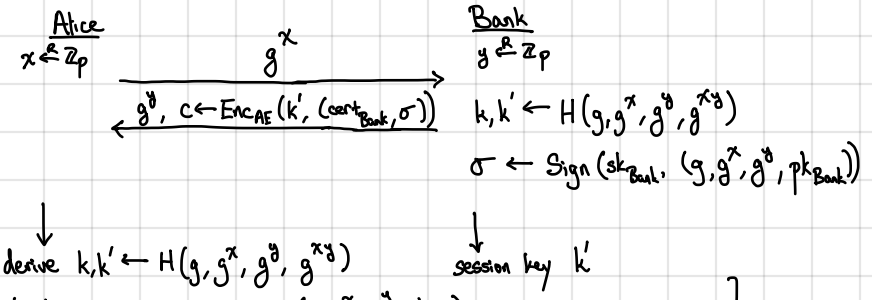
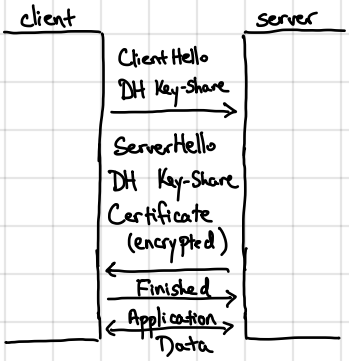
Basic flow of Diffie-Hellman based AKE:



Intuition: $cert_{Bank}$ identifies server as Bank (with pk_{Bank})
σ binds the session parameters (g, g^x, g^y) to the public key identified by $cert_{Bank}$

End of protocol: Alice knows she is talking to Bank (but not vice versa!)
"one-sided AKE" — most common mode on the web

↳ Basis of TLS 1.3 handshake ("one-sided" AKE) ALWAYS USE TLS 1.3 - Don't invent your own AKE protocols!

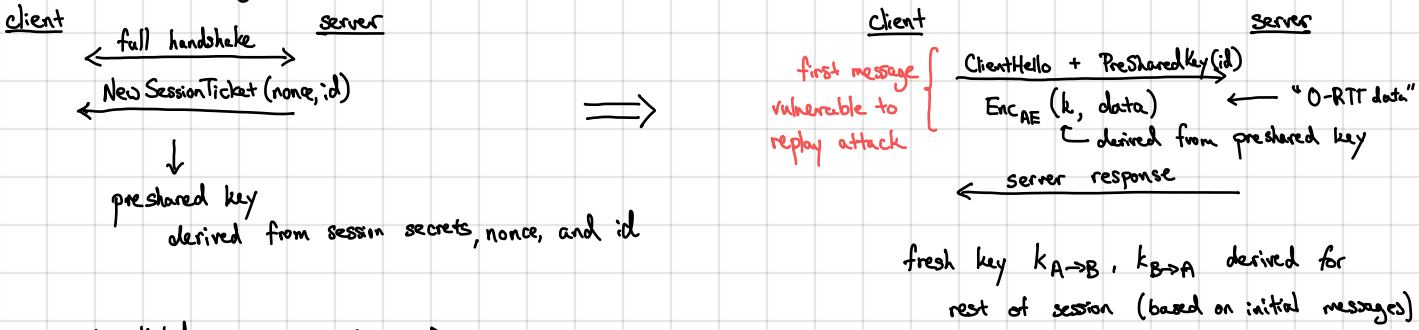


ClientHello: List of supported ciphersuites
(e.g., AES-GCM-128, AES-GCM-256)
Possible TLS extensions
ServerHello: Chosen ciphersuite

older systems / foreign systems may prefer different ciphers
older versions of TLS vulnerable to cipher downgrade attacks

Application layer secured using unidirection keys $k_{A \to B}$ and $k_{B \to A}$

TLS supports session setup using a "pre-shared key" (so full handshake not needed):



negotiated key — identity of peer

Output of AKE protocol: (key, id)

Authenticity: Only party that knows key is id (i.e., the party identified by id)
Secrecy: All parties other than client and id cannot distinguish key from random (i.e., key is hidden)
Consistency: If id also completes protocol, then it outputs (key, id_{client})
↳ if we do not have client authentication, then id_{client} is empty

Often also require forward secrecy: compromise of server in the future cannot affect secrecy of sessions in the past
↳ In TLS, server secret is a signing key — fresh Diffie-Hellman secret used for each session is fresh ("ephemeral")
Compromising signing key allows impersonation of server, but does not break secrecy of past sessions
↳ As we will see, not all AKE protocols provide forward secrecy (e.g., if server holds a public key and client simply encrypts an encryption key with respect to the server's public key — compromise of server would compromise all past communications)

Replay protection: adversary cannot replay messages from a previous session since server chooses a fresh Diffie-Hellman key-share on each interaction

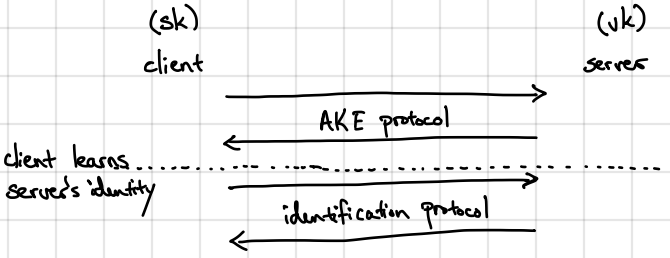
[Within the record layer itself, there is a counter (sequence number) associated with each message which is authenticated — protects against replay of record layer messages]

Very tricky to get right as we will see... Just use TLS!

TLS 1.3 and authenticated key-exchange protocols on the Internet typically provide one-sided authentication (i.e., client learns id of the server, but not vice versa)

Question: how does the client authenticate to the server (without providing a certificate)
↳ e.g., how does client login to a web service?

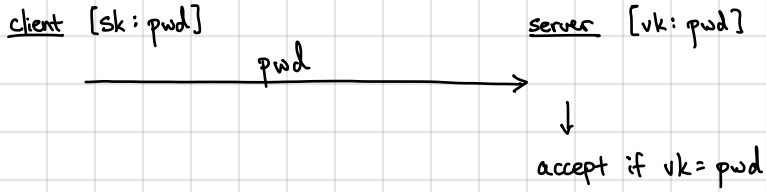
Typical setting: client and server assumed to have some shared state [e.g., client has a password and server has an HMAC of the password]



} cannot replace this with anonymous key exchange
↳ becomes vulnerable to a man-in-the-middle attack

Threat models: Adversary's goal is to authenticate to server
- Direct attack: adversary only sees vk and needs to authenticate
  (e.g., physical analogy: door lock — adversary can observe the lock, does not see the key sk)
- Eavesdropping attack: adversary gets to observe multiple interactions between honest client and the server
  (e.g., physical analogy: wireless car key — adversary observes communication between car key and car)
- Active attack: adversary can impersonate the server and interact with the honest client
  (e.g., physical analogy: fake ATM in the mall — honest clients interact directly with the adversary)

Simple (insecure) password-based protocol:



Not secure even against direct attacks! Adversary who learns vk can authenticate as the client [adversary who breaks into server learns user's password!]

NEVER STORE PASSWORDS IN THE CLEAR!