Applied Cryptography and Network Security

Adam J. Lee
adamlee@cs.pitt.edu
6111 Sennott Square

Lecture #16: Real Time Security
February 27, 2014

University of Pittsburgh

Outline

Today we're going to talk about real time security issues

Issues related to session key disclosure

- Perfect forward secrecy
- Forward secrecy
- Backward secrecy

Deniable authentication protocols

Denial of service

- Computational puzzles
- Guided tour puzzles

Real time what?

Real time security is a "catch all" term defined in the textbook

- i.e., Any interactive security protocol
- So basically most authentication protocols

Note that the security protocols that we've discussed to date can be viewed as "soft" real time protocols

- We require liveness (read: progress) in the system
- However, we do not usually have hard deadlines

We'll look at two classes of "real time" properties today:

Key negotiation properties

Timeliness properties

Session key recap

Parties in cryptographic protocols typically have long-lived secrets

- Kerberos: K_A shared between Alice and the KDC
- PKI: Public/private key pairs (k_A, k_A^{-1})

Even so, it is good practice to generate new session keys regularly (Why?)

- Help limit compromises due to overuse
- Prevent breaks of one key from compromising all sessions
- …

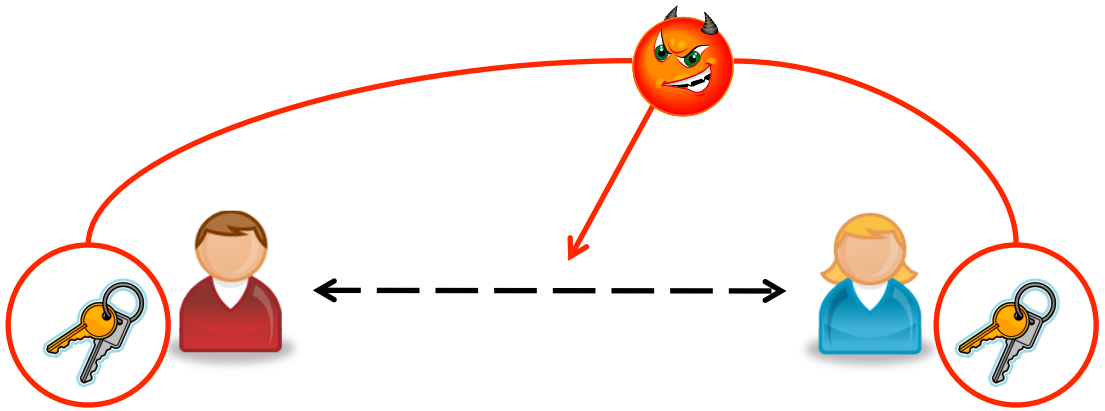
In general, session keys should be used for only a single session

Typically both parties contribute to the session keys for a protocol (Why?)

- If either party can generate "good" pseudorandom numbers, the derived key will be reasonably secure
- Less likely to have replay attacks lead to successful impersonations

Ideally, the session keys that we choose should have a property known as perfect forward secrecy

A key exchange protocol is said to have perfect forward secrecy if an adversary that (i) watches all messages exchanged and (ii) later compromises both parties cannot recover the agreed-upon session key



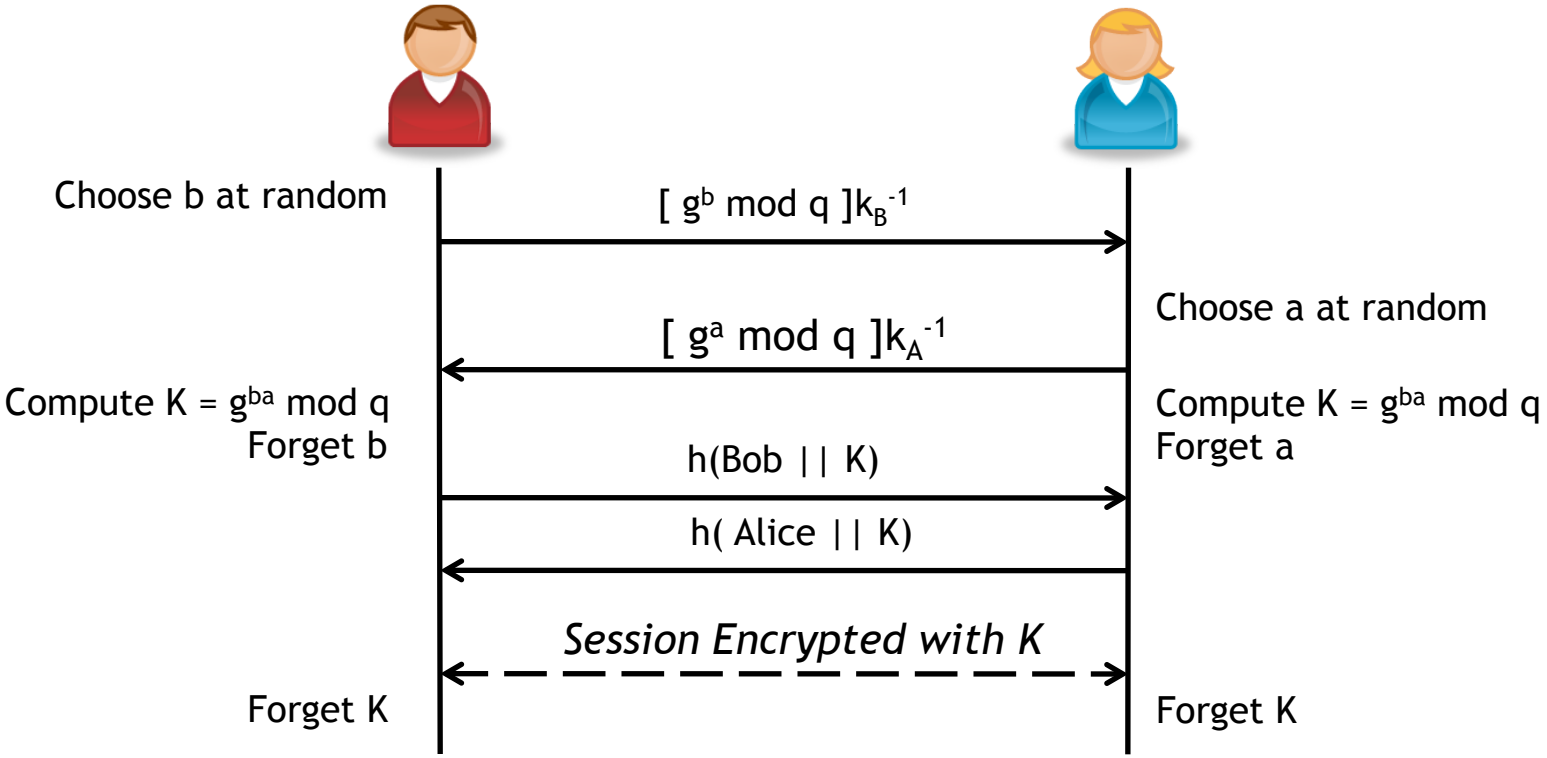
How do we achieve perfect forward secrecy?

- Generate a temporary secret
- Use only information that is not stored at the node long-term

---

This sounds good, but what protocols actually give us this property?

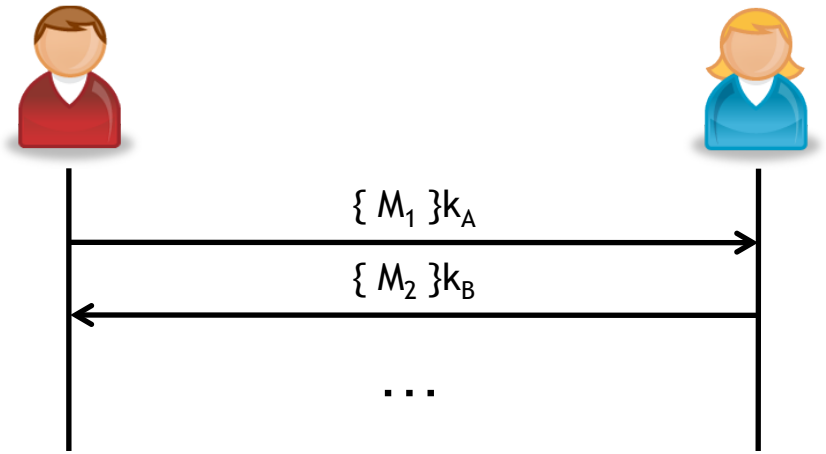
Signed Diffie-Hellman key exchange!



Question: Why does this protocol provide perfect forward secrecy?

- Adversary learns g^a mod q and g^b mod q by snooping
- Adversary learns k_B^{-1} and k_A^{-1} by compromise
- The important pieces are a, b, and K
  - These are not stored in the long term

Not all seemingly reasonable key exchange protocols provide perfect forward secrecy



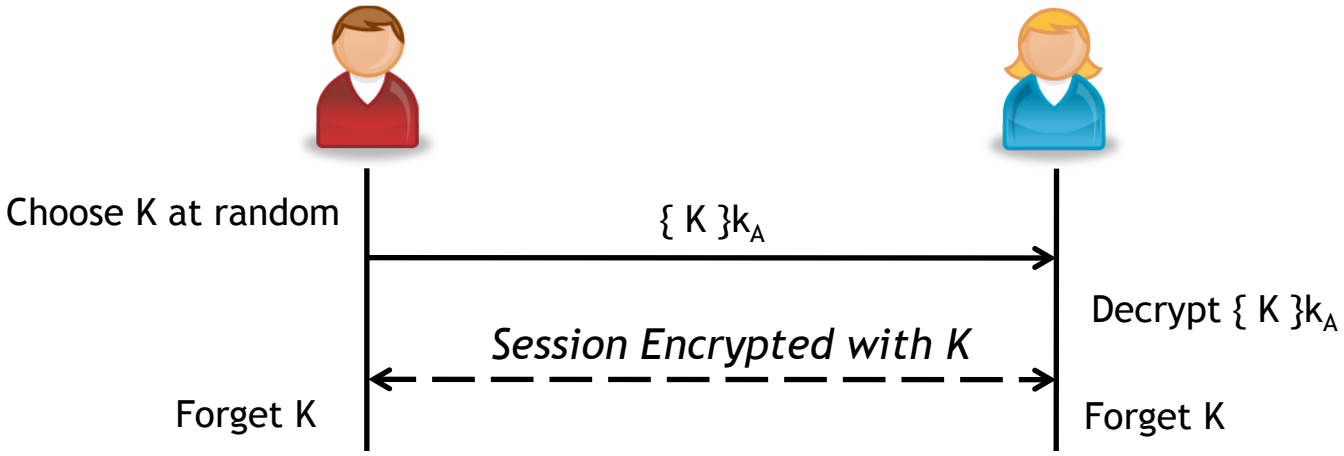
This protocol is inefficient and fails to provide perfect forward secrecy

What does the adversary learn?

- Monitoring: Nothing
- Compromising Alice and Bob: k_A^{-1} and k_B^{-1}

This compromises the protocol, as all messages can be recovered

Not all seemingly reasonable key exchange protocols provide perfect forward secrecy



This completely reasonable hybrid cryptosystem also fails to provide perfect forward secrecy (Why?)

What does the adversary learn?

- Monitoring: { K }k_A
- Compromising Alice and Bob: k_A^{-1} and k_B^{-1}

Given k_A^{-1} and { K }k_A, the adversary can recover K even though Alice and Bob have both forgotten it!

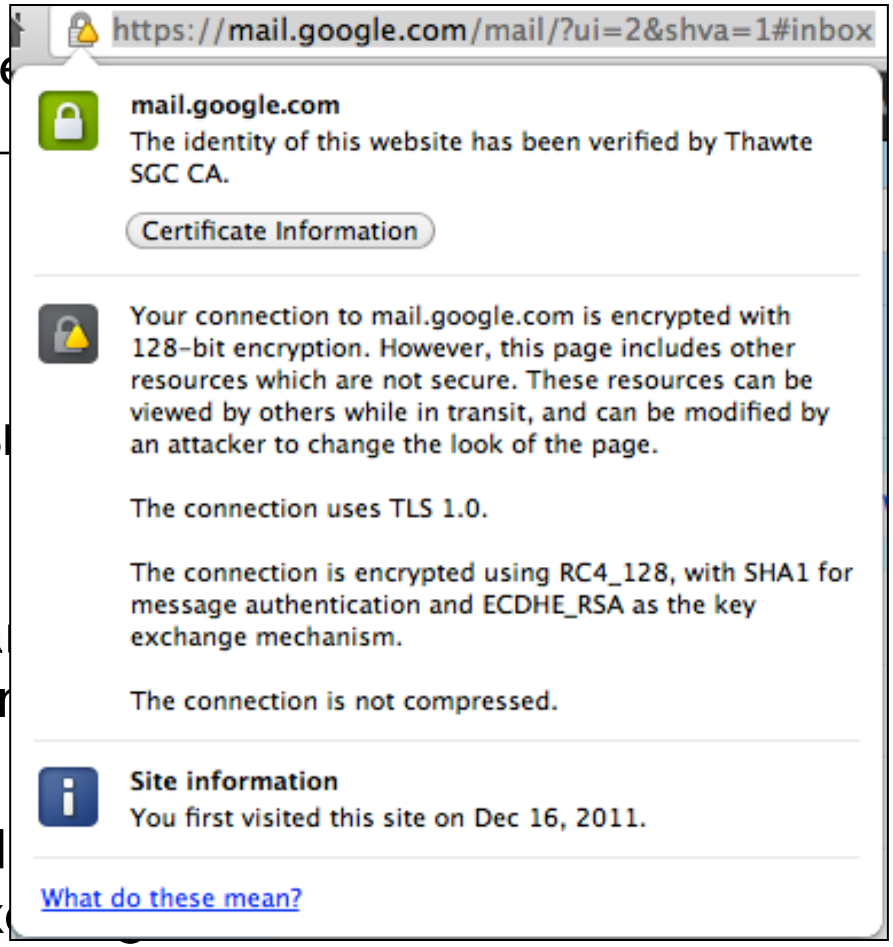
The previous protocol is similar to the “RSA key exchange” method supported in SSL/TLS

Many TLS cipher suites use RSA ke

- TLS_RSA_EXPORT_WITH_RC4_40_
- TLS_RSA_WITH_RC4_128_MD5
- TLS_RSA_WITH_RC4_128_SHA
- TLS_RSA_WITH_IDEA_CBC_SHA
- TLS_RSA_WITH_DES_CBC_SHA
- TLS_RSA_WITH_3DES_EDE_CBC_S
- …

Using these cipher suites, one pa
with the RSA key of the other par

In November 2011, Google added
using the ECDHE family of key ex



So far, we've only talked about pairwise keys… What about group keys?

Scenario: Secure chat rooms

Assume that we want a way for geographically distributed people to carry out secure conversations.

By secure, we mean that only parties that are part of the chat room can understand what is being said, even though all messages are exchanged over the public Internet.

Simple idea: Set up a shared key for the chat room, and encrypt all messages using this key

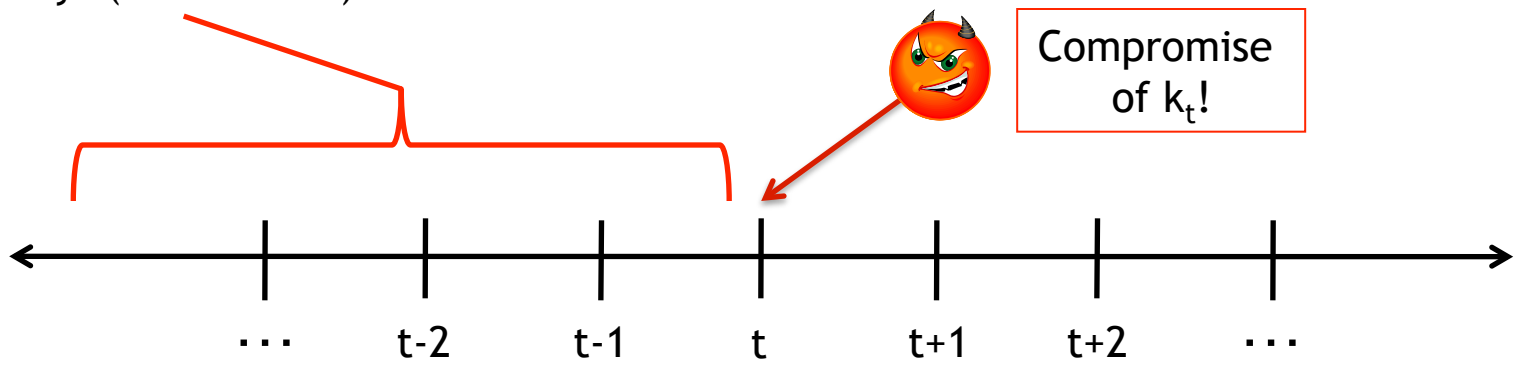
Questions:

- Should people that join the group be able to decrypt old messages?
- Should people that leave the group be able to decrypt new messages?

Forward secrecy protects old information

A group communication scheme has forward security if learning the key k_t for time t does not reveal any information about keys k_i for all $i < t$.

Previous keys (and secrets) are safe

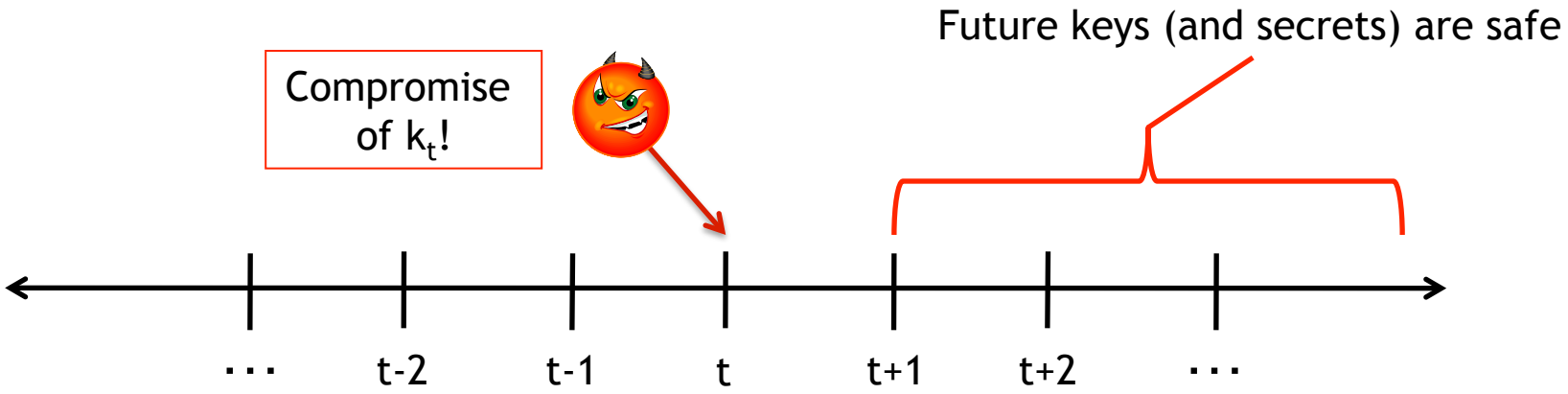


Informally: Later compromise does not reveal earlier keys

How can we achieve forward secrecy?

Backward secrecy is the complement of forward secrecy

A group communication scheme has backward security if learning the key k_t for time t does not reveal any information about keys k_i for all i > t.



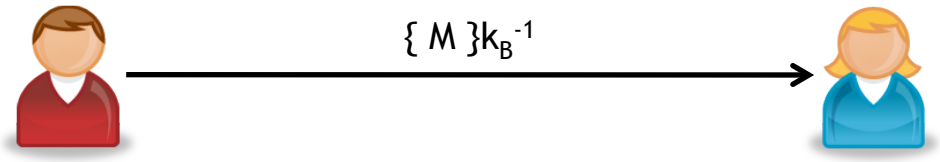
Informally: Earlier compromise does not reveal later keys

Question: Does our previous protocol also provide backward secrecy?

Deniability is another interesting property of authentication/key exchange protocols

Recall that a digitally signed message provides non-repudiability

- Alice can use k_B to verify that Bob signed the message
- But so can anyone else!

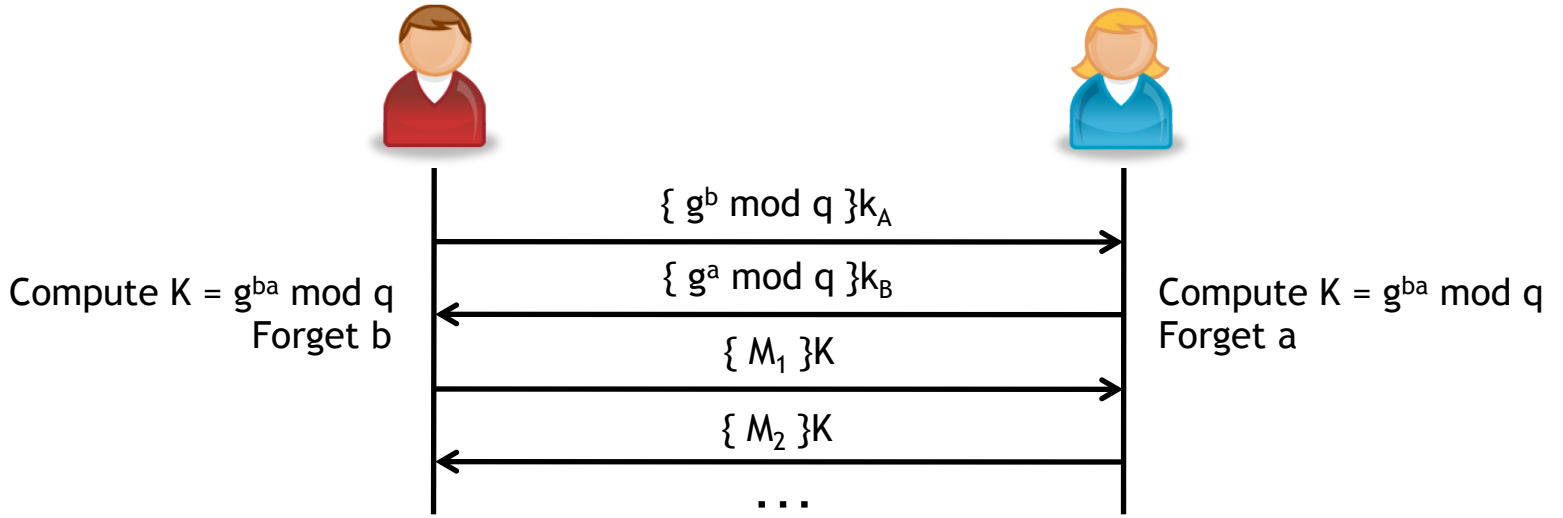


Sometimes, we would like Alice to be able to verify the authenticity of a message without being able to convince a third party that the message came from Bob

In essence, Bob would like some form of plausible deniability

More formally, a protocol between two parties provides plausible deniability if the transcript of messages exchanged could have been generated by a single party

How can we provide plausible deniability?



This protocol mutually authenticates Alice and Bob

- Alice knows that only Bob could have derived K
- Bob knows that only Alice could have derived K
- Each party knows which M_is they sent

Anyone could have generated the transcript!

- Only need k_A and k_B, which are public
- The rest is random numbers!

Lesson: Causality is important. Notions of "I sent" or "I received" are *not* part of the transcript, but are part of real life.

Computer security is typically defined with respect to three types of properties

How do I ensure that my secrets remain secret?

Confidentiality

Can I trust the services that I use?

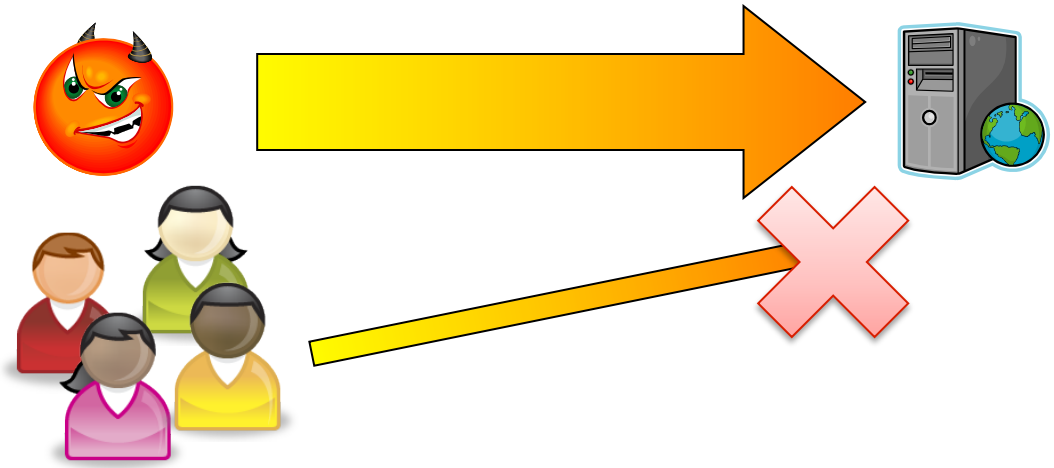
Integrity

Availability

Am I able to do what I need to do?

Denial of Service (DoS)

A denial of service (DoS) attack occurs when a malicious client is able to overwhelm a server, thereby preventing service to legitimate clients



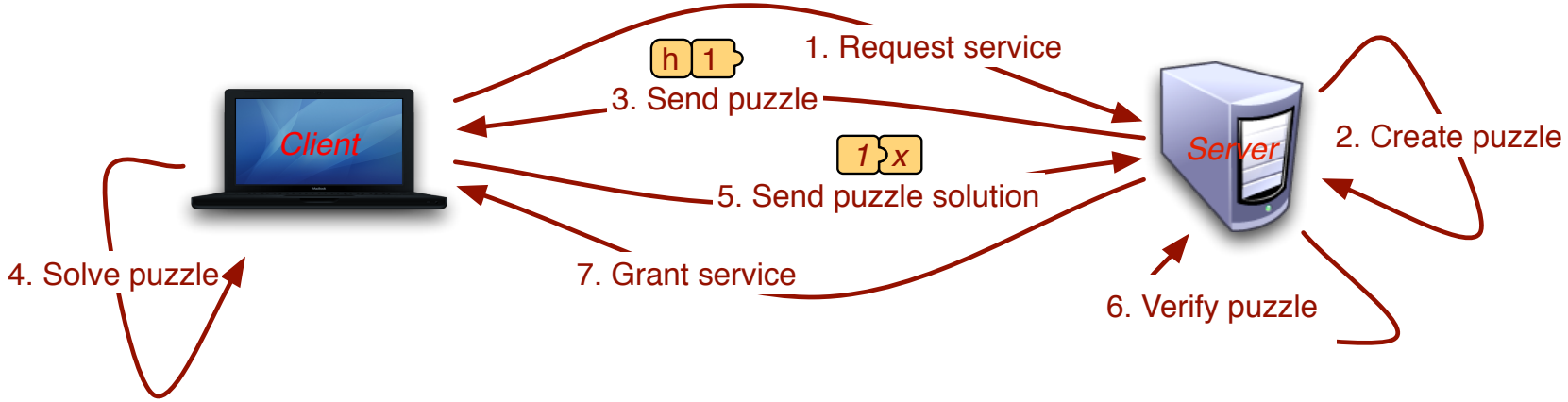
Denial of service attacks are typically abusing a resource disparity

- Easy to generate lots of network requests, hard to maintain server state
- Easy to create random application data, decryption takes time/cycles
- …

Avoiding resource disparity is a good design principle, but is not always easy to do in practice

Computational puzzles can be used to mitigate DoS attacks

Idea: Make clients pay for their requests by solving a hard puzzle first



Computational puzzles must satisfy three requirements:

1. Puzzles should be easy for the server to generate
2. Puzzles should be hard for the client to solve
3. Puzzle solutions should be easy for the server to verify

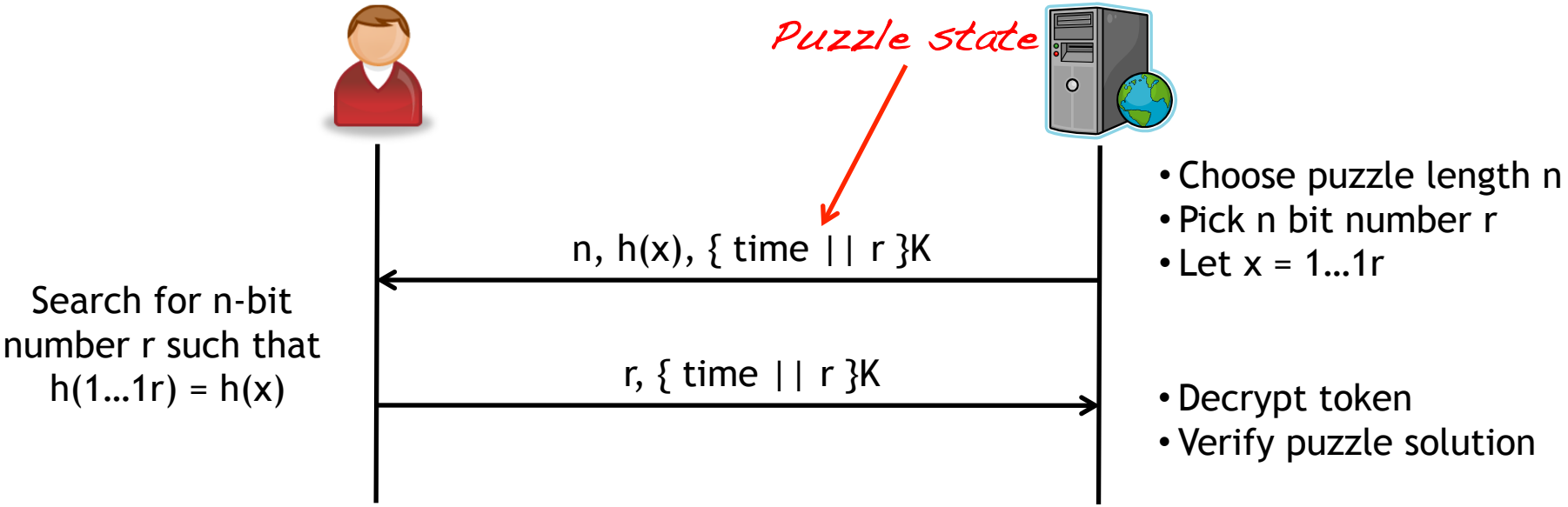
Question: Why do computational puzzles have these requirements?

Example: Hash inversion

Intuition: It should be very hard to invert a cryptographic hash function

- Recall that "hard" means $O(2^m)$ work where m is the hash bit length

Hash inversion puzzles work as follows:



Note: Puzzle hardness can be tuned by using different length values (n)

Question: Why is the puzzle state offloaded to the client?

Discussion

When would computational puzzles be effective for mitigating DoS attacks? When are they ineffective?

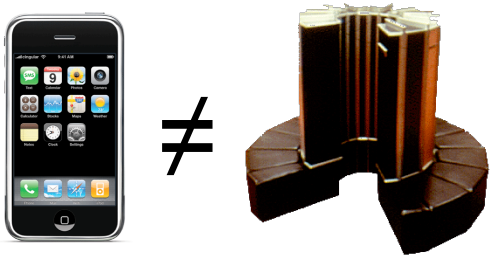
Strengths and weaknesses of computational puzzles

When do computational puzzles work well?

- All clients have approximately similar computational ability
- Puzzles cannot be parallelized between multiple clients
- Puzzles can be efficiently generated

Unfortunately, computational puzzles are not a panacea…

- Not all clients are created equal
- Clients have better things to do than burn cycles solving puzzles
- Attackers often have a means of parallelizing puzzles



Recent work in our department is attempting to address this problem

Observation: Attackers can fairly easily control

- Their own computational abilities
- A subset of nodes in the network (e.g., via compromise)
- The quality of their connection to the network

However, they cannot control the overall latency characteristics of routes that traverse segments of the Internet

Rather than asking clients to solve computational puzzles, we can ask them to provide evidence that they have carried out an ordered traversal of some number of nodes

Guided tour puzzles do exactly this!

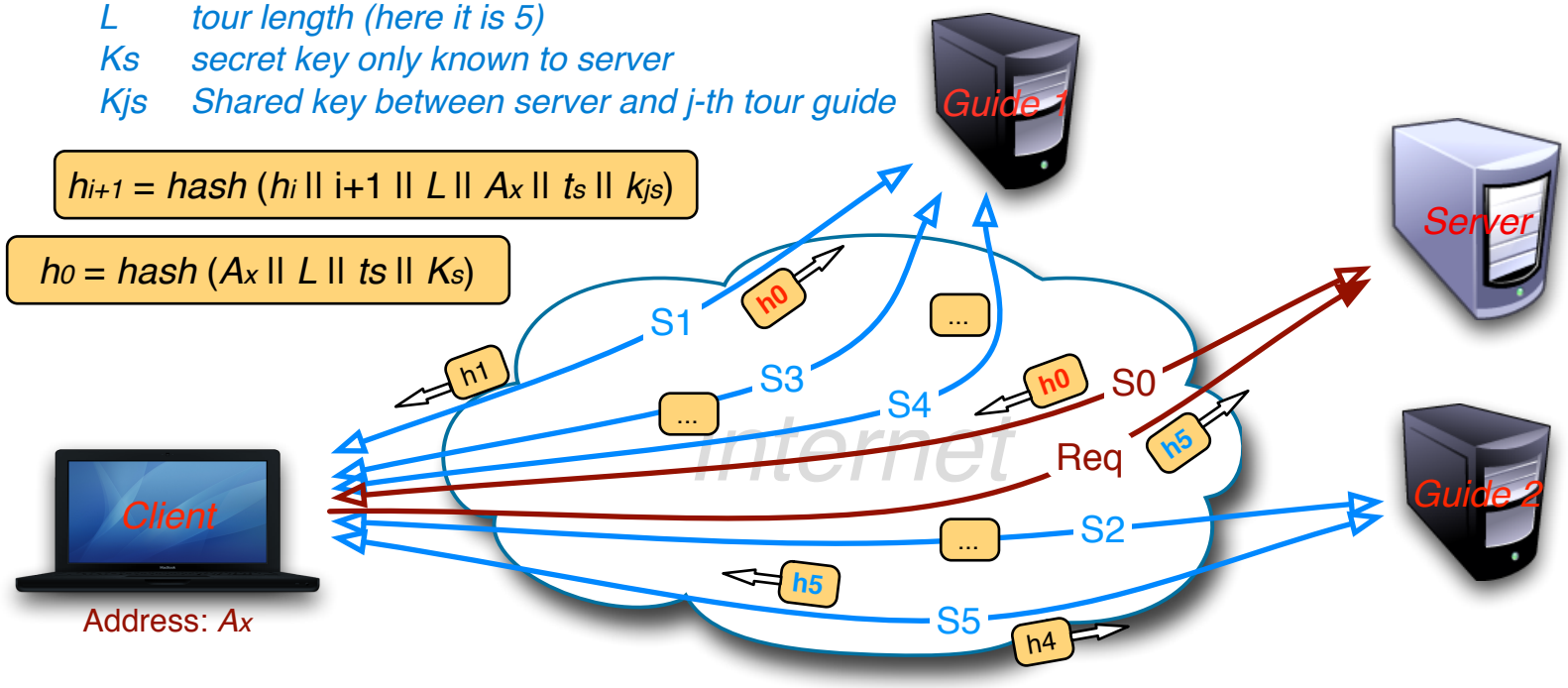
- Puzzle: An ordered list of tour guides to contact
- Solution: A serial computation carried out by these nodes in order

How do guided tours work?

L tour length (here it is 5)
Ks secret key only known to server
Kjs Shared key between server and j-th tour guide

$h_{i+1} = hash(h_i \,||\, i+1 \,||\, L \,||\, A_x \,||\, t_s \,||\, k_{js})$

$h_0 = hash(A_x \,||\, L \,||\, t_s \,||\, K_s)$



Guided tour puzzles are efficient to check

- Next tour guide chosen using a single hash operation
- A length n tour is checked using n hash operations

Cheating is hard

- Cannot guess next tour guide without knowing the secret key
- Cannot control the delay characteristics of the Internet

Conclusions

Today we talked about a variety of real time issues

Session key security issues include:

- Perfect forward secrecy: Session keys safe even if long term keys compromised
- Forward secrecy: Compromising current key does not compromise previous keys
- Backward secrecy: Compromising current key does not lead to the compromise of future keys

Deniable protocols have completely forgeable transcripts, yet still provide authentication, confidentiality, and integrity protection

DoS attacks impact system availability

Puzzles can be used to mitigate DoS attacks