HTTPS

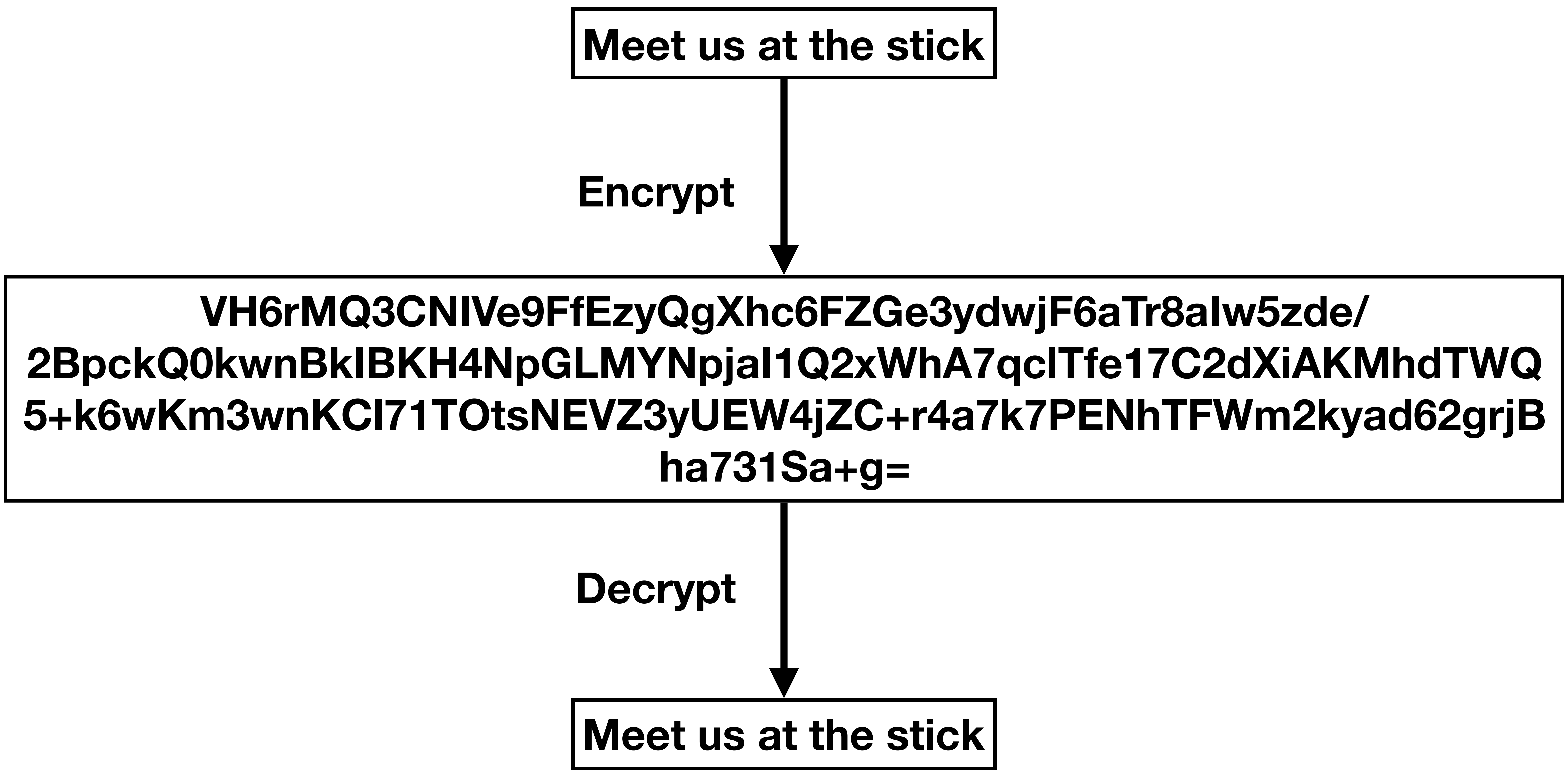
The Problem

- Using HTTP, anyone with access to your packets can see everything you are doing online
    - This includes your passwords <- discussion
- Who has access to my packets?
    - Your ISP at home
    - UBIT on campus
    - Tier 1 Networks
    - .. and **everyone within wifi range of your device!**

Encryption

- Hide communication from attackers/eavesdroppers
- Want to send a **plaintext** message
  - Could be HTML, JSON, JavaScript, text, image, etc
  - Encode as bytes before sending so any format can be encrypted
- Encrypting the **plaintext** outputs a random looking **cyphertext**
- No one should be able to read this cypher text except the intended recipient
- Only the intended recipient can decrypt the **cyphertext** and read the **plaintext** message

Encryption


Meet us at the stick
Encrypt
VH6rMQ3CNIVe9FfEzyQgXhc6FZGe3ydwjF6aTr8aIw5zde/
2BpckQ0kwnBkIBKH4NpGLMYNpjaI1Q2xWhA7qcITfe17C2dXiAKMhdTWQ
5+k6wKm3wnKCl71TOtsNEVZ3yUEW4jZC+r4a7k7PENhTFWm2kyad62grjB
ha731Sa+g=
Decrypt
Meet us at the stick

Public Key Encryption

- How do we ensure that only the intended recipient can decrypt the message?
    - Public Key Encryption
- Generate a public and private encryption key
    - Private key is kept private
    - Public key is shared with anyone/everyone
- A message encrypted with the public key can only be decrypted by the corresponding private key
- If I want to send a message to someone, I can encrypt the message with their public key

Public Key Encryption

- The public and private key are inherently related
- An attacker must not be able to determine the private key given the public key [In any reasonable amount of time]
- An attacker must not be able to decrypt cyphertext without the private key [In any reasonable amount of time]
- Any public key encryption algorithm must have these property [And more]
    - Math and theory will help us

Public Key Encryption - RSA

- RSA - Rivest-Shamir-Adleman
  - A public key crypto system with the desired properties [we hope]

- Provides algorithms to:
  - Generate a public/private key pair
  - Encrypt with the public key
  - Decrypt with the private key

Public Key Encryption - RSA

Key Generation

- Choose two [large] prime numbers p and q
- $n = p*q$
- $\lambda(n) = \text{lcm}(p-1, q-1)$
- Choose $e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$
- d = e^{-1} mod λ(n)

- Share e and n as the public key
- Keep d as the private key
- Discard p, q, and λ(n)

Public Key Encryption - RSA

Encryption

- To send a message m
- Compute $c = m^e \bmod(n)$
- Send c

Decryption

- Receive c
- Compute $m = c^d \bmod(n)$
- Read m

Public Key Encryption - RSA

- RSA key generation gives us: m = m^(e*d) mod(n)
- This means that we can also encrypt with the private key and decrypt with the public key
  - We call this a signature
  - Everyone can decrypt the message so there is no privacy
  - However, there is a guarantee that the author is legitimate (You know I'm the one who sent this message)
  - Useful to sign authentication tokens so the server know that it authorized this user

Public Key Encryption - RSA

- What about a brute force attack?
  - Attacker has the public key
  - Run the same attack we used for hashing
- Use a padding algorithm
  - Add random bits to the end of a message
  - Attacker must guess these random bits
  - Adds security!
  - Unlike salt, this padding can be kept secret and adds entropy
  - Make encryption of even short messages secure

Public Key Encryption - RSA

Key Generation

- Choose two [large] prime numbers p and q
- $n = p*q$
- $\lambda(n) = \mathrm{lcm}(p-1, q-1)$
- Choose $e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$
- d = e^{-1} mod λ(n)

- Given p, q, and e an attacker can easily compute d (private key)
- Everyone already has e and n (the public key)
- So just factor n to get p and q, then compute the private key

Public Key Encryption - RSA

Factoring is hard [We hope]

- There is no publicly know algorithm that can efficiently factor a number
- Worst case is factoring the multiplication of two large primes
  - Exactly what RSA relies on for security
- It has not been proven that factoring is a "hard" problem
  - Quantum computers can factor efficiently
- We call the factoring problem a cryptographic primitive
  - Another cryptographic primitive is the discrete log problem

Public Key Encryption

- Java Demo

Key Exchange

- In practice, a server will have a public/private key pair verified by a certificate authority (CA) <- More later
- The client can verify that the public key is valid
- Server can encrypt and send data
- Issues:
  - RSA [and other public key cryptosystems] is slow
  - The client can't securely encrypt (No CA verification)
- Instead, use RSA to generate a symmetric key

Key Exchange

- Generate a temporary symmetric key that only the client/ server know
    - Symmetric key is used to encrypt and decrypt
- Diffie-Hellman key exchange is used to securely generate a private key using the servers public/private keys to verify the server
- All specific details of the symmetric key depend on the specific algorithms being used

HTTPS

- HTTP over TLS (Transport Layer Security)
- TLS is the protocol that dictates the public/private key encryption, key exchange, and symmetric key encryption
- Encrypts the entire HTTP requests/responses using the symmetric key
- **Eavesdroppers can still see TCP/IP**
    - Including source/destination IP addresses!
        - They don't know what you're saying, but they know who you're talking to
    - This is why VPNs are still popular even though most sites use HTTPS in current year

HTTPS

Man-in-the-middle attack

- The first step in an HTTPS connection:
  - Client requests the server's public key
- An attacker controlling a router in one of the networks handling your packets can intercept this request and replace it with their own public key
- Attacker then intercepts all subsequent requests, decrypts them and responds with their responses
- It looks like you're talking to the server..
- Certificate Authorities (CA) can fix that

Certificate Authority (CA)

- A CA is a trusted source with a known public key
  - Public key is pre-installed in your browser (Called a root CA)
  - Assume no man-in-the-middle attack during your browser download and installation
- The CA issues certificates for domains and subdomains
  - You verify that you control the domain
  - Send them your public key
  - They send you a certificate
- Certificate includes
  - Your public key
  - Domain name and CA name
  - All algorithms [PKI/Key Exchange/Hashing] Needed to verify and use the public key
  - A **cryptographic signature** of a **hash** of the **certificate** body
  - The signature uses the CA's private key so you can verify with their pre-installed public key that this was in fact issued by the CA
    - Man-in-the-middle cannot fake this without the CA's private key!

Certificate Authority (CA)

- Key chain
  - Not all CA public keys are pre-installed in your browser
  - A CA can have their public key certified by a root CA
  - A domain must provide a key chain that leads to a root CA
- Example key chain
  - Let's Encrypt certificate is signed by DST Root CA
  - Let's Encrypt will sign your certificate
  - Your key chain contains your public key signed by Let's Encrypt and Let's Encrypt's certificate signed by DST
  - Your browser starts with it's installed DST cert to verify the chain
- If a cert cannot be verified by a root CA it is called Self-signed and should not be trusted

Online Payments

- User inputs their credit card information
- Send this information to a payment API
  - They will charge a fee per transaction
    - Ex. Stripe charges 2.9% + 30 cents
- Storing credit cards <- Discussion
  - But you probably don't want to store credit cards
    - Lots of risk!
  - Use PayPal/Amazon Pay/Google Wallet/etc instead

Demo

- Certs installed
- In browser, cert
  - Include root certs
- cert chain and algorithms
- Stanford