Security, part 2

The systems

Announcements

- HW3, coming soon
- No class next week for Thanksgiving

Last time

- Max Krohn and OKCupid

Last Thursday

- Security, part one: The tools
  - Public- and private-key cryptography
  - Feistel block ciphers and DES
  - Cryptographic hashing

Today: Security, part 2

- Digital signatures
- Needham-Schroeder and Kerberos
- Hybrid cryptographic protocols
  - TLS / SSL

Digital signature goals

- Authentication
  - Prove that a message has not been altered
- Unforgeability
  - Prove that the message was created by a specific person (a.k.a. the principal)
- Non-repudiation
  - Once a message is signed, the principal cannot deny that they signed the message

Signatures with public-key crypto

- One option (not used in practice):
  - Encrypt with private key to sign a message:

    $s = E(K_{priv}, m)$

    Send m,s
  - Decrypt with public key to verify the signature:

    $m' = D(K_{pub}, s)$

    Check that $m == m'$
  - Because private key is not shared, the signature is unforgeable and unrepudiable
  - Because public key is shared, anyone can verify the signature
  - A problem: public-key cryptography is slow

Signatures with public-key crypto

- An improvement:
  - Hash the message with a cryptographic hash function first, sign the hash:

    $h = H(m)$
    $s = E(K_{priv}, h)$
    Send m,s
  - Use the hash function and public key to verify the signature:

    $h = H(m)$
    $h' = D(K_{pub}, s)$
    Check that $h == h'$
  - Cryptographic hash functions are often 30-100x faster than public-key cryptography
    - Public-key crypto needed just to sign the short hash
  - Hash function must be cryptographic to prevent attacker from replacing *m* with *m'* such that $H(m') == H(m)$

Signatures with private-key crypto

- Using a cryptographic hash function H and shared private key K:

  $s = H(m + K)$

  Send m,s

  Bit-string append (not addition)

- To verify:

  Compute $s' = H(m + K)$

  Check $s == s'$

- Very fast: no encryption/decryption needed
- A problem: need to reveal private key to verify the signature

Needham-Schroeder and Kerberos

- Goal: to create a secure, usable system providing authentication and privacy without public-key cryptography
  - Will use private-key cryptography
    - A key problem to solve: private-key cryptography requires a shared private key
    - Will use a trusted third party to negotiate the shared private key

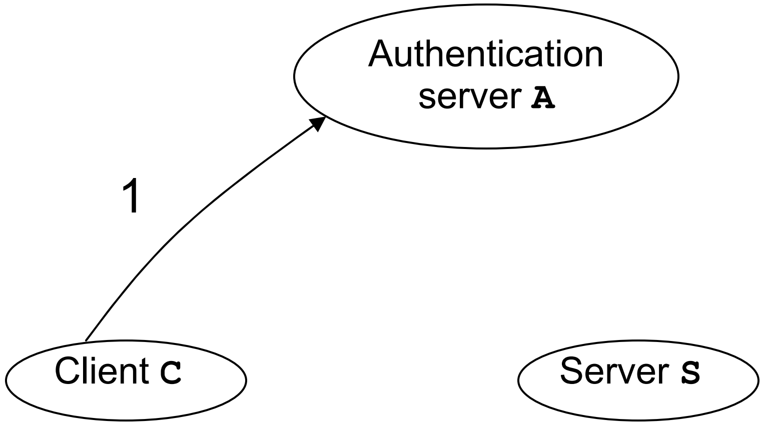
The trusted third party

- Stores private keys for all users
- Generates “tickets” which contain a session key when two parties need to communicate

Needham-Schroeder and Kerberos

- In following diagrams:
  - Client **`C`** initiating a connection to server **`S`**
    - Authentication server **`A`** generates a session key K_{SC}
  - Client **`C`** has private key K_C, which only **`A`** and **`C`** share
  - Server **`S`** has private key K_S, which only **`A`** and **`S`** share

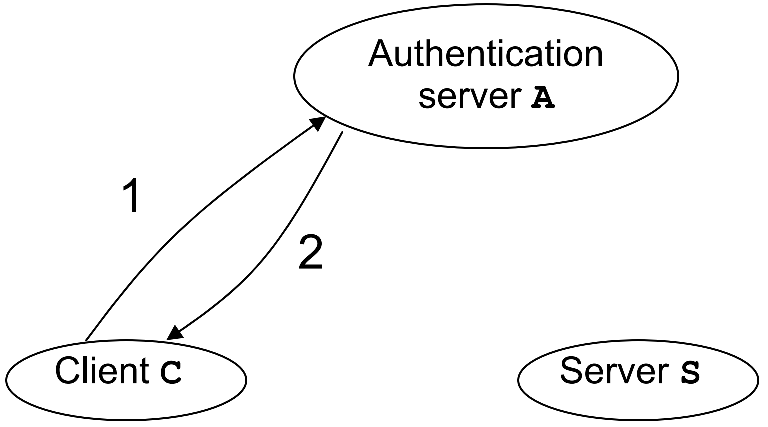
Needham-Schroeder and Kerberos



- Messages:
  - 1: C to A: C,S,n

A nonce: a "number used once." In Kerberos this is usually the time.

Needham-Schroeder and Kerberos



- Messages:

  1: C to A: C,S,n

  2: A to C: $\{K_{CS},S,n\}_{K_C}$ $\{C,S,K_{CS},t_1,t_2\}_{K_S}$

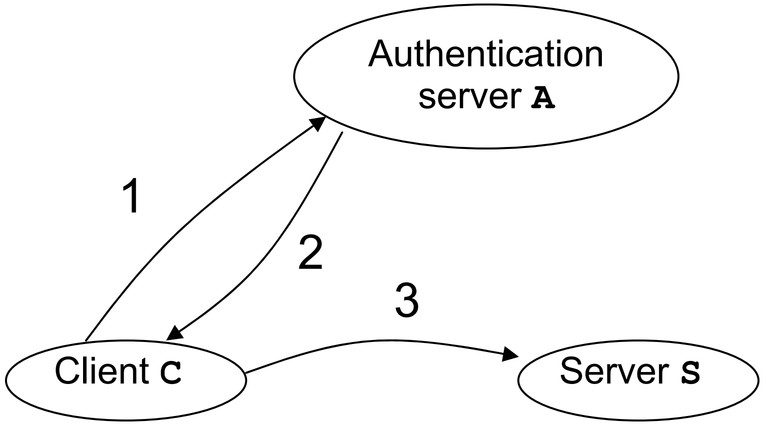
start and end time for K_{CS}

the session key

K_{CS},S,n encrypted with private key K_C

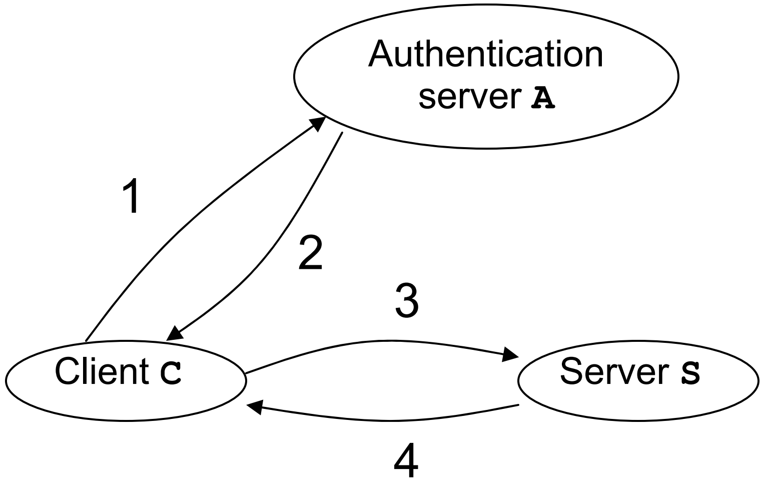
C,S,K_{CS},t_1,t_2 encrypted with private key K_S

Needham-Schroeder and Kerberos



- Messages:
  - 1: C to A: C,S,n
  - 2: A to C: $\{K_{cs},S,n\}_{K_c}$ $\{C,S,K_{cs},t_1,t_2\}_{K_s}$
  - 3: C to S: $\{\text{request},n',\ldots\}_{K_{sc}}$ $\{C,S,K_{cs},t_1,t_2\}_{K_s}$

Needham-Schroeder and Kerberos



- Messages:
  - 1: C to A: C,S,n
  - 2: A to C: $\{K_{cs},S,n\}_{K_c}$ $\{C,S,K_{cs},t_1,t_2\}_{K_s}$
  - 3: C to S: $\{\text{request},n',\ldots\}_{K_{sc}}$ $\{C,S,K_{cs},t_1,t_2\}_{K_s}$
  - 4: S to C: $\{n',\text{response},\ldots\}_{K_{sc}}$

Needham-Schroeder and Kerberos

- Not shown here:
  - In Kerberos, this is just the process for negotiating a session key for a new client-server connection. There's a separate process (with its own authentication server and exchange of messages) for initially authenticating to the Kerberos system.
  - The client and server typically exchange a subsession key as part of their handshake, and use that subsession key for encrypting subsequent communication. (They periodically use the original session key to renegotiate new subsession keys, to avoid encrypting too much information with a single private key.)
  - Kerberos sometimes just used for authentication, not necessarily for encrypting the requests and responses themselves. (e.g., AFS)

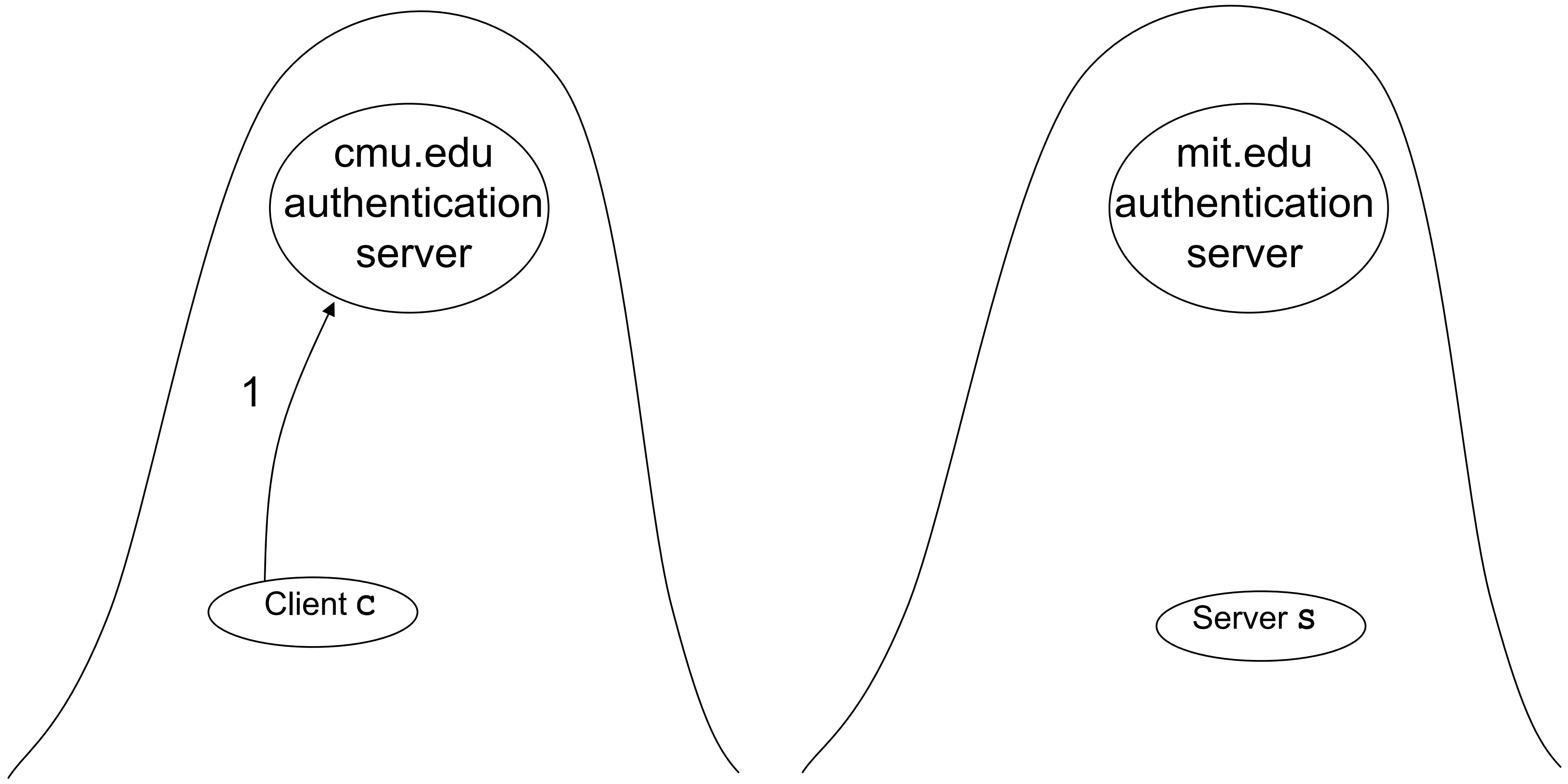
Needham-Schroeder and Kerberos

- Problems:
  - Trust!
    - The trusted 3rd party can authenticate as any user, and can read any communication
  - Scalability
    - The authentication server needs all keys
  - Single point of failure
    - If the authentication server fails, no new connections can be established

Scaling Kerberos

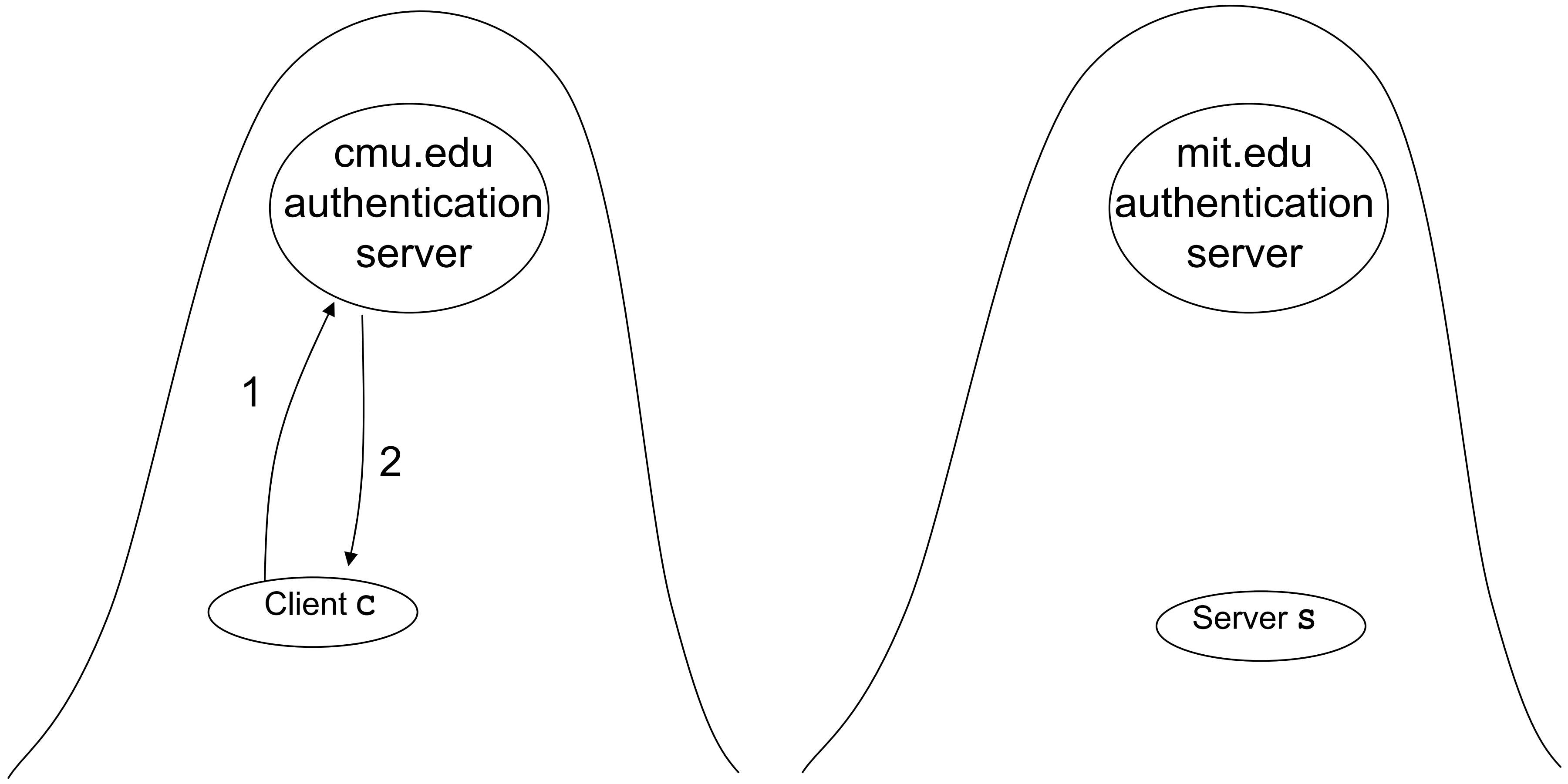
- Divide the world into *realms*
  - Authentication server in each realm has private keys for all users in that realm, but none for users from other realms
  - In Kerberos ver. 4, each realm authentication server has cross-realm private keys for every other realm

Scaling Kerberos



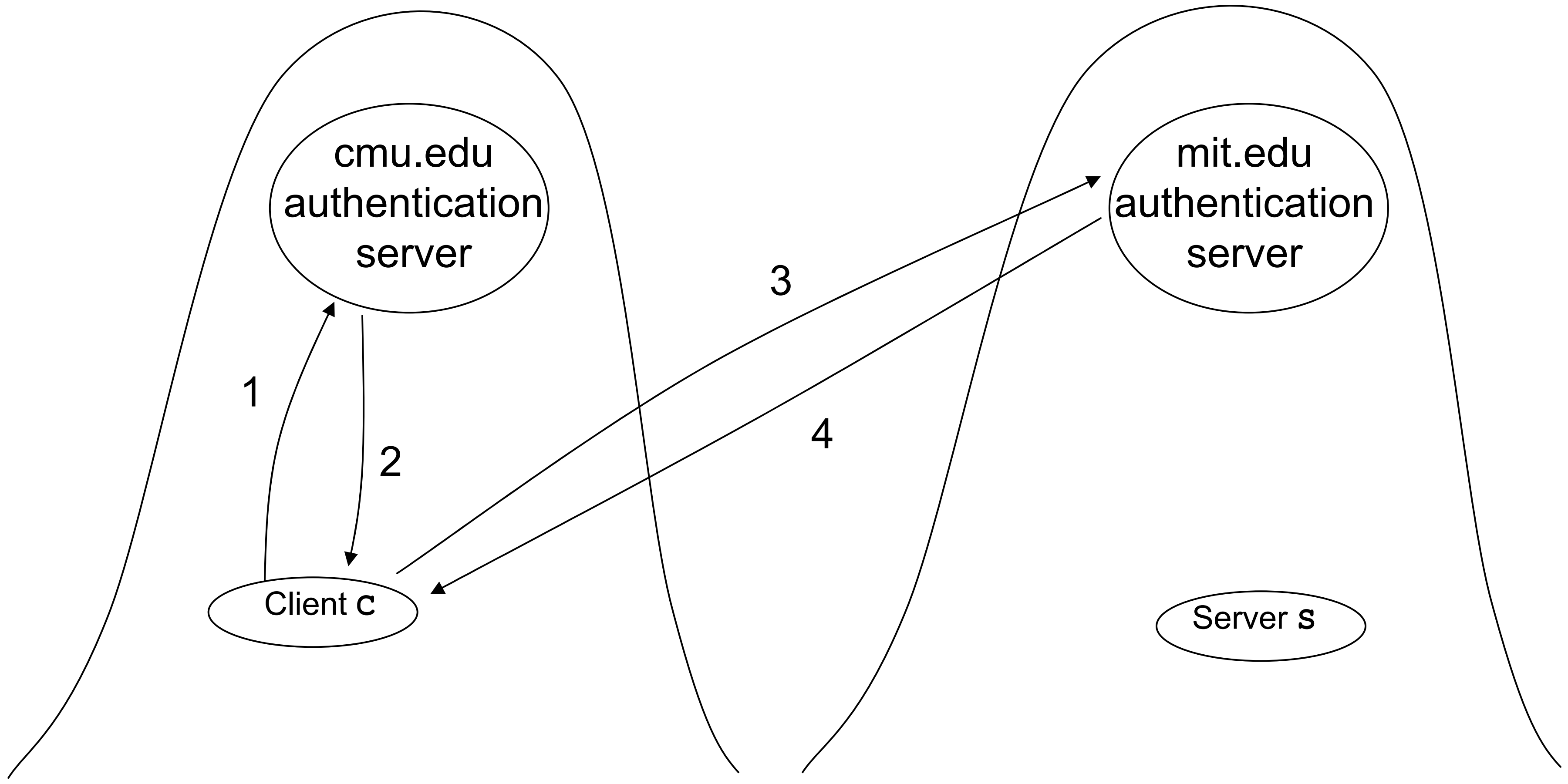
1: To initiate a connection with a server in a remote realm, client first sends request to authentication server in its own realm

Scaling Kerberos



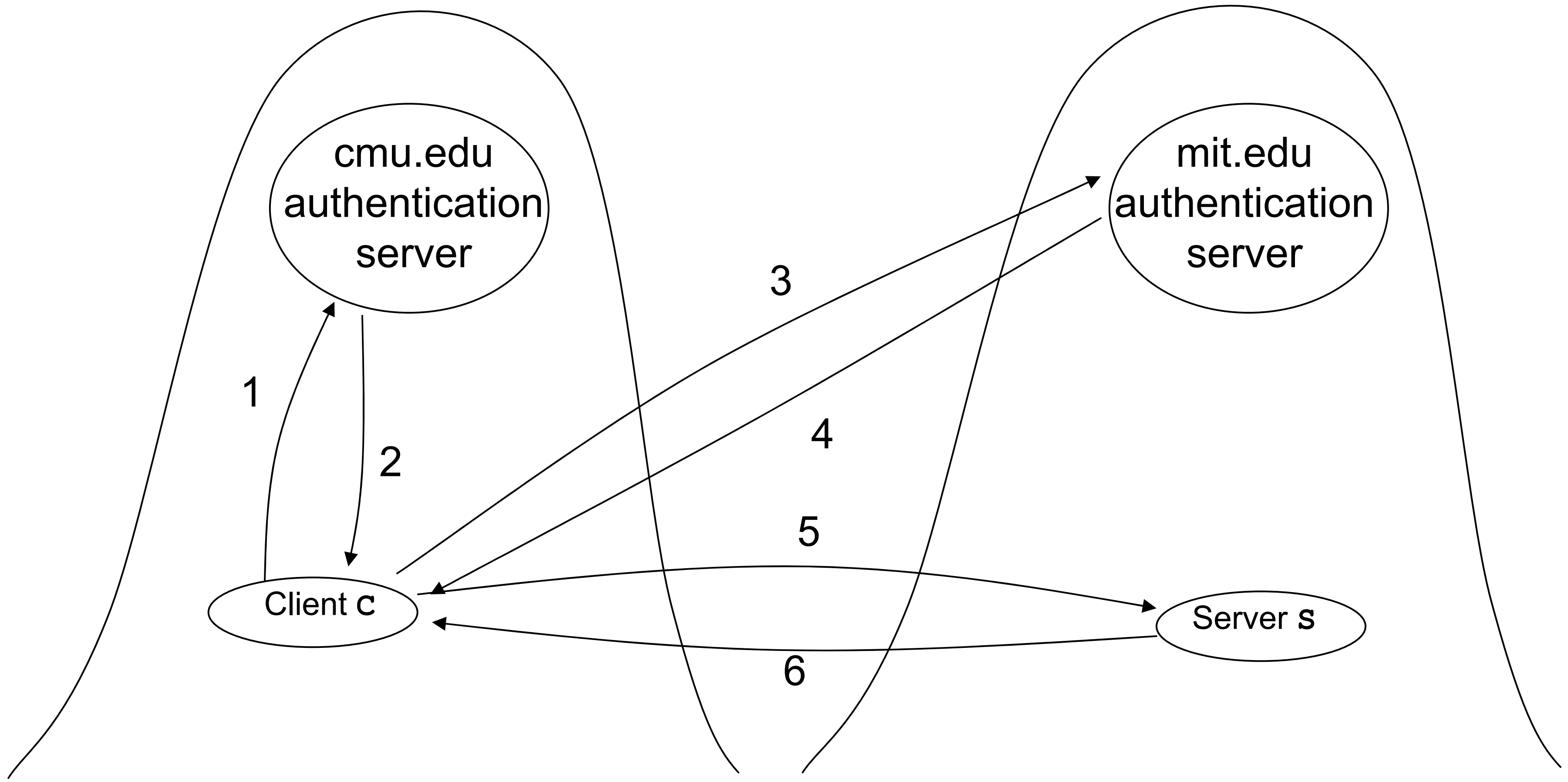
2: Client gets ticket-granting key; one ticket encrypted with cross-realm key, the other with c's private key, much as before.

Scaling Kerberos



3&4: Client uses ticket-granting ticket to authenticate to remote realm authentication server, which sends a session key for **c** and **s**

Scaling Kerberos



5&6: Client and server can now communicate much as before

Scaling Kerberos

- Problems:
  - Realm servers can authenticate as any users in their realm, read private client-server communication
  - Each realm server needs cross-realm private key for each other realm server they might want to authenticate to
    - O(n) keys for each realm server, O(n^2) keys total for n realm servers

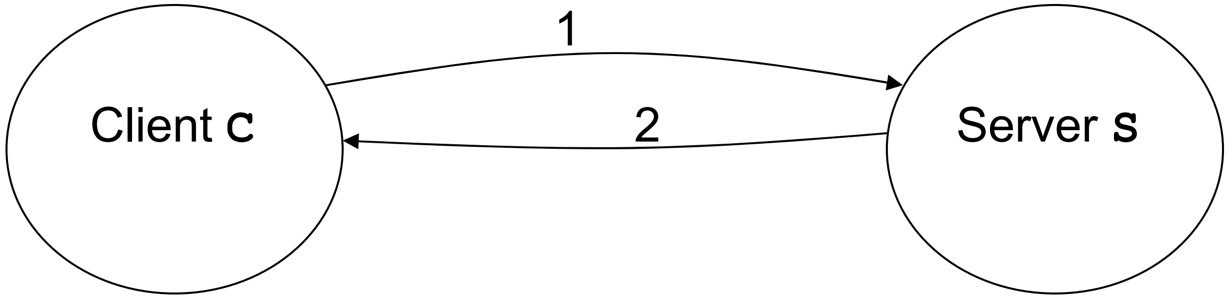
Scaling Kerberos

- Improvements:
  - Kerberos ver. 5 allows multi-hop cross-realm authentication
    - Allows a hierarchy of servers
      - Any realm server in your authentication path can read your private communication
      - When connecting, you get the list of realm servers in your authentication path, so you can decide whether or not you trust them

Hybrid cryptosystems

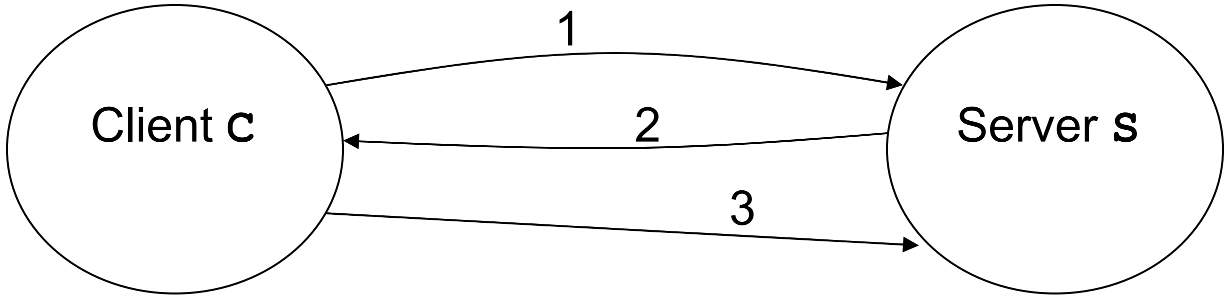
- Use public-key cryptography to negotiate a private session key
- Use private-key cryptography for the actual session
- E.g., SSH, Secure Socket Layer (SSL), Transport Layer Security (TLS)

Simplified SSL



- Messages:
  - 1: request
  - 2: s’s X.509v3 certificate, containing its public key signed by a certificate authority

Simplified SSL



- Messages:
  1: request
  2: s's X.509v3 certificate, containing its public key signed by a certificate authority
  3: Client verifies the certificate using the certificate authority's public key, sends session key for subsequent communication (encrypted with s's public key)

Hidden from the simplified view

- Hello messages initiating the communication
- Client and server negotiate which cryptosystem they will use for the session
- Client can send its own certificate, for client authentication

Hybrid cryptosystem problems

- Verifying the public key / certificate in a usable manner is hard
  - SSH essentially makes you verify it
  - How do you get the public certificate for the certificate authority?
    - Pre-installed in web browsers
      - Do you trust your web browser?
      - Did you trust your network connection when you downloaded your browser?