

# 文档(WPF 版)

陈健文(双学位) 中山大学化学学院 13322016

## 1. 前言

这个项目是我的第二篇毕业论文的后续，将当中的已重新实现的 RSA 算法再编写一个图形界面，做成可执行文件的形式。因完成毕业论文以后复习时间匆忙，故一直延迟到暑假阶段才开始继续编写，为了完整性，我打算将 RSA 的实现部分也一并写进文档里。

(2018.7.25 添加:做到今天我实在是 Debug 不下去了,生成的 dll 文件可能有未知的 bug,所以没办法下我放弃继续 debug,转而使用 Qt 重写一遍界面, WPF 版的实现效果见后文描述。)

(2018.7.26 添加:今天又 debug 了一天(真香!),没有使用 Qt 写,因为重新再来太麻烦了,结果上基本上确定 bug 点在 C++向 C#返回字符串的时候,如果加密位数相对于明文来说过长的时候有可能就会返回空字符串。)

(2019.2.20 添加:为了整理以前的代码决定将这部分传上 github,等考研这堆事情过去以后再慢慢查这个 bug。)

## 2. RSA 实现部分

RSA 的实现部分主要有两步,第 1 步是引入 MPIR 库到项目中,第 2 步是利用 MPIR 库引入的大整数类型完成算法。

### 2.1 引入 MPIR 库

(1) 首先在网站上下载 MPIR 库以及相应的文档(<http://www.mpir.org/downloads.html>)

#### Downloads

For a list of changes in each release, see the [NEWS](#) file.

Note that MPIR is licensed LGPLv3+.

In memory of Jason Moxham, long time MPIR developer who passed away in 2012.

MPIR tarballs:

- 2017-03-01 [MPIR 3.0.0 source bz2](#), [MPIR 3.0.0 source zip](#), documentation: [Documentation.pdf](#) (MPIR 3.0.0)
- 2015-11-20 [MPIR 2.7.2 source bz2](#), [MPIR 2.7.2 source zip](#), documentation: [Documentation.pdf](#) (MPIR 2.7.2)
- 2012-11-08 [MPIR 2.6.0 source bz2](#), documentation: [Documentation.pdf](#)
- 2012-10-02 [MPIR 2.5.2 source bz2](#), documentation: [Documentation.pdf](#)

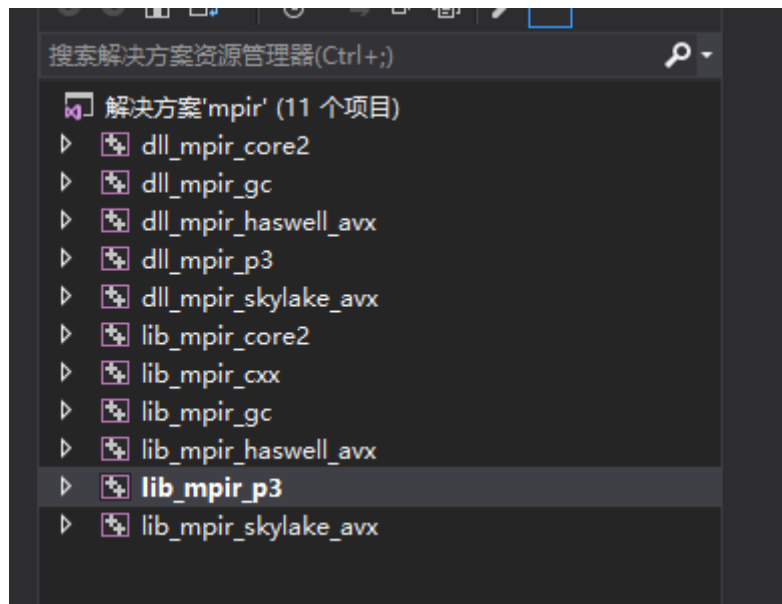
(2) 选择编译 build.vcxx, 由于我使用的是 vs2017, 所以选择 build.vc15

 build.vc11	2017/2/23 0:18	文件夹
 build.vc12	2017/2/23 0:18	文件夹
 build.vc14	2017/2/23 0:18	文件夹
 build.vc15	2017/2/23 0:18	文件夹
 cxx	2017/3/1 19:11	文件夹

(3) 打开 build.vc15, 直接打开解决方案文件 mpir.sln

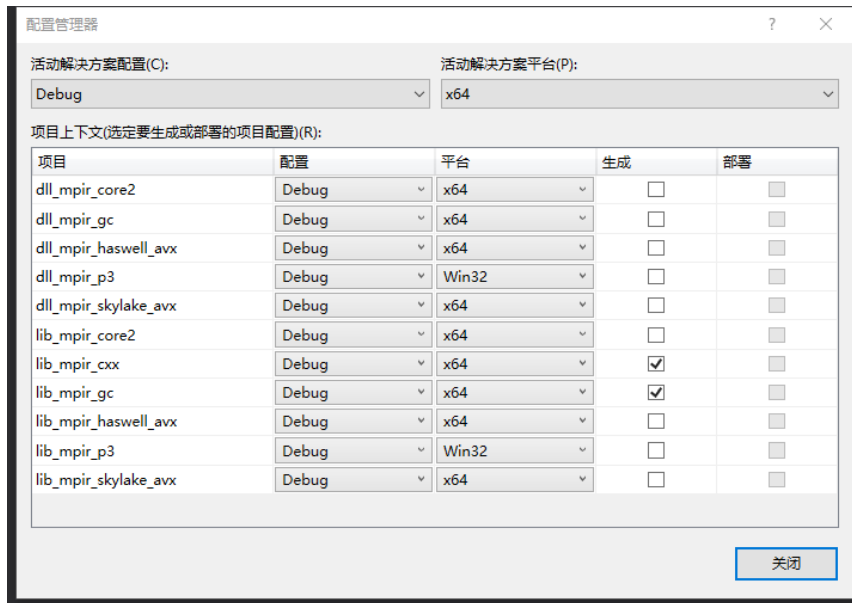
_init_.py	2017/1/31 23:30	JetBrains PyCharm...	0 KB
mpir.sln	2017/2/17 12:18	Visual Studio 解	6 KB
mpir-tests.sln	2017/2/13 18:19	Visual Studio 解	128 KB
mpir-tune.sln	2017/1/31 23:36	Visual Studio 解	5 KB
msbuild.bat	2017/2/23 0:18	Windows 批处理	2 KB
run-speed.py	2017/2/22 0:22	JetBrains PyCharm...	11 KB

(4) 进入到 VS 的界面, 可以看到这个解决方案包含了如下的库

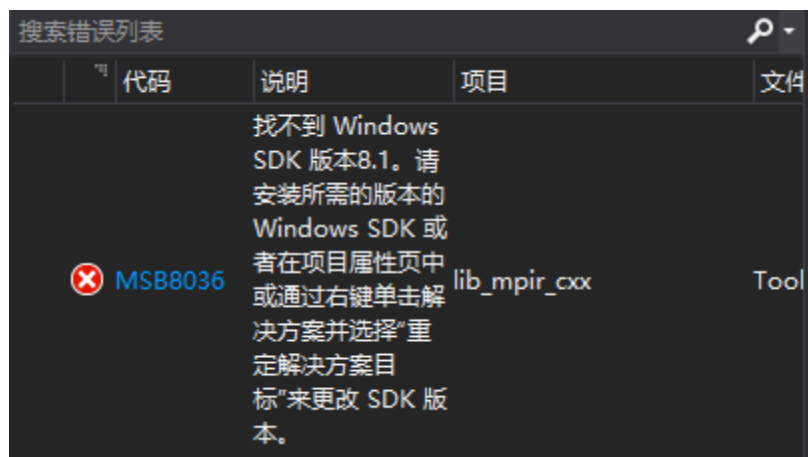


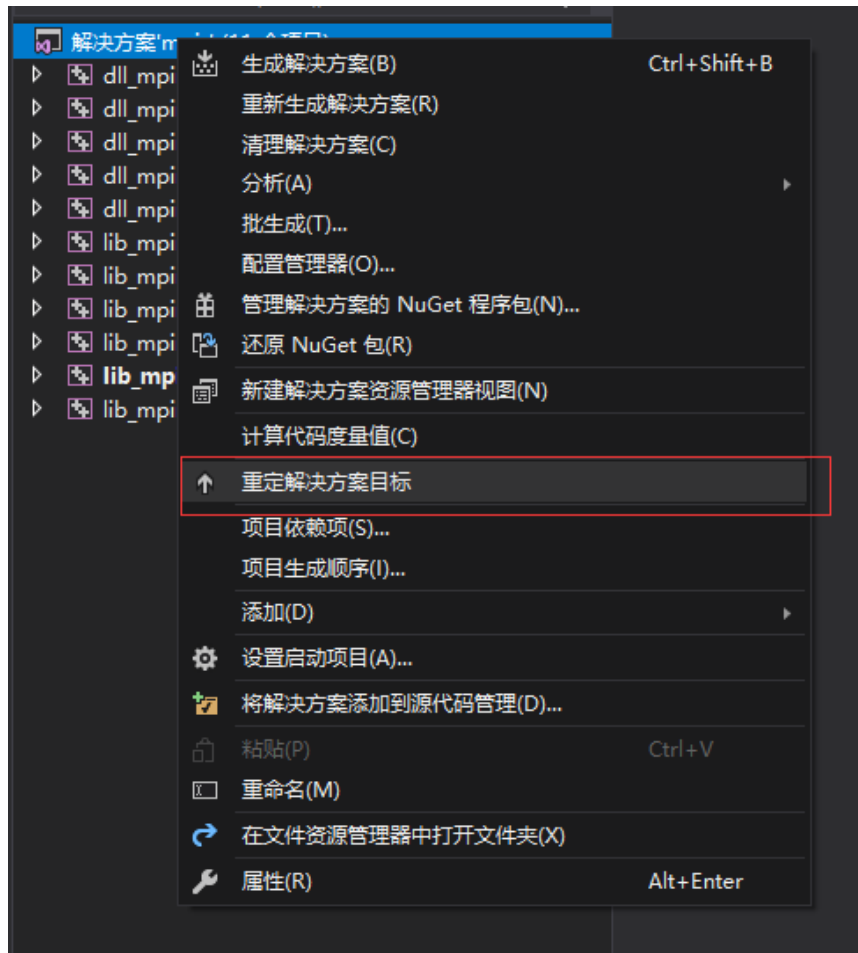
由于 mpir 是大整数运算库, 运算精度是与 CPU 相关的, 所以这里给出了多种 CPU 的解决方案, 我们这里需要的是 **lib\_mpir\_cxx** 以及 **lib\_mpir\_gc**, 为什么是生成 lib 呢, 因为我们需要的是内部库, 而不是进程 dll 文件, 后面整合的时候才需要将整个 RSA 项目编译为 dll 进程文件。

选中解决方案'mpir', 右击选择“属性”, 弹出属性页以后, 在“配置属性”中勾出需要更改的 lib 库, 将下拉框选中“Release”, 活动解决方案平台选择 win64, 这样子可以避免后面的警告, 我们这里需要在“Debug”和“Release”模式下分别进行生成操作。






- (5) 再右键点击解决方案“mpir”，点击生成解决方案，会发生报错。这里是因为从 vs2017 开始使用的 Windows SDK 为 10 开头的编号，因此我们需要修改 SDK 的版本。然后会在弹出框中看到项目升级完成。





```
正在升级项目 "lib_mpir_p3" ...  
正在升级项目 "dll_mpir_p3" ...  
正在升级项目 "dll_mpir_gc" ...  
正在升级项目 "lib_mpir_gc" ...  
正在升级项目 "dll_mpir_core2" ...  
正在升级项目 "lib_mpir_core2" ...  
正在升级项目 "lib_mpir_cxx" ...  
正在升级项目 "dll_mpir_skylake_avx" ...  
正在升级项目 "lib_mpir_skylake_avx" ...  
正在升级项目 "dll_mpir_haswell_avx" ...  
正在升级项目 "lib_mpir_haswell_avx" ...  
重定目标结束: 11 个已完成, 0 个未通过, 0 个已跳过
```

- (6) 接着重新开始编译，可以进行编译，但会看到当时我遇到的比较麻烦的第一个报错，不过后来发现只要直接选中项目重新生成解决方案就没有发生报错了。

代码	说明	项目	文件
 C1083	无法打开包括文件: "mpir.h": No such file or directory	lib_mpir_cxx	isfur
 C1083	无法打开包括文件: "mpir.h": No such file or directory	lib_mpir_cxx	ismpr
 C4819	该文件包含不能在当前代码页(936)中表示的字符。请将该文件保存为 Unicode 格式以防止数据丢失	lib_mpir_gc	gcd





- (7) 编译大概需要等待 1 分钟的时间，成功以后，在 build.vc15/lib\_mpir\_gc/Win32(或者 X64) 以及 build.vc15/lib\_mpir\_cxx/Win32(或者 X64)的两个文件夹里面均会发现 Debug 和 Release 文件夹。

电脑 > Ftp (E:) > C++ Project > mpir-3.0.0 > build.vc15 > lib_mpir_gc > Win32				
名称	修改日期	类型	大小	
 Debug	2018/7/21 11:20	文件夹		
 Release	2018/7/21 11:19	文件夹		

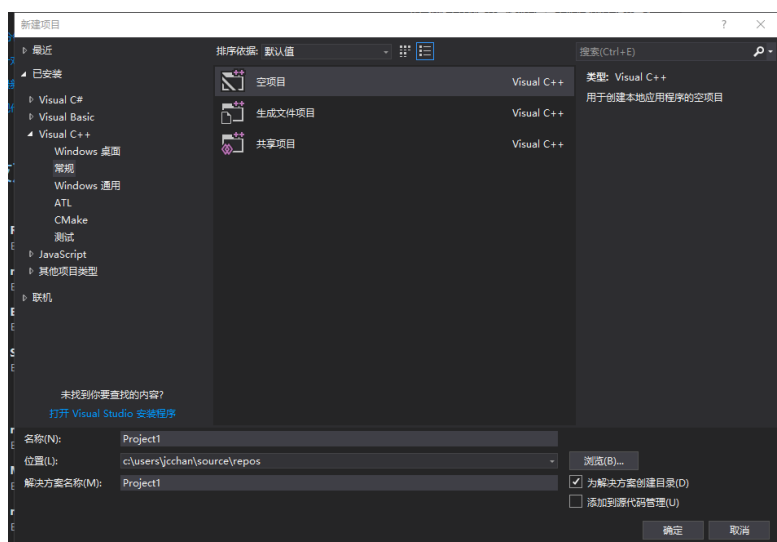
  

电脑 > Ftp (E:) > C++ Project > mpir-3.0.0 > build.vc15 > lib_mpir_cxx > Win32				
名称	修改日期	类型	大小	
 Debug	2018/7/21 11:20	文件夹		
 Release	2018/7/21 11:18	文件夹		

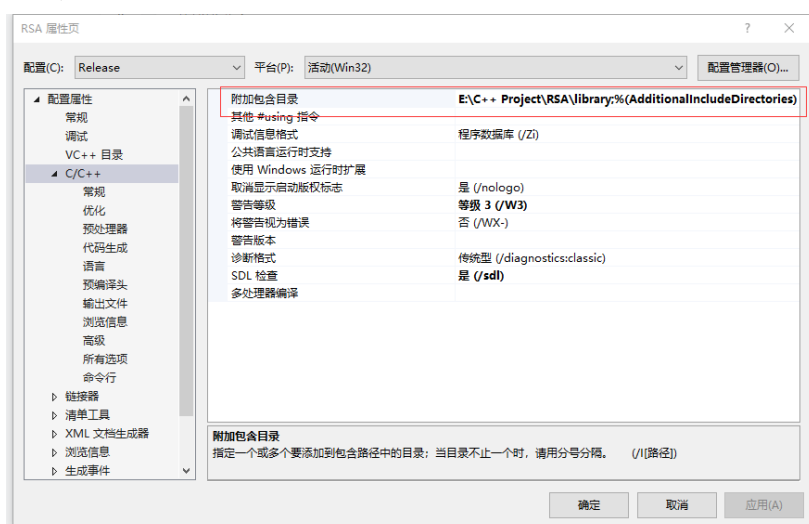
- (8) 接下来在我们新建的项目中新建一个 library 文件夹，将上述两个的 Debug 和 Release 文件直接合并，然后在 mpir-3.0.0 的文件夹下找到 mpir.h 以及 mpirxx.h 一起放入到 library 文件夹中。

> 此电脑 > Ftp (E:) > C++ Project > RSA > library				
名称	修改日期	类型	大小	
 Debug	2018/7/21 11:53	文件夹		
 Release	2018/7/21 11:53	文件夹		
 mpir.h	2018/7/21 11:12	C/C++ 标头	96 KB	
 mpirxx.h	2017/2/23 0:18	C/C++ 标头	121 KB	

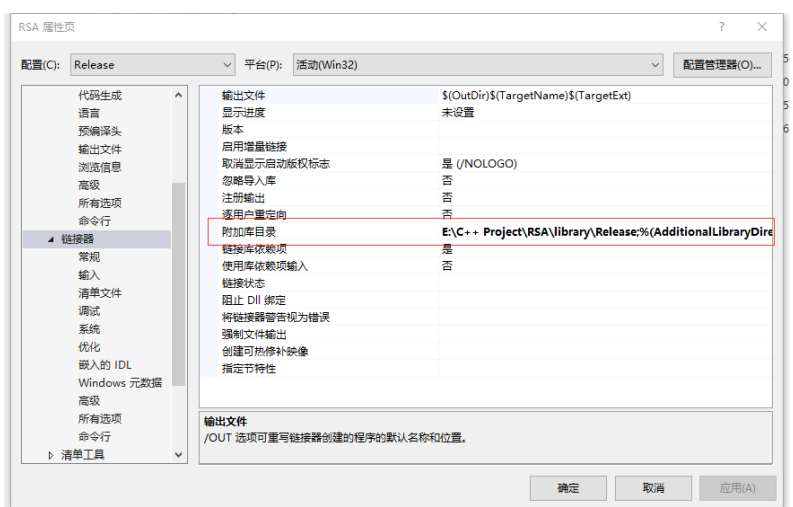
- (9) 新建一个普通的控制台项目，就是跟平时写 C++控制台代码一样的项目即可。



(10)接着在我们的项目中，右键解决方案打开属性，在 C/C++的附加包含目录中选择刚才新建的 library 文件夹的路径。

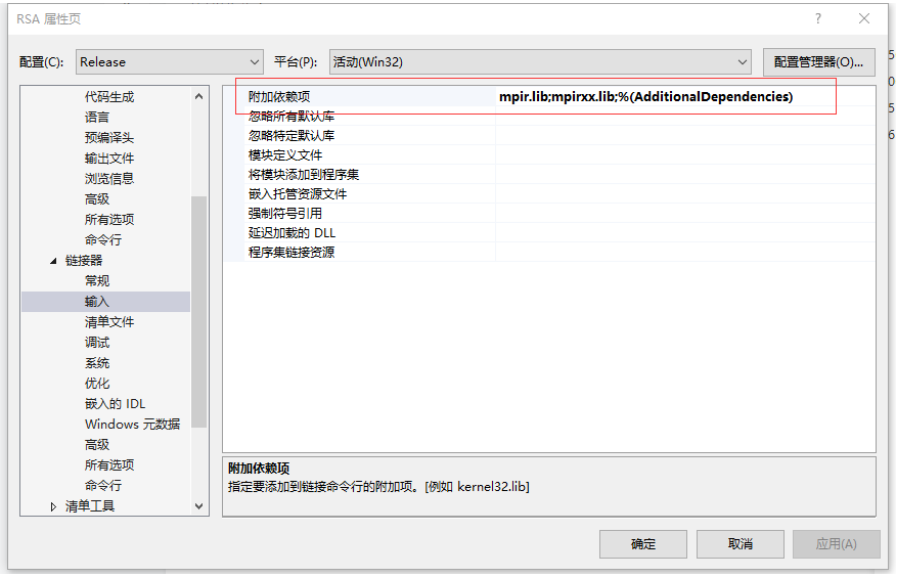


(11)在相同目录下的链接器里面的附加库目录将 library 下的 Release 文件选择好，这样子我们一会儿编译的时候就选择在 Release 模式下。

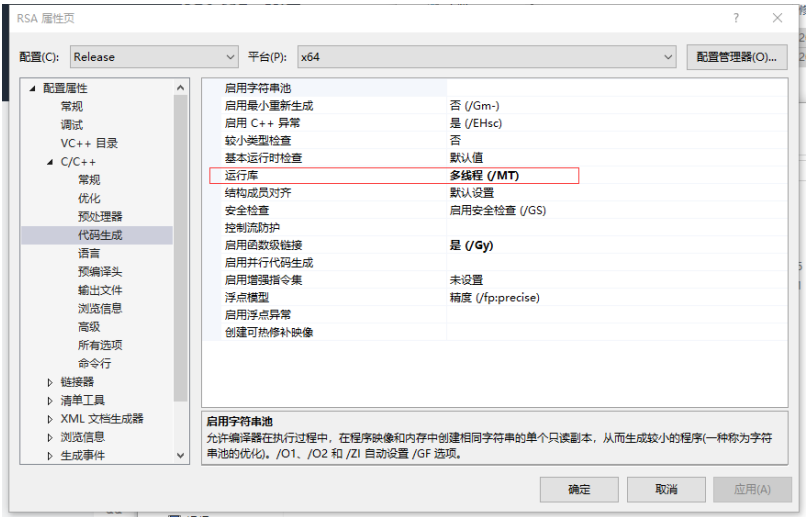


(12)接着在链接器→输入→附加依赖项下添加 mpir.lib 和 mpirxx.lib，点击确定，这样子这个

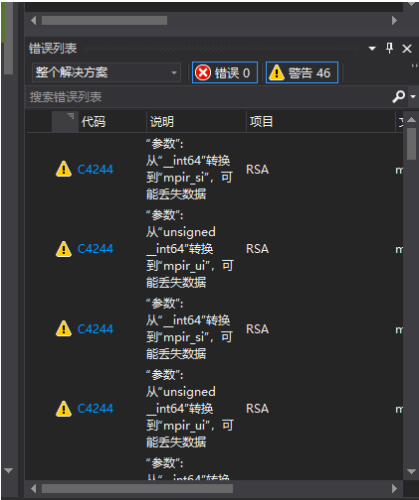
项目就配置好了。



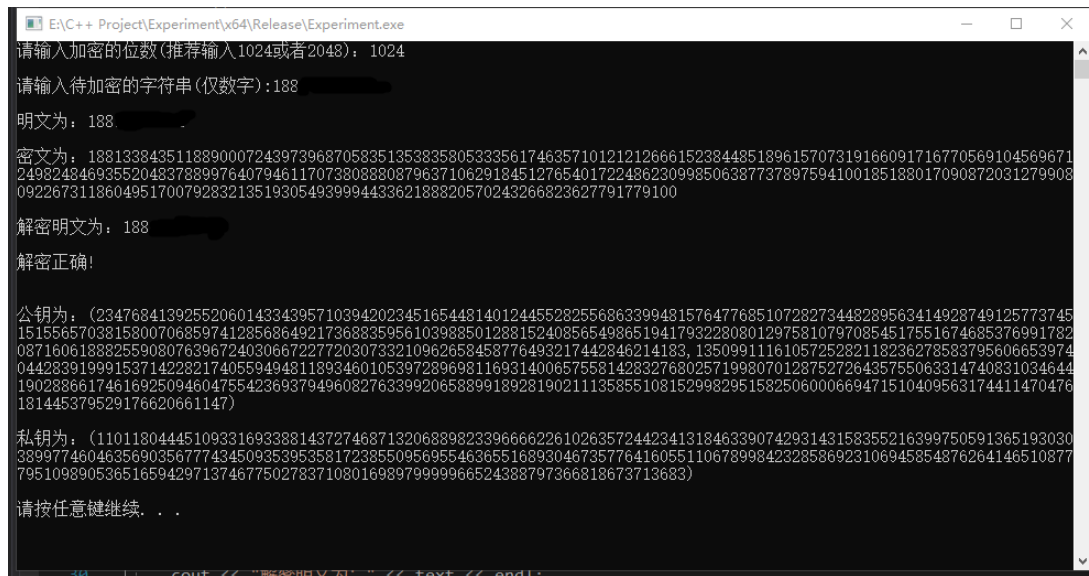
(13)还需要将代码生成的运行库调整为多线程(/MT)。



(14)接着根据第 2 部分添加上面写好的代码(Experiment 文件夹下)，重新编译，可能会有一些参数丢失数据的警告，暂且不用理会，如果使用 x64 位就不会有这个警告。



(15)添加上 RSA 的 C++代码，选择 Release x64 模式，可以在控制台中看到测试结果。



## 2.2 RSA 算法部分

这部分内容我直接参考毕业论文的内容。下面给出 RSA 加密体制。

**定理 3.3** RSA 公钥密码体制描述如下：

(1) 生成公钥与私钥

选择两个随机的大素数  $p$  和  $q$ ，并计算  $n = pq$  和  $\varphi(n) = (p - 1)(q - 1)$

选择一个随机数  $e$ ,  $1 < e < \varphi(n)$ , 并且满足  $\gcd(e, \varphi(n)) = 1$ , 计算  $d = e^{-1} \bmod(\varphi(n))$ .

公钥为  $(e, n)$ , 私钥为  $d$ .

(2) 加密

将给定的明文分成几部分，使得每一部分的长度小于  $n$ ，假定某一部分明文为  $m$ ，加密的计算式子写为： $c = m^e \bmod(n)$ .

(3) 解密

密文  $c$  对应的明文为： $m = c^d \bmod(n)$ .

首先要证明加密和解密是互为逆运算，由(2)得  $de \equiv 1 \bmod(\varphi(n))$ ，因此存在整数  $k$  满足  $de = k\varphi(n) + 1$ . 于是有

$$\begin{aligned}(m^e)^d &\equiv m^{k\varphi(n)+1} \pmod{n} \\ &\equiv m^{k\varphi(n)} \cdot m \pmod{n} \\ &\equiv 1 \cdot m \pmod{n} \\ &\equiv m \pmod{n},\end{aligned}$$

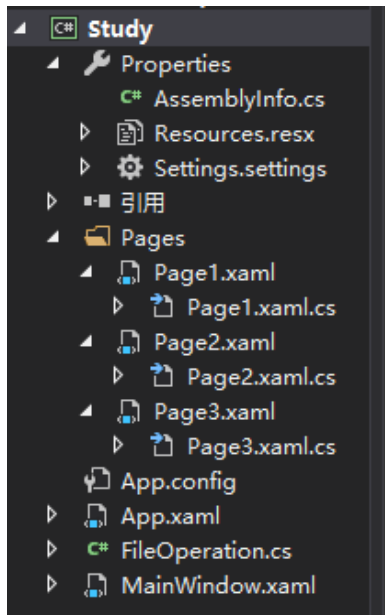
这就证明了加密和解密互为逆运算。

## 3. C#图形界面部分

这部分我最终敲定的是使用 WPF 来完成，因为 WPF 将界面和动作分离开来，界面部分完全使用 XAML，动作部分完全使用 C#，这对于熟悉 HTML 和 Java 的我来说上手是挺快的，下面简单说下 XAML 部分的功能。

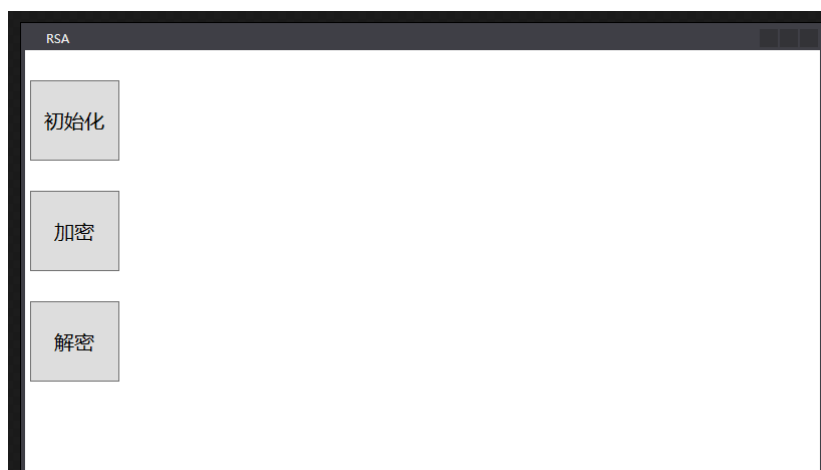
整个 C# 目录结构如下：





可以看到是主窗口外加 3 个页面，然后还有一个文件操作类，非常的简单明了。

主窗口用 Grid 分为两个部分，第一个部分是左边的三个按钮，宽度为 100，第二个部分是右边的空白界面，宽度为 700。第一个部分的三个按钮用了 stackpanel 排在一起，设置好字体和边距以及内容即可。第二个部分使用了一个 Frame，目的是生成三个 Page。



第一个 Page 就是一个 TextBlock，一个 TextBox 和一个按钮，宽度为 700。



第二个 Page 和第三个 Page 在样式上完全相同，只是文本上面会略有不同。



这里需要注意的是需要将每个部件，如 Button，如 Label 全都语义化命名好，给后面的事件调用带来很大的便利，然后就需要给所有的按钮增加点击事件。

在 MainWindows 的 C#文件中，我们首先要明确的是 MainWindows 需要实现的是点击三个按钮切换三个不同的 page，这里我查阅了资料以后借用了 Dictionary 以及 Navigate 来切换，详情只要看 MainWindows.xaml.cs 就能明白。

在 Page1 的 C#文件中，我们明确的是在 Textbox 中输入一个数字，然后点击初始化按钮，就可以在应用程序同层目录下生成“Big\_Integer.txt”，“Public\_Key.txt”，“Private\_Key.txt”三个文件。因此在代码中我就先获取输入值 getInput，判断是否为空以后以后，转换为数字，然后调用后面会提到的 dll 生成三个数字字符串，最后写入文件即可，具体更为详细的细节可以见后面。

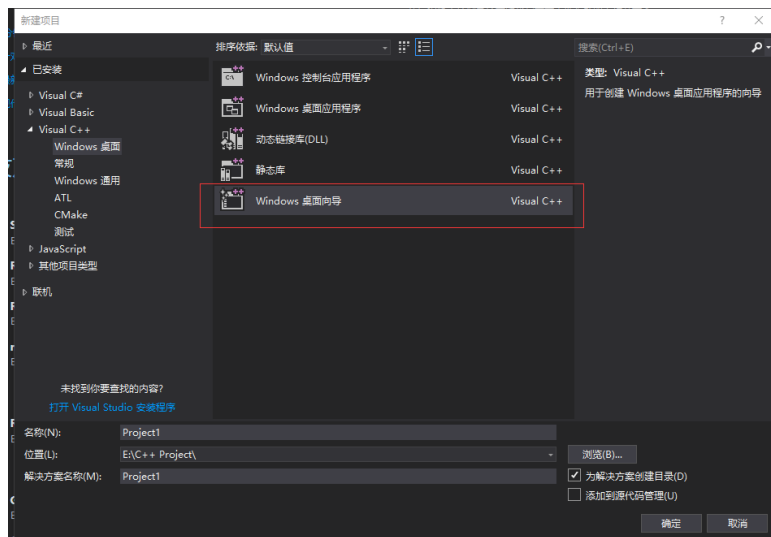
Page2 和 Page3 的 C#文件是类似的，所以直说一个就可以，这两个页面都是需要读入三个文件，点击以后生成一个文件，这里我使用了三个全局变量，使得这三个点击事件的函数都可以调用这些变量，再使用 dll 文件计算需要的数字字符串，最后写入到 txt 文件。

## 4. 整合部分与测试

### 4.1 RSA 导出 dll 文件

我查阅了资料以后发现 C#想使用 C++的算法的话要将 C++导出为 dll 文件，再在 C#中使用 `DllImport` 就可以了，因此需要先导出 dll 文件。

(1) 新建 C++项目，不一样的是这里我们选择新建的是 Windows 桌面向导。



(2) 确定以后根据向导，这里选择动态链接库和空项目。



(3) 创建以后导入在第二部分中的 `RSA.h` 以及 `RSA.cpp`，然后还需要导入 `MPIR` 库，这些部分完成以后，我们要对代码进行一些修改。

(4) 如果仔细看过去代码会发现，我这里将 C++中出现的 `String` 类型全部修改为 `LPCSTR` 类型，这是因为 C#接受 C++的 Dll 文件时只有 `LPCSTR` 类型才能传递 `String` 类型的参数，与此相关的是还有一些本来可以直接用 `mpz_class` 作为参数的类型也被我修改为 `LPCSTR`，这是为了方便 C#使用字符串类型作为参数调用 C++的 dll 文件。

(5) 除开这些以外，还有一些修改。

```
class __declspec(dllexport) RSA
{
public:
    RSA();
};
```

- (6) 新建了一个 RSAWrapper.h 和 RSAWrapper.cpp 文件，里面主要是实现了一个静态指针变量 g\_pObj，它相当于一个委托人，C#的所有计算请求都经过这个指针变量去 dll 中去运算，再将结果返回给 C#，按照格式写即可。

```
#pragma once
#include "RSA.h"

namespace RSAWrapper
{
    extern "C" __declspec(dllexport) LPCSTR Cal(LPCSTR text, LPCSTR key, LPCSTR integer);
    extern "C" __declspec(dllexport) void InitialNumbers(int InputBits);
    extern "C" __declspec(dllexport) LPCSTR GetBigInteger_n();
    extern "C" __declspec(dllexport) LPCSTR GetEncryption_Index();
    extern "C" __declspec(dllexport) LPCSTR GetDecryption_Index();
}
```

- (7) 在 C#中，为了导入 dll 文件，需要这样子写

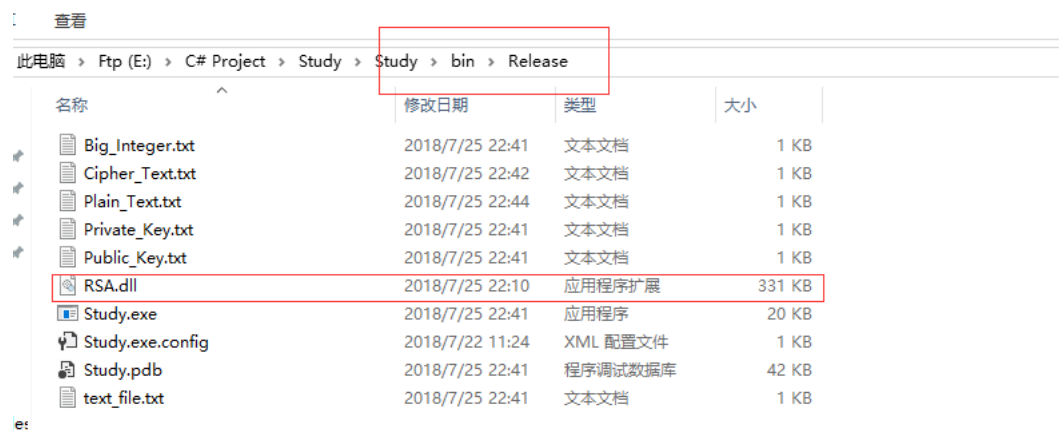
```
[DllImport("RSA.dll")]
public static extern void InitialNumbers(int InputBits);
[DllImport("RSA.dll")]
public static extern IntPtr GetBigInteger_n();
[DllImport("RSA.dll")]
public static extern IntPtr GetEncryption_Index();
[DllImport("RSA.dll")]
public static extern IntPtr GetDecryption_Index();
```

这里将 LPCSTR 类型再转为 IntPtr 类型，我认为可能是我后面会提到的 bug 时不时发生的原因，可是我实在不知道是为什么了。详情可以观察代码，每一个 page 需要什么方法就导入什么方法即可。

- (8) 最后是一些文件操作方法，代码在 FileOperation.cs，写入文件的方法和 Java 类似，都是使用流文件的形式按行写入即可。
- (9) 上传文件是用在 Page2 和 Page3 的选择按钮中，点击它可以打开文档目录，选择好文件以后就打开文件，读入文件中的数据到变量中，用的是 OpenFileDialog 和 StreamReader 即可。
- (10) 最后还需要解释的一点就是在 Page2 和 Page3 里面对于变量使用了编码转换，这是 IntPtr 类型相适应的转换代码。

```
//调用C++的dll生成三个数字，大整数n，公钥d，私钥e
string Big_Integer = Encoding.UTF8.GetString(Encoding.Unicode.GetBytes(Marshal.PtrToStringAuto(GetBigInteger_n())));
string Public_Key = Encoding.UTF8.GetString(Encoding.Unicode.GetBytes(Marshal.PtrToStringAuto(GetEncryption_Index())));
string Private_Key = Encoding.UTF8.GetString(Encoding.Unicode.GetBytes(Marshal.PtrToStringAuto(GetDecryption_Index())));
```

- (11) 写完以后，将 RSA 部分的代码直接导出 Release x64 的 dll 文件，然后将该文件放到 C# 工程应用程序中就可以了。



后面每次更新 dll 文件的时候需要重新复制粘贴到 Release 文件夹就可以了，然后生成 C# 工程的解决方案，运行的时候就会调用上了。

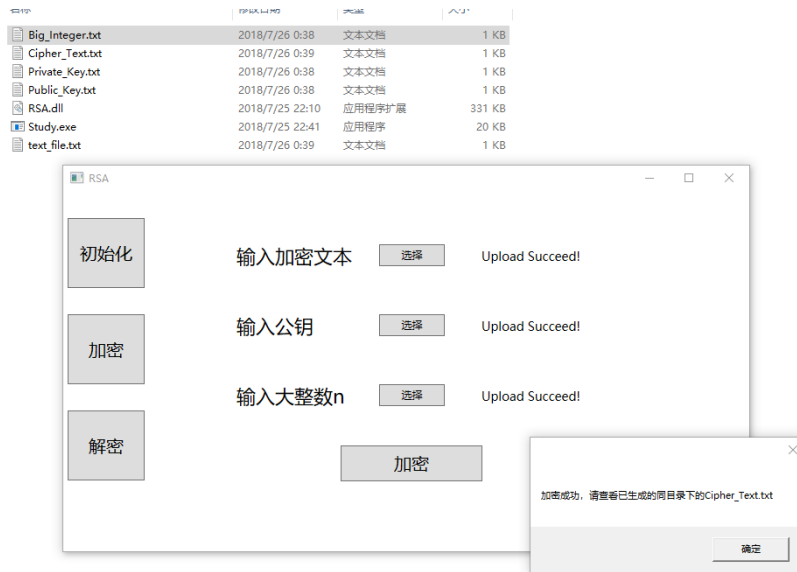
最后说一下我做出来的这个程序出现了还解决不了的问题，那就是对于某些特殊的测试用例，无法重新生成准确的明文，举个例子(这也是程序的使用方法)，先生成一个 1024 位文件。



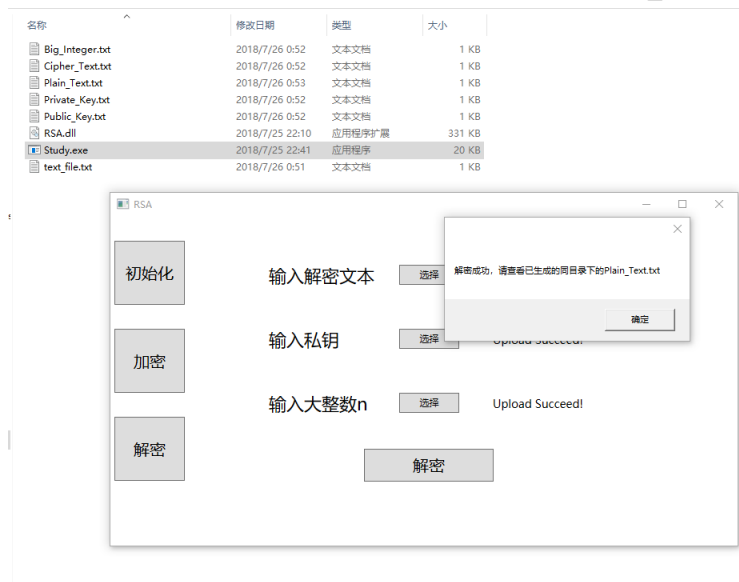
然后再 text\_file.txt 中输入文本 “57477815478418789454” (随便输的)



点击到 “加密”，输入相应的三个文本，生成 Cipher\_Text.txt，再生成



最后点击“解密”界面，输入三个文件，点击解密，得到 Plain\_Text.txt



这里点开 Plain\_Text.txt 会发现能生成原来的明文数字，但是比较奇怪的是，如果在明文较短，而加密位数较长的时候，有可能会生成空白文件。



比如 11 位数字(我用了我的手机号做测试)，在 1024 位的文件下无法生成原来的明文，但在 256 位的文件下可以生成原来的明文，经过 7.26 一天的 debug，我修改了如下的部分：



名称	值
返回 System.Runtime.InteropServices.Marshal	""
返回 System.Text.Encoding.GetBytes	{byte[0]}
返回 System.Text.Encoding.GetString	""
▸ this	{Study.Page3}
▸ sender	{System.Windows.Controls.Button: 解密}
▸ e	{System.Windows.RoutedEventArgs}
result	""

这里就直接返回了一个空的字符串了，所以最后写入的也是一个空字符串，为什么是空字符串我想了特别久，但是实在是没有头绪，如果能解决这个地方的话整个程序就完成了。

最后特别感谢老师给我这个机会尝试写这个程序，时间有限，8 月份我就要接着复习了，没有太多时间处理它了，如果后面还有机会的话(应该要到 1 月份了)，我会尝试用 Qt 重写一遍界面程序，Qt 能直接调用 C++ 的类，写起来应该比较简单一些，而且这次为了尝试这个 bug 的问题查了很多网上的资料，C# 调用 C++ 的 dll 传递字符串一个很大很困难的问题，或许未来我学了更多计算机知识以后能解决这里的问题。