Module 10.6: Introducing the RSA Cipher

Gregory V. Bard

February 4, 2020

- This is a practice workbook for the RSA cipher.
- 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).
- Vital information about how RSA is used in practice is given at the end of the module. Be sure to read it.

Now, let's start by looking at the RSA cryptosystem as a three-procedure collection of algorithms.

The RSA Cryptosystem

The RSA Cryptosystem consists of three pieces. There is a key-generation procedure, executed once (or once in a while) by each user. That protocol gives each user that executes it a public key and a private key. In some sense, these terms are major understatements. The public key should be published in a very public way, and the private key should be kept extremely private—never shown to anyone other than the person who generated it. The second piece is an encryption procedure, which anyone can use to transmit an encrypted message to any person whose public key is known to the sender. The third piece is a decryption procedure, where the recipient uses their private key to recover the plaintext of some ciphertext sent to them.

We will now explore those three procedures. It should be noted that RSA can also be used for digital signatures, a topic of enormous importance to e-commerce. However, we cannot explore digital signatures just yet.

The Key-Generation Procedure:

1. Randomly generate a huge prime integer p.
2. Randomly generate another huge prime integer q.
3. Compute $N = pq$
4. Compute $\varphi = \Phi(N) = (p-1)(q-1)$
5. Randomly generate an integer e such that $2 < e < \varphi - 2$ and e is coprime to φ.
6. Compute $d \equiv e^{-1} \bmod \varphi$.
7. Destroy p, q, φ.
8. Publish e and N as the public key.
9. Retain d as your private key.

By the way, most sites on the internet say $1 < e < \varphi$. Nonetheless, by the end of Module 10-7, you'll see why I am right and most websites are incorrect. I think you will find Question 10-7-13 particularly convincing about why we should rule out $e = \varphi - 1$. Oddly, most websites about RSA on the internet appear to ignore the vulnerability described in Question 10-7-13.

What is a "Huge" Prime?! Naturally, the word "huge" requires some clarification. In this module, and the next, we will mostly use 3-digit primes, but also some 4-digit primes. In practice, no qualified cryptologist in 2019 would dispute the statement that 100-digit primes are definitely too small to be considered secure in actual usage. The primes that should be used, in real applications, are considerably larger than 100 digits.

The goal is to imagine an adversary gathering together all the computing power in the world, and throwing it at factoring your N back into p and q. You must choose to make p and q large enough, so that the adversary is very unlikely to succeed, even using all the computing power in the world, perhaps for many years.

If you'd like to get an idea of where the horizon lies, between integers that are factorable and integers that are too large to be factored, then you might want to consult a list of world factorization records. Wikipedia maintains a list of integer-factorization world records. The article has the title "Integer Factorization Records," but be sure to look under the heading "Numbers of a General Form." `https://en.wikipedia.org/wiki/Integer_factorization_records`

However, when I resigned from the US National Security Agency (the NSA) in 2002, one of the things that I agreed to was that I would never endorse any particular key size for any cryptosystem. Therefore, I cannot and will not recommend a specific size for the primes p and q in actual use. Mathematically, we can learn much just by working with 3-digit and 4-digit primes.

The Encryption Procedure:

1. Some plaintext message m, which is an integer mod N, is ready to be sent to the receiver.
2. Download the receiver's public key, e and N.
3. Compute $c \equiv m^e \bmod N$.
4. Transmit c as the ciphertext to the receiver.

The Plaintext Messages are Actually Integers: Most students are surprised that the plaintext "messages" are actually integers in the range $0 < m < N$. It is easy to imagine sending a PIN (Personal Identification Number), a time of day, a calendar date, the combination of a safe, a postal code or ZIP-code, in this way. At the very end of this module, we'll see that bit strings for use with block ciphers are frequently encrypted using RSA during e-commerce on the World Wide Web.

The Decryption Procedure:

1. Some ciphertext c, an integer mod N, has been received.
2. Compute $m' \equiv c^d \bmod N$.
3. The plaintext is m', read and enjoy it.

A Note about m' versus m in notation:

It is extremely worthwhile to study the proof that the receiver, upon computing m', has the same message that was transmitted, m. Mathematically, $m' \equiv m$ mod N. There are actually two very different proofs of this fact. (The simpler one is given on Page 20 of this module.) After those proofs have been studied, there is no need to distinguish between m and m', and therefore the symbol m' will not be needed again. Instead, m is used in both places for the rest of this book and in actual practice.

A Quick Note about Phi:

The symbols ϕ, φ, and Φ all represent the letter "phi" in Greek, which makes the same sound as F or f in English. Mathematicians pronounce this to rhyme with "lie," "my," and "bye," like the last syllable in "defy." However, actual Greeks pronounce it like "fee." In any case, ϕ looks too similar to the symbol for the empty set, so I don't like to use it at all. The capital letter Φ is used to denote the Euler Totient Function $\Phi(N)$, and indicates how many of the integers between 0 and N are coprime to N. The output of this function is denoted by φ. In emails or text messages, where using φ or $\Phi(N)$ would be inconvenient, you can type "`phi`" or "`Phi(N)`" and it will be clear to any reader with sufficient mathematical background.

However, the book *Cryptography with Coding Theory*, written by my dissertation advisor, Prof. Lawrence Washington, and his student Prof. Wade Trappe, 2nd edition, 2005, uses $\phi(n)$ instead of $\Phi(N)$, and ϕ instead of φ. It is not uncommon in mathematics for different textbooks to use slightly different notation.

In any case, remember that

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

because N is the product of two distinct primes. If N were not the product of two distinct primes, then this formula for $\Phi(N)$ would be incorrect.

A Quick Note about $=$ versus $\equiv$ as symbols:

If a computation is being performed in the ordinary integers, then I will use an $=$ sign. However, if a computation is being performed with modular arithmetic, then I will use the $\equiv$ sign. Most readers will treat these as equivalent. Indeed, you probably don't have to worry about this distinction unless you want to write and publish research papers about cryptography. Nonetheless, it can be helpful to stop and think, once in a great while, why $=$ is used here and $\equiv$ is used there. Try not to lose any sleep over this matter.

Because p, q, N, and φ are actual integers, computations involving them should use the $=$ sign. Because c and m unambiguously live in the world of "mod N" arithmetic, computations involving them should use the $\equiv$ sign.

With e and d it is not so obvious, and many readers might want to skip this paragraph. If I were to replace e with $e + z\varphi$, for any z in the positive integers, then all encryption and decryption behavior would remain unchanged. (This is far from obvious, and we will explore this property as a problem in the next module, namely 10-7-19.) Because e, $e+\varphi$, $e+2\varphi$, $e+3\varphi$, $e+4\varphi$, $\ldots$, would all encrypt identically, I consider e and d to be living in the world of "mod φ" arithmetic, and therefore I use the $\equiv$ sign with e and d.

How do you actually generate a large random prime?

To generate a large random prime, first, generate a large random integer. Then, test[1] that integer for primality. If the generated integer is prime, then that's a success. If not, then start over.

It is an extremely interesting question to compute the expected value of the number of integers that would have to be generated until a prime is found.

[1]There are several efficient probabilistic primality tests, and at least one primality test that is not probabilistic has been published. My favorite primality test is the Fermat primality test, but it is probabilistic. A non-probabilistic primality test was discovered in 2002, by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena, so it is called the AKS test. Keep in mind that RSA is from 1978, so for the first 24 years, the world only had probabilistic primality tests. In addition to their cryptologic utility, discussions of primality tests have a lot of cool math topics in them.

How do you actually generate e so that it is coprime to φ?

First, generate a random integer e within the specified range of permitted values. Then, using the Extended Euclidean Algorithm—a very fast way of calculating gcds, compute `gcd`(e, φ). If it equals 1, then e and φ are coprime, which is a success. If the gcd does not equal 1, then that's bad luck. Start over.

This method almost always succeeds on the first or second try. If a third try is needed, then that's a surprise. Proving this fact is moderately interesting, but we will not attempt it at this time.

Working with RSA in Sage

Nearly everything in this module and the next module can be computed using only one command in Sage. This means that even if the reader has little or no familiarity with Sage, it is still a good idea to have a Sage window open as one goes through the exercises of the module.

`https://sagecell.sagemath.org/`

- The command for computing x^y in the ordinary integers (or rational numbers, and other domains) is `pow( x, y )`.
- The command for computing x^y mod z is `pow( x, y, z )`.
- One can find the inverse of x mod z using `pow( x, -1, z )`, recalling the notation x^{-1} for the inverse of x.
- Though it is not needed, there is a command `inverse_mod( x, z )`, which also finds the inverse of x mod z.
- For ease of reading, a number such as 462,391 can be typed as `462_391` instead of as `462391` if desired. Sage will interpret `462,391` as two integers, 462 and 391, not as one 6-digit integer.
- The command `factor( 462_391 )` or `factor( 462391 )` can be used to factor integers.

Note, the fact that so basic a command as `pow` will use modular arithmetic when given a third (optional) parameter clearly demonstrates the centrality of modular arithmetic to modern mathematics. This is by no means an obscure or rare mathematical operation anymore.

Problem 10-6-1

Bob has just started the key generation for RSA. He has two primes, $p = 103$ and $q = 107$, but clearly he isn't finished yet.

1. What is N going to be?

   $N = pq = (103)(107) = 11,021$

2. What is φ going to be?

   $\varphi = (p-1)(q-1) = (103-1)(107-1) = (102)(106) = 10,812$

3. What are the requirements for e?

   We need e to be coprime to $\varphi = 10,812$. This means that we have to factor 10,812 to understand the requirements further. Factoring 10,812 would be tedious with a scientific calculator, but for this problem, we already know p and q, so we already know $p-1$ and $q-1$.

   Therefore, we can easily obtain

   $$10,812 = (102)(106) = (2)(51)(2)(53) = (2)(3)(17)(2)(53) = 2^2(3)(17)(53)$$

   which means that the prime factorization of e must have no 2s, no 3s, no 17s, and no 53s. Furthermore, $2 < e < 10,810$. Any integer e that meets these requirements will do.

4. Okay, now Bob chooses $e \equiv 2141$. What are the requirements for d?

   It is a simple calculation. We require $d \equiv e^{-1} \bmod \varphi$, which means $d \equiv 2141^{-1} \bmod 10,812$. We need to compute the inverse of 2141 mod 10,812.

   We learn that this is 101, after using `pow(2141, -1, 10812)` in Sage.

5. Which of the six items will comprise Bob's public key? (Variable names are fine.)

   Bob's public key will consist of N and e.

6. Which of the six items will comprise Bob's private key? (Variable names are fine.)

   Only d.

7. As it comes to pass, Bob computes $d \equiv 101$. Later, he asks Charlene what she wants to do this weekend. Charlene downloads Bob's public key, and sends him an encrypted message, $c \equiv 9798$. What does Bob have to compute to decipher the message? (Your answer should be a formula, with numbers but not variables.)

   Bob must compute 9798^{101} mod 11,021.

   After Bob inputs `pow(9798, 101, 11021)` into Sage, it outputs that this is 420.

8. Next, Bob asks Charlene about what time she wants to come over to his place. She responds with the encrypted message $c \equiv 7785$. What does Bob have to compute to decipher the message? (Your answer should be a formula, with numbers but not variables.)

   Bob must compute 7785^{101} mod 11,021.

   After Bob inputs `pow(7785, 101, 11021)` into Sage, it outputs that this is 1430, which is 2:30 PM.

Problem 10-6-2

For each of the following, tell me what is the standard choice of letter (from English or from Greek) for representing that item in RSA.

- The plaintext message. m
- The encryption exponent. e
- The decryption exponent. d
- The two huge primes. p, q
- The ciphertext, or encrypted message. c
- The modulus. N
- The Euler Totient Function of the modulus. φ

Problem 10-6-3

Agent X has recently accepted a job as the head of computer security for a government. In order to be ultra-secure, he's going to use larger numbers than most of the other people in this module.

Agent X has just started the key generation for RSA. He has two primes, $p = 2003$ and $q = 3001$, but clearly he isn't finished yet.

1. What is N going to be?

   $N = pq = (2003)(3001) = 6,011,003$

2. What is φ going to be?

   $\varphi = (p-1)(q-1) = (2003-1)(3001-1) = (2002)(3000) = 6,006,000$

3. What are the requirements for e?

   We need e to be coprime to $\varphi = 6,006,000$. This means that we have to factor 6,006,000 to understand the requirements further. Factoring 6,006,000 would be tedious with a scientific calculator, but for this problem, we already know p and q, so we already know $p-1$ and $q-1$.

   Therefore, we can easily obtain

$$\begin{aligned} 6,006,000 &= (2002)(3000) \\ &= (2)(1001)(30)(100) \\ &= (2)(7)(11)(13)(5)(6)(25)(4) \\ &= (2)(7)(11)(13)(5)(2)(3)(5^2)(2^2) \\ &= (2^4)(3)(5^3)(7)(11)(13) \end{aligned}$$

   which means that the prime factorization of e must have no 2s, no 3s, no 5s, no 7s, no 11s, and no 13s in it, but any other primes are fine. Furthermore, $2 < e < 6,006,000-2$. Any integer e that meets these requirements will do.

   You can ask Sage to find that factorization with `factor( 6006000 )` or more clearly `factor( 6_006_000 )` can be used. Not many of my students would recognize $1001 = 7(11)(13)$ when working with a pencil, but some might.

4. Okay, now Agent X chooses $e \equiv 130,321$. Since this is 19^4, its factorization has no 2s, no 3s, no 5s, no 7s, no 11s, and no 13s, and it also satisfies $2 < e < 6,005,998$. Next, what do we know about Agent X's d?

   We require $d \equiv e^{-1} \bmod \varphi$, which means

$$d \equiv 130,321^{-1} \bmod 6,006,000$$

   We need to compute the inverse of 130,321 mod 6,006,000.

   Sage says that this is 364,081 after using `pow(130321, -1, 6006000)` which can also be typed as `pow(130_321, -1, 6_006_000)` for clarity.

5. President Skroob wants to send Agent X a secret message. (It is the combination for the lock on his luggage, that contains a secret item that will save the universe.) To transmit this vital information securely, President Skroob's Chief of Staff, Colonel Sanders, downloads Agent X's public key. The message is $m \equiv 12345$. What should Colonel Sanders compute in order to encrypt this vital message for Agent X? (Your answer should be a formula, with numbers but not variables.)

   Colonel Sanders must compute $12345^{130,321}$ mod 6,011,003.

   Sage says that this is 1,424,722, after using `pow(12345, 130321, 6011003)` or alternatively `pow(12_345, 130_321, 6_011_003)`

6. What must Agent X compute, in order to decipher $c \equiv 1,424,722$, the ciphertext that he has received from Colonel Sanders? Hint: Agent X's $d \equiv 364,081$.

   Agent X must compute $1,424,722^{364,081}$ mod 6,011,003.

   Sage says that this is 12345, using `pow(1424722, 364081, 6011003)` or `pow(1_424_722, 364_081, 6_011_003)`.

7. The previous two sub-problems are a reference to a movie that was (at one point) famous. To what movie do the previous two subparts refer? (This sub-problem is not officially part of this problem.)

   The movie was *Spaceballs*, released in the year 1987.

   `https://www.youtube.com/watch?v=a6iW-8xPw3k`

The Real Lesson of the Previous Problem

In 2012, several senior administrators in the Syrian Ministry of Presidential Affairs of Bashar al-Assad (the President of Syria), had used the same password, 12345, for their email accounts. Others had used 123456. Before I continue, here are some citations so that you can verify my claims.
`https://o.canada.com/technology/syrian-government-loves-the-password-12345`
`https://www.haaretz.com/1.5182336`
`https://mashable.com/2012/02/07/anonymous-assad-email-password/`
`https://www.forbes.com/sites/parmyolson/2012/02/07/hacked-syrian-e-mails-advise-pres-assad-that-american-psyche-is-easily-manipulated/`

Now I'd like to talk about what this means for computer engineers, computer scientists, and cybersecurity students today. Perhaps you might have the most up-to-date virus scanners, excellent firewall rules, properly implemented cryptography, and carefully controlled permissions. Nonetheless, if even one of your users is an idiot, then all your work has been for naught and your system is vulnerable. Therefore, the paramount responsibility of a computer-security professional today is educating the network's users, so that they know what *not* to do.

Problem 10-6-4

Let's return to the previous problem, where Agent X was decrypting the message sent to him by President Skroob through Colonel Sanders. It is important to realize that we don't compute

$$c^d = 1,424,722^{364,081}$$

in the ordinary integers, and then follow that up by reducing mod $N = 6,011,003$ to obtain

$$1,424,722^{364,081} \bmod 6,011,003$$

That's because $1,424,722^{364,081}$ is a rather large integer. In fact, it has 801,961 decimal digits. In hexadecimal, it has 666,014 digits, and in binary, it has 2,664,055 bits. If you ask Sage to compute $1,424,722^{364,081}$ using `pow(1424722, 364081)` or `pow(1_424_722, 364_081)` then it will display that number, accompanied by a scroll bar for your convenience. It is unclear what you might actually do with such an enormous integer.

It is vital that I mention that even using 100-digit primes in RSA is far, far too small to be computationally secure in 2019. If we were to use primes that were only 100 digits long, then m, e and N would be 200-digit numbers. The value of m^e in the ordinary integers would be insanely large. If we were to be so foolish as to compute it, it would be roughly 2×10^{202} digits long. (That means that the number of digits in the number of digits of m^e is about 202.) Instead, we must use the algorithm that is usually called either "rapid modular exponentiation" or "the method of repeated squaring."

If the reader grasps this point, then it is okay to move on to the next problem now, because the point has been made. However, some advanced students might want to explore this point more precisely. In general, we might want a general formula for computing how many (decimal) digits x^y would have in the ordinary integers. Similarly, we might want to know how many bits x^y would have as an ordinary integer written in binary.

This problem is too hard as written, so I'll give you a hint. If r is an 8-digit number, then $10^7 \leq r < 10^8$. Likewise, if s is a 9-bit number, then $2^8 \leq s < 2^9$. I will offer you another hint: use logarithms.

Starting with

$$10^7 \leq r < 10^8$$

we will take the common logarithm ($\log_{10}$) of both sides to obtain

$$7 \leq \log_{10} r < 8$$

which means that the common logarithm of r is between 7 and 8.

In general, for any k-digit integer r, our inequality $10^{k-1} \leq r < 10^k$ becomes

$$(k-1) \leq \log_{10} r < k$$

which means that the common logarithm of r is between $k-1$ and k, for any k-digit integer r written in decimal.

Therefore, we can see that the ceiling of $\log_{10} r$, usually written $\lceil \log_{10} r \rceil$, equals k, the number digits of r. Recall that the ceiling of a real number t is formally the least integer greater than t. Informally, we take t and round it up to the next integer, unless t is already an integer, in which case t remains unchanged.

Applying this to x^y results in the following

$$\lceil \log_{10} x^y \rceil = \lceil y \log_{10} x \rceil \approx y \lceil \log_{10} x \rceil = y \left\lceil \frac{\log_e x}{\log_e 10} \right\rceil$$

where $\log_e x$ represents the natural logarithm of x. In American calculus textbooks, this is usually written $\ln x$.

For binary, we simply replace the common logarithm with the binary logarithm (we replace $\log_{10}$ with $\log_2$) to obtain

$$y\left\lceil \frac{\log_e x}{\log_e 2} \right\rceil$$

as the number of bits required to write x^y as an ordinary integer in binary.

This can be remembered more easily by saying that the length of x^y in digits is approximately y times the length of x in digits. Likewise, the length of x^y in bits is approximately y times the length of x in bits.

If you want to do calculations of this kind in Sage, such as to count precisely the number of decimal digits in $1,424,722^{364,081}$, then you can type

```
ceil( log( pow( 1_424_722, 364_081 ), 10 ) )
```

but replace the 10 with 16 for hexadecimal or 2 for binary.

Problem 10-6-5

Vernon and Marge are accountants for a very large bank, and have started a friendship. They communicate via email, because they live thousands of miles apart. Marge gets curious and asks Vernon the year that he was born. He doesn't mind telling Marge, but he knows that the bank monitors all employee emails, and he's afraid of being the victim of age discrimination.

Therefore, Marge suggests that they use RSA, and she provides her public key: 57 and 2173. Vernon replies with the ciphertext 409. Marge's private key is 73. In what year was Vernon born?

Note, for this problem, we want the actual plaintext (Vernon's exact age), not a formula. Compute it by hand (perhaps with a scientific calculator to assist you), but not using a computer. I recommend the "rapid modular exponentiation" algorithm, sometimes called "the method of repeated squaring."

First, we have to understand RSA and that what we need to compute is $m \equiv c^d \bmod N$ in general, or

$$m \equiv 409^{73} \bmod 2173$$

for this problem. We begin by observing

$$d = 73 = \text{binary}(1001001) = 64 + 8 + 1$$

which informs us that we need the 64th, 8th, and 1st powers.

Now by repeated squaring, we must get the powers of c. We must be careful to reduce mod N at each step (to prevent overflow), and to stop at the 64th power (to avoid wasting time).

- Obviously, $409^1 = 409$.
- First squaring, $409^2 = 167,281 = 165,148 + 2133 = 76(2173) + 2133 \equiv 2133$.

- Second squaring, $409^4 \equiv 2133^2 \equiv 4,549,689 \equiv 4,548,089+1600 \equiv 2093(2173)+1600 \equiv 1600$.
- Third squaring, $409^8 \equiv 1600^2 \equiv 2,560,000 \equiv 2,559,794 + 206 \equiv 1178(2173) + 206 \equiv 206$.
- Fourth squaring, $409^{16} \equiv 206^2 \equiv 42,436 \equiv 41,287 + 1149 \equiv 19(2173) + 1149 \equiv 1149$.
- Fifth squaring, $409^{32} \equiv 1149^2 \equiv 1,320,201 \equiv 1,319,011 + 1190 \equiv 607(2173) + 1190 \equiv 1190$.
- Sixth squaring, $409^{64} \equiv 1190^2 \equiv 1,416,100 \equiv 1,414,623 + 1477 \equiv 651(2173) + 1477 \equiv 1477$.

Careful users of a scientific calculator would want to check their work. For example, to check the sixth squaring, we will verify if $1,416,100 - 1477 = 1,414,623$ is a multiple of 2173. We ask our calculator to find $1,414,623 \div 2173$ and we get 651, an integer, so we are happy. Likewise, to check the fifth squaring, we will verify if $1,320,201 - 1190 = 1,319,011$ is a multiple of 2173, by asking our calculator what $1,319,011 \div 2173$ equals. Happily, we get an integer (607), so we know that 1,319,011 is a multiple of 2173. Similarly, to check the fourth squaring, we will verify if $42,436 - 1149 = 41,287$ is a multiple of 2173, and that turns out to be true also. A prudent student would check all the squaring answers in this way.

Now we are prepared to compute

$$409^{73} = 409^{64+8+1} = (409^{64})(409^8)(409^1) \equiv (1477)(206)(409)$$

I think that on most scientific calculators, (1477)(206)(409) will overflow. Therefore, we need to do it in two separate steps, which is not so unusual.

The first step is

$$(1477)(206) = 304,262 = 304,220 + 42 = (140)(2173) + 42 \equiv 42$$

and then the second step is

$$(1477)(206)(409) \equiv (42)(409) = 17,178 = 15,211 + 1967 = 7(2173) + 1967 \equiv 1967$$

These can be checked by verifying that both $304,262 - 42 = 304,220$ and $17,178 - 1967 = 15,211$ are multiples of 2173. Happily, they are multiples of 2173.

In conclusion, Vernon was born in the year 1967.

Now you know what Sage is doing when you ask for `pow( 409, 73, 2173 )` or something like that.

Problem 10-6-6

Let's suppose that Jing is interested in cryptocurrencies, but to warm up to that advanced topic, she decides to study RSA first. Having read Problem 10-6-1 (on Page 6 of this module) carefully, she decides to try $p = 101$ and $q = 107$.

1. What φ will Jing have?

   $\varphi = (p-1)(q-1) = (101-1)(107-1) = (100)(106) = 10,600$

2. What N will Jing have?

   $N = pq = (101)(107) = 10,807$.

3. What are the requirements for e?

   We need e to be coprime to $\varphi = 10,600$. This means that we have to factor 10,600 to understand the requirements further. Luckily, we know that $\varphi = (100)(106)$, so we have a huge head start on computing the factorization into primes. We obtain

   $$10,600 = (106)(100) = (2)(53)(4)(25) = (2^3)(5^2)(53)$$

   which means that the factorization of e must have no 2s, no 5s, and no 53s. Furthermore, $2 < e < 10,598$. Any integer e that meets these requirements will work.

4. Okay, now Jing chooses $e \equiv 2143$. What are the requirements for d?

   We require $d \equiv e^{-1} \bmod \varphi$, which means we need to compute the inverse of 2143 mod 10,600. That is $d \equiv 2143^{-1} \bmod 10,600$.

   Sage says that this is 5807, using `pow(2143, -1, 10600)`.

Note: As it comes to pass, Jing computes $d \equiv 5807$. You will need this to answer future subproblems.

Okay, got it.

5. Which of the six variables should Jing destroy? (Variable names are fine.)

   Jing should destroy φ, p, and q.

6. Which of the six variables will become Jing's public key? (Variable names are fine.)

   Jing's public key will consist of N and e.

7. Which of the six variables will become Jing's private key? (Variable names are fine.)

   Jing's private key is only d.

8. Jing's uncle Zhang is excited about Jing's prospects, and sends her an encrypted message, $c \equiv 6076$. What does Jing have to compute to decipher the message? (Your answer should be a formula, with numbers but not variables.)

   Jing must compute 6076^{5807} mod 10,807.

   When Jing inputs `pow(6076, 5807, 10807)`, Sage outputs that this is 888. This number means "good fortune" or "lots of money" in some classical Chinese numerology.

9. Louis is in a jazz band that Jing likes, and he wants to send Jing a secret message. He downloads her public key. The message is $m \equiv 5678$. What should Louis compute in order to encrypt this message for Jing? (Your answer should be a formula, with numbers but not variables.)

   Louis must compute 5678^{2143} mod 10,807.

   Sage says that this is 6973, using `pow(5678, 2143, 10807)`.

Problem 10-6-7

Let's see how much you remember of the "alphabet soup" in RSA. Describe each of these with a short phrase.

- What is m? The plaintext message.
- What is d? The decryption exponent.
- What is N? The modulus.
- What is e? The encryption exponent.
- What is q? One of the two huge primes.
- What is φ? The Euler Totient Function of the modulus.
- What is c? The ciphertext message.
- What is p? One of the two huge primes.

Problem 10-6-8

As it comes to pass, a Canadian CEO named Elon is on vacation in Cuba, and his intern Billy needs a document that is stored in the company safe. They are communicating via email, and both know it is very possible that the Cuban government is monitoring all communications going in and out of Cuba.

At times, many governments have banned the use of SSL/TLS, and in this problem we are assuming that Cuba is doing that. This means that any emails between Billy and Elon can be read (in the clear) by any computer or router between them en route.

Elon wants to send Billy the combination to the company safe, and you can assume that nothing more needs to be communicated. Billy is a newly hired intern, so he does not yet have any cryptographic credentials with the company. He runs the RSA key-generation algorithm, and gets the following output, but unfortunately Billy accidentally destroyed two entries with a coffee stain.

$$\begin{aligned} d &= 119{,}093 \\ e &= 191{,}381 \\ q &= 653 \\ p &= 719 \\ N &= \text{<coffee stain/>} \\ \varphi &= \text{<coffee stain/>} \end{aligned}$$

- What are the two numbers that were obliterated by the coffee stain? (I'd like the actual numbers.)

  They are $N = (719)(653) = 469{,}507$ as well as

  $$\varphi = (719-1)(653-1) = (718)(652) = 468{,}136$$

- Billy needs to tell Elon some key material to enable encrypted communications between them, knowing well that anything typed on the hotel computer can be read by untrustworthy parties. Which numbers does Billy send to Elon? (For the Science Olympiad, I added "You will not get any points if you omit one or more needed numbers, nor will you get any points if you include any extraneous numbers.")

  Billy should send Elon 191,381 and 469,507.

- Now, Elon wants to transmit the (encrypted) combination to the safe in the response email, encrypted with RSA. The combination is `27-18-31`, which will be encoded as the number 271,831. What should Elon compute in order to know what the ciphertext is? Write a formula with numbers and mathematical symbols, but no letters. Don't compute the final answer, which would be very difficult by hand

  Elon must compute

  $$(271{,}831)^{191{,}381} \mod 469{,}507$$

  If Elon types `pow( 271831, 191381, 469507 )` or equivalently `pow( 271_831, 191_381, 469_507 )` into Sage, the output is 253,055.

- As it comes to pass, Elon transmits 253,055 but now you must tell me what Billy should do with that to decrypt it. Again, write a formula with numbers and mathematical symbols, but no letters. Don't compute the final answer, which would be very difficult by hand.

  Billy must compute
  $$(253,055)^{119,093} \mod 469,507$$

  If Billy types `pow( 253055, 119093, 469507 )` or equivalently `pow( 253_055, 119_093, 469_507 )` into Sage, the output is 271,831, representing the combination `27-18-31`, as desired.

The Real Lesson of the Previous Problem

If Elon is well known as a technology CEO, even of a medium-sized Canadian company, it is very possible that he might be passively monitored by the Cuban intelligence services. Every phone call can be listened to, and if TLS/SSL is banned, then all emails can be read. Despite those limitations, if Elon's laptop has Sage or some other computer-algebra system installed, he can encrypt the company safe's combination in his hotel room using Sage (or a competing product). So long as his screen isn't viewed by some sort of hidden camera, the combination will remain private because the only things that are transmitted are Billy's public key (which is public after all) and the ciphertext. If Elon were worried that there was a camera (or ten) in the ceiling of his hotel room, then he could operate his laptop underneath his bedsheets.

The only thing that would be different in practice is that the numbers would be much larger. Also, we could easily imagine something more sensitive being transmitted, such as the password to the company DropBox account.

In any case, the power of public-key cryptography is that it enables private communications even when every aspect of the communications between the two parties is being intensely monitored.

Problem 10-6-9

Alice and Bob are at the beginning of a romantic relationship, but because of strict parents and slightly conservative elderly neighbors, they don't want anyone to know anything about it. They are using RSA with the following numbers to communicate:

Alice		Bob	
N	$= 4,060,081$	N	$= 4,056,187$
φ	$= 4,056,052$	φ	$= 4,052,160$
p	$= 2003$	p	$= 2011$
q	$= 2027$	q	$= 2017$
e	$= 23$	e	$= 19$
d	$= 1,939,851$	d	$= 2,559,259$

- Which numbers comprise Bob's public key?

  19 and 4,056,187

- Which numbers comprise Alice's public key?

  23 and 4,060,081

- When we write the two numbers that make up the public key, why do we not need to say which is e and which is N?

  Because of three facts. First, $e < \varphi - 2$; second, $\varphi - 2 < \varphi$; third, $\varphi < N$ because

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

  Together, these three facts imply that $e < N$. So when the two numbers that make up the public key are written, we will always know that the smaller one is e, and the larger one is N.

- Bob wants to send Alice his proposed time for their visit later this evening. He wishes to send 1915. What should Bob compute, to determine the ciphertext that he should transmit to Alice? (Your answer should be a formula with numbers and operations, but no variables.)

  Bob should compute 1915^{23} mod 4,060,081.

  Using Sage, we input `pow(1915, 23, 4060081)` or `pow(1915, 23, 4_060_081)` and see that the output is 3,543,671.

- Alice has too much homework, so she decides that Bob's choice won't do. Alice wants Bob to come over at a different time, to balance the need to do homework with the need for fun. Alice transmits the ciphertext 489,846 to Bob. What should Bob compute to find out what time Alice wants him to come over? (Your answer should be a formula with numbers and operations, but no variables.)

  Bob should compute $489,846^{2,559,259}$ mod 4,056,187.

  Using Sage, we input `pow(489846, 2559259, 4056187)` or `pow(489_846, 2_559_259, 4_056_187)` and see that the output is 2345. That's 11:45 PM.

- Instead of 19, Bob was thinking about other candidate values of e. Which of the following would have worked, or would not have worked, as values of e for Bob?

  $$\{14, 15, 16, 17, 18, 21, 49\}$$

  We can rule out anything even, because φ is even. Clearly, two even numbers cannot be coprime, because they are both divisible by two. That knocks out 14, 16, and 18 from further consideration. Since Bob's $\varphi = 4,052,160$ ends in a zero, we know it is divisible by five. Since 15 is also divisible by 5, we know that 15 and 4,052,160 are not coprime, and this knocks out 15.

  This leaves 17, 21, and 49. Since both 21 and 49 are divisible by 7, we should ask our hand calculator if 4,052,160 is divisible by 7. Since

  $$4,052,160 \div 7 = 578,880$$

  is an integer, we know that 21 and φ, as well as 49 and φ, share 7 as a divisor. This eliminates both 21 and 49 from further consideration.

  Since 17 is actually prime, we can easily see if it is coprime to φ by dividing φ by 17 on a hand calculator. We obtain

  $$4,052,160 \div 17 = 238,362.352\cdots$$

  is not an integer. Thus we know that 4,052,160 is not divisible by 17, and therefore 17 is coprime to 4,052,160.

  In conclusion, Bob could have chosen $e \equiv 17$, but none of those other es could work.

Problem 10-6-10

This problem is just further practice, encrypting and decrypting messages in the situation of the previous problem. As before, your answers should be formulas, with numbers and operations, but no variables. For example,

$$5678^{1234} \bmod 9876$$

Refer to the previous problem to get the key material for Alice and Bob. In order to make this problem more fun, rather than blindly repetitive, let's raise the stakes! I'd like to ask you to write down your formulas to answer these four subproblems, and finalize your choices for all four of your answers, before checking anything in the with-solutions version of this packet. At this point, you really should be able to get all four answer-formulas absolutely perfectly. After all, a formula that is slightly off will give a wrong answer, and in real life, correct answers are required.

- Bob wants to send the plaintext message 1945 to Alice, representing 7:45 PM. What must he compute to encrypt that?

  Bob must compute $1945^{23} \bmod 4,060,081$.

  By typing `pow( 1945, 23, 4_060_081 )` into Sage, Bob learns that this is 74,116.

- Bob receives the encrypted ciphertext 2,013,961 from Alice. What must he compute to decrypt that?

  Bob must compute $2,013,961^{2{,}559{,}259} \bmod 4,056,187$.

  By typing `pow( 2_013_961, 2_559_259, 4_056_187 )` into Sage, Bob learns that this is 2330, or 11:30 PM.

- Alice wants to send the plaintext message 2315 to Bob, representing 11:15 PM. What must she compute to encrypt that?

  Alice must compute $2315^{19} \bmod 4,056,187$.

  By typing `pow( 2315, 19, 4_056_187 )` into Sage, Alice learns that this is 1,570,086.

- Alice receives the encrypted ciphertext 2,166,987 from Bob. What must she compute to decrypt that?

  Alice must compute $2,166,987^{1{,}939{,}851} \bmod 4,060,081$.

  By typing `pow( 2_166_987, 1_939_851, 4_060_081 )` into Sage, Alice learns that this is 1930, or 7:30 PM.

- Note: You might be interested to know that I gave these four subproblems on the final exam in Math-270: *Discrete Mathematics* in December of 2019.

Problem 10-6-11

Continuing with the previous two problems, I'd like to bring up an advanced point now. This problem is rather hard. So, don't be devastated if you can't get it, but please *give it the old college try*!

Suppose that Bob was sloppy when generating his key pair. More precisely, suppose that he didn't properly destroy φ. Perhaps he simply wrote it on a scrap of paper, and when he was done with it, he just crumpled up the scrap of paper and tossed it in a waste-paper basket. That doesn't really qualify as having "destroyed" φ, but perhaps he doesn't realize that or perhaps he doesn't realize how important it is to destroy φ.

Now let's suppose that Alice really wants Bob's d, and that when he isn't looking, she finds this scrap of paper in his waste-paper basket, and learns φ. How can Alice use this information to get Bob's d?

The method of computing d is that we invert $e \bmod \varphi$. Alice now has φ, and e has always been public information. (Alice already knows $e \equiv 19$.) Therefore, Alice can ask Sage to compute `pow( 19, -1, 4052160 )`.

When Alice asks Sage the value of `pow( 19, -1, 4052160 )`, she learns it is 2,559,259.

As you can see, Alice now has access to Bob's d. This means that she can read his email. While we haven't discussed digital signatures yet, Alice can also use Bob's d to sign documents, such as checks, as if she were Bob.

The Proof of Correctness of RSA

Typically, a cryptosystem is published with a proof of correctness, and a proof of security. For RSA, there are two proofs of correctness. (Permit me to mention that by "proof of correctness," we mean a proof that shows that when the receiver decrypts the ciphertext, the result is the same as the original plaintext that the sender encrypted.) Here, I'm presenting the short proof, which is actually incorrect. However, this is the proof that most textbooks contain. Later, we will examine the longer proof.

We know, from the encryption procedure, that $c \equiv m^e \bmod N$. We also know, from the decryption procedure, that $m' \equiv c^d \bmod N$. (We seek to prove that $m' \equiv m \bmod N$.)

Observe,

$$m' \equiv c^d \equiv (m^e)^d \equiv m^{ed} \bmod N$$

Now, we should apply what I had called "the upstairs-downstairs" principle, formally called the "Corollary to Euler's Totient Theorem," and reduce the exponent $ed \bmod \Phi(N)$. However, we know from the key-generation procedure that e and d are inverses mod $\varphi = \Phi(N)$. This means that $ed \equiv 1 \bmod \Phi(N)$, so we know that

$$m^{ed} \equiv m^1 \equiv m \bmod N$$

Thus we have proven that $m \equiv m' \bmod N$. Problem 10-7-7, in the next module, will ask you to identify the flaw in the reasoning here. Can you figure out what makes this proof incorrect?

Problem 10-6-12

This problem was inspired by Problem 6-9-1 in *Cryptography with Coding Theory*, 2nd edition, written by my former dissertation supervisor (Prof. Lawrence Washington) and his student, Prof. Wade Trappe. It is a parody of the famous story of Paul Revere, and the "1 if by land, 2 if by sea" legend.

A lookout has been posted to keep an eye out for attacks by invading forces. He's going to report any invasion using RSA. These are the possible plaintext messages:

- 1 if by land
- 2 if by sea
- 3 if by airborne assault

- 4 if undersea (by submarine)
- 0 if by cyberattack!

Upon interception of the ciphertext, we want to be able to determine the plaintext *with the absolute minimum possible amount of work*, but with absolute certainty. As it turns out, only one proper RSA-style modular exponentiation, and then a modular squaring, will be required. How can this be done? You may assume that you have the lookout's public key.

- Observe that if the plaintext is 0 in RSA, then the ciphertext is 0.
- Observe that if the plaintext is 1 in RSA, then the ciphertext is 1.
- Therefore, if the ciphertext is 0 or 1, we know the plaintext. If not, then we know that the plaintext is 2, 3, or 4.
- Next, encrypt the plaintext message 2 with the lookout's public key, and call that ciphertext c_2.
- If c_2 equals the intercepted ciphertext, then we know the plaintext is 2.
- If we still don't know the plaintext, compute the square of c_2 mod N. Because

  $$c_2^2 \equiv (2^e)^2 \equiv 2^{2e} \equiv (2^2)^e \equiv 4^e$$

  we know that the value of c_2^2 mod N will be equal to the encryption of 4 with the lookout's public key.
- If the intercepted ciphertext does not equal either c_2 mod N or c_2^2 mod N, then by process of elimination, we are certain that the plaintext must be 3.
- As you can see, using only one proper RSA-style modular exponentiation (computing c_2), and one additional squaring mod N, we can be certain that we know the plaintext message.

Problem 10-6-13

As you might know, there are 500 stocks that make up the S&P500 index. Now suppose there is a mutual fund that is being spied upon by some Russian mafia organization. This mutual fund only buys/sells S&P500 stocks, and they have two traders: one in charge of buy orders, and one in charge of sell orders. The Russian mobsters want to spy on the mutual fund, to be able to invest their ill-gotten gains with the skills, but not the fees, of the fund managers. They're going to monitor the messages to the buy-order trader and the sell-order trader, and mimic the fund's moves. Let's assume that the message is only the ticker symbol of the stock being bought or sold. (For example `MMM` for 3M and `ADBE` for Adobe, or `IBM` for IBM.)

Unfortunately for the Russians, the messages are encrypted with RSA. Moreover, the mutual fund is using huge primes and following the protocol precisely. With the absolute least amount of time possible, the Russians want to be able to decrypt any message that the fund sends to the traders. What is their best plan?

The Russians should take all possible plaintexts (the list of 500 possible ticker symbols) and encrypt them using the public keys of the two traders—after all, those keys are public. That's only 1000 modular exponentiations, which is definitely not a big deal. After this is done, they have a set of plaintext-ciphertext pairs, which can be thought of as a dictionary. If they sort these pairs by ciphertext, then whenever an encrypted message is intercepted, they can look up the corresponding plaintext in that dictionary. They could then, almost immediately, execute the same order on the market that the fund's traders executed.

Another way of imagining this situation is that there is one trader, and the orders are things like "`buy MMM`," "`sell ADBE`," "`buy IBM`," and so forth. Again, there are only 1000 possible plaintexts, so it is trivial to encrypt all of them with the target's public key. Even if we are worried about the capitalization of "buy" and "sell," the number of possible messages remains very modest. Therefore, using the public keys of the trader or traders, such a dictionary could easily be constructed.

It turns out that a technique called "padding" will resolve this problem, but we won't discuss padding just yet.

Problem 10-6-14

Let's suppose that Charlene has been asked out by Ned, Jed, Ted, and Fred. She needs to figure out who to say "yes" to, if anyone. To maintain some degree of privacy, she has asked the lads to generate and publish RSA public keys. Let's assume that the lads correctly generate different key material.

If Charlene's answer is positive, she'll reply with a date and time within the next 365 days. For example 201906031945 is an instruction to knock on her door at 1945 (7:45 PM) on the 3rd of June (month #6), 2019. She uses the special code 219902301111, which would otherwise represent 11:11 AM on the 30th of February, in the year 2199, to indicate a rejection.

- If Jed and Fred are to be sent the same plaintext message, then will they receive the same ciphertext message?

  No. Jed's ciphertext will be encrypted using Jed's public key, and Fred's ciphertext will be encrypted using Fred's public key. Since we assume that these keys are different, the ciphertexts will "almost surely" be different, even though the plaintexts are the same.

  It is actually possible, if Jed has e_1, N_1 as his public key, and Fred has e_2, N_2 as his public key, that

$$219,902,301,111^{e_1} \bmod N_1 \text{ and } 219,902,301,111^{e_2} \bmod N_2$$

are the same numerically, but that would be really improbable.

- Let's suppose Jed has been rejected. He is curious if his rivals (Ted, Fred, and Ned) have also been rejected or not. He has a copy of the ciphertexts that were sent to his rivals. He is only moderately curious, and he therefore does not want to do a lot of computations. He does have access to SageMathCell. Is there an extremely small amount of work that he could do, such as perhaps requiring only 5 or fewer Sage commands, that could satisfy his curiosity?

  Using one modular exponentiation per rival, Jed can encrypt the "rejection code" of 219902301111 using his rival's public key. After all, the public keys are public! If Ted, Fred, or Ned has a public key of e_3, N_3, then Jed only needs to compute

  $$219902301111^{e_3} \bmod N_3$$

  and see if the resulting number matches the intercepted ciphertext. A match indicates that the rival was rejected, and a non-match indicates that the rival got himself a date with Charlene.

- Continuing with the previous sub-problem, and assuming that Charlene did not reject all four lads, is there a moderately small amount of work that Jed could do, such as perhaps requiring only 200 or fewer modular exponentiations, that could determine the date and time of Charlene's date(s)? You should assume that the lads have used cryptographically appropriate prime numbers, not tiny 100-digit primes which would result in an N that Sage can factor.

  No. When a request for a date is granted, the problem stated that it would be within the next 365 days. Because there are 365 possible days, and many possible start times for a date, there are too many possible plaintexts. Even if we were to restrict the date's start time to have only :00, :15, :30, and :45 for the minutes, and restrict the hours to one of perhaps six reasonable choices (5 PM, 6 PM, 7 PM, 8 PM, 9 PM, 10 PM), we'd still have $(365)(6)(4) = 8760$ possible choices. That's much more than 200.

  Of course, if we use Sage and lift the restriction of 200 or fewer modular exponentiations, we can do better. If we consider all 365 possible days, all 24 possible hours, and all 60 possible minutes, we have 525,600 possible start times. It is not at all a computationally infeasible task to check all those possible plaintexts—a modern laptop could do it in a few seconds or less.

- Is there an easy modification to Charlene's protocol that could give some degree of privacy to the rejection status of the lads? In other words, I'm asking for a very small modification to the situation that would make it impossible for anyone to determine if a lad has been rejected using only 10,000 or fewer Sage commands.

Numerous correct answers exist. Charlene could send a date in the past to indicate a rejection. There are an extremely large number of dates in the twenty centuries preceding the current one, and it would be tedious to iterate through all of them.

My suggestion would be that Charlene could choose a random year in the 22nd century. She could choose a random time of day. Together, this makes for

$$(100)(24)(60) = 144,000$$

possible rejection codes, even keeping the date fixed at February 30th. That more than meets the requirements of this sub-problem.

As I mentioned at the end of the previous problem, a technique called "padding" will resolve this privacy issue in a different way, but we won't discuss padding just yet.

Problem 10-6-15

Many textbooks for a course in number theory, or a course in discrete mathematics, will have the students carryout RSA as follows. Someone would take a message, such as `HELLO` and break it up into five plaintext messages, each of length one letter, and encrypt them using the RSA protocol. If someone were to take a message of even two or three sentences, even if someone were using huge primes and following the protocol precisely, then it would be very easy to recover the plaintexts. Why?

Note, some textbooks use the affine cipher's representation of the alphabet

$$A = 0, B = 1, C = 2, D = 3, E = 4, \ldots$$

and some use the standard `ASCII` codes, where you can use capital letters, lowercase letters, digits, punctuation marks, and a handful of math symbols. Of course, you already know that there are 26 letters in the ordinary English alphabet, or 27 if you want to include a space. You might also know that there are 95 printable ASCII characters.

Now, after taking a moment to review the previous paragraphs, tell me why this is totally insecure as a method of encryption.

In Problem 10-6-13 (on Page 21 of this module), we saw that we could easily encrypt all 1000 possible messages from that situation, and assemble a dictionary that would enable us to read the plaintexts with essentially zero effort. In this problem, there are 95 possible "plaintexts" using the printable ASCII characters, or 26–27 if using only the English alphabet, with or without a space. After encrypting all possible 95, 26, or 27 plaintext messages with the target's public key (and each message consisting of one character), we would have the required dictionary. From that point forward, we could decrypt any ciphertext that we intercept, using that dictionary. The number of possible plaintexts, 95, 26, or 27, is far too small and it is too easy to create a ciphertext-to-plaintext dictionary for anyone's public key.

The Real Lesson of the Previous Problem: This is Why we Never Encrypt English Words One Letter at a Time Using RSA

At liberal-arts colleges in the USA, it is common to have a “math for everyone” course that is only for students majoring in subjects extremely distant from mathematics. The literature, art, philosophy, and history majors would get a tiny taste of geometry, logic, graph theory, combinatorics, and perhaps modular arithmetic. In some sense, these courses are a tiny version of discrete mathematics, covering about 25% of the content of most, but not all, topics in discrete mathematics.

If modular arithmetic were chosen as one of the topics for such a course, then RSA seems like a natural choice of subtopic. However, nearly all the textbooks for that course, being written by non-cryptologists, demonstrate the RSA algorithm using the idiotic implementation of encrypting English-language messages one letter at time, as we saw in the previous problem. The resulting communications would be as transparent as a fishbowl—even if using huge primes.

This now brings us to the important matter of discussing how RSA is actually used in practice.

Critical Information: How RSA is used in Practice

In practice, RSA is actually used to exchange a third key, called a secret key. That secret key is used to encrypt bulk data, or an entire conversation, using something called a block cipher. It is important to note that the secret key and the private key are different from each other, and different from the public key.

Today, the block cipher that is used almost all the time is called the Advanced Encryption Standard (or AES). You might enjoy reading about the AES. Encrypted communications inside of Russia are an exception, where the laws require the use of the block cipher GOST (designed by the Soviet Union in 1989), sometimes called Magma, or the block cipher Kuznyechik (designed by the Russian Federation in 2015). However, except inside of Russia, the block cipher that is used in practice globally is the AES.

In previous decades, the Data Encryption Standard (DES), published by IBM and the US Government in 1977, and its successor, triple-DES (3DES), were also used both inside the USA and globally (but not inside the former Soviet Union, so far as I am aware). Another block cipher that is discussed at times is IDEA (the International Data Encryption Algorithm), published in Switzerland in 1991, and which saw some use in the 1990s, and the following decade.

Whenever you use your browser in “secure mode,” using `HTTPS` in place of `HTTP`, you are exchanging a secret key with the web server using either RSA or the Diffie-Hellman Key Exchange. The Diffie-Hellman Key Exchange is a completely different algorithm from RSA. However, like RSA, the Diffie-Hellman Key Exchange is based entirely on the mathematics of modular exponentiation. Sometimes the phrase “hybrid encryption” is used to describe the act of exchanging a block cipher’s secret key using an RSA public & private key pair.

You might wonder why we don’t use the AES and a secret key only. Why do we bother with RSA and the public & private keys? The basic answer is that any block cipher, including the AES, requires a shared secret between the users. Imagine if you want to do some eCommerce with a website using your laptop (or your smartphone). Your laptop (or smartphone) and the web-server would both need to know a secret key—a bit string of length 128, 192, or 256 bits, that nobody else in the world knows. That is very impractical to pre-arrange.

Imagine a world where everyone who wanted to potentially shop at your favorite online retailer would be forced to call that retailer on the phone, and get a sequence of 128, 192, or 256 bits dictated to them. The potential customer would have to enter that code into their computer, without any errors, in order to be permitted to shop at that website. Clearly, very few people would have done this in the early days of the World Wide Web, and e-Commerce would not have developed without some form of public-key cryptography having been invented.

I’m also entirely leaving out another crucial application of RSA, and that is the entire subject of digital signatures, an exciting topic in its own right.