Furthermore, $N = pq$, which implies

$$\phi(N) = (p-1)(q-1)$$

(see the Appendix if you are not familiar with the ϕ function). These two facts together imply that

$$ed - 1 = k\phi(N)$$

for some integer k. We don't need to know the precise value of k.

Now we have all of the necessary pieces of the puzzle to verify that RSA decryption works. Observe that

$$\begin{aligned} C^d &= M^{ed} = M^{(ed-1)+1} = M \cdot M^{ed-1} \\ &= M \cdot M^{k\phi(N)} = M \cdot 1^k = M \bmod N. \end{aligned} \tag{4.3}$$

In the first line of equation (4.3) we simply added zero to the exponent and in the second line we used Euler's Theorem to eliminate the ominous-looking $M^{\phi(N)}$ term. This confirms that the RSA decryption exponent does, in fact, decrypt the ciphertext C. Of course, the game was rigged since e and d were chosen so that Euler's Theorem would make everything come out as desired in the end. That's just the way mathematicians do things.

4.3.1 Textbook RSA Example

Let's consider a simple RSA example. To generate, say, Alice's keypair, we'll select the two "large" primes $p = 11$ and $q = 3$. Then the modulus is $N = pq = 33$ and $(p-1)(q-1) = 20$. Next, we choose the encryption exponent $e = 3$, which is, as required, relatively prime to $(p-1)(q-1)$. We then compute the corresponding decryption exponent, which in this case is $d = 7$, since $ed = 3 \cdot 7 = 1 \bmod 20$. Now, we have

$$\text{Alice's public key: } (N, e) = (33, 3)$$

and

$$\text{Alice's private key: } d = 7.$$

As usual, Alice's public key is public but only Alice has access to her private key.

Now suppose Bob wants to send Alice a message M. Further, suppose that as a number, the message is $M = 15$. Bob looks up Alice's public key $(N, e) = (33, 3)$ and computes the ciphertext as

$$C = M^e \bmod N = 15^3 = 3375 = 9 \bmod 33,$$

which he then sends to Alice.

To decrypt the ciphertext $C = 9$, Alice uses her private key $d = 7$ to find

$$M = C^d \bmod N = 9^7 = 4{,}782{,}969 = 144{,}938 \cdot 33 + 15 = 15 \bmod 33.$$

Alice has thereby recovered the original message $M = 15$ from the ciphertext $C = 9$.

There are a couple of major problems with this textbook RSA example. For one, the "large" primes are not large—it would be trivial for Trudy to factor the modulus. In the real world, the modulus N is typically at least 1024 bits, with a 2048-bit or large modulus often used.

An equally serious problem with most textbook RSA examples (ours included) is that they are subject to a forward search attack, as discussed in Chapter 2. Recall that in a forward search, Trudy can guess a possible plaintext message M and encrypt it with the public key. If the result matches the ciphertext C, then Trudy has recovered the plaintext M. The way to prevent this attack (and several others) is to pad the message with random bits. For simplicity, we do not discuss padding here, but it is worth noting that several padding schemes are in common use, including the oddly named PKCS#1v1.5 [91] and Optimal Asymmetric Encryption Padding (OAEP) [226]. Any real-world RSA implementation must use a padding scheme such as one of these.

4.3.2 Repeated Squaring

Modular exponentiation of large numbers with large exponents is an expensive proposition. To make this more manageable (and thereby make RSA more efficient and practical), several tricks are commonly used. The most basic trick is the method of *repeated squaring* (also known as *square and multiply*).

For example, suppose we want to compute 5^{20}. Naïvely, we would simply multiply 5 by itself 20 times and then reduce the result modulo 35, that is,

$$5^{20} = 95{,}367{,}431{,}640{,}625 = 25 \bmod 35. \tag{4.4}$$

However, this method results in an enormous value prior to the modular reduction, in spite of the fact that the final answer is restricted to the range 0 to 34.

Now suppose we want to do RSA encryption $C = M^e \bmod N$ or decryption $M = C^d \bmod N$. In a secure implementation of RSA, the modulus N is at least 1024 bits. As a result, for typical values of e or d, the numbers involved will be so large that it is impossible to compute $M^e \bmod N$ by the naïve approach in equation (4.4). Fortunately, the method of repeated squaring allows us to compute such an exponentiation without creating unmanageably large numbers at any intermediate step.