# Формулировка задания

Программный комплекс должен состоять из двух частей: программа для микроконтроллера и приложение для ПК. Программа для МК должна осуществлять хранение параметров криптосистемы (ключей), управление ключами в соответствии с поступающими от ПК командами, шифрование и расшифрование двоичных данных, поступающих с ПК, а также подписание и проверку подписи хэша, поступающего с ПК. Недопустима предварительная передача всего файла на МК с последующей обработкой и возвратом полностью сформированного зашифрованного/расшифрованного файла – обработка двоичных данных, поступающих с ПК, должна выполняться побайтно (пословно) или блоками (при соответствующем указании в варианте задания). При шифровании необходимо использовать функцию f(m, by) = m + by (mod p). Приложение для микроконтроллера должно позволять одновременно хранить в памяти EEPROM несколько троек параметров (a, p, x). При шифровании файла на контроллер должна передаваться пара (a, p), в ответном сообщении вместе с зашифрованным файлом должно возвращаться значение ay (одно на всё сообщение), допускается в качестве значения y использовать предопределённый набор однобайтных величин. При расшифровании вместе с телом файла на микроконтроллер должна передаваться тройка (a, p, ay) и микроконтроллер должен выполнять поиск секретного ключа по записям в EEPROM.

Приложение для ПК должно предоставлять пользователю интерфейс (допускается разработка консольного приложения) для управления параметрами криптосистемы (считывание параметров из памяти МК, генерация и запись ключей в память МК, удаление отдельных ключей), возможность ввода (выбора) имён файлов для считывания и записи двоичных данных, указания выполняемого действия (шифрование, расшифрование, вычисление хэша и электронной подписи, вычисление хэша и проверка электронной подписи).

Взаимодействие между устройствами должно осуществляться по интерфейсу UART- виртуальный COM-порт. Протокол передачи данных между ПК и МК разрабатывается самостоятельно. В протоколе необходимо предусмотреть передачу информации об ошибках от МК, например, при указании неизвестного открытого ключа в командах, не связанных с установкой ключей.

Индивидуальное задание: программа для МК должна осуществлять постоянное отображение открытых ключей криптосистемы, хранящихся в EEPROM, на семисегментном индикаторе. По прерыванию int0 должен циклически сменяться выводимый ключ, по прерыванию int1 должно производиться переключение между параметрами *a* и *p*.

# Схема лабораторной установки

Изображение выглядит как седзи, текст

Автоматически созданное описание

# Форматы пересылаемых данных (в оба направления)

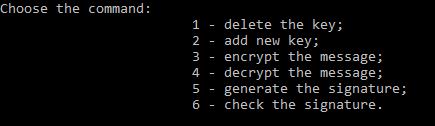
Данные, пересылаемые МК от ПК.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Удаление ключа | Номер запроса (1) | Первый байт параметра а | Второй байт параметра а | Первый байт параметра р | Второй байт параметра р | - | - | - | - | - |
| Добавление нового ключа | Номер запроса (2) | Первый байт параметра х | Второй байт параметра х | - | - | - |
| Зашифровка сообщения | Номер запроса (3) | Первый байт длины сообщения | Второй байт длины сообщения | Далее пересылается сообщение побайтово | … | … |
| Расшифровка сообщения | Номер запроса (4) | Первый байт а^y | Второй байт а^y | Первый байт длины сообщения | Второй байт длины сообщения | Далее пересылается сообщение побайтово |
| Генерация подписи | Номер запроса (5) | хэш | - | - | - | - |
| Проверка подписи | Номер запроса (6) | хэш | Первый байт s1 | Второй байт s1 | Первый байт s2 | Второй байт s2 |

Данные, пересылаемые ПК от МК.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 |  |  |  |
| Удаление ключа | Подтверждение удаления ключа | - | - | - | - |
| Добавление нового ключа | Подтверждение удаления ключа | - | - | - | - |
| Зашифровка сообщения | Подтверждение наличия в памяти полученного ключа | Первый байт а^y | Второй байт а^y | Далее пересылается зашифрованное сообщение побайтово | - |
| Расшифровка сообщения | Подтверждение наличия в памяти полученного ключа | Далее пересылается зашифрованное сообщение побайтово | - | - | - |
| Генерация подписи | Подтверждение наличия в памяти полученного ключа | Первый байт s1 | Второй байт s1 | Первый байт s2 | Второй байт s2 |
| Проверка подписи | Подтверждение, включающее в себя результата проверки подписи | - | - | - | - |

# Описание интерфейса программы для ПЭВМ (скриншот)



# Блок-схема алгоритма работы программы для МК

# Блок-схема алгоритма работы программы для ПЭВМ

# Временные диаграммы логических сигналов на портах МК (фрагмент)

В момент времени 4,75 с было произведено прерывание int0, перешли к выводу ключа p, теперь пол секунды на семисигментном индикаторе мигает имя параметра (“p.”), пол секунды – четырехзначное значение параметра в десятичной системе счисления;

в 9,2 с было произведено прерывание int1, перешли к выводу ключа p следующей пары ключей (а, р).

# Результаты работы

Реализовали программный комплекс, состоящий из двух частей: программа для микроконтроллера и приложение для ПК. Программа для МК осуществляет хранение параметров криптосистемы (ключей), управление ключами, шифрование и расшифрование двоичных данных, подписание и проверку подписи хэша в соответствии с заданием. Также МК осуществляет постоянное отображение открытых ключей криптосистемы, хранящихся в EEPROM, на семисегментном индикаторе. Реализована возможность просмотра всех хранящихся в памяти ключей. Приложение для ПК предоставляет пользователю интерфейс для управления параметрами криптосистемы, возможность ввода имён файлов для считывания и записи двоичных данных, указания выполняемого действия.

# Выводы по лабораторной работе

Ознакомились с методами взаимодействия между МК и ПК, реализовали метод взаимодействия по интерфейсу UART- виртуальный COM-порт. Разработали самостоятельно протокол передачи данных между ПК и МК с возможностью сообщения об ошибке.

# Приложение 1 Комментированный листинг программы для МК на языке Си

#define F\_CPU 8000000

#define UBRR\_C (F\_CPU / (9600 \* 16) - 1)

//#include <asf.h>

#include <math.h>

//#include <avr/iom32.h>

#include <avr/io.h>

#include <avr/eeprom.h>

#include <avr/interrupt.h>

#include <stdlib.h>

#include <util/delay.h>

char command\_number, // какой байт команды сейчас в обработке

request, // номер запроса от ПК:

// 1 - удаление ключа

// 2 - добавление ключа

// 3 - шифрование сообщения

// 4 - дешифрование сообщения

// 5 - вычисление хэша и электронной подписи

// 6 - вычисление хэша и проверка электронной подписи

cond, // при работе с ключами: 1/3/5 -> пересылается первый байт а/р/х;

// 2/4/6 -> пересылается второй байт

key\_counter,

message\_length,

m, // при работе с шированием: текущий байт сообщения

output\_mode, // при выводе на семисегметный индикатор:

// 1 - вывод имени а

// 2 - вывод значения а на PA3

// 3 - вывод значения а на PA2

// 4 - вывод значения а на PA1

// 5 - вывод значения а на PA0

// 6 - вывод имени р

// 7 - вывод значения p на PA3

// 8 - вывод значения p на PA2

// 9 - вывод значения p на PA1

// 10 - вывод значения p на PA0

numbers[10] = {0b00111111, // вывод чисел на семисигментные индикаторы

0b00000110,

0b01011011,

0b01001111,

0b01100110,

0b01101101,

0b01111101,

0b00000111,

0b01111111,

0b01101111},

letter\_a = 0b01011111, // вывод букв на семисигментные индикаторы

letter\_p = 0b01110011,

print\_key\_number, // номер выводимого на индикаторы ключа

error\_msg; // ответ мк после выполнения команды пк:

// 1 = выполнено; 0xFF = ошибка

unsigned int

var\_y, f, a, k, // для шифрования и дешифровки сообщения

hash\_, r, // для подписи. hash - один байт!

var\_delay, // для зарержки вывода на индикатор

work\_key[3], // ключ (а, р, х)

print\_key[2]; // а и р, выводимые на семисегментный индикатор

unsigned long int s1, s2, v1, v2, a\_pow\_y, b\_pow\_y;

char uart\_read()

{

while(!(UCSRA&(1<<RXC))) ;

return UDR;

}

void uart\_write(char c)

{

while (!(UCSRA&(1<<UDRE))) ;

UDR = c;

}

void EEPROM\_Write(unsigned int uiAddress, unsigned char ucData)

{

/\* Wait for completion of previous write \*/

while(EECR & (1<<EEWE))

;

/\* Set up address and data registers \*/

EEAR = uiAddress;

EEDR = ucData;

/\* Write logical one to EEMWE \*/

EECR |= (1<<EEMWE);

/\* Start eeprom write by setting EEWE \*/

EECR |= (1<<EEWE);

}

unsigned char EEPROM\_Read(unsigned int uiAddress)

{

/\* Wait for completion of previous write \*/

while(EECR & (1<<EEWE))

;

/\* Set up address register \*/

EEAR = uiAddress;

/\* Start eeprom read by writing EERE \*/

EECR |= (1<<EERE);

/\* Return data from data register \*/

return EEDR;

}

void key\_element() // все int передаются двумя словами (побайтно)

{

if (cond == 1)

{

//work\_key[0] = uart\_read() - '0';

work\_key[0] = uart\_read();

work\_key[0] = work\_key[0] << 8;

}

else if (cond == 2)

{

char temp;

//temp = uart\_read() - '0';

temp = uart\_read();

work\_key[0] += temp;

}

else if (cond == 3)

{

work\_key[1] = uart\_read();

//work\_key[1] = uart\_read() - '0';

work\_key[1] <<= 8;

}

else if (cond == 4)

{

char temp;

//temp = uart\_read() - '0';

temp = uart\_read();

work\_key[1] += temp;

}

else if (cond == 5)

{

//work\_key[2] = uart\_read() - '0';

work\_key[2] = uart\_read();

work\_key[2] <<= 8;

}

else if (cond == 6)

{

char temp;

//temp = uart\_read() - '0';

temp = uart\_read();

work\_key[2] += temp;

}

}

void sent\_confirmation()

{

uart\_write(error\_msg);

\_delay\_ms(100);

error\_msg = 1;

}

int find\_the\_key()

{

if (key\_counter == 0)

return -1;

unsigned int i = 0, a = 0, p = 0;

// ищем нужный ключ

while(1)

{

// ищем а

while(1)

{

a = EEPROM\_Read(i\*6);

a <<= 8;

a += EEPROM\_Read(i\*6+1);

if (a == work\_key[0])

break;

i++;

}

// если пара (а, р) правильная, выходим из цикла поиска

p = EEPROM\_Read(i\*6+2);

p <<= 8;

p += EEPROM\_Read(i\*6+3);

if (p == work\_key[1])

break;

i++;

if (i > key\_counter)

return -1;

}

work\_key[2] = EEPROM\_Read(i\*6+4);

work\_key[2] <<= 8;

work\_key[2] += EEPROM\_Read(i\*6+5);

return i;

}

void load\_the\_key()

{

print\_key[0] = EEPROM\_Read(print\_key\_number\*6);

print\_key[0] <<= 8;

print\_key[0] += EEPROM\_Read(print\_key\_number\*6+1);

print\_key[1] = EEPROM\_Read(print\_key\_number\*6+2);

print\_key[1] <<= 8;

print\_key[1] += EEPROM\_Read(print\_key\_number\*6+3);

}

void write\_the\_key()

{

EEPROM\_Write(key\_counter\*6, work\_key[0]>>8);

EEPROM\_Write(key\_counter\*6+1, work\_key[0]);

EEPROM\_Write(key\_counter\*6+2, work\_key[1]>>8);

EEPROM\_Write(key\_counter\*6+3, work\_key[1]);

EEPROM\_Write(key\_counter\*6+4, work\_key[2]>>8);

EEPROM\_Write(key\_counter\*6+5, work\_key[2]);

key\_counter++;

if (key\_counter == 1)

load\_the\_key();

command\_number = 0;

sent\_confirmation();

}

int prepare\_the\_key()

{

a\_pow\_y = work\_key[2];

return find\_the\_key();

}

void sent\_back\_int(unsigned int a)

{

uart\_write(a>>8);

\_delay\_ms(100);

uart\_write(a);

\_delay\_ms(100);

}

void sent\_back\_before\_encrypt()

{

long temp, i, j;

if (prepare\_the\_key() < 0)

{

error\_msg = 0xFF;

command\_number = 0;

sent\_confirmation();

return;

}

sent\_confirmation();

message\_length = a\_pow\_y; // костыль, чтоб можно было использовать prepare\_the\_key()

// выбираем случайное var\_y

//var\_y = rand() % work\_key[1];

var\_y = rand() % (work\_key[1]-3) + 1;

// вычисляем a^var\_y и b^var\_y mod p

/\*temp = 1;

i = var\_y;

while (i > 0)

{

temp \*= work\_key[0];

temp %= work\_key[1];

i--;

}

a\_pow\_y = temp;

\*/

i = 0;

a\_pow\_y = 1;

while (i < var\_y) {

a\_pow\_y = a\_pow\_y \* work\_key[0];

a\_pow\_y = a\_pow\_y % work\_key[1];

i++;

}

/\*

temp = 1;

i = work\_key[2];

while (i > 0)

{

temp \*= a\_pow\_y;

temp %= work\_key[1];

i--;

}

b\_pow\_y = temp;

\*/

b\_pow\_y = 1;

j = 0;

while (j < work\_key[2]) {

b\_pow\_y = b\_pow\_y \* a\_pow\_y;

b\_pow\_y = b\_pow\_y % work\_key[1];

j++;

}

// передаем ПК a^y mod p

sent\_back\_int(a\_pow\_y);

//sent\_back\_int(b\_pow\_y);

//sent\_back\_int(var\_y);

}

void encrypt\_message() //f = (m + b^y) mod p

{

//m = uart\_read() - '0';

m = uart\_read();

f = (char)((m + b\_pow\_y) % work\_key[1]);

//f += '0';

uart\_write(f);

\_delay\_ms(10);

if (command\_number == message\_length + 7)

command\_number = 0;

}

void decrypt\_message() //m = (f - b^y) mod p

{

long temp, i;

//f = uart\_read() - '0';

f = uart\_read();

/\*

temp = 1;

i = work\_key[2];

while (i > 0)

{

temp \*= a\_pow\_y;

temp %= work\_key[1];

i--;

}

b\_pow\_y = temp;

\*/

b\_pow\_y = 1;

i = 0;

while (i < work\_key[2]) {

b\_pow\_y = b\_pow\_y \* a\_pow\_y;

b\_pow\_y = b\_pow\_y % work\_key[1];

i++;

}

//sent\_back\_int(a\_pow\_y);

//sent\_back\_int(b\_pow\_y);

//\_delay\_ms(30);

long f\_ = f, b\_pow\_y\_ = b\_pow\_y;

m = (char)((f\_ - b\_pow\_y\_) % work\_key[1]);

uart\_write(m);

\_delay\_ms(10);

if (command\_number == message\_length + 9)

command\_number = 0;

}

void decrypt\_help1()

{

int i;

a\_pow\_y = work\_key[2];

i = prepare\_the\_key();

if (i < 0)

error\_msg = 0x05;

//else

// a\_pow\_y = work\_key[2];

work\_key[2] = 0;

}

void decrypt\_help2()

{

int i;

message\_length = work\_key[2];

i = find\_the\_key();

if (error\_msg > 1 || i < 0)

{

error\_msg = 0xFF;

command\_number = 0;

}

sent\_confirmation();

}

int gcd (int a, int b)

{

if (b == 0)

return a;

else

return gcd (b, a % b);

}

void calculate\_signature()

{

command\_number = 0;

long temp;

int i;

i = find\_the\_key();

if (i < 0)

{

error\_msg = 0xFF;

command\_number = 0;

sent\_confirmation();

return;

}

sent\_confirmation();

// сгенерировать случайное число r

r = rand() % (work\_key[1] - 3) + 2;

while ((work\_key[1] - 1) % r == 0)

r = rand() % (work\_key[1] - 3) + 2;

//sent\_back\_int(r);

// вычислить s1 = a^r mod p

temp = 1;

i = r;

while (i > 0)

{

temp \*= work\_key[0];

temp %= work\_key[1];

i--;

}

s1 = temp;

// передать s1

sent\_back\_int(s1);

// вычислить s2 = (hash\_ - x\*s1)\*(r^(-1)) mod (p-1)

i = 1;

if (work\_key[2]\*s1 > hash\_)

{

temp = work\_key[2]\*s1 - hash\_;

i = -1;

}

else

temp = hash\_ - work\_key[2]\*s1;

temp %= work\_key[1] - 1;

if (i == -1)

temp = work\_key[1] - 1 - temp;

int inv\_r = 1;

while (gcd(r, inv\_r) != 1 || inv\_r\*r%(work\_key[1] - 1) != 1)

{

inv\_r++;

if (inv\_r == work\_key[1] - 1)

{

inv\_r = 1;

break;

}

}

//sent\_back\_int(inv\_r);

s2 = (temp \* inv\_r) % (work\_key[1] - 1);

// передать s2

sent\_back\_int(s2);

}

void check\_signature()

{

long temp;

int i;

command\_number = 0;

if (find\_the\_key() < 0)

{

error\_msg = 0xFF;

command\_number = 0;

sent\_confirmation();

return;

}

// вычисляем v1 = a^hash\_ mod p

temp = 1;

i = hash\_;

while (i > 0)

{

temp \*= work\_key[0];

temp %= work\_key[1];

i--;

}

v1 = temp;

//sent\_back\_int(v1);

// вычисляем v2 = (a^x)^s1 \* s1^s2 mod p

temp = 1;

i = work\_key[2] \* s1;

while (i > 0)

{

temp \*= work\_key[0];

temp %= work\_key[1];

i--;

}

v2 = temp;

temp = 1;

i = s2;

while (i > 0)

{

temp \*= s1;

temp %= work\_key[1];

i--;

}

v2 \*= temp;

v2 %= work\_key[1];

//sent\_back\_int(v2);

// если v1 == v2, подпись верна, отвправляем ПК единицу, иначе 0x0F

if (v1 - v2 == 0)

error\_msg = 1;

else

error\_msg = 0x0F;

sent\_confirmation();

}

void clean()

{

work\_key[0] = 0;

work\_key[1] = 0;

work\_key[2] = 0;

}

void delete\_the\_key()

{

// удалить ключ:

// найти нужный ключ

int i, j;

i = find\_the\_key();

if (key\_counter == 0 || i < 0)

{

command\_number = 0;

error\_msg = 0xFF;

sent\_confirmation();

return;

}

// заменить на 0xFF

EEPROM\_Write(i\*6,0xFF);

EEPROM\_Write(i\*6+1,0xFF);

EEPROM\_Write(i\*6+2,0xFF);

EEPROM\_Write(i\*6+3,0xFF);

EEPROM\_Write(i\*6+4,0xFF);

EEPROM\_Write(i\*6+5,0xFF);

// сдвинуть все ключи ниже

i \*= 6;

for (j = i; j < key\_counter\*6; j++)

{

a = EEPROM\_Read(j+6);

EEPROM\_Write(j, a);

}

command\_number = 0;

sent\_confirmation();

}

void full\_buffer()

{

if (error\_msg == 0xFF)

return;

command\_number++;

if (command\_number > 7 && request == 3)

{

encrypt\_message();

return;

}

if (command\_number > 9 && request == 4)

{

decrypt\_message();

return;

}

switch (command\_number)

{

case 1: // запрос

//request = uart\_read() - '0';

request = uart\_read();

clean();

break;

case 2: // получаем первый байт a

cond = 1;

key\_element();

break;

case 3: // получаем второй байт a

cond = 2;

key\_element();

break;

case 4: // получаем первый байт р

cond = 3;

key\_element();

break;

case 5: // получаем второй байт p

cond = 4;

key\_element();

if (request == 1)

delete\_the\_key();

break;

case 6: // при работе в ключами (добавление) - получаем первый байт х

// при работе с шифрованием - получаем первый байт длины будущего сообщения

// при работе с дешифровкой - получаем первый байт a^y

// при работе с подписью - получаем байт хэша

cond = 5;

key\_element();

if (request > 4)

{

work\_key[2] >>= 8;

hash\_ = work\_key[2];

if (request == 5)

calculate\_signature();

}

break;

case 7: // при работе в ключами (добавление) - получаем второй байт х

// при работе с шифрованием - получаем второй байт длины будущего сообщения

// при работе с дешифровкой - получаем второй байт a^y

// при работе с проверкой подписи - получаем первый байт s1

if (request < 6)

{

cond = 6;

key\_element();

if (request == 2)

write\_the\_key();

else if (request == 3)

sent\_back\_before\_encrypt();

else if (request == 4)

decrypt\_help1();

/\*else if (request == 5)

{

if (prepare\_the\_key() < 0)

error\_msg = 0xFF;

message\_length = a\_pow\_y; // костыль, чтоб можно было пользоваться функцией сверху

}\*/

}

else

{

cond = 5;

key\_element();

}

break;

case 8: // при работе с дешифровкой - получаем первый байт длины будущего сообщения

// при работе с проверкой подписи - получаем второй байт s1

if (request == 6)

{

cond = 6;

key\_element();

s1 = work\_key[2];

work\_key[2] = 0;

}

else if (request == 4)

{

cond = 5;

key\_element();

}

break;

case 9: // при работе с дешифровкой - получаем второй байт длины будущего сообщения

// при работе с проверкой подписи - получаем первый байт s2

if (request == 6)

{

cond = 5;

key\_element();

}

else if (request == 4)

{

cond = 6;

key\_element();

decrypt\_help2();

}

break;

case 10: // при работе с проверкой подписи - получаем второй байт s2

cond = 6;

key\_element();

s2 = work\_key[2];

check\_signature();

break;

}

}

void print\_on\_port()

{

unsigned int temp;

switch (output\_mode)

{

case 1:

PORTA = 1;

PORTC = letter\_a;

break;

case 2:

PORTA = 0b00001000;

temp = print\_key[0] / 1000;

PORTC = numbers[temp];

break;

case 3:

PORTA = 0b00000100;

temp = print\_key[0] / 100;

temp %= 10;

PORTC = numbers[temp];

break;

case 4:

PORTA = 0b00000010;

temp = print\_key[0] / 10;

temp %= 10;

PORTC = numbers[temp];

break;

case 5:

PORTA = 1;

temp = print\_key[0];

temp %= 10;

PORTC = numbers[temp];

break;

case 6:

PORTA = 1;

PORTC = letter\_p;

break;

case 7:

PORTA = 0b00001000;

temp = print\_key[1] / 1000;

PORTC = numbers[temp];

break;

case 8:

PORTA = 0b00000100;

temp = print\_key[1] / 100;

temp %= 10;

PORTC = numbers[temp];

break;

case 9:

PORTA = 0b00000010;

temp = print\_key[1] / 10;

temp %= 100;

PORTC = numbers[temp];

break;

case 10:

PORTA = 1;

temp = print\_key[1];

temp %= 10;

PORTC = numbers[temp];

break;

}

}

void change\_print\_key()

{

if (key\_counter < 2)

return;

print\_key\_number++;

print\_key\_number %= key\_counter;

load\_the\_key();

output\_mode = 1;

//var\_delay = 4000;

print\_on\_port();

}

void change\_print\_letter()

{

output\_mode += 5;

output\_mode %= 10;

//var\_delay = 4000;

print\_on\_port();

}

void change\_print\_on\_port()

{

var\_delay--;

if (output\_mode == 1 || output\_mode == 6)

{

// задержать на секунду

if (var\_delay > 0)

return;

output\_mode++;

}

else

{

// var\_delay на каждый индикатор == 10,

// т. е. каждый десятый тик выводим новое число на индикатор;

if (var\_delay % 10 == 0)

{

output\_mode++;

if (output\_mode == 6)

output\_mode = 2;

else if (output\_mode == 11)

output\_mode = 7;

print\_on\_port();

}

if (var\_delay > 0)

return;

if (output\_mode < 6)

output\_mode = 1;

else

output\_mode = 6;

}

var\_delay = 4000;

print\_on\_port();

}

ISR(USART\_RXC\_vect)

{

// костыль, чтоб понять, срабатывает ли это прерывание

/\*output\_mode = 6;

var\_delay = 10000;

print\_on\_port();\*/

//

full\_buffer();

//

}

ISR(INT0\_vect)

{

change\_print\_key();

}

ISR(INT1\_vect)

{

change\_print\_letter();

}

ISR(TIMER0\_OVF\_vect)

{

change\_print\_on\_port();

//

}

int main()

{

DDRA=0xFF;

DDRB=0xFF;

DDRC=0xFF;

//DDRD=0b11110010;

DDRD=0b11110000;

PORTA=0b00000001;

PORTB=0;

PORTC=letter\_a;

PORTD=0;

// работа с таймером

TIFR=0b00000001;

TCCR0=0b00000010; // последние 3 - коэффициент предделителя

TIMSK=0b00000001; // разрешение прерываний по таймеру0

//TIFR = (1<<TOV0);

// разрешение прерываний

MCUCR=0b00001111;

GICR=0b11000000;

//UART init

UBRRH = 0b10000000;

UBRRL = 0b11001111;

UCSRB = (1<<RXCIE)|(1<<RXEN)|(1<<TXEN); // Биты RXCIE(7), TXCIE(6), UDRIE(5) разрешеют прерывания по завершению приема, передачи и опустошению буфера

UCSRC = (1<<UCSZ0)|(1<<UCSZ1)|(1<<URSEL);

// тело программы

command\_number = 0;

request = 100;

cond = 0;

key\_counter = 0;

message\_length = 0;

m = 0;

var\_y = 0;

f = 0;

a = 0;

k = 0;

a\_pow\_y = 0;

b\_pow\_y = 0;

output\_mode = 1;

print\_key\_number = 0;

var\_delay = 4000;

error\_msg = 1;

sei();

while (1)

{ ; }

return 0;

}

**Приложение 2**Комментированный листинг программы для ПЭВМ на языке Си

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <Windows.h>

#include <conio.h>

#include <stdlib.h>

HANDLE port;

DCB com\_port;

FILE \*input\_msg, \*output\_msg;

void last\_msg(char \*str);

char command1();

void delete\_the\_key();

void add\_new\_key();

void encrypt\_message();

void decrypt\_message();

void generate\_signature();

void check\_signature();

void sent\_a\_and\_p();

void sent\_int(int a);

char get\_confirmation(char \*success\_msg);

char calculate\_hash();

int main()

{

// открыть порт

port = CreateFileA("COM4", GENERIC\_READ | GENERIC\_WRITE, NULL, 0, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (port == INVALID\_HANDLE\_VALUE)

{

last\_msg("COM-port connection is failed.\n");

return 0;

}

else printf("COM-port connection is established.\n");

// установить параметры com\_port

com\_port.DCBlength = sizeof(com\_port);

if (GetCommState(port, &com\_port) == 0)

last\_msg("COM-port state error.\n");

com\_port.BaudRate = CBR\_1200;

com\_port.Parity = NOPARITY;

com\_port.ByteSize = 8;

com\_port.StopBits = ONESTOPBIT;

if (SetCommState(port, &com\_port) == 0)

last\_msg("COM-port state error.\n");

// начинаем взаимодействие с пользователем

char command\_number;

while (1)

{

command\_number = command1();

switch (command\_number)

{

case 1:

delete\_the\_key();

break;

case 2:

add\_new\_key();

break;

case 3:

encrypt\_message();

break;

case 4:

decrypt\_message();

break;

case 5:

generate\_signature();

break;

case 6:

check\_signature();

break;

}

}

return 0;

}

void delete\_the\_key()

{

// узнать у пользователя, какой ключ он хочет удалить

printf("Print A and P parameteres of the key you want to delete (use space to separate parameteres).\n");

// отправить МК значения а и р

sent\_a\_and\_p();

get\_confirmation("The key was successfully deleted.\n");

}

void add\_new\_key()

{

// узнать у пользователя а, р, х

printf("Print A, P and X parameteres of new key (use space to separate parameteres).\n");

// отправить параметры МК

sent\_a\_and\_p();

int x;

scanf("%d", &x);

sent\_int(x);

// получить подтверждение

get\_confirmation("New key was added successfully.\n");

}

void encrypt\_message()

{

// узнать у пользователя файл с сообщением

printf("Print the input file name.\n");

char \*name;

scanf("%s", &name);

input\_msg = fopen(name, "r");

// узнать у пользователя имя файла, куда записать зашифрованное сообщение

printf("Print the crypt file name (the encrypted message would be written in there.\n");

scanf("%s", &name);

output\_msg = fopen(name, "w");

// узнать у пользователя параметры а и р, и отправить МК

printf("Print A and P parameteres of encryption key (use space to separate parameteres).\n");

sent\_a\_and\_p();

// посчитать длину сообщения и отправить МК

int length;

fseek(input\_msg, 0, SEEK\_END);

length = ftell(input\_msg) + 1;

fseek(input\_msg, 0, SEEK\_SET);

sent\_int(length);

// получить подтверждение

if (get\_confirmation("The key was founded. MC is waiting for the message.\n") > 1)

return;

// получить a^y и сообщить его пользователю

char a\_pow\_y;

ReadFile(port, &a\_pow\_y, 1, NULL, NULL);

printf("A^Y = %d.\n", a\_pow\_y);

// отправить сообщение

int i;

char buffer;

for (i = 0; i < length; i++)

{

// считать байт шифруемого соощения, отправить его МК

buffer = fgetc(input\_msg);

WriteFile(port, &buffer, 1, NULL, NULL);

Sleep(10);

// считать присланый МК байт, записать в output\_msg

ReadFile(port, &buffer, 1, NULL, NULL);

fputc(buffer, output\_msg);

}

}

void decrypt\_message()

{

// узнать у пользователя файл с зашифрованным сообщением

printf("Print the input file name.\n");

char \*name;

scanf("%s", &name);

input\_msg = fopen(name, "r");

// узнать у пользователя имя файла, куда записать расшифрованное сообщение

printf("Print the decrypt file name (the decrypted message would be written in there.\n");

scanf("%s", &name);

output\_msg = fopen(name, "w");

// узнать у пользователя а, р, a^y

printf("Print A, P and A^Y parameteres of new key (use space to separate parameteres).\n");

// отправить параметры МК

sent\_a\_and\_p();

int a\_pow\_y;

scanf("%d", &a\_pow\_y);

sent\_int(a\_pow\_y);

// посчитать длину сообщения и отправить МК

int length;

fseek(input\_msg, 0, SEEK\_END);

length = ftell(input\_msg) + 1;

fseek(input\_msg, 0, SEEK\_SET);

sent\_int(length);

// получить подтверждение

if (get\_confirmation("The key was founded. MC is waiting for the message.\n") > 1)

return;

// отправить сообщение

int i;

char buffer;

for (i = 0; i < length; i++)

{

// считать байт зашифрованного соощения, отправить его МК

buffer = fgetc(input\_msg);

WriteFile(port, &buffer, 1, NULL, NULL);

Sleep(10);

// считать присланый МК байт, записать в output\_msg

ReadFile(port, &buffer, 1, NULL, NULL);

fputc(buffer, output\_msg);

}

}

void generate\_signature()

{

// узнать у пользователя, по какому ключу он хочет вычислить подпись

printf("Print A and P parameteres of the key you want to generate a signature (use space to separate parameteres).\n");

sent\_a\_and\_p();

// узнать у пользователя, по какому сообщению он хочет вычислить подпись

printf("Print the input file name.\n");

char \*name;

scanf("%s", &name);

input\_msg = fopen(name, "r");

// отправить а и р, вычислить хэш сообщения и отправить его

char hash = calculate\_hash();

WriteFile(port, &hash, 1, NULL, NULL);

Sleep(10);

// получить подтверждение

if (get\_confirmation("The signature was generated successfully.\n") > 1)

return;

// получить подписи

int s1, s2;

char buffer;

ReadFile(port, &buffer, 1, NULL, NULL);

s1 = buffer;

s1 <<= 8;

ReadFile(port, &buffer, 1, NULL, NULL);

s1 += buffer;

ReadFile(port, &buffer, 1, NULL, NULL);

s2 = buffer;

s2 <<= 8;

ReadFile(port, &buffer, 1, NULL, NULL);

s2 += buffer;

printf("The signatures: s1 = %d; s2 = %d.\n", s1, s2);

}

void check\_signature()

{

// узнать у пользователя, по какому ключу он хочет вычислить подпись

printf("Print A and P parameteres of the key you want to check the signature (use space to separate parameteres).\n");

sent\_a\_and\_p();

// узнать у пользователя, по какому сообщению он хочет вычислить подпись

printf("Print the input file name.\n");

char \*name;

scanf("%s", &name);

input\_msg = fopen(name, "r");

// отправить а и р, вычислить хэш сообщения и отправить его

char hash = calculate\_hash();

WriteFile(port, &hash, 1, NULL, NULL);

Sleep(10);

// узнать у пользователя подписи для проверки

printf("Print signatures S1 and S2 to check (use space to separate parameteres).\n");

sent\_a\_and\_p();

// получить подтверждение

char buffer;

ReadFile(port, &buffer, 1, NULL, NULL);

if (buffer == 1)

printf("The signatures are correct.\n");

else if (buffer == 0x0F)

printf("The signatures are not correct.\n");

else

printf("Error.\n");

}

void sent\_a\_and\_p()

{

// отправить а

int a;

scanf("%d", &a);

sent\_int(a);

// отправить р

scanf("%d", &a);

sent\_int(a);

}

void sent\_int(int a)

{

char buffer = a >> 8;

WriteFile(port, &buffer, 1, NULL, NULL);

Sleep(10);

buffer = a;

WriteFile(port, &buffer, 1, NULL, NULL);

Sleep(10);

}

char get\_confirmation(char \*success\_msg)

{

// получить подтверждение

char buffer;

ReadFile(port, &buffer, 1, NULL, NULL);

// вывести пользователю результат

if (buffer == 1)

printf(success\_msg);

else

printf("Error.\n");

return buffer;

}

void last\_msg(char \*str)

{

printf(str);

\_getch();

}

char command1()

{

printf("Choose the command:\n\

1 - delete the key;\n\

2 - add new key;\n\

3 - encrypt the message;\n\

4 - decrypt the message;\n\

5 - generate the signature;\n\

6 - check the signature.\n");

char c, tmp;

scanf("%c", &c);

// считываем \n

scanf("%c", &tmp);

c -= '0';

WriteFile(port, &c, 1, NULL, NULL);

Sleep(10);

return c;

}

char calculate\_hash()

{

// посчитать длину сообщения

int length;

fseek(input\_msg, 0, SEEK\_END);

length = ftell(input\_msg) + 1;

fseek(input\_msg, 0, SEEK\_SET);

// высчитать хэш и вернуть его

char buffer, hash = 0;

for (int i = 0; i < length; i++)

{

hash <<= 1;

buffer = fgetc(input\_msg);

hash += buffer;

}

return hash;

}