

CSE331: Introduction to Networks and Security

Lecture 26

Fall 2006



Announcements

- Project 3 is due next Monday, November 20th
- Plan for today:
 - Diffie-Hellman key exchange protocol
 - Cryptographic protocols (in general)
 - Authentication protocols



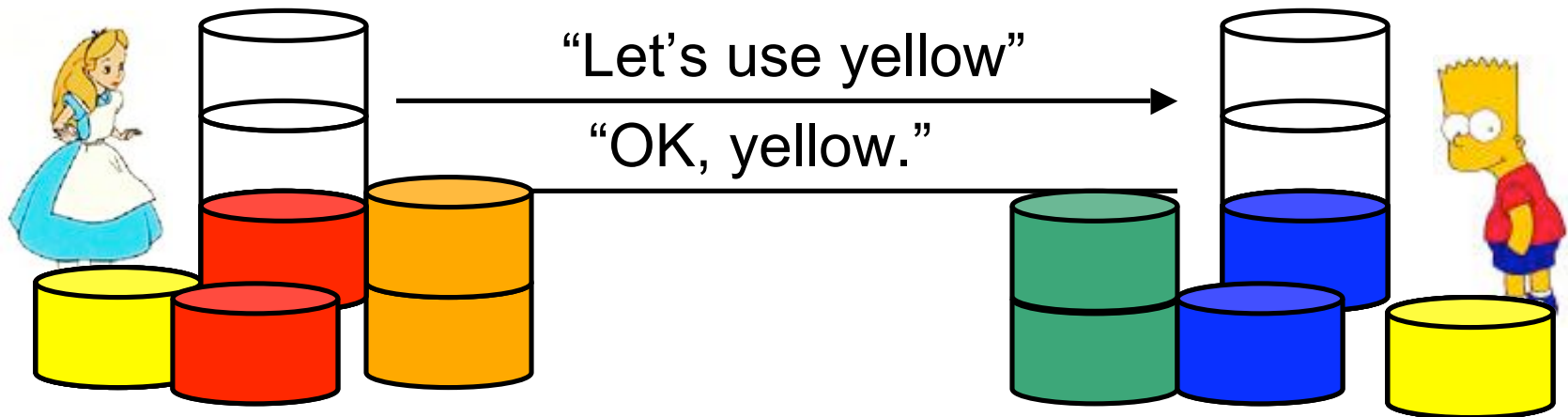
Diffie-Hellman Key Exchange

- Problem with shared-key systems:
Distributing the shared key
- Suppose that Alice and Bart want to agree on a secret (i.e. a key)
 - Communication link is public
 - They don't already share any secrets

Diffie-Hellman by Analogy: Paint

Alice

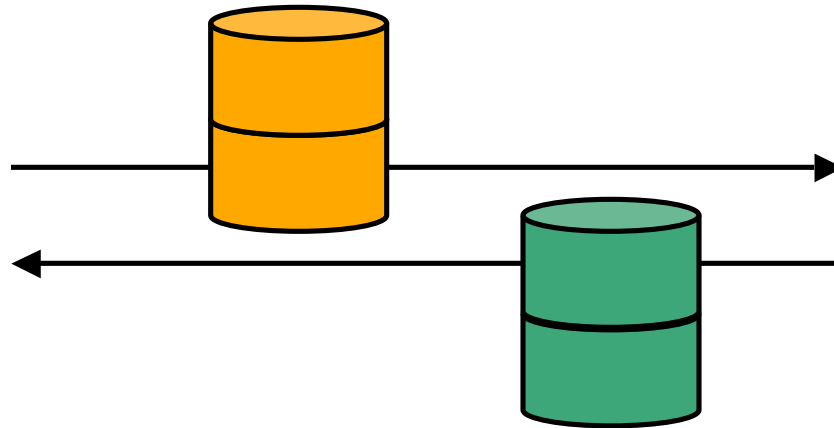
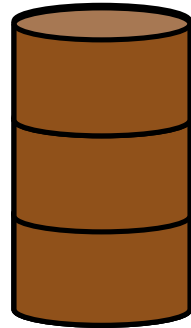
Bart



1. Alice & Bart decide on a public color, and mix one liter of that color.
2. They each choose a random secret color, and mix two liters of their secret color.
3. They keep one liter of their secret color, and mix the other with the public color.

Diffie-Hellman by Analogy: Paint

Alice



Bart



4. They exchange the mixtures over the public channel.
5. When they get the other person's mixture, they combine it with their retained secret color.
6. The secret is the resulting color: Public + Alice's + Bart's

Diffie-Hellman Key Exchange

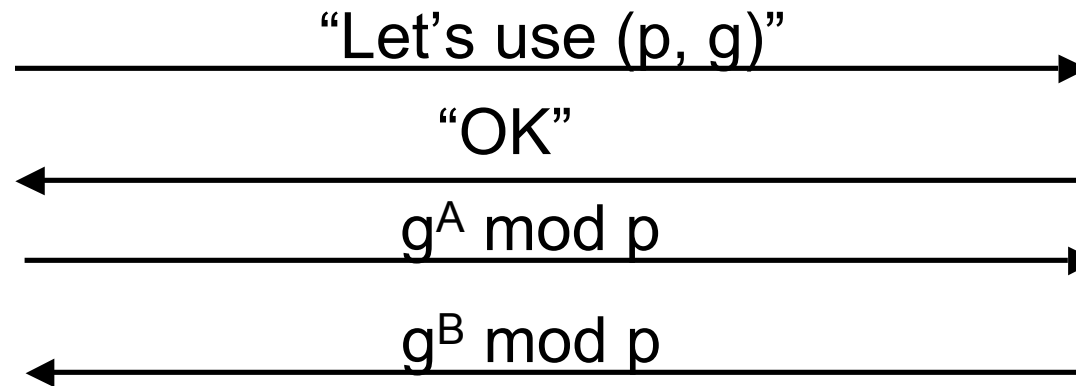
- Choose a prime p (publicly known)
 - Should be about 512 bits or more
- Pick $g < p$ (also public)
 - g must be a *primitive root* of p .
 - A primitive root generates the finite field p .
 - Every n in $\{1, 2, \dots, p-1\}$ can be written as $g^k \pmod p$
 - Example: 2 is a primitive root of 5
 - $2^0 = 1$ $2^1 = 2$ $2^2 = 4$ $2^3 = 3 \pmod 5$
 - Intuitively means that it's hard to take logarithms base g because there are many candidates.

Diffie-Hellman

Alice



Bart



1. Alice & Bart decide on a public prime p and primitive root g .
2. Alice chooses secret number A . Bart chooses secret number B
3. Alice sends Bart $g^A \bmod p$.
4. The shared secret is $g^{AB} \bmod p$.

Details of Diffie-Hellman

- Alice computes $g^{AB} \bmod p$ because she knows A :
 - $g^{AB} \bmod p = (g^B \bmod p)^A \bmod p$
- An eavesdropper gets $g^A \bmod p$ and $g^B \bmod p$
 - They can easily calculate $g^{A+B} \bmod p$ but that doesn't help.
 - The problem of computing discrete logarithms (to recover A from $g^A \bmod p$) is hard.

Example

- Alice and Bart agree that $q=71$ and $g=7$.
- Alice selects a private key $A=5$ and calculates a public key $g^A \equiv 7^5 \equiv 51 \pmod{71}$. She sends this to Bart.
- Bart selects a private key $B=12$ and calculates a public key $g^B \equiv 7^{12} \equiv 4 \pmod{71}$. He sends this to Alice.
- Alice calculates the shared secret:
 $S \equiv (g^B)^A \equiv 4^5 \equiv 30 \pmod{71}$
- Bart calculates the shared secret
 $S \equiv (g^A)^B \equiv 51^{12} \equiv 30 \pmod{71}$

Why Does it Work?

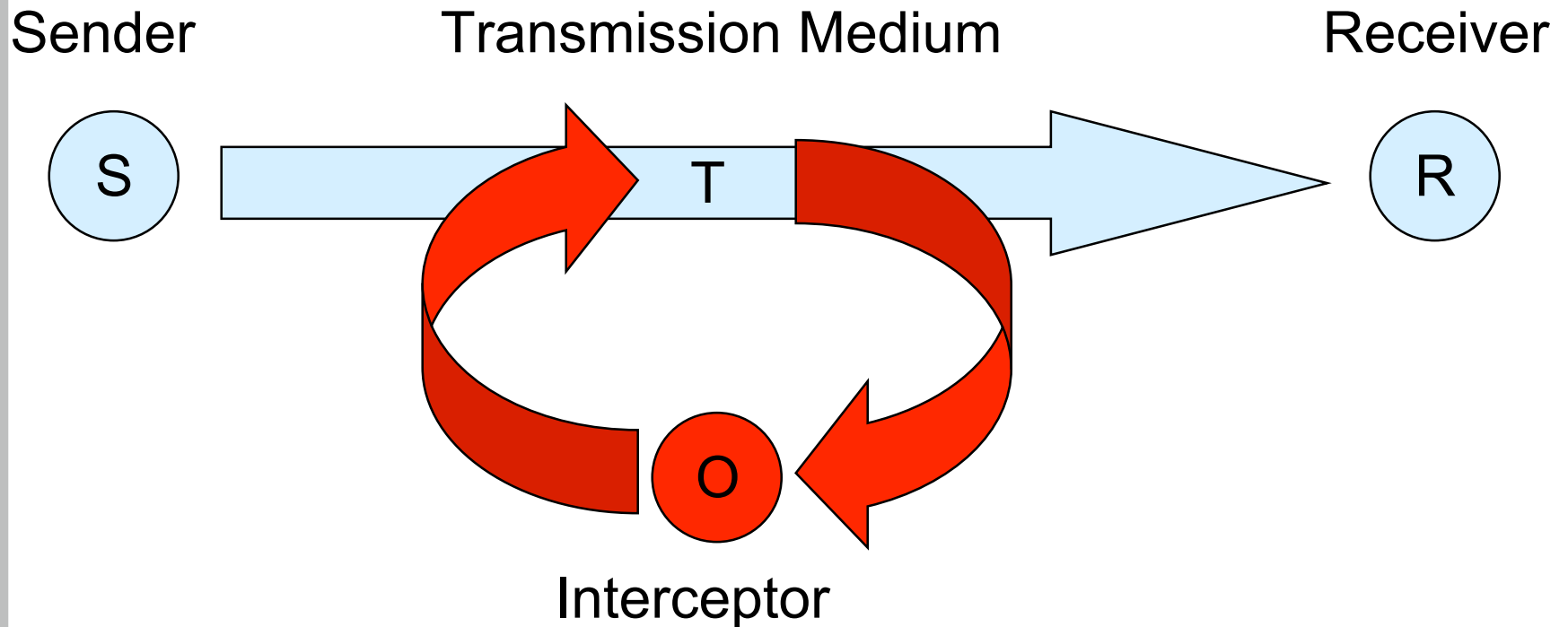
- Security is provided by the difficulty of calculating discrete logarithms.
- Feasibility is provided by
 - The ability to find large primes.
 - The ability to find primitive roots for large primes.
 - The ability to do efficient modular arithmetic.
- Correctness is an immediate consequence of basic facts about modular arithmetic.
- What could go wrong?

General Definition of “Protocol”

- A *protocol* is a multi-party algorithm
 - A sequence of steps that precisely specify the actions required of the parties in order to achieve a specified objective.
 - Every participant must know the protocol and the steps in advance.
 - Every participant must agree to follow the protocol
 - *Honest participants*
- Typically a situation of heterogeneous trust
 - Alice may not trust Bart
 - Bart may not trust the network
- Big problem: How to deal with bad participants?

Cryptographic Protocols

- Consider communication over a network...
- What is the threat model?
 - What are the vulnerabilities?





What Can the Attacker Do?

- Intercept them (confidentiality)
- Modify them (integrity)
- Fabricate other messages (integrity)
- Replay them (integrity)

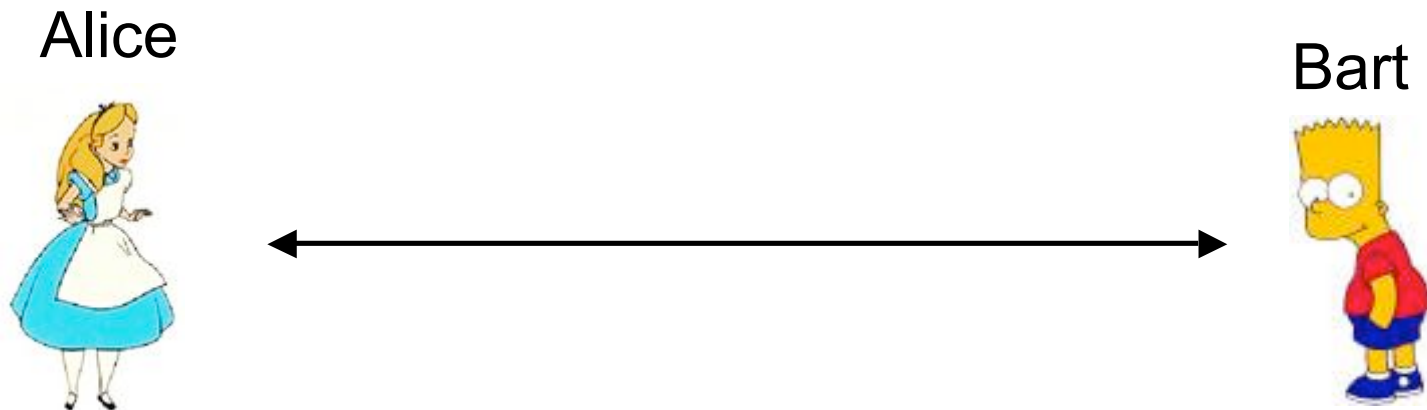
- Block the messages (availability)
- Delay the messages (availability)
- Cut the wire (availability)
- Flood the network (availability)

Dolev-Yao Model

- Treat cryptographic operations as "black box"
- Simplifies reasoning about protocols (doesn't require reduction to computational complexity)
- Given a message $M = (c_1, c_2, c_3, \dots)$ attacker can deconstruct message into components c_1 c_2 c_3
- Given a collection of components c_1, c_2, c_3 , attacker can forge message (c_1, c_2, c_3)
- Given an encrypted object $K\{c\}$, attacker can learn c only if attacker knows decryption key corresponding to K
- Attacker can encrypt components by using:
 - fresh keys, or
 - keys they have learned during the attack

Authentication

- For honest parties, the claimant A is able to authenticate itself to the verifier B. That is, B will complete the protocol having accepted A's identity.



Shared-Key Authentication

Alice



K_{AB}

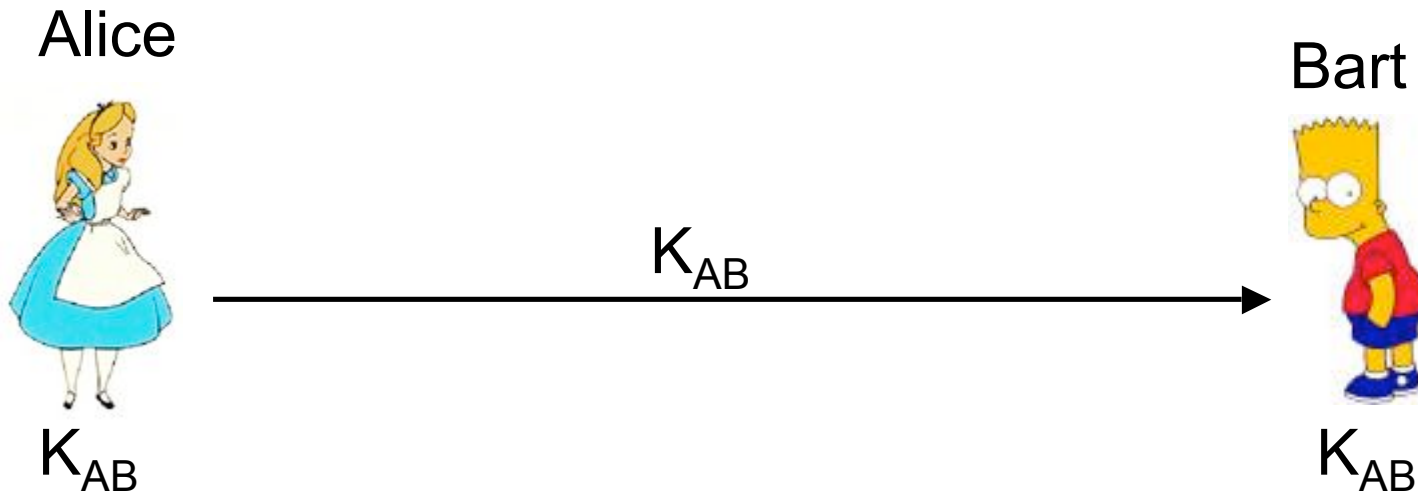
Bart



K_{AB}

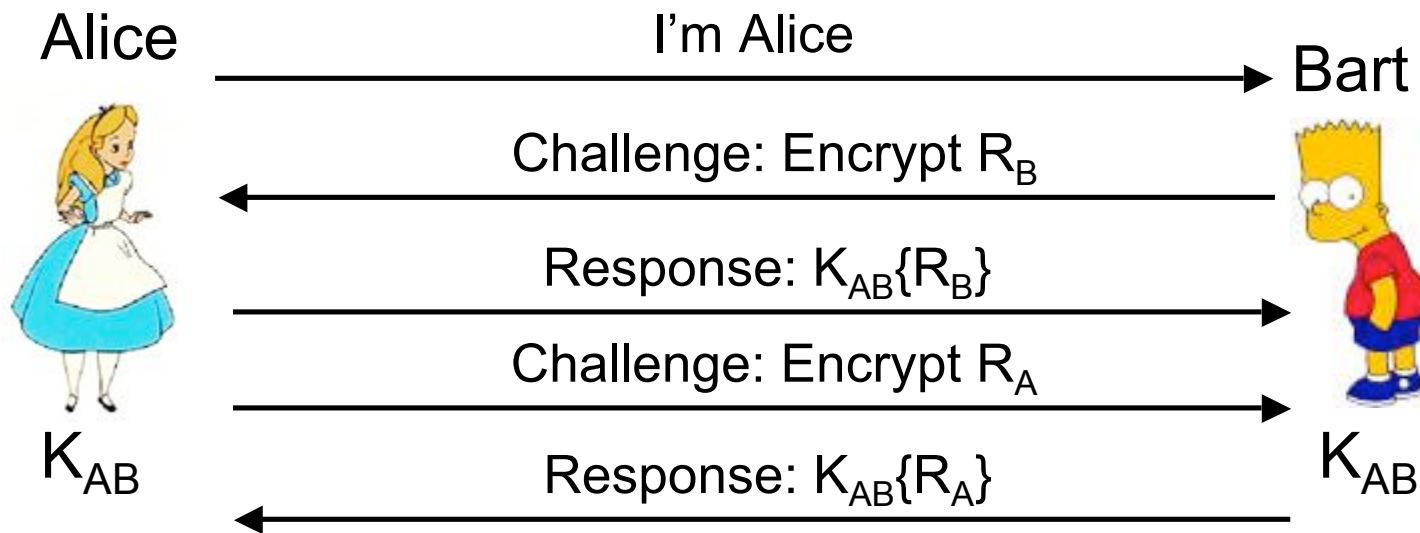
- Assume Alice & Bart already share a key K_{AB} .
 - The key might have been decided upon in person or obtained from a trusted 3rd party.
- Alice & Bart now want to communicate over a network, but first wish to authenticate to each other

Solution 1: Weak Authentication



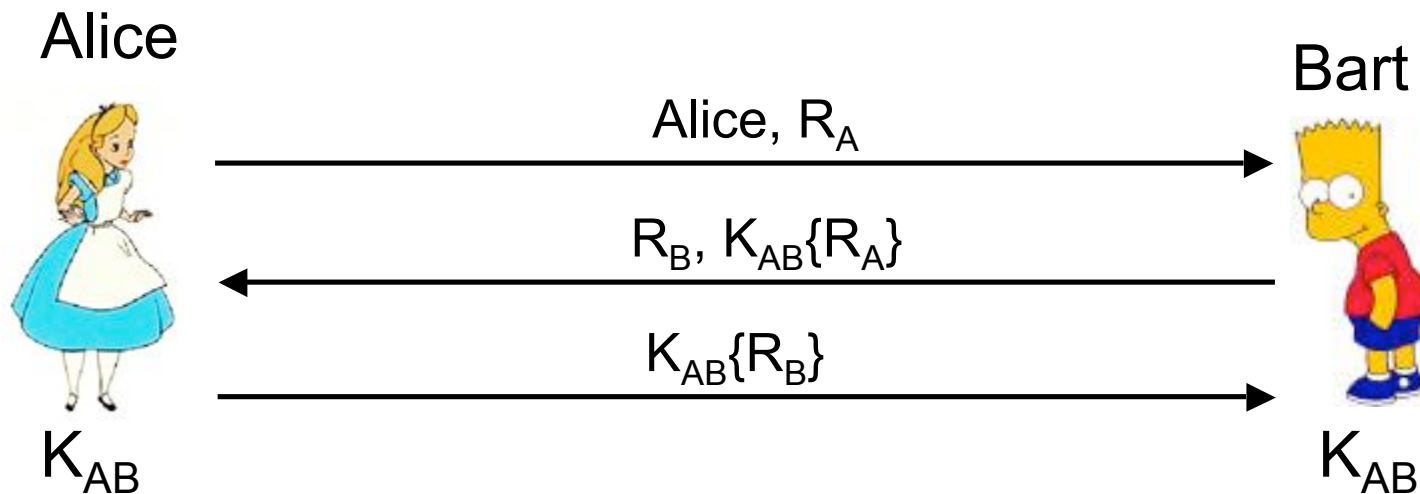
- Alice sends Bart K_{AB} .
 - K_{AB} acts as a password.
- The secret (key) is revealed to passive observers.
- Only works one-way.
 - Alice doesn't know she's talking to Bart.

Solution 2: Strong Authentication



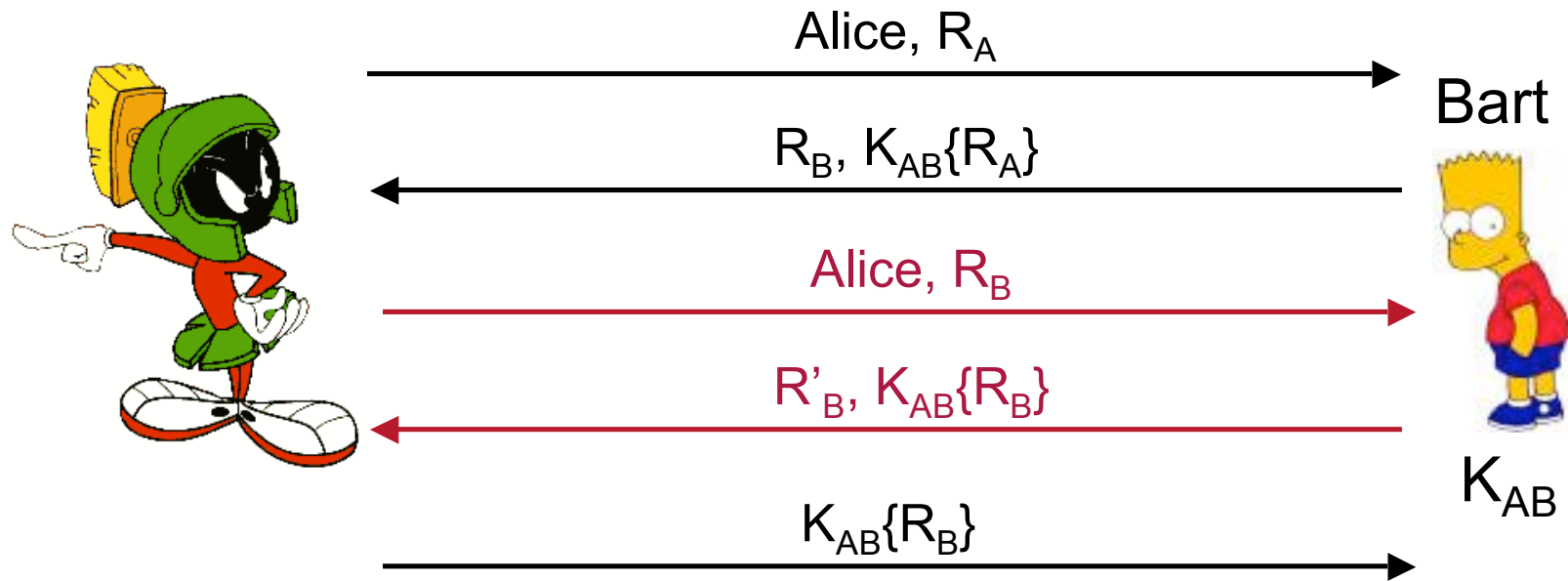
- Protocol doesn't reveal the secret.
- *Challenge/Response*
 - Bart requests proof that Alice knows the secret
 - Alice requires proof from Bart
 - R_A and R_B are randomly generated numbers

(Flawed) Optimized Version



- Why not send more information in each message?
- This seems like a simple optimization.
- But, it's broken... how?

Attack: Marvin can Masquerade as Alice



- Marvin pretends to take the role of Alice in two runs of the protocol.
 - Tricks Bart into doing Alice's part of the challenge!
 - Interleaves two instances of the same protocol.

Lessons

- Protocol design is tricky and subtle
 - “Optimizations” aren’t necessarily good
- Need to worry about:
 - Multiple instances of the same protocol running in parallel
 - Intruders that play by the rules, mostly
- General principle:
 - Don’t do anything more than necessary until confidence is built.
 - Initiator should prove identity *before* responder takes action (like encryption)