- We're now going to use what we've seen to build a powerful encryption scheme. Recall the result we used to calculate high powers, namely $a^{\varphi(n)} = 1 \bmod n$ if a and n are coprime. We can exploit this to find roots in the following way.

- For example, we'll solve $x^{131} = 758 \bmod 1073$.

Working in the group $\mathbb{Z}_{1073}$ we want to change the equation $x^{131} = 758$ into something having $x^{\varphi(1073)+1}$, ie x, by raising both sides to the power u where $u = (131)^{-1} \bmod \varphi(1073)$

Firstly we compute $\varphi(1073) = \varphi(29 \times 37) = 28 \times 36 = 1008$. Notice that this is coprime to 131, so $\gcd(1008, 131) = 1$.

Now we use the Euclidean algorithm to find u and v satisfying $131u - 1008v = 1$, getting $u = 731$ and $v = 95$.

So $x^{131} = 758 \bmod 1073 \implies (x^{131})^{731} = 758^{731} \bmod 1073$

$\implies x^{1+1008(95)} \bmod 1073 = x \bmod 1073 = 758^{731} \bmod 1073$.

Unfortunately we can't use our friend 'little Fermat' to calculate this power of 758 quickly, but we can keep squaring to observe :

$731 = 2^9 + 2^7 + 2^6 + 2^4 + 2^3 + 2 + 1$

$\implies x = 758^{731} \bmod 1073 = (1011)(712)(663)(625)(1011)(509)(758) \bmod 1073 = 905 \bmod 1073$.

A table of 2^k-th powers of $758 \bmod 1073$:

$758^1 = 758 \bmod 1073$

$758^2 = 758^2 \bmod 1073 = 509 \bmod 1073$

$758^4 = 509^2 \bmod 1073 = 488 \bmod 1073$

$758^8 = 488^2 \bmod 1073 = 1011 \bmod 1073$

$758^{16} = 1011^2 \bmod 1073 = 625 \bmod 1073$

$758^{32} = 625^2 \bmod 1073 = 53 \bmod 1073$

$758^{64} = 53^2 \bmod 1073 = 663 \bmod 1073$

$758^{128} = 663^2 \bmod 1073 = 712 \bmod 1073$

$758^{256} = 712^2 \bmod 1073 = 488 \bmod 1073$

$758^{512} = 488^2 \bmod 1073 = 1011 \bmod 1073$

Notice that the values are now cycling!!

- Consider the following.

The recipient picks two very large prime numbers p and q, and multiplies to get $m = pq$, then $\varphi(m) = (p-1)(q-1)$.

Now choose any k coprime to $\varphi(m)$, then the recipient publishes the values of k and m in order to receive messages.

If the plaintext message is a string of numbers $a_1, a_2, \ldots, a_t$, the sender computes $b_1, b_2, \ldots, b_t$ where each $b = a^k \bmod m$.

Send the ciphertext message $b_1, b_2, \ldots, b_t$.

The receiver only has to solve $x^k = b \bmod m$ for each b to decrypt the message, which is easy since they know $\varphi(m)$.

- Two examples* to illustrate this *RSA* encryption scheme:

  Convert letters to numbers via $A = 11$, $B = 12, \ldots, Z = 36$. Pick primes $p = 12553$ and $q = 13007$ so $m = 163276871$.

  Then $\varphi(m) = (p-1)(q-1) = 163251312$, and choose $k = 79921$, which is coprime to m.

  The plaintext message "tobeornottobe" becomes $30251215252824425303025l215$ after conversion.

  Since m has 9 digits, we break the message into 8-digit strings: 3021215 25282425 30302512 15.

  Raising each of these to the power $k = 79921$ working mod 163276871 gives:

  149419241 62721998 23054767 40481382 as the encrypted text to send.

  This time, you've received an encrypted message:

  145387828 47164891 152020614 27279275 35356191

  Calculating via the Euclidean algorithm gives: 79921^{-1} mod $163251312 = 145604785 = u$

  Raising each of the terms in the encrypted text to the power u working mod 163276871 gives:

  30182523 26292524 19291924 30282531 122215 as the decrypted 'text', which translates to:

  "thompsonisintrouble" as the received plaintext.

- As you can see, there is some computational overhead involved! Hence this system tends not to be used to communicate long messages, rather it's used to communicate short information on how to decode other long messages which have been encrypted (e.g., using variations of one-time pads) via vastly less computationally intensive methods.

- The reason that this is both viable and potentially hard to crack is that it's relatively easy to encrypt and decrypt if you know the value of $\varphi(m)$, but finding that value if you don't know the factorisation** of m is decidedly tricky. Hence if m was constructed from two extremely large primes, then *it is presumed that**** it's very hard to factorise m.

* These are taken from a really nice, yet elementary book: Silverman, J.H. (1997). *A Friendly Introduction to Number Theory,* Prentice-Hall, NJ.

** A rich and fascinating array of techniques have been applied to understanding factorisation, ranging from 'regular' algebraic number theory to geometric ideas in fairly exotic contexts, often peppered with delicate probabilistic methods.

*** This is a fairly careful statement. A great deal of work has been expended on trying to understand the complexities of integer factorisation. As of the date of these notes (*March 2014*), it is not widely known if any reasonably efficient approaches to factorisation exist. No theorems are widely known indicating *decisively* the degree of computational complexity of factorisation. It could be that reasonably good algorithms do exist and are not widely known, but we live in a world of bluff, double bluff, etc..

- This is an example of *public key cryptography*, the principle of which is as follows.
  - Let P be the set of potential plaintext messages, Z the set of encrypted texts, and C the set of keys.
  - There are functions $\varepsilon : P \times C \longrightarrow Z$ and $\delta : Z \times C \longrightarrow P$, being the *encryption* and *decryption* maps respectively, satisfying $\delta(\varepsilon(p, c), c) = p$, i.e., decrypting the encrypted message should give you the original message!!
  - Evaluating ε should be easy and evaluating δ should be hard.
  - There's an additional layer; namely a set S of secret keys, together with a pair of functions $\sigma : S \longrightarrow C$ and $\sigma^{-1} : C \longrightarrow S$ making public (or hiding) the keys.
  - Evaluating σ and δ^* should be easy, where $\delta^*(z, s) = \delta(z, \sigma(s)) = \delta(z, c)$, but evaluating σ^{-1} should be hard.

Such functions are called one-way functions

- Suppose *Alice* wants to send a message p to *Bob* in an environment where *Eve* is eavesdropping.
  - *Bob* chooses a secret key $b \in S$, keeping that information private. He then easily computes $c = \sigma(b) \in C$ and tells that to *Alice*, knowing that *Eve* will see it.
  - Alice uses c to compute $z = \varepsilon(p, c)$ easily, and sends that to *Bob*.
  - *Bob* knows the secret key b so easily computes $\delta^*(z, b) = \delta(s, c) = p$.
  - *Eve* doesn't know the secret key b, so has either to compute $\delta(z, c)$ or $\sigma^{-1}(c)$, both of which are hard.

T*rapdoor one-way functions* are such that adding a piece of information makes computation of the inverse easy.

- There's a further problem: since *Eve* knows *Bob's* public key, she can impersonate *Alice* and send her own encrypted message to *Bob*, claiming that it's coming from *Alice*.
  - We assume explicitly that $P = Z$, and symmetrically that $\varepsilon(\delta(z, c), c) = z$ for all $z \in Z$ and $c \in C$.
  - Bizarrely, *Alice* treats p as if it were encrypted, and *decrypts it* using her own secret key a to get $w = \delta(p, a)$.
  - She then writes q = "this is a signed message from Alice", then creates v = (q appended by w) and sends $z = \varepsilon(v, c)$ to *Bob* using his public key.
  - *Bob* decrypts this using his secret key to get $v = \delta(z, b)$, sees that it's from Alice, even though much of it is encrypted, and *encrypts it* using *Alice's* public key c' to get $w = \varepsilon(v, c')$.