Numbers: RSA[1]

- **Recap.** Let us recap some facts we will be needing for today's class.

  a. *(Modular Exponentiation) For any three positive numbers a, b, n, we can efficiently compute $a^b \bmod n$ using* ModExp.

  b. *(Bezout) For any two positive numbers a, b, we can efficiently compute integers x, y such that $xa + yb = \gcd(a, b)$ using* ExtGCD.

  c. *In particular, if $\gcd(a, n) = 1$, we can efficiently compute integers x, y such that $xa + yb = 1$.*

  d. *(Multiplicative Inverse) Therefore, if $\gcd(a, n) = 1$, we can efficiently compute a^{-1} modulo n; the number b such that $ab \equiv_n 1$. We do this by taking $x \bmod n$ for the x in the above bullet point.*

  e. *(Fermat's Little Theorem.) If p is a prime and $\gcd(a, p) = 1$, then $a^{p-1} \equiv_p 1$.*

  f. *(Problem Set 8, Problem 2(a).) If $p|a$ and $q|a$ where p and q are distinct primes, then $pq|a$.*

- **Cryptography.** Alice wants to send a *message* m to Bob. Unfortunately, the channel in which Alice is speaking to Bob is completely transparent and can be plainly read. So, she wants to instead send a *cipher* c such that (a) upon receiving c, Bob can figure out m, but (b) any one else, say Eve, upon receiving c can't obtain *any information* about m.

  As can be seen, some asymmetry is *required* between Bob and Eve. The "traditional" way of achieving this is that Alice and Bob pre-decide on some information called a *key* and use it to figure out c from m. This *key* is something that only Alice and Bob know; in particular, the eavesdropper Eve doesn't.

  For instance, the key could be some long integer k of the same length as m, and Alice can *encrypt* m to get *cipher* c by letting $c_i = (m_i + k_i) \mod 10$ for every digit i. Note that Bob can easily *decrypt* since he has the key k: he does the opposite action of $(c_i - k_i) \mod 10$. Also note that Eve can have no idea what m was by just looking at c since k can be an arbitrary key.

  One issue with the above *protocol* is that Alice and Bob need to agree upon the *key* beforehand. It can be shown that if the same key is used repeatedly, then Eve can actually figure out the key (especially if she can impersonate as Alice). So, keys need to be constantly generated and shared; but then if Alice and Bob can share keys secretly often, why not just use that time to swap the messages?

- **Public Key Cryptography (PKC).** This is a *fantastic* idea which gets over the key sharing business.

  In this every person who wishes to receive a message (say Bob, or any website who needs credit card info) generates *two keys*. One key is the *public key pk* which they announce to the world. The other is the *secret key sk* which they guard with their lives. To summarize, the key they generate is a tuple (pk, sk); pk they tell everyone, and sk they tell no one (including Alice).

  A PKC protocol consists of two functions/algorithms Enc and Dec. Both of these are also *public*; the code is also published by Bob.

[1] *Lecture notes by Deeparnab Chakrabarty. Last modified : 28th Aug, 2021*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Now, if Alice wants to send a message to Bob, she can *encrypt* a message m using the public key to get

$$\mathsf{Enc}(m, pk) \mapsto c$$

She then sends c across to Bob. Note that Eve knows c and knows pk and also knows the algorithm Enc. *She still shouldn't have any clue what m is.* In other words, it shouldn't be easy for Eve to *invert* this function Enc.

Bob, upon receiving the cipher c, then uses the *decryption* algorithm Dec to get the message back. This decryption algorithm will use *both* keys.

$$\mathsf{Dec}(c, pk, sk) \mapsto m$$

- **The RSA Algorithm.** *These lecture notes are "direct" and do not mimic the lecture I gave where I tried to run through the idea of "how one could come up with RSA?". Whichever works.*

  a. **Key Generation.**

  - Bob picks two primes p and q; these will be *large, distinct* primes.
  - Let $N := pq$ and let $\phi := (p-1)(q-1)$.
  - Bob picks another number e such that $\gcd(e, \phi) = 1$.
  - Bob computes the *multiplicative inverse* of e modulo ϕ. Call it d.
  - Bob's *public key* is (e, N).
  - Bob's *secret key* is d.

```
procedure RSAGENKEYS($p, q, e$) ▷ Assumes $p, q$ are primes, and $\gcd(e, (p-1)(q-1)) = 1$.
    ▷ Returns tuple $(\mathsf{pk}, \mathsf{sk})$ of public key, secret key.
    $N \leftarrow pq$.
    $\phi \leftarrow (p-1)(q-1)$.
    $d \leftarrow$MULTINVERSE($e, \phi$)
    $\mathsf{pk} \leftarrow (e, N)$.
    $\mathsf{sk} \leftarrow d$.
    return $(\mathsf{pk}, \mathsf{sk})$.
```

  b. The Encryption algorithm is as follows.

  - Suppose Alice wants to send m to Bob. We assume $m \in \{1, 2, \ldots, N-1\}$; otherwise, Alice needs to break her message into pieces.
  - Alice's cipher $c = m^e \bmod N$; she evaluates this using Bob's public key (e, N) and uses modular exponentiation.

```
procedure RSAENC($m, \mathsf{pk}$) ▷ Assumes $\mathsf{pk} = (e, N)$
    ▷ Returns cipher $c$.
    $c \leftarrow$ MODEXP($m, e, N$).
    return $c$.
```

c. The Decryption algorithm is as follows.

   - Upon receiving c, Bob recovers the message m using his secret key d by computing $c^d \bmod N$.

```
procedure RSADec($c$, pk, sk)    ▷ Assumes pk $= (e, N)$ and sk $= d$
    ▷ Returns message $m$.
    $m \leftarrow$ ModExp$(c, d, N)$.
    return $m$.
```

- **RSA example.** Suppose Bob selects two primes say $p = 5$ and $q = 11$. Then $N = 55$ and $\phi = 40$. Bob selects a number $e = 13$ such that $\gcd(e, \phi) = 1$. He then calculates $d = e^{-1}$ w.r.t ϕ using the ExtGCD algorithm; in this case $37 = 13^{-1}$ with respect to 40. Bob's public key is $(13, 55)$ while is secret key is 37.

  Now suppose Alice wants to encrypt a message in $\{1, 2, \ldots, 54\}$; say 29. The encryption is

  $$\mathsf{Enc}(21, 13, 55) = 29^{13} (\bmod\ 55) = 24$$

  To decrypt this, Bob does the following

  $$\mathsf{Dec}(24, 37) = 24^{37} (\bmod\ 55) = 29$$

- **Correctness of RSA.** We prove that as long as $m \in \{0, 1, 2, \ldots, N-1\}$, then if Alice sends the cipher according to the RSA encryption algorithm, then Bob will get back the same m when he decrypts. In particular, we prove the following theorem.

  Theorem 1. Let $(e, N), d$ be the (public,secret) key pairs generated by Bob. Then for any $m \in \{0, 1, \ldots, N-1\}$, Alice sends $c = m^e \bmod N$. Then, $c^d \bmod N = m$.

  Proof. We need to show $c^d \bmod N = m$, that is, we need to show $m^{ed} \equiv_N m$, that is

  $$\text{We need to show} \quad \left(m^{ed} - m\right) \equiv_N 0 \tag{1}$$

  Now, d is the inverse of e modulo $\phi = (p-1)(q-1)$. Thus,

  $$ed \equiv_\phi 1 \quad \Rightarrow \quad ed = \phi \cdot x + 1 \quad \text{for some integer } x$$

  Therefore,

  $$\left(m^{ed} - m\right) \quad \equiv_N \quad \left(m^{\phi \cdot x + 1} - m\right) \quad \equiv_N \quad m \cdot \left(m^{\phi \cdot x} - 1\right) \tag{2}$$

  Next, we show the RHS of (2) is $\equiv_p 0$ ***and*** $\equiv_q 0$. By the Pset 8, Problem 1(b), namely if p divides a and q divides a, then pq divides a, we would get (1).

  Let us prove $m \cdot \left(m^{\phi \cdot x} - 1\right) \equiv_p 0$ with the $\equiv_q 0$ being analogous. Two cases. One, $\gcd(m, p) \neq 1$, that is p divides m. In that case $m \equiv_p 0$ implies $m \cdot \left(m^{\phi \cdot x} - 1\right) \equiv_p 0$.

  Otherwise, $\gcd(m, p) = 1$. Fermat's Little Theorem then implies, $m^{p-1} \equiv_p 1$. Taking both sides to the power $(q-1)x$, we get $m^{(p-1)(q-1)x} \equiv_p 1$, that is, $m^{\phi \cdot x} - 1 \equiv_p 0$. Thus in this case also we get $m \cdot \left(m^{\phi \cdot x} - 1\right) \equiv_p 0$. □

Remark: *One corollary of the above theorem, and something to actually appreciate, is that* every *distinct message in $\mathbb{Z}_N$ actually must have a* distinct *cipher.*

- **A very short discussion on security of RSA.** Why RSA is secure is beyond the scope of this course. But take CS62 someday or some other security course.

  There are two "beliefs" which are involved in the security of RSA. It is good to state them out.

  - It is believed that knowing N and e and $m^e \bmod N$, one cannot "quickly" figure out m.
  - It is believed that knowing N, one cannot "quickly" find the *factors* of N.

  The former is probably clear — if the cipher could tell us the message, then we are in trouble. For the latter, note that if N can be factored into p and q, then any Eve could figure out $\phi = (p-1)(q-1)$ and then could figure out the sk by running MULTINVERSE herself.

  Is factoring really hard? This *may* seen weird. Especially, since our grade school teachers taught *that* to be the way to compute $\gcd$(!!) ALso, sure we can factor 21 into 3×7. However, if N has hundreds of digits, things are not that clear anymore.

  A final word. Although no one knows of an algorithm for factoring, in 1994 Peter Shor showed an algorithm for factoring using *quantum computers*. And this started the whole field of *post quantum cryptography* where people try to build a public key cryptosystem which can't be broken even with quantum computers. Since I have not said what quantum computers are, let me say no more.