武汉大学国家网络安全学院

本科生课程实验报告

基于公钥密码体系的

文件传输系统

指导教师： 彭聪

课程名称： 密码学实验

专业名称： 信息安全

团队成员： 关皓匀 学号

周业营 2022302181145

2025年 1 月 7日

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 实验地点 | 网安基地新珈楼C106 | 指导教师 | 彭聪 | |
| 一、实验目的及要求  **实验目的**   1. **掌握公钥密码学的基本原理** 理解公钥密码的核心算法及其实现方法，包括但不限于大整数运算、加密、解密、签名和验证等核心功能。 2. **提升密码工程实践能力** 深入了解密码学的底层实现，熟悉加密算法开发工具和开源库（如 GMP），并通过项目实践锻炼解决实际问题的能力。 3. **探索公钥密码的实际应用** 将公钥密码学与具体场景相结合，设计并搭建一个具备实际应用价值的系统或功能。   **实验要求**   1. **基本要求：**    * 实现一个公钥密码算法（如 RSA、ECDH、或其他基于椭圆曲线的算法）。    * 搭建一个基于公钥密码的应用，如密钥交换、数字签名、身份认证等。 2. **进阶要求：**    * **底层实现优先**：尽可能从底层（如自定义大整数运算）实现公钥密码算法。    * **利用开源库**：可以选择使用开源库（如 GMP）进行开发，使用底层实现者评分更高。    * **应用背景**：优先选择具有实际应用背景的场景设计（如文件加密、网络通信、物联网设备安全等）。   二、实验设备（环境）及要求  Ubuntu 20.04  GMP  三、实验内容与步骤  1.  2.   1. 使用CFB工作模式下的AES加密文件   **原理介绍**  CFB是 AES的一种工作模式。它是一种基于块加密的模式，但具备流加密的特点，适用于对数据流，如文件或网络数据流，进行加密和解密。CFB 模式利用前一次加密的结果作为当前块的输入，并结合明文生成密文，从而避免了直接操作明文块。根据CFB的工作原理，它具有以下的特点和优势：  (1)适用于流加密，CFB 模式可以加密不固定长度的数据流，而不仅限于固定大小的块数据。适用于流式数据的加密，如文件或网络数据包。  (2)错误传播范围有限，如果密文中的一个位出错，那么解密时，错误会影响该位以及之后的一个明文块，但不会影响更多的内容。  (3)不需要填充数据，CFB 模式支持任意长度的输入数据，不需要像 CBC 模式那样填充明文至块大小的整数倍。    在CFB 模式中，块大小可以根据具体需求调整，通常是 128 位、64 位、8位或1 位。但由于AES 的设计原生支持 128 比特块的处理，使用 128 比特块大小可以避免额外的分割或拼接操作，直接利用 AES 硬件或软件实现，充分发挥 AES 的加密性能。同时还可以减少分组加密算法中因块大小不匹配引发的额外处理，比如需要对小于 128 比特的块进行扩展或对多于 128 比特的块进行分割。这使得 CFB 模式在使用 128 比特块时运行效率更高，尤其是在使用AES-NI 指令集进行硬件加速的时候，可以直接处理整个 128 比特数据块。  除了加快加解密速率之外，使用比特块的大小也影响密码系统的安全强度。128 比特块可以提供更多的随机性，相比更小的块大小，128 比特数据块更难通过穷举攻击恢复明文。同时在较长的数据流中，如果块大小太小，重复的输入可能导致输出的密文块也出现重复的现象，从而泄露模式信息。而128 比特的块大小可以降低这种重复概率。  综合以上的考虑，我们选择使用128 位的块大小进行CFB模式的更新。  **代码实现**   1. **int** aes\_cfb\_encrypt(**const** unsigned **char** \*plaintext, **size\_t** plaintext\_len, 2. **const** unsigned **char** key[AES\_KEY\_SIZE], 3. **const** unsigned **char** iv[AES\_BLOCK\_SIZE], 4. unsigned **char** \*ciphertext) { 5. unsigned **char** feedback[AES\_BLOCK\_SIZE]; 6. unsigned **char** encryptedBlock[AES\_BLOCK\_SIZE]; 7. unsigned **char** encSubKeys[AES\_EXPANDED\_KEY\_BLOCK][AES\_BLOCK\_SIZE]; 8. **size\_t** i, j; 9. **size\_t** remaining; 11. // 生成加密子密钥 12. **if** (aes\_make\_enc\_subkeys(key, encSubKeys) != 0) { 13. **return** 1; 14. } 16. // 初始化反馈寄存器为IV 17. memcpy(feedback, iv, AES\_BLOCK\_SIZE); 19. // 分块处理数据 20. **for** (i = 0; i < plaintext\_len; i += AES\_BLOCK\_SIZE) { 21. // 加密 22. aes\_encrypt\_block(feedback, encSubKeys, encryptedBlock); 24. // 计算当前块还剩下多少字节 25. remaining = plaintext\_len - i; 26. **if** (remaining > AES\_BLOCK\_SIZE) { 27. remaining = AES\_BLOCK\_SIZE; 28. } 30. // 与明文异或生成密文 31. **for** (j = 0; j < remaining; j++) { 32. ciphertext[i + j] = plaintext[i + j] ^ encryptedBlock[j]; 33. } 35. // 更新反馈寄存器 36. **if** (remaining == AES\_BLOCK\_SIZE) { 37. memcpy(feedback, ciphertext + i, AES\_BLOCK\_SIZE); 38. } **else** { 39. // 最后一个不完整块的处理 40. memcpy(feedback, ciphertext + i, remaining); 41. } 42. } 44. **return** 0; 45. }   由于CFB的解密过程与加密过程相似，这里不再赘述。   1. 基于椭圆曲线的Diffie Helman密钥协商协议 — ECDH   **原理介绍**  基于椭圆曲线的Diffie Helman密钥协商协议允许双方在不安全的信道中建立共享密钥，实现安全通信。ECDH相较于传统的RSA公钥密码系统，能够以更小的密钥尺寸提供相同的安全性，同时还能节省计算资源与传输带宽。密钥协商的过程如下：  (1) 生成密钥对： 每一方各自生成一对密钥。密钥对包括一个私钥和一个公钥。   * 私钥是一个随机生成的大整数，用于保持秘密。 * 公钥是通过椭圆曲线的生成点 G 和私钥相乘得到：   （2）公钥交换： 通信双方在不安全的信道中交换各自的公钥 A 和 B。这些公钥可以被公开，但私钥 a 和 b 必须严格保密。  （3）共享密钥计算： 收到对方的公钥后，通信双方各自使用自己的私钥与对方的公钥进行计算，得出相同的共享密钥：   * A 使用自己的私钥 a 和 B 的公钥 B，计算 * B 使用自己的私钥 b 和 A 的公钥 A，计算   根据椭圆曲线的数学性质，aB 和 bA 的结果是相同的，因此 A 和 B 得到了相同的共享密钥 S。  具体以Alice和Bob通信为例，假设Alice 和 Bob，他们希望通过 ECDH 协议建立一个共享密钥以进行安全通信。  首先，Alice 和 Bob 各自生成一个密钥对。Alice 选择一个随机的私钥 a，并通过将 a 乘以椭圆曲线上的生成点 G 来计算出她的公钥 A=aG。同样，Bob 选择一个随机的私钥 b，并计算出他的公钥 B=bG。私钥 a 和 b 保密，而公钥 A 和 B 则可以公开。  接下来Alice 和 Bob 交换各自的公钥。Alice 收到 Bob 的公钥 B，然后使用她的私钥 a 计算 S=aB，即 S=a(bG)。由于椭圆曲线的数学性质，这个计算的结果等同于 S=b(aG)，而 Bob 则使用他的私钥 b 和 Alice 的公钥 A 计算相同的共享密钥 S=bA。最终，双方独立得出了相同的共享密钥 S，但任何窃听者由于无法获取私钥 a 或 b，无法计算出 S。  对于曲线的选择，我们选择的是**secp256r1**标准化椭圆曲线。secp256r1是一条经过NIST审查的曲线，具有良好的可靠性和兼容性，也被广泛运用在TLS、HTTPS 、JWT 和数字签名中。  对于 secp256r1，曲线的参数为：   * a = -3 * b = 0x5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B   对应的有限域是一个素数阶的有限域，其中：  是一个 256 位的大素数。256位符合国家商用密码要求的密钥长度，如果选用更小的曲线可能在安全性上不足以抵御未来的攻击，而选用更大的曲线则会消耗更多的计算和带宽资源。  **代码实现**  本次实验中对大整数的存储和计算使用了GMP开源库工具。根据椭圆曲线的要素定义出椭圆曲线的结构体。   1. // 椭圆曲线参数的结构定义 2. **typedef** **struct** { 3. mpz\_t p;  // 模数 4. mpz\_t a;  // 曲线参数a 5. mpz\_t b;  // 曲线参数b 6. ECPoint G;  // 基点 7. mpz\_t n;  // 阶 8. } ECurve;   以及椭圆曲线上点的结构体。   1. // 椭圆曲线上点的结构定义 2. **typedef** **struct** { 3. mpz\_t x; 4. mpz\_t y; 5. **int** infinity; 6. } ECPoint;   然后定义出曲线和曲线上点的初始化和释放函数。   1. **void** ec\_init\_curve(ECurve \*curve) { 2. mpz\_init(curve->p); 3. mpz\_init(curve->a); 4. mpz\_init(curve->b); 5. mpz\_init(curve->n); 6. ec\_init\_point(&curve->G); 7. } 9. **void** ec\_clear\_curve(ECurve \*curve) { 10. mpz\_clear(curve->p); 11. mpz\_clear(curve->a); 12. mpz\_clear(curve->b); 13. mpz\_clear(curve->n); 14. ec\_clear\_point(&curve->G); 15. }   接下来是实现ECDH的计算基础，椭圆曲线上点的运算。先是最基础的点加运算，在点加运算中注意区分两个相加的点是否为同一个点，如果相同则为点倍。   1. **static** **void** ec\_point\_add(ECPoint \*result, **const** ECPoint \*p1, **const** ECPoint \*p2, **const** ECurve \*curve) { 2. **if** (p1->infinity) { 3. mpz\_set(result->x, p2->x); 4. mpz\_set(result->y, p2->y); 5. result->infinity = p2->infinity; 6. **return**; 7. } 8. **if** (p2->infinity) { 9. mpz\_set(result->x, p1->x); 10. mpz\_set(result->y, p1->y); 11. result->infinity = p1->infinity; 12. **return**; 13. } 15. mpz\_t lambda, temp1, temp2, temp3; 16. mpz\_init(lambda); 17. mpz\_init(temp1); 18. mpz\_init(temp2); 19. mpz\_init(temp3); 21. // 检查是否是同一点 22. **if** (mpz\_cmp(p1->x, p2->x) == 0) { 23. **if** (mpz\_cmp(p1->y, p2->y) != 0) { 24. // P + (-P) = O 25. result->infinity = 1; 26. **goto** cleanup; 27. } 28. // 检查是否是y=0的点 29. **if** (mpz\_cmp\_ui(p1->y, 0) == 0) { 30. result->infinity = 1; 31. **goto** cleanup; 32. } 33. // 点加倍: lambda = (3x^2 + a) / (2y) 34. mpz\_mul(temp1, p1->x, p1->x); 35. mpz\_mod(temp1, temp1, curve->p); 36. mpz\_mul\_ui(temp1, temp1, 3); 37. mpz\_add(temp1, temp1, curve->a); 38. mpz\_mod(temp1, temp1, curve->p); 40. mpz\_mul\_ui(temp2, p1->y, 2); 41. **if** (!mpz\_invert(temp2, temp2, curve->p)) { 42. result->infinity = 1; 43. **goto** cleanup; 44. } 46. mpz\_mul(lambda, temp1, temp2); 47. mpz\_mod(lambda, lambda, curve->p); 48. } **else** { 49. // 点加法: lambda = (y2-y1)/(x2-x1) 50. mpz\_sub(temp1, p2->y, p1->y); 51. mpz\_mod(temp1, temp1, curve->p); 53. mpz\_sub(temp2, p2->x, p1->x); 54. mpz\_mod(temp2, temp2, curve->p); 56. **if** (!mpz\_invert(temp2, temp2, curve->p)) { 57. result->infinity = 1; 58. **goto** cleanup; 59. } 61. mpz\_mul(lambda, temp1, temp2); 62. mpz\_mod(lambda, lambda, curve->p); 63. } 65. // x3 = lambda^2 - x1 - x2 66. mpz\_mul(result->x, lambda, lambda); 67. mpz\_sub(result->x, result->x, p1->x); 68. mpz\_sub(result->x, result->x, p2->x); 69. mpz\_mod(result->x, result->x, curve->p); 71. // y3 = lambda(x1 - x3) - y1 72. mpz\_sub(temp1, p1->x, result->x); 73. mpz\_mul(temp1, lambda, temp1); 74. mpz\_sub(result->y, temp1, p1->y); 75. mpz\_mod(result->y, result->y, curve->p); 77. result->infinity = 0; 79. cleanup: 80. mpz\_clear(lambda); 81. mpz\_clear(temp1); 82. mpz\_clear(temp2); 83. mpz\_clear(temp3); 84. }   然后通过倍数的二进制展开，迭代调用点加运算来实现点乘运算。   1. **void** ec\_point\_mul(ECPoint \*result, **const** ECPoint \*p, **const** mpz\_t k, **const** ECurve \*curve) { 2. // 处理特殊情况 3. **if** (p->infinity || mpz\_cmp\_ui(k, 0) == 0) { 4. result->infinity = 1; 5. **return**; 6. } 8. ECPoint R0, R1, temp; 9. ec\_init\_point(&R0); 10. ec\_init\_point(&R1); 11. ec\_init\_point(&temp); 13. // R0 = O, R1 = P 14. R0.infinity = 1; 15. mpz\_set(R1.x, p->x); 16. mpz\_set(R1.y, p->y); 17. R1.infinity = 0; 19. // 从最高位开始处理 20. **for** (**int** i = mpz\_sizeinbase(k, 2) - 1; i >= 0; i--) { 21. **if** (mpz\_tstbit(k, i)) { 22. // 如果当前位为1: R0 = R0 + R1, R1 = 2R1 23. ec\_point\_add(&temp, &R0, &R1, curve); 24. mpz\_set(R0.x, temp.x); 25. mpz\_set(R0.y, temp.y); 26. R0.infinity = temp.infinity; 28. ec\_point\_add(&temp, &R1, &R1, curve); 29. mpz\_set(R1.x, temp.x); 30. mpz\_set(R1.y, temp.y); 31. R1.infinity = temp.infinity; 32. } **else** { 33. // 如果当前位为0: R1 = R0 + R1, R0 = 2R0 34. ec\_point\_add(&temp, &R0, &R1, curve); 35. mpz\_set(R1.x, temp.x); 36. mpz\_set(R1.y, temp.y); 37. R1.infinity = temp.infinity; 39. ec\_point\_add(&temp, &R0, &R0, curve); 40. mpz\_set(R0.x, temp.x); 41. mpz\_set(R0.y, temp.y); 42. R0.infinity = temp.infinity; 43. } 44. } 46. // 结果存放在R0中 47. mpz\_set(result->x, R0.x); 48. mpz\_set(result->y, R0.y); 49. result->infinity = R0.infinity; 51. // 释放临时变量 52. ec\_clear\_point(&R0); 53. ec\_clear\_point(&R1); 54. ec\_clear\_point(&temp); 55. }   进入到ECDH协议的实现中，先是对曲线的参数进行定义。   1. **int** ecdh\_init\_context(ECurve \*curve) { 2. **if** (!curve) **return** ECDH\_ERROR\_INVALID\_PARAM; 4. ec\_init\_curve(curve); 6. // 使用 secp256r1 参数 7. mpz\_set\_str(curve->p, "FFFFFFFF00000001000000000000000000000000FFFFFFFFFFFFFFFFFFFFFFFF", 16); 9. mpz\_t temp; 10. mpz\_init(temp); 11. mpz\_set\_si(temp, -3); 12. mpz\_mod(curve->a, temp, curve->p); 13. mpz\_clear(temp); 15. mpz\_set\_str(curve->b, "5AC635D8AA3A93E7B3EBBD55769886BC651D06B0CC53B0F63BCE3C3E27D2604B", 16); 16. mpz\_set\_str(curve->n, "FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551", 16); 17. mpz\_set\_str(curve->G.x, "6B17D1F2E12C4247F8BCE6E563A440F277037D812DEB33A0F4A13945D898C296", 16); 18. mpz\_set\_str(curve->G.y, "4FE342E2FE1A7F9B8EE7EB4A7C0F9E162BCE33576B315ECECBB6406837BF51F5", 16); 19. curve->G.infinity = 0; 21. **return** ECDH\_SUCCESS; 22. }   然后是生成密钥对的过程。对于产生私钥所需要的随机数，如果直接使用random生成的伪随机数，攻击者通过可预测的随机数生成器种子，能够对生成的随机数本身进行预测，从而破坏加密系统的安全性。为了生成高质量的随机数，我们使用了**/dev/random**，基于Linux系统中收集的**环境噪声**，包括键盘输入、鼠标移动、硬件中断等随机事件。通过这些随机事件是不可预测性，来生成具有很高的熵值的随机数。   1. // 从/dev/random读取随机字节 2. **static** **int** get\_random\_bytes(**void** \*buf, **size\_t** len) { 3. **int** fd = open("/dev/random", O\_RDONLY); 4. **if** (fd == -1) **return** -1; 6. **size\_t** read\_bytes = 0; 7. **while** (read\_bytes < len) { 8. ssize\_t result = read(fd, (**char**\*)buf + read\_bytes, len - read\_bytes); 9. **if** (result < 0) { 10. close(fd); 11. **return** -1; 12. } 13. read\_bytes += result; 14. } 16. close(fd); 17. **return** 0; 18. }   然后生成密钥对。   1. **void** generate\_keypair(mpz\_t private\_key, ECPoint \*public\_key, **const** ECurve \*curve) { 2. // 使用足够的字节来生成私钥 3. unsigned **char** random\_bytes[32];  // 256位随机数 4. **if** (get\_random\_bytes(random\_bytes, **sizeof**(random\_bytes)) < 0) { 5. fprintf(stderr, "Error: Failed to read from /dev/random\n"); 6. exit(1); 7. } 9. // 将随机字节转换为mpz\_t 10. mpz\_import(private\_key, **sizeof**(random\_bytes), 1, 1, 0, 0, random\_bytes); 12. // 确保私钥在正确范围内 (1 < private\_key < n-1) 13. mpz\_mod(private\_key, private\_key, curve->n); 14. **if** (mpz\_cmp\_ui(private\_key, 1) <= 0) { 15. mpz\_add\_ui(private\_key, private\_key, 2);  // 如果太小，加2确保大于1 16. } 18. // 计算公钥 Q = kG 19. ec\_point\_mul(public\_key, &curve->G, private\_key, curve); 20. }   由于ECPoint结构体包含GMP的大数(mpz\_t),无法直接网络传输，需要转换为标准字节格式进行传输，所以这里还需要对生成的公钥进行序列化和反序列化的转换。   1. **int** ecdh\_deserialize\_pubkey(ECurve \*curve, ECPoint \*pubkey, 2. **const** uint8\_t \*data, **size\_t** len) { 3. **if** (!curve || !pubkey || !data || len < PUBKEY\_SERIALIZED\_LEN) { 4. **return** ECDH\_ERROR\_INVALID\_PARAM; 5. } 7. **if** (data[0] != 0x04) { 8. **return** ECDH\_ERROR\_INVALID\_PARAM; 9. } 11. // 使用一致的字节序导入 12. mpz\_import(pubkey->x, 32, 1, 1, 0, 0, data + 1); 13. mpz\_import(pubkey->y, 32, 1, 1, 0, 0, data + 33); 14. pubkey->infinity = 0;  17. **return** ECDH\_SUCCESS; 18. } 20. **int** ecdh\_serialize\_pubkey(**const** ECPoint \*pubkey, 21. uint8\_t \*out, **size\_t** out\_len) { 22. **if** (!pubkey || !out || out\_len < PUBKEY\_SERIALIZED\_LEN) { 23. **return** ECDH\_ERROR\_INVALID\_PARAM; 24. } 26. out[0] = 0x04; 27. memset(out + 1, 0, 64);  // 清零x和y坐标的空间 28. **size\_t** count; 29. mpz\_export(out + 1, &count, 1, 1, 0, 0, pubkey->x); 30. mpz\_export(out + 33, &count, 1, 1, 0, 0, pubkey->y); 32. **return** ECDH\_SUCCESS; 33. }   除此之外，为了提供一种轻便化的公钥交换消息格式，在曲线选择固定的条件下，公钥的传输（kG点的X坐标与Y坐标）可以压缩至原先的一半，只传输X坐标，在接收端接受到X坐标之后再通过压缩格式和X坐标来计算出Y坐标，从而得到完整的公钥坐标。   1. **int** ecdh\_serialize\_pubkey\_compressed(**const** ECPoint \*pubkey, 2. uint8\_t \*out, **size\_t** out\_len) { 3. **if** (!pubkey || !out || out\_len < PUBKEY\_COMPRESSED\_LEN) { 4. **return** ECDH\_ERROR\_INVALID\_PARAM; 5. } 7. // 判断y坐标的奇偶性 8. mpz\_t temp; 9. mpz\_init(temp); 10. mpz\_mod\_ui(temp, pubkey->y, 2); 12. // 设置压缩格式标记 13. out[0] = mpz\_cmp\_ui(temp, 0) == 0 ? 14. PUBKEY\_FORMAT\_COMPRESSED\_EVEN : 15. PUBKEY\_FORMAT\_COMPRESSED\_ODD; 17. mpz\_clear(temp); 19. // 导出x坐标 20. **size\_t** count; 21. mpz\_export(out + 1, &count, 1, 1, 0, 0, pubkey->x); 23. **return** ECDH\_SUCCESS; 24. }   在单方面的密钥生成结束之后，还需要结合对方传输过来的公钥计算出共享密钥。   1. **void** compute\_shared\_secret(mpz\_t shared\_secret, **const** ECPoint \*others\_public, **const** mpz\_t my\_private, **const** ECurve \*curve) { 2. ECPoint shared\_point; 3. ec\_init\_point(&shared\_point); 5. // 计算共享点: shared\_point = my\_private \* others\_public 6. ec\_point\_mul(&shared\_point, others\_public, my\_private, curve); 8. // 使用x坐标作为共享密钥 9. mpz\_set(shared\_secret, shared\_point.x); 11. ec\_clear\_point(&shared\_point); 12. }   **测试与结果**  Alice和Bob通过随机数生成密钥对，生成的私钥、公钥与对应压缩之后的公钥，结合对方生成的公钥，双方经过密钥协商，得到共享密钥。     1. SHA-256   **原理介绍**  SHA-256是 SHA-2 家族的一种哈希函数，由美国国家安全局设计，并由美国国家标准与技术研究院（NIST）在 2001 年发布为标准。它是一种哈希算法，可将任意长度的数据输入转换为长度固定的 256 位散列值，被广泛应用于数据完整性校验、数字签名和区块链技术中。  **代码实现**   1. **void** sha256\_init(SHA256\_CTX \*ctx) { 2. ctx->state[0] = 0x6a09e667; 3. ctx->state[1] = 0xbb67ae85; 4. ctx->state[2] = 0x3c6ef372; 5. ctx->state[3] = 0xa54ff53a; 6. ctx->state[4] = 0x510e527f; 7. ctx->state[5] = 0x9b05688c; 8. ctx->state[6] = 0x1f83d9ab; 9. ctx->state[7] = 0x5be0cd19; 10. ctx->count = 0; 11. } 13. **void** sha256\_update(SHA256\_CTX \*ctx, **const** **void** \*data, **size\_t** len) { 14. **const** uint8\_t \*input = (**const** uint8\_t\*)data; 15. **size\_t** bufsize = ctx->count % SHA256\_BLOCK\_SIZE; 17. ctx->count += len; 19. **if** (bufsize > 0) { 20. **size\_t** need = SHA256\_BLOCK\_SIZE - bufsize; 21. **if** (len < need) { 22. memcpy(ctx->buffer + bufsize, input, len); 23. **return**; 24. } 25. memcpy(ctx->buffer + bufsize, input, need); 26. sha256\_transform(ctx, ctx->buffer); 27. input += need; 28. len -= need; 29. } 31. **while** (len >= SHA256\_BLOCK\_SIZE) { 32. sha256\_transform(ctx, input); 33. input += SHA256\_BLOCK\_SIZE; 34. len -= SHA256\_BLOCK\_SIZE; 35. } 37. **if** (len > 0) { 38. memcpy(ctx->buffer, input, len); 39. } 40. } 42. **void** sha256\_final(SHA256\_CTX \*ctx, uint8\_t \*digest) { 43. uint32\_t i; 44. uint64\_t total\_bits; 45. **size\_t** pad\_len; 46. uint8\_t padding[SHA256\_BLOCK\_SIZE]; 48. total\_bits = ctx->count \* 8; 49. pad\_len = SHA256\_BLOCK\_SIZE - ((ctx->count % SHA256\_BLOCK\_SIZE) + 8); 50. **if** (pad\_len <= 0) 51. pad\_len += SHA256\_BLOCK\_SIZE; 53. memset(padding, 0, pad\_len); 54. padding[0] = 0x80; 56. **for** (i = 0; i < 8; i++) 57. padding[pad\_len + i] = (total\_bits >> ((7 - i) \* 8)) & 0xff; 59. sha256\_update(ctx, padding, pad\_len + 8); 61. **for** (i = 0; i < 8; i++) { 62. digest[i\*4] = (ctx->state[i] >> 24) & 0xff; 63. digest[i\*4+1] = (ctx->state[i] >> 16) & 0xff; 64. digest[i\*4+2] = (ctx->state[i] >> 8) & 0xff; 65. digest[i\*4+3] = ctx->state[i] & 0xff; 66. } 67. } 69. **void** sha256\_hash(**const** **void** \*data, **size\_t** len, uint8\_t \*digest) { 70. SHA256\_CTX ctx; 71. sha256\_init(&ctx); 72. sha256\_update(&ctx, data, len); 73. sha256\_final(&ctx, digest); 74. }   **测试与结果**  对测试文件生成Hash值，与预期目标进行比对。     1. 基于椭圆曲线的数字签名算法 — ECDSA   **原理介绍**  ECDSA是基于 椭圆曲线密码学的数字签名算法，结合了椭圆曲线密码学的高效性和数字签名算法的安全性，被广泛应用于现代加密系统中。ECDSA的核心功能是通过椭圆曲线上的数学运算实现数据签名和验证，确保消息的完整性、真实性和不可抵赖性。签名和验签的流程如下：   1. 生成密钥    选择一条椭圆曲线。随机选择一个整数 ，作为私钥。计算 作为公钥，G 是椭圆曲线的基点。   1. 签名   签名的过程就是使用私钥对文件的哈希值进行的运算操作，证明签名者拥有对应的私钥。先使用SHA-256计算文件的哈希值。然后随机选择一个整数 。再计算点 ，提取R点的X坐标 。如果 r=0，则重新选择 k。最后来计算签名值。如果 s=0则重新选择 k。得到的签名值即为 。  (3) 验签  签名的验证就是确认签名确实由私钥的持有者生成的过程。先检查是否满足 。若不满足，则签名直接无效。然后计算消息的哈希值 。计算 和 和 。计算点 ，提取 P点 的 X坐标 。最后检查r与 是否相等 。若相等，则签名有效。  **代码实现**  密钥对生成的部分与ECDH中展示的相同，这里不再赘述，主要展示签名部分的代码与验签部分的代码。  首先是签名的部分，计算思路与原理介绍中叙述的一致。   1. **int** ecdsa\_sign(ECurve \*curve, **const** uint8\_t \*private\_key, **size\_t** private\_key\_len, 2. **const** uint8\_t \*hash, **size\_t** hash\_len, 3. ECDSASignature \*signature) { 4. **if** (!curve || !private\_key || !hash || !signature || 5. private\_key\_len < 32 || hash\_len < 32) { 6. **return** -1; 7. } 9. mpz\_t d, k, r, s, e, kinv; 10. ECPoint kG; 11. **int** result = -1; 13. mpz\_init(d); 14. mpz\_init(k); 15. mpz\_init(r); 16. mpz\_init(s); 17. mpz\_init(e); 18. mpz\_init(kinv); 19. ec\_init\_point(&kG); 21. // 导入私钥和消息哈希 22. mpz\_import(d, private\_key\_len, 1, 1, 0, 0, private\_key); 23. mpz\_import(e, hash\_len, 1, 1, 0, 0, hash); 25. **int** retry = 0; 26. **int** max\_retries = 10; 28. **do** { 29. // 生成随机数k 30. **if** (generate\_k(k, curve->n) < 0) { 31. result = -1; 32. **goto** cleanup; 33. } 35. // 计算 kG 36. ec\_point\_mul(&kG, &curve->G, k, curve); 38. // r = kG.x mod n 39. mpz\_mod(r, kG.x, curve->n); 40. **if** (mpz\_cmp\_ui(r, 0) == 0) { 41. retry++; 42. **continue**; 43. } 45. // k^(-1) 46. **if** (!mpz\_invert(kinv, k, curve->n)) { 47. retry++; 48. **continue**; 49. } 51. // s = k^(-1)(e + dr) mod n 52. mpz\_mul(s, d, r); 53. mpz\_add(s, s, e); 54. mpz\_mul(s, kinv, s); 55. mpz\_mod(s, s, curve->n); 57. **if** (mpz\_cmp\_ui(s, 0) == 0) { 58. retry++; 59. **continue**; 60. } 62. // 导出签名时确保32字节对齐 63. memset(signature->r, 0, 32); 64. memset(signature->s, 0, 32); 66. **size\_t** count; 67. **size\_t** rsize = (mpz\_sizeinbase(r, 2) + 7) / 8; 68. **size\_t** ssize = (mpz\_sizeinbase(s, 2) + 7) / 8; 70. mpz\_export(signature->r + (32 - rsize), &count, 1, 1, 0, 0, r); 71. mpz\_export(signature->s + (32 - ssize), &count, 1, 1, 0, 0, s); 73. result = 0; 74. **break**; 76. } **while** (retry < max\_retries); 78. cleanup: 79. mpz\_clear(d); 80. mpz\_clear(k); 81. mpz\_clear(r); 82. mpz\_clear(s); 83. mpz\_clear(e); 84. mpz\_clear(kinv); 85. ec\_clear\_point(&kG); 87. **return** result; 88. }   然后是验签部分。   1. **int** ecdsa\_verify(ECurve \*curve, **const** uint8\_t \*public\_key, **size\_t** public\_key\_len, 2. **const** uint8\_t \*hash, **size\_t** hash\_len, 3. **const** ECDSASignature \*signature) { 4. **if** (!curve || !public\_key || !hash || !signature || 5. public\_key\_len < 65 || hash\_len != 32) { 6. **return** -1; 7. } 9. ECPoint Q; 10. mpz\_t r, s, e, w, u1, u2; 11. **int** ret = -1; 13. // 初始化变量 14. ec\_init\_point(&Q); 15. mpz\_init(r); 16. mpz\_init(s); 17. mpz\_init(e); 18. mpz\_init(w); 19. mpz\_init(u1); 20. mpz\_init(u2);  23. // 导入公钥 24. **if** (ecdh\_deserialize\_pubkey(curve, &Q, public\_key, public\_key\_len) != 0) { 25. printf("Debug - Public key deserialize failed\n"); 26. **goto** cleanup; 27. } 29. // 导入签名值r,s和消息哈希e 30. mpz\_import(r, 32, 1, 1, 0, 0, signature->r); 31. mpz\_import(s, 32, 1, 1, 0, 0, signature->s); 32. mpz\_import(e, hash\_len, 1, 1, 0, 0, hash);  35. // 检查r,s是否在[1,n-1]范围内 36. **if** (mpz\_sgn(r) <= 0 || mpz\_cmp(r, curve->n) >= 0 || 37. mpz\_sgn(s) <= 0 || mpz\_cmp(s, curve->n) >= 0) { 38. printf("Debug - r or s out of range\n"); 39. **goto** cleanup; 40. } 42. // 计算 w = s^(-1) mod n 43. **if** (!mpz\_invert(w, s, curve->n)) { 44. printf("Debug - Failed to compute s inverse\n"); 45. **goto** cleanup; 46. } 48. // u1 = ew mod n 49. mpz\_mul(u1, e, w); 50. mpz\_mod(u1, u1, curve->n); 52. // u2 = rw mod n 53. mpz\_mul(u2, r, w); 54. mpz\_mod(u2, u2, curve->n);  57. // 计算 R = u1G + u2Q 58. ECPoint R1, R2, R; 59. ec\_init\_point(&R1); 60. ec\_init\_point(&R2); 61. ec\_init\_point(&R); 63. ec\_point\_mul(&R1, &curve->G, u1, curve); 64. ec\_point\_mul(&R2, &Q, u2, curve); 65. ec\_point\_add(&R, &R1, &R2, curve); 67. **if** (R.infinity) { 68. printf("Debug - Result point at infinity\n"); 69. **goto** cleanup\_points; 70. } 72. // 验证 R.x mod n == r 73. mpz\_mod(R.x, R.x, curve->n); 75. mpz\_mod(r, r, curve->n); 76. mpz\_mod(R.x, R.x, curve->n); 78. ret = (mpz\_cmp(R.x, r) == 0) ? 0 : -1; 80. cleanup\_points: 81. ec\_clear\_point(&R); 82. ec\_clear\_point(&R1); 83. ec\_clear\_point(&R2); 85. cleanup: 86. ec\_clear\_point(&Q); 87. mpz\_clear(r); 88. mpz\_clear(s); 89. mpz\_clear(e); 90. mpz\_clear(w); 91. mpz\_clear(u1); 92. mpz\_clear(u2); 94. **return** ret; 95. }   **测试与结果**  对共享密钥进行签名，在模拟通信之后，结合接收方接受到的文件进行验签。    尝试修改文件Hash值或修改签名值进行攻击测试。    四、实验结果与数据处理  五、分析与讨论  六、成员分工   |  |  |  |  | | --- | --- | --- | --- | | 姓名 | 学号 | 分工 | 贡献占比 | | 关皓匀 |  |  |  | | 周业营 | 2022302181145 |  |  | |  |  |  |  | | | | | |
| 七、教师评语  签名：  日期： | | | | 成绩 |