Notes on RSA Encryption

Paul Zeitz

October 21, 2022

Contents

1 Overview 1

2 Encryption Using Prime n 2
  2.1 A Small Prime n . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  2.2 A Problem From the 2003 Bay Area Math Meet . . . . . . . . . . . . . . . . 3
  2.3 The General Case for $n = p$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

3 Encryption Using $n = pq$ 6
  3.1 An Example with Small Primes . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  3.2 What if $p, q$ are HUGE? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  3.3 Computational Issues . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  3.3 Exercises . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

1 Overview

Please note the following conventions: all Roman-alphabet variables ($a, b, c, \text{ etc.}$) are integers, and $p$ and $q$ denote distinct primes.

The RSA$^1$ encryption method is a type of **public-key cryptography**, which has the following amazing property: The method of **encoding** (putting a message into code) is known to everyone, but nevertheless it is virtually impossible to **decode** the message (restore it from coded form back to the original text).

The way RSA encryption is done is quite simple:

1. A modulus $n$ is chosen, known to all. Also, an **encoding exponent** $e$ is chosen, also known to all. Typically, $n$ is very large (greater than $10^{100}$).

2. The message that is to be encoded is converted from letters to numbers, in a standard way, known to all. If needed, the message is broken down into blocks so that each block is an integer smaller than $n$.

---

$^1$RSA stands for Rivest-Shamir-Adelman, the names of the inventors of the algorithm.
3. Each block $x$ is encoded by raising it to the power $e$ modulo $n$. Thus the encoded block is the number

$$y \equiv x^e \pmod{n},$$

where it is understood that $y$ is a positive integer less than $n$.

4. The encoded block $y$ can be decoded by raising it to the decoding exponent $d$. This decoding exponent is not public; it is only known to Headquarters. This exponent $d$ is cleverly chosen so that

$$y^d \equiv (x^e)^d \equiv x \pmod{n},$$

and thus raising the encoded block to this exponent magically recovers the original number block $x$.

5. It is now a simple matter to convert the number into letters, and we are done.

There are two important things to understand: first, how to find the magical decoder $d$, when we are given $n$ and $e$; and second, why this can be virtually impossible to do under certain circumstances, unless one knows the prime factorization of $n$.

## 2 Encryption Using Prime $n$

Let’s practice with a simple example using a very small value of $n$.

### 2.1 A Small Prime $n$

Suppose that $n = 17$ and $e = 13$. To encode the number $x = 11$, we compute $11^{13} \pmod{17}$. This can be done by hand quite easily.

\begin{align*}
11 &\equiv -6 \pmod{17} \\
\implies 11^2 &\equiv (-6)^2 = 36 \equiv 2 \pmod{17} \\
\implies 11^{12} &\equiv 2^6 = 64 \equiv -4 \pmod{17} \\
\implies 11^{13} &\equiv -4 \cdot 11 = -44 \equiv 7 \pmod{17}.
\end{align*}

(Alternatively, this could be done with Sage.)

Thus the encoded number $y = 7$. In this case, the magical decoding exponent $d = 5$. Let’s check this: We wish to verify that $y^5 \pmod{17}$ is equal to $x$, or 11. Once again, let’s do it by hand, for practice.

\begin{align*}
y^2 &\equiv 7^2 \equiv 49 \equiv -2 \pmod{17} \\
\implies y^4 &\equiv (-2)^2 = 4 \pmod{17} \\
\implies y^5 &\equiv 7 \cdot 4 = 28 \equiv 11 \pmod{17},
\end{align*}

and thus $y^5 \equiv x \pmod{17}$. 
2.2 A Problem From the 2003 Bay Area Math Meet

A fairly difficult question on the written exam (answered correctly, nevertheless, by 15% of the students!) asked for the smallest positive integer \( x \) such that \( x^{801} \) had a remainder of 10 when divided by 2003. In other words, find \( x \), given that

\[
y \equiv x^{801} \equiv 10 \pmod{2003}.
\]

In this case, \( n = 2003, e = 801 \). It turns out that the decoding exponent \( d \) is equal to 5.

Thus,

\[
x \equiv y^5 \equiv 10^5 = 100000 \pmod{2003}.
\]

It is easy to reduce this to a positive integer less than 2003:

\[
100000 = 2000 \cdot 50 \equiv -3 \cdot 50 = -150 \equiv 2003 - 150 = 1853 \pmod{2003}.
\]

It is nearly impossible to verify, by hand, that this is correct, since 1853

\[
x^{801}
\]

is equal to the 2,618-digit number

\[
209 303 457 619 747 486 725 304 027 741 279 521 121 158 509 960 237 434 506 042 762
141 442 113 674 405 894 801 185 432 988 343 807 394 974 373 264 032 617 903 916 398
343 367 325 236 605 152 673 336 791 113 948 192 984 462 353 615 033 880 466 888 060
333 605 952 190 969 514 431 608 524 480 765 256 062 930 612 631 519 511 331 501 529
634 852 418 817 082 249 853 380 494 599 385 295 662 992 768 960 389 195 759 159 545
424 752 063 188 763 993 284 913 403 291 295 385 564 865 535 662 562 060 864 503 119
388 692 835 167 389 586 216 433 157 607 652 490 219 986 139 595 225 844 276 863 683
907 613 127 414 007 570 076 323 039 330 820 280 888 881 126 817 504 292 772 804 432
899 248 871 554 500 865 425 470 648 957 000 582 476 596 216 545 006 016 029 507 741
320 273 205 687 811 587 126 794 021 758 817 867 374 390 251 347 530 921 789 220 437
616 409 649 682 992 055 020 073 300 268 566 609 853.

We now need to divide this monster by 2003, and check to see that the remainder indeed is 10.

Of course, this can be verified quickly using Sage. When we ask it to compute \( \text{mod}(1853^{801}, 2003) \) it outputs 10 in a blink of an eye. But the interesting question is, how did the BAMM contestants deduce the decoding exponent was 5 without using a computer?

Because 2003 is a prime, we can use Fermat’s Little Theorem, which says that
\[
a^{2002} \equiv 1 \pmod{2003},
\]
for all \( a \) that are not multiples of 2003. Consequently, if \( a \) is not a multiple of 2003, we have
\[
a^{4004} = a^{2002 \cdot 2} = (a^{2002})^2 \equiv 1^2 = 1 \pmod{2003},
\]
and thus
\[
a^{4005} = a \cdot a^{4004} \equiv a \cdot 1 = a \pmod{2003}.
\]
The reason we are interested in the exponent 4005 is because 4005 = 801 · 5. Hence for any \( a \) which is not a multiple of 2003,
\[
(a^{801})^5 = a^{4005} \equiv a \pmod{2003}.
\]
This verifies that 5 is the decoder exponent corresponding to the encoder exponent 801. If you raise something to the 801st power, and then raise that to the 5th, you get back to where you started \( \pmod{2003} \).

### 2.3 The General Case for \( n = p \)

Let’s solve the problem in general for \( n = p \), a prime. Suppose we are given \( y \equiv x^e \pmod{p} \). How can we find the decoder exponent \( d \) that recovers \( x \)? By Fermat’s Little Theorem, as long as \( x \not\equiv 0 \pmod{p} \), then
\[
x^{p-1} \equiv 1 \pmod{p}.
\]
Thus, for any positive integer \( t \),
\[
(x^{p-1})^t \equiv 1 \pmod{p}.
\]

Multiplying by \( x \), we get
\[
x(x^{p-1})^t \equiv x \pmod{p},
\]
which is equivalent to
\[
x^{(p-1)t+1} \equiv x \pmod{p},
\]
for any positive integer \( t \). Since we are given \( y \equiv x^e \pmod{p} \), and want to raise \( y \) to an exponent \( d \) that gives us \( x \), we need to find \( d \) so that
\[
ed \text{ is equal to } (p-1)t + 1 \text{ for some integer } t.
\]

If we can accomplish this, we will have
\[
y^d \equiv (x^e)^d = x^{ed} = x^{(p-1)t+1} \equiv x \pmod{p}.
\]
But this is easy to do. Let’s try some examples.

(a) \( p = 17, e = 13 \). This was the first example. We need to find \( d \) so that \( 13d = 16t + 1 \).

By inspection, it is easy to guess \( d = 5 \).

(b) \( p = 2003, e = 801 \). This is the BAMM problem. We need to find \( d \) so that \( 801d = 2002t + 1 \) for some \( t \). Again, it is easy to guess \( d = 5 \).

(c) \( p = 79, e = 19 \). We must find \( d \) such that \( 19d = 78t + 1 \). We can perform the Euclidean Algorithm, and then backtrack:

\[
\begin{align*}
78 &= 4 \cdot 19 + 2 \\
19 &= 9 \cdot 2 + 1.
\end{align*}
\]

Hence
\[
\begin{align*}
1 &= 19 - 9 \cdot 2 \\
&= 19 - 9(78 - 4 \cdot 19) \\
&= 37 \cdot 19 - 9 \cdot 78,
\end{align*}
\]
and \( d = 37 \). In other words, if \( y \equiv x^{19} \pmod{79} \), then \( x \equiv y^{37} \pmod{79} \).

The condition (2.1) that \( ed \) is equal to \( (p-1)t + 1 \) for some integer \( t \) can be formulated more cleanly. This is the same as saying that \( ed \) must be congruent to 1 modulo \( (p-1) \). In other words,
If 
\[ y = x^e \pmod{p}, \]
then 
\[ x \equiv y^d \pmod{p}, \]
where \( d \) is chosen so that 
\[ ed \equiv 1 \pmod{p - 1}. \] (2.2)

The only problem with the method used above is that it is too easy to find \( d \). If you know \( p \) and \( e \), then you can always solve (2.2). After all, you are merely computing the multiplicative inverse of \( e \) modulo \( p - 1 \). As long as \( e \) and \( p - 1 \) are relatively prime, this can be computed quickly with the Euclidean Algorithm, or even faster, using the computer. The \( \text{mod} \) command works beautifully, using the exponent \(-1\). Sage is smart enough to understand that. For example, \( \text{mod}(19^{-1}, 78) \) will output 37 in a flash, since

\[ 19 \cdot 37 \equiv 1 \pmod{78} \]
is the same as saying

\[ 19^{-1} \equiv 37 \pmod{78}. \]

But we do not need to restrict ourselves to the \( n = p \) case. In place of Fermat’s Little Theorem, we will use Euler’s Extension of Fermat’s Little Theorem, which says that if \( x \perp n \), then \( x^{\phi(n)} \equiv 1 \pmod{n} \).

### 3 Encryption Using \( n = pq \)

As we mentioned at the end of Section 2, encryption using \( n = p \) fails, since it is easy to compute the decoder \( d \)—it is just the multiplicative inverse of the encoder \( e \), modulo \( p - 1 \). But what if \( n = pq \), where \( p \) and \( q \) are distinct primes?

Suppose that \( y \equiv x^e \pmod{n} \). By Euler’s Theorem, \( x^{\phi(n)} \equiv 1 \pmod{n} \), and, rehashing the ideas in Section 2, we have, for any integer \( t \),

\[ (x^{\phi(n)})^t \equiv 1 \pmod{n}, \]

so

\[ x^{t\phi(n)+1} \equiv x \pmod{n}. \]

Thus, if we can find \( d \) such that

\[ ed = t\phi(n) + 1 \] (3.3)

for some integer \( t \), we will be done, for then

\[ y^d \equiv (x^e)^d = x^{ed} = x^{t\phi(n)+1} \equiv x \pmod{n}. \]

But (3.3) is equivalent to saying that

\[ ed \equiv 1 \pmod{\phi(n)}, \]
in other words,
The decoder $d$ is the multiplicative inverse of $e$ modulo $\phi(n)$.

3.1 An Example with Small Primes

Suppose $n = 77, e = 17$. We are given $y \equiv x^e \pmod{n}$, and wish to find $x$. Since $n = 7 \cdot 11$, we compute $\phi(77) = 6 \cdot 10 = 60$. We need to find $d$ so that

$$17d \equiv 1 \pmod{60}.$$

We shall find this using the Euclidean Algorithm. We have

$$
\begin{align*}
60 &= 3 \cdot 17 + 9 \\
17 &= 1 \cdot 9 + 8 \\
9 &= 1 \cdot 8 + 1,
\end{align*}
$$

and thus

$$1 = 9 - 8 = 9 - (17 - 9) = 2 \cdot 9 - 17 = 2 \cdot (60 - 3 \cdot 17) - 17 = 2 \cdot 60 - 7 \cdot 17.$$

Thus

$$7 \cdot 17 = -1 + 2 \cdot 60,$$

so $7 \cdot 17 \equiv -1 \pmod{60}$. Thus

$$-7 \cdot 17 \equiv +1 \pmod{60},$$

so the multiplicative inverse of 17 modulo 60 is equal to $-7$, or 53. So 53 is our magical decoding exponent.

While the above Euclidean Algorithm work was pretty tedious, in practice the Sage command $\text{mod}(17^{-1}, 60)$ would yield the answer 53 instantly. An alternative method is to use the $\text{xgcd}$ command, the “extended greatest common divisor” command. For example, the output of $\text{xgcd}(17, 60)$ is the triple $(1, -7, 2)$, which says that the greatest common divisor of 17 and 60 is 1, and that $(-7)(17) + 2(60) = 1$. Hence the multiplicative inverse of 17 $\pmod{60}$ is $-7$, and converted to a positive number, this is 53.

3.2 What if $p, q$ are HUGE?

Suppose that $n = pq$, with $p, q$ both very large, on the order of $10^{100}$, and suppose that this factorization is not known to the general public. Then even if $e$ is known to the public, there is no easy way to find $d$, for $d$ is the multiplicative inverse of $e$ modulo $\phi(n)$, but

You need to know $p$ and $q$ in order to find $\phi(n)$!
And there is no polynomial-time way known, at the present time, to factor a large number; there is not much that is better than simple division-by-odd-numbers-less-than-the-square-root. And for all practical purposes, this takes forever.

That’s why RSA is such a great idea; truly a world-changing application of mathematics. For the first time in history, it is possible for just about anyone to create an unbreakable code that is easy to use. All you need is a good supply of super-big primes. And they are, in general, not too hard to find. But that is another story.

Computational Issues

In general, both $n$ and $e$ are gigantic numbers (it is easy to pick $e$ to be a large prime between $p$ and $q$, which then guarantees that $e \perp \phi(n)$ (WHY?), and the encoding procedure will take a number $x < n$ and raise it to the $e$ power modulo $n$. This seems computationally "expensive," but it is not. Here’s a simple concrete example with very small numbers. Suppose you wanted to compute $19^{99} \pmod{n}$. Instead of performing the one exponentiation and then finding the remainder, we first compute $19^2 \pmod{n}$. Call this number $r$, and note that it is smaller than $n$. Now compute $r^2 \pmod{n}$. Then square that, etc. In other words, we compute $19^2, 19^4, 19^8, 19^{16}, \ldots$. We only need to do this a few more times to get close to the target exponent of 99. Then we consider 99 in base two and multiply the remainders that we have computed. Specifically, we have

$$99 = 64 + 32 + 2 + 1,$$

and hence

$$19^{99} = 19^{64}19^{32}19^219^1.$$

In general, to compute $a^e \pmod{n}$, we need only first compute $a, a^2, (a^2)^2 = a^4, \ldots \pmod{n}$ and we only need at most $\log_2 e$ computations. Then to find the precise remainder, we must multiply some of these (depending on $e$ in base-2), with at most $\log_2 e$ multiplications.

So it is easy to raise a 100-digit number to a 100-digit exponent and find the remainder modulo a 100-digit number. Try it!

The other computational issue is finding the magic decoding exponent $d$. But this requires solving the equation $ed \equiv 1 \pmod{\phi(n)}$. If you do this by hand, you need to do the Euclidean Algorithm and then work it out in reverse. This also takes very little computational effort. In general, the number of steps is “logarithmic.” WHY?

In sum the only hard computation—effectively impossible until quantum computing or other mathematical breakthroughs occur—is factoring $n$ if it is a very large number.
3.3 Exercises

I am providing the answers to some of these so that you can test yourself.

1 For the following values of \( n \) and \( e \), find the magic decoding exponent \( d \) by hand (feel free to use a calculator, and you can check your work with the computer.)

   (a) \( n = 17, e = 5 \). (Ans: \( d = 13 \))
   (b) \( n = 21, e = 11 \). (Ans: \( d = 11 \))
   (c) \( n = 391, e = 19 \). (Ans: \( d = 315 \))

2 Use the computer to find the decoder for \( n = 62884891, e = 7937 \). Use the computer to check that your decoder actually works, by encoding a value \( x < n \), and then decoding it. (Ans: \( d = 14859809 \)).

3 What is wrong (if anything), with just letting \( n \) equal a prime, instead of a product of two distinct primes? Is the RSA method still effective?

4 (Challenging). The RSA method finds a decoder exponent \( d \) such that \( ed \equiv 1 \pmod{\phi(n)} \) and uses Euler’s extension of Fermat’s Little Theorem, namely that if \( x \perp n \), then \( x^{\phi(n)} \equiv 1 \pmod{n} \). But what if \( x \) is NOT relatively prime to \( n \)? Since \( n = pq \), this would only happen if \( x \) is a multiple of \( p \) or a multiple of \( q \) (it cannot be a multiple of both, since \( x \) has to be smaller than \( n \)). Show that in this case, that even though it is NOT the case that \( x^{\phi(n)} \equiv 1 \pmod{n} \), we will still have \( x^{ed} \equiv x \pmod{n} \).