1. Main method

   Because the range of primes used in their code is very small (less than 250), a straightforward attack would be to try to decompose the public key n. We can easily list all prime numbers less than 250 and then try all combinations to decompose n and find the corresponding prime number pair. Once the decomposition is successful, we can calculate **ϕ**(n), and in turn can find the private key d. We randomly generated a public key and mod n to prove his weakness. The public key will change after every reboot, so we picked one example to prove the vulnerability every time.

2. Meet in middle attack
   [Is RSA padding needed for single recipient, one-time, unique random message? - Cryptography Stack Exchange]

   **Overview**

   Because the black hat part attack uses no padding for their RSA, we can use a meet in the middle attack to guess their key, which is more efficient than just guessing keys. The messages that RSA are encrypting are small (letter by letter) so we know that m is between 0 and 255 (even less if we assume that only ascii letters are being sent).

   **Steps**

   1. Intercept a ciphertext that the client is sending to the server. This will be an array of ciphertexts because they encrypted a string letter by letter.
   2. For each letter encryption c, since we know m (letter plaintext) is between 0 and 255, we can create 2 sorted lists A and B between 0-127. Note A and B are the same.
   3. For each message in A, calculate $(M^e mod N) * (A_i^{-e} mod N) = (M * A_i^{-1})^e mod N$
      Then Check if this exists in the list $B_i^e mod N$.
      If this pair exists, then M is equal to $A_i * B_j$

3. Since they are using textbook RSA, the encryption is deterministic so it is not semantically secure. This is not secure under Ind-CPA threat model
4. No use of Authentication codes. This makes it susceptible to replay attacks where someone can sit between the client and server and modify messages. So instead of the message going from Client -> Server, it would go from Client -> attacker -> Server
5. Single Client Handling Capability
   Issue:
   The server shuts down after servicing a single client, preventing concurrent client connections. This limits scalability and practicality.
   Potential Impact:

Scalability: Inability to serve multiple ATM terminals simultaneously restricts real-world application.
Efficiency: Handling one request at a time increases client wait times, impacting user experience.

6. Hardcoded Account Information
   Issue:
   Account details like account numbers and balances are hardcoded in the code, posing security and maintenance challenges.

   Potential Impact:

   Security Risk: Hardcoded account information is susceptible to internal threats or leakage through code, increasing the risk of unauthorized access.
   Maintenance Difficulty: Account information changes require code modification, leading to inconvenience and error introduction.
7. Lack of Semantic Secure PKC and Homomorphic Cipher Algorithm
   Issue:
   The system lacks semantic secure public-key encryption and homomorphic cipher algorithms, limiting data security and encryption flexibility.

   Potential Impact:

   Communication Security Limitations: Lack of semantic security may make encrypted data vulnerable to certain types of attacks, such as chosen-plaintext attacks.
   Functionality Limitations: Absence of homomorphic encryption means inability to perform calculations on encrypted data, restricting the implementation of advanced data privacy features.
8. Lack of Balance Query Functionality
   Issue:
   The system does not provide functionality to query account balances, a fundamental requirement for a banking system.

   Potential Impact:

   Incomplete Functionality: Users cannot check their account balances, reducing system usability and user satisfaction.
   Poor User Experience: Lack of basic functionality may lead to reduced trust in the entire banking system.
9. Insufficient Input Validation
   Issue:
   The system only checks that deposit amounts are non-negative, without further validation such as checking for valid numeric input.

Potential Impact:

Injection Attack Risk: Lack of proper input validation may allow attackers to execute injection attacks, such as SQL injection (if there are database operations behind the scenes) or code injection.
Data Integrity Risk: Incorrect input types may lead to program exceptions or data errors, affecting system stability and reliability.

Conclusion：
The analysis of the current ATM and Bank system reveals multiple critical vulnerabilities and inefficiencies, which severely limit its functionality, security, and scalability. The reliance on outdated encryption methods without RSA padding compromises the semantic security of the system, making it vulnerable to various cyber-attacks such as replay and chosen-plaintext attacks. Additionally, the system's inability to handle multiple client requests simultaneously due to its design to service only one client at a time, alongside hardcoded account information, significantly hampers its operational capabilities and poses serious security risks.