```
#include #include #include // Function to compute (base^exp) % mod using modular
exponentiation int mod_exp(int base, int exp, int mod) { int result = 1; while (exp > 0) { if (exp %
2 == 1) result = (result * base) % mod; base = (base * base) % mod; exp /= 2; } return result; }
int main() { int p, g, x, y, k, m, c1, c2, decrypted, K, K_inv; // Public key components printf("Enter
a prime number (p): "); scanf("%d", &p); printf("Enter a primitive root (g): "); scanf("%d", &g); //
Private key printf("Enter a private key (x): "); scanf("%d", &x); // Compute public key component
y = g^x mod p y = mod_exp(g, x, p); printf("Public key: (p=%d, g=%d, y=%d)\n", p, g, y); //
Encryption printf("Enter message (m) to encrypt: "); scanf("%d", &m); printf("Enter random key
(k): "); scanf("%d", &k); K = mod_exp(y, k, p); printf("Computed K value: %d\n", K); c1 =
mod_exp(g, k, p); c2 = (m * K) % p; printf("Ciphertext: (c1=%d, c2=%d)\n", c1, c2); // Decryption
K = mod_exp(c1, x, p); // Compute K = c1^x mod p printf("Computed K value during decryption:
%d\n", K); K_inv = mod_exp(K, p - 2, p); // Modular inverse using Fermat's theorem
printf("Computed K_inv value: %d\n", K_inv); decrypted = (c2 * K_inv) % p; printf("Decrypted
message: %d\n", decrypted); return 0; }
```