*Finished by student PC 12-2 Beklenischev Vladislav*

# Program Description

## General

Program "Digital signing" is the result of the laboratory work for the course "Methods of protection of information" and created only for educational purposes to demonstrate the encrypting / decrypting the various text messages.

To run the program you need the following software:

* Windows 7, 8, 8.1,
* Library .NET Framework version 4.5.

This program was written by means of C # on the .NET Framework in the environment Microsoft Visual Studio 2012.

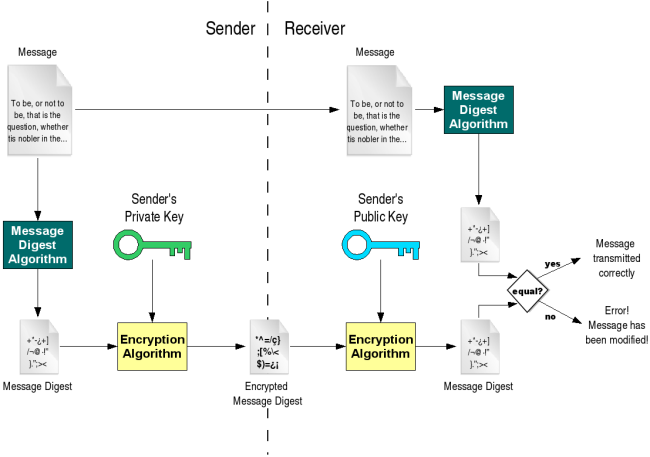
## Functional purpose

Application is designed for the signature text messages using a hash algorithm SHA1 and asymmetric encryption algorithm RSA.

## Description logical structure

As mentioned above, the program "Digital signing" is based on the hash algorithm SHA1 and algorithm asymmetric encryption encryption RSA.

Common encryption scheme can be submitted in the following form:



### Hash algorithm SHA1

It implements the SHA-1 hashfunction, built on the idea of the compression function. Inputs compression function is a message block length of 512 bits and the output of the previous block posts. The output is the value of the hash blocks up to that point.

Pseudocode for the SHA-1 algorithm follows:

*Note 1: All variables are unsigned 32-bit quantities and wrap modulo 232 when calculating, except for*  
 *ml, the message length, which is a 64-bit quantity, and*  
 *hh, the message digest, which is a 160-bit quantity.*  
*Note 2: All constants in this pseudo code are in big endian.*  
 *Within each word, the most significant byte is stored in the leftmost byte position*  
  
*Initialize variables:*  
  
h0 = 0x67452301  
h1 = 0xEFCDAB89  
h2 = 0x98BADCFE  
h3 = 0x10325476  
h4 = 0xC3D2E1F0  
  
ml = message length in bits (always a multiple of the number of bits in a character).  
  
*Pre-processing:*  
append the bit '1' to the message e.g. by adding 0x80 if message length is a multiple of 8 bits.  
append 0 ≤ k < 512 bits '0', such that the resulting message length in *bits*  
 is [congruent](https://en.wikipedia.org/wiki/Modular_arithmetic) to −64 ≡ 448 (mod 512)  
append ml, in a 64-bit [big-endian](https://en.wikipedia.org/wiki/Endianness) integer. Thus, the total length is a multiple of 512 bits.  
  
*Process the message in successive 512-bit chunks:*  
break message into 512-bit chunks  
**for** each chunk  
 break chunk into sixteen 32-bit big-endian words w[i], 0 ≤ i ≤ 15  
  
 *Extend the sixteen 32-bit words into eighty 32-bit words:*  
 **for** i **from** 16 to 79  
 w[i] = (w[i-3] **xor** w[i-8] **xor** w[i-14] **xor** w[i-16]) [**leftrotate**](https://en.wikipedia.org/wiki/Circular_shift) 1  
  
 *Initialize hash value for this chunk:*  
 a = h0  
 b = h1  
 c = h2  
 d = h3  
 e = h4  
  
 *Main loop:*[[47]](https://en.wikipedia.org/wiki/SHA-1#cite_note-47)  
 **for** i **from** 0 **to** 79  
 **if** 0 ≤ i ≤ 19 **then**  
 f = (b **and** c) **or** ((**not** b) **and** d)  
 k = 0x5A827999  
 **else if** 20 ≤ i ≤ 39  
 f = b **xor** c **xor** d  
 k = 0x6ED9EBA1  
 **else if** 40 ≤ i ≤ 59  
 f = (b **and** c) **or** (b **and** d) **or** (c **and** d)   
 k = 0x8F1BBCDC  
 **else if** 60 ≤ i ≤ 79  
 f = b **xor** c **xor** d  
 k = 0xCA62C1D6  
  
 temp = (a **leftrotate** 5) + f + e + k + w[i]  
 e = d  
 d = c  
 c = b **leftrotate** 30  
 b = a  
 a = temp  
  
 *Add this chunk's hash to result so far:*  
 h0 = h0 + a  
 h1 = h1 + b   
 h2 = h2 + c  
 h3 = h3 + d  
 h4 = h4 + e  
  
*Produce the final hash value (big-endian) as a 160 bit number:*  
hh = (h0 **leftshift** 128) **or** (h1 **leftshift** 96) **or** (h2 **leftshift** 64) **or** (h3 **leftshift** 32) **or** h4

The number hh is the message digest, which can be written in hexadecimal (base 16), but is often written using [Base64](https://en.wikipedia.org/wiki/Base64) binary to ASCII text encoding.

The constant values used are chosen to be [nothing up my sleeve numbers](https://en.wikipedia.org/wiki/Nothing_up_my_sleeve_number): the four round constants k are 230 times the square roots of 2, 3, 5 and 10. The first four starting values for h0 through h3 are the same with the MD5 algorithm, and the fifth (for h4) is similar.

Instead of the formulation from the original FIPS PUB 180-1 shown, the following equivalent expressions may be used to compute f in the main loop above:

*Bitwise choice between* c *and* d*, controlled by* b*.*  
(0 ≤ i ≤ 19): f = d **xor** (b **and** (c **xor** d)) *(alternative 1)*  
(0 ≤ i ≤ 19): f = (b **and** c) **xor** ((**not** b) **and** d) *(alternative 2)*  
(0 ≤ i ≤ 19): f = (b **and** c) + ((**not** b) **and** d) *(alternative 3)*  
(0 ≤ i ≤ 19): f = vec\_sel(d, c, b) *(alternative 4)*  
   
*Bitwise majority function.*  
(40 ≤ i ≤ 59): f = (b **and** c) **or** (d **and** (b **or** c)) *(alternative 1)*  
(40 ≤ i ≤ 59): f = (b **and** c) **or** (d **and** (b **xor** c)) *(alternative 2)*  
(40 ≤ i ≤ 59): f = (b **and** c) + (d **and** (b **xor** c)) *(alternative 3)*  
(40 ≤ i ≤ 59): f = (b **and** c) **xor** (b **and** d) **xor** (c **and** d) *(alternative 4)*  
(40 ≤ i ≤ 59): f = vec\_sel(c, b, c **xor** d) *(alternative 5)*

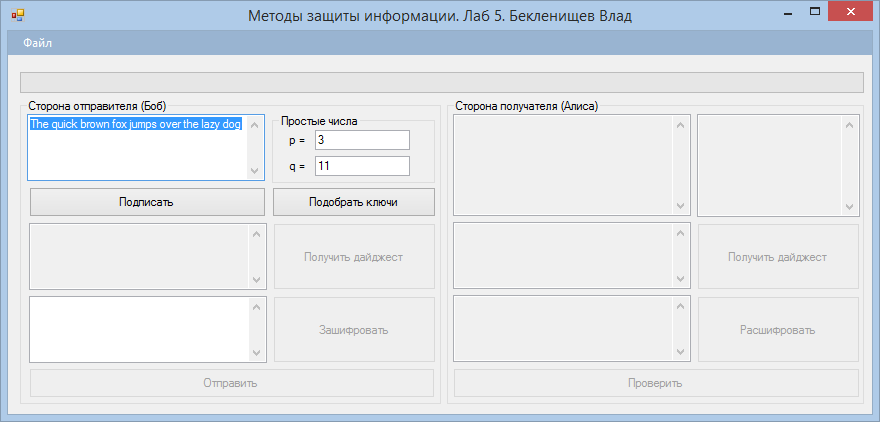
### Steps for encryption

1. Choose public and private keys using an asymmetric encryption algorithm. The program uses RSA algorithm.
2. Sign the original message by adding in the end of the message signature: the **name** of the creator of the signature, **time**, **date** etc.
3. Hash signed original message using the SHA1 algorithm.
4. Encrypt digest from the step 3 using the private key.
5. Sends to receiver message, encrypted digest and public key.
6. The receivert generates a digest of the original message using SHA1.
7. Recipient decrypts encrypted digest (obtained in step 4) by the public key.
8. These digests of steps 6 and 7 are compared. If they are identical, the message is correctly transmitted. Otherwise - the message has changed in the course of transmission.

## Call and load

The program is an application for Windows operating systems 7, 8, 8.1, 10. It can be run from the "Start" menu or by using the shortcut on the desktop.

After starting the main application window will appear:



## Input

Input data are:

1. A text message that can be signed by using the "Subscribe" button.
2. Prime numbers p and q. After their input is necessary to press the button "to find the keys" to be able to adapt RSA public and private keys.
3. After receipt of the encrypted digest (after pressing the "Encrypt") have the opportunity to change the encrypted message. This is done just to be able to show a validation error of the two digests in the recipient after clicking on the button "Check".

The program also allows you to read from the file a text message for transmission to the recipient.

## The output

The result of the program will be:

1. Digests on the side of the recipient and the sender.
2. Encrypted digests on the side of the recipient and the sender.
3. Digest decrypted at the receiving side. Actually digest obtained by the encryption algorithm SHA1 is checked with digest that obtained by decryption with the public key at the recipient side.