CSE 443: Introduction to Computer Security
Module: Authentication Protocols

Prof. Syed Rafiul Hussain
Department of Computer Science and Engineering
The Pennsylvania State University

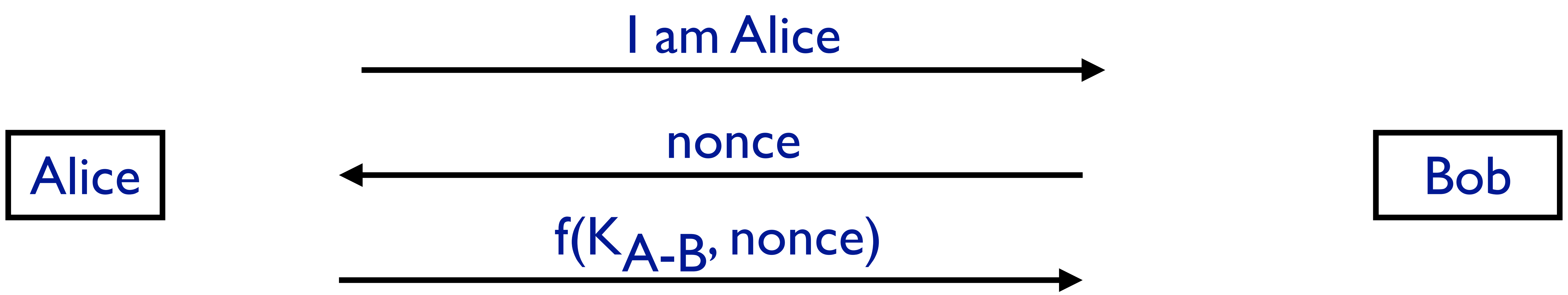
Authentication

- "Who are you"
- Long answer: evaluates the authenticity of identity proving credentials
  - Credential: is proof of identity
  - Evaluation: process that accesses the correctness of the association between credential and claimed identity
    - For some purpose
    - Under some policy (what constitutes a good credential?)

Types of Authentication Protocols

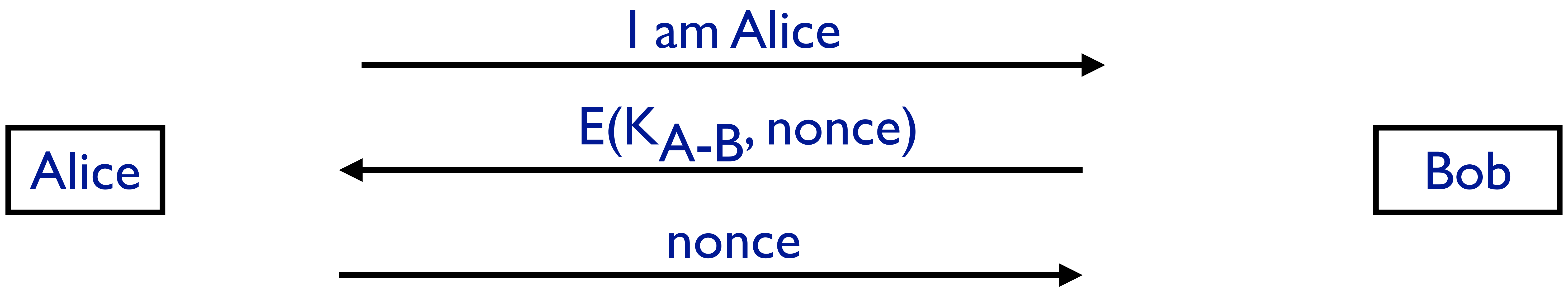
- Authentication may provide single (client, server) or mutual authentication
- Authentication may be based on:
  - Shared secret (e.g., symmetric key, password)
  - Public Key(s)

Client Authentication with Shared Secret



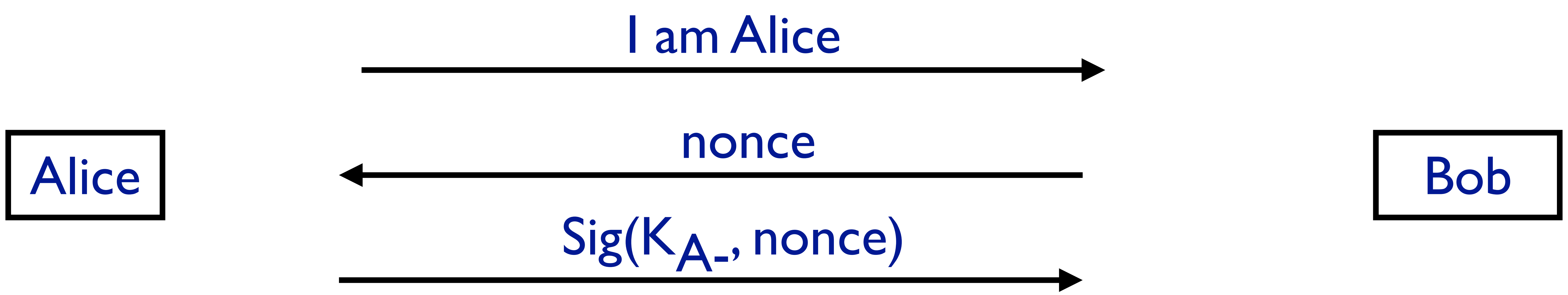
- Weaknesses?
  - Authentication is not mutual; Trudy can convince Alice she is Bob
  - Trudy can hijack conversation after initial exchange
  - If shared key from password, Trudy can mount off-line password guessing attack
  - Trudy may compromise Bob's database and later impersonate Alice

Client Authentication with Shared Secret



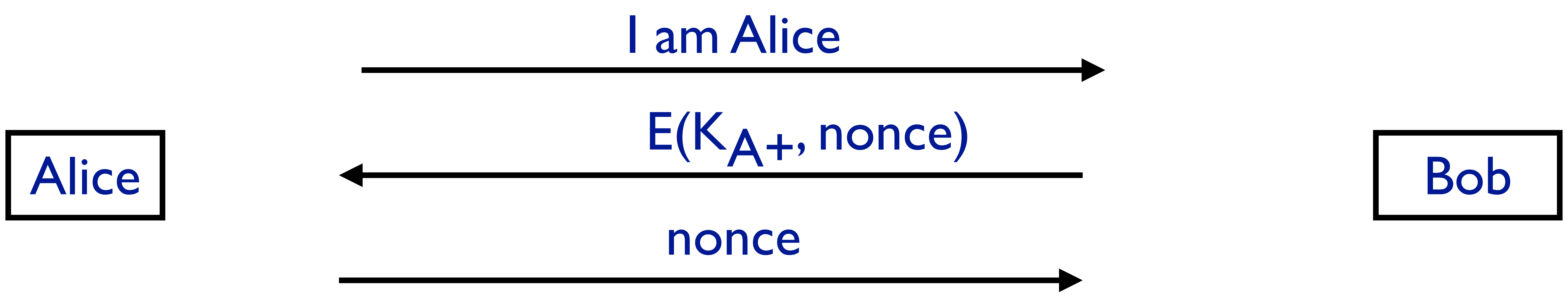
- Weaknesses?
  - All previous weaknesses remain
  - Trudy doesn't have to see nonce to mount off-line password guessing if it has certain patterns (e.g., concatenated with a timestamp)
    - Trudy can send a message to Bob, pretending to be Alice

Client Authentication with Public Key



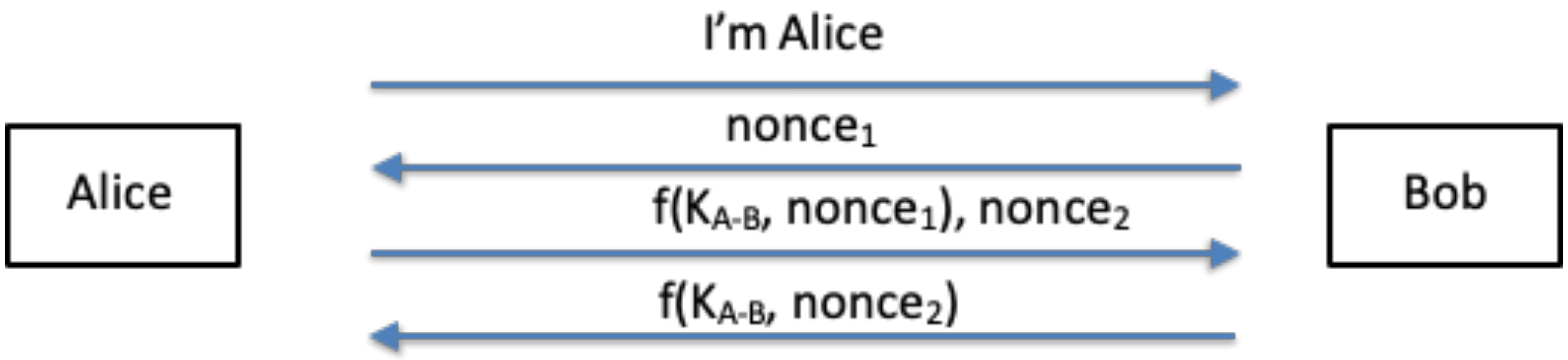
- Bob's database is less risky
- Weaknesses?
  - Authentication not mutual
  - Trudy can hijack after initial exchange
  - Trudy can trick Alice into signing something
    - Use different private key for authentication!

Client Authentication with Public Key

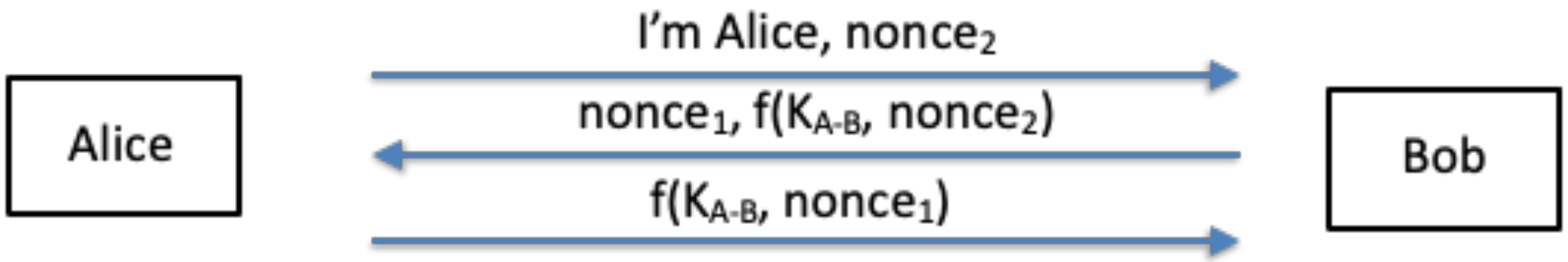


- Why is this not "Alice send E(K_{B+} , nonce)"?

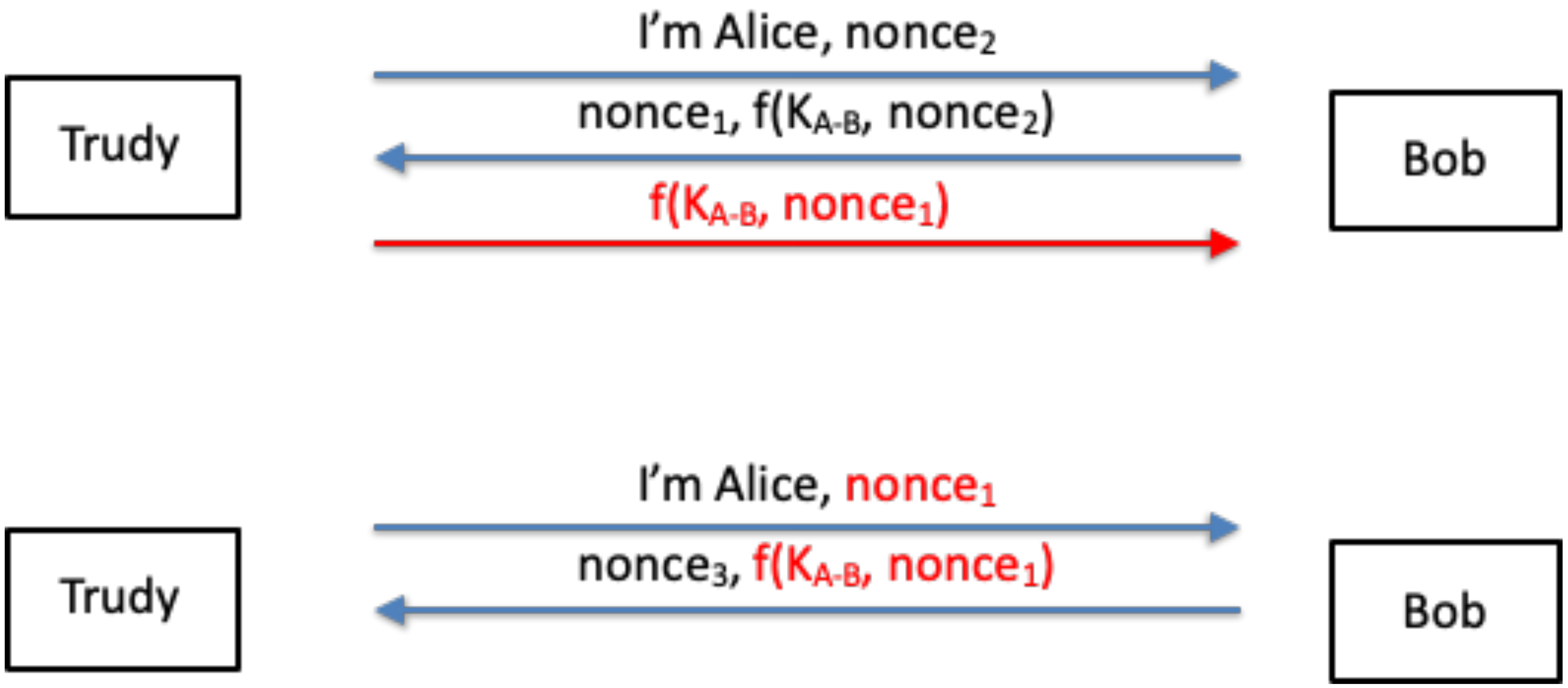
Mutual Authentication with Shared Secret



Optimize



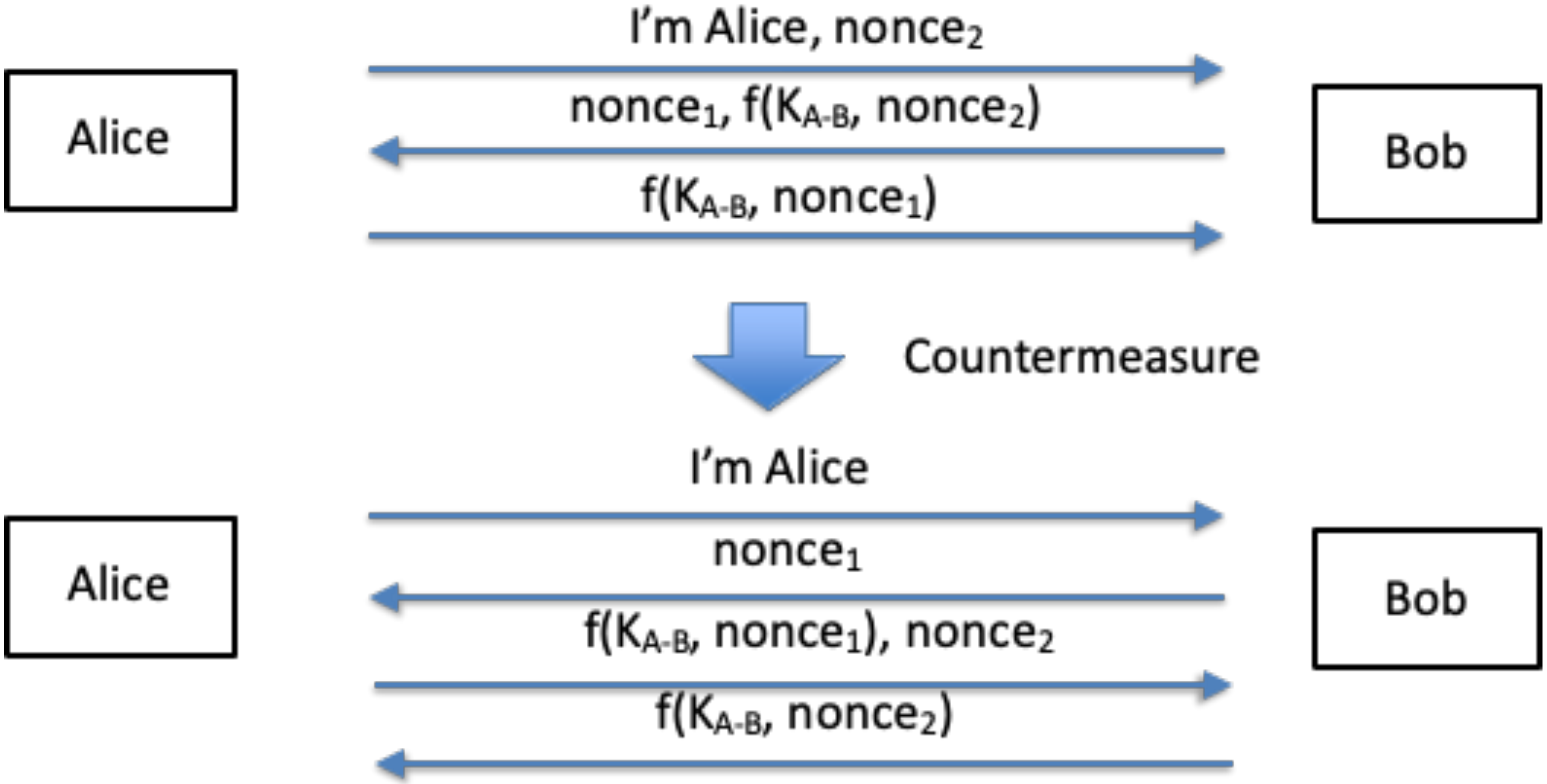
Reflection Attack



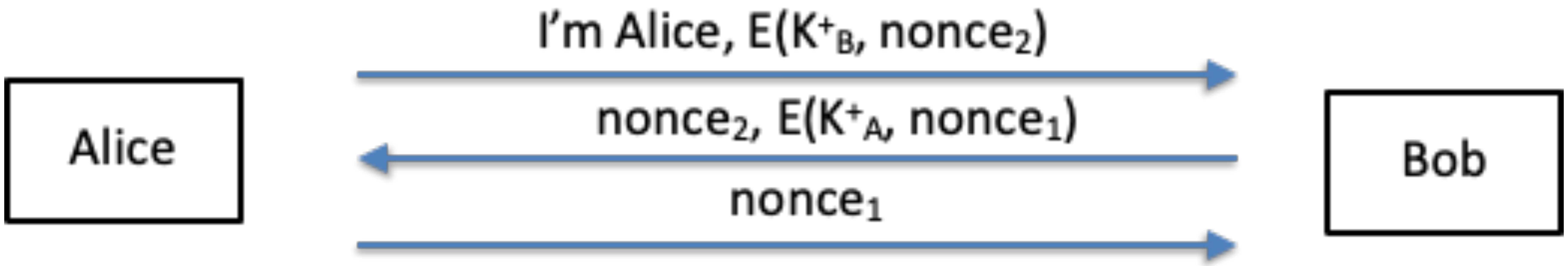
Defense against Reflection Attack

- Alice and Bob should never do exactly the same thing
  - Different keys
    - Totally different keys
    - $K_{A-B} = K_{B-A} + 1$
  - Different challenges (e.g., append "client", "server")
  - Initiator should be the first to prove its identity
    - Assumption: initiator is more likely to be the bad guy

Password Guessing

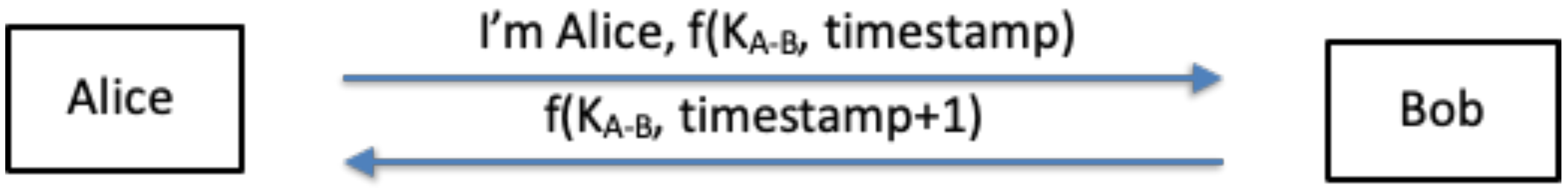


Mutual Authentication With Public Key



- Still need to authenticate public keys!
- Other variations are possible.

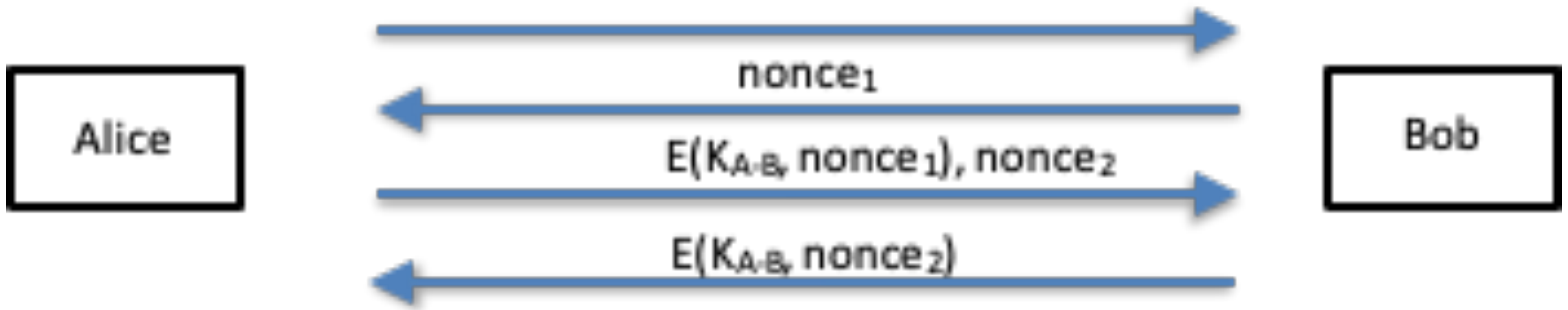
Mutual Authentication with Timestamps



- Requires synchronized clocks
- Alice and Bob must encrypt different timestamps
  - What if they use the same timestamp?

Establishment of Session Keys

- Authentication can also establish a session key to protect the confidentiality and integrity of subsequent messages
- Example: shared secret based authentication



  - Can we use $E(K_{A\text{-}B}, nonce)$ as the session key?
  - Can we use $E(K_{A\text{-}B}, nonce+1)$ as the session key?
  - Better Option: modify $K_{A\text{-}B}$ and encrypt nonce

Session Keys for Public Key

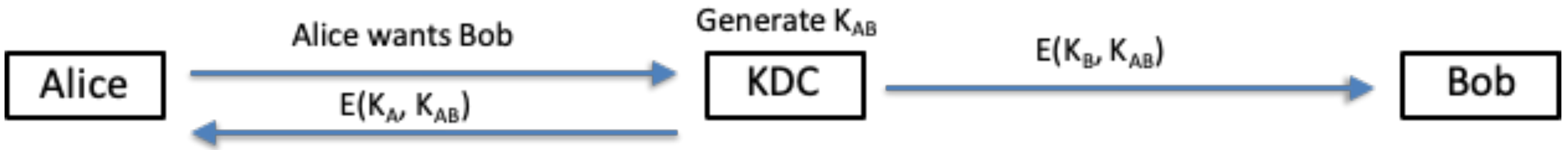
- Alice chooses random Ks, sends E(K+B, Ks) to Bob
  - Trudy may hijack the conversation
- Alice sends $E(K^{+}_{B}, Ks)$ | $Sig(K^{-}_{A}, E(K^{+}_{B}, Ks))$
  - Trudy saves traffic, decrypt after compromising Bob (less severe)
- Alice sends $E(K^{+}_{B}, R1)$; Bob sends $E(K^{+}_{A}, R2)$; $K_{s} = R1 \oplus R2$
  - Trudy has to compromise both Alice and Bob
- Alice and Bob use authenticated Diffie-Hellman
  - Trudy can’t learn session key even if compromise both

- What if only one public key is known? (e.g., Web SSL)

Mediated Authentication

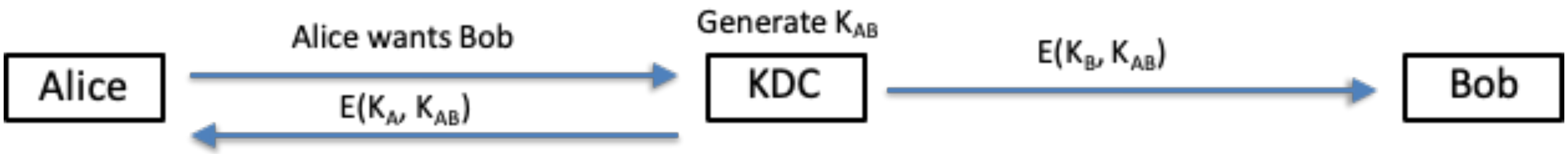
- Assume trusted third party (TTP) with shared keys with each party
- Example: Kerberos
-

Key Distribution Center (KDC)



- KDC operation (in principle)
  - KDC has a shared key with each party (e.g., K_A, K_B)
  - When Alice wants to talk to Bob, the KDC creates a new key (e.g., K_{AB}) and securely gives it to both Alice and Bob.
  - Alice and Bob then use K_{AB} for mutual authentication

KDC Concerns



- Trudy may claim to be Alice and talk to KDC
  - Trudy must not get anything useful!
- Messages encrypted by Alice may get to Bob before the KDC's message
- It may be difficult for KDC to connect to Bob

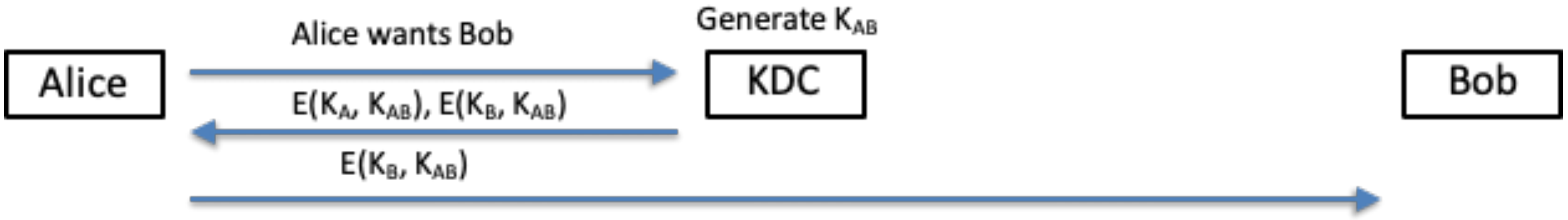
Exercise: KDC can't send to Bob directly

PennState

- How can the KDC get K_{AB} to Bob without directly sending Bob $E(K_B, K_{AB})$?
- Construct a protocol.

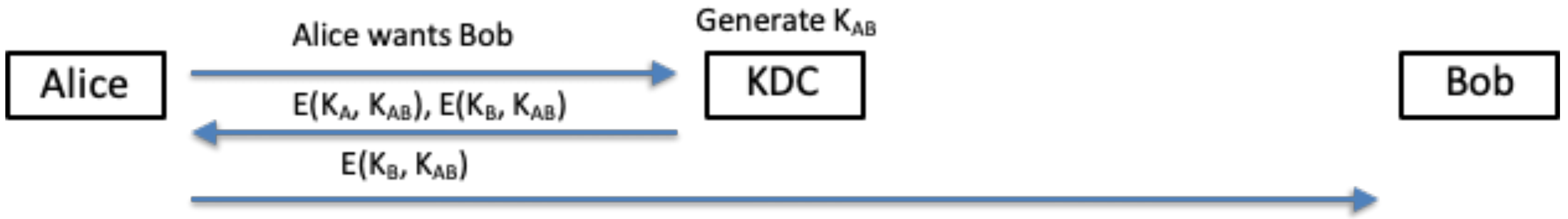
Answer: Tickets

PennState



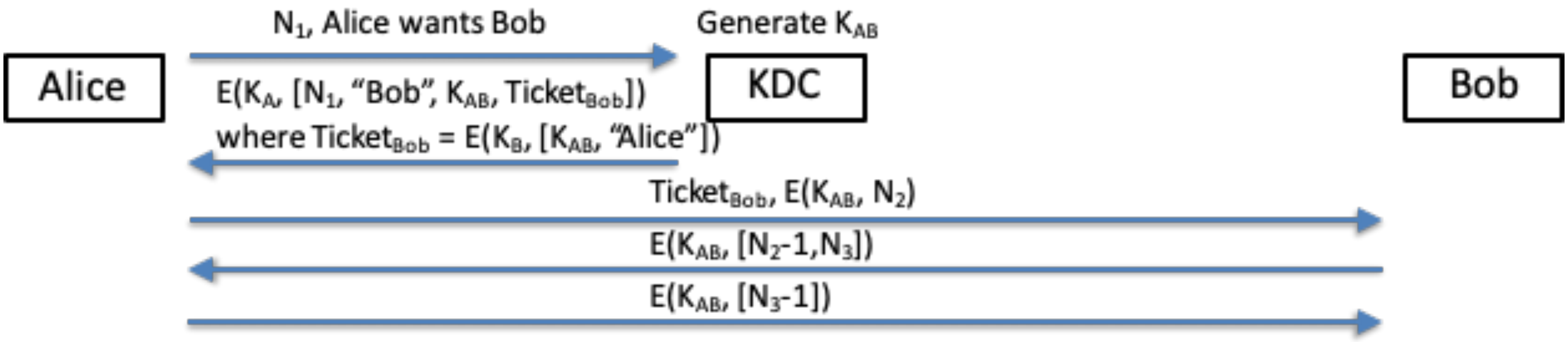
- KDC creates a ticket $E(K_B, K_{AB})$ that is relayed through Alice
  - Bob knows K_{AB} comes from KDC, because only Bob and KDC know K_B
- There are still some limitations
  - Trudy can replay $[E(K_A, K_{AB}), E(K_B, K_{AB})]$
  - Must still be followed by mutual authentication using K_{AB}

Exercise: Incorporate Mutual Authentication



- Extend the protocol to
  - Prevent replay attacks
  - Perform mutual authentication between Alice and Bob

Answer: Needham-Schroeder Protocol

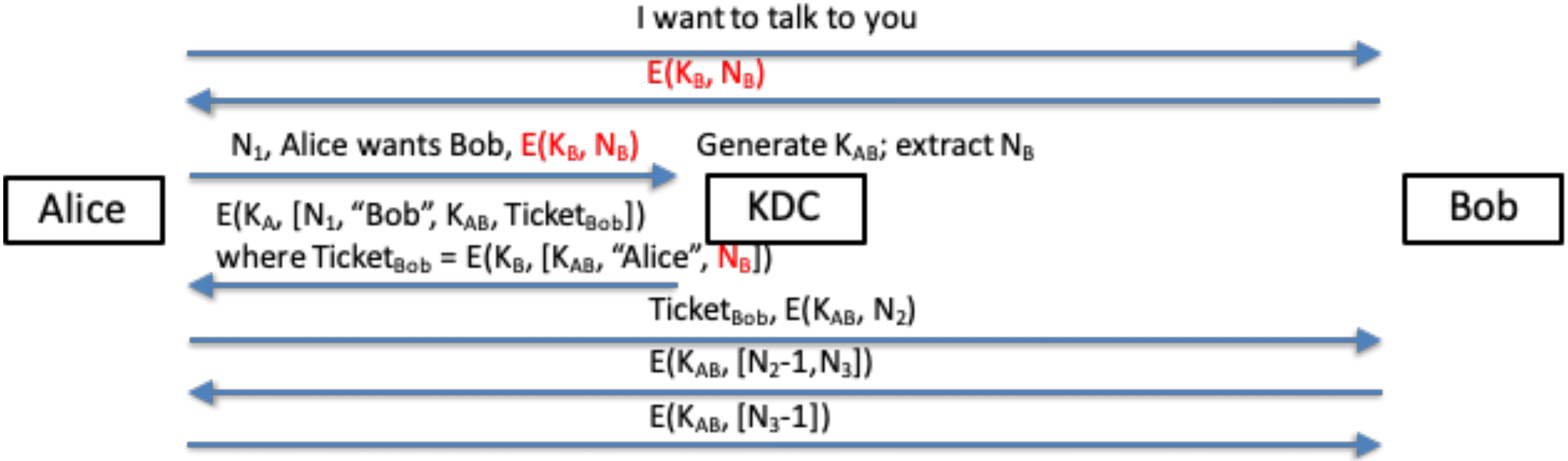


- Many others have been modeled after it (e.g., Kerberos)
  - What provides authentication?
  - N1 used to authenticate KDC to Alice
  - N2 used to authenticate Bob to Alice (has KAB, so must have KB)
  - N3 used to authenticate Alice to Bob (has KAB, which KDC gave to “Alice” in TicketBob)
    - KA needed to get TicketBob

Needham-Schroeder Vulnerability

- When Trudy gets a previous key used by Alice, Trudy may reuse a previous ticket issued to Bob for Alice
    - Ticket to Bob stays valid even if Alice changes her key
-

Expanded Needham-Schroeder



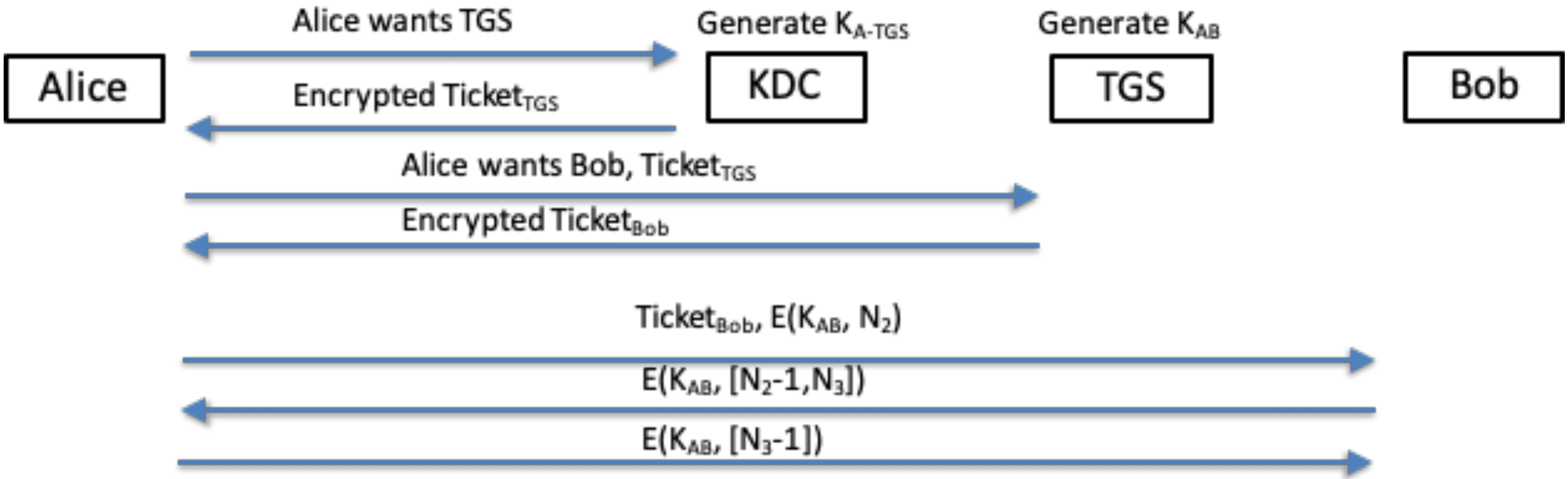
- The additional two messages assure Bob that the initiator has talk to KDC, since bob generates NB
- Other variations, e.g., Otway-Rees Protocol (see reading)

Single Sign On (SSO)

- In practice, Alice is a client workstation and Bob is a server.
  - Alice's "key" is derived from a password
- Alice will want to talk to many "Bobs" throughout the day
  - Does not want to enter password each time
  - Might be frequent (e.g., every file access, print job)

- How can Alice type her password to log into her workstation and seamlessly authenticate to servers?

Answer: Ticket Granting Ticket (TGT)



Kerberos

- An online system that resists password eavesdropping and achieves mutual authentication
- First single sign-on system (SSO)
- Most widely used (non-web) centralized password system in existence
- Easy application integration API
  - Now part of Windows Active Directory
- Provides both authentication and authorization
-

Kerberos Tickets

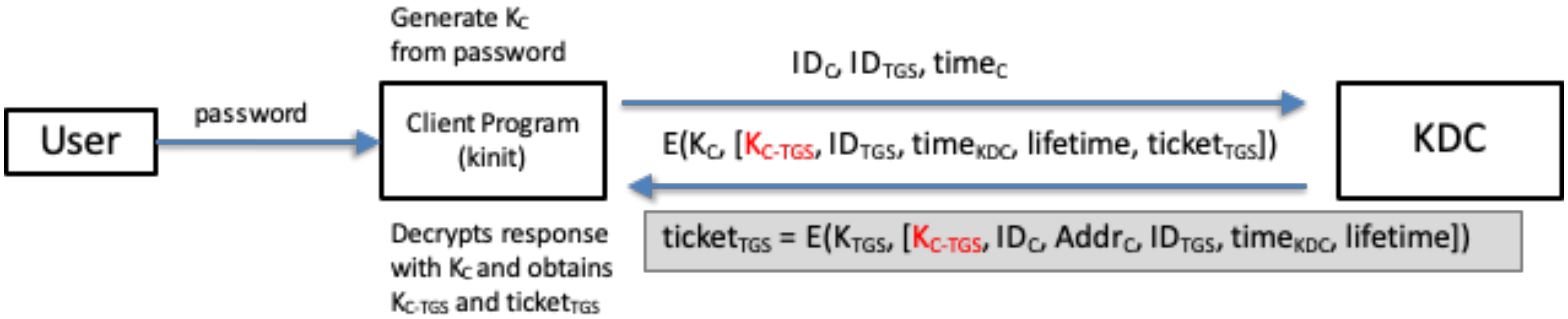
- The ticket includes (amongst other fields):
  - Username – server must verify ticket is for the stated user
  - Server name – server must verify the ticket is for itself
  - IP address of workstation (why?)
  - Ticket lifetime (why?)
  - Session key

- Ticket hijacking is still possible in certain cases
  - Malicious user may steal the service ticket of another user on the same workstation and use it
  - Need to handle freshness as part of the Kerberos protocol

Kerberos Symmetric Keys

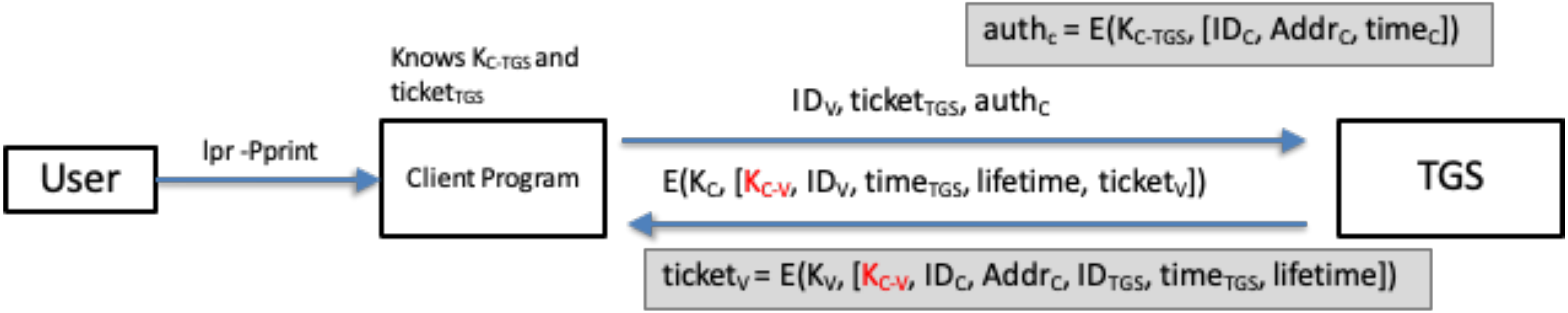
- K_C is long-term key of client C
  - Derived from user's password
  - Known to client and KDC
- K_{TGS} is long-term key of TGS
  - Known to KDC and TGS
- K_V is long-term key of network service V
  - Known to V and TGS; separate key for each service
- $K_{C\text{-}TGS}$ is short-term session key between C and TGS
  - Created by KDC, known to C and TGS
- $K_{C\text{-}V}$ is short-term session key between C and V
  - Created by TGS, known to C and V

Simplified Kerberos – Single Login

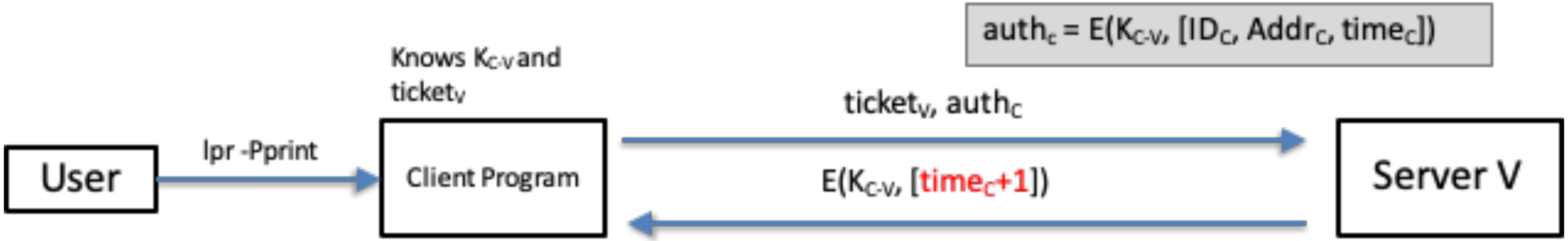


- Client only needs to obtain TGS ticket once (say every morning)
  - Ticket is encrypted; client cannot forge it or tamper with it

Simplified Kerberos – Service Ticket



- Client uses TGS ticket to obtain a service ticket and a short-term key for each network service
    - One encrypted, unforgeable ticket per service (printer, email, etc)
-



- For each service request, client uses short-term key for service and the ticket received from TGS
- Authenticates server to client, because
  - Server can produce this message only if it knows K_{C-V}
  - Server can learn K_{C-V} only if it can decrypt $ticket_V$
  - Server can decrypt $ticket_V$ only if it knows the correct K_V
  - If server knows correct K_V, the it is the right server
- Authenticates client to server – why?
  - Recall $ticket_V = E(K_V, [K_{C-V}, ID_C, Addr_C, ID_{TGS}, time_{TGS}, lifetime])$

Kerberos Security

- Key storage issues
  - KDC is the focal point of security
  - However, user passwords and session keys may be stolen on compromised clients
  - Password cracking was done on Windows Kerberos messages
- Timestamps are an issue (not nonces like NH)
  - Don't have to track what nonces have been used
  - Authenticators use timestamps as challenge-responses
  - However, timestamps are accepted with range of minutes
- Some crypto attacks have been proposed
- Despite these, Kerberos broadly used
  - Not the lowest hanging fruit

Secure SHell

- Secure login, file transfer, X11, TCP/IP over Internet

- Replaces old insecure protocols for such things that used passwords in cleartext

- Uses strong cryptography for communication
  - RSA is used for key exchange and authentication
  - Symmetric algorithms for data security

Basic SSH Protocol

- (1) Client opens connection to server
- (2) Server sends public *host key*
  - Enables approval of new hosts
  - Rejects changed host keys
  - Notifies on expired host keys
- (3) Client generates random number as session key
  - Encrypts for the server using the host key
- (4) Server decrypts the session key
  - Confirms receipt (authenticating itself to the client)
- (5) Client can then authenticate using traditional means
  - E.g., Password

SSH Security

- Client encrypts session key in server's host key
  - Q: Does this guarantee integrity?
  - Q: Can you prove that this is not susceptible to man-in-middle attacks?

- In SSH v2, communication is protected via HMAC-SHA1
  - You should be able to write these messages

SSH Services

- Value of SSH comes from the services that it runs…
  - Remote services
    - scp, sftp, …
  - Support for connections
    - X11 forwarding, TCP forwarding, …
- Over a secure channel…
  - Using strong crypto
- And it's straightforward to setup the server and easy for clients
  - Has to deal with a modest number of error cases

SSH Vulnerabilities

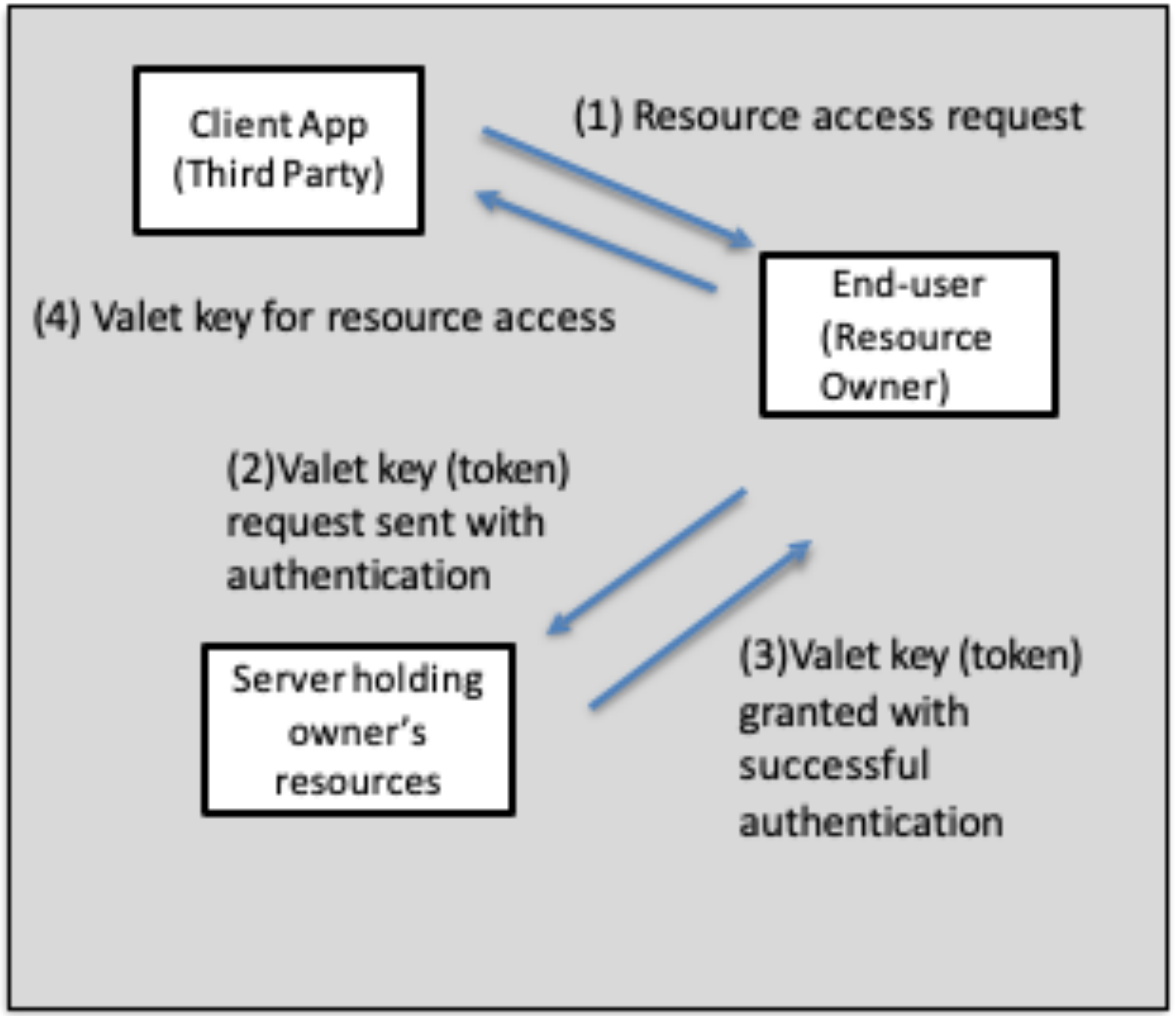
- The communication is secure, so what to attack...
- Several problems: circa 2001-2002
  - Buffer Overflows (sshd runs as root)
    - Several of these
  - Integer overflows
  - Confuse the program (ssh-agent on client runs as root)
  - Also, attack the client side (run as client)
  - DoS attacks

- OpenSSH system has been rearchitected

- Q: *We'll talk about how to fix these problems later...*

- •OAuth is an open standard foraccess delegation, not authentication
- … but it is frequently used for authentication
- Sign on with {Google, Facebook}
- How?

- Somewhat like Kerberos for the Web, without the key distribution part
  - Everything is based on “tokens”
  - Problem: What if client does not properly verify the token?



Sun and Beznosov, **The Devil is in the (Implementation) Details: An Empirical Analysis of OAuth SSO Systems**. In Proc of ACM CCS 2012.

Chen et al., **OAuth Demystified for Mobile Application Developers**., In Proc. ACM CCS 2014.

Take Away

- Systems for authentication have been constructed
  - Powerful, broadly used
  - Cryptography is generally above reproach
  - System challenges
    - Kerberos timestamps
    - Key storage
    - System security
- Communication is probably not not th