Lab 9:
Encrypting and Decrypting with RSA

In this lab, you will create and brute-force attack 16-bit RSA encryption.

Before you can play out these scenarios, you will need code to create and use a public & private key.

Once your code is written, assign the roles of Alice, Bob, and Trudy to a person on your team.

Procedure

1. ***Download*** rsa.py
2. ***Put*** your names at the top of the file.
3. Create a ***design*** for the methods `create_keys` and `apply_key` in rsa.py. See the documentation for these methods in the rsa.py template. One of these methods requires significantly more work than the others. Complete your design as a team, then divide up the work for the most challenging method among the members of the team.
4. ***Fill*** out the design for the methods in part 3.
5. As a team, create a ***design*** for the method `break_key`.
6. As a team, ***implement*** `break_key`.
7. **Bob**: ***Run*** the program and create a public/private key pair. Deliver the public key to Alice. (You can reuse the key from Step 5 if you like.)
8. **Alice**: ***Create*** a secret message. ***Encrypt*** it with Bob's public key using the `encrypt_message` option of the program. ***Supply*** Bob and Trudy with the encrypted message. (You may need to email the hexadecimal characters to Bob and Trudy – or share them on IM.)
9. **Bob**: Run the program with the `decrypt_message` option to read Alice's secret message using your private key.
10. **Trudy**: ***Run*** the program with the `break_key` option to read Alice's secret message using only the public key.
11. **Whole team**: In the comments at the end of the lab, ***answer*** the questions and ***comment*** about what you learned in the lab. Your comments should include:
    a. Answers to the questions included in the comments at the end of the template.
    b. A description of the functionality you implemented and the results of your testing.
    c. Comments on your experience in completing the lab, including any problems you encountered. Briefly explain what you learned.
    d. Any questions you have about the lab (optional)
    e. Comments on the lab and suggestions for improvement.

If you have time

In this bonus exercise, we will pretend that we are using enough bits so that break_key is ineffective. Nevertheless, because we use a non-cryptographic hash, Alice can forge a message to look like the one Bob signed with his public key.

1. **Bob:** ***Run*** the program with the `compute_checksum` option to create an encrypted checksum for the message "Bob owes Trudy $100.99". ***Save*** the public & private keys, as well as the encrypted checksum for your records. ***Provide*** Alice and Trudy with the public key. ***Provide*** Trudy with the plain-text message and the encrypted checksum. (Suppose that Trudy is an unscrupulous online store…)
2. **Trudy**: ***Create*** a message that results in the same checksum as Bob's message, but implies that Bob owes a larger amount of money. Hint: If you rearrange the characters in the string, how does that change the checksum? ***Supply*** Alice with the forged message and the encrypted checksum that Bob gave you.
3. **Alice**: ***Check*** Trudy's message using the `verify_checksum` option of the program. Does it check out OK? If not, Trudy should keep trying.
4. **As a team**: Explain in your final comments how Trudy can be prevented from performing this trick in a real application. (Suppose Alice is the bank responsible for transferring the money from Bob to Trudy…)