Lecture 10
October 15, 2025

ECS 235A, Computer and Information Security

Administrative Stuff

- In Extra Credit 2, problem 2, p and q are both prime
  - I have posted a revision with this change
  - Thanks to the folks on Piazza who pointed this out
- Zoom office hour on Wednesday moved to 4:00pm–4:50pm
  - Same Zoom meeting ID and password

About the Use of AI

- This is okay:

```
please fix up the English in "the sky are blew because
of fotons hitting moleculs"
```

- ChatGPT gives you

```
Sure! The corrected version is:
```

“The sky is blue because of photons hitting molecules.”

- The ideas and basic expression of those ideas is yours; here, ChatGPT is simply helping you with the English grammar and spelling. It's acting as a proofreader, nothing more.

About the Use of AI

- This is ***not*** okay:

```
in one sentence, tell me why is the sky blue
```

- ChatGPT gives you

```
The sky is blue because air molecules scatter shorter
blue wavelengths of sunlight more than longer red ones
```

- The idea is not yours; it is from ChatGPT. This is cheating, and is not allowed.

El Gamal Digital Signature

- Relies on discrete log problem
  - Choose p prime, g, $d < p$; compute $y = g^d \bmod p$
- Public key: (y, g, p); private key: d
- To sign contract m:
  - Choose k relatively prime to p–1, and not yet used
  - Compute $a = g^k \bmod p$
  - Find b such that $m = (da + kb) \bmod p–1$
  - Signature is (a, b)
- To validate, check that
  - $y^a a^b \bmod p = g^m \bmod p$

Example

- Alice chooses $p = 262643$, $g = 9563$, $d = 3632$, giving $y = 274598$
- Alice wants to send Bob signed contract PUP (152015)
  - Chooses $k = 601$ (relatively prime to 262642)
  - This gives $a = g^k \bmod p = 9563^{601} \bmod 29 = 202897$
  - Then solving $152015 = (3632 \times 202897 + 601b) \bmod 262642$ gives $b = 225835$
  - Alice sends Bob message $m = 152015$ and signature $(a,b) = (202897, 225835)$
- Bob verifies signature: $g^m \bmod p = 9563^{152015} \bmod 262643 = 157499$ and $y^a a^b \bmod p = 27459^{202897} 202897^{225835} \bmod 262643 = 157499$
  - They match, so Alice signed

Attack

- Eve learns k, corresponding message m, and signature (a, b)
  - Extended Euclidean Algorithm gives d, the private key
- Example from above: Eve learned Alice signed last message with $k = 5$

  $$m = (da + kb) \bmod p-1 \Rightarrow 152015 = (202897d + 601 \times 225835) \bmod 262642$$

  giving Alice's private key $d = 3632$

Notation

- $X \rightarrow Y : \{ Z \mid\mid W \} k_{X,Y}$
  - X sends Y the message produced by concatenating Z and W enciphered by key $k_{X,Y}$, which is shared by users X and Y
- $A \rightarrow T : \{ Z \} k_A \mid\mid \{ W \} k_{A,T}$
  - A sends T a message consisting of the concatenation of Z enciphered using k_A, A's key, and W enciphered using $k_{A,T}$, the key shared by A and T
- r_1, r_2 nonces (nonrepeating random numbers)

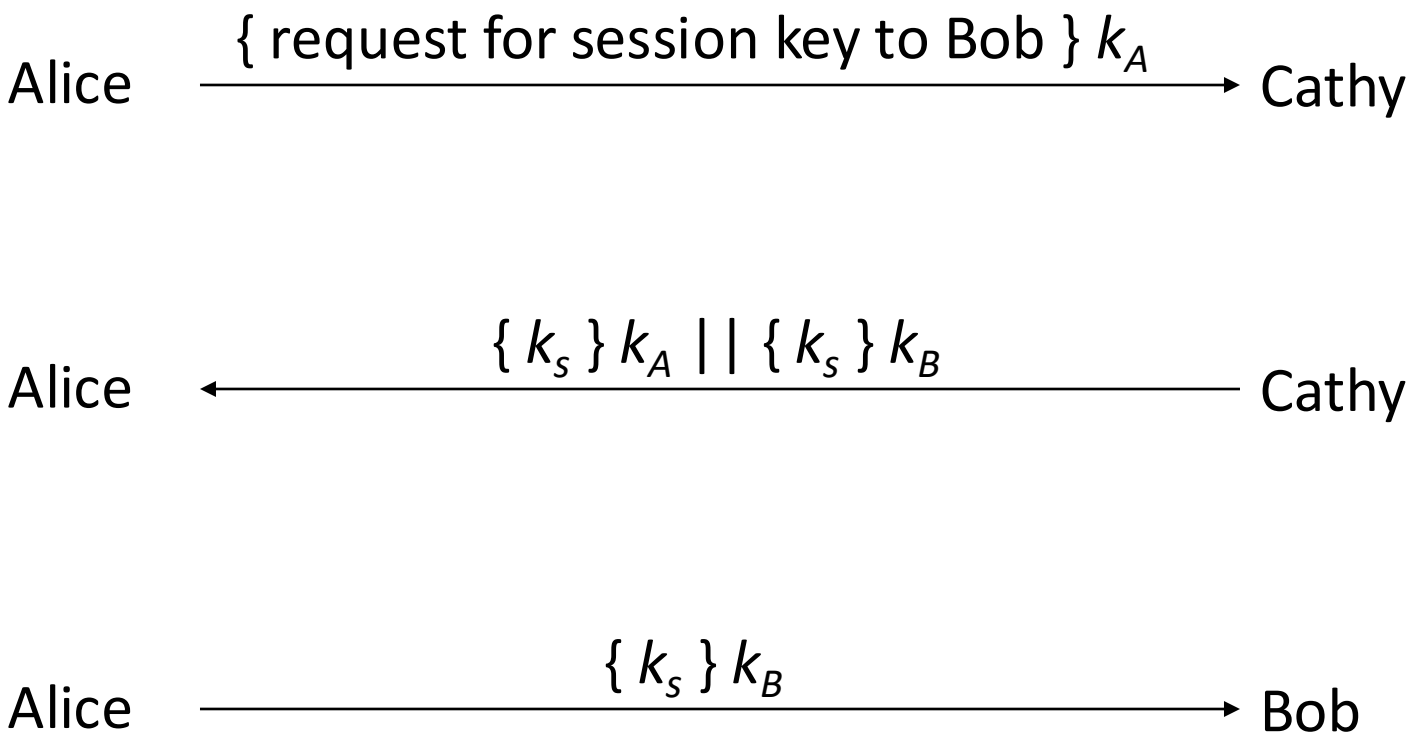
Key Exchange Algorithms

- Goal: Alice, Bob get shared key
  - Key cannot be sent in clear
    - Attacker can listen in
    - Key can be sent enciphered, or derived from exchanged data plus data not known to an eavesdropper
  - Alice, Bob may trust third party
  - All cryptosystems, protocols publicly known
    - Only secret data is the keys, ancillary information known only to Alice and Bob needed to derive keys
    - Anything transmitted is assumed known to attacker

Symmetric Key Exchange

- Bootstrap problem: how do Alice, Bob begin?
  - Alice can't send it to Bob in the clear!
- Assume trusted third party, Cathy
  - Alice and Cathy share secret key k_A
  - Bob and Cathy share secret key k_B
- Use this to exchange shared key k_s

Simple Protocol



Problems

- How does Bob know he is talking to Alice?
  - Replay attack: Eve records message from Alice to Bob, later replays it; Bob may think he's talking to Alice, but he isn't
  - Session key reuse: Eve replays message from Alice to Bob, so Bob re-uses session key
- Protocols must provide authentication and defense against replay

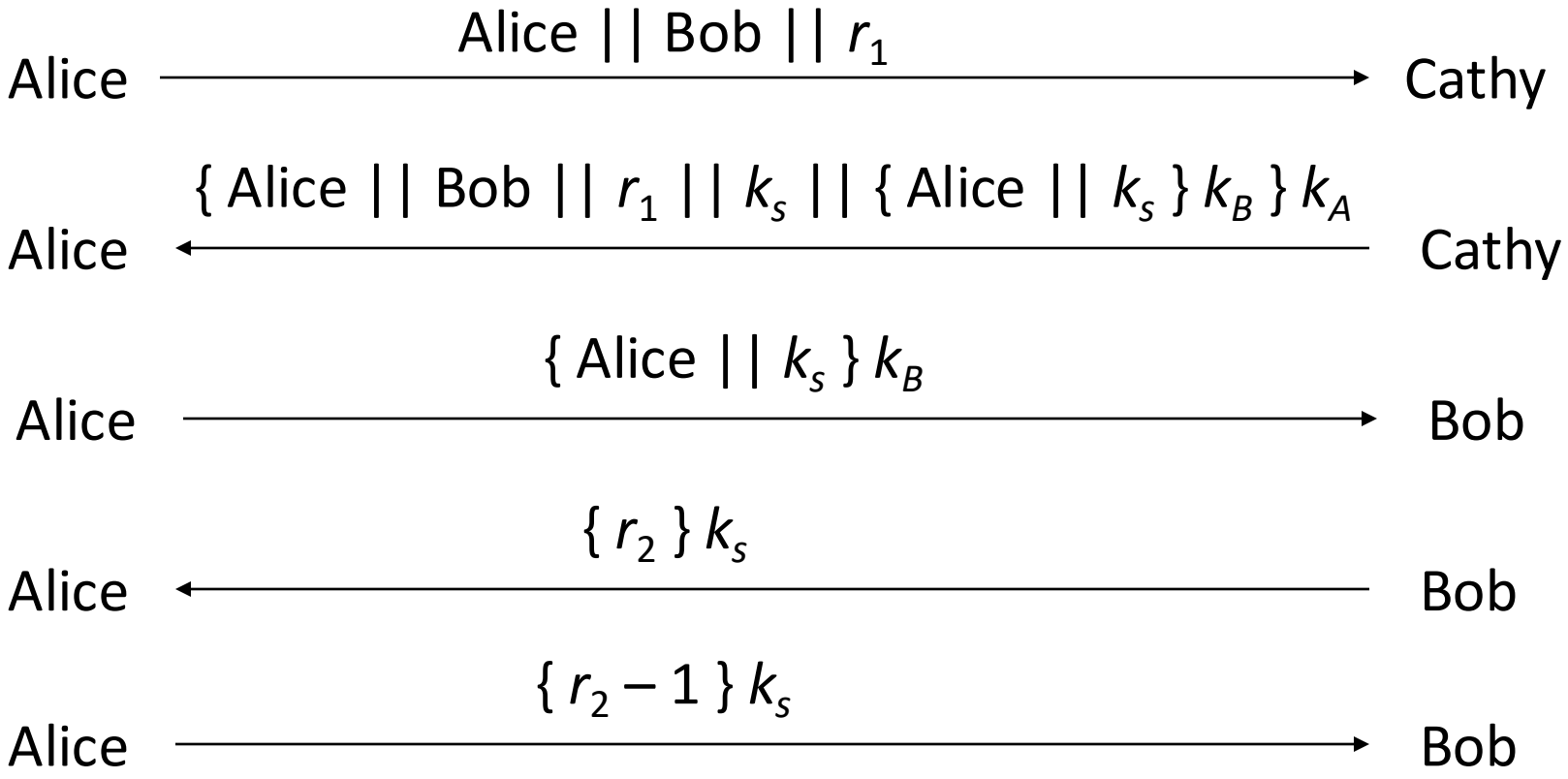
Session, Interchange Keys

- Alice wants to send a message *m* to Bob
  - Assume public key encryption
  - Alice generates a random cryptographic key k_s and uses it to encipher *m*
    - To be used for this message *only*
    - Called a *session key*
  - She enciphers k_s with Bob's public key k_B
    - k_B enciphers all session keys Alice uses to communicate with Bob
    - Called an interchange *key*
  - Alice sends $\{ m \} k_s \{ k_s \} k_B$

Benefits

- Limits amount of traffic enciphered with single key
  - Standard practice, to decrease the amount of traffic an attacker can obtain
- Prevents some attacks
  - Example: Alice will send Bob message that is either “BUY” or “SELL”. Eve computes possible ciphertexts { “BUY” } k_B and { “SELL” } k_B. Eve intercepts enciphered message, compares, and gets plaintext at once

Needham-Schroeder



Argument: Alice talking to Bob

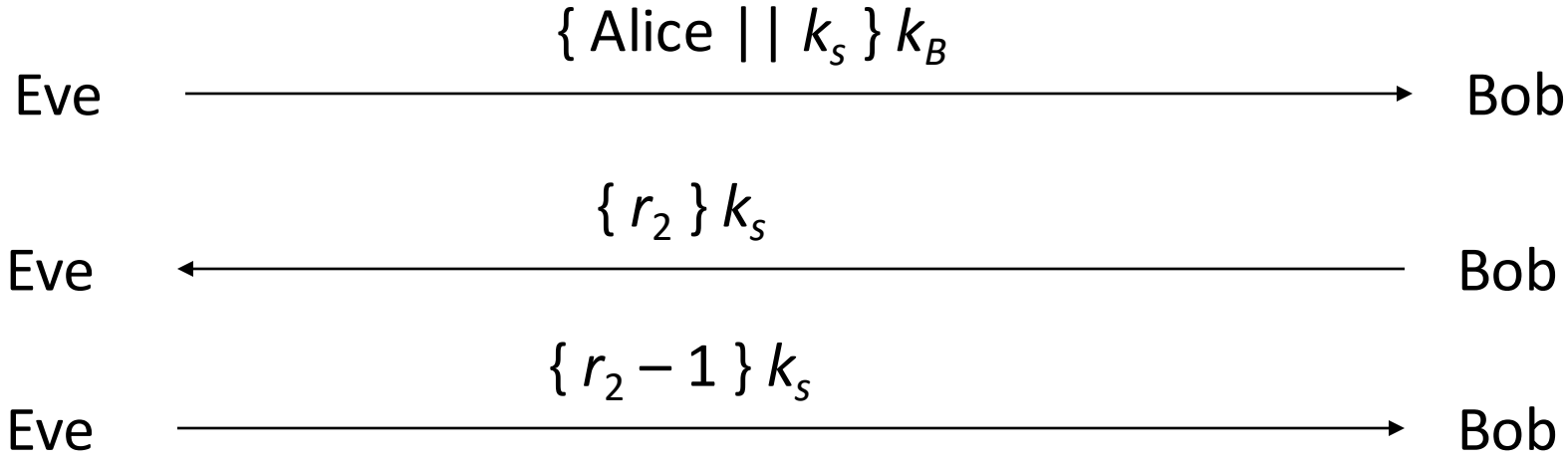
- Second message
  - Enciphered using key only she, Cathy knows
    - So Cathy enciphered it
  - Response to first message
    - As r_1 in it matches r_1 in first message
- Third message
  - Alice knows only Bob can read it
    - As only Bob can derive session key from message
  - Any messages enciphered with that key are from Bob

Argument: Bob talking to Alice

- Third message
  - Enciphered using key only he, Cathy know
    - So Cathy enciphered it
  - Names Alice, session key
    - Cathy provided session key, says Alice is other party
- Fourth message
  - Uses session key to determine if it is replay from Eve
    - If not, Alice will respond correctly in fifth message
    - If so, Eve can't decipher r_2 and so can't respond, or responds incorrectly

Denning-Sacco Modification

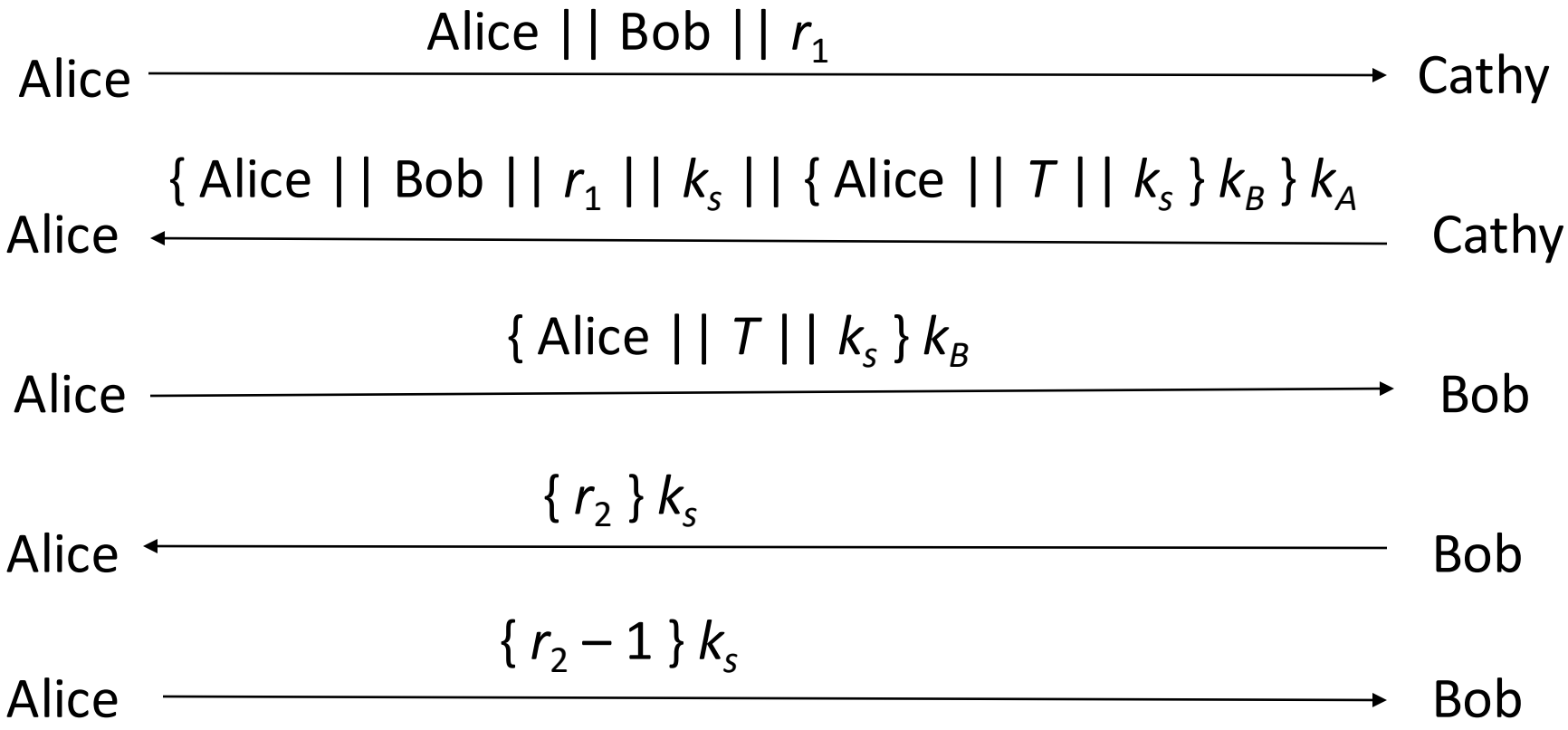
- Assumption: all keys are secret
- Question: suppose Eve can obtain session key. How does that affect protocol?
  - In what follows, Eve knows k_s



Problem and Solution

- In protocol above, Eve impersonates Alice
- Problem: replay in third step
  - First in previous slide
- Solution: use time stamp *T* to detect replay
- Weakness: if clocks not synchronized, may either reject valid messages or accept replays
  - Parties with either slow or fast clocks vulnerable to replay
  - Resetting clock does *not* eliminate vulnerability

Needham-Schroeder with Denning-Sacco Modification



Kerberos

- Authentication system
  - Based on Needham-Schroeder with Denning-Sacco modification
  - Central server plays role of trusted third party (“Cathy”)
- Ticket
  - Issuer vouches for identity of requester of service
- Authenticator
  - Identifies sender

Idea

- User *u* authenticates to Kerberos server
  - Obtains ticket $T_{u,TGS}$ for ticket granting service (TGS)
- User *u* wants to use service *s*:
  - User sends authenticator A_u, ticket $T_{u,TGS}$ to TGS asking for ticket for service
  - TGS sends ticket $T_{u,s}$ to user
  - User sends A_u, $T_{u,s}$ to server as request to use *s*
- Details follow

Ticket

- Credential saying issuer has identified ticket requester
- Example ticket issued to user *u* for service *s*

$$T_{u,s} = s \,||\, \{\, u \,||\, u\text{'s address} \,||\, \text{valid time} \,||\, k_{u,s} \,\}\, k_s$$

where:

- $k_{u,s}$ is session key for user and service
- Valid time is interval for which ticket valid
- *u*'s address may be IP address or something else
  - Note: more fields, but not relevant here

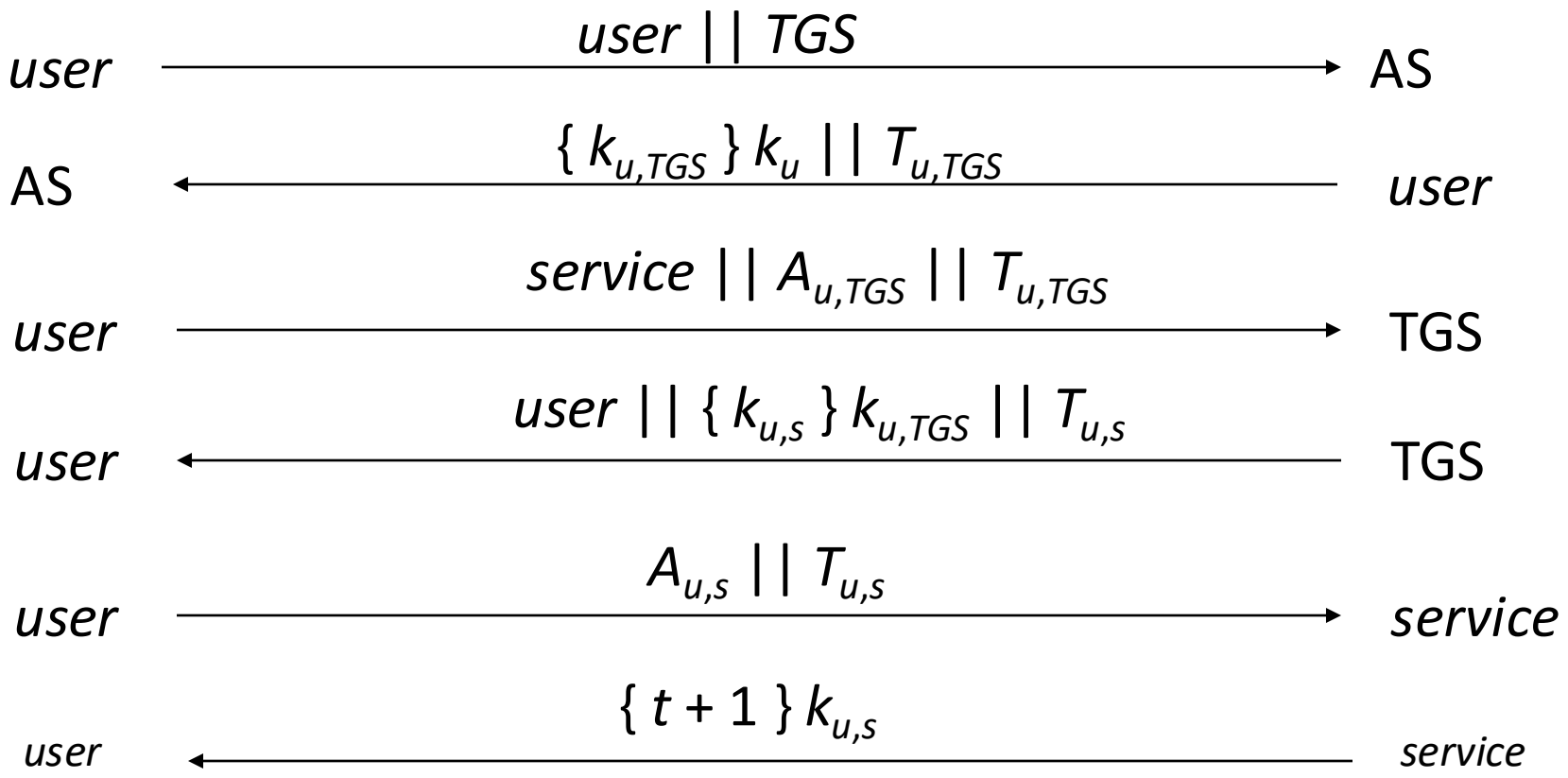
Authenticator

- Credential containing identity of sender of ticket
  - Used to confirm sender is entity to which ticket was issued
- Example: authenticator user *u* generates for service *s*

$$A_{u,s} = \{ u \,||\, \text{generation time} \,||\, k_t \}\, k_{u,s}$$

  where:
  - k_t is alternate session key
  - Generation time is when authenticator generated
    - Note: more fields, not relevant here

Protocol


user
user || TGS
AS
AS
{ $k_{u,TGS}$ } k_u || $T_{u,TGS}$
user
user
service || $A_{u,TGS}$ || $T_{u,TGS}$
TGS
user
user || { $k_{u,s}$ } $k_{u,TGS}$ || $T_{u,s}$
TGS
user
$A_{u,s}$ || $T_{u,s}$
service
user
{ t + 1 } $k_{u,s}$
service

Analysis

- First two steps get user ticket to use TGS
  - User *u* can obtain session key only if *u* knows key shared with AS
- Next four steps show how *u* gets and uses ticket for service *s*
  - Service *s* validates request by checking sender (using $A_{u,s}$) is same as entity ticket issued to
  - Step 6 optional; used when *u* requests confirmation

Problems

- Relies on synchronized clocks
  - If not synchronized and old tickets, authenticators not cached, replay is possible
- Tickets have some fixed fields
  - Dictionary attacks possible
  - Kerberos 4 session keys weak (had much less than 56 bits of randomness); researchers at Purdue found them from tickets in minutes