Diffie-Hellman 密钥交换算法实验报告

信息安全导论课程小组

2025 年 11 月 21 日

1 实验目的

1. 验证 Diffie-Hellman（D-H）密钥交换协议的实际可行性
2. 通过代码运行结果，直观理解 D-H 协议中素数、原根、私钥、公钥及共享密钥的生成逻辑
3. 验证在 100-255 范围内选择素数及原根时，D-H 协议的有效性
4. 观察多次密钥交换过程，确认协议的稳定性
5. 分析不同素数对密钥安全性的影响

2 实验原理

Diffie-Hellman 密钥交换协议通过以下步骤实现安全密钥协商：

1. 双方协商并公开素数 p 和原根 g
2. 通信方 A 生成私钥 a，计算公钥 $A = g^a \mod p$ 并发送给 B
3. 通信方 B 生成私钥 b，计算公钥 $B = g^b \mod p$ 并发送给 A
4. A 计算共享密钥：$K = B^a \mod p$
5. B 计算共享密钥：$K = A^b \mod p$
6. 双方得到相同的共享密钥 K，其数学依据为 $(g^b \mod p)^a \mod p = (g^a \mod p)^b \mod p = g^{ab} \mod p$

3 实验环境

- **实现工具：** Python 3.11
- **素数范围：** 100-255 之间
- **参与角色：**模拟通信双方 Alice 和 Bob
- **测试次数：** 145 次全面测试（29 个素数 × 5 次测试）

4 实验结果与分析

4.1 单次密钥交换详细演示

4.1.1 演示用例参数

- 素数 $p = 107$
- 原根 $g = 2$

4.1.2 密钥对生成

- Alice 的私钥：86
- Alice 的公钥：49（计算方式：$2^{86} \mod 107$）
- Bob 的私钥：30
- Bob 的公钥：34（计算方式：$2^{30} \mod 107$）

4.1.3 共享密钥计算

- Alice 计算的共享密钥：36（计算方式：$34^{86} \mod 107$）
- Bob 计算的共享密钥：36（计算方式：$49^{30} \mod 107$）
- 验证结果：双方共享密钥完全一致，交换成功

4.2 多次密钥交换测试

为验证协议稳定性，进行 3 次独立测试，结果如下：

表 1: 多次密钥交换测试结果

测试次数	素数 p	原根 g	共享密钥	密钥匹配
1	107	2	79	✓
2	101	2	88	✓
3	109	6	68	✓

4.3 全面测试结果

为验证协议在 100-255 范围内所有素数上的有效性，进行了全面测试：

4.3.1 测试范围

- 测试素数总数：29 个（100-255 范围内的所有素数）
- 每个素数测试次数：5 次
- 总测试次数：145 次

4.3.2 测试结果统计

表 2: 全面测试结果统计

统计指标	数值
测试素数总数	29 个
有效测试素数	29 个
总测试次数	145 次
成功次数	145 次
成功率	100%

4.3.3 素数列表及对应原根

表 3: 素数与原根对应表

素数 p	原根 g	测试共享密钥示例
101	2	65
103	5	7
107	2	48
109	6	16
113	3	63
127	3	117
131	2	44
137	3	133
139	2	1
149	2	3
151	6	15
157	5	64
163	2	93
167	5	117
173	2	6
179	2	146
181	2	170
191	19	23
193	5	151
197	2	16
199	3	18
211	2	60
223	3	201
227	2	92
229	6	64
233	3	97
239	7	12
241	7	143
251	6	189

4.4 典型测试用例分析

4.4.1 用例 1：小素数测试（$p = 101$）

- 原根 $g = 2$
- 测试 5 次均成功
- 共享密钥示例：65
- 验证了小素数情况下协议同样有效

4.4.2 用例 2：中等素数测试（$p = 151$）

- 原根 $g = 6$
- 私钥空间：149 种可能
- 共享密钥示例：15
- 共享密钥具有良好的随机分布特性

4.4.3 用例 3：较大素数测试（$p = 251$）

- 原根 $g = 6$
- 私钥空间：249 种可能
- 共享密钥示例：189
- 在范围内提供了最大的密钥空间

4.5 结果分析

4.5.1 协议稳定性

- 145 次测试全部成功，成功率 100%
- 证明 D-H 协议在 100-255 素数范围内具有极高的稳定性
- 不同素数、不同随机私钥组合下均能正确工作

4.5.2 密钥随机性

- 相同素数不同测试生成的共享密钥完全不同
- 共享密钥在 1 到 $p - 1$ 范围内均匀分布
- 体现了协议的密码学强度

4.5.3 原根有效性

- 29 个素数均找到了有效的原根
- 原根的存在确保了模 p 乘法循环群的生成
- 为协议的正确性提供了数学基础

5 单向函数特性验证

从实验结果可验证 D-H 协议的单向函数特性：

1. 在演示用例中，已知公开参数（$p = 107, g = 2$）和公钥（Alice 的公钥 49、Bob 的公钥 34）
2. 无法通过公开信息直接推导出私钥（86 和 30）和共享密钥（36）
3. 该特性保障了即使公开信道被监听，攻击者也难以获取核心密钥信息
4. 离散对数问题的困难性确保了协议的安全性基础

6 安全性深度分析

6.1 密钥空间分析

表 4: 密钥空间分析

素数范围	最小私钥空间	最大私钥空间	平均私钥空间
100-150	99 ($p = 101$)	148 ($p = 149$)	∼125
151-200	149 ($p = 151$)	198 ($p = 199$)	∼175
201-255	209 ($p = 211$)	249 ($p = 251$)	∼230

6.2 计算复杂度验证

通过程序运行时间分析：

- 原根查找：平均耗时较长，但只需一次计算
- 模幂运算：计算效率高，适合实时密钥交换
- 整体协议：在测试范围内可在毫秒级完成

6.3 密码学强度评估

虽然 100-255 范围的素数在实际应用中安全性不足，但验证了：

- 离散对数问题的困难性
- 单向函数的有效性
- 协议的正确性和可靠性

7 实验结论

1. 实验结果验证了 Diffie-Hellman 密钥交换协议的可行性，双方能够通过公开信道协商得到相同的共享密钥
2. 在 100-255 范围内选择的素数及原根能够有效支持 D-H 协议的实现
3. 145 次全面测试结果表明，该协议在不同参数配置下均能稳定工作，成功率达到 100%
4. 单向函数特性确保了协议的安全性，符合密钥交换的安全需求
5. 协议的计算效率较高，适合在实际安全通信系统中应用
6. 建议在实际应用中使用更大的素数（推荐 2048 位以上）以提高安全性

8 改进建议

1. 在实际应用中应使用更大位数的素数（推荐 2048 位以上）
2. 可以增加对生成素数的强素数验证
3. 可以考虑添加对中间人攻击的防护机制
4. 可以结合数字签名技术增强身份认证
5. 可以扩展支持椭圆曲线 Diffie-Hellman（ECDH）以获得更好的安全性和性能

A 核心代码片段

```python
def is_prime(self, n):
    """检查是否为素数"""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    # 检查从3到sqrt(n)的奇数
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True

def find_primitive_root(self, p):
    """找到给定素数的原根"""
    if p == 2:
        return 1

    # 分解p-1的质因数
    phi = p - 1
    prime_factors = self._prime_factors(phi)

    # 测试每个候选数
    for g in range(2, p):
        if all(pow(g, phi // factor, p) != 1 for factor in
            prime_factors):
            return g
    return None

def generate_parameters(self):
    """生成D-H参数（素数和原根）"""
    self.prime = self.generate_prime_in_range()
    self.primitive_root = self.find_primitive_root(self.prime)

    if self.primitive_root is None:
        raise ValueError(f"无法找到素数 {self.prime} 的原根")

```

```
39      return self.prime, self.primitive_root
40
41  def generate_keys(self):
42      """生成私钥和公钥"""
43      if self.prime is None or self.primitive_root is None:
44          self.generate_parameters()
45
46      # 生成私钥 (随机数)
47      self.private_key = random.randint(2, self.prime - 2)
48
49      # 计算公钥: g^private_key mod p
50      self.public_key = pow(self.primitive_root, self.private_key, self
            .prime)
51
52      return self.public_key
53
54  def compute_shared_secret(self, other_public_key):
55      """计算共享密钥"""
56      if self.private_key is None:
57          raise ValueError("请先生成密钥对")
58
59      # 计算共享密钥: other_public_key^private_key mod p
60      self.shared_secret = pow(other_public_key, self.private_key, self
            .prime)
61      return self.shared_secret
```

Listing 1: Diffie-Hellman 类核心方法

```
1   def demonstrate_dh_exchange():
2       """演示完整的D-H密钥交换过程"""
3       print("=" * 60)
4       print("Diffie-Hellman␣密钥交换算法演示")
5       print("=" * 60)
6
7       # 创建两个通信方: Alice和Bob
8       print("\n1.␣初始化双方参数...")
9       alice = DiffieHellman(100, 255)
10      bob = DiffieHellman(100, 255)
11
12      # 双方使用相同的素数p和原根g
13      print("\n2.␣生成共享参数...")
```

```
    p, g = alice.generate_parameters()
    bob.prime = p
    bob.primitive_root = g

    print(f"␣␣␣共享素数␣p␣=␣{p}")
    print(f"␣␣␣共享原根␣g␣=␣{g}")

    # 双方生成各自的密钥对
    print("\n3.␣生成各自的密钥对...")
    alice_public = alice.generate_keys()
    bob_public = bob.generate_keys()

    print(f"␣␣␣Alice的公钥:␣{alice_public}")
    print(f"␣␣␣Bob的公钥:␣{bob_public}")

    # 双方交换公钥并计算共享密钥
    print("\n4.␣交换公钥并计算共享密钥...")
    alice_secret = alice.compute_shared_secret(bob_public)
    bob_secret = bob.compute_shared_secret(alice_public)

    print(f"␣␣␣Alice计算的共享密钥:␣{alice_secret}")
    print(f"␣␣␣Bob计算的共享密钥:␣{bob_secret}")

    # 验证共享密钥是否相同
    print("\n5.␣验证共享密钥...")
    if alice_secret == bob_secret:
        print("␣␣␣␣␣共享密钥匹配!␣密钥交换成功!␣")
    else:
        print("␣␣␣␣␣共享密钥不匹配!␣")

def test_all_primes_in_range():
    """测试100-255范围内的所有素数"""
    primes_in_range = [
        101, 103, 107, 109, 113, 127, 131, 137, 139, 149,
        151, 157, 163, 167, 173, 179, 181, 191, 193, 197,
        199, 211, 223, 227, 229, 233, 239, 241, 251
    ]

    success_count = 0
    total_tests = 0
```

```
54
55      for prime in primes_in_range:
56          # 为每个素数进行5次测试
57          for test_num in range(5):
58              alice = DiffieHellman()
59              bob = DiffieHellman()
60
61              # 设置相同的素数
62              alice.prime = prime
63              bob.prime = prime
64
65              # 找到该素数的原根
66              alice.primitive_root = alice.find_primitive_root(prime)
67              bob.primitive_root = alice.primitive_root
68
69              # 生成密钥对并计算共享密钥
70              alice_public = alice.generate_keys()
71              bob_public = bob.generate_keys()
72              alice_secret = alice.compute_shared_secret(bob_public)
73              bob_secret = bob.compute_shared_secret(alice_public)
74
75              total_tests += 1
76              if alice_secret == bob_secret:
77                  success_count += 1
78
79      print(f"总测试次数:␣{total_tests}")
80      print(f"成功次数:␣{success_count}")
81      print(f"成功率:␣{(success_count␣/␣total_tests)␣*␣100:.2f}%")
```

Listing 2: 测试和演示函数