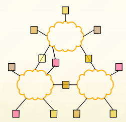
15-440 Distributed Systems
Spring 2014



L-24 Security

Today's Lecture

- Internet security weaknesses
- Establishing secure channels (Crypto 101)
- Key distribution

What is "Internet Security" ?

- Denial-of-Service
- Worms & Viruses
- Traffic modification
- Password Cracking
- Trojan Horse
- DNS Poisoning
- Phishing
- Spyware
- IP Spoofing
- End-host impersonation
- Route Hijacks
- Traffic Eavesdropping
- Spam

Internet Design Decisions: (ie: how did we get here?)

- Origin as a small and cooperative network
  (➔ largely trusted infrastructure)
- Global Addressing
  (➔every sociopath is your next-door neighbor)
- Connection-less datagram service
  (➔can't verify source, hard to protect bandwidth)

Internet Design Decisions: (ie: how did we get here?)

- Anyone can connect
  - (➔ ANYONE can connect)
- Millions of hosts run nearly identical software
  - (➔ single exploit can create epidemic)
- Most Internet users know about as much as Senator Stevens aka "the tubes guy"
  - (➔ God help us all…)

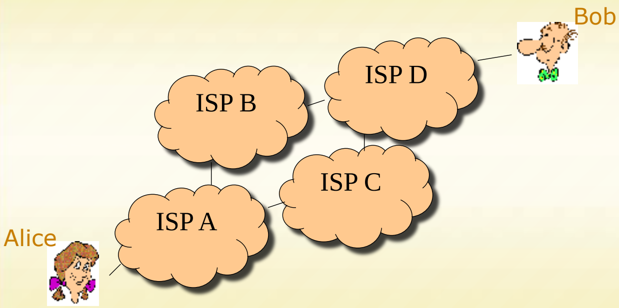
Our "Narrow" Focus

Yes:
1) Creating a "secure channel" for communication

Some:
2) Protecting resources and limiting connectivity

No:
1) Preventing software vulnerabilities & malware, or "social engineering".

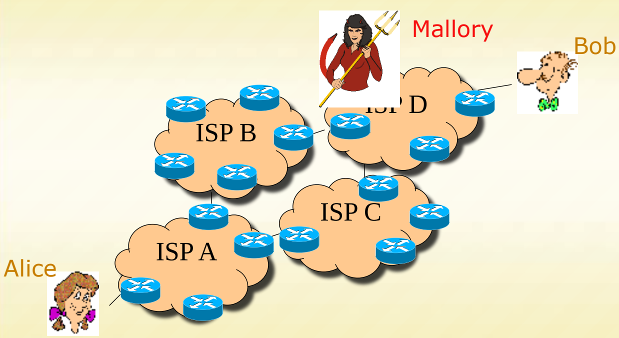
Secure Communication with an Untrusted Infrastructure



What do we need for a secure communication channel?

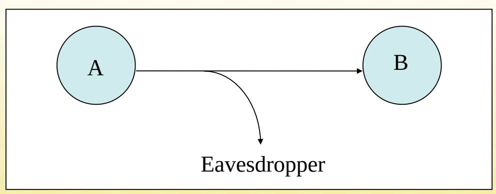
- Authentication (Who am I talking to?)
- Confidentiality (Is my data hidden?)
- Integrity (Has my data been modified?)
- Availability (Can I reach the destination?)

Example: Eavesdropping - Message Interception (Attack on Confidentiality)



Example: Eavesdropping - Message Interception (Attack on Confidentiality)

Unauthorized access to information
Packet sniffers and wiretappers
Illicit copying of files and programs



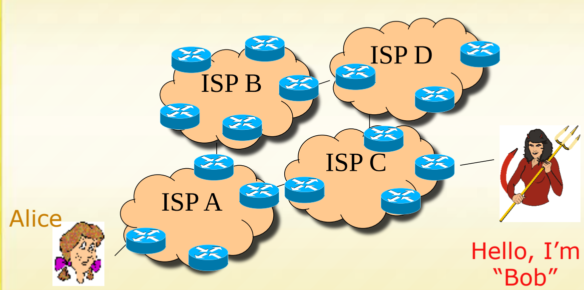
slide derived from original by Nick Feamster

Eavesdropping Attack: Example

- tcpdump with promiscuous network interface
  - On a switched network, what can you see?
- What might the following traffic types reveal about communications?
  - Full IP packets with unencrypted data
  - Full IP packets with encrypted payloads
  - Just DNS lookups (and replies)

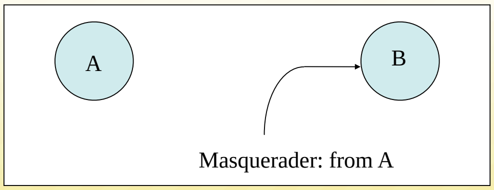
slide derived from original by Nick Feamster

Authenticity Attack - Fabrication



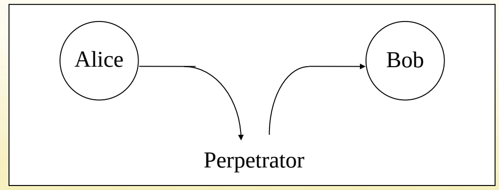
Authenticity Attack - Fabrication

Unauthorized assumption of other's identity
Generate and distribute objects under this identity



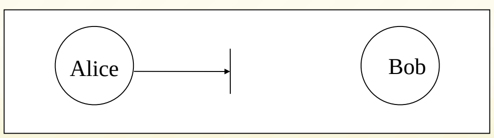
Integrity Attack - Tampering

- Stop the flow of the message
- Delay and optionally modify the message
- Release the message again



Attack on Availability

- Destroy hardware (cutting fiber) or software
- Modify software in a subtle way
- Corrupt packets in transit



- Blatant denial of service (DoS):
  - Crashing the server
  - Overwhelm the server (use up its resource)

Example: Web access

- Alice wants to connect to her bank to transfer some money...
- Alice wants to know ...
  - that she's really connected to her bank. Authentication
  - That nobody can observe her financial data Confidentiality
  - That nobody can modify her request Integrity
  - That nobody can steal her money! (A mix)
- The bank wants to know ...
  - That Alice is really Alice (or is authorized to act for Alice)
  - The same privacy things that Alice wants so they don't get sued or fined by the government.

Today's Lecture

- Internet security weaknesses
- Crypto 101
- Key distribution

Cryptography As a Tool

- Using cryptography securely is not simple
- Designing cryptographic schemes correctly is near impossible.

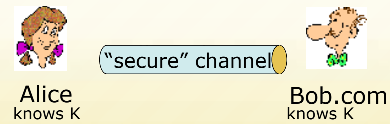
  Today we want to give you an idea of what can be done with cryptography.
  Take a security course if you think you may use it in the future (e.g. 18-487)

Well...

- What tools do we have at hand?
- Hashing
  - e.g., SHA-1
- Secret-key cryptography, aka symmetric key.
  - e.g., AES
- Public-key cryptography
  - e.g., RSA

Secret Key Cryptography

- Given a key k and a message m
  - Two functions: Encryption (E), decryption (D)
  - ciphertext c = E(k, m)
  - plaintext m = D(k, c)
  - Both use the same key k.



But... how does that help with authentication? They both have to know a pre-shared key K before they start!

Symmetric Key: Confidentiality

Motivating Example:
You and a friend share a key K of L random bits, and a message M also L bits long.

Scheme:
You send her the *xor(M,K)* and then they "decrypt" using *xor(M,K)* again.

1) Do you get the right message to your friend?
2) Can an adversary recover the message M?

Symmetric Key: Confidentiality

- One-time Pad (OTP) is secure but usually impractical
  - Key is as long at the message
  - Keys cannot be reused (why?)

In practice, two types of ciphers are used that require only constant key length:

Stream Ciphers:

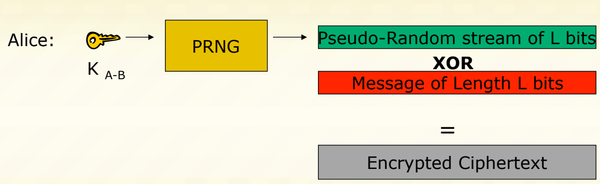
Ex: RC4, A5

Block Ciphers:

Ex: DES, AES, Blowfish

Symmetric Key: Confidentiality

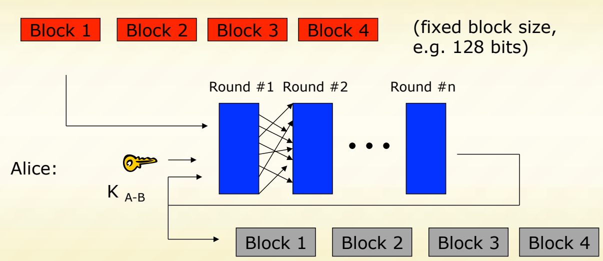
- Stream Ciphers (ex: RC4)



Bob uses K_{A-B} as PRNG seed, and XORs encrypted text to get the message back (just like OTP).

Symmetric Key: Confidentiality

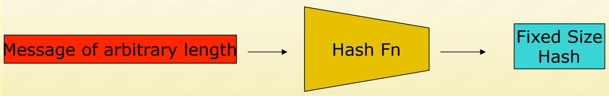
- Block Ciphers (ex: AES)



Bob breaks the ciphertext into blocks, feeds it through decryption engine using K_{A-B} to recover the message.

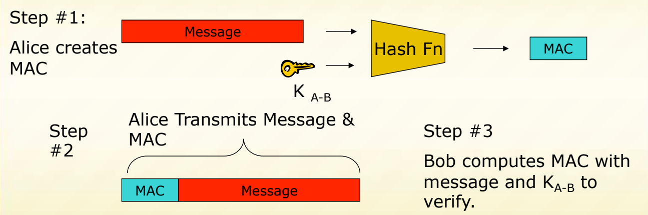
Symmetric Key: Integrity

- Background: Hash Function Properties
  - Consistent
    hash(X) always yields same result
  - One-way
    given X, can't find Y s.t. hash(Y) = X
  - Collision resistant
    given hash(W) = Z, can't find X such that hash(X) = Z



Symmetric Key: Integrity

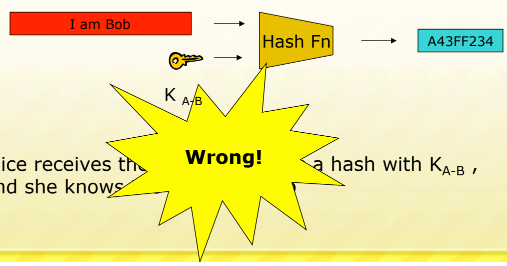
- Hash Message Authentication Code (HMAC)



Why is this secure? How do properties of a hash function help us?

Symmetric Key: Authentication

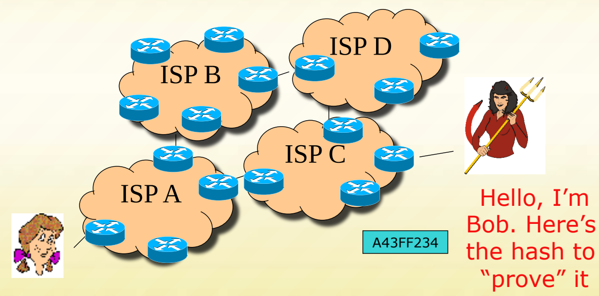
- You already know how to do this!
  (hint: think about how we showed integrity)



Alice receives th... a hash with K_{A-B} , and she knows...

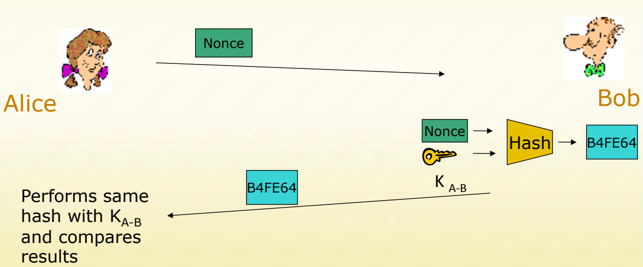
Symmetric Key: Authentication

What if Mallory overhears the hash sent by Bob, and then "replays" it later?



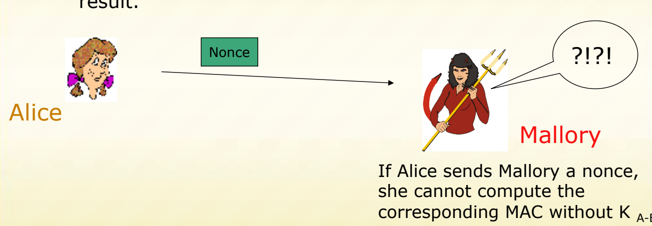
Symmetric Key: Authentication

- A "Nonce"
  - A random bitstring used only once. Alice sends nonce to Bob as a "challenge". Bob Replies with "fresh" MAC result.



Symmetric Key: Authentication

- A "Nonce"
  - A random bitstring used only once. Alice sends nonce to Bob as a "challenge". Bob Replies with "fresh" MAC result.



If Alice sends Mallory a nonce, she cannot compute the corresponding MAC without K_{A-B}

Symmetric Key Crypto Review

- Confidentiality: Stream & Block Ciphers
- Integrity: HMAC
- Authentication: HMAC and Nonce

Questions??

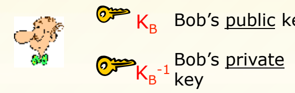
Are we done? Not Really:

1)Number of keys scales as O(n^2)

2)How to securely share keys in the first place?

Asymmetric Key Crypto:

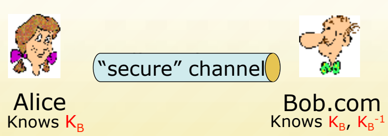
- Instead of shared keys, each person has a "key pair"



- The keys are inverses, so: $K_B^{-1}(K_B(m)) = m$

Asymmetric/Public Key Crypto:

- Given a key *k* and a message *m*
  - Two functions: Encryption (E), decryption (D)
  - ciphertext $c = E(K_B, m)$
  - plaintext $m = D(K_B^{-1}, c)$
  - Encryption and decryption use *different* keys!



But how does Alice know that K_B means "Bob"?

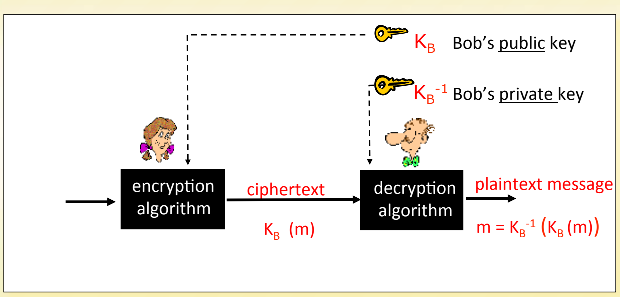
Asymmetric Key Crypto:

- It is believed to be computationally unfeasible to derive K_B^{-1} from K_B or to find any way to get M from $K_B(M)$ other than using K_B^{-1} .

➔K_B can safely be made public.

Note: We will not detail the computation that $K_B(m)$ entails, but rather treat these functions as black boxes with the desired properties.

Asymmetric Key: Confidentiality

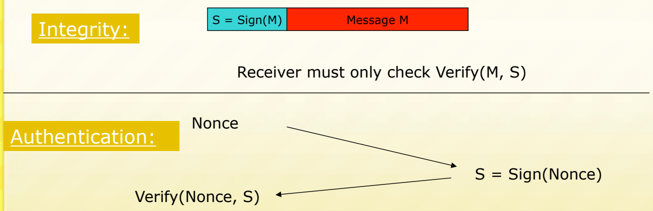


Asymmetric Key: Sign & Verify

- If we are given a message M, and a value S such that $K_B(S) = M$, what can we conclude?
- The message must be from Bob, because it must be the case that $S = K_B^{-1}(M)$, and only Bob has K_B^{-1} !
- This gives us two primitives:
  - Sign (M) = $K_B^{-1}(M)$ = Signature S
  - Verify (S, M) = test($K_B(S)$ == M)

Asymmetric Key: Integrity & Authentication

- We can use Sign() and Verify() in a similar manner as our HMAC in symmetric schemes.



Asymmetric Key Review:

- Confidentiality: Encrypt with Public Key of Receiver
- Integrity: Sign message with private key of the sender
- Authentication: Entity being authenticated signs a nonce with private key, signature is then verified with the public key

But, these operations are computationally expensive*

The Great Divide

	Symmetric Crypto: (Private key) Example: AES	Asymmetric Crypto: (Public key) Example: RSA
Requires a pre-shared secret between communicating parties?	Yes	No
Overall speed of cryptographic operations	Fast	Slow

Today's Lecture

- Internet security weaknesses
- Crypto 101
- Key distribution

One last "little detail"…

How do I get these keys in the first place?? Remember:

- Symmetric key primitives assumed Alice and Bob had already shared a key.
- Asymmetric key primitives assumed Alice knew Bob's public key.

This may work with friends, but when was the last time you saw Amazon.com walking down the street?

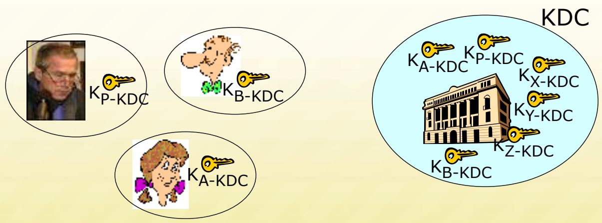
Symmetric Key Distribution

- How does Andrew do this?

Andrew Uses Kerberos, which relies on a Key Distribution Center (KDC) to establish shared symmetric keys.

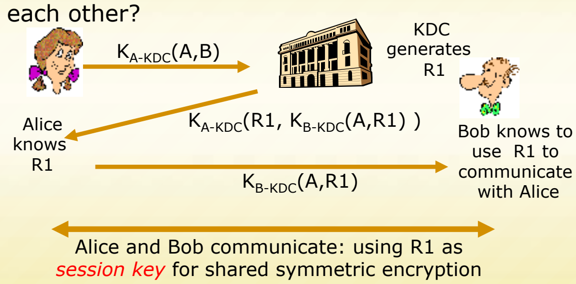
Key Distribution Center (KDC)

- Alice, Bob need shared <u>symmetric key</u>.
- KDC: server shares different secret key with *each* registered user (many users)
- Alice, Bob know own symmetric keys, $K_{A\text{-}KDC}$ $K_{B\text{-}KDC}$, for communicating with KDC.



Key Distribution Center (KDC)

Q: How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?



How Useful is a KDC?

- Must always be online to support secure communication
- KDC can expose our session keys to others!
- Centralized trust and point of failure.

In practice, the KDC model is mostly used within single organizations (e.g. Kerberos) but not more widely.

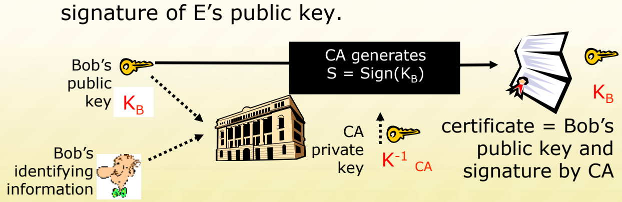
The Dreaded PKI

- Definition:
  Public Key Infrastructure (PKI)

1) A system in which "roots of trust" authoritatively bind public keys to real-world identities
2) A significant stumbling block in deploying many "next generation" secure Internet protocol or applications.

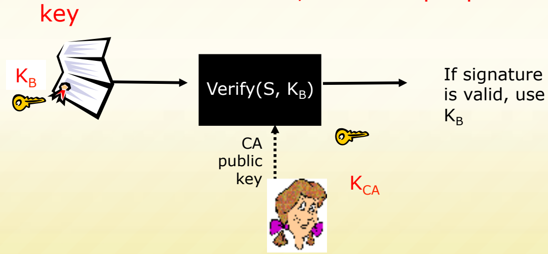
Certification Authorities

- Certification authority (CA): binds public key to particular entity, E.
- An entity E registers its public key with CA.
  - E provides "proof of identity" to CA.
  - CA creates certificate binding E to its public key.
  - Certificate contains E's public key AND the CA's signature of E's public key.



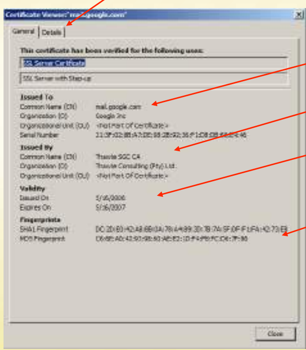
Certification Authorities

- When Alice wants Bob's public key:
  - Gets Bob's certificate (Bob or elsewhere).
  - Use CA's public key to verify the signature within Bob's certificate, then accepts public key



Certificate Contents

- info algorithm and key value itself (not shown)

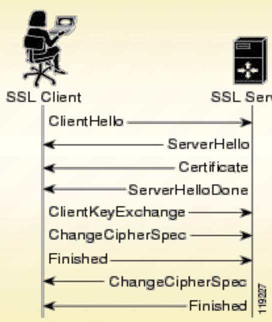


- Cert owner
- Cert issuer
- Valid dates
- Fingerprint of signature

Transport Layer Security (TLS) aka Secure Socket Layer (SSL)

- Used for protocols like HTTPS
- Special TLS socket layer between application and TCP (small changes to application).
- Handles confidentiality, integrity, and authentication.
- Uses "hybrid" cryptography.

Setup Channel with TLS "Handshake"



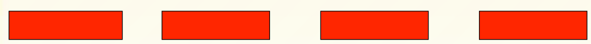
Handshake Steps:

1) Clients and servers negotiate exact cryptographic protocols
2) Client's validate public key certificate with CA public key.
3) Client encrypt secret random value with servers key, and send it as a challenge.
4) Server decrypts, proving it has the corresponding private key.
5) This value is used to derive symmetric session keys for encryption & MACs.

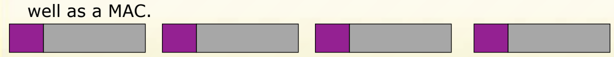
How TLS Handles Data

1) Data arrives as a stream from the application via the TLS Socket

2) The data is segmented by TLS into chunks



3) A session key is used to encrypt and MAC each chunk to form a TLS "record", which includes a short header and data that is encrypted, as well as a MAC.



4) Records form a byte stream that is fed to a TCP socket for transmission.

Analysis

- Public key lets us take the trusted third party offline:
  - If it's down, we can still talk!
  - But we trade-off ability for fast revocation
    - If server's key is compromised, we can't revoke it immediately...
    - Usual trick:
      - Certificate expires in, e.g., a year.
      - Have an on-line revocation authority that distributes a revocation list. Kinda clunky but mostly works, iff revocation is rare. Clients fetch list periodically.
- Better scaling: CA must only sign once... no matter how many connections the server handles.
- If CA is compromised, attacker can trick clients into thinking they're the real server.

Important Lessons

- Symmetric (pre-shared key, fast) and asymmetric (key pairs, slow) primitives provide:
  - Confidentiality
  - Integrity
  - Authentication
- "Hybrid Encryption" leverages strengths of both.
- Great complexity exists in securely acquiring keys.
- Crypto is hard to get right, so use tools from others, don't design your own (e.g. TLS).

Forward secrecy

- In KDC design, if key $K_{server\text{-}KDC}$ is compromised a year later,
  - from the traffic log, attacker can extract session key (encrypted with auth server keys).
  - attacker can decode all traffic retroactively.
- In SSL, if CA key is compromised a year later,
  - Only new traffic can be compromised. Cool…
- But in SSL, if server's key is compromised...
  - Old logged traffic can still be compromised...

Diffie-Hellman Key Exchange

- Different model of the world: How to generate keys between two people, securely, no trusted party, even if someone is listening in.

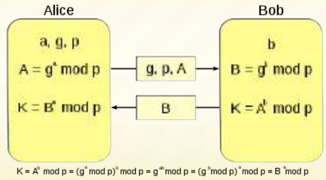


image from wikipedia

- This is cool. But: Vulnerable to man-in-the-middle attack. Attacker pair-wise negotiates keys with each of A and B and decrypts traffic in the middle. No authentication...

Authentication?

- But we already have protocols that give us authentication!
  - They just happen to be vulnerable to disclosure if long-lasting keys are compromised later...
- Hybrid solution:
  - Use diffie-hellman key exchange with the protocols we've discussed so far.
  - Auth protocols prevent M-it-M attack if keys aren't yet compromised.
  - D-H means that an attacker can't recover the real session key from a traffic log, even if they can decrypt that log.
  - Client and server discard the D-H parameters and session key after use, so can't be recovered later.
- This is called "perfect forward secrecy". Nice property.

Big picture, usability, etc.

- public key infrastructures (PKI)s are great, but have some challenges…
  - Yesterday, we discussed how your browser trusts many, many different CAs.
  - If any one of those is compromised, an attacker can convince your browser to trust their key for a website... like your bank.
  - Often require payment, etc.
- Alternative: the "ssh" model, which we call "trust on first use" (TOFU). Sometimes called "prayer."

Signatures

- Assume Alice *does* know that Bob's key is *K*...
  - Let's build a more powerful primitive: A digital signature
  - s = signature(K, M)
    - *s* is ideally small, while M might be huge
    - Only the holder of key K can create s
      - In other words, K is proving that it "said" M
- Using secret key crypto, pre-shared key K:
  - HMAC(K, m) ("Hash-based Message Authentication Code")
    - H((K xor opad) | H((K xor ipad) | m))
    - Where "opad" and "ipad" are globally known constants that just mix the bits up
- Why so complex? Why not just...
  - H(key | message) for example?
    - Concatenation attack! Many hash functions can be iterated...
    - H(m1 | m2) = f(H(m1), m2)
    - So if you sent me a MAC for "hi!" I could turn it into "hi! I want to drop the class"
  - H(message, key) is better, but suffers some weaknesses for collision resistance.

Uses of HMAC

- Drawback to previous: Had to have a pre-shared key.
  - HMAC is used all over the place; hugely useful! (Don't implement it yourself, lots of libraries).
- A common use:
  - I create a message
  - I give it to you
  - You give it back to me later
  - I want to verify that it's what I originally gave you
- Why would I want to do this?

Web page authentication

- Low-security example:
  - User logs into the NY Times website using username + password.
    - That login is protected using SSL
    - SSL is expensive! 10-100x more CPU to use SSL than unencrypted HTTP
  - Want to let them return and browse articles without logging in and without SSL
    - (But only browse articles - low security requirement)
- How can we accomplish this?
  - Cookies!