Public Key Cryptography

Terminology

- Asymmetric cryptography
- Public key (known to entire world)
- Private key (not called "secret" key)
  - That's term used for symmetric crypto
- Encryption process (P to C with public key)
- Decryption Process (C to P with private key)

Terminology

- Symmetry:
  - Encrypt with public key, then decrypt with private
  - Encrypt with private key, then decrypt with public key
- Digital signature (P signed with private key)
  - Only holder of private key can sign, so can't be forged
  - But, can be recognized!

Uses

- Orders of magnitude (?) slower than symmetric key crypto, so usually used to initiate symmetric key session
- Much easier to configure, so used widely in network protocols to establish temporary shared key that is used to transmit secret (symmetric) key

Uses

- Transmitting over insecure channel
- Alice <A_{pu}, A_{pr}> , Bob <B_{pu}, B_{pr}>
- Alice to Bob encrypt m with B_{pu}
- Bob to Alice encrypt m with A_{pu}
- Knowing public key of other person is one of the biggest challenges of using public key crypto
  - That is, being sure that the public key you have actually belongs to the entity you think it does!

Uses

- Secure storage on insecure media
  - Encrypt not whole file, but a randomly generated secret key with public key. Then encrypt file using secret key.
  - Note if you lose private key, you're out of luck. To backup, encrypt secret key with public key of a trusted friend (lawyer).
- Important advantage: Alice can enrypt a message for Bob without knowing Bob's decryption key

Uses

- Authentication
  - If Bob wants to prove his identity with symmetric key crypto, he needs a different symmetric key shared with each potential correspondent (otherwise friends can impersonate him)
  - Alice can verify she's talking to Bob (assuming she knows his public key) by sending a message r to Bob encrypted with Bob's public key. Bob sends back the cleartext message r (which only he could have decrypted).

Uses

- Authentication
  - Note Alice need not keep any secret information in order to verify Bob. (Unlike symmetric key crypto, in which a backup tape with a copy of the symmetric key might be used to impersonate Bob)

Uses

- Digital Signatures: prove message generated by particular individual
  - If Bob encrypts a message with his private key, this proves both
    - Bob generated the message
    - The message has not been modified (to do so requires Bob's private key)
      - Without Bob's private key, creating ciphertext that decrypts to something meaningful with Bob's public key is computationally infeasible

Uses

- Digital Signatures: prove message generated by particular individual
  - Non-repudiation: Bob cannot deny having generated the message, since Alice could not have generated the proper signature without knowledge of Bob's private key.
  - Note that this can't be done with symmetric key. If Bob tries to claim he didn't send the message, Alice would know he's lying (because no one but herself and Bob would have the secret key), but Alice could not prove this to anyone else (since she herself could have generated the authentication code).

The General Idea

- We use two one way functions
  - Multiplication vs factoring
  - modular exponentiation vs modular logarithm
    - Either can be a one way trap door process

The General Idea

- Multiplication
  - Relatively easy, even if you are multiplying two huge numbers
- Factoring
  - Difficult: No matter how it is done, need to check many possible factors
  - Think of it as finding the combination for a lock (prime factorization)
- Here: n = pq, where p and q are both large primes

The General Idea

- Modular exponentiation
  - Relatively easy: Think of a clock face with the requisite number of numbers on it
  - Modular multiplication like winding a length of rope around it and seeing where it stops.

46 mod 12 = 10

Thanks to Kahn Academy

The General Idea

- Computing modular logarithm (discrete logarithm) is difficult
  - modular exponentiation distributes values in manner close to uniformly random around clock face
  - Finding discrete log means testing many possible values
    - For large numbers, this is a prohibitively expensive operation

The General Idea

```
[m2-mcs-dszajda:slides dszajda$ java ModularExponentiation 17 5
 5^0 -> 1
 5^1 -> 5
 5^2 -> 8
 5^3 -> 6
 5^4 -> 13
 5^5 -> 14
 5^6 -> 2
 5^7 -> 10
 5^8 -> 16
 5^9 -> 12
 5^10 -> 9
 5^11 -> 11
 5^12 -> 4
 5^13 -> 3
 5^14 -> 15
 5^15 -> 7
```

The General Idea

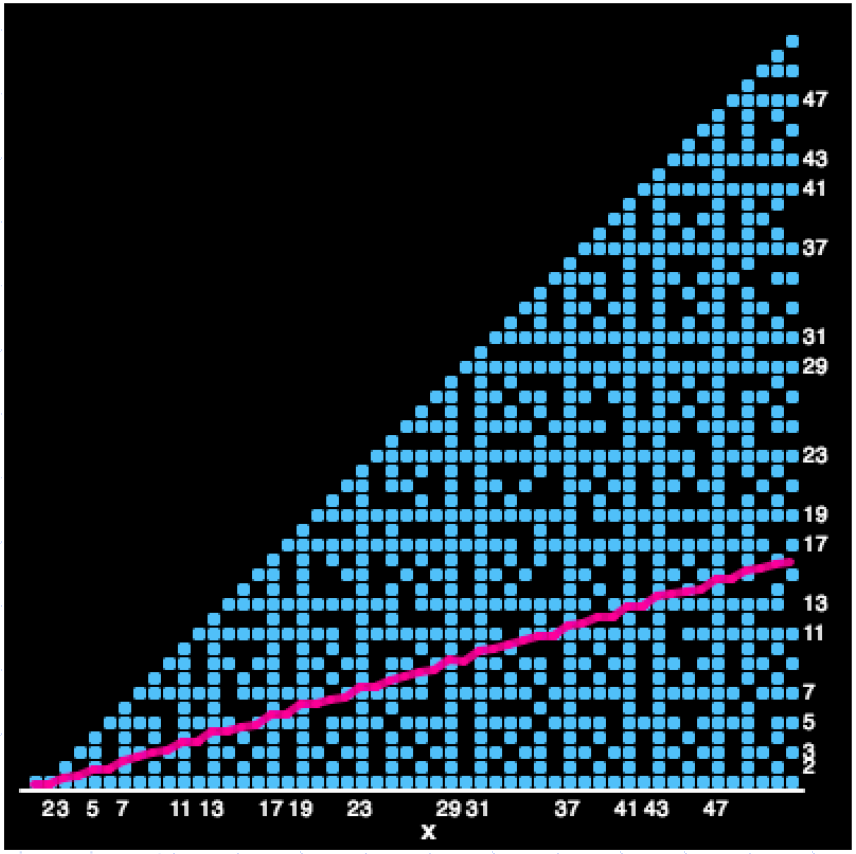
- We use two tools to make this work
- First, the Euler totient function
  - This is a one-way trap door function!
- We use Euler's Theorem

Totient Function

- Allegedly from total and quotient
- How many numbers less than n are relatively prime to n?
- Totient function, $\phi(n)$ gives this.
- If n is prime, $\phi(n) = n-1$ (1,2,…n-1)
- If p and q are prime, $\phi(pq) = (p-1)(q-1)$
  - p, 2p, … (q-1)p q, 2q, … (p-1)q not rel. prime so have
  - $pq - 1 - [(p-1) + (q-1)] = (p-1)(q-1)$
- Can also be seen by noting that $\phi(ab) = \phi(a)\,\phi(b)$

Totient Function

- This is trap door
  - difficult, in general, to determine value
  - easy if you know the prime factorization



Euler's Theorem

For a and n relatively prime,

$$a^{\phi(n)} \equiv 1 \bmod n$$

Euler's Theorem

Thus, for all positive integers k,

$$a^{\phi(n)k} \equiv 1 \bmod n$$

Euler's Theorem

So, for any positive integer x,

$$a^x \bmod n = a^{x \bmod \phi(n)} \bmod n$$

Euler's Theorem

So, for any positive integer x,

$$a^x \bmod n = a^{x \bmod \phi(n)} \bmod n$$

Why? Let $x = r\phi(n) + s$. Then

$$a^x \bmod n = a^{r\phi(n)+s} \bmod n = a^{r\phi(n)} a^s \bmod n = a^s \bmod n$$

Euler's Theorem

So, for any positive integer x,

$$a^x \bmod n = a^{x \bmod \phi(n)} \bmod n$$

Upshot: We can do exponentiation mod the totient function

RSA

- Key length variable (but should now be at least 2048 bits)
  - However, if you need security beyond 2030, at least 3072 bits recommended
- Plaintext block must be smaller than key length
- Ciphertext block will be length of key

RSA

- Choose two large primes (around 1048 bits each) p and q. Let n = pq (very difficult to factor)
  - Number of bits of n is the RSA "key length"
- Choose number e that is relatively prime to $\phi(n)$. Can do this since you know p and q and thus $\phi(pq)$ and from the derivation know exactly which numbers are relatively prime!

RSA

- Public key is <e, n>
- To make private key, find d that is the multiplicative inverse of e mod $\phi(n)$ (so $ed = 1 \bmod \phi(n)$) (use Euclid's algorithm)
- Private key is <d,n>
- To encrypt a number m, compute $c = m^e \bmod n$.
- To decrypt: $m = c^d \bmod n$.

RSA Example

1. Select primes: $p=17$ & $q=11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e: $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de=1 \bmod 160$ and $d < 160$ Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key $KU=\{7,187\}$
7. Keep secret private key $KR=\{23,17,11\}$

RSA Example cont

- sample RSA encryption/decryption is:
- given message $M = 88$ (nb. $88<187$)
- encryption:

$$C = 88^7 \bmod 187 = 11$$

- decryption:

$$M = 11^{23} \bmod 187 = 88$$

Questions

- Why does it work?
- Why is it secure?
- Are operations sufficiently efficient?
- How do we find big primes?

Why Does It Work?

- We chose d and e so that $de = 1 \bmod \phi(n)$, so for any x,
- $x^{(ed)} \bmod n = x^{(ed \bmod \phi(n))} \bmod n = x^1 \bmod n = x \bmod n$.
- And $(x^e)^d = x^{(ed)}$

Why Is It Secure?

- We're not sure it is, but it seems to be
- Based on premise that factoring a big number is difficult.
  - Semiprimes, the product of two (not necessarily distinct) primes, are most difficult numbers to factor.
    - Largest such semiprime factored (in Feb. 2020) is RSA-250, 829 bits, 250 decimal digits.
      - Took 2700 core-years of computing power.
      - Larger semiprimes factor, but these were "weaker" (some semi-primes have specific forms or prime factors that are very close)

Why Is It Secure?

- If you can factor n, you're golden:
  - Problem is one of finding modular log (i.e. inverse of exponential)
  - Why? Adversary knows <e,n>. So for message m, knows ciphertext is $c = m^e \bmod n$.
  - So if adversary can reverse the exponentiation (that is, find the number x s.t. $x^e \bmod n = c$), she's got the original message m!
  - Remember how we originally find this inverse: By knowing $\phi(n)$. Which is difficult to know if you can't factor n

Why Is It Secure?

- We don't know that there are not easier ways to break it (we do know that breaking it is no harder than factoring)
- We do know that it can be broken with a quantum computer using Shor's Algorithm (1994) which has cubic time and linear space complexity in the number of bits of the number being factored
  - So if quantum computers become practical...

This Can be Broken if Used Carelessly!

- Suppose Bob knows I'm going to send Alice a message with the identity of a congressperson who is crooked
  - So I'll encrypt name with Alice's public key and send resulting ciphertext
  - Bob knows Alice's public key, so he can encrypt all possible names and identify the crook.
  - A way around this: pad the name with a large random number, say 128 bits long. Then Bob has to try all of a space that is huge. (I.e. 535 x 2^{128})

Progress in Factorization

- In 1977, RSA inventors dare Scientific American readers to decode a ciphertext printed in Martin Gardner's column.
  - Reward of $100
  - Predicted it would take 40 quadrillion years
  - Challenge used a public key size of 129 decimal digits (about 428 bits)
- In 1994, a group working over the Internet solved the problem in 8 months.

Finding Big Primes

The *prime number theorem* states that $\pi(x)$, or the number of primes less than or equal to x obeys the following property:

$$\lim_{x \to \infty} \frac{\pi(x)}{x/\ln x} = 1$$

Meaning that we can approximate the number of primes less than x by $x/\ln x$.

Numbers representable by n bits are those less than 2^n. Therefore, with 1024 bits, you can count up to 2^{1024}, which is an enormous number. I believe you want to know about the prime numbers which are representable by 1024 bits and by *no fewer* than that amount. We can approximate this by:

$$\pi(2^{1024}) - \pi(2^{1023}) \approx \frac{2^{1024}}{\ln 2^{1024}} - \frac{2^{1023}}{\ln 2^{1023}} \approx 1.26 \cdot 10^{305}$$

Finding Big Primes

- No nice way of absolutely determining that a huge number is prime, but we can guess pretty accurately
- Fermat's Theorem: If p is prime, and 0 < a < p, then a^(p-1) mod p = 1 mod p.
  - Works because though it's possible for a^(n-1) = 1 mod n for a non-prime, it's not likely. For a randomly generated number of about 100 digits, probability that n is not prime but relation holds is about 1 in 10^(13).
  - Other similar probabilistic algorithms for finding large primes

Finding Big Primes

- Update: usually Fermat test with base 2 is applied
  - because it can be optimized
- Then several Miller-Rabin tests applied
  - How many depends on how small you want the probability of being wrong
  - Typically somewhere around 20 tests run
  - Gets probability of being wrong down to around 2^{100}
- See FIPS Pub 186-5 for details

The AKS Algorithm

- The Agrawal-Kayal-Saxena Primality Test
  - Published in 2002
  - A deterministic ***polynomial time*** primality-proving algorithm
  - Developed by three researchers at the Indian Institute of Technology Kanpur
  - Answered a centuries old question (and in a surprising way)!
  - Won 2006 Godel Prize and 2006 Fulkerson Prize
- Unfortunately the "constants" involved in the computational complexity estimates are very large
  - So not practical for identifying large primes (but making this competitive with probabilistic algorithms is a ongoing research area)
  - The probabilistic algorithms more efficient and limit odds of finding non-prime to tiny probability

The AKS Algorithm

- Real difficulty: probability of a hardware error running this algorithm is higher than the probability of accidentally choosing a non-prime with the earlier methods!

Diffie-Hellman

- Oldest public key cryptosystem still in use
- Does neither encryption nor digital signatures.
- It allows two individuals to agree on a symmetric key even though they can only communicate over insecure channels.

Diffie-Hellman

- Remarkable because neither Alice nor Bob need any apriori information, yet after the exchange of two messages, they share a secret number.
- One bad thing: no authentication, so Alice may be setting up a key with Trudy!

The Process

- Alice and Bob agree on two primes, p and g, where p is a large prime and g is a number less than p (with some restrictions)
- Each chooses a random 1024 bit number (SA for Alice, SB for Bob).
- Alice computes TA = g^{SA} mod p. Bob computes TB = g^{SB} mod p.
- They exchange their T values
- Alice computes TB^{SA} mod p, Bob computes TA^{SB} mod p.
- Done: $TB^{SA} = (g^{SB})^{SA} = g^{(SB*SA)} = g^{(SA*SB)} = (g^{SA})^{SB} = TA^{SB}$ mod p.

Why It Is Secure

- Whole world knows g^{SA} and g^{SB}, but getting $g^{(SA*SB)}$ means having to do a modular logarithm
  - If can find y such that $g^{y} = g^{SA}$, then know SA.
- And well, it's not exactly secure -- it has a problem with a person-in-the-middle attack (the lack of authentication of endpoints)