t c = load32(in + 8);
 uint32_t d = load32(in + 12);

 uint32_t e;
 std::size_t i;
 uint32_t tmp;
 uint32_t key[8] = {0};

 for (i = 0; i < 8; ++i) {
  key[i] = load32(ks + (4 * i));
 }

 for (i = 0; i < 8; ++i) {
  b ^= G((a + key[KeyIndex[i][0]]), H, 5);
  c ^= G((d + key[KeyIndex[i][1]]), H, 21);
  a -= G((b + key[KeyIndex[i][2]]), H, 13);
  e = (G((b + c + key[KeyIndex[i][3]]), H, 21) ^ (uint32_t)(i + 1));
  b += e;
  c -= e;
  d += G((c + key[KeyIndex[i][4]]), H, 13);
  b ^= G((a + key[KeyIndex[i][5]]), H, 21);
  c ^= G((d + key[KeyIndex[i][6]]), H, 5);
  SWAP(a, b, tmp);
  SWAP(c, d, tmp);

  SWAP(b, c, tmp);
 }
 store32(out + 0, b);
 store32(out + 4, d);
```

```
  store32(out + 8, a);
  store32(out + 12, c);
}

void belt_decrypt(uint8_t *out, const uint8_t *in, const uint8_t *ks) {
  uint32_t a = load32(in + 0);
  uint32_t b = load32(in + 4);
  uint32_t c = load32(in + 8);
  uint32_t d = load32(in + 12);
  uint32_t e;
  std::size_t i;
  uint32_t tmp;
  uint32_t key[8] = {0};

  for (i = 0; i < 8; ++i) {
    key[i] = load32(ks + (4 * i));
  }

  for (i = 0; i < 8; ++i) {
    b ^= G((a + key[KeyIndex[7 - i][6]]), H, 5);
    c ^= G((d + key[KeyIndex[7 - i][5]]), H, 21);
    a -= G((b + key[KeyIndex[7 - i][4]]), H, 13);
    e = (G((b + c + key[KeyIndex[7 - i][3]]), H, 21) ^ (uint32_t)(7 - i + 1));
    b += e;
    c -= e;
    d += G((c + key[KeyIndex[7 - i][2]]), H, 13);
    b ^= G((a + key[KeyIndex[7 - i][1]]), H, 21);
    c ^= G((d + key[KeyIndex[7 - i][0]]), H, 5);
    SWAP(a, b, tmp);
    SWAP(c, d, tmp);
    SWAP(a, d, tmp);
  }
  store32(out + 0, c);
  store32(out + 4, a);
  store32(out + 8, d);
  store32(out + 12, b);
}
```

modes.cpp

```
#include "../include/modes.h"

// прямая замена
void encrypt_plain(uint8_t *outenc, const uint8_t *inenc, const uint8_t *ks,
                   const int32_t len) {
  uint8_t buffer[16] = {0};
  uint8_t outchunk[16] = {0};
  for (int i = 0; i < (len + 15) / 16; ++i) {
    store128(buffer, inenc + i * 16);
    belt_encrypt(outchunk, buffer, ks);
    store128(outenc + i * 16, outchunk);
  }
}

void decrypt_plain(uint8_t *outdec, const uint8_t *outenc, const uint8_t *ks,
                   const int32_t len) {
  uint8_t buffer[16] = {0};
  uint8_t outchunk[16] = {0};
  for (int i = 0; i < (len + 15) / 16; ++i) {
    store128(buffer, outenc + i * 16);
    belt_decrypt(outchunk, buffer, ks);
    store128(outdec + i * 16, outchunk);
  }
}

// сцепление блоков
```

```
void encrypt_block(uint8_t *outenc, const uint8_t *inenc, const uint8_t *ks,
                   const int32_t len, const uint8_t *s) {
  uint8_t buffer[16] = {0};
  uint8_t outchunk[16] = {0};
  uint8_t synhro[16] = {0};
  belt_encrypt(synhro, s, ks);
  for (int i = 0; i < (len + 15) / 16; ++i) {
    store128(buffer, inenc + i * 16);

    for (std::size_t j = 0; j < 16; ++j) {
      buffer[j] = buffer[j] ^ synhro[j];
    }
    belt_encrypt(outchunk, buffer, ks);

    store128(outenc + i * 16, outchunk);
    store128(synhro, outchunk);
  }
}

void decrypt_block(uint8_t *outdec, const uint8_t *outenc, const uint8_t *ks,
                   const int32_t len, const uint8_t *s) {
  uint8_t buffer[16] = {0};
  uint8_t outchunk[16] = {0};
  uint8_t synhro[16] = {0};
  belt_encrypt(synhro, s, ks);
  for (int i = 0; i < (len + 15) / 16; ++i) {
    store128(buffer, outenc + i * 16);

    belt_decrypt(outchunk, buffer, ks);
    for (std::size_t j = 0; j < 16; ++j) {
      outchunk[j] = outchunk[j] ^ synhro[j];
    }

    store128(outdec + i * 16, outchunk);
    store128(synhro, buffer);
  }
}

// гаммирование
void encrypt_gamming(uint8_t *outenc, const uint8_t *inenc, const uint8_t *ks,
                     const int32_t len, const uint8_t *s) {
  uint8_t buffer[16] = {0};
  uint8_t outchunk[16] = {0};
  uint8_t synhro[16] = {0};
  uint8_t ns[16] = {0};
  store128(ns, s);
  for (int i = 0; i < (len + 15) / 16; ++i) {
    belt_encrypt(synhro, ns, ks);
    store128(buffer, inenc + i * 16);

    for (std::size_t j = 0; j < 16; ++j) {
      outchunk[j] = buffer[j] ^ synhro[j];
    }

    store128(outenc + i * 16, outchunk);
    store128(ns, outchunk);
  }
}

void decrypt_gamming(uint8_t *outdec, const uint8_t *outenc, const uint8_t *ks,
                     const int32_t len, const uint8_t *s) {
  uint8_t buffer[16] = {0};
  uint8_t outchunk[16] = {0};
  uint8_t synhro[16] = {0};
  uint8_t ns[16] = {0};
  store128(ns, s);
```

```
  for (int i = 0; i < (len + 15) / 16; ++i) {
    belt_encrypt(synhro, ns, ks);
    store128(buffer, outenc + i * 16);

    for (std::size_t j = 0; j < 16; ++j) {
      outchunk[j] = buffer[j] ^ synhro[j];
    }

    store128(outdec + i * 16, outchunk);
    store128(ns, buffer);
  }
}

// счётчик
void encrypt_counter(uint8_t *outenc, const uint8_t *inenc, const uint8_t *ks,
                     const int32_t len, const uint8_t *s) {
  uint8_t buffer[16] = {0};
  uint8_t outchunk[16] = {0};
  uint8_t synhro[16] = {0};
  uint8_t ns[16] = {0};
  belt_encrypt(ns, s, ks);
  for (int i = 0; i < (len + 15) / 16; ++i) {
    belt_encrypt(synhro, ns, ks);
    square_plus(ns, synhro);
    store128(buffer, inenc + i * 16);

    for (std::size_t j = 0; j < 16; ++j) {
      outchunk[j] = buffer[j] ^ ns[j];
    }

    store128(outenc + i * 16, outchunk);
  }
}

void decrypt_counter(uint8_t *outdec, const uint8_t *outenc, const uint8_t *ks,
                     const int32_t len, const uint8_t *s) {
  encrypt_counter(outdec, outenc, ks, len, s);
}
```

Fileio.cpp

```
#include "../include/fileio.h"
#include <iostream>

std::vector<uint8_t> read_str_from_file(const std::string &input) {
  std::ifstream t(input);
  t.seekg(0, std::ios::end);
  size_t size = t.tellg();
  std::string buffer(size, ' ');
  t.seekg(0);
  t.read(&buffer[0], size);
  return {buffer.begin(), buffer.end()};
}

void write_str_to_file(const std::vector<uint8_t> &text,
                       const std::string &output) {
  std::ofstream t(output);
  t << std::string(text.begin(), text.end());
}

void encrypt_file(Mode m, const uint8_t *keyenc, const uint8_t *synhro,
                  const std::string &init_text_file,
                  const std::string &encrypted_text_file) {
  std::vector<uint8_t> inenc = read_str_from_file(init_text_file);
```

```
  std::vector<uint8_t> outenc(inenc.size() + 10, 0);
  uint8_t ks[32] = {0};

  belt_init(ks, keyenc, 32);
  switch (m) {
  case Mode::PLAIN:
    encrypt_plain(outenc.data(), inenc.data(), ks, inenc.size());
    break;
  case Mode::BLOCK:
    encrypt_block(outenc.data(), inenc.data(), ks, inenc.size(), synhro);
    break;
  case Mode::GAMMING:
    encrypt_gamming(outenc.data(), inenc.data(), ks, inenc.size(), synhro);
    break;
  case Mode::COUNTER:
    encrypt_counter(outenc.data(), inenc.data(), ks, inenc.size(), synhro);
    break;
  }

  write_str_to_file(outenc, encrypted_text_file);
}

void decrypt_file(Mode m, const uint8_t *keyenc, const uint8_t *synhro,
                  const std::string &encrypted_text_file,
                  const std::string &decrypted_text_file) {
  std::vector<uint8_t> outenc = read_str_from_file(encrypted_text_file);

  std::vector<uint8_t> outdec(outenc.size(), 0);
  uint8_t ks[32] = {0};

  belt_init(ks, keyenc, 32);
  switch (m) {
  case Mode::PLAIN:
    decrypt_plain(outdec.data(), outenc.data(), ks, outenc.size());
    break;
  case Mode::BLOCK:
    decrypt_block(outdec.data(), outenc.data(), ks, outenc.size(), synhro);
    break;
  case Mode::GAMMING:
    decrypt_gamming(outdec.data(), outenc.data(), ks, outenc.size(), synhro);
    break;
  case Mode::COUNTER:
    decrypt_counter(outdec.data(), outenc.data(), ks, outenc.size(), synhro);
    break;
  }

  write_str_to_file(outdec, decrypted_text_file);
}
```

utility.cpp

```
#include "utility.h"

uint32_t load32(const void *in) {
  const uint8_t *p = (const uint8_t *)in;
  return ((uint32_t)p[0] << 0) | ((uint32_t)p[1] << 8) |
         ((uint32_t)p[2] << 16) | ((uint32_t)p[3] << 24);
}

void store32(void *out, const uint32_t v) {
  uint8_t *p = (uint8_t *)out;
  p[0] = (uint8_t)(v >> 0);
  p[1] = (uint8_t)(v >> 8);
  p[2] = (uint8_t)(v >> 16);
  p[3] = (uint8_t)(v >> 24);
}
```

```
void store128(void *out, const uint8_t *ins) {
  uint8_t *p = (uint8_t *)out;
  for (std::size_t i = 0; i < 16; ++i) {
    p[i] = ins[i];
  }
}

void square_plus(void *out, const uint8_t *ins) {
  uint8_t *p = (uint8_t *)out;
  for (std::size_t i = 15;; i = (i + 15) % 16) {
    if (p[i] == (uint8_t)((1 << 8) - 1))
      p[i] = 0;
    else {
      ++p[i];
      break;
    }
  }
}
```