8.11 RSA Public Key Encryption

Turing's code did not work as he hoped. However, his essential idea—using number theory as the basis for cryptography—succeeded spectacularly in the decades after his death.

In 1977, Ronald Rivest, Adi Shamir, and Leonard Adleman at MIT proposed a highly secure cryptosystem, called **RSA**, based on number theory. The purpose of the RSA scheme is to transmit secret messages over public communication channels. As with Turing's codes, the messages transmitted are nonnegative integers of some fixed size.

Moreover, RSA has a major advantage over traditional codes: the sender and receiver of an encrypted message need not meet beforehand to agree on a secret key. Rather, the receiver has both a *private key*, which they guard closely, and a *public key*, which they distribute as widely as possible. A sender wishing to transmit a secret message to the receiver encrypts their message using the receiver's widely-distributed public key. The receiver can then decrypt the received message using their closely held private key. The use of such a *public key cryptography* system allows you and Amazon, for example, to engage in a secure transaction without meeting up beforehand in a dark alley to exchange a key.

Interestingly, RSA does not operate modulo a prime, as Turing's hypothetical Version 2.0 may have, but rather modulo the product of *two* large primes—typically primes that are hundreds of digits long. Also, instead of encrypting by multiplication with a secret key, RSA exponentiates to a secret power—which is why Euler's Theorem is central to understanding RSA.

The scheme for RSA public key encryption appears in the box.

If the message m is relatively prime to n, then a simple application of Euler's Theorem implies that this way of decoding the encrypted message indeed reproduces the original unencrypted message. In fact, the decoding always works—even in (the highly unlikely) case that m is not relatively prime to n. The details are worked out in Problem 8.81.

The RSA Cryptosystem

A **Receiver** who wants to be able to receive secret numerical messages creates a *private key*, which they keep secret, and a *public key*, which they make publicly available. Anyone with the public key can then be a **Sender** who can publicly send secret messages to the **Receiver**—even if they have never communicated or shared any information besides the public key.
Here is how they do it:

Beforehand The **Receiver** creates a public key and a private key as follows.

1. Generate two distinct primes, p and q. These are used to generate the private key, and they must be kept hidden. (In current practice, p and q are chosen to be hundreds of digits long.)
2. Let $n ::= pq$.
3. Select an integer $e \in [0..n)$ such that $\gcd(e, (p-1)(q-1)) = 1$. The *public key* is the pair (e, n). This should be distributed widely.
4. Let the *private key* $d \in [0..n)$ be the inverse of e in the ring $\mathbb{Z}_{(p-1)(q-1)}$. This private key can be found using the Pulverizer. The private key d should be kept hidden!

Encoding To transmit a message $m \in [0..n)$ to **Receiver**, a **Sender** uses the public key to encrypt m into a numerical message

$$\widehat{m} ::= m^e \ (\mathbb{Z}_n).$$

The **Sender** can then publicly transmit $\widehat{m}$ to the **Receiver**.

Decoding The **Receiver** decrypts message $\widehat{m}$ back to message m using the private key:

$$m = \widehat{m}^d \ (\mathbb{Z}_n).$$

Why is RSA thought to be secure? It would be easy to figure out the private key d if you knew p and q—you could do it the same way the **Receiver** does using the Pulverizer. But assuming the conjecture that it is hopelessly hard to factor a number that is the product of two primes with hundreds of digits, an effort to factor n is not going to break RSA.

Could there be another approach to reverse engineer the private key d from the public key that did not involve factoring n? Not really. It turns out that given just the private and the public keys, it is easy to factor n[15] (a proof of this is sketched in Problem 8.83). So if we are confident that factoring is hopelessly hard, then we can be equally confident that finding the private key just from the public key will be hopeless.

But even if we are confident that an RSA private key won't be found, this doesn't rule out the possibility of decoding RSA messages in a way that sidesteps the private key. It is an important unproven conjecture in cryptography that *any* way of cracking RSA—not just by finding the secret key—would imply the ability to factor. This would be a much stronger theoretical assurance of RSA security than is presently known.

But the real reason for confidence is that RSA has withstood all attacks by the world's most sophisticated cryptographers for nearly 40 years. Despite decades of these attacks, no significant weakness has been found. That's why the mathematical, financial, and intelligence communities are betting the family jewels on the security of RSA encryption.

You can hope that with more studying of number theory, you will be the first to figure out how to do factoring quickly and, among other things, break RSA. But be further warned that even Gauss worked on factoring for years without a lot to show for his efforts—and if you do figure it out, you might wind up meeting some humorless fellows working for a Federal agency in charge of security....

8.12 What has SAT got to do with it?

So why does society, or at least everybody's secret codes, fall apart if there is an efficient test for satisfiability (SAT), as we claimed in Section 3.5? To explain this, remember that RSA can be managed computationally because multiplication of two primes is fast, but factoring a product of two primes seems to be overwhelmingly demanding.

[15] In practice, for this reason, the public and private keys should be randomly chosen so that neither is "too small."

Let's begin with the observation from Section 3.2 that a digital circuit can be described by a bunch of propositional formulas of about the same total size as the circuit. So testing circuits for satisfiability is equivalent to the SAT problem for propositional formulas (see Problem 3.18).

Now designing digital multiplication circuits is completely routine. We can easily build a digital "product checker" circuit out of AND, OR, and NOT gates with 1 output wire and $4n$ digital input wires. The first n inputs are for the binary representation of an integer i, the next n inputs for the binary representation of an integer j, and the remaining $2n$ inputs for the binary representation of an integer k. The output of the circuit is 1 iff $ij = k$ and $i, j > 1$. A straightforward design for such a product checker uses proportional to n^2 gates.

Now here's how to factor any number m with a length $2n$ binary representation using a SAT solver. First, fix the last $2n$ digital inputs—the ones for the binary representation of k—so that k equals m.

Next, set the first of the n digital inputs for the representation of i to be 1. Do a SAT test to see if there is a satisfying assignment of values for the remaining $2n-1$ inputs used for the i and j representations. That is, see if the remaining inputs for i and j can be filled in to cause the circuit to give output 1. If there is such an assignment, fix the first i-input to be 1, otherwise fix it to be 0. So now we have set the first i-input equal to the first digit of the binary representations of an i such that $ij = m$.

Now do the same thing to fix the second of the n digital inputs for the representation of i, and then third, proceeding in this way through all the n inputs for the number i. At this point, we have the complete n-bit binary representation of an $i > 1$ such $ij = m$ for some $j > 1$. In other words, we have found an integer i that is a factor of m. We can now find j by dividing m by i.

So after n SAT tests, we have factored m. This means that if SAT for digital circuits with $4n$ inputs and about n^2 gates could be determined by a procedure taking a number of steps bounded above by a degree d polynomial in n, then $2n$ digit numbers can be factored in n times this many steps, that is, with a number of steps bounded by a polynomial of degree $d+1$ in n. So if SAT could be solved in polynomial time, then so could factoring, and consequently RSA would be "easy" to break.

6.042J / 18.062J Mathematics for Computer Science
Spring 2015