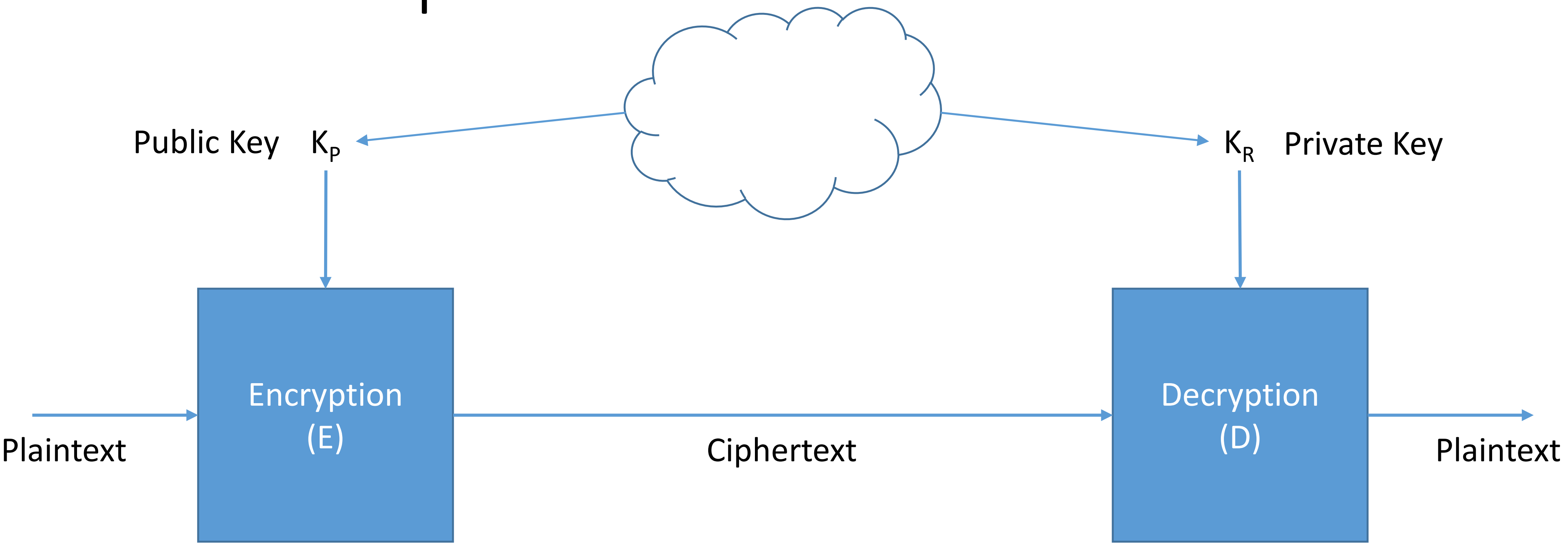
Public Key Cryptography

Vermont Technical College

CIS-4040

Peter Chapin

Basic Concept



Public Key Encryption: $K_P \neq K_R$

Basic Use Case

- The steps…
  1. Alice computes (public, private) key-pair.
  2. Alice posts public key anywhere (e. g., her web site)
  3. Alice keeps private key secure
  4. Bob downloads Alice’s public key
  5. Bob encrypts a message for Alice’s eyes only
  6. Bob sends message to Alice
  7. Alice uses private key to decrypt message
- *Not feasible for attacker to compute private key from public key*

A Few Observations

- Public key algorithms are (usually) slower than symmetric algorithms
- Public key algorithms are based on mathematically “hard” problems, not “confusion” and “diffusion”
- Key lengths can’t be compared with symmetric algorithms
- Public key algorithms are not necessary more secure than symmetric algorithms
- Public key algorithms compliment, but do not replace symmetric algorithms
  - **Good for symmetric key distribution; not good for bulk data transfer**

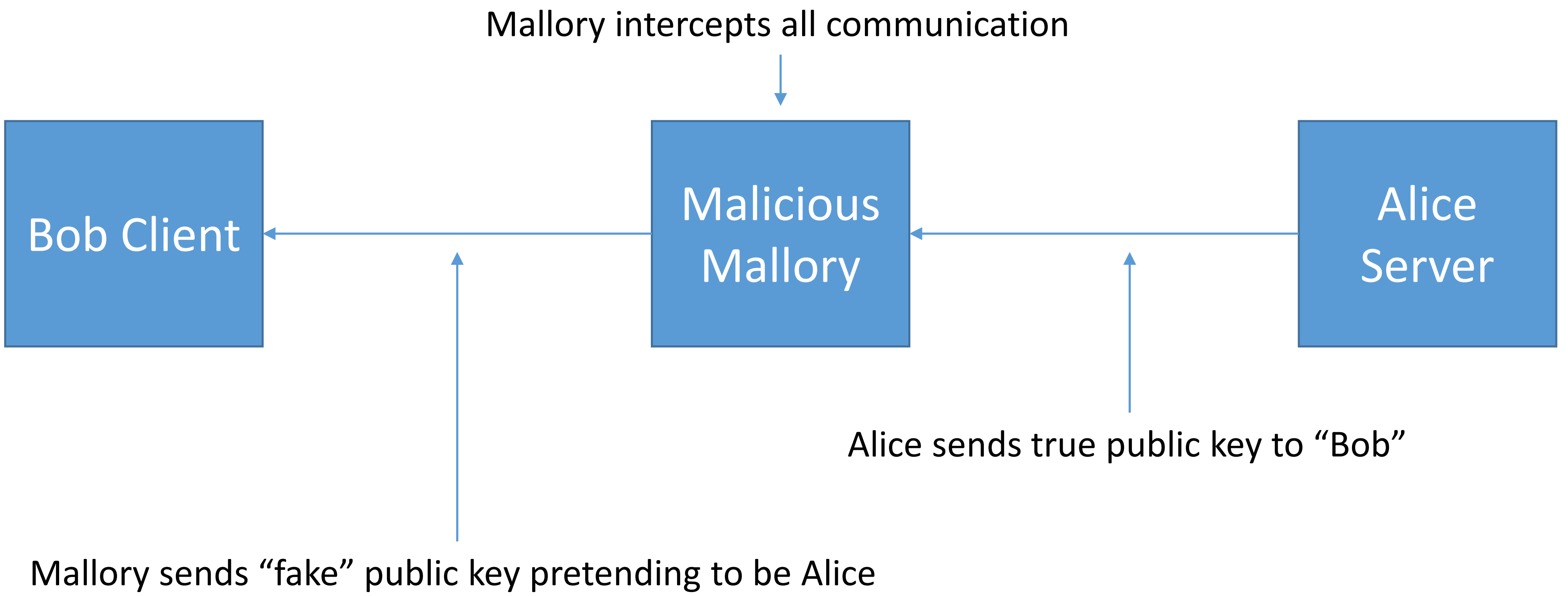
Key Exchange

- Public key algorithms solve the key exchange problem
  - SERVER: Sends its public key to the client. Doesn’t matter if seen on network
  - CLIENT: Generates a random session key for a symmetric algorithm
  - CLIENT: Encrypts session key with server’s public key
  - CLIENT: Sends encrypted session key to server
  - SERVER: Decrypts session key with private key
- Client & Server now share a symmetric key
  - All subsequent communication (bulk data transfer) encrypted with a symmetric algorithm. *Faster!*
  - Eve can’t decrypt session key (can’t compute server’s private key)

Security Problem

- The steps…
  1. Mallory creates (public, private) key-pair
  2. Mallory posts public key as Alice’s (she lies)
  3. Bob downloads “Alice’s” public key
  4. Bob encrypts message he thinks only Alice can read
  5. Mallory intercepts message on its way to Alice
  6. Mallory decrypts message intended for Alice

New Problem: (Wo)Man-in-the-Middle



Mallory intercepts all communication

Bob Client

Malicious Mallory

Alice Server

Alice sends true public key to “Bob”

Mallory sends “fake” public key pretending to be Alice

Mallory negotiates (separate) session keys with Alice and Bob; can now read and modify all messages!

Digital Signatures

- The steps…
  - Alice encrypts message with her private key
  - Alice posts message on her web site (or sends to Bob)
  - Bob uses Alice’s public key to decrypt message
    - If successful, Bob knows only Alice could have encrypted the message
- What is “success” in this context?
  - There’s a bit more involved
  - … must wait until we talk about cryptographic hashes

It Gets Worse

- The steps…
  1. Mallory creates reply, pretending to be Alice
  2. Mallory digitally signs reply using the private key
  3. Mallory sends reply to Bob
  4. Bob verifies digital signature using “Alice’s” public key
  5. Bob feels confident he is talking to Alice
- Problem: **Bob doesn’t realize the public key is not really Alice’s**

One (Non)-Solution

- The steps…
  - Alice and Bob meet in a café (i. e., use a secure channel)
  - Bob authenticates Alice
    - Maybe he knows what she looks like
    - Maybe he insists she brings documentation (passport, birth certificate, driver's license)
    - *Notice in the second case Bob relies on documentation from a trusted third party*
  - Alice personally gives Bob her public key
- Problem: *How is this better than symmetric encryption?*

Authentication: *The act of verifying the identity of a principle*

Trusted Third Party

- The steps…
  - Alice generates a (public, private) key-pair
  - Alice submits her public key to Trent, a trusted third party
  - Trent authenticates Alice
  - Trent digitally signs Alice’s public key: creates a *certificate*
  - Alice posts her public key certificate on her web site
- Now…
  - Bob download’s Alice’s certificate
  - Bob verifies Trent’s digital signature on the certificate
  - Bob believes he has Alice’s true key

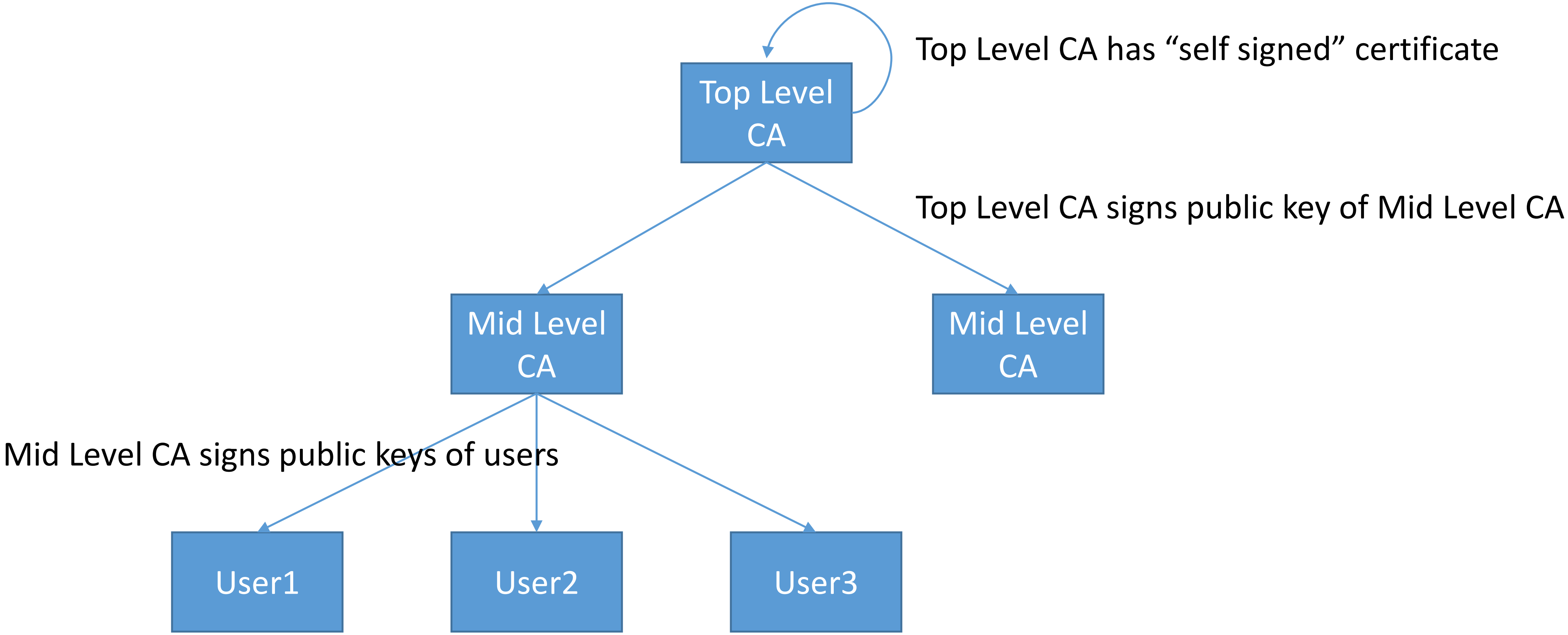
What’s Mallory To Do?

- The steps…
  - Mallory creates a (public, private) key-pair
  - Mallory presents public key to Trent as “Alice”
  - Trent requires Mallory prove she is Alice
  - Mallory fails to do so. No certificate is made
- *Assumes: Mallory can’t trick Trent*. **Trent must know his business**

But Wait…

- How did Bob get Trent's public key?
  - We haven't solved anything… just moved the problem to Trent
- But… Trent can be a trusted third party for many users
- Consider:
  - IT department at VTC as "Trent"
  - Students authenticate to the VTC helpdesk to get a certificate
  - Students get a copy of VTC IT's public key at that time (*mutual authentication*)
  - Students/faculty can send encrypted messages to each other without prior arrangements
  - … as long as we trust VTC's IT department to sign keys appropriately

Certificate Authorities

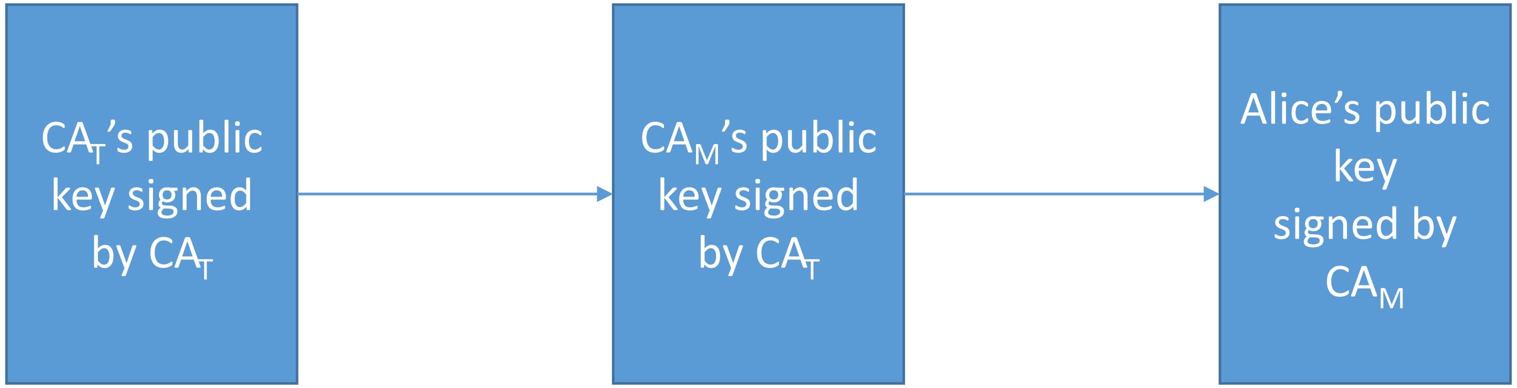


Top Level CA has “self signed” certificate

Top Level CA signs public key of Mid Level CA

Mid Level CA signs public keys of users

Certificate Chains



CA$_T$'s public key signed by CA$_T$

CA$_M$'s public key signed by CA$_T$

Alice's public key signed by CA$_M$

Self-signed certificate

Bob downloads Alice's certificate

Bob verifies certificate chain

Bob Must Trust…

  - … that each CA in the chain signs keys “appropriately” (after authentication)
  - … that he has the correct public key for the top level CA
- We still haven’t solved the problem!
  - We’ve only moved it to the top level CA
  - And we now rely on proper behavior of various intermediate CAs

Example

- Consider:
  - Alice is at UCLA. Bob is at VTC
  - Alice’s key is certified by the UCLA IT department
  - The UCLA IT department’s key is certified by Thawte (a commercial certificate authority)
  - Bob does not know Alice personally; downloads her key from her web page
  - Bob has the Thawte public key (it’s common knowledge)
  - Bob trusts Thawte
  - Bob trusts the UCLA IT department
  - Bob believes he has the correct key for Alice

RSA

- Ron Rivest, Adi Shamir, and Len Adleman
  - “A method for obtaining digital signatures and public key cryptosystems.” *Communications of the ACM*, February 1978
- Key generation…
  - Pick two large primes, p and q (for example 100 digits in size)
  - Compute $n = pq$
  - Compute the Euler Totient function of n: $\varphi(n) = (p - 1)(q - 1)$
  - Select a small, odd integer e that is relatively prime to $\varphi(n)$. That is, select an e such that $gcd(\varphi(n), e) = 1$
  - Compute d as the multiplicative inverse of e, modulo $\varphi(n)$
  - Public key = $\{ e, n \}$. Private key = $\{ d, n \}$

Why Hard?

- The attacker knows { *e, n* }. Wants to find { *d, n* }
  - Attacker wants to compute *d*.
  - Problem: Attacker needs to know *φ(n)*
  - Easy to do if the prime factors of *n* (we called them *p* and *q*) are known
  - Problem: *Finding prime factors of huge numbers is hard!*
- It has been shown there is no “easier” way of finding *φ(n)* than computing the prime factors of *n*.

A Little Math…

- Every positive number has a prime factorization:
    - 12 = 2 * 2 * 3
    - 13 = 13 (this is a prime number)
    - 14 = 2 * 7
    - 15 = 3 * 6
    - 16 = 2 * 2 * 2 * 2
    - 17 = 17 (this is a prime number)
    - 18 = 2 * 3 * 3
- The prime factorization of a number is unique

Finding Prime Factors

- Trial Division
  - Totally infeasible when *n* is, say, 100 digits in size!
- The Quadratic Sieve
- The General Number Field Sieve
  - The fastest general purpose factoring algorithm known for integers greater than 10^{100}
- The Special Number Field Sieve
  - Faster still, but only works on numbers with specific characteristics. Not suitable for general use.

Not Good Enough

- None of the factoring methods are fast enough!
  - Attacker's problem: *Generating RSA keys is asymptotically more efficient than any known factoring method*.
  - **User will simple use a larger key** (no problem to generate) **to frustrate even the most resourceful attacker**
  - RSA keys typically in the 2048 bit range today
- Note: *The key is not 2^{2048} operations hard!*
  - The problem is not trying every key, but rather factoring a 2048 bit integer

More Math

- Two numbers a, b are *relatively prime* if they share no common factors (besides 1)
  - Example 10 and 21
  - 10 = 2 * 5
  - 21 = 3 * 7
  - No common factors! Thus 10 and 21 are relatively prime
- Euler Totient Function, $\varphi(n)$, gives number of values less than n that are relatively prime to n
  - $\varphi(15) = 8$ because { *1, 2, 4, 7, 8, 11, 13, 14* } are relatively prime to 15
  - For $n = pq$ (p, q both prime), $\varphi(n) = (p – 1)(q – 1)$ ***Easy to calculate!***

RSA Encrypt/Decrypt?

- So how do we use the public key { *e, n* } to encrypt a message?
    - Treat message like a huge integer *M*
        - Think about writing down all the bits of the message as one big number
    - Ciphertext $C = M^e \ (mod\ n)$
- So how do we use the private key { *d, n* } to decrypt a message?
    - Plaintext $M = C^d \ (mod\ n)$
- Why does this work?
    - *Proof omitted!*

Finding Large Primes

- It is easier than it sounds…
  - Generate 1024 random bits
  - Ensure the MSB and LSB are both 1
    - MSB = 1 to force the number to actually be 1024 bits
    - LSB = 1 to force the number to be odd (no even number is prime except for 2)
  - Use efficient primality testing algorithm such as Miller-Rabin
  - Not prime? Just add 2 (to advance to the next odd number) and repeat
  - Won't this take a while? Prime numbers must be rare
    - Not as rare as you think…

How Many Primes?

- Let *π(n)* be the number of primes less than or equal to *n*
  - Called the Prime Counting Function
- The Prime Number Theorem states
  - $\pi(n) \approx \frac{n}{\ln(n)}$
  - Example: Let *n* = 10^9, then *π(n)* = 50,847,534 (exact), the approximation = 48,254,942. Better approximations are known (see linked page above)
  - Consequence: To find a prime in the vicinity of *n*, we must look, on average, at *ln(n)* values.
  - Example: ln(2^{1024}) = 710. That’s not very many!

ElGamal: Another Public Key System

- Key generation…
  - Choose large prime *p* and two random numbers *g*, *x* such that $g < p$ and $x < p$. The values of p and g can be shared
  - Compute $y = g^x (mod\ p)$. It is efficient to compute modular exponentiation.
  - Public key is { *y, p, g* }
  - Private key is { *x* }

ElGamal

- To encrypt…
  - Choose random *k* relatively prime to *p – 1*
  - Compute $a = g^k (mod\ p)$
  - Compute b = $y^k M (mod\ p)$, where *M* is the message
  - Cipher text is { *a, b* }
- To decrypt…
  - Plaintext is $b(a^x)^{-1} (mod\ p)$
  - Notice that $M < p$ for this to work

Why is ElGamal Secure?

- ElGamal depends on the difficulty of the **discrete logarithm** problem.
  - Consider $y = g^x (mod\ p)$ for some large p and x (with $x < p$)
  - If { p, g } are known and y is known. Computing x is finding the discrete logarithm of y base g modulo p.
  - For large p this is computationally infeasible.