密码学实验五、 ElGamal 签名算法实验报告

班级 572221 学号 LK123425 姓名 黄睿扬

实验目的	1. 掌握 ElGamal 签名算法； 2. 利用 C、C++、Java 实现 ElGamal 签名算法。
实验要求	1. 提交实验报告； 2. 提交实验代码。
实验内容	1. 实现 ElGamal 签名算法中的公共参数； 2. 实现 ElGamal 签名算法中的公、私钥； 3. 实现 ElGamal 签名算法中的签名和验证； 4. 展示实验结果。
实验环境	1. 系统：Windows 11（64 位）； 2. 处理器：AMD Ryzen9 7945HX（16 核）； 3. 显卡：NVIDIA GeForce RTX 4060（8GB） 4. IDE：Microsoft Visual Studio 2022（x64）； 5. 语言：C++、C 语言； 6. 环境：MinGW。
实验步骤	一、代码部分 1. ElGamal 签名算法概述（Introduction） ElGamal 签名算法是一种基于离散对数问题的公钥加密体系中的签名算法，由塔希尔·埃尔伽马尔（Taher Elgamal）于 1985 年提出。该算法的安全性基础在于离散对数问题的计算难度，这使得在有限的计算资源下解决这一问题变得不可行。ElGamal 签名算法在数字签名领域广泛应用，尤其在确保文档或消息未被篡改以及验证身份方面非常有效。 2. 算法原理（Theory） ElGamal 签名算法包括三个基本步骤：密钥生成、签名生成和签名验证。 （1）密钥生成 • 首先，选择一个大素数 p 和一个原根 g（在模 p 的剩余类中 g 的阶等于 p−1）；

- 然后，随机选择一个整数 a 作为私钥，其中 a 的取值范围是（1，p−2）；
- 计算公钥 $A \equiv g^a \bmod p$，并公开（A，g，p）。

（2）签名生成

- 对于一个给定的消息 M，首先计算该消息的哈希函数值 h（M）；
- 接着选择一个随机数 b，该随机数需满足与 p−1 互质；
- 计算 $B \equiv g^b \bmod p$，B 是签名的一部分；
- 接着，计算 $Sig \equiv (h(M) - a \times B)\, b^{-1} \bmod (p-1)$，这里的 b^{-1} 是 b 在模 p−1 下的逆元；
- 签名的结果为（Sig，B）。

（3）签名验证

- 收到消息 M 后，首先计算该消息的哈希函数值 h（M）；
- 然后，计算左值，$LHS \equiv A^B \times B^{Sig} \bmod p$；
- 接着，计算右值，$RHS \equiv g^{h(M)} \bmod p$；
- 如果左值等于右值，则签名被认为是有效的。

3. 安全性基础（Safety Basement）

ElGamal 签名的安全性依赖于离散对数问题的困难性。如果一个攻击者试图伪造签名而不知道私钥 a，他必须解决离散对数问题，这在实际中是不可行的，特别是当 p 足够大时。此外，每次签名时选择不同的 b 对安全性至关重要；如果重用 b，将可能导致私钥的泄露。

ElGamal 签名算法的一个重要特点是它的非确定性，即对于同一消息的多次签名会产生不同的 B 和 Sig 值，这主要是由于每次都随机选择了 b，这增加了算法的安全性，因为无法通过比较从不同签名中推断出任何有用信息。

4. 代码要点（Key of Codes）

（1） 大整型（Hugeint）类

考虑到一开始的几次实验，我仅仅依赖于 long long 类型来实现大数运算。尽管这种方法能够处理相对较长数位的输入、输出和基本运算，但在现实的加密算法中，这样的实现仍然存在一些不

	足。首先，long long 类型的范围仍然有限，无法满足加密算法中所需处理的极大数字。其次，由于缺乏对大数的专门优化，性能可能受到限制，无法满足对计算速度和资源利用的高要求。 在实际的加密算法中，安全性至关重要。使用较小的数据类型，可能无法提供足够的安全性保障，因为它们很容易受到穷举搜索等攻击的影响。而使用大素数作为 ElGamal 签名算法中的模数，则可以显著提高算法的安全性，因为它增加了攻击者解决离散对数问题的难度。 因此，基于上次实验的成功实现，我继续沿用上次自己写的一个 Hugeint 的大整型类，该类可以存储最长 1024 位的数据（基于 2^{1024} 的数据折合成 10 进制数，取对数约为 309 位，远远满足其基本四则运算和取模运算等），并且针对字符串类型的数据完善了常用的四则运算、不等关系、取模、次方等运算，同时还在 SHA-1 的实现部分为此次实验额外写了十六进制字符串转十进制 Hugeint 类的函数，保证在该实验中 Hugeint 类能够支持签名和加密算法的正常运行。 （2） 哈希函数 SHA-1 的实现 这部分代码是用于实现 SHA-1 哈希算法，其中包括了几个关键的函数用于计算 SHA-1 哈希值，以及一个额外的函数用于将十六进制字符串转换为 Hugeint 类型的数值。每个函数的功能和实现原理如下： a) Ft 函数，此函数是 SHA-1 算法中的布尔函数，根据不同的轮次 t 选择不同的函数来处理输入的位。根据 SHA-1 的规范： • 当 0≤t≤19 时，使用（B and C） or （not B）and D）； • 当 20≤t≤39 和 60≤t≤79 时，使用 B xor C xor D； • 当 40≤t≤59 时，（B and C） or （B and D） or （C and D）。 这些布尔操作提供了基本的逻辑功能，用于在生成消息摘要时混合输入数据。 b) SHA_PAD 函数，该函数用于为原始消息填充，以使其长度满足 SHA-1 算法的要求。SHA-1 要求消息的长度在填充后是 512 位的倍数。此函数将：

	• 添加一个 1 位，然后添加足够的 0 位，直到消息长度达到 512 位的整数倍且仅剩下 64 位空间。 • 在最后的 64 位中存储原始消息长度（单位为位），这部分是大端存储（big-endian format）。 c) ROTL 函数，该函数实现了循环左移操作，即将输入的位向左移动 s 位，超出的位重新从右侧开始填充。循环左移是 SHA-1 中用于生成复杂度和不可预测性的基本操作之一。 d) ADD 函数，该函数实现了 32 位的模 2^32 加法，用于 SHA-1 中的部分加法步骤。它逐位计算和，同时处理进位。 e) DO 函数，这是 SHA-1 算法的核心处理函数，它执行主要的哈希计算循环。算法首先将 512 位的块分解为 16 个 32 位的字（Wt 数组），然后扩展到 80 个 32 位的字。接着使用 Ft 函数和 ADD、ROTL 函数进行多轮处理，更新内部状态（A, B, C, D, E）。最终，这些状态被累加到算法的当前哈希值中（Ht 数组）。 f) hash 函数，这个函数是该类的核心，为公开的接口，用于接受一个字符串 x，调用 SHA_PAD 进行填充处理后，再通过循环处理每一个 512 位的块（调用 DO 函数），最终生成整个消息的哈希值。哈希值的格式化部分使用十六进制表示。 g) hextoHugeint 函数，此函数将十六进制字符串转换为 Hugeint 类型，用于处理可能非常大的数值。它从字符串的最后一个字符开始，将每个十六进制字符转换为相应的十进制值，并利用幂次累加（每次乘以 16）来计算最终的大整数值。 上述就是 SHA-1 的整个实现原理和过程的清晰描述。 （3） 封装的 ElGamal 类 在本实验中，我建立了 ElGamal 类，并把 ElGamal 密钥交换算法需要用到的一些函数，在基于支持我自己写的 Hugeint 类数据类型的基础上，如公因数计算（Gcd）、素数判断（isPrime）等封装为 ElGamal 类的成员函数，以方便其调用， 并且在上次的经验（教训）基础上，本次还增加了原根检验函数（isPrimitiveRoot），并利用该函数写了一个用于原根生成的函数（generatePrimitiveRoot），方便实验中快速生成原根。

同时，为了确保加密算法的实用性和可靠性，需要采用专门针对大数运算进行优化的实现，并通过对算法的性能进行评估和测试，确保其在实际应用中能够达到要求的加密强度和运行效率。因此，我还使用基于基于贝祖等式的公因数算法（Bezout）、模重复平方算法（DeMo2）等来优化我的计算过程。

5. 用户交互（User Interaction）

为了增强实验的互动性，提高用户的使用体验，我沿用以往几次实验的经验，在文件 Sign_Machine 中的 main 函数里加入了简单的命令行交互界面，在本实验中，用户将先后被假设成通信的一方“Alice”和另一方“Bob”，双方进行 ElGamal 签名算法的通信，为了更好的感受 ElGamal 密钥交换的过程，在本次实验中，我为每一个阶段都设置了自动的过程。其中，如果不想手动输入的话，关于公共参数的协商（选取）可以是自动的，帮助避免了用户自行选取大素数以及后续一系列复杂运算的不必要过程。

值得一提的是，经过上次老师的提醒，这次增加了对输入公共参数进行检验的过程，如果用户输入的 p 并非素数，或者 g 并非 p 的原根，用户将会被要求重新输入。而自动生成的 p 和 g 也始终满足上述条件。

而后面，用户选择私钥的过程则是全自动，并在输入完明文 M 后，后续的几秒钟内持续显示交互的结果。这不仅使得本实验关于 ElGamal 算法的实现更加实用，也为贴心地为用户提供了一个简便的方式来理解 ElGamal 密钥交换的全过程。

二、实验部分

由于在写代码时，本实验的大部分数据的选取都可以是自动生成的，因此本次实验部分则容易得多。

首先，在用户交互提示界面，我们先选择“1”进入本次实验部分。 接下来是：

1. 自动参数协商与自动密钥选取

（1）首先，你是 Alice，将要与 Bob 进行通信，选择“1”（Automatic allocation），得到协商公开参数“p”和“g”；

	（2）接下来，等待代码自动运行，得到公开参数集（A，g，p）； （3）输入明文 M=“IloveSEU_572!”，等待系统自动选取临时私钥 b 并生成对应公钥 B。 （4）视角切换，现在你是 Bob，等待代码自动运行，输出消息 M 的哈希函数值 h（M），系统计算得到左式 LHS 和右式 RHS，比较两者是否相等，完成对于签名的验证，本轮实验结束！ 2. 自选公共参数与自动密钥选取 （1） 首先，你是 Alice，将要与 Bob 进行通信，选择“2”（Choose for yourself），自行选取公开参数大素数“p”=97，和原根“g”=5。 （2） 接下来，等待代码自动运行，得到公开参数集（A，g，p） （3） 输入明文 M=“IloveSEU_572!”，等待系统自动选取临时私钥 b 并生成对应公钥 B。 （4） 视角切换，现在你是 Bob，等待代码自动运行，输出消息 M 的哈希函数值 h（M），系统计算得到左式 LHS 和右式 RHS，比较两者是否相等，完成对于签名的验证，本轮实验结束！ 3. 使用错误私钥签名测试 （1）找到代码中计算 Sig 的那行，在上面加入一行“a=666”，表示在签名 Sig 时使用的是错误的私钥“666”，而并非 Alice 的私钥（本实验中 Alice 选取的私钥 a 设定范围在 1000~2000 之间）。 （2）运行程序。现在，你是 Alice，将要与 Bob 进行通信，选择“1”（Automatic allocation），得到协商公开参数“p”和“g”； （3）接下来，等待代码自动运行，得到公开参数集（A，g，p）； （4）输入明文 M=“IloveSEU_572!”，等待系统自动选取临时私钥 b 并生成对应公钥 B。 （5）视角切换，现在你是 Bob，等待代码自动运行，输出消息 M 的哈希函数值 h（M），系统计算得到左式 LHS 和右式 RHS，比较两者发现不相等，对于签名的验证结果为签名失效，本轮实验结束！ 4. 多次实验，重复上述步骤

注意：本实验中关于 M 的哈希函数值是通过十六进制转十进制的的方式得到的，因此可以在代码中分别对其进行输出查看十进制和十六进制的哈希函数。此外，老师之前提到如果太长没法运算的话可以哈希函数值只取前 8 位。但是由于本实验中使用了自己编写的 Hugeint 类，支持长达 1024 位的运算，再加上模重复平方计算对于取模运算的时间复杂度低，完全能够支持完整长度哈希函数值的计算，为了尽可能还原本实验算法，故在实验中没有采用取前 8 位的做法，而是保留了完整长度的哈希函数值。

三、实验分析

通过本次 ElGamal 密钥交换的实验，我们对这一经典的密钥协商协议进行了深入的探索，并验证了其在现实应用中的可行性和安全性。在实验过程中，我们观察到了以下几个重要的分析点：

1. 素数选择的重要性：

在 ElGamal 密钥交换中，素数的选择直接影响着密钥的安全性。通过调节素数的大小和范围，我们发现当使用较大的素数作为公共参数时，攻击者破解密钥的难度显著增加。这是因为大素数的选取使得离散对数问题更难以解决，从而增强了算法的安全性。

2. 算法的计算复杂度：

与之前几次实验的算法不同，ElGamal 密钥交换的计算复杂度相对较低，特别是在处理大量数据时。在实验中，我们测试了不同大小的素数对密钥交换时间的影响，结果显示计算时间与素数的大小呈线性关系。这提示我们在实际应用中，可以更轻松地调整算法的参数以满足不同性能需求。

3. 算法的安全性：

通过尝试使用不正确的私钥 a 来测试算法的安全性，我们发现 Bob 计算得到的左值和右值不相等，签名失效。这表明 ElGamal 密钥交换算法具有较高的安全性，能够有效地证明通信一方的身份信息。此外，算法对于各种类型的输入数据都能正确处理，表现出了良好的适应性和稳定性。

	四、实验结论 ElGamal 签名算法作为一种重要的公钥加密和签名协议，在本实验中展示了其在确保数据完整性和验证发送者身份方面的优势。通过实际的编程实现和测试，我们不仅加深了对算法原理的理解，还验证了其在不同的使用场景下的有效性和高安全性。实验结果清晰地表明，正确选择密钥生成参数和私钥，以及精心设计算法的实现细节，是确保数字签名安全的关键因素。此外，实验还突出了 ElGamal 签名算法处理签名的相对较高的效率，为其在实际应用中的广泛部署提供了重要的实践经验和理论支持。 通过这次实验，不仅加深了我对 ElGamal 签名算法的理解，也积累了宝贵的实践经验，为未来在网络通信、数字版权保护和信息安全领域的研究和应用提供了坚实的基础！

实验结果	1. 自动参数协商与自动密钥选取
	（1）首先，你是 Alice，将要与 Bob 进行通信，选择“1”（Automatic allocation），得到协商公开参数“p”和“g”；
	（2）接下来，等待代码自动运行，得到公开参数集（A，g，p）；
	（3）输入明文 M=“IloveSEU_572!”，等待系统自动选取临时私钥 b 并生成对应公钥 B。
	（4）视角切换，现在你是 Bob，等待代码自动运行，输出消息 M 的哈希函数值 h（M），系统计算得到左式 LHS 和右式 RHS，比较两者是否相等，完成对于签名的验证，本轮实验结束！
	实验结果 1：
	`You are welcome to use the ElGamal-Signature-Algorithm of SEUer_LK123425!`
	`-Please enter the corresponding number for operation:`
	`  1.Simulate the process of ElGamal-Signature-Algorithm  2.Quit the Machine`
	`-Choice: 1`
	`-Now,you're Alice, and you'd like to send a message to Bob with your signature.`
	`#Key Generation Section:`
	`-About the two public parameters 'p' and 'g', which would you prefer?`
	`  1.Automatic allocation(It may take a few seconds)  2.Choose for yourself`
	`-You prefer: 1`
	`-The Large prime number 'p' is: 479429`
	`-The Primordial root 'g' is: 10`
	`-Now, you've got the two public parameters 'p' and 'g',`
	` and your Private-Key 'a' is automatically allocated:`
	` a = 961`
	`-Then, you've got your Public-Key: (A, g ,p) = (416190, 10, 479429)`
	`#Signature Section`
	`-You want to send your message to Bob, here is your message:`
	` IloveSEU_572!`
	`-Thus, your temporary Private-Key 'b' is automatically allocated:`
	` b = 853`
	`-And, you've got your signature of message: (Sig, B) = (251771, 274514)`

```
#Verification Section
- Now, you're Bob, and you'd received a message Alice with her signature.
-First, you compute the (hex)hash-value 'h(M)' of message
 The hash of message is: 69cff735ce595aa1ebeba933d1841ebce14fe7af
-Secondly, you compute LHS value of (A^B)*(B^Sig) and RHS value of g^h(M) mod p:
-LHS value of (A^B)*(B^Sig) = 46870
-RHS value of g^h(M) mod p = 46870

-Well done! You've achieved the ElGamal-Signature Process!
```

2. 自选公共参数与自动密钥选取

（1） 首先，你是 Alice，将要与 Bob 进行通信，选择“2”（Choose for yourself），自行选取公开参数大素数“p”=97，和原根“g”=5。

（2） 接下来，等待代码自动运行，得到公开参数集（A，g，p）

（3） 输入明文 M=“IloveSEU_572!”，等待系统自动选取临时私钥 b 并生成对应公钥 B。

（4） 视角切换，现在你是 Bob，等待代码自动运行，输出消息 M 的哈希函数值 h（M），系统计算得到左式 LHS 和右式 RHS，比较两者是否相等，完成对于签名的验证，本轮实验结束！

实验结果 2：

```
You are welcome to use the ElGamal-Signature-Algorithm of SEUer_LK123425!

-Please enter the corresponding number for operation:
  1.Simulate the process of ElGamal-Signature-Algorithm         2.Quit the Machine
-Choice: 1
-Now,you're Alice, and you'd like to send a message to Bob with your signature.

#Key Generation Section:
-About the two public parameters 'p' and 'g', which would you prefer?
  1.Automatic allocation(It may take a few seconds)     2.Choose for yourself
-You prefer: 2
-The Large prime number 'p' is: 97
-The Primordial root 'g' is: 5

-Now, you've got the two public parameters 'p' and 'g',
 and your Private-Key 'a' is automatically allocated:
 a = 227
-Then, you've got your Public-Key: (A, g ,p) = (10, 5, 97)

#Signature Section
-You want to send your message to Bob, here is your message:
 IloveSEU_572!
-Thus, your temporary Private-Key 'b' is automatically allocated:
 b = 1121
-And, you've got your signature of message: (Sig, B) = (71, 14)

#Verification Section
- Now, you're Bob, and you'd received a message Alice with her signature.
-First, you compute the (hex)hash-value 'h(M)' of message
 The hash of message is: 69cff735ce595aa1ebeba933d1841ebce14fe7af
-Secondly, you compute LHS value of (A^B)*(B^Sig) and RHS value of g^h(M) mod p:
-LHS value of (A^B)*(B^Sig) = 83
-RHS value of g^h(M) mod p = 83

-Well done! You've achieved the ElGamal-Signature Process!
```

3. 使用错误私钥签名测试

（1）找到代码中计算 Sig 的那行，在上面加入一行“a=666”，表示在签名 Sig 时使用的是错误的私钥“666”，而并非 Alice 的

私钥（本实验中 Alice 选取的私钥 a 设定范围在 1000~2000 之间）。

（2）运行程序。现在，你是 Alice，将要与 Bob 进行通信，选择“1”（Automatic allocation），得到协商公开参数“p”和“g”；

（3）接下来，等待代码自动运行，得到公开参数集（A，g，p）；

（4）输入明文 M=“IloveSEU_572!”，等待系统自动选取临时私钥 b 并生成对应公钥 B。

（5）视角切换，现在你是 Bob，等待代码自动运行，输出消息 M 的哈希函数值 h（M），系统计算得到左式 LHS 和右式 RHS，比较两者发现不相等，对于签名的验证结果为签名失效，本轮实验结束！

实验结果 3：

```
You are welcome to use the ElGamal-Signature-Algorithm of SEUer_LK123425!

-Please enter the corresponding number for operation:
  1.Simulate the process of ElGamal-Signature-Algorithm         2.Quit the Machine
-Choice: 1
-Now,you're Alice, and you'd like to send a message to Bob with your signature.

#Key Generation Section:
-About the two public parameters 'p' and 'g', which would you prefer?
  1.Automatic allocation(It may take a few seconds)     2.Choose for yourself
-You prefer: 1
-The Large prime number 'p' is: 88547
-The Primordial root 'g' is: 6
-Now, you've got the two public parameters 'p' and 'g',
 and your Private-Key 'a' is automatically allocated:
 a = 473
-Then, you've got your Public-Key: (A, g ,p) = (3484, 6, 88547)

#Signature Section
-You want to send your message to Bob, here is your message:
 IloveSEU_572!
-Thus, your temporary Private-Key 'b' is automatically allocated:
 b = 173
-And, you've got your signature of message: (Sig, B) = (31043, 49438)

#Verification Section
- Now, you're Bob, and you'd received a message Alice with her signature.
-First, you compute the (hex)hash-value 'h(M)' of message
 The hash of message is: 69cff735ce595aa1ebeba933d1841ebce14fe7af
-Secondly, you compute LHS value of (A^B)*(B^Sig) and RHS value of g^h(M) mod p:
-LHS value of (A^B)*(B^Sig) = 62123
-RHS value of g^h(M) mod p = 88511
Sorry, some error occurred!
```