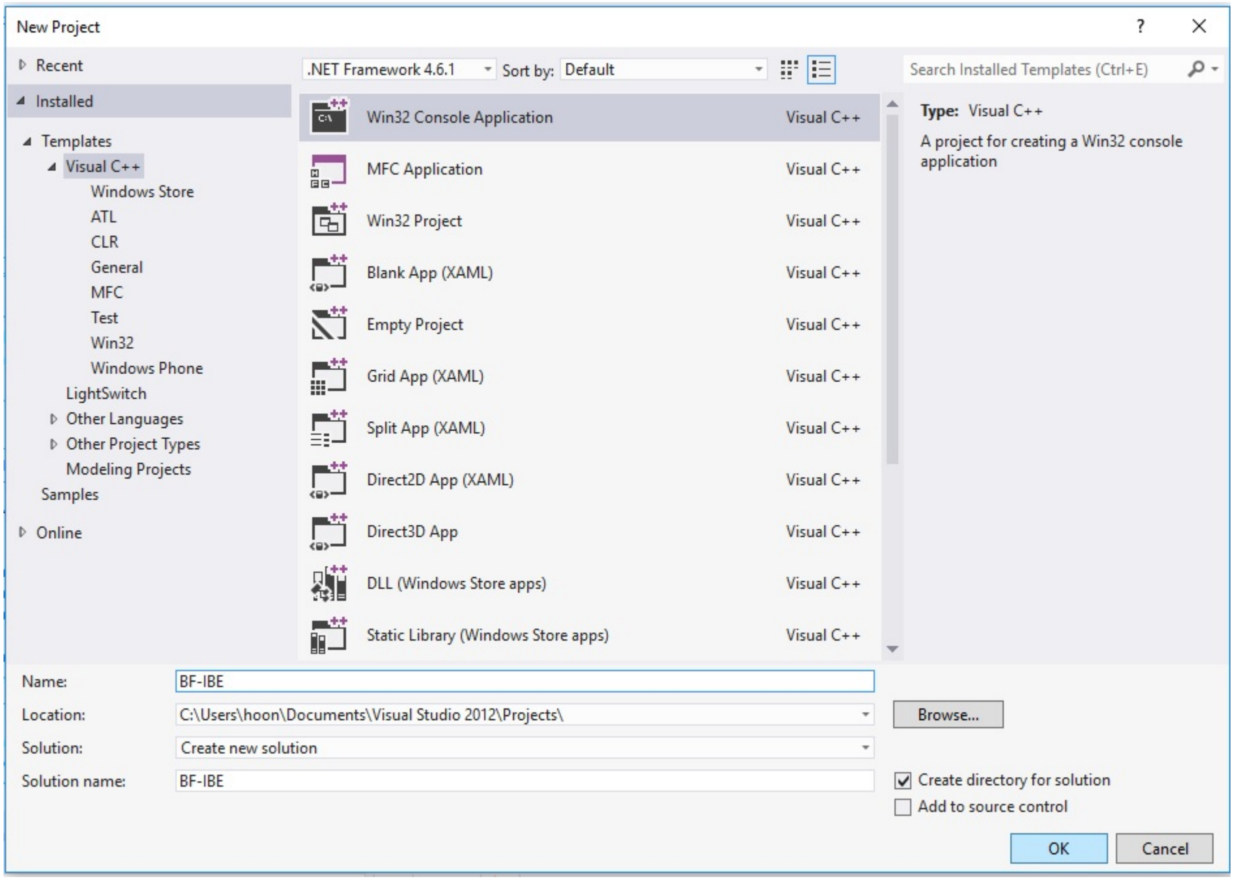
1. BF-IBE (Boneh-Franklin IBE)

The program was compiled in Visual Studio 2012 and the operating system was Windows 10. We will see how to compile the Boneh-Franklin IBE scheme (BF-IBE) implemented using the MIRACL library. The construction of the Boneh-Franklin IBE scheme called Basic Ident is shown. The scheme is known to be secure in IND-ID-CPA.

- Start with creating a new Win 32 Console Application. Write the project name as "BF-IBE" and the solution name as "BF-IBE" and click ok.



Type: Visual C++

A project for creating a Win32 console application

Name: BF-IBE

Location: C:\Users\hoon\Documents\Visual Studio 2012\Projects\

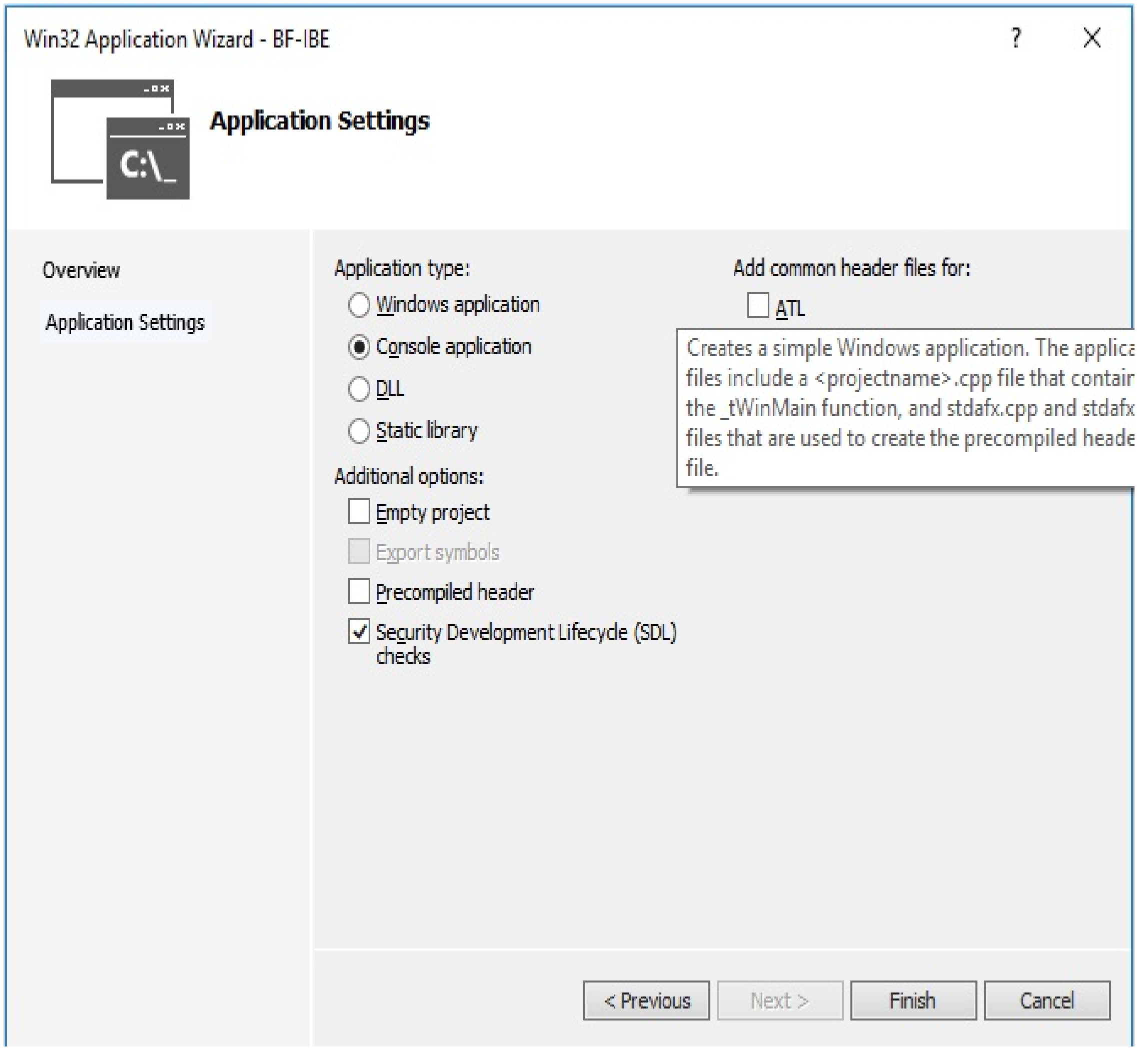
Solution: Create new solution

Solution name: BF-IBE

☑ Create directory for solution

☐ Add to source control

- Now check Console application and Security Development Lifecycle, uncheck the "Precompiler header" in program settings and click finish. You can go to program setting from the left panel as shown in the figure below.



Win32 Application Wizard - BF-IBE

Application Settings

Overview

Application Settings

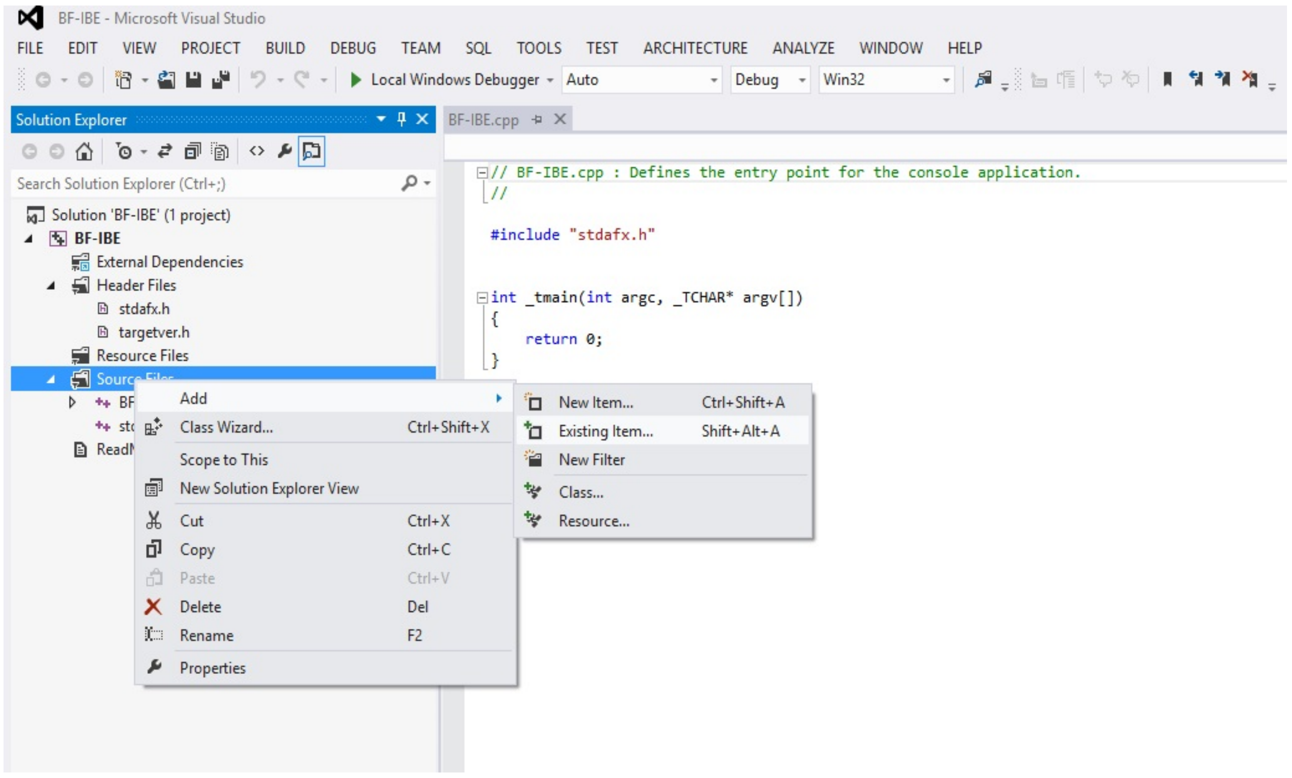
Application type:
- ( ) Windows application
- (●) Console application
- ( ) DLL
- ( ) Static library

Additional options:
- [ ] Empty project
- [ ] Export symbols
- [ ] Precompiled header
- [✓] Security Development Lifecycle (SDL) checks

Add common header files for:
- [ ] ATL

Creates a simple Windows application. The applica
files include a <projectname>.cpp file that contair
the _tWinMain function, and stdafx.cpp and stdafx
files that are used to create the precompiled heade
file.

- Right click the project "Source File" in the left panel and go to Add→Existing Item.
  Click "Existing Item".

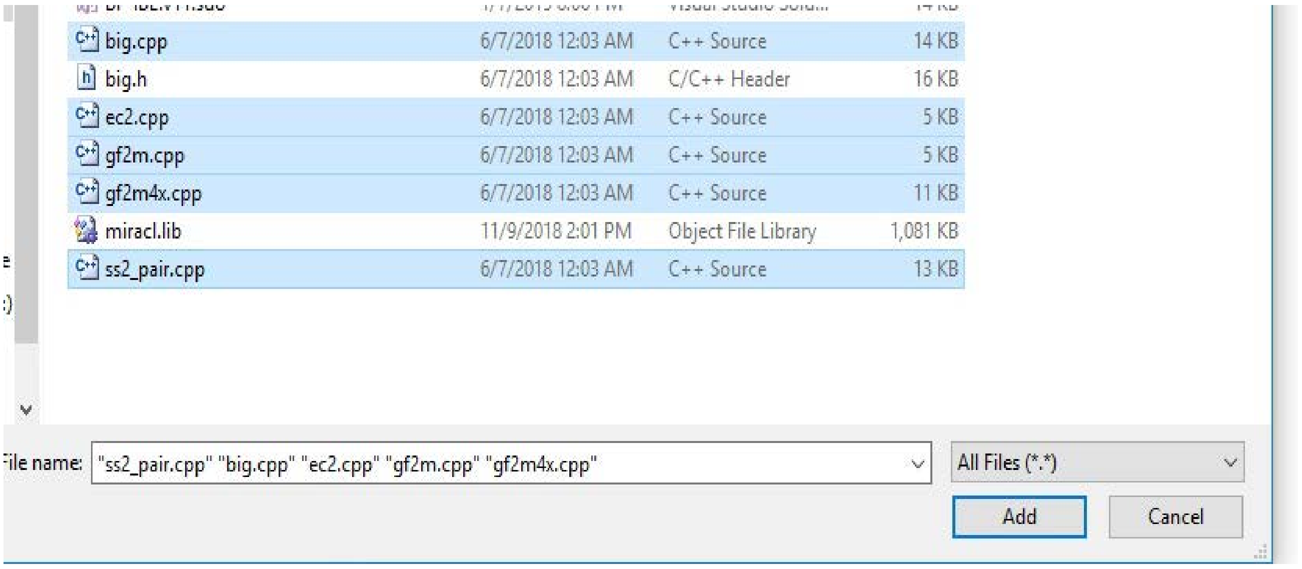


```
// BF-IBE.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{
	return 0;
}
```

- Now add the files mentioned below against each type of pairing.
  - For MR_PAIRING_SSP curves
    ssp_pair.cpp ecn.cpp zzn2.cpp zzn.cpp big.cpp miracl.lib

  - For MR_PAIRING_SS2 curves
    ss2_pair.cpp ec2.cpp gf2m4x.cpp gf2m.cpp big.cpp miracl.lib
    OR
    ss2_pair.cpp ec2.cpp gf2m4x.cpp gf2m.cpp big.cpp miracl.lib

Note: Code for BF-IBE.cpp file is not present in default miracl distribution. Write down all the code provided at the “program code” section below into the BF-IBE.cpp file.

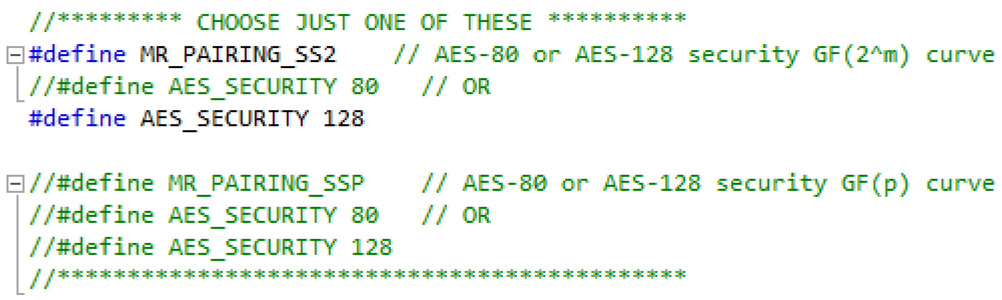


big.cpp	6/7/2018 12:03 AM	C++ Source	14 KB
big.h	6/7/2018 12:03 AM	C/C++ Header	16 KB
ec2.cpp	6/7/2018 12:03 AM	C++ Source	5 KB
gf2m.cpp	6/7/2018 12:03 AM	C++ Source	5 KB
gf2m4x.cpp	6/7/2018 12:03 AM	C++ Source	11 KB
miracl.lib	11/9/2018 2:01 PM	Object File Library	1,081 KB
ss2_pair.cpp	6/7/2018 12:03 AM	C++ Source	13 KB

ile name: "ss2_pair.cpp" "big.cpp" "ec2.cpp" "gf2m.cpp" "gf2m4x.cpp"

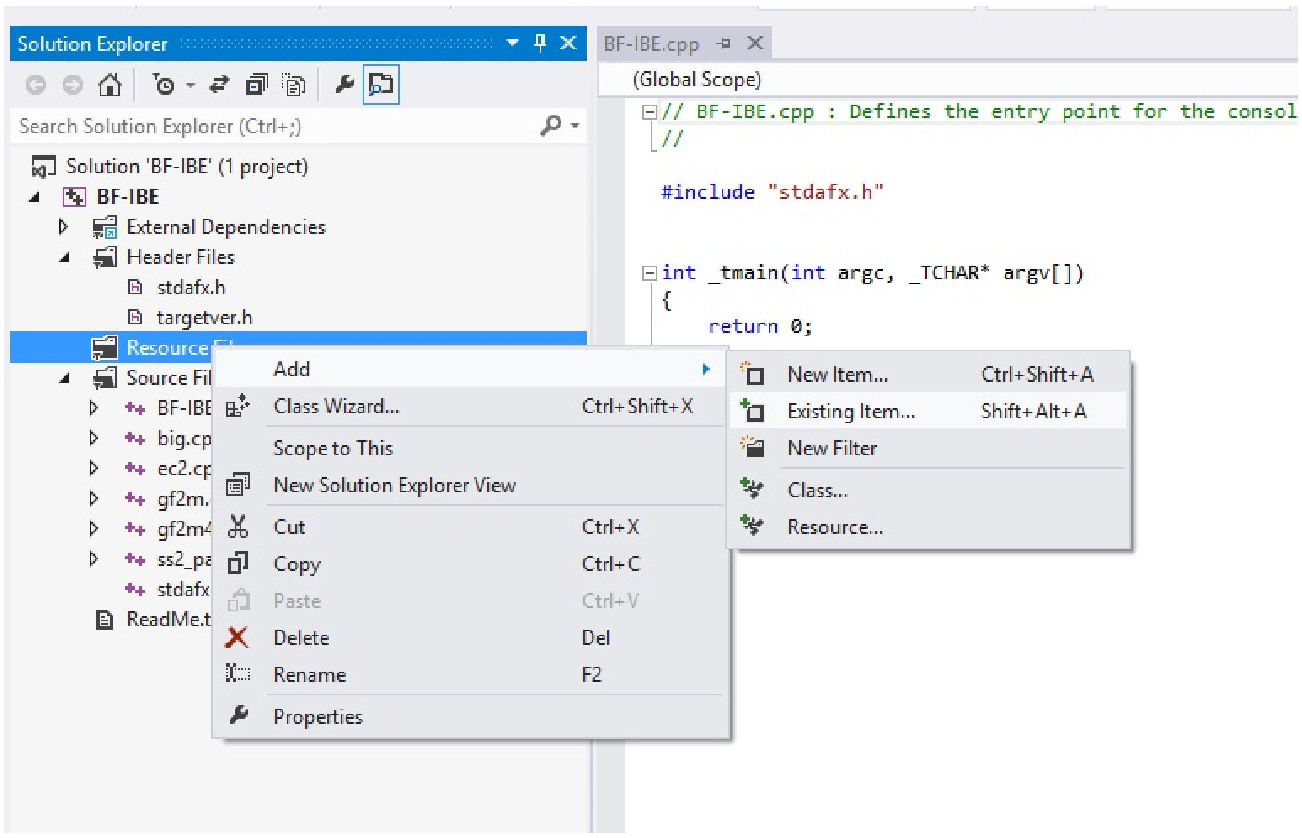
All Files (*.*)

- To choose a pairing, do the followings:
  - Open the BF-IBE.cpp file. Go to the code section as shown in figure below (Code has been copied in the BB-IBE.cpp file in above note).
  - Just uncomment the type of pairing and the security you want to select.
  - The pairing chosen is "MR_PAIRING_SS2" and the security chosen is "AES Security 128". As shown in the figure.
  - We have removed "//" at the starting of "#Define MR_PAIRING_SS2" and "#Define AES_SECURITY 128".

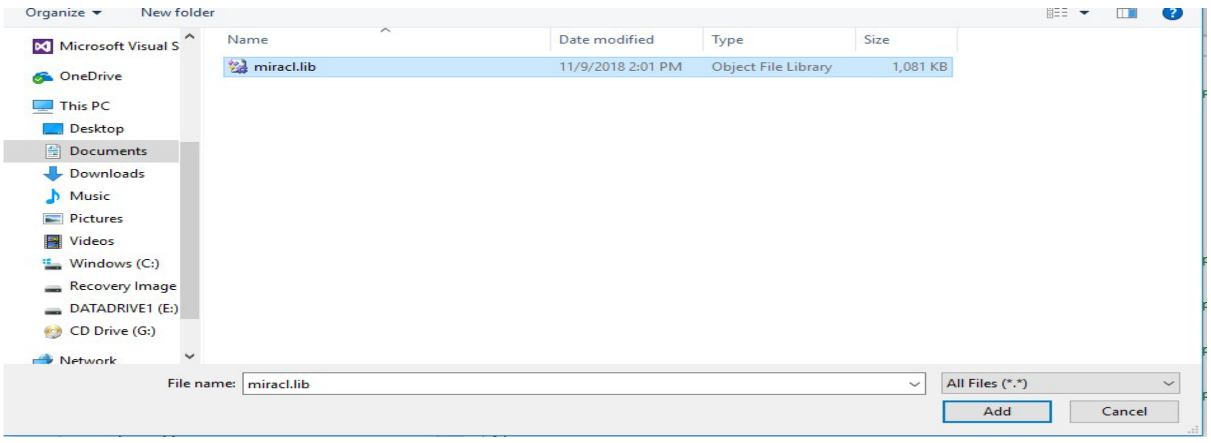


```
//********* CHOOSE JUST ONE OF THESE **********
#define MR_PAIRING_SS2    // AES-80 or AES-128 security GF(2^m) curve
//#define AES_SECURITY 80    // OR
#define AES_SECURITY 128

//#define MR_PAIRING_SSP    // AES-80 or AES-128 security GF(p) curve
//#define AES_SECURITY 80    // OR
//#define AES_SECURITY 128
//*********************************************
```

- Next, add the library file "miracl.lib" in resource folder. Right click on resource folder and go to ADD→Existing Item.



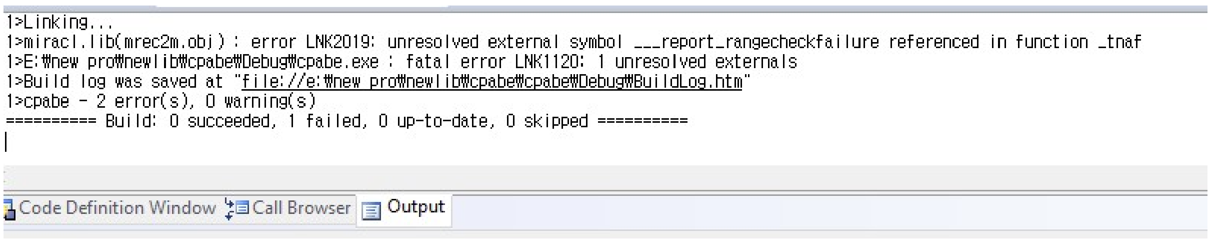
```
// BF-IBE.cpp : Defines the entry point for the consol
//

#include "stdafx.h"

int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
```



Name	Date modified	Type	Size
miracl.lib	11/9/2018 2:01 PM	Object File Library	1,081 KB

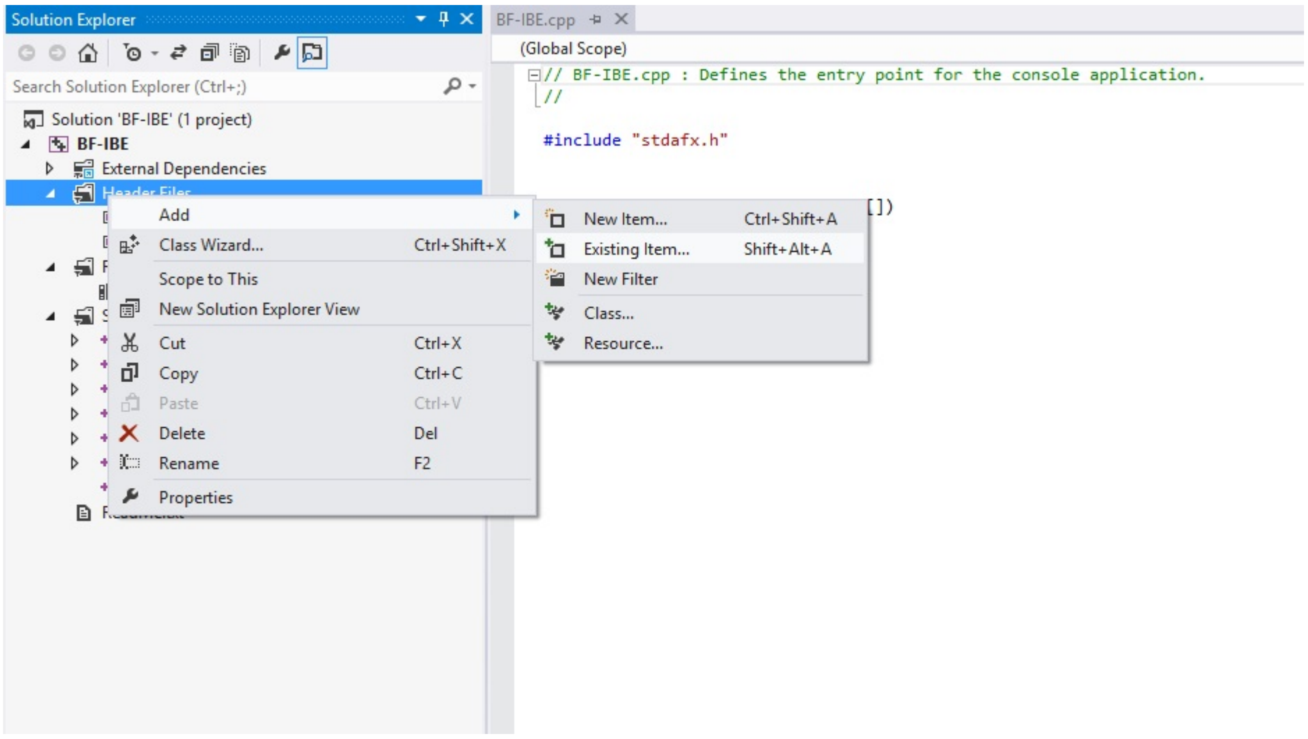
File name: miracl.lib

- Now, we can add the “miracle.lib” to resource files. Otherwise you may see the linking errors like error LNK2019 and error LNK2001 as shown in figure below.



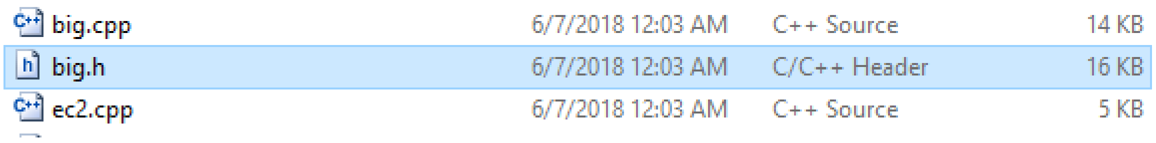
```
1>Linking...
1>miracl.lib(mrec2m.obj) : error LNK2019: unresolved external symbol ___report_rangecheckfailure referenced in function _tnaf
1>E:₩new pro₩newlib₩cpabe₩Debug₩cpabe.exe : fatal error LNK1120: 1 unresolved externals
1>Build log was saved at "file://e:₩new pro₩newlib₩cpabe₩cpabe₩Debug₩BuildLog.htm"
1>cpabe - 2 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

- Right click the project “Header Files” in the left panel and go to Add→Existing Item. Click “Existing Item”.



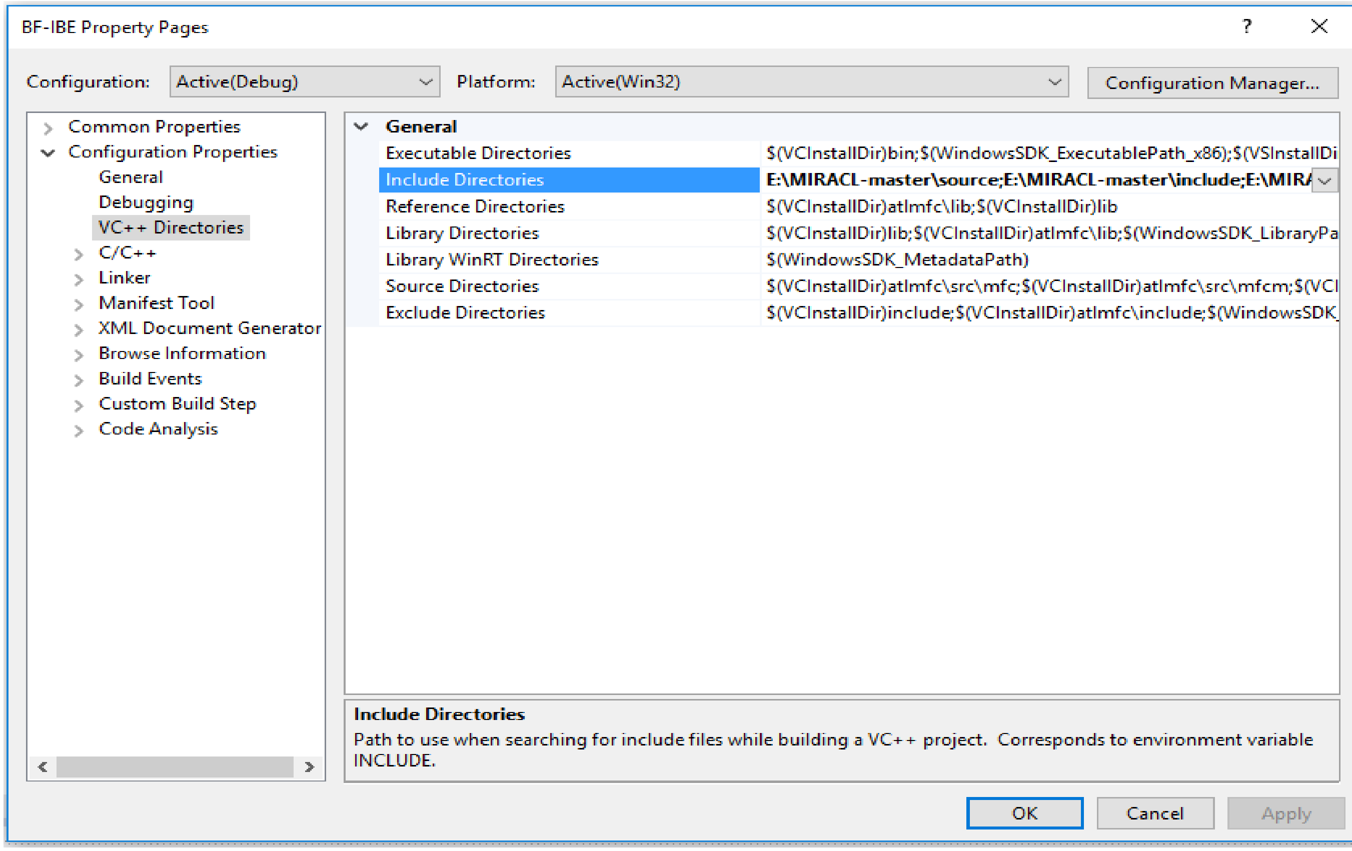
```
// BF-IBE.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
```

- Include the “big.h” file from the miracl distribution.



big.cpp	6/7/2018 12:03 AM	C++ Source	14 KB
big.h	6/7/2018 12:03 AM	C/C++ Header	16 KB
ec2.cpp	6/7/2018 12:03 AM	C++ Source	5 KB

- Go to the properties→Configuration settings→VC++ directories→Include directories. Click on the drop down menu and select the “pairing” folder from the miracl distribution.
- Go to the properties→Configuration settings→VC++ directories→Include directories. Click on the drop down menu and select the “include” folder from the miracl distribution.
- Go to the properties→Configuration settings→VC++ directories→Include directories. Click on the drop down menu and select the “source” folder from the miracl distribution. Then click on APPLY and OK.


BF-IBE Property Pages

Configuration: Active(Debug) Platform: Active(Win32) Configuration Manager...

- Common Properties
- Configuration Properties
  - General
  - Debugging
  - VC++ Directories
  - C/C++
  - Linker
  - Manifest Tool
  - XML Document Generator
  - Browse Information
  - Build Events
  - Custom Build Step
  - Code Analysis

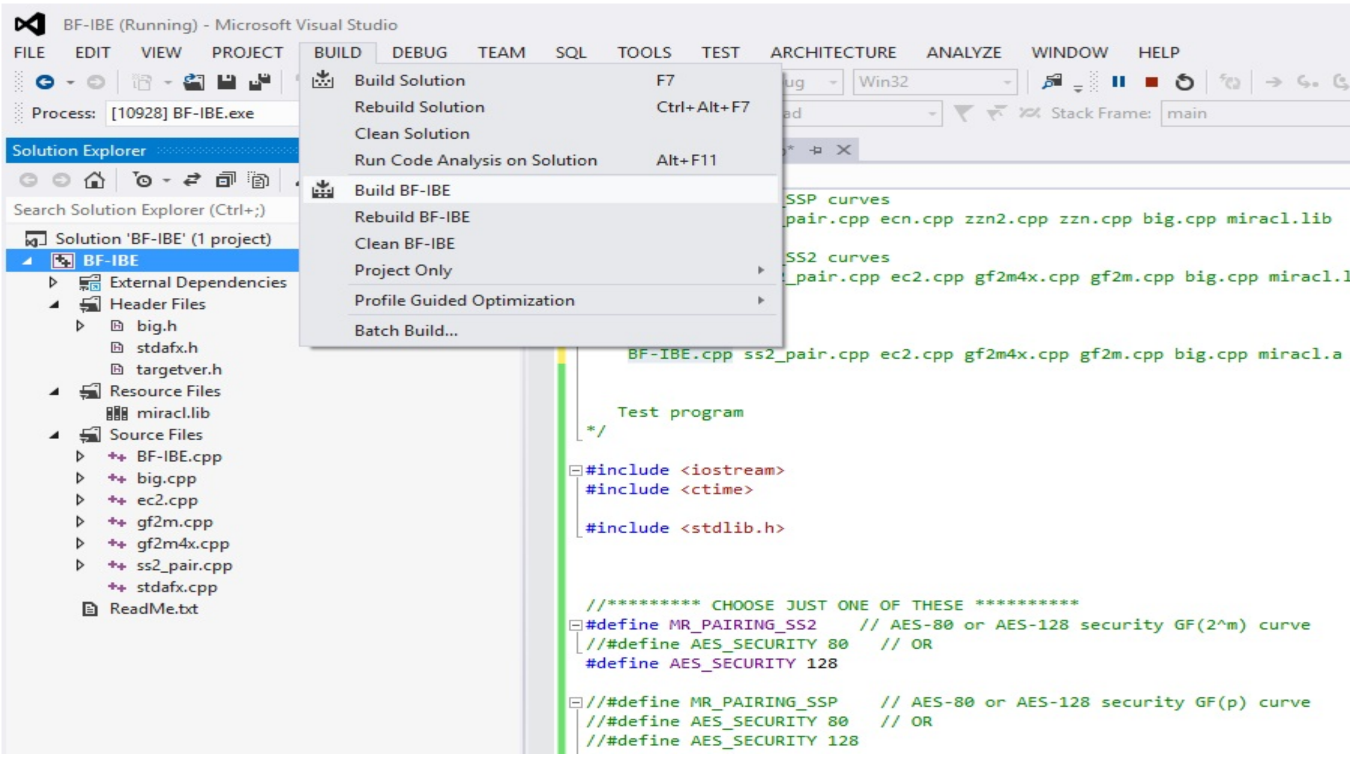
General

Executable Directories	$(VCInstallDir)bin;$(WindowsSDK_ExecutablePath_x86);$(VSInstallDi
Include Directories	**E:\MIRACL-master\source;E:\MIRACL-master\include;E:\MIRA**
Reference Directories	$(VCInstallDir)atlmfc\lib;$(VCInstallDir)lib
Library Directories	$(VCInstallDir)lib;$(VCInstallDir)atlmfc\lib;$(WindowsSDK_LibraryPa
Library WinRT Directories	$(WindowsSDK_MetadataPath)
Source Directories	$(VCInstallDir)atlmfc\src\mfc;$(VCInstallDir)atlmfc\src\mfcm;$(VCI
Exclude Directories	$(VCInstallDir)include;$(VCInstallDir)atlmfc\include;$(WindowsSDK

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

OK Cancel Apply

- Build the program from the Build Tab. Click Build BF-IBE.


BF-IBE (Running) - Microsoft Visual Studio

FILE EDIT VIEW PROJECT BUILD DEBUG TEAM SQL TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP

Process: [10928] BF-IBE.exe

Build Solution	F7
Rebuild Solution	Ctrl+Alt+F7
Clean Solution	
Run Code Analysis on Solution	Alt+F11
Build BF-IBE	
Rebuild BF-IBE	
Clean BF-IBE	
Project Only	
Profile Guided Optimization	
Batch Build...	

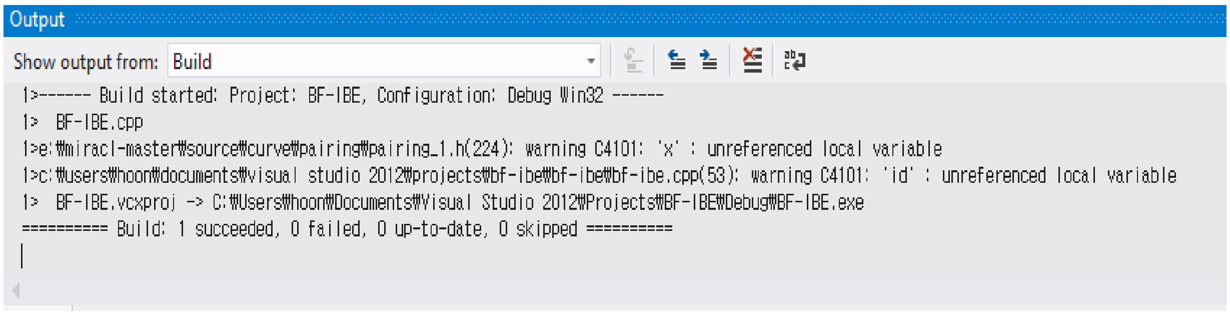
Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'BF-IBE' (1 project)
  - BF-IBE
    - External Dependencies
    - Header Files
      - big.h
      - stdafx.h
      - targetver.h
    - Resource Files
      - miracl.lib
    - Source Files
      - BF-IBE.cpp
      - big.cpp
      - ec2.cpp
      - gf2m.cpp
      - gf2m4x.cpp
      - ss2_pair.cpp
      - stdafx.cpp
    - ReadMe.txt

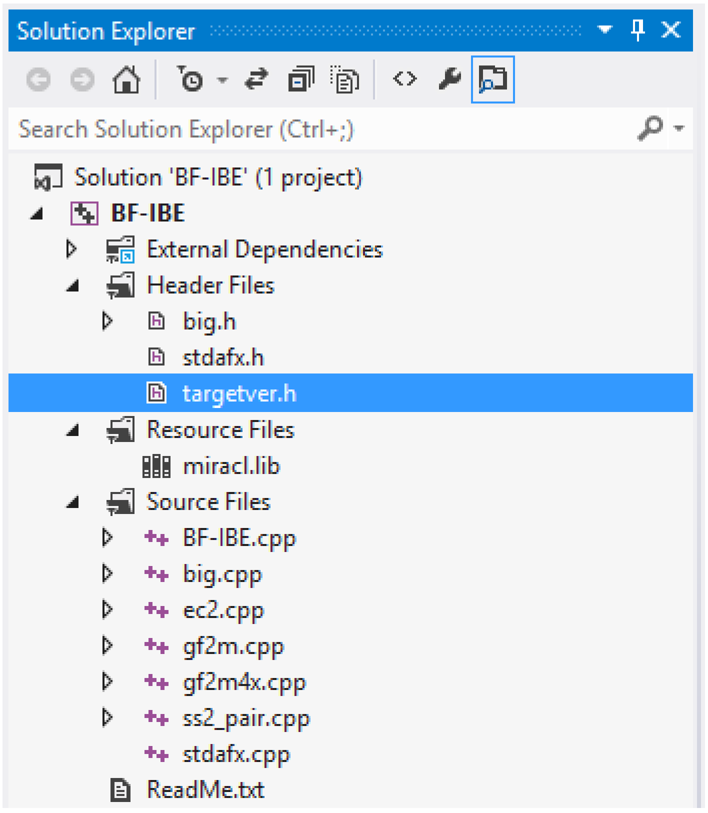
```
SSP curves
pair.cpp ecn.cpp zzn2.cpp zzn.cpp big.cpp miracl.lib

SS2 curves
_pair.cpp ec2.cpp gf2m4x.cpp gf2m.cpp big.cpp miracl.l

   BF-IBE.cpp ss2_pair.cpp ec2.cpp gf2m4x.cpp gf2m.cpp big.cpp miracl.a


   Test program
*/

#include <iostream>
#include <ctime>

#include <stdlib.h>


//********* CHOOSE JUST ONE OF THESE **********
#define MR_PAIRING_SS2    // AES-80 or AES-128 security GF(2^m) curve
//#define AES_SECURITY 80    // OR
#define AES_SECURITY 128

//#define MR_PAIRING_SSP    // AES-80 or AES-128 security GF(p) curve
//#define AES_SECURITY 80    // OR
//#define AES_SECURITY 128
```

- The program should successfully build like below.

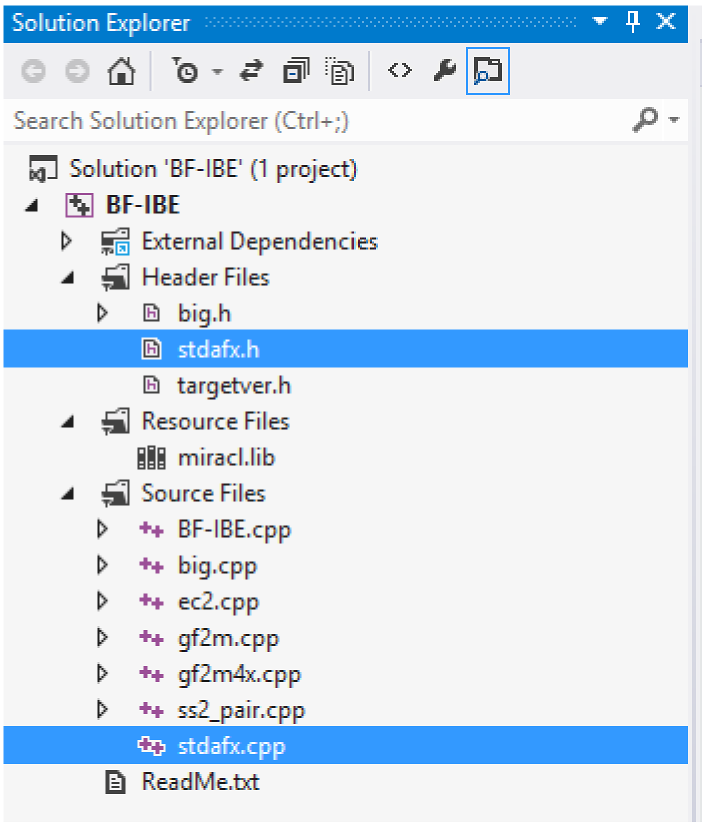


```
1>------ Build started: Project: BF-IBE, Configuration: Debug Win32 ------
1>  BF-IBE.cpp
1>e:₩miracl-master₩source₩curve₩pairing₩pairing_1.h(224): warning C4101: 'x' : unreferenced local variable
1>c:₩users₩hoon₩documents₩visual studio 2012₩projects₩bf-ibe₩bf-ibe₩bf-ibe.cpp(53): warning C4101: 'id' : unreferenced local variable
1>  BF-IBE.vcxproj -> C:₩Users₩hoon₩Documents₩Visual Studio 2012₩Projects₩BF-IBE₩Debug₩BF-IBE.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

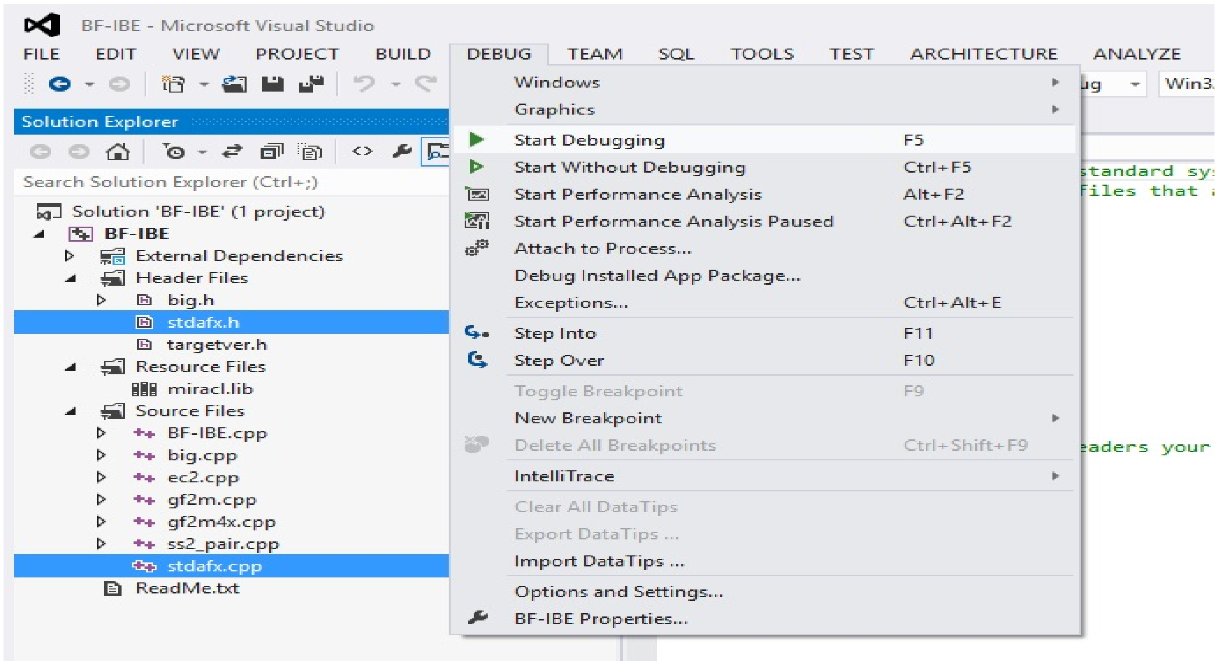
Note: From the solution explorer, make sure that program has "targetver.h" in header files as shown in below figure. This file Include SDKDDKVer.h which defines the highest available Windows platform. If you wish to build your application for a previous Windows platform, include WinSDKVer.h and set the _WIN32_WINNT macro to the platform you wish to support before including SDKDDKVer.h. “targetver.h” and “SDKDDKVer.h” are used to control what functions and constants are included into your code from the Windows headers, based on the OS. The “targetver.h” sets defaults to using the latest version of Windows unless thedefines are specified elsewhere. These two files (targetver.h and SDKDDKVer.h) are auto generated when you create the project, you do not need to manually add them to the project.



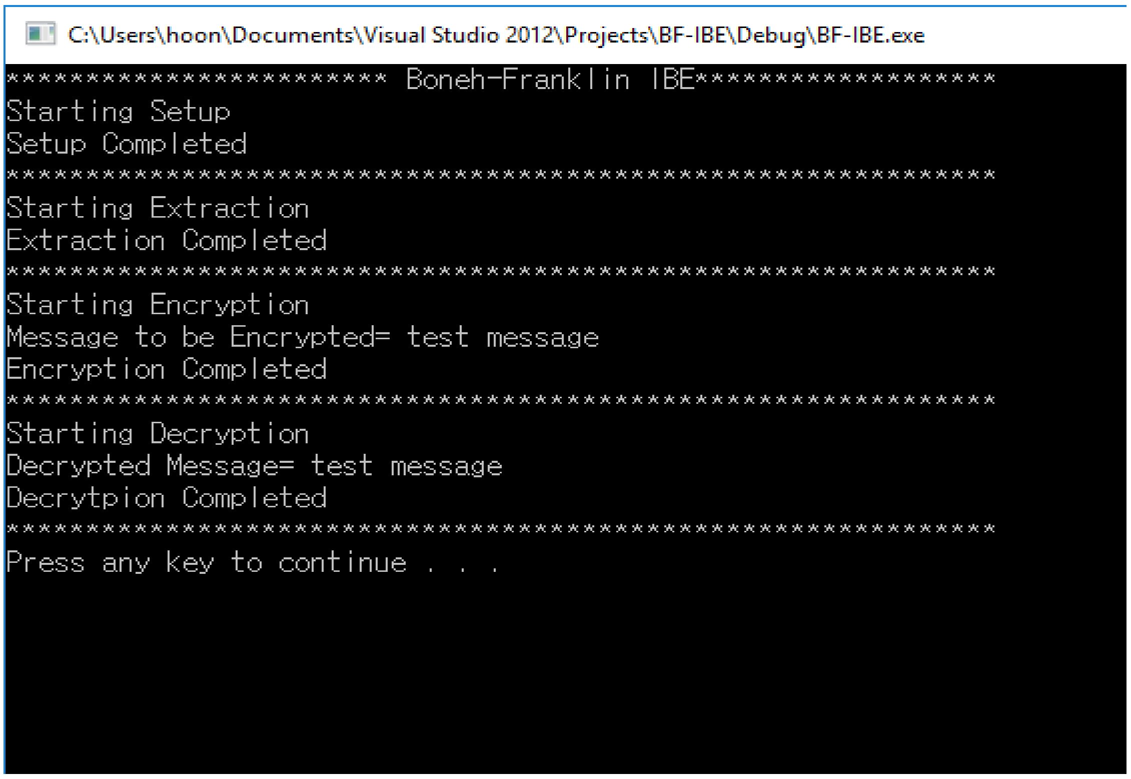
- Also make sure from the solution explorer that the program has "stdafx.h" and "stdafx.cpp". stdafx.h is a "precompiled header file", in which any headers you include are pre-processed to save time during subsequent compilations. These two files (stdafx.cpp and stdafx.h) are auto generated when you create the project. Therefore, you do not need to manually add them to the project or to create them.



- Now click on Debug and click on Start debugging.



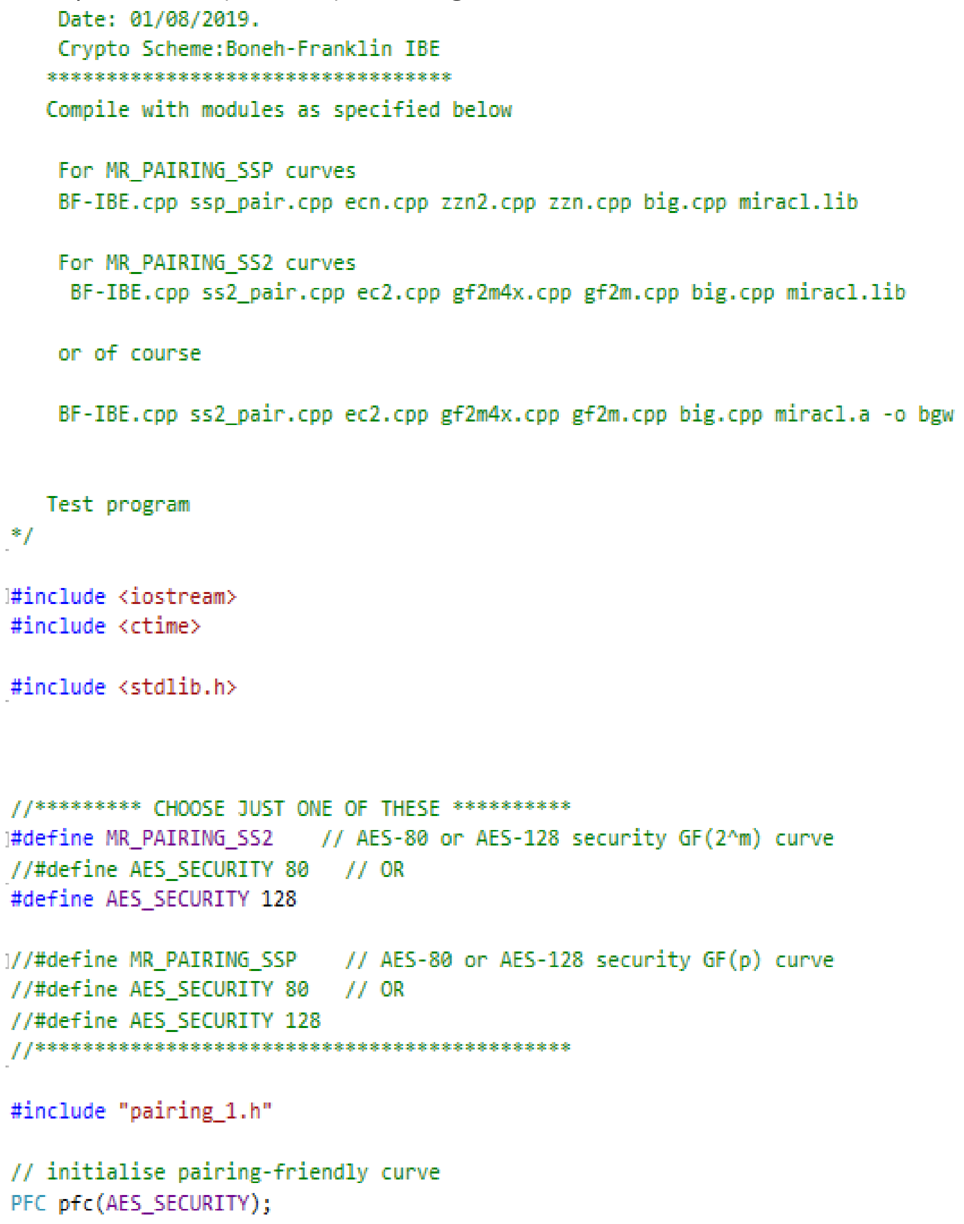
- When you will click on start without debugging after successfully building the program, the program will output the following as shown in figure below.



```
C:\Users\hoon\Documents\Visual Studio 2012\Projects\BF-IBE\Debug\BF-IBE.exe
************************* Boneh-Franklin IBE*******************
Starting Setup
Setup Completed
***************************************************************
Starting Extraction
Extraction Completed
***************************************************************
Starting Encryption
Message to be Encrypted= test message
Encryption Completed
***************************************************************
Starting Decryption
Decrypted Message= test message
Decrytpion Completed
***************************************************************
Press any key to continue . . .
```

1.1 Program Code

- The figure below shows the main function of the “BF-IBE” program, where it initializes parameters for the construction of the Boneh-Franklin IBE scheme. The purpose for each parameter (variable) declaration is mentioned against each parameter (variable) in the figure.



```
     Date: 01/08/2019.
     Crypto Scheme:Boneh-Franklin IBE
    **********************************
    Compile with modules as specified below

     For MR_PAIRING_SSP curves
     BF-IBE.cpp ssp_pair.cpp ecn.cpp zzn2.cpp zzn.cpp big.cpp miracl.lib

     For MR_PAIRING_SS2 curves
      BF-IBE.cpp ss2_pair.cpp ec2.cpp gf2m4x.cpp gf2m.cpp big.cpp miracl.lib

     or of course

     BF-IBE.cpp ss2_pair.cpp ec2.cpp gf2m4x.cpp gf2m.cpp big.cpp miracl.a -o bgw


    Test program
*/

#include <iostream>
#include <ctime>

#include <stdlib.h>



//********* CHOOSE JUST ONE OF THESE **********
#define MR_PAIRING_SS2    // AES-80 or AES-128 security GF(2^m) curve
//#define AES_SECURITY 80   // OR
#define AES_SECURITY 128

//#define MR_PAIRING_SSP    // AES-80 or AES-128 security GF(p) curve
//#define AES_SECURITY 80   // OR
//#define AES_SECURITY 128
//*********************************************

#include "pairing_1.h"

// initialise pairing-friendly curve
PFC pfc(AES_SECURITY);
```

```
/* H2 maps a GT element in to hash of (0,1)*/
Big H2(GT g){

    Big HASH;
    HASH=pfc.hash_to_aes_key(g);
    return HASH;}


int main()
{

     // get handle on mip (Miracl Instance Pointer)
    miracl* mip=get_mip();
    // get pairing-friendly group order
    Big order=pfc.order();
    //Initilization of public parameters
    //parameters of big type
    Big ID,s,M,r,V;
      // Group G1 type elements
    G1 P,Ppub,Qid,did,U;
    /*The time_t datatype is a data type in the ISO C library defined for storing system time values. Such values
    are returned from the standard time() library function
    we use get a seed for our random number.
    */
    time_t seed;
    // initialise (insecure!) random numbers
    time(&seed);
      //  creat random number of long type from the seed value.
    irand((long)seed);
    //Character id
    char id;
     // Groupt GT elements.
    GT gID,gid2,g2;

    cout << "************************* Boneh-Franklin IBE ******************* " << endl;
```

- **Setup:**

```
cout << "Starting Setup" << endl;

//generate random value of s
pfc.random(s);
//generate random value of P
pfc.random(P);
//precompute P
pfc.precomp_for_mult(P);
 // ppub=P*s
Ppub=pfc.mult(P,s);
//precompute ppub
pfc.precomp_for_mult(Ppub);

cout << "Setup Completed" << endl;
```

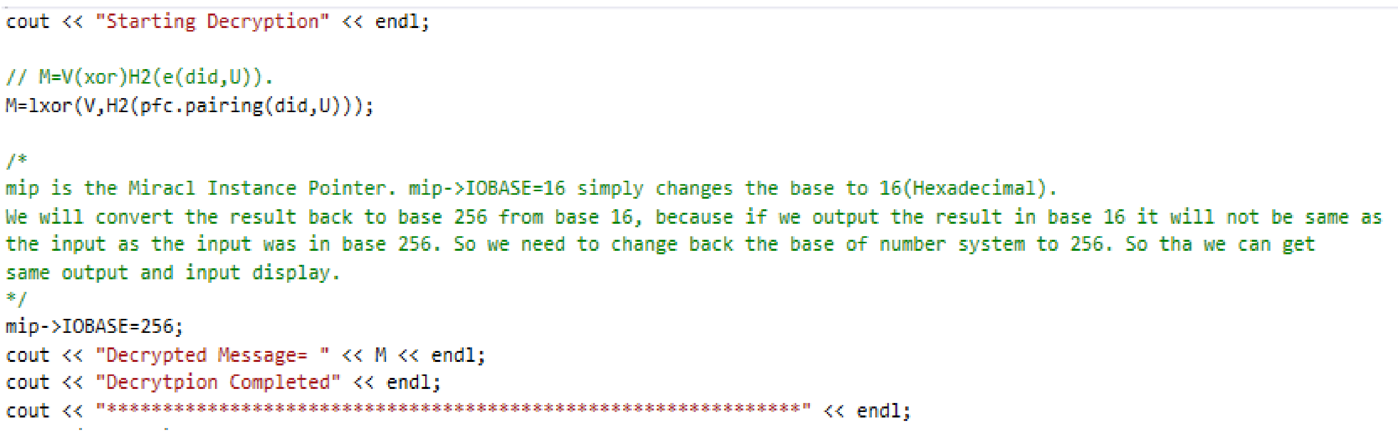
- **Extraction :**

```
cout << "Starting Extraction" << endl;

// Qid=H1(ID). Here ID= Alice and H1 (we use pfc.hash_and_map() as H1) maps a character string to G1 element.
pfc.hash_and_map(Qid,"Alice");

// did=Qid*s . Here ID= Alice
did=pfc.mult(Qid,s);
cout << "Extraction Completed" << endl;
```

- **Encryption:**

```
cout << "Starting Encryption" << endl;

mip is the Miracl Instance Pointer. mip->IOBASE=256 simply changes the base to 256.
We take input in a base 256 to componsate all the real world letters and special characters.
Which are easy to be represented in base 256 system of numbers.

mip->IOBASE=256;
// to be encrypted to Alice, convert it from char to Big data type.
M=(char *)"test message";

//print "Message to be encrypted"
cout << "Message to be Encrypted= " << M << endl;
/*
mip is the Miracl Instance Pointer. mip->IOBASE=16 simply changes the base to 16(Hexadecimal).
We use the hexadecimal numbers to make coding for microprocessor. But it coonverts that to binary
for computation. After the computation the result will be in hexadecimal format by inverse conversion.
*/
mip->IOBASE=16;



 // generate random r
pfc.random(r);

 // U= r*P
U=pfc.mult(P,r);

// Qid=H1(ID) . Here ID= Alice
pfc.hash_and_map(Qid,"Alice");
//gID= e(QID,Ppub).
gID=pfc.pairing(Qid,Ppub);

 // gid2= (gID)^r
gid2=pfc.power(gID,r);

// V= M (XOR) H2((gID)^r)
V=lxor(M,H2(gid2));
cout << "Encryption Completed" << endl;
```

- **Decryption:**



```
cout << "Starting Decryption" << endl;

// M=V(xor)H2(e(did,U)).
M=lxor(V,H2(pfc.pairing(did,U)));

/*
mip is the Miracl Instance Pointer. mip->IOBASE=16 simply changes the base to 16(Hexadecimal).
We will convert the result back to base 256 from base 16, because if we output the result in base 16 it will not be same as
the input as the input was in base 256. So we need to change back the base of number system to 256. So tha we can get
same output and input display.
*/
mip->IOBASE=256;
cout << "Decrypted Message= " << M << endl;
cout << "Decrytpion Completed" << endl;
cout << "**************************************************************" << endl;
```