Module 10.7: Exploring the RSA Cipher

Gregory V. Bard

February 6, 2020

- This is a practice workbook for the RSA cipher, but it assumes that you have already read and mastered the previous module.
- There is a with-answers version, and a without-answers version.
- In the with-answers version of this workbook, the black ink represents the question, and the blue ink represents the answer. The red ink is additional information that would normally only be calculated with technology (e.g. the computer-algebra system called Sage, or perhaps Maple).

Problem 10-7-1

A friend of yours is generating an RSA public-private key pair. He has followed all the steps of the key-generation process, but has forgotten what to do with each of the six things he has computed.

Of the six items computed during RSA key generation, p, q, N, φ, e, d, label each one as "public," "private," or "destroyed."

- When finished, d should be: private
- When finished, N should be: public
- When finished, q should be: destroyed
- When finished, φ should be: destroyed
- When finished, e should be: public
- When finished, p should be: destroyed

Problem 10-7-2

There is a distrustful user, Vincent. He doesn't fully understand the RSA protocol and wants to verify why each particular specification is important. Let's help Vincent verify these things, one at a time.

First, Vincent doesn't understand why φ needs to be destroyed. Suppose Vincent's φ becomes public, but nothing else (besides Vincent's public key) becomes public. (Let's also assume that no one publishes a new algorithm to make it easy to factor huge numbers.) What would happen then?

We know that e is part of the public key. If φ becomes public also, then one can compute e^{-1} mod φ. Clearly, the number obtained would be d.

Anyone could do this with Sage, by typing `pow( e, -1, phi )`. The answer returned by Sage is the inverse of e mod φ.

Whoever does this computation can now see Vincent's d, which means that they can read all his messages. While we haven't discussed digital signatures yet, they can also sign documents, such as checks, using Vincent's d as if he were Vincent.

Problem 10-7-3

Continuing with the previous problem, Vincent now understands that φ should be destroyed. We learn that RSA is very dependent on the hardness of factoring. Vincent wants to know why it would be so bad if factoring huge numbers became feasible. In particular, what would happen if nothing other than Vincent's public key was public, but factoring huge numbers became easy because of some newly published algorithm?

Again, we know that e and N comprise the public key. If factoring became easy, someone could compute p and q by factoring $N = pq$, because N is public. Then since the attacker knows p and q, it is extremely easy to compute $\varphi = (p-1)(q-1)$ as a second step.

Once the attacker knows φ, they can do what we wrote in the answer to the previous problem. Namely, as a third step, they would compute e^{-1} mod φ, since e is public. The output would be the inverse of e mod φ, and that is Vincent's d. Since the attacker can now see Vincent's d, the attacker can read all of Vincent's messages and sign important documents, such as checks, as if he were Vincent.

Problem 10-7-4

Suppose that we know that the ambassador to Kuwait has the following RSA public key:

$$N = 34,189 \text{ and } e \equiv 6393$$

Now suppose that the Russian government builds a quantum computer, and factors

$$N = 34,189 = (191)(179)$$

Tell me how this makes it easy to find the ambassador's private key.

Since we know $N = pq = (191)(179)$ we can easily compute

$$\varphi = (p-1)(q-1) = (190)(178) = 33,820$$

Next, the ambassador's d can easily be found by computing the inverse of e mod φ. Specifically, $d \equiv 6393^{-1}$ mod $33,820$ for this problem. That can be done by Sage, or it can even be done by hand by using the Extended Euclidean Algorithm. (As you can see, we're simply carrying out the procedure of the previous problem, but now using specific numbers.)

Typing `pow( 6393, -1, 33_820 )` into Sage results in 9517. Because we are using tiny numbers, we can verify this with a hand calculator:

$$(9517)(6393) = 1 + 60,842,180 = 1 + 1799(33,820) \equiv 1 \bmod 33,820$$

Because $(9517)(6393) \equiv 1$ mod φ, we know that $d \equiv 9517$ is the inverse of $e \equiv 6393$.

Problem 10-7-5

We return to Vincent, who doesn't trust all the details of the RSA algorithm's specification. Let's assume that no one publishes an algorithm to make it easier to factor huge numbers. Suppose Vincent's q becomes public, but nothing else (besides Vincent's public key) becomes public. What could happen then?

Given q, simple ordinary division would reveal p. Because $N = pq$, the attacker just needs to first compute $N/q = p$. Recall, N is public after all. Also, note that this is ordinary division, in the sense of the ordinary integers, not modular arithmetic. Second, since the attacker now knows p and q, it is extremely easy to compute $\varphi = (p-1)(q-1)$.

Now that the attacker knows φ, they can do what we wrote in the answer to the previous problem. Namely, as a third step, they would compute e^{-1} mod φ, since e is public. The output would be Vincent's d. As before, since the attacker can now see Vincent's d, the attacker can read all of Vincent's messages and sign important documents, such as checks, using Vincent's d.

Problem 10-7-6

Background Information

Whenever you use your web browser in `https` mode, instead of `http` mode, and you therefore see that key icon or lock icon that indicates an encrypted communication, then you're using either RSA, or a related algorithm called Diffie-Hellman, for a step that is called "establishing a shared secret" or "key exchange." The mechanism for doing this is a protocol called SSL (the secure sockets layer) and its successor, TLS (transport layer security).

In 2012, some researchers (see bibliographic references below) took a bunch of RSA public keys (11,400,000 of them) from the Electronic Frontier Foundation's SSL Observatory, and started taking `gcd`s of pairs of Ns. It turns out that this exceptionally simple process—downloading lots of public keys, picking random pairs of Ns, and taking gcds—factored 0.2% of the Ns. That's a lot larger than it sounds like. There were 26,965 public keys with N that were thusly factored, which is alarming since most programmers who use cryptography think that RSA is unbreakable.

Citations:

A. Lenstra, J. Hughes, M. Augier, J. Bos, T. Kleinjung, and C. Wachte. "Ron was wrong, Whit is right." Posted to `eprint.iacr.org`, 17 February, 2012.

`https://eprint.iacr.org/2012/064.pdf`

There is also a NY Times article about this: J. Markoff. "Flaw Found in an Online Encryption Method." *The New York Times.* February 15th, 2012.

`http://www.nytimes.com/2012/02/15/technology/researchers-find-flaw-in-an-online-encryption-method.html`

Let's see if we can do it ourselves...

Suppose $e_1 \equiv 151,153$ and $e_2 \equiv 176,303$. Further suppose $N_1 = 302,303$ and $N_2 = 352,603$. First, you compute $\gcd(N_1, N_2) = 503$ using the Extended Euclidean Algorithm, or Sage.

In Sage, the gcd is computed by `gcd( 302303, 352603 )` or `gcd( 302_303, 352_603 )` for clarity.

- What is the factorization of N_1? Because $N_1/g = 302,303/503 = 601$, we know $(503)(601) = 302,303$.
- What is φ_1? We compute $(502)(600) = 301,200$.
- How can we obtain the private key, d_1? Compute the inverse of 151,153 mod 301,200, namely $151,153^{-1}$ mod 301,200.

  According to Sage, $d \equiv 817$, using `pow( 151153, -1, 301200)`.
- What is the factorization of N_2? Because $N_2/g = 352,603/503 = 701$, we know $(503)(701) = 352,603$.
- What is φ_2? We compute $(502)(700) = 351,400$.
- How can we obtain the private key, d_2? Compute the inverse of 176,303 mod 351,400, namely $176,303^{-1}$ mod 351,400.

  According to Sage, $d \equiv 24,767$, using `pow( 176303, -1, 351400)`.

- What is the impact of d_1 and d_2 becoming known?

  Now that the attacker has both private keys, he can read the email of both of those users, and sign important documents, such as checks, as if he were either of them!

Problem 10-7-7

Flip back to "the short proof of correctness of RSA," which was given in the previous module, between Problems 10-6-11 and 10-6-12. Carefully examine the proof. What makes it incorrect?

The Corollary to Euler's Totient Theorem, which I called the upstairs-downstairs principle, formally states

> For any x coprime to N, if $x \equiv x' \bmod N$ and $y \equiv y' \bmod \Phi(N)$, then $x^y \equiv (x')^{(y')} \bmod \Phi(N)$.

Since we have no reason to expect m' to be coprime to N, we do not have the right to apply this corollary. We have not satisfied all the antecedents (requirements) of the theorem. As it turns out, RSA will work correctly even if m' and N are not coprime. Thus, it is the proof that is flawed, not RSA itself. The "longer proof of correctness of RSA" will remedy the situation by taking a different road to proving this theorem.

Problem 10-7-8

We return to Vincent, who doesn't trust all the details of the RSA algorithm's specification. Suppose Vincent's p becomes public, but nothing else (besides Vincent's public key) becomes public. Let's further suppose that no one publishes an algorithm to make it easier to factor huge numbers. What could happen then?

This is clearly the same problem as a previous one (Problem 10-7-5 from Page 3 of this module), where q and the public key were known. Given p, simple division would reveal q. Because $N = pq$, the attacker just needs to compute $N/p = q$. Then since the attacker knows p and q, it is extremely easy to compute $\varphi = (p-1)(q-1)$. Once the attacker knows φ, they can compute the inverse of e mod φ using Sage. That inverse is d. Once the attacker can see Vincent's d, the attacker can read all of Vincent's messages. He can also use Vincent's d to sign important documents, such as checks, as if he were Vincent.

Problem 10-7-9

We return to Vincent, who doesn't trust all the details of the RSA algorithm's specification. When we require that $2 < e < \varphi - 2$, we are (of course) explicitly ruling out $e \equiv 1$. What would go wrong if $e \equiv 1$?

When you encrypt a message, the ciphertext is $c \equiv m^e \bmod N$. We would have

$$c \equiv m^e \equiv m^1 \equiv m \bmod N$$

Your messages would not be encrypted at all—you would be transmitting your messages in plaintext.

Problem 10-7-10

We continue with Vincent, who doesn't trust all the details of the RSA algorithm's specification. When we require that $2 < e < \varphi - 2$, we are (of course) explicitly ruling out $e \equiv 0$. What would go wrong if $e \equiv 0$?

When you encrypt a message, the ciphertext is $c \equiv m^0 \bmod N$. We would have

$$c \equiv m^e \equiv m^0 \equiv 1 \bmod N$$

As you can see, the ciphertext would be 1 regardless of the message. The receiver would have no idea which message you had wanted to send, because all messages turn into a 1. This is like the sender putting a letter into an envelope, and burning it, rather than mailing it. While no one will be able to read the original message, the intended receiver will have no idea what the message was.

Problem 10-7-11

We are still talking with Vincent, who doesn't trust all the details of the RSA algorithm's specification. When we require that $2 < e < \varphi - 2$, we are (of course) explicitly ruling out $e \equiv \varphi$. What would go wrong if $e \equiv \varphi$?

When you encrypt a message, the ciphertext is $c \equiv m^e \bmod N$. We would have

$$c \equiv m^e \equiv m^\varphi \equiv m^0 \equiv 1 \text{ mod N}$$

That's because exponents, when working mod N, should be reduced mod φ, and $\varphi \equiv 0 \bmod \varphi$. The ciphertext would be 1 regardless of the message, like the previous problem. Since all possible plaintext messages encrypt to 1, the receiver would have no idea what the sender had wanted to say.

Problem 10-7-12

We have yet another inquiry from Vincent, who doesn't trust all the details of the RSA algorithm specification. When we require that $2 < e < \varphi - 2$, we are (of course) explicitly ruling out $e \equiv 2$. What would go wrong if $e \equiv 2$?

Since p and q are huge primes, they are odd. This means that $(p-1)$ and $(q-1)$ are even. The product of two even numbers is even. Since $\varphi = (p-1)(q-1)$ is even, and $e \equiv 2$ is even, then $\gcd(e, \varphi) = \gcd(2, \varphi) = 2$. This means that $\gcd(e, \varphi) \neq 1$ and thus e is not coprime to φ. That means e is not invertible mod φ, and therefore d will not exist. The receiver cannot decrypt the message, because d does not exist.

Problem 10-7-13

We have still another inquiry from Vincent, who doesn't trust all the details of the RSA algorithm specification. When we require that $2 < e < \varphi - 2$, we are (of course) explicitly ruling out $e \equiv \varphi - 1$. What would go wrong if $e \equiv \varphi - 1$?

Let's suppose that an adversary intercepts c and knows the public key, e and N. Further suppose that the adversary encrypts the ciphertext again, doubly encrypting it. That only requires information that the adversary already has. We have

$$c^e \equiv (m^e)^e \equiv (m^{\varphi-1})^{\varphi-1} \equiv m^{(\varphi-1)(\varphi-1)} \equiv m^{(\varphi^2-2\varphi+1)} \equiv m^{(0-0+1)} \equiv m^1 \equiv m$$

because

$$\varphi^2 - 2\varphi + 1 \equiv 0 - 0 + 1 \equiv 1 \bmod \varphi$$

recalling that exponents mod N work mod φ.

All that was possible because $e^2 \equiv 1 \bmod \varphi$. In the Theory of Groups, any g such that $g^2 \equiv 1$ but $g \not\equiv 1$ is called "an element of order two," or sometimes "an involution." This is the same meaning of "the order of g" that we learned about in Module 10-5: Euler's Totient Function and Modular Exponentiation, right before Problem 10-5-10. Since $g \not\equiv 1$ but $g^2 \equiv 1$, the smallest positive integer k such that $g^k \equiv 1$ is $k = 2$.

Any e such that $e^2 \equiv 1 \bmod \varphi$ is a disaster for RSA, because anyone who intercepts a ciphertext can just encrypt it again using the public key, and they will have the original plaintext.

Problem 10-7-14

Let's return to Alice and Bob at the beginning of their romantic relationship. This was discussed in Problem 10-6-9 to Problem 10-6-11 of the previous module. Suppose Bob had chosen $e \equiv 32,831$ instead of $e \equiv 19$, but his N had remained unchanged, namely 4,056,187.

For this problem, you'll definitely want to carryout the computations in Sage to get final numerical answers.

- Suppose that Bob wants to pre-encrypt some times for dates with Alice, in case he might need to respond to Alice while he's away from his laptop. While Sage will work on a smartphone, it can be a bit clumsy, so he's going to encrypt 1830, 1900, and 1930, to represent 6:30 PM, 7:00 PM, and 7:30 PM. He'll keep the resulting ciphertexts handy for use when needed. What are those ciphertexts?

We must compute $1830^{32{,}831}$ mod 4,056,187.

We type `pow( 1830, 32_831, 4_056_187 )` into Sage, and obtain that the answer is 725,790.

We must compute $1900^{32{,}831}$ mod 4,056,187.

We type `pow( 1900, 32_831, 4_056_187 )` into Sage, and obtain that the answer is 1,068,586.

We must compute $1930^{32{,}831}$ mod 4,056,187.

We type `pow( 1930, 32_831, 4_056_187 )` into Sage, and obtain that the answer is 2,523,288.

- Now suppose that he decides to encrypt those ciphertexts again, effectively "double encrypting" them. What are the new ciphertexts?

  We must compute $(725,790)^{32{,}831}$ mod 4,056,187.

  We type `pow( 725_790, 32_831, 4_056_187 )` into Sage, and obtain that the answer is 1830.

  We must compute $(1,068,586)^{32{,}831}$ mod 4,056,187.

  We type `pow( 1_068_586, 32_831, 4_056_187 )` into Sage, and obtain that the answer is 1900.

  We must compute $(2,523,288)^{32{,}831}$ mod 4,056,187.

  We type `pow( 2_523_288, 32_831, 4_056_187 )` into Sage, and obtain that the answer is 1930.

- What is e^2 mod φ? To save you from flipping back to the other module, permit me to share that Bob's $\varphi = 4,052,160$.

  We must compute $(32,831)^2$ mod 4,052,160.

  We type `pow( 32_831, 2, 4_052_160 )` into Sage, and obtain that the answer is 1. As we learned in the previous problem, we can say that e is "an element of order two." An old-fashioned term for that is "an involution."

- What can you tell me about Bob's d?

  We type `pow( 32_831, -1, 4_052_160 )` into Sage, and obtain that the answer is 32,831. How unusual! We see that $e \equiv d$ in this case. In the next subproblem, we will see why that has happened.

- Stepping away from this particular problem, if a set of RSA credentials has been properly generated by carefully following the key-generation procedure, but $e^2 \equiv 1$ mod φ, then what can you tell me about d?

  Let's take $e^2 \equiv 1$, and multiply both sides by e^{-1}. Note that we know e is invertible, because we were told that this set of RSA credentials has been properly generated by carefully following the key-generation procedure, and that procedure guarantees that e is coprime to φ. We obtain $e^2(e^{-1}) \equiv 1(e^{-1})$ but of course

$$e^2(e^{-1}) = e^{2+(-1)} = e^1 = e$$

and $1(e^{-1}) = e^{-1}$. Therefore, we know that $e \equiv e^{-1}$. Since $d \equiv e^{-1}$, this means that $e \equiv d$.

Here is the Sage code that I used, in case you are curious about how I found 32,831.

```
for k in range(1, 1000):
    temp = 1 + 4_052_160*k
    if (sqrt(temp) in ZZ):
        print(temp, "= 1 + 4,056,052 *", k, "=", sqrt(temp), "^ 2.")
```

That code produced the output:

```
1077874561 = 1 + 4,056,052 * 266 = 32831 ^ 2.
3351136321 = 1 + 4,056,052 * 827 = 57889 ^ 2.
```

This means that I could also have chosen $e \equiv 57,889 \equiv d$ instead of $e \equiv 32,831 \equiv d$.

The following code will encrypt (and encrypt again) every possible time from 1800 to 1858.

```
for time in range(1800, 1859):
    ciphertext1 = pow( time, 32_831, 4_056_187 )
    ciphertext2 = pow( ciphertext1, 32_831, 4_056_187 )

    print(time, "--->", ciphertext1, "--->", ciphertext2)
```

Problem 10-7-15

We continue with Vincent, who doesn't trust all the details of the algorithm specification. When we require that $2 < e < \varphi - 2$, we are (of course) explicitly ruling out $e \equiv \varphi - 2$. What would go wrong if $e \equiv \varphi - 2$?

Since p and q are large primes, they are odd. This means that $p - 1$ and $q - 1$ are even, so their product, $\varphi = (p-1)(q-1)$, must also be even. Since 2 is also even, this means that $\varphi - 2$, the difference between two even numbers, must also be even. Since $\varphi - 2$ and φ are both even, they are both divisible by two. This means that they are not coprime. Thus, $e \equiv \varphi - 2$ would not have an inverse mod φ. In other words, if you choose $e \equiv \varphi - 2$, then your d will not exist.

Problem 10-7-16

This problem was inspired by Problem 6-8-8 in *Cryptography with Coding Theory*, 2nd edition, written by my former dissertation supervisor, Prof. Lawrence Washington, and his student Prof. Wade Trappe.

Let's now consider the possibility that Porter has some highly confidential contacts, Boris and Natasha. To protect their highly confidential reports, he generates two public keys, with the same N but with two different encryption exponents, e_1 and e_2. Of course, he also generates the two matching private keys d_1 and d_2. When either Boris or Natasha wants to send him something, they are instructed to first encrypt with e_1, and then encrypt that ciphertext again with e_2.

- What must Porter do when receiving a message that is encrypted in this way? Boris thinks that Porter must first decrypt with d_1 and then decrypt with d_2. However, Natasha thinks that Porter must first decrypt with d_2 and then decrypt with d_1. Who is correct? Or are both contacts wrong?

  Actually, the order doesn't matter. Basically, this comes down to the fact that $d_1 d_2 = d_2 d_1$. We can also be much more precise.

  Let m be the plaintext message. The first ciphertext is $c_1 \equiv m^{e_1} \bmod N$. Then the doubly encrypted ciphertext is $c_2 \equiv c_1^{e_2} \bmod N$. Suppose Porter decrypts using d_2 first and d_1 second. Observe,

  $$\left(c_2^{d_2}\right)^{d_1} \equiv \left((c_1^{e_2})^{d_2}\right)^{d_1} \equiv \left(c_1^{e_2 d_2}\right)^{d_1} \equiv \left(c_1^1\right)^{d_1} \equiv c_1^{d_1} \equiv (m^{e_1})^{d_1} \equiv m^{e_1 d_1} \equiv m^1 \equiv m$$

  because $e_2 d_2 \equiv 1 \bmod \varphi$ and $e_1 d_1 \equiv 1 \bmod \varphi$. Those two congruence equations hold because e_2 and d_2 are inverses mod φ, and because e_1 and d_1 are inverses mod φ.

  Alternatively, suppose Porter decrypts using d_1 first and d_2 second. We obtain

  $$\left(c_2^{d_1}\right)^{d_2} \equiv \left((c_1^{e_2})^{d_1}\right)^{d_2} \equiv c_1^{e_2 d_1 d_2} \equiv \left(c_1^{e_2 d_2}\right)^{d_1} \equiv \left(c_1^1\right)^{d_1} \equiv c_1^{d_1} \equiv (m^{e_1})^{d_1} \equiv m^{e_1 d_1} \equiv m^1 \equiv m$$

- As it turns out, this two-stage encryption process is identical to encrypting once with a different public key. What is that public key? What is the corresponding private key?

  Consider $e_3 \equiv e_1 e_2 \equiv e_2 e_1 \bmod \varphi$ and $d_3 \equiv d_2 d_1 \equiv d_1 d_2 \bmod \varphi$. First, we should see that encryption with e_3 will always produce the same ciphertext as encrypting first with e_1 and then with e_2. Let $c_3 \equiv m^{e_3}$, then

  $$c_2 \equiv (c_1)^{e_2} \equiv (m^{e_1})^{e_2} \equiv m^{e_1 e_2} \equiv m^{e_3} \equiv c_3$$

  so we know that c_3 and c_2, the final ciphertexts in each case, will always be identical.

To show that d_3 is the correct private key for e_3 merely requires showing that $e_3 d_3 \equiv 1 \bmod \varphi$. This is also very easy

$$e_3 d_3 \equiv (e_1 e_2)(d_2 d_1) \equiv e_1(e_2 d_2) d_1 \equiv e_1(1) d_1 \equiv e_1 d_1 \equiv 1 \bmod \varphi$$

since $e_2 d_2 \equiv 1 \bmod \varphi$ and $e_1 d_1 \equiv 1 \bmod \varphi$.

The Real Lesson of the Previous Problem:

Tell me, what is the real lesson of the problem that we just finished?

Since encrypting once with e_1 and then again with e_2 produces identical ciphertexts to encrypting once with e_3, for all messages, it is clear that this double-encryption algorithm is no more secure than encrypting once. More precisely, there does not exist an adversary capable of breaking single-encryption with RSA and this particular size of N who cannot break the double-encryption as described here.

However, note that our argument in the previous problem worked because both public keys used the same modulus N. It is a much more advanced problem to address what happens if the Ns and φs are different.

An extremely minor point is that with the affine cipher, as we saw in Problem 10-4-1 of Module 10-4: Some Historical Ciphers, the order of the decryptions mattered—we had to follow the notion of the socks-shoes theorem. However, for the shift cipher (Problem 10-4-10), the Vigenère cipher, and now the RSA cipher, the order does not matter. Lastly, encrypting twice with the Atbash cipher would be equivalent to not encrypting at all, and transmitting the plaintext. To me, this is a minor point because the affine cipher, the shift cipher, and the Vigenère cipher have not been used in over a century (or so I hope) and the Atbash cipher has not been used for two thousand years.

Problem 10-7-17

Now Vincent is getting bolder, and wants to question why RSA uses a product of two prime numbers. What if there were only one prime number? In particular, suppose that $N = 54,323$ and $e \equiv 2003$, both of which are prime numbers. What goes wrong in this case?

For any prime number p, we know that $\Phi(p) = \varphi = p - 1$. In Vincent's specific example, $N = 54,323$ and therefore $\varphi = 54,323 - 1 = 54,322$. Since we know φ and since e is public, we can compute d by inverting $e \bmod \varphi$. For Vincent's specific example, if we want Vincent's d, we can easily compute the inverse of 2003 mod 54,322.

Sage tells us that this is 10,143, using `pow(2003, -1, 54322)` or `pow(2003, -1, 54_322)` which is equivalent.

In general, if N is prime then the "private" key d would always be computable by any person holding the public key and with access to a computer algebra system, such as Sage. Any such person would be able to read the email of any other person whose public key is known. In other words, everyone's d would become public.

Problem 10-7-18

Let's say that an investment bank gets an international call from a wealthy customer, Rich, who is on vacation overseas. The person on the phone is claiming that he has lost his wallet, and with that, his credit cards and his cash. He wants to withdraw a few thousand dollars, and has wiring instructions so that the bank can send the money directly to him overseas. Surely the bank needs to know that it is really Rich that they are speaking to, and not some criminal who is impersonating Rich.

Suppose that Rich has the following credentials:

$$\begin{aligned} d &= 192,191 \\ \varphi &= 461,032 \\ e &= 179,607 \\ N &= 462,391 \\ p &= 683 \\ q &= 677 \end{aligned}$$

Further suppose that the bank already has Rich's public key on file. The bank is going to ask Rich two questions.

- The bank asks Rich, "What number, encrypted with your public key, will yield 50,669?" What does Rich have to do, in order to compute that? Your answer to this subproblem, and the three after it, should be a formula with numbers and operations, but no variables.

  Rich must compute $(50,669)^{192,191} \bmod 462,391$.

  Typing `pow(50_669, 192_191, 462_391)` into Sage, Rich obtains 399,842.

- The bank asks Rich, "What number, encrypted with your public key, will yield 46,375?

  Rich must compute $(46,375)^{192,191} \bmod 462,391$.

  Typing `pow(46_375, 192_191, 462_391)` into Sage, Rich obtains 214,689.

- How will they check Rich's answer to the first question? What will they compute?

  The bank must compute $(399,842)^{179,607} \bmod 462,391$. Note, they can actually do this because the only numbers that they need are 399,842, which was provided to them by Rich just now, as well as 179,607 and 462,391, which comprise Rich's public key.

  Typing `pow(399_842, 179_607, 462_391)` into Sage, the bank obtains 50,669, as desired.

- How will they check Rich's answer to the second question? What will they compute?

  The bank must compute $(214,689)^{179,607} \bmod 462,391$.

  Typing `pow(214_689, 179_607, 462_391)` into Sage, the bank obtains 46,375, as desired.

- If both answers are correct, what does this prove?

  It proves that with very high probability, the person they are speaking to on the phone has access to Rich's d. That's because Rich's d is needed to find those numbers. Of course, we're assuming that Rich has not lost or misplaced his d.

- Suppose it really was a criminal on the phone and not Rich. Suppose that when the bank posed the questions to the criminal on the phone, that the criminal merely guessed. What is the probability that the criminal would have guessed correctly both times, if the criminal responded with random numbers strictly between 1 and N?

  There are $N-2$ integers z such that $1 < z < N$. Getting the first answer correct would occur with probability $1/(N-2)$, and likewise for the second answer. Getting both correct would occur with probability

$$\left(\frac{1}{N-2}\right)^2 = \left(\frac{1}{462,391-2}\right)^2 = \left(\frac{1}{462,389}\right)^2 = (0.00000216268\cdots)^2 = 4.67718\cdots\times 10^{-12}$$

  which is more easily understood as 1 in 213,803,587,321.

- Suppose the bank had instead asked the person on the phone to provide their four-digit PIN (personal identification number), and the person was a criminal impersonating Rich. If the criminal does not know Rich's PIN, and has to guess, then what is the probability of success? You can assume that the PIN is a 4-digit number, possibly starting with one or several zeroes.

  We have 1 valid PIN among 10^4 possibilities, so the probability is

$$\frac{1}{10^4} = 0.0001$$

  which is more easily understood as 1 in 10,000. That's rather different than the probability in the previous bullet.

- Using RSA in this way, as a method of authentication, is called "the RSA-challenge-response method." It is an alternative method of authentication compared to using passwords. Suppose a less technically proficient bank were to give each customer a password to memorize. In situations like this, a customer would call on the phone, and dictate the password. What advantages does the challenge-response method have over the password method?

  There are many advantages. Here are two:

  1. The bank employee who is receiving the customer's phone call would learn the customer's password. In contrast, when using the challenge-response method of authentication with RSA, the bank employees never learn anything that isn't already public.

2. An eavesdropper on the phone call, if using the password method, would obtain the customer's password. In contrast, an eavesdropper would learn nothing useful with the challenge-response method of authentication and RSA. If the eavesdropper on Rich's phone call were to try to impersonate Rich later, the bank would challenge him with different random numbers, not 50,669 and 46,375, so the eavesdropper would not know what to say.

Note, conceptually we could think of 399,842 as the number 50,669 having been encrypted "negative one" times, and similarly, 214,689 is the number 46,375 having been encrypted "negative one" times. By using Rich's public key to encrypt 399,842 and 214,689 once (positive one times), these actions "cancel out" leaving the original integers (50,669 and 46,375) as a result.

Authentication is as vast a topic as encryption. The above is only the simplest version of the RSA challenge-response authentication method. Additional safeguards can be added.

For example, in this problem, if Rich already knew the phone number of his bank, had it in his smartphone, or if he found it on the web, then he can be confident that he's communicating with his bank and not an entity that is pretending to be his bank. However, if two email servers communicate with each other across the internet, until they finish their mutual authentication process, they should not trust each other. Rich and his bank have a one-way authentication problem. On the internet, we typically encounter two-way authentication problems, which are more complicated.

To make matters even more complex, there could be an adversary in the middle of the communications process. This gives rise to a family of cryptographic attacks called man-in-the-middle attacks.

Problem 10-7-19

Near the beginning of the previous module, I made the claim that if I were to replace someone's e with $e+z\varphi$, for any z in the positive integers, then all encryption and decryption behavior would remain unchanged. Vincent does not understand why this must be true. Therefore, write a proof (using mathematical induction) that if e is replaced with $e+z\varphi$ for any positive integer z, that the value of $m^e \bmod N$ remains unchanged.

Hint: for all $x \bmod pq$, it is true that $x^{\varphi+1} \equiv x \bmod pq$, where $\varphi = (p-1)(q-1)$, $p \neq q$, and both p and q are primes.

Proceed by induction on z. The base case is $z = 1$. Since $c \equiv m^e \bmod N$, if we replace e with $e + 1\varphi$, then we have

$$m^{e+\varphi} = m^{e-1+1+\varphi} = (m^{e-1})(m^{1+\varphi}) \equiv (m^{e-1})m^1 \equiv m^e \equiv c$$

or as suggested by my proofreader Tanner Verber, we can prove that more simply by

$$m^{e+\varphi} = m^e m^\varphi \equiv m^e m^0 = m^e 1 = m^e = c$$

Suppose the theorem works for some z_0. If $z = 1 + z_0$, then if we replace e with $e + z\varphi$, we have

$$m^{e+z\varphi} = m^{e+(1+z_0)\varphi} = m^{e+\varphi+z_0\varphi} = m^{e+z_0\varphi+\varphi} = m^{e+z_0\varphi}m^{\varphi} \equiv m^e m^{\varphi} \equiv m^{e+\varphi}$$

yet from the base case, we already know that

$$m^{e+\varphi} \equiv m^e$$

thus we know that

$$m^{e+z\varphi} \equiv m^e$$

In conclusion, for all positive integers z, $m^{e+z\varphi} \equiv m^e \bmod N$. Equivalently, RSA encryption remains unchanged if we replace e with $e + z\varphi$, for any positive integer z.

Because e, $e + \varphi$, $e + 2\varphi$, $e + 3\varphi$, $e + 4\varphi$, ..., would all encrypt identically, I consider e and d to be living in the world of "mod φ" arithmetic. As I mentioned near the start of the previous module, this is why I use the $\equiv$ sign with e and d, and not the $=$ sign.

For small moduli, you can test the hint given above with the following code. Please don't let the modulus be more than 1000, however. Otherwise you will put too much strain on the `sagecell.sagemath.org` servers, which are being provided to the world for free, but which nonetheless have only finite computing power.

```
modulus = 13*17

#####
##### Don't change anything that follows...
##### ...unless you know what you are doing

assert modulus in ZZ
assert modulus > 1
assert modulus < 1500

exponent = euler_phi(modulus) + 1

found = 0

for k in range(0, 2*modulus):
    test = pow( k, exponent, modulus )
    if (test != k):
        print(k, "^", exponent, "mod", modulus, "=", test)
        found = found + 1

if (found == 0):
    print("No exceptions found.")
else:
    print(found, "exceptions were found.")
```

Interestingly, the more general statement of the hint is false. If I replace pq with N, and I allow N to be something other than the product of two distinct primes, then the hint becomes "for all x mod N, it is true that $x^{\varphi+1} \equiv x \bmod N$, where $\varphi = \Phi(N)$." As a counterexample, consider $N = 13^2 = 169$, which is not the product of two distinct primes. As it turns out, $\Phi(169) = 156$. We might expect that raising 26 to the 157th power would yield 26 mod 169, but this is false. You can test that by typing `pow( 26, 157, 169 )` into Sage, and you will see that the answer is zero. It is also fun to change the line `modulus = 13*17` into `modulus = 13*13` in the Sage code that was given above.

Problem 10-7-20

Consider the standard "alphabet soup" of RSA. Give me a formula for $N - \varphi$, in terms of p and q. (This might seem pointless, but trust me, this will be useful momentarily.)

$$N - \varphi = pq - (p-1)(q-1) = pq - (pq - p - q + 1) = pq - pq + p + q - 1 = p + q - 1$$

A Taste of Computational Hardness

Suppose there existed a device that could take N and return $\varphi = \Phi(N)$, for any N that is a product of two large distinct primes. Clearly, such a device would be a catastrophe for RSA, because we saw in Problem 10-7-2 (from Page 2 of this module) that when φ becomes known, it is easy to compute the private key that matches the corresponding public key. We will now prove that this device can be used, in essentially the same amount of time, to factor N. (That might seem to be an irrelevant objective right now, but it will turn out that this will allow us to get a small glimpse of an otherwise extremely advanced topic: the connections between complexity theory and cryptography.)

We will actually be making use of an old trick from algebra classes in middle schools (and high schools). Stated as a theorem:

> Two real numbers r and t satisfy $r + t = S$ and $rt = P$ if and only if r and t are the roots of this polynomial:
>
> $$x^2 - Sx + P = 0$$

In the previous problem, we learned that $N - \varphi = p + q - 1$, but you probably were wondering what the use of that might be. We will vary this formula only slightly to write $N - \varphi + 1 = p + q$, by adding one to each side.

Now imagine that Alice has Bob's public key, and this device that takes N as an input, and which outputs φ. First, she will take the N from the public key and use the device to get φ. While she does not yet know what p and q are, she does know that $pq = N$ and that

$p + q = N - \varphi + 1$. After computing $S = N - \varphi + 1$, and $P = N$, all Alice only needs to solve the quadratic equation

$$x^2 - Sx + P = 0$$

and the two distinct roots will be $x = p$ and $x = q$. The quadratic formula from middle school (or high school) will do that work for Alice.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = \frac{S \pm \sqrt{S^2 - (4)(1)(P)}}{2}$$

We can model the amount of computation to do this either with time, with the number of arithmetic operations, or with the number of assembly-language instructions that a microprocessor will require. Personally, I prefer the last choice, but it is actually the least popular among cryptologists. Most cryptologists count operations, so we'll do that instead.

- We need two operations to find $S = p + q = N - \varphi + 1$.
- We need two more to compute S^2 and $4P$, and a third to subtract them.
- The sixth operation would be the square root of $S^2 - 4P$.
- Then $(S - \sqrt{S^2 - 4P})/2$ and $(S + \sqrt{S^2 - 4P})/2$ each require two more operations.
- In total, we need only 10 additional operations. (This would be a matter of nanoseconds on a modern computer.)

We have now proven that "if a device can (in time T) take N and return $\varphi = \Phi(N)$, for any N that is a product of two large distinct primes, then one can factor N in only slightly more time." Alternatively, "if a device can (with K operations) take N and return $\varphi = \Phi(N)$, for any N that is a product of two large distinct primes, then one can factor N using only $K + 10$ operations."

Before we continue, I think it is useful to perform this operation with two specific cases, using actual numbers.

Problem 10-7-21

Consider Boris, who has a public key of $e \equiv 262,795$ and $N = 588,983$, and Natasha, who has a public key of $e \equiv 192,073$ and $N = 687,811$. Alice has the device that we were discussing moments ago, that can take N and return $\varphi = \Phi(N)$, for any N that is a product of two large distinct primes.

- First, Alice puts Boris's $N = 588,983$ into her device and obtains $\Phi(588,983) = 587,448$. What does this tell us about the sum of Boris's primes?

$$p + q = N - \varphi + 1 = 588,983 - 587,448 + 1 = 1536 = S$$

For completeness, note that the Sage command is `euler_phi( 588983 )` or `euler_phi( 588_983 )` if you prefer.

- What will the product of Boris's primes be? $pq = N = 588{,}983 = P$
- What quadratic equation must Alice now solve? $x^2 - 1536x + 588{,}983 = 0$
- What are the roots of that quadratic equation? 739 and 797

  Alice can also do this with Sage, by typing
  `solve( x^2 - 1536*x + 588_983 == 0, x )`
  She would receive as output

  $$[x == 739, x == 797]$$

  so she checks that

  $$(739)(797) = 588{,}983 \text{ and also } 739 + 797 = 1536$$

  and clearly, she is correct.

- Please note that Alice would probably never solve the quadratic equations above. That's because once she has Boris's φ in her hands, she can just compute $d \equiv e^{-1} \bmod \varphi$ and that will let her see Boris's d (as we saw in Problem 10-7-2, from Page 2 of this module). There is no additional advantage for her to see Boris's two primes as well, unless she wants to see them.

  Okay, noted.

- Second, Alice puts Natasha's $N = 687{,}811$ into her device and obtains $\Phi(687{,}811) = 686{,}152$. What does this tell us about the sum of Natasha's primes?

  $$p + q = N - \varphi + 1 = 687{,}811 - 686{,}152 + 1 = 1660 = S$$

  For completeness, note that the Sage command is `euler_phi( 687811 )` or `euler_phi( 687_811 )` if you prefer.

- What will the product of Natasha's primes be? $pq = N = 687{,}811 = P$
- What quadratic equation must Alice now solve? $x^2 - 1660x + 687{,}811 = 0$
- What are the roots of that quadratic equation? 797 and 863

  Alice checks that

  $$(863)(797) = 687{,}811 \text{ and also } 863 + 797 = 1660$$

  and clearly, she is correct.

It is worth noting that I generated these public keys for Boris and Natasha randomly. It is unfortunate luck that Boris and Natasha had the prime 797 in common. This would make their public keys easily breakable, using the gcd method explained earlier in Problem 10-7-6 (on Page 3 of this module). I did not cause that to happen deliberately—the shared prime occurred by coincidence.

The Real Lesson of the Previous Problem

With considerable effort over the last few pages, we proved that "if a device can (using K operations) take N and return $\varphi = \Phi(N)$, for any N that is a product of two large distinct primes, then one can factor N using only $K + 10$ operations." I'm now going to state a sequence of sentences that are logically equivalent, so that you can get an idea of what complexity theory statements sound like. It would take a large number of pages to rigorously define all the concepts laid out here, so please don't be frustrated if some of what follows seems to be vague. Entire books[1] have been written on the connections between complexity theory and cryptography, so there is no need for this textbook to dwell too long on this topic.

We can phrase our conclusions more mathematically by saying "if there exists a device that can (in polynomial time) take N and return $\varphi = \Phi(N)$, for any N that is a product of two large distinct primes, then there exists a device that can factor N in only slightly more time."

The contrapositive is much more exciting. The contrapositive is "if there does not exist a device that can factor N (in polynomial time), for any N that is a product of two large distinct primes, then there does not exist a device that can take N and return $\varphi = \Phi(N)$."

Computer scientists will abbreviate this by saying "if factoring the product of two distinct large primes is computationally infeasible, then computing the Euler Totient Function of the product of two distinct large primes is computationally infeasible."

We can compress this even further by saying "if factoring the product of two distinct large primes is hard, then computing the Euler Totient Function of the product of two distinct large primes is hard." (Personally, this is how I prefer to say such things when I am thinking about this topic in my research.)

Some would even say that "computing the Euler Totient Function of the product of two distinct large primes is no easier than factoring the product of two distinct large primes."

Of course, if I have a device that can factor the product of two distinct large primes, in addition to being able to break RSA (and read everyone's mail plus sign important documents, such as checks, as if I were any person whose public key I have), I could also compute $\Phi(N)$ for those numbers. Once I've factored $N = pq$, then it is trivial to compute $\varphi = (p-1)(q-1) = \Phi(N)$. Therefore, I can confidently say "computing the Euler Totient Function of the product of two distinct large primes is no harder than factoring the product of two distinct large primes."

In conclusion, I should summarize this discussion by saying we've proven that "computing

[1]A notable example is *Complexity Theory and Cryptology*, by Jörg Rothe, published by Springer in 2005.

the Euler Totient Function of the product of two distinct large primes is computationally equivalent to factoring the product of two distinct large primes."

A Happy Note to End this Module

Believe it or not, this insight that knowledge of $\Phi(N)$ can be used to factor N launched the undergraduate research project of a student under my supervision, when I was teaching at Fordham University in The Bronx. His name is Kyle Kloster, and his Honor's Bachelor's thesis was a new method of factoring integers, aimed at integers that are the product of two large distinct primes. You can read that Honor's Bachelor's thesis, entitled "Factoring a semiprime n by estimating phi(n)," which he presented on May 7th, 2010, and I'm sure you'll agree that it could have been a Master's thesis. Kyle later went on to earn a PhD in Mathematics from Purdue University in 2016. That Honor's Bachelor's thesis is located at

`http://www.gregory-bard.com/publications.html`