北京电子科技学院（BESTI）

**实 验 报 告**

课程：信息安全系统设计 班级：2214

姓名：徐鹿鸣 学号：20221414

成绩： 指导教师：娄嘉鹏 实验日期：2024/11/03

实验密级：无 预习程度：全预习

实验时间：3，4节 仪器组次：无 必修/选修：必修

实验序号：实验二

实验名称： 密码算法实现

实验目的与要求：掌握常见商用密码算法的原理与实现。基于Arm等平台和国产化操作系统使用C语言编程实现SM2、SM3、SM4算法；对比分析算法实现的正确性和效率。

实验仪器：

|  |  |  |
| --- | --- | --- |
| 名称 | 型号 | 数量 |
| 计算机 |  | 1 |
|  |  |  |

实验内容、步骤与体会（附纸）：

**1. 实验内容**

基于国产化平台使用C语言编程实现SM2、SM3、SM4等算法；

**2. 实验步骤**

**2.1步骤1** 在 Ubuntu中调试运行商用密码检测中心供的源代码，至少运行SM2，SM3，SM4代码。使用GmSSL命令验证你代码的正确性。

**2.1.1 操作内容**

进入[商用密码检测中心](https://www.scctc.org.cn/xzzx/sfydm/ydmxz/)下载源代码PDF文件。将其转为word文件。由于转换不是很准确，所以先在VS2022中运行测试，修正错误后再拷贝到Ubuntu中运行。

在测试中发现源代码均无main函数，需要自己编写。同时注意到自检函数。所以首要的目标就是编写一个简单的调用自检函数的mian函数进行测试，测试成功后就可以模仿自检函数自己构建数据、调用函数、进行测试了。

按照先易后难的顺序，依次完成SM3、SM4和SM2的内容。其中SM3较为简单，这里不多论述。

首先要判断这是ECB模式下的SM4实现。在测试中发现SM4程序对输入与输出有严格的要求，这需要对数据进行填充与解填充。这里采用PKCS#7填充方法，在SM4代码内部也加入了相关填充功能，同时使用AI辅助生成了python脚本实现填充相关功能。

实际测试时，使用命令行传入要测试的数据”xlm”，输出十六进制密钥、密文和解密的明文。使用测试数据生成明文文件，使用pad.py将明文文件填充生成填充后的明文文件，用gmssl命令和十六进制密钥加密填充后的明文文件，查看密文是否与程序输出的密文一致。如果一致，则程序功能验证成功。

对于SM2，情况复杂很多。

首先要在Ubuntu中下载编译Miracl库。这里需要编译32位的Miracl库，所以需要预先准备32位环境，可以参考[这个对话中的命令](https://kimi.moonshot.cn/share/cser2rgjdo7r0h8fqahg)。从GitHub上下载Miracl库的zip包到一个新的文件夹，使用unzip -j -aa -L MIRACL-master.zip命令无视文件结构解压。执行bash linux命令即可编译32位的Miracl库。使用./pk-demo命令测试安装是否成功。在miracl.h的第96行取消注释，避免找不到compare函数。

为了在不同地方都能链接这个库，我们将其拷贝到用户文件夹的指定位置。可以参考以下命令：

cp ./miracl.a /usr/lib/libmiracl\_32.a

mkdir /usr/include/miracl

cp \*.h /usr/include/miracl

采用这个方法编译Miracl库后，之前源代码的相关头文件需要修改，一般是加上”miracl/”。编译时采取32位编译即加上-m32，链接时使用-lmiracl\_32。

在实践中发现，sm2密钥交换代码在vs2022中可以正常运行，但在Linux环境下会报错:SM2\_KEY\_EX.c:95:23: error: lvalue required as unary ‘&’ operand。将源代码中将错误代码

memcpy(IDlen, &(unsigned char)ELAN + 1, 1);

memcpy(IDlen + 1, &(unsigned char)ELAN, 1);

改为：

memcpy(IDlen, (unsigned char\*)&ELAN + 1, 1);

memcpy(IDlen + 1, (unsigned char\*)&ELAN, 1);

由此可以完成调试运行商用密码检测中心供的SM2，SM3，SM4源代码。其中使用GmSSL命令验证的部分需要根据自己的main函数实现与测试数据进行调整，具体的可以参考下面操作结果截图。

**2.1.2操作结果**

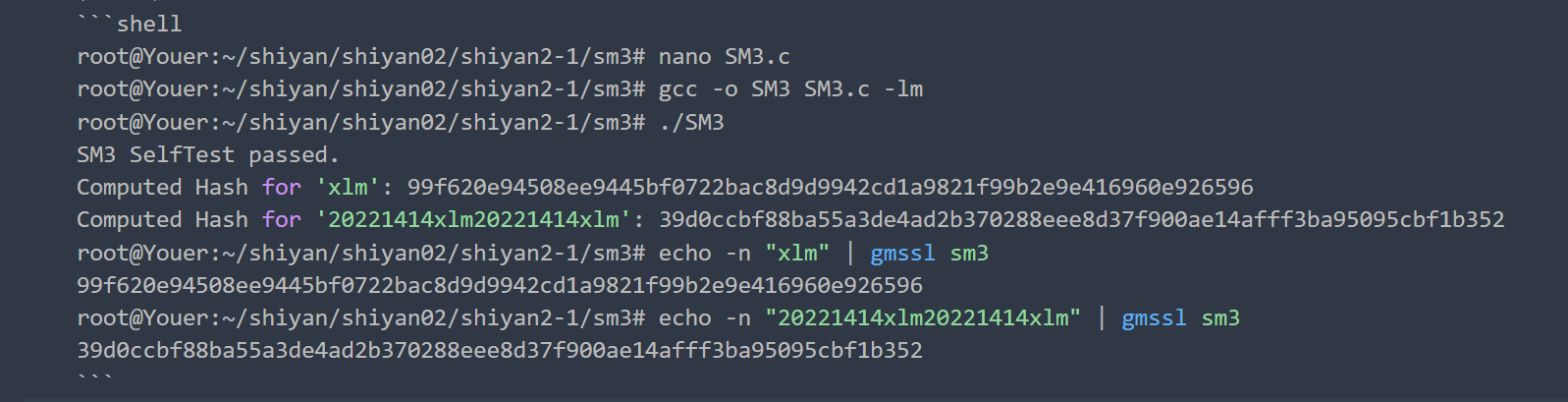


图 1 SM3程序运行与验证过程

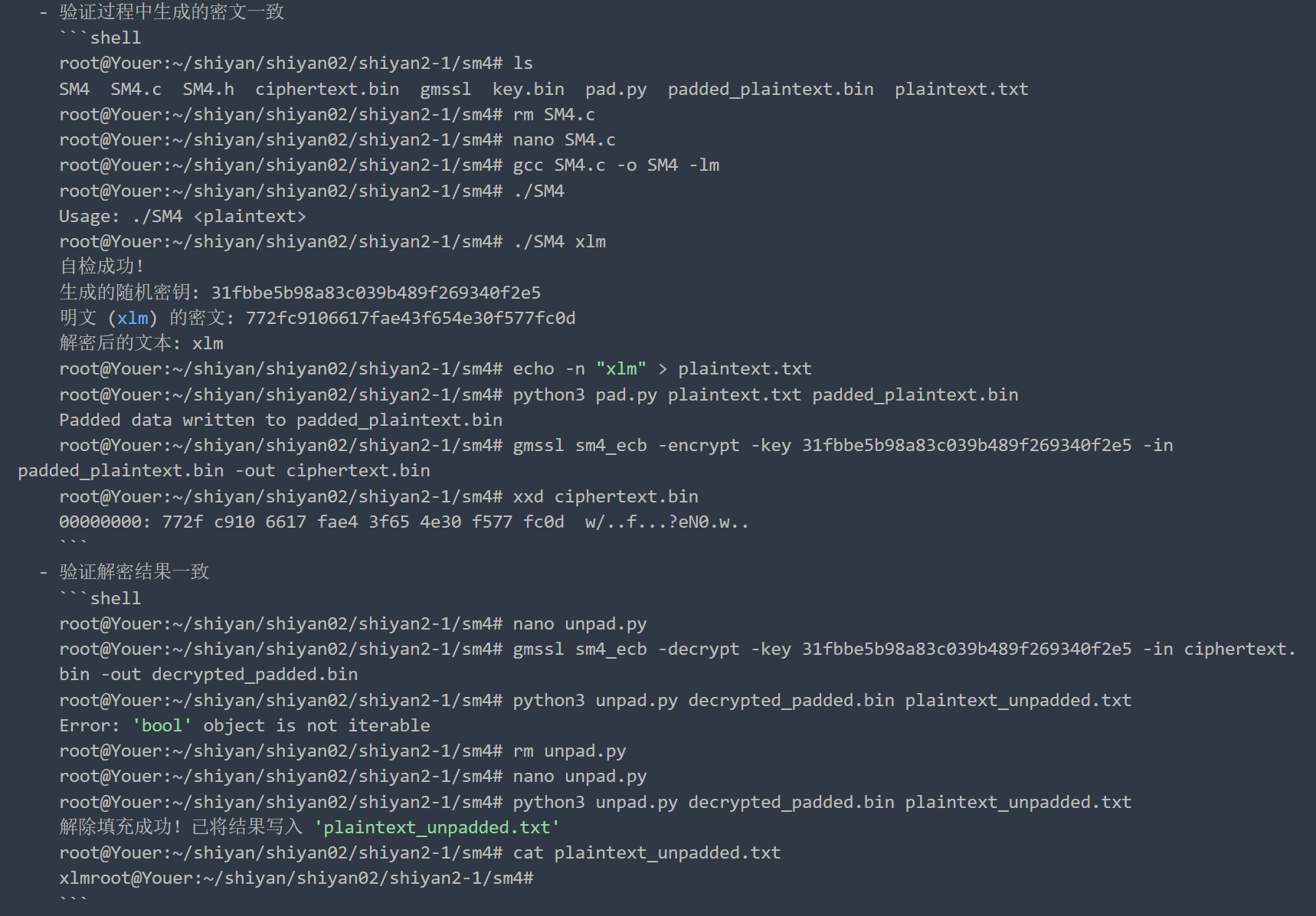


图 2 SM4的程序运行与GmSSL命令验证过程



图 3 SM2程序运行过程

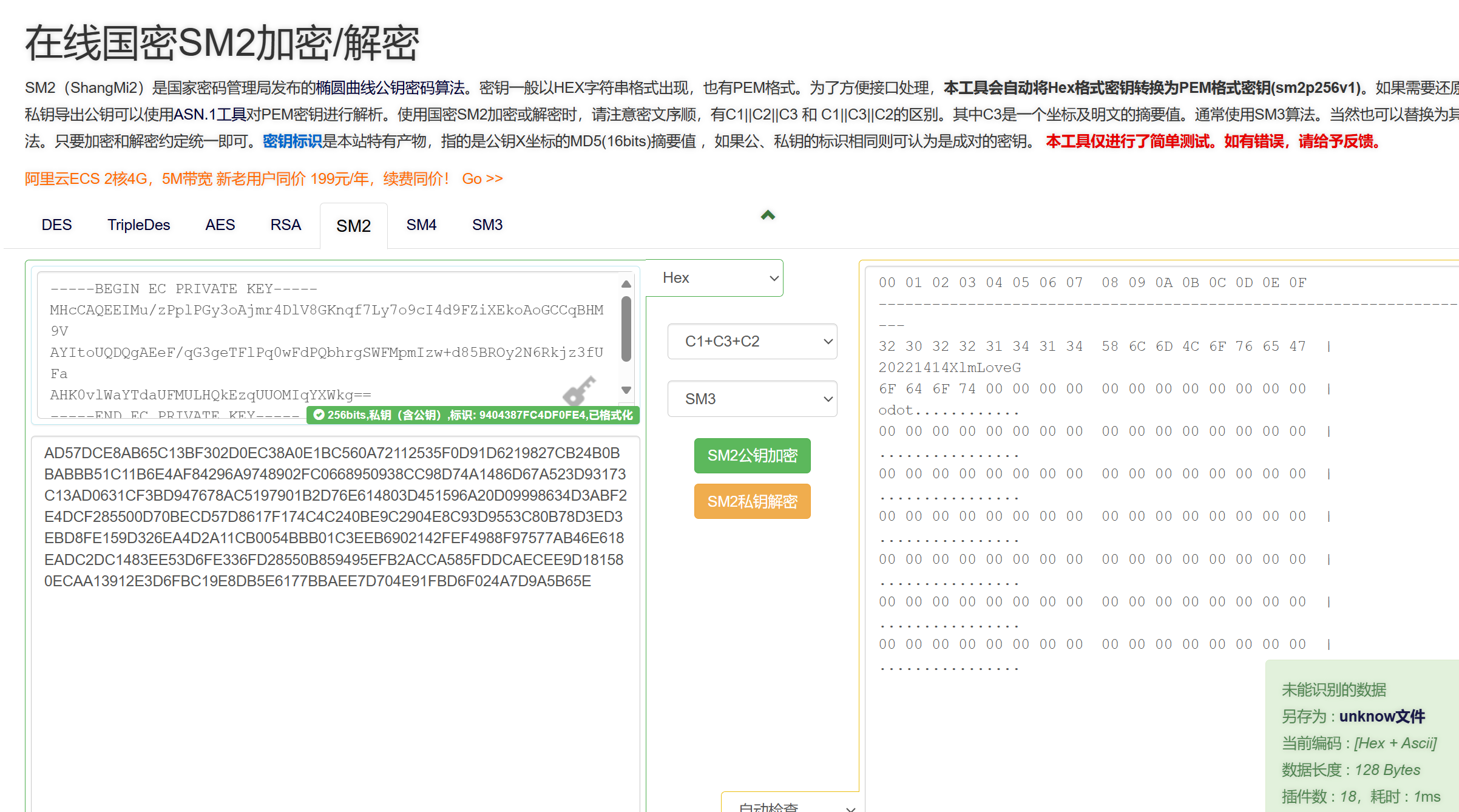


图 4 SM2程序验证（通过[在线网站](https://the-x.cn/cryptography/Sm2.aspx)）

**2.2步骤2** 在[密标委网站](http://www.gmbz.org.cn/main/bzlb.html)查找SM2，SM3，SM4相关标准，分析步骤一中代码实现与标准的对应关系

**2.2.1 操作内容**

在[密标委网站](http://www.gmbz.org.cn/main/bzlb.html)通过标准中文名称（例如“SM2”）和标准状态(“现行”)快速查找SM2，SM3，SM4相关标准。

通过目录和AI的总结把握标准的大体内容，再与代码进行对比。从参数命名和数值的一致、算法实现的一致、测试方法与数据的一致、输入输出的格式要求的一致等方面寻找对应关系。可以使用AI来辅助理解代码逻辑。

**2.2.2操作结果**



图 5 查询标准的方法

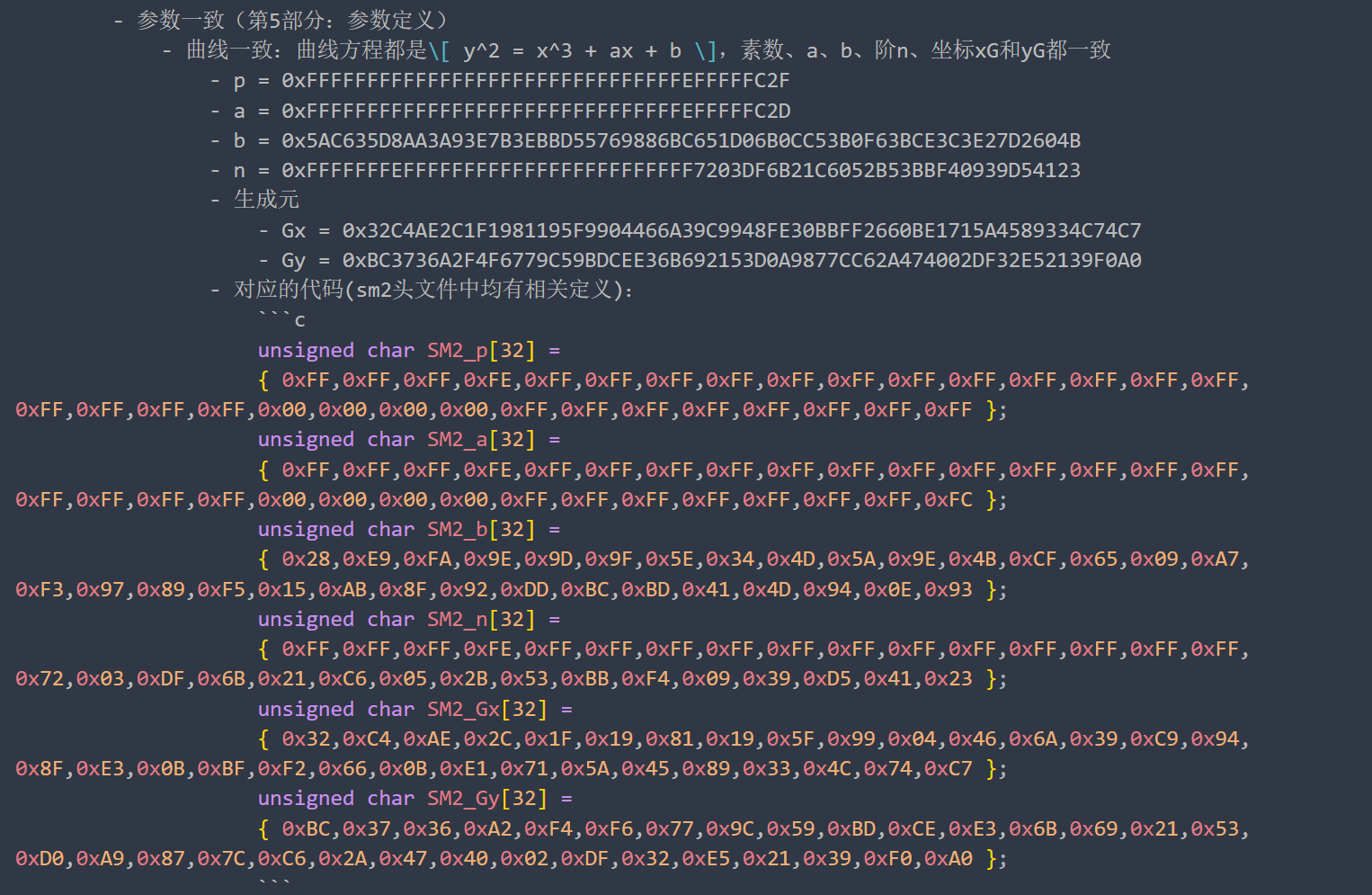


图 6 SM2曲线参数的一致



图 7 AI辅助理解代码([对话链接](https://kimi.moonshot.cn/share/csf21hcrmee54nn7ne30))

**2.3步骤3** 在Ubuntu中调试运行教材提供SM2，SM3，SM4源代码，使用GmSSL命令验证你代码的正确性。

**2.3.1 操作内容**

下载并解压云班课/资源/课程参考资料/Windows.C.C++.加密解密实战.rocsrc.zip并结合教材确定代码所属内容。（见[我的GitHub仓库](https://github.com/youer0219/Information-Security-System-Design-Assignment)中“实验/实验二/代码/教材源代码”）

根据步骤一的经验按SM3、SM4、SM2的顺序进行实验。

在SM3的实践中，4-1的代码会报错。修复后的代码见实验报告第三部分。同时注意，SM3需要在32位环境中运行，所以编译时要加上-m32。同时，包括之后的代码，预编译头文件pch.h与其cpp文件都会被删除，因为这些并没有被使用。

4-1到4-3的测试内容基本一致，所以用gmssl sm3 命令输出哈希值再对比即可验证。4-4的密钥为string类型，转换后的长度也不符合gmssl命令要求，所以在[在线网站](https://www.lddgo.net/encrypt/hmac)中验证。

这次的SM4代码调试与运行并没有难度，正常编译执行即可。

在SM2的实践中，由于步骤一已经安装了Miracl库且可以在任何地方链接，这里就不再按照教材的方法进行（教材的方法也不是适用于Linux的方法）。所以需要对源代码的头文件进行修改。同时修复重复定义报错（具体修复见实验报告第三部分），对照教材修复一些乱码。

由于SM4和SM2代码均有自检函数，不需要再通过GmSSL命令进行验证。由此便可完成步骤三。

**2.3.2操作结果**

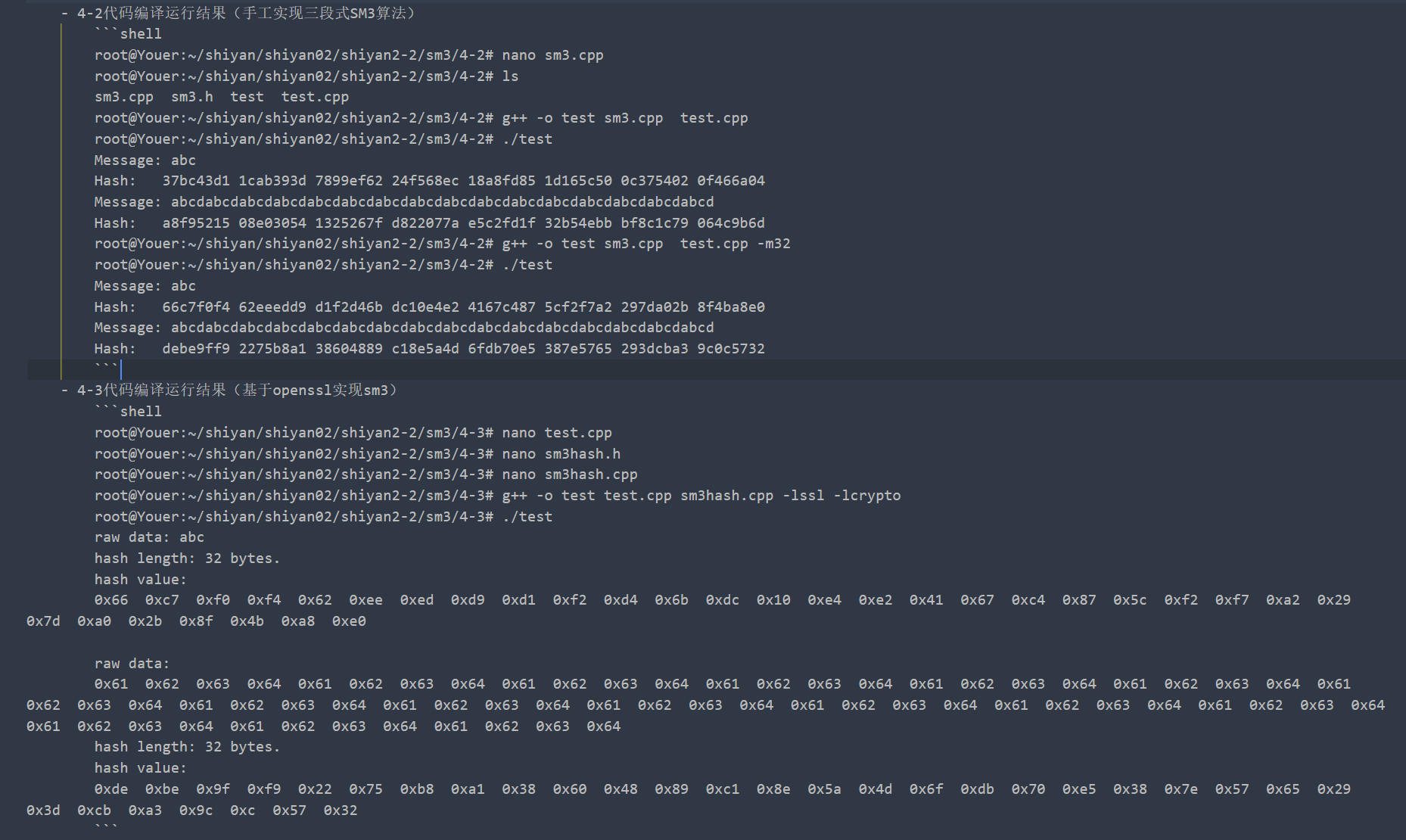


图 8 SM3程序运行结果

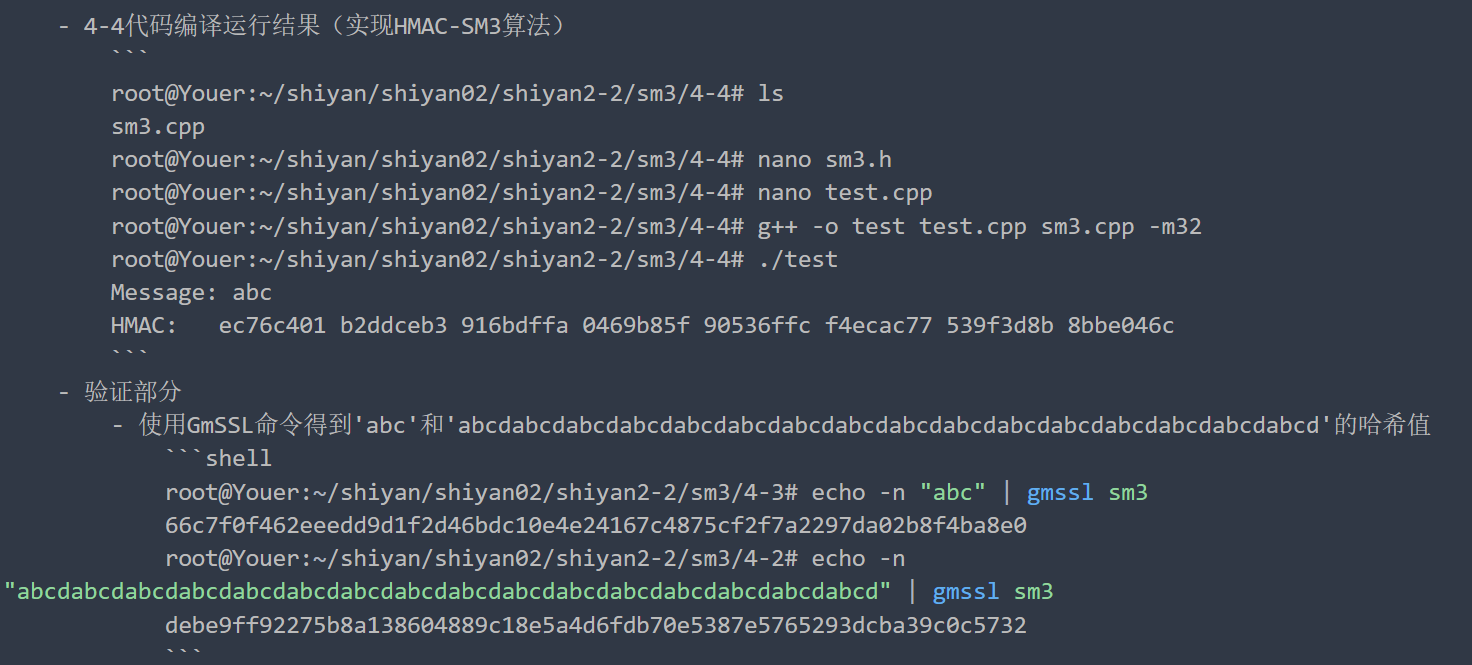


图 9 SM3 4-4运行结果 和gmssl命令验证结果

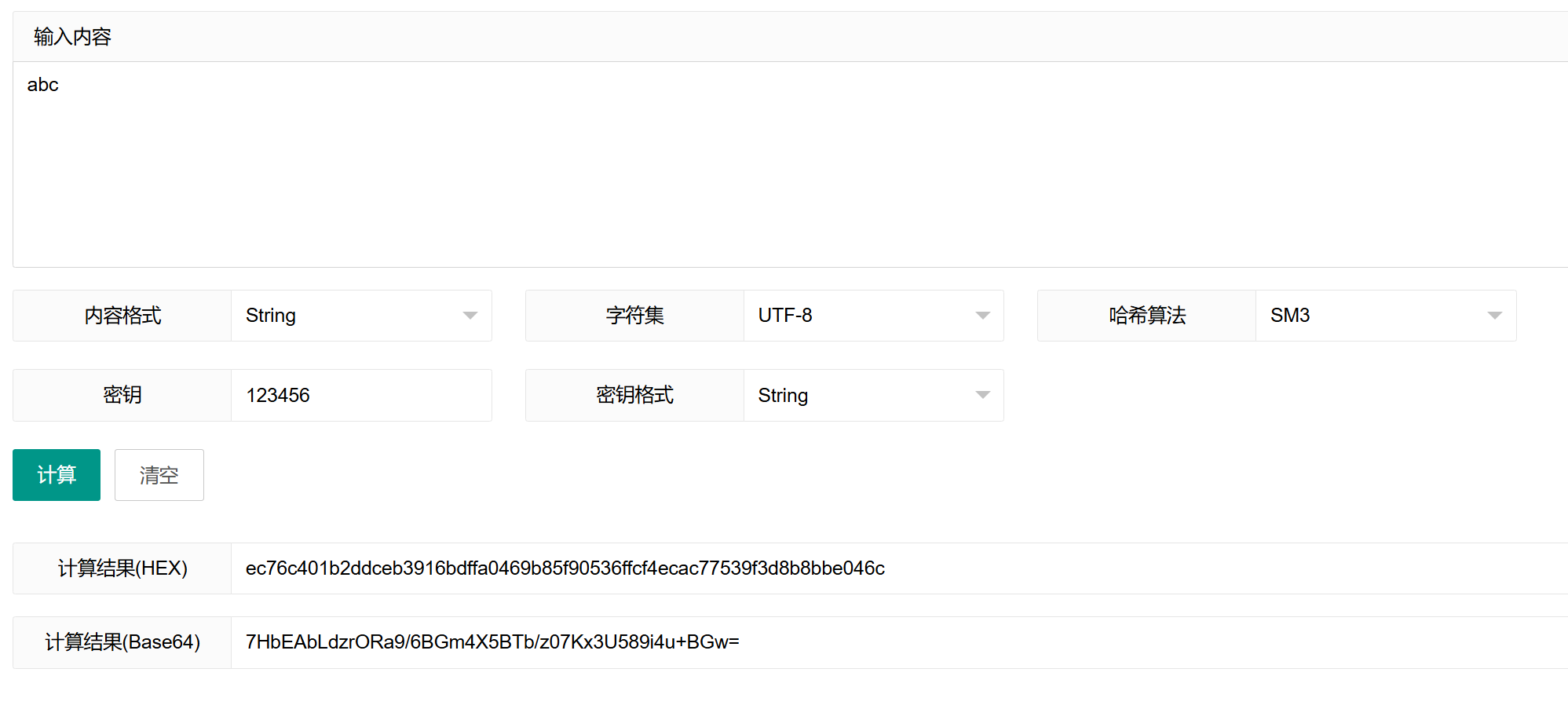


图 10 对SM3 4-4的验证（[在线网站](https://www.lddgo.net/encrypt/hmac)）

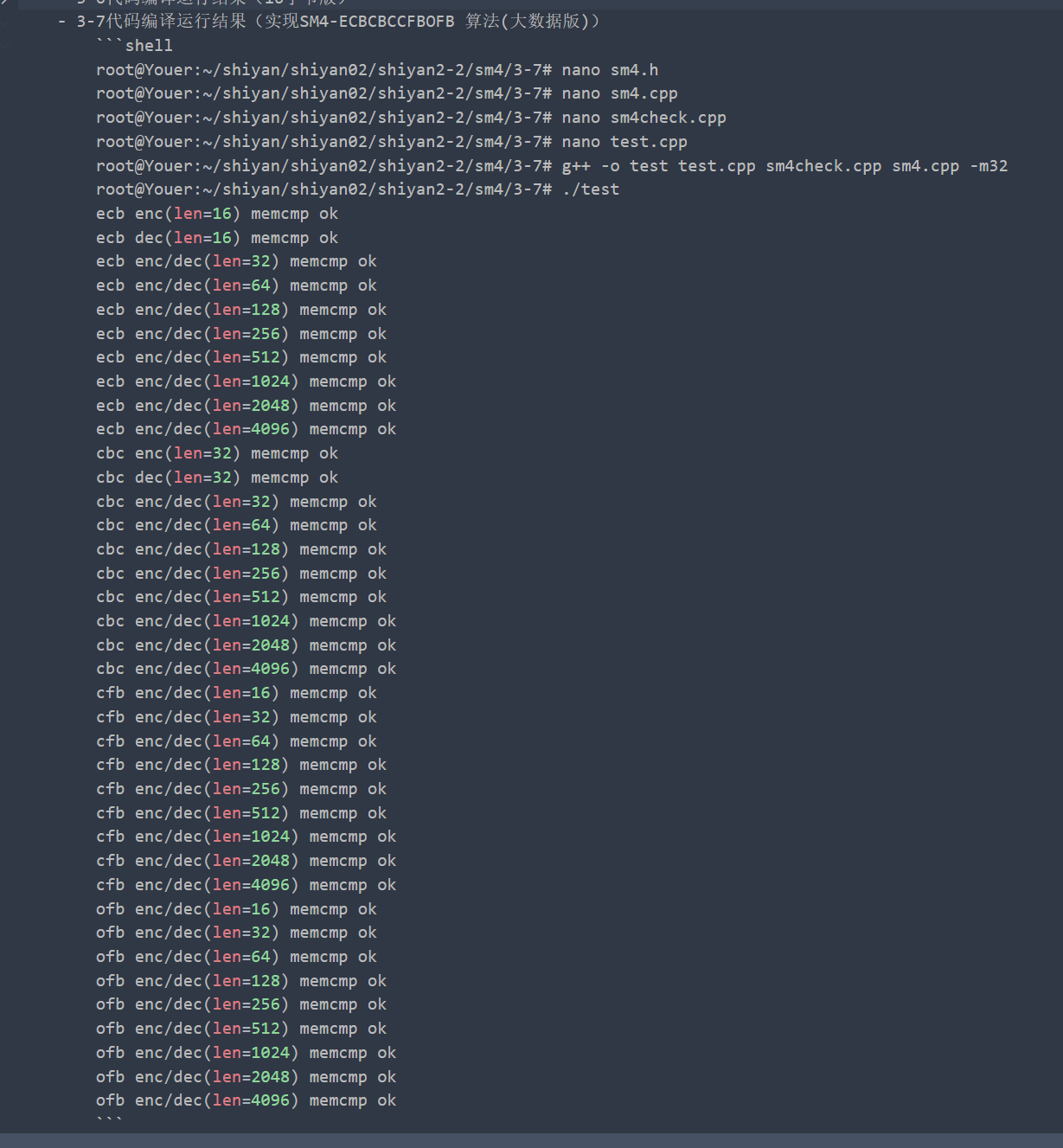


图 11 SM4运行结果

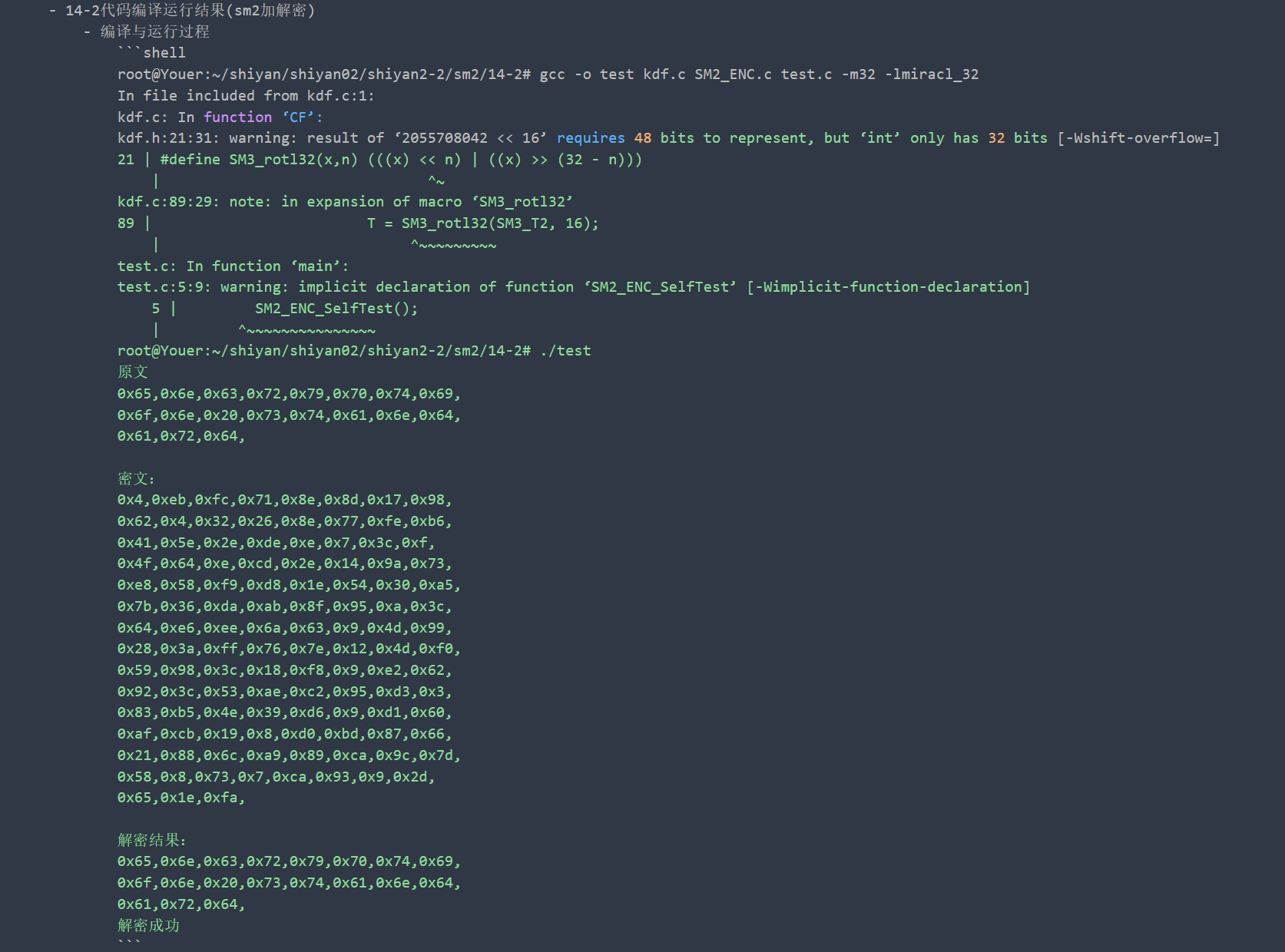


图 12 SM2加解密的运行结果

**2.4步骤4** 在[密标委网站](http://www.gmbz.org.cn/main/bzlb.html)查找SM2，SM3，SM4相关标准，分析步骤三中代码实现与标准的对应关系。

**2.4.1 操作内容**

进入[商用密码检测中心](https://www.scctc.org.cn/xzzx/sfydm/ydmxz/)下载源代码PDF文件。将其转为word文件。由于转换不是很准确，所以先在VS2022中运行测试，修正错误后再拷贝到Ubuntu中运行。

**2.4.2操作结果**

啊

**2.1步骤1** 在 Ubuntu中调试运行商用密码检测中心供的源代码，至少运行SM2，SM3，SM4代码。使用GmSSL命令验证你代码的正确性。

**2.1.1 操作内容**

进入[商用密码检测中心](https://www.scctc.org.cn/xzzx/sfydm/ydmxz/)下载源代码PDF文件。将其转为word文件。由于转换不是很准确，所以先在VS2022中运行测试，修正错误后再拷贝到Ubuntu中运行。

**2.1.2操作结果**

啊

**3．实验体会**

**3.1 调试中出现的问题及解决过程**

**3.1.1 OpenSSL中sm2代码解密失败，提示ASN.1编码出错问题**

**报错信息：**

        root@Youer:~/shiyan/shiyan01/shiyan1-2/task01/test\_sm2# ./sm2\_decrypt sm2\_private.pem encrypted\_file.bin decrypted\_file.bin

        OpenSSL error: error:068000A8:asn1 encoding routines::wrong tag

        OpenSSL error: error:0688010A:asn1 encoding routines::nested asn1 error

        OpenSSL error: error:1A800068:SM2 routines::reason(104)

**具体表现：**

* **无法通过解密程序解密，但可以通过openssl命令解密**
* **AI的代码无法解决这一bug，且总是重复一些已经排除的可能**

**解决问题的关键在于明确根本问题，之前尝试解决时猜测了很多解决思路，比如转换密文格式等，但这些只是猜测，最终浪费了自己的时间。这让我想到了[“XY问题”](https://coolshell.cn/articles/10804.html)：没有去问怎么解决问题X，而是去问解决方案Y应该怎么去实现和操作。**

**同时不要就一个代码问AI太久，AI可能跳不出原有的框架，就会一直给出错误代码。要学会及时开启新对话。**

**3.1.2 GmSSL中SM2私钥格式问题**

**在GmSSL的SM2生成签名时遇到私钥读取失败的问题。在处理过程中发现是私钥格式的问题，但没能很好的总结经验，而是重复询问AI-报错-继续询问的循环。**

**其实在之前就已经实现了SM2加解密的程序，已经有了读取私钥的代码实现。可以直接把相关代码拷贝过来给AI作为参考，这样就可以快速解决这个问题了。**

**3.2 心得体会**

**实验1-1部分还好实验1-2部分即代码编程部分是最难的部分，这一过程简直就是“赛博炼丹”，以我的知识储备来说（特别是刚开始时），很难判断AI的回答正确与否，只能重复“询问-报错-继续询问”的循环，期望某一次能够解决问题。**

**但在这一过程中，我也有一些经验与教训。有些是针对这一次实验的，也有一些是通用的方法。**

**首先是对于调用openssl库编程的部分，一定要知道openssl本身不是直接支持sm2等算法，需要调用EVP接口来实现相关功能。同时其不存在”openssl/sm2.h”等头文件，这些是混用了gmssl的代码。**

**其次是对于调用gmssl库编程的部分，这里与openssl不同，不应该使用evp接口，同时也不存在”gmssl/evp.h”的头文件。同时其错误处理的头文件是”gmssl/error.h”而不是”gmssl/err.h”，不要混用了openssl的代码。**

**总结一下，就是要知道这些库对相关算法的支持形式，与能够分辨容易混淆的代码。**

**同时，也有对AI提问的经验教训的总结。3.1中其实已经提到了一些经验：要明确根本问题，避免“XY问题”；及时开启新对话；利用好已经成功的代码做参考。**

**但还有一些经验需要总结：根据反馈及时调整自己的提问，知道自己在做什么，避免AI重复犯错浪费时间；如果C语言代码不好实现，可以考虑换python实现程序（但要注意实验要求）等。**

**3.3 实验改进建议**

**建议告诉大家调用openssl和gmssl编程时有哪些方法与注意事项，或者早一些给出示例代码。**