UNIVERSITY of SAN FRANCISCO
department of computer science

Internet Key Exchange (IKE)

EJ Jung

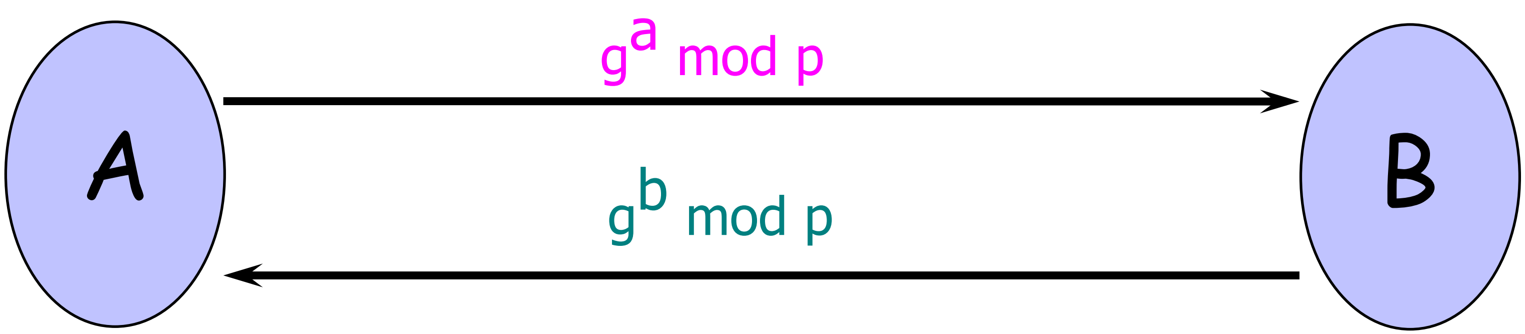
Secure Key Establishment

- Goal: generate and agree on a session key using some public initial information
- What properties are needed?
  - Authentication (know identity of other party)
  - Secrecy (generated key not known to any others)
  - Forward secrecy (compromise of one session key does not compromise keys in other sessions)
  - Prevent replay of old key material
  - Prevent denial of service
  - Protect identities from eavesdroppers
  - Other properties you can think of???

Key Management in IPSec

- Manual key management
  - Keys and parameters of crypto algorithms exchanged offline (e.g., by phone), security associations established by hand
- Pre-shared symmetric keys
  - New session key derived for each session by hashing pre-shared key with session-specific nonces
  - Standard symmetric-key authentication and encryption
- Online key establishment
  - Internet Key Exchange (IKE) protocol
  - Use Diffie-Hellman to derive shared symmetric key

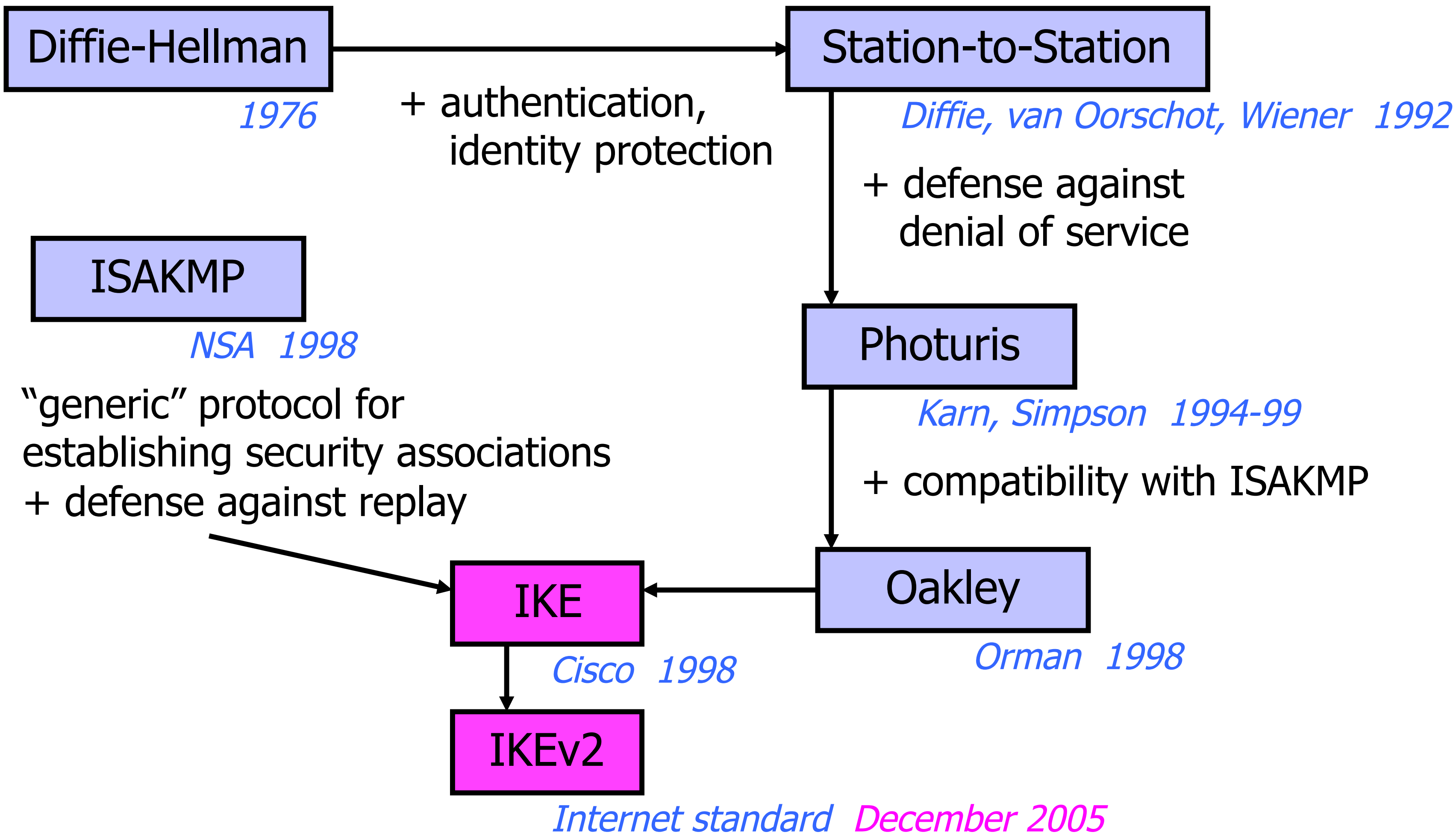
Diffie-Hellman Key Exchange



Authentication?	No
Secrecy?	Only against passive attacker
Replay attack?	Vulnerable
Forward secrecy?	Yes
Denial of service?	Vulnerable
Identity protection?	Yes

Participants can't tell g^x mod p from a random element of G: send them garbage and they'll do expensive exponentiations

IKE Genealogy



Design Objectives for Key Exchange

- Shared secret
  - Create and agree on a secret which is known only to protocol participants
- Authentication
  - Participants need to verify each other's identity
- Identity protection
  - Eavesdropper should not be able to infer participants' identities by observing protocol execution
- Protection against denial of service
  - Malicious participant should not be able to exploit the protocol to cause the other party to waste resources

Ingredient 1: Diffie-Hellman

$A \rightarrow B: \; g^a$

$B \rightarrow A: \; g^b$

- Shared secret is g^{ab}, compute key as $k=hash(g^{ab})$
  - Diffie-Hellman guarantees perfect forward secrecy
- Authentication
- Identity protection
- DoS protection

Ingredient 2: Challenge-Response

$$A \rightarrow B: \; m, A$$

$$B \rightarrow A: \; n, sig_B(m, n, A)$$

$$A \rightarrow B: \; sig_A(m, n, B)$$

- Shared secret
- Authentication
  - A receives his own number m signed by B's private key and deduces that B is on the other end; similar for B
- Identity protection
- DoS protection

DH + Challenge-Response

ISO 9798-3 protocol:

$A \rightarrow B$: g^a, A

$B \rightarrow A$: g^b, $sig_B(g^a, g^b, A)$

$A \rightarrow B$: $sig_A(g^a, g^b, B)$

$m := g^a$

$n := g^b$

- Shared secret: g^{ab}
- Authentication
- Identity protection
- DoS protection

Ingredient 3: Encryption

Encrypt signatures to protect identities:

$$A \rightarrow B: \; g^a, A$$

$$B \rightarrow A: \; g^b, Enc_K(sig_B(g^a, g^b, A))$$

$$A \rightarrow B: \; Enc_K(sig_A(g^a, g^b, B))$$

$k=hash(g^{ab})$

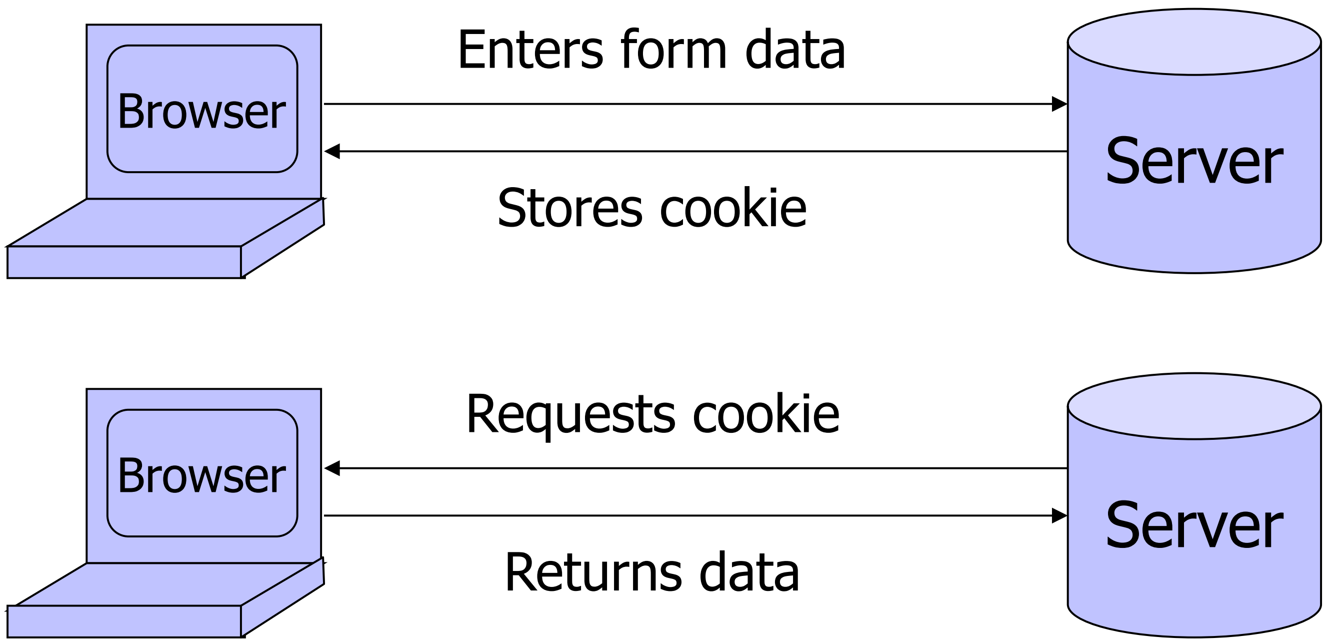
- Shared secret: g^{ab}
- Authentication
- Identity protection (for responder only!)
- DoS protection

Refresher: DoS Prevention

- Denial of service due to resource clogging
  - If responder opens a state for each connection attempt, attacker can initiate thousands of connections from bogus or forged IP addresses
- Cookies ensure that the responder is stateless until initiator produced at least 2 messages
  - Responder's state (IP addresses and ports) is stored in an unforgeable cookie and sent to initiator
  - After initiator responds, cookie is regenerated and compared with the cookie returned by the initiator
  - The cost is 2 extra messages in each execution

Storing Info Across Sessions

- A cookie is a file created by an Internet site to store information on your computer



HTTP is a stateless protocol; cookies add state

Refresher: Anti-DoS Cookie

- Typical protocol:
  - Client sends request (message #1) to server
  - Server sets up connection, responds with message #2
  - Client may complete session or not (potential DoS)
- Cookie version:
  - Client sends request to server
  - Server sends hashed connection data back
    - Send message #2 later, after client confirms his address
  - Client confirms by returning hashed data
  - Need an extra step to send postponed message #2

Ingredient 4: Anti-DoS Cookie

"Almost-IKE" protocol:

Doesn't quite work: B must remember his DH exponent b for every connection

$A \rightarrow B$: g^a, A

$B \rightarrow A$: g^b, $hash_{Kb}(g^b, g^a)$

$A \rightarrow B$: g^a, g^b, $hash_{Kb}(g^b, g^a)$

$Enc_K(sig_A(g^a, g^b, B))$

$B \rightarrow A$: g^b, $Enc_K(sig_B(g^a, g^b, A))$

$k=hash(g^{ab})$

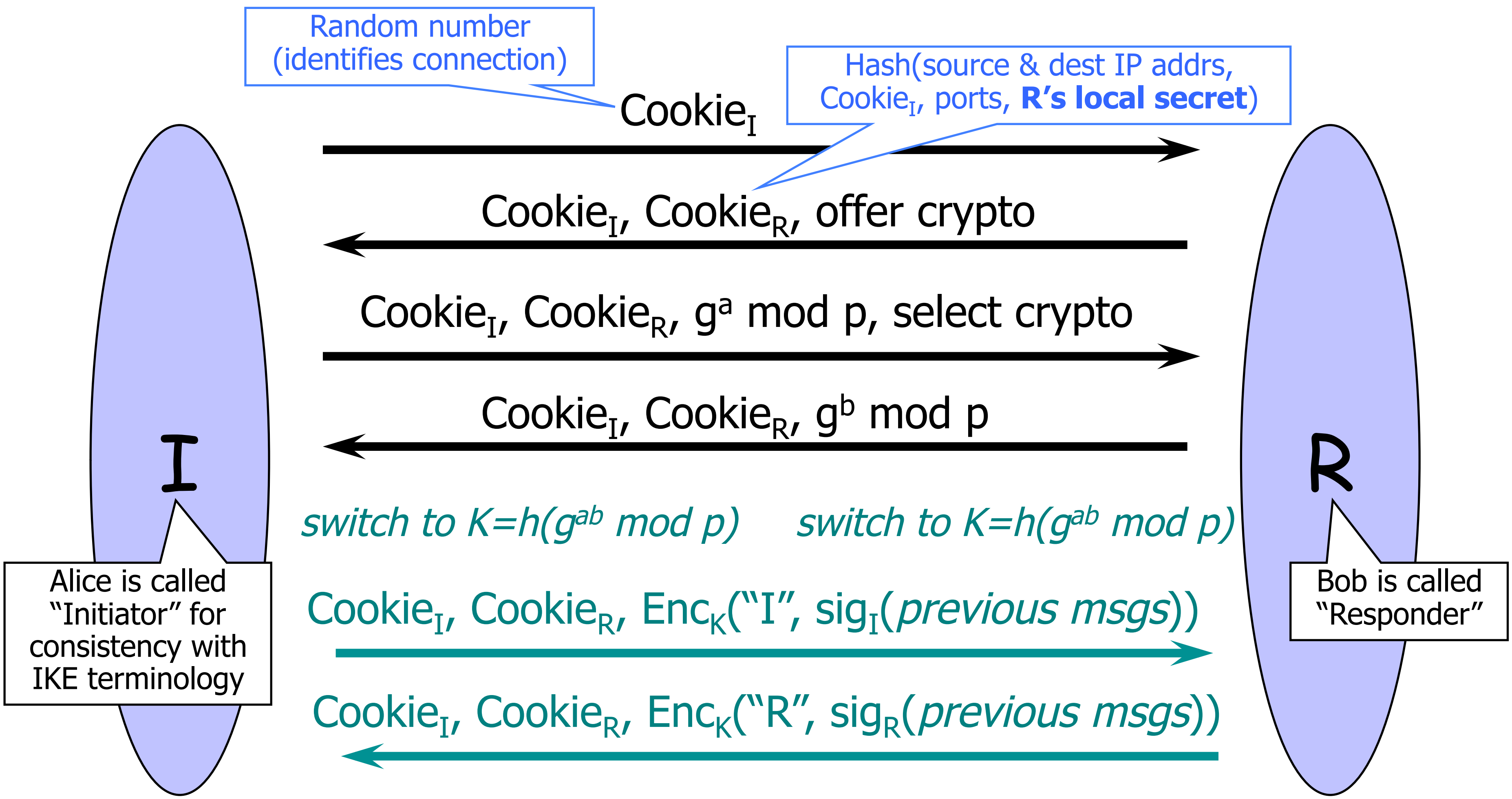
- Shared secret: g^{ab}
- Authentication
- Identity protection
- DoS protection?

Medium-Term Secrets and Nonces

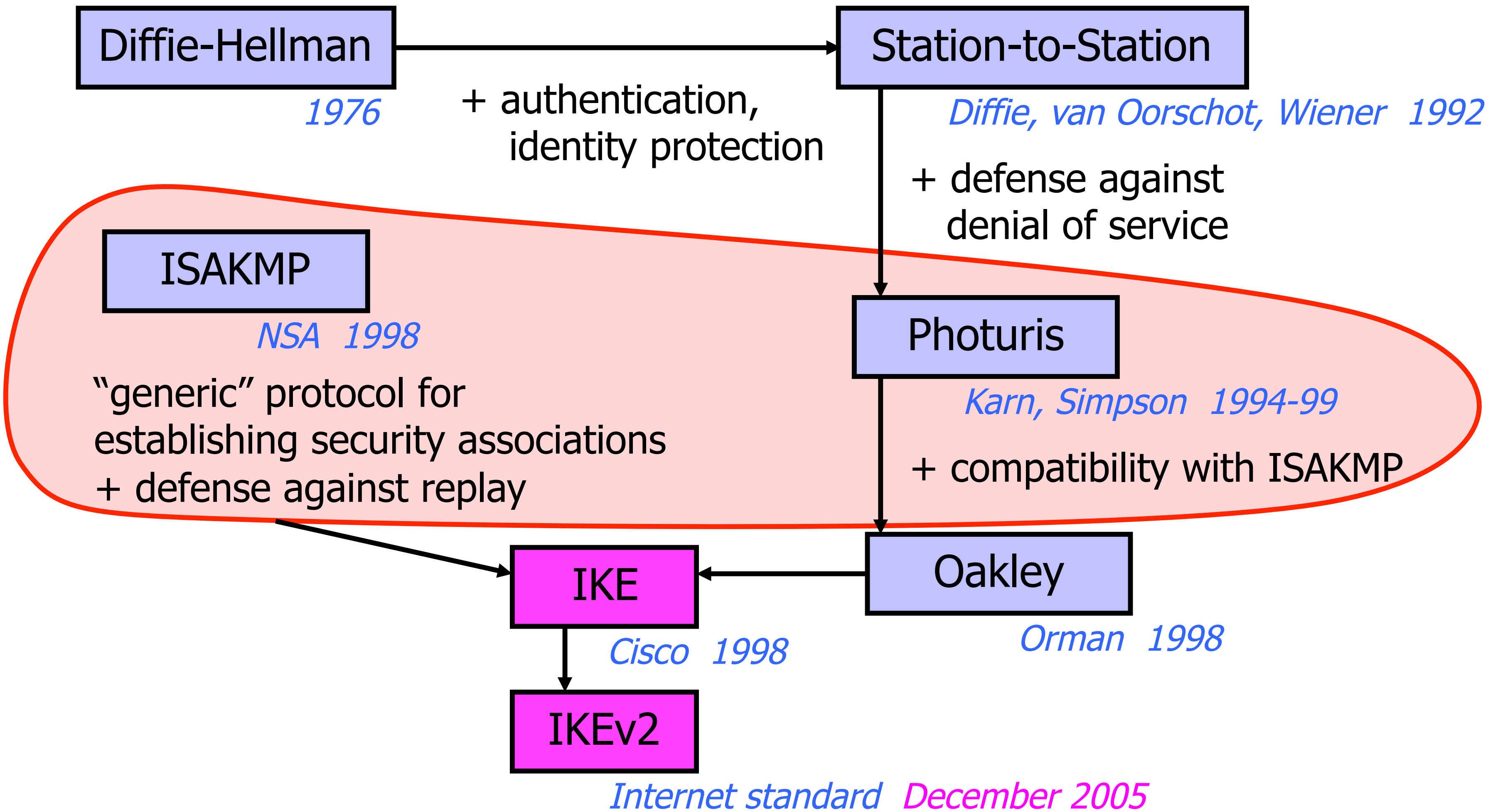
- Idea: use the same Diffie-Hellman value g^{ab} for every session, update every 10 minutes or so
  - Helps against denial of service
- To make sure keys are different for each session, derive them from g^{ab} and session-specific nonces
  - Nonces guarantee freshness of keys for each session
  - Re-computing g^a, g^b, g^{ab} is costly, generating nonces (fresh random numbers) is cheap
- This is more efficient and helps with DoS, but no longer guarantees forward secrecy (why?)

(Simplified) Photuris

[Karn and Simpson]



IKE Genealogy Redux



Cookies in Photuris and ISAKMP

- Photuris cookies are derived from local secret, IP addresses and ports, counter, crypto schemes
  - Same (frequently updated) secret for all connections
- ISAKMP requires <u>unique</u> cookie for each connect
  - Add timestamp to each cookie to prevent replay attacks
  - Now responder needs to keep state ("cookie crumb")
    - Vulnerable to denial of service (why?)
- Inherent conflict: to prevent replay, need to remember values that you've generated or seen before, but keeping state allows denial of service

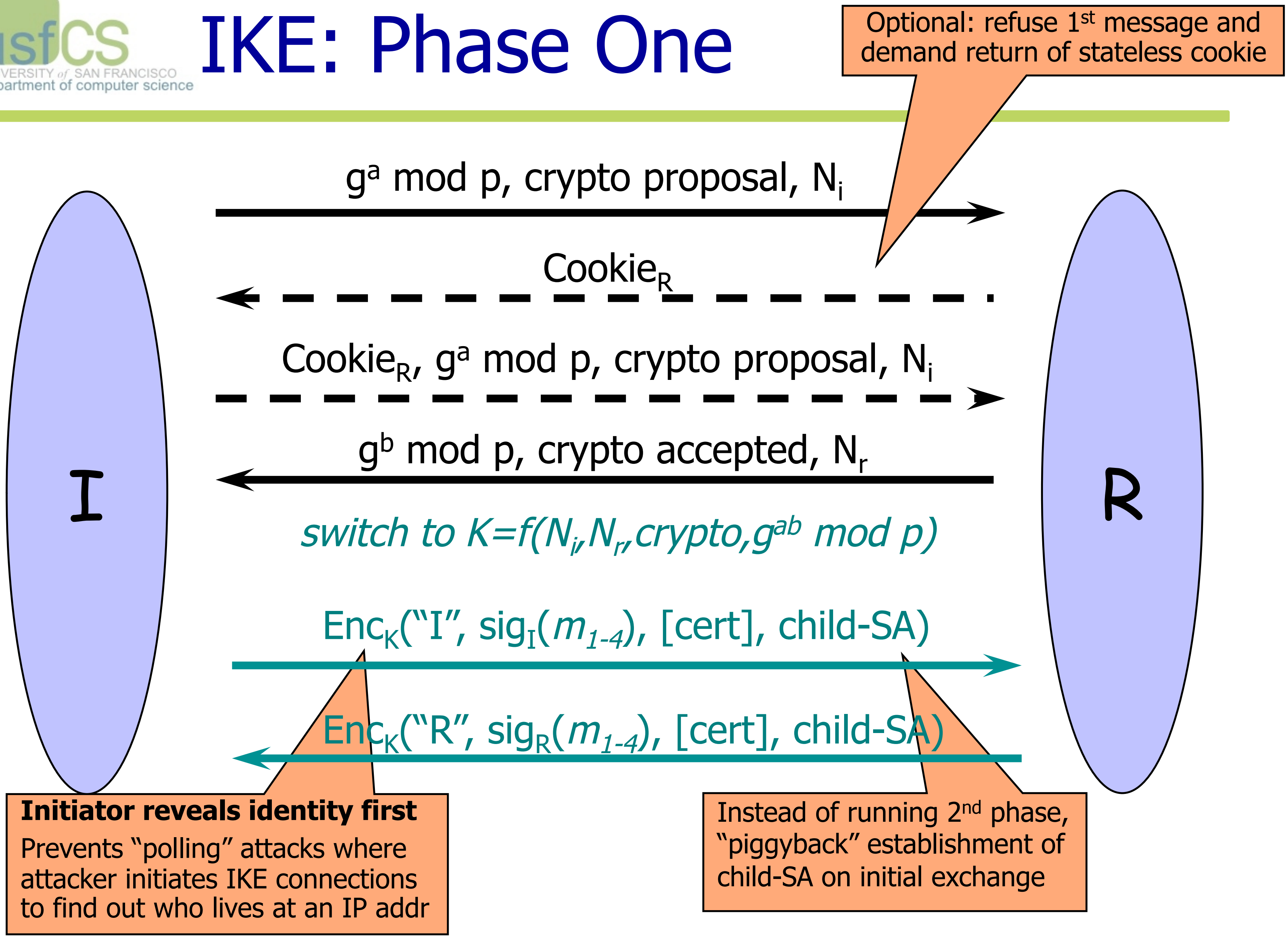
IKE Overview

- Goal: create security association between 2 hosts
  - Shared encryption and authentication keys, agreement on crypto algorithms
- Two phases: 1st phase establishes security association (IKE-SA) for the 2nd phase
  - Always by authenticated Diffie-Hellman (expensive)
- 2nd phase uses IKE-SA to create actual security association (child-SA) to be used by AH and ESP
  - Use keys derived in the 1st phase to avoid DH exchange
  - Can be executed cheaply in "quick" mode
    - To create a fresh key, hash old DH value and new nonces

Why Two-Phase Design?

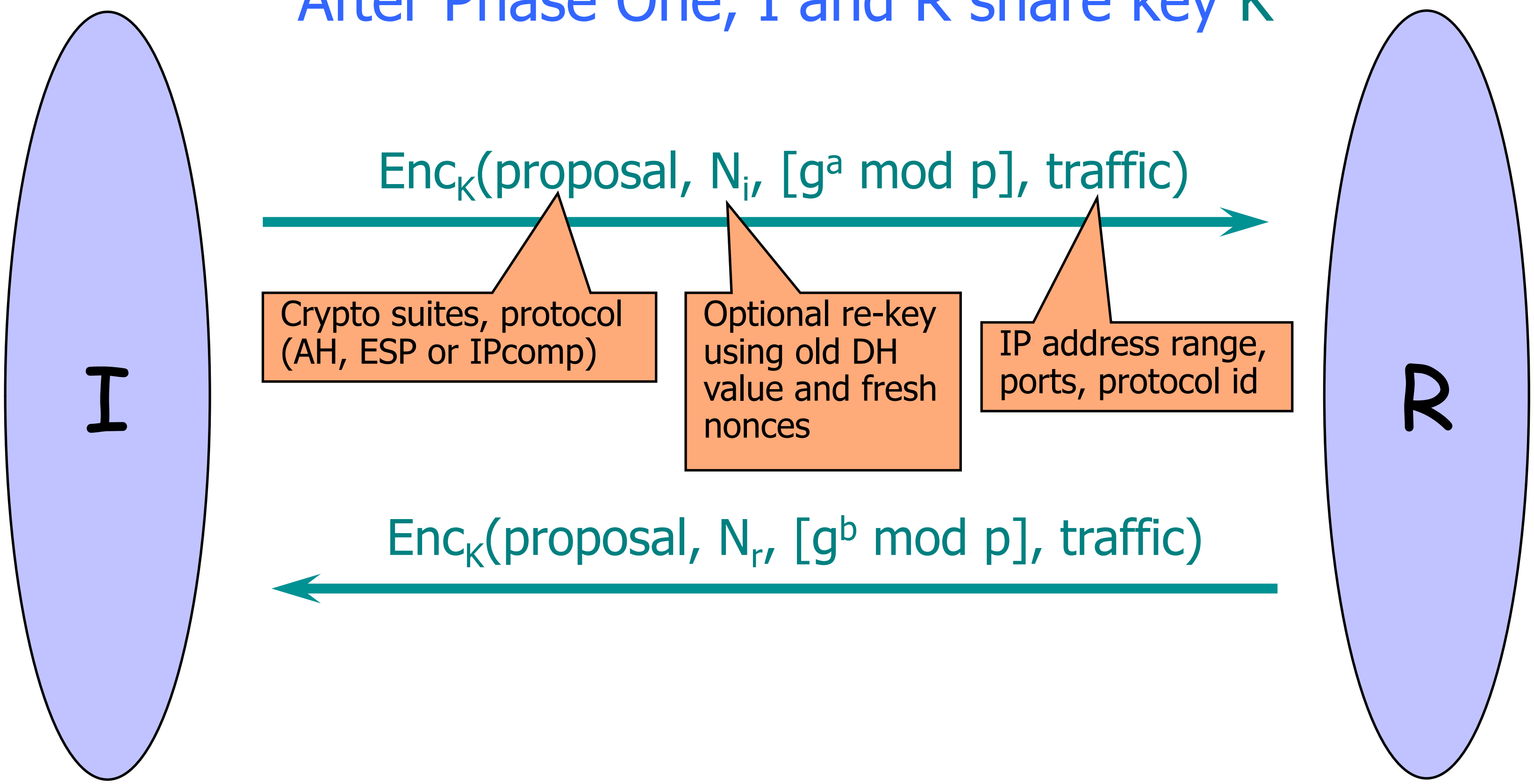
- Expensive 1st phase creates "main" SA
- Cheap 2nd phase allows to create multiple child SAs (based on "main" SA) between same 2 hosts
  - Example: one SA for AH, another SA for ESP
  - Different conversations may need different protection
    - Some traffic only needs integrity protection or short-key crypto
    - Too expensive to always use strongest available protection
  - Avoid multiplexing several conversations over same SA
    - For example, if encryption is used without integrity protection (bad idea!), it may be possible to splice the conversations
  - Different SAs for different classes of service

IKE: Phase One



IKE: Phase Two (Create Child-SA)

After Phase One, I and R share key K



Can run this several times to create multiple SAs

Other Aspects of IKE

We did not talk about…

- Interaction with other network protocols
  - How to run IPSec through NAT (Network Address Translation) gateways?
- Error handling
  - Very important! Bleichenbacher attacked SSL by cryptanalyzing error messages from an SSL server
- Protocol management
  - Dead peer detection, rekeying, etc.
- Legacy authentication
  - What if one of the parties doesn't have a public key?

Current State of IPSec

- Best currently existing VPN standard
  - For example, used in Cisco PIX firewall, many remote access gateways
- IPSec has been out for a few years, but wide deployment has been hindered by complexity
  - ANX (Automotive Networking eXchange) uses IPSec to implement a private network for the Big 3 auto manufacturers and their suppliers