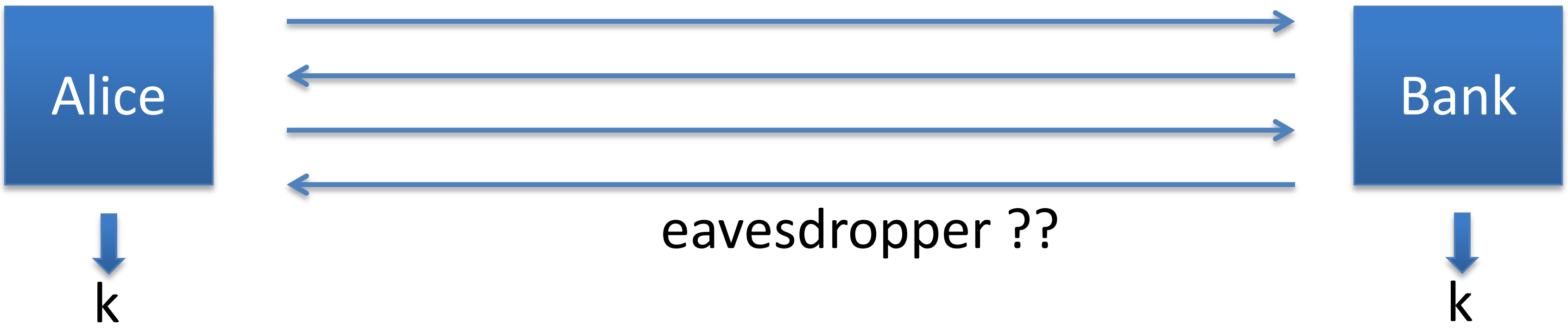
# Auth. Key Exchange

# Review: key exchange

Alice and Bank want to generate a secret key

- So far we saw key exchange secure against eavesdropping




- This lecture: **Authenticated Key Exchange (AKE)**

  key exchange secure against <u>active</u> adversaries

# Active adversary

Adversary has complete control of the network:

- Can modify, inject and delete packets
- Example: man-in-the-middle

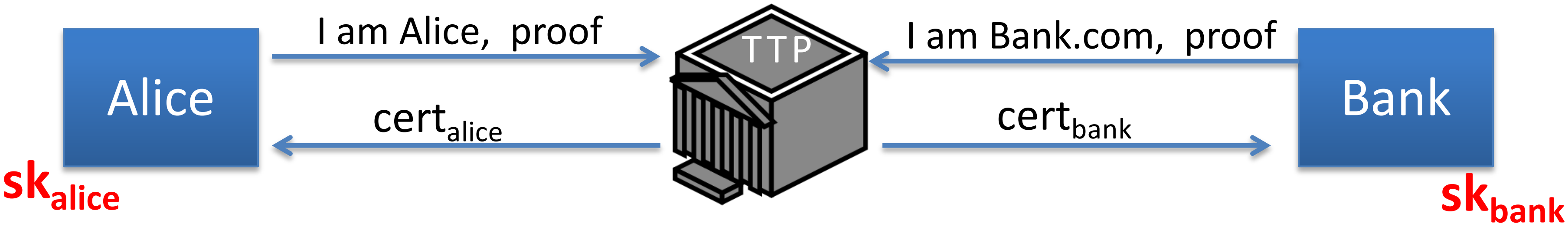

Moreover, some users are honest and others are corrupt

- Corrupt users are controlled by the adversary
  – Key exch. with corrupt users should not “affect” other sessions

# Trusted Third Party (TTP)

All AKE protocols require a TTP to certify user identities.

Registration process:




Two types of TTP: (here, we only consider offline TTP)

- **Offline TTP (CA)**: contacted only during registration (and revocation)
- **Online TTP**: actively participates in every key exchange (Kerberos)

  Benefit: security using only symmetric crypto

# AKE: syntax


Followed by Alice sending E(k, "data") to Bank and vice versa.

# Basic AKE security (very informal)

Suppose Alice successfully completes an AKE to obtain **(k, Bank)**

If Bank is not corrupt then:

**Authenticity** for Alice: (similarly for Bank)

- If Alice's key k is shared with anyone, it is only shared with Bank

**Secrecy** for Alice: (similarly for Bank)

- To the adversary, Alice's key k is indistinguishable from random
  (even if adversary sees keys from other instances of Alice or Bank)

**Consistency**: if Bank completes AKE then it obtains **(k, Alice)**

# AKE security levels (very informal)

Three levels of (core) security:

- **Static security**: previous slide
- **Forward secrecy**: static security, and if adv. learns $sk_{bank}$ at time T then all sessions with Bank from time t<T remain secret.
- **HSM security**: if adv. queries an HSM holding $sk_{bank}$ n times, then at most n sessions are compromised. Moreover, forward secrecy holds.

Several other AKE requirements …

Hardware Security Module (HSM)

# One-sided AKE: syntax


Used when <u>only</u> one side has a certificate.

- Similarly, three security levels.

# Things to remember …

Do not design AKE protocol yourself …

## Just use latest version of TLS

# Building blocks

**cert$_{bank}$:** contains $pk_{bank}$ . Bank has **$sk_{bank}$** .

**$E_{bank}((m,r)) = E(pk_{bank}, (m,r))$** where E is *chosen-ciphertext secure*

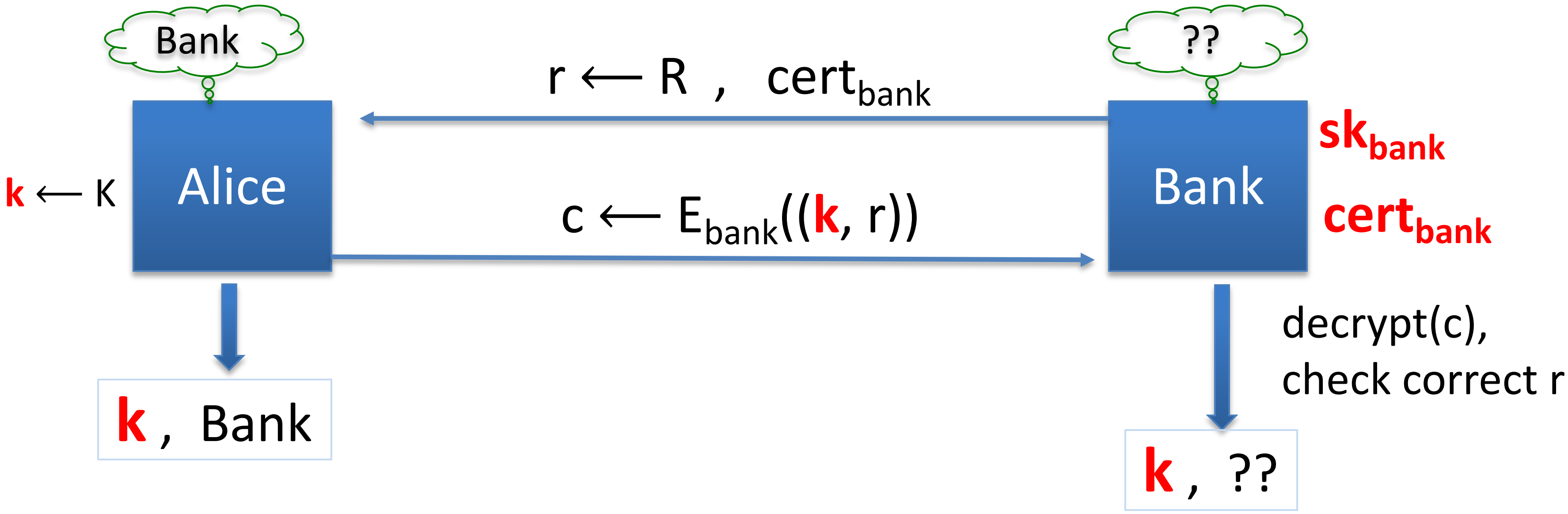
- Recall: from $E_{bank}((m,r))$ adv. cannot build $E_{bank}((m,r'))$ for $r' \neq r$

**$S_{alice}((m,r)) = S( sk_{alice}, (m,r) )$** where S is a secure signing alg.

**R**: some large set, e.g. $\{0,1\}^{256}$

# Protocol #1

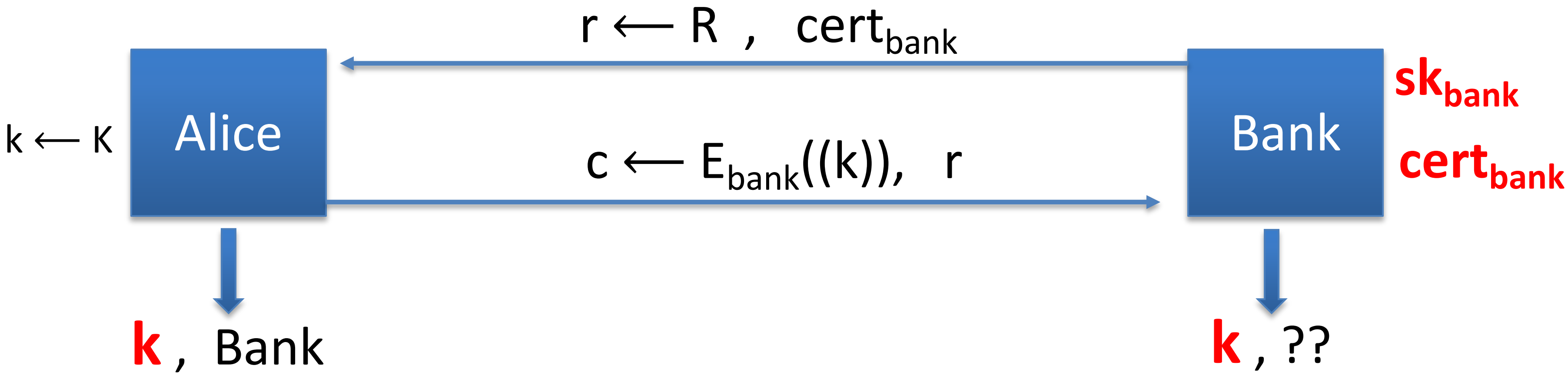
# Simple one-sided AKE protocol




"Thm": protocol is a statically secure one-sided AKE

Informally: if Alice and Bank are not corrupt then we have (1) secrecy for Alice and (2) authenticity for Alice

# Insecure variant 1: r not encrypted




Problem: replay attack

# Replay attack

Alice: $k \leftarrow K$

Bank → Alice: $r \leftarrow R$ , $cert_{bank}$

Alice → Bank: $c \leftarrow E_{bank}((k))$, $r$

Alice → Bank: $c_1 \leftarrow E_{sym}(k,$ "I am Alice, pay Bob 30$")

Bank: $sk_{bank}$, $cert_{bank}$

Later:

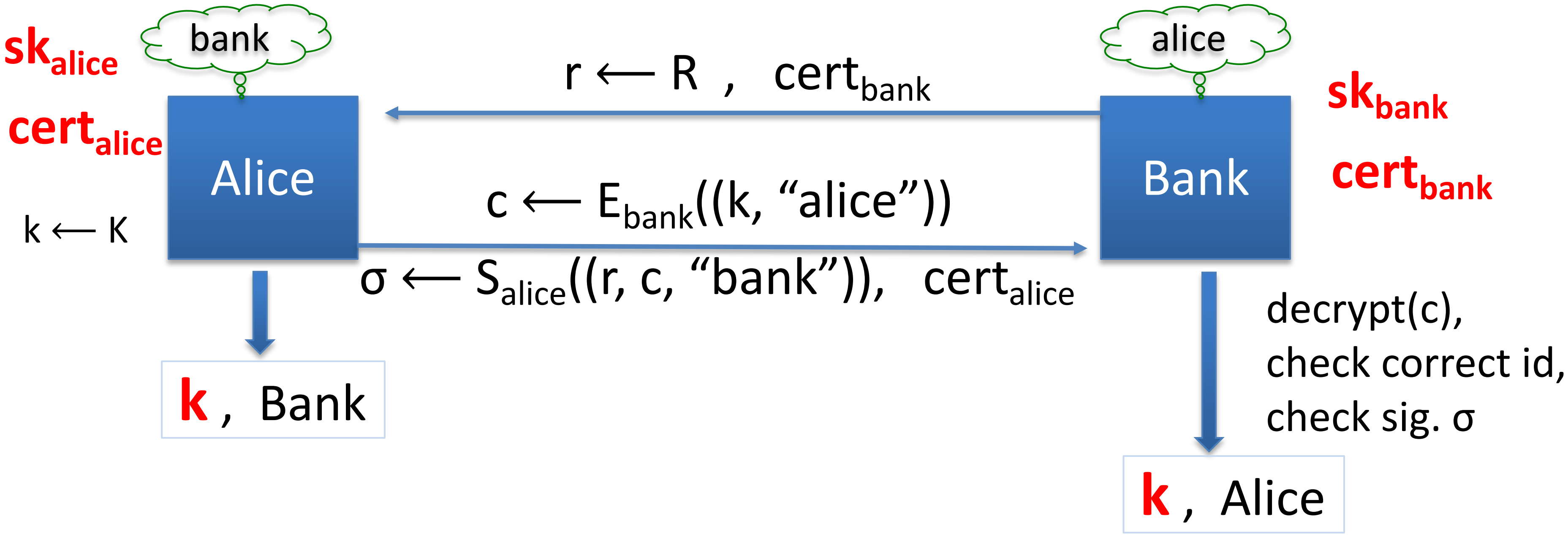
Bank → attacker: $r' \leftarrow R$ , $cert_{bank}$

Attacker → Bank: $c$, $r'$

Attacker → Bank: $c_1$

Bank: $sk_{bank}$, $cert_{bank}$

# Two-sided AKE (mutual authentication)

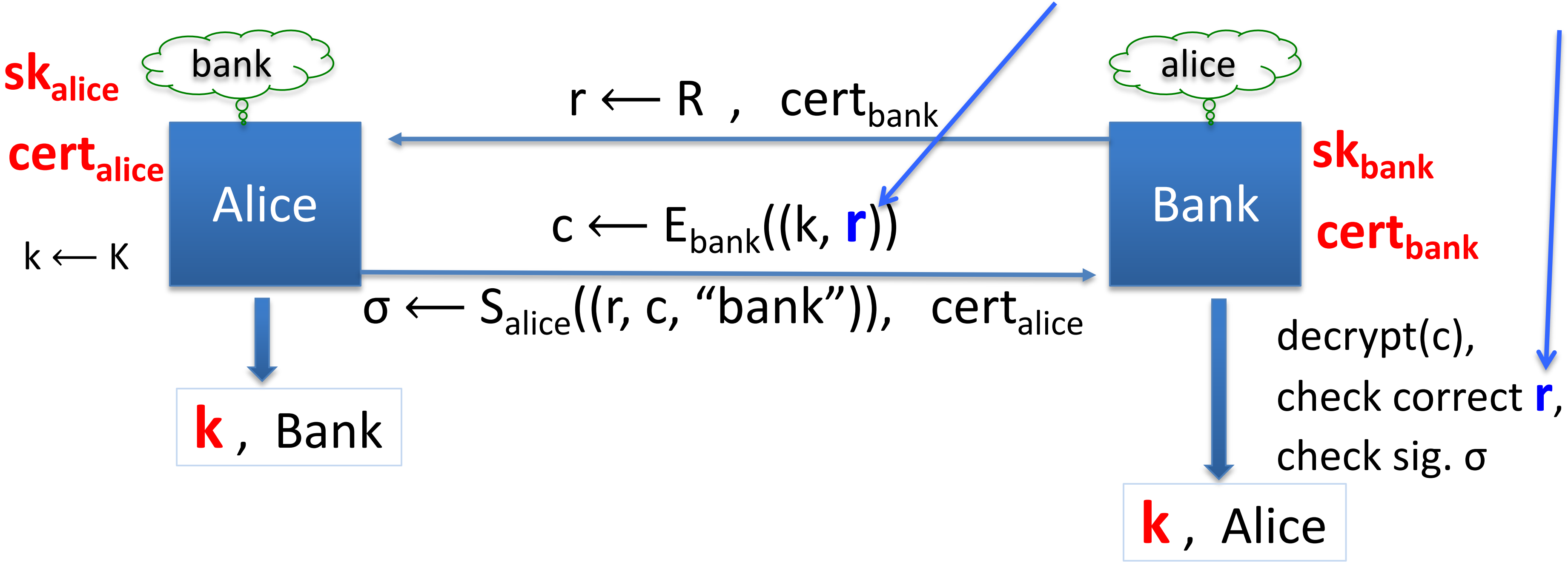



“Thm”: this protocol is a statically secure AKE

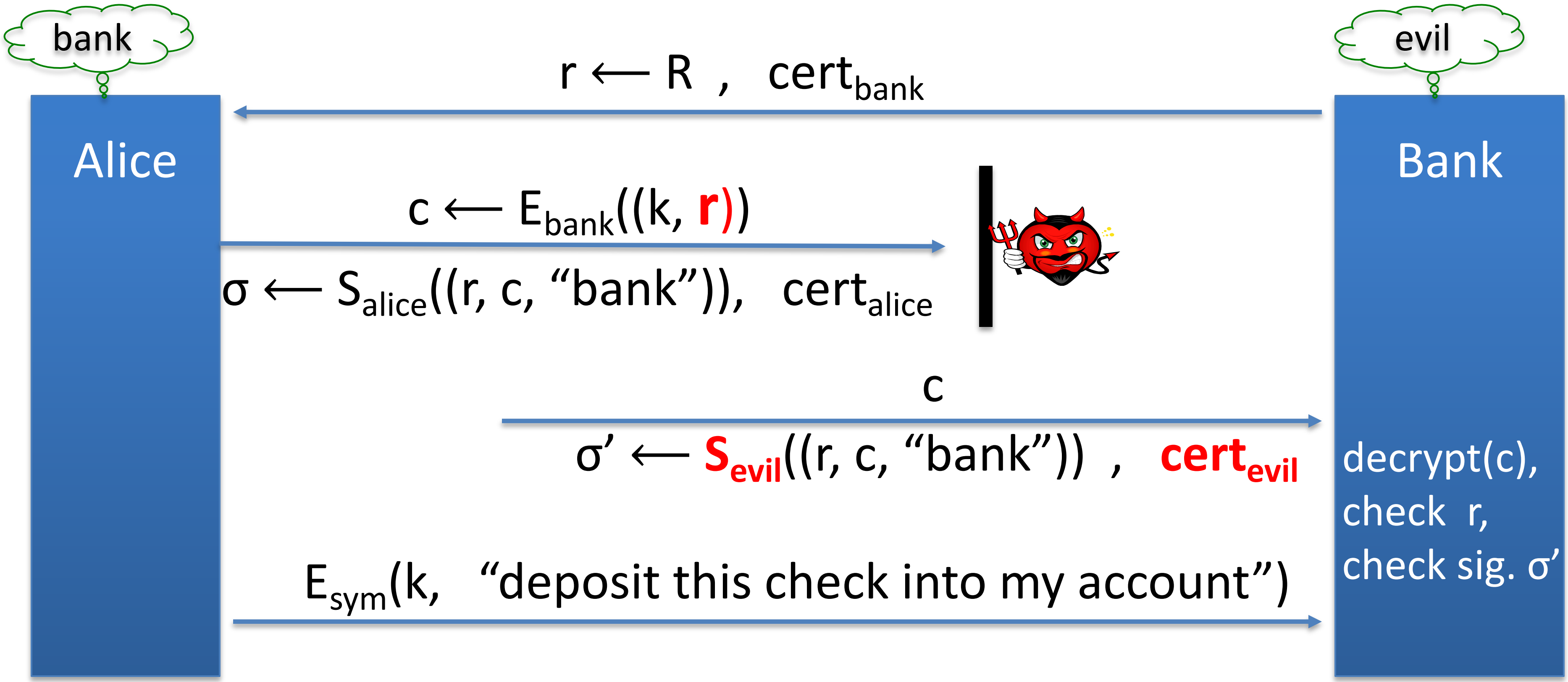
# Insecure variant: encrypt r instead of "Alice"

Any change to protocol makes it insecure, sometime in subtle ways

Example:

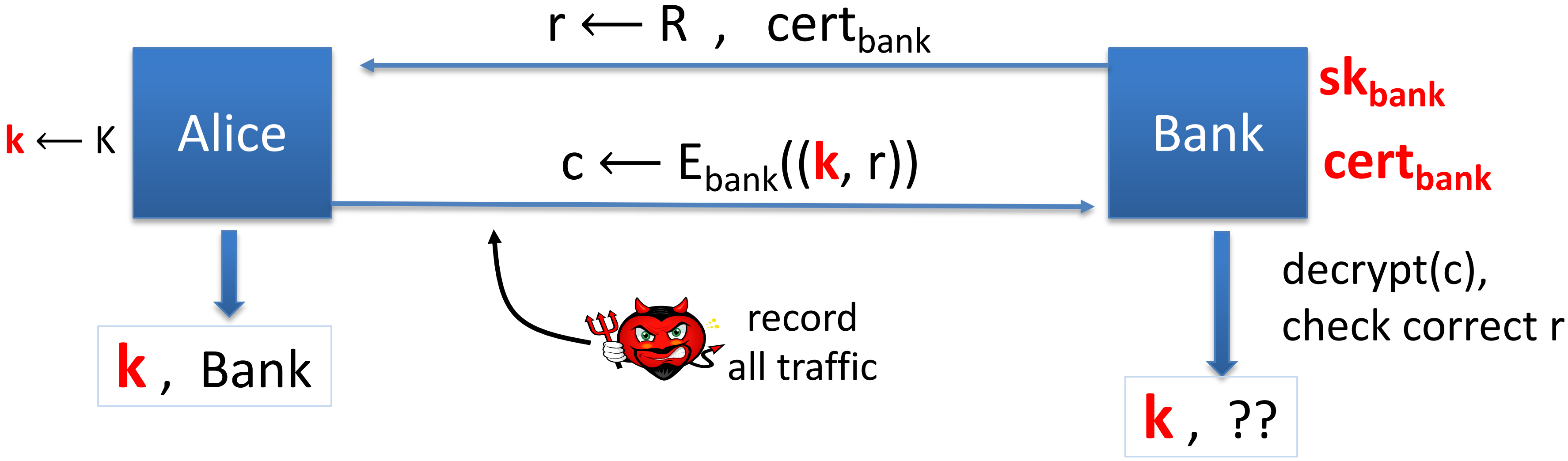


# Attack: identity misbinding



# Problem: no forward secrecy

Recall the one-sided AKE:




Suppose a year later adversary obtains $sk_{bank}$

$\Rightarrow$ can decrypt all recorded traffic

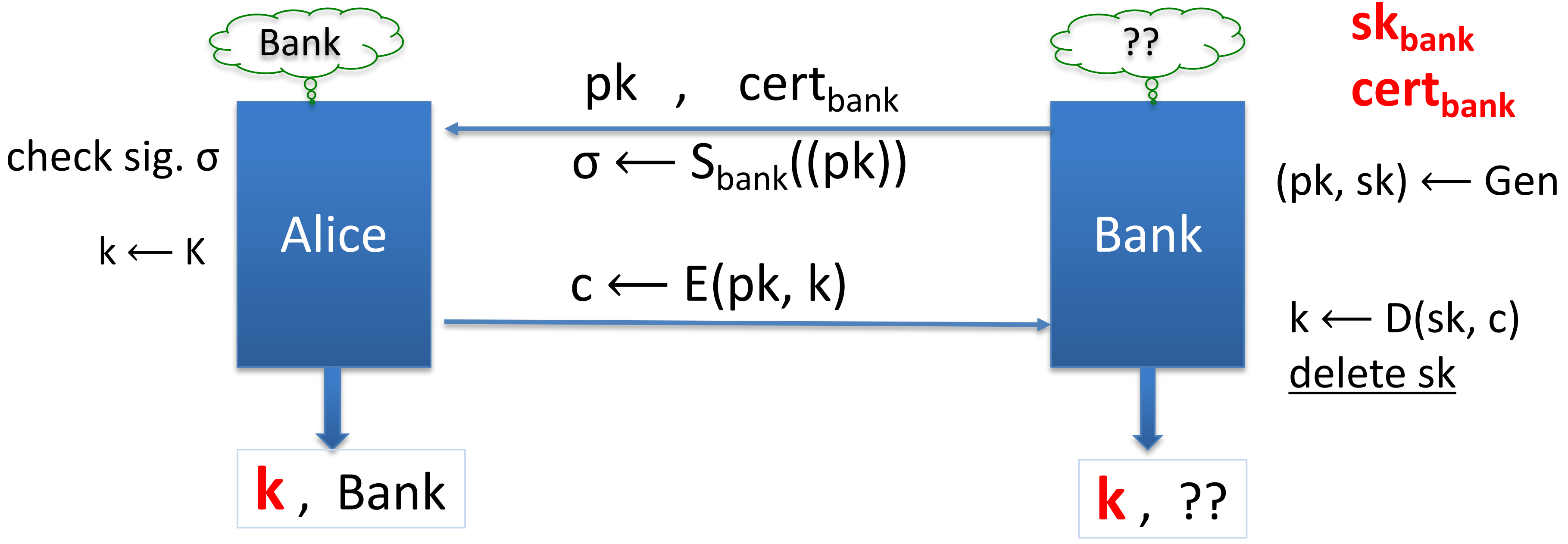
Same attack on the two-sided AKE

This protocol is used in TLS 1.2, deprecated in TLS 1.3

# Protocol #2:  forward secrecy

Server compromise at time T should not compromise sessions at time t<T

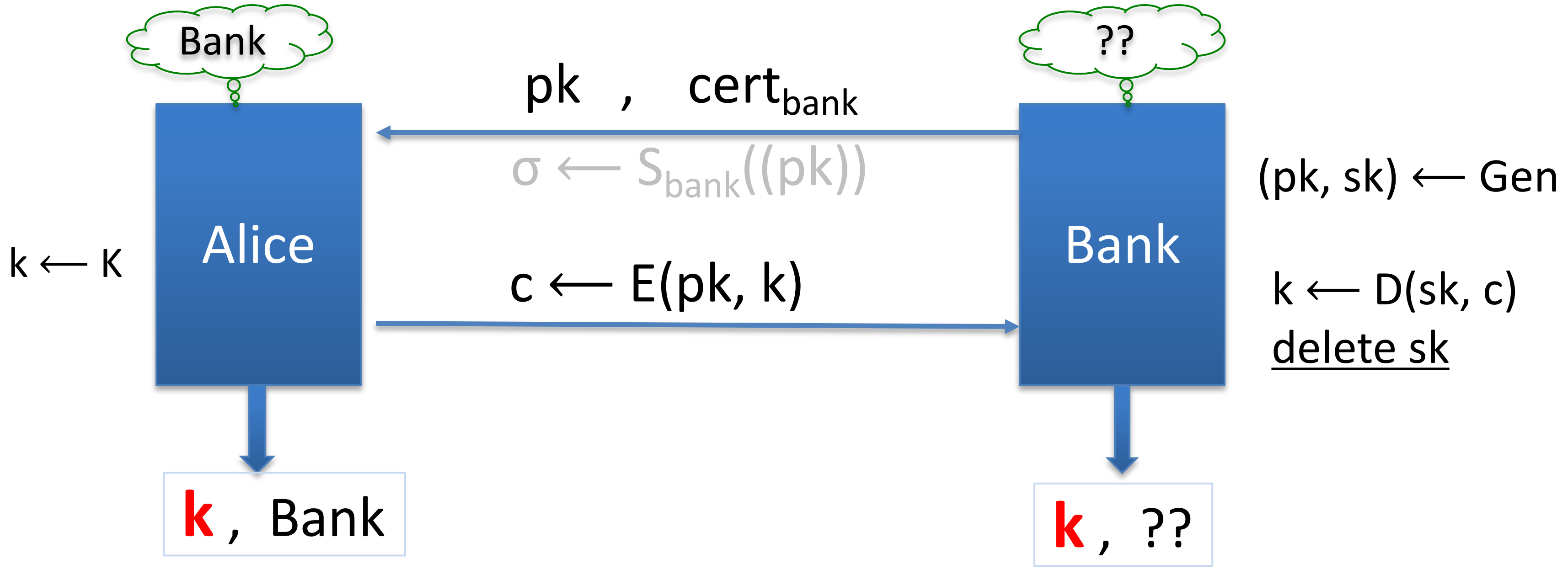
# Simple one-sided AKE with forward-secrecy




(pk, sk) are ephemeral: sk is deleted when protocol completes

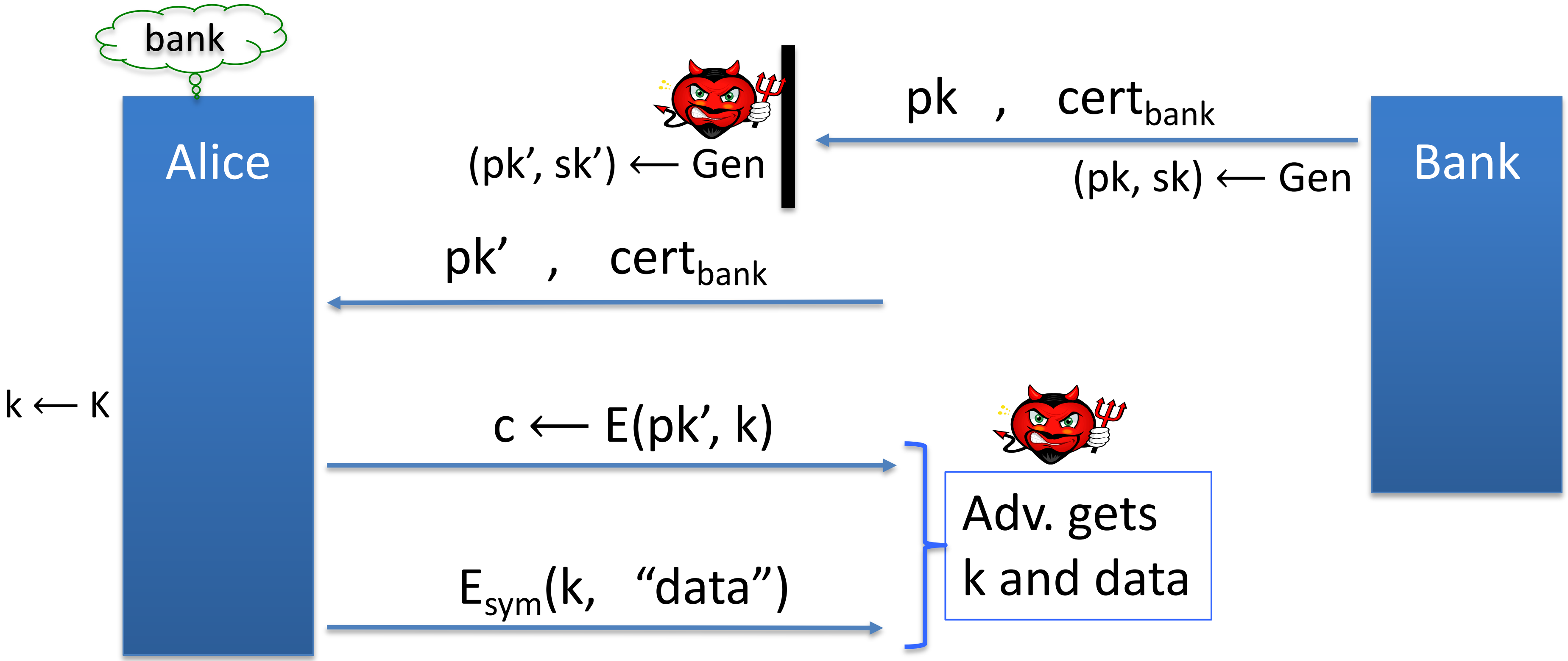
Compromise of Bank: past sessions are unaffected

# Insecure variant:  do not sign pk

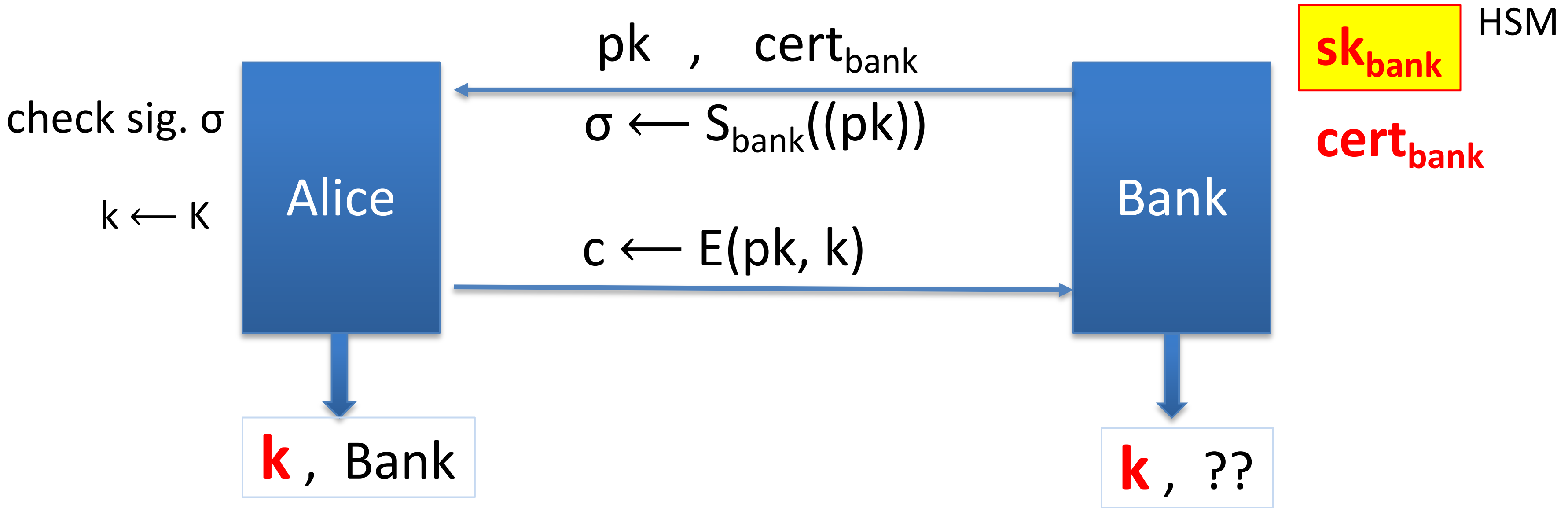



Attack:   complete key exposure

# Attack: key exposure



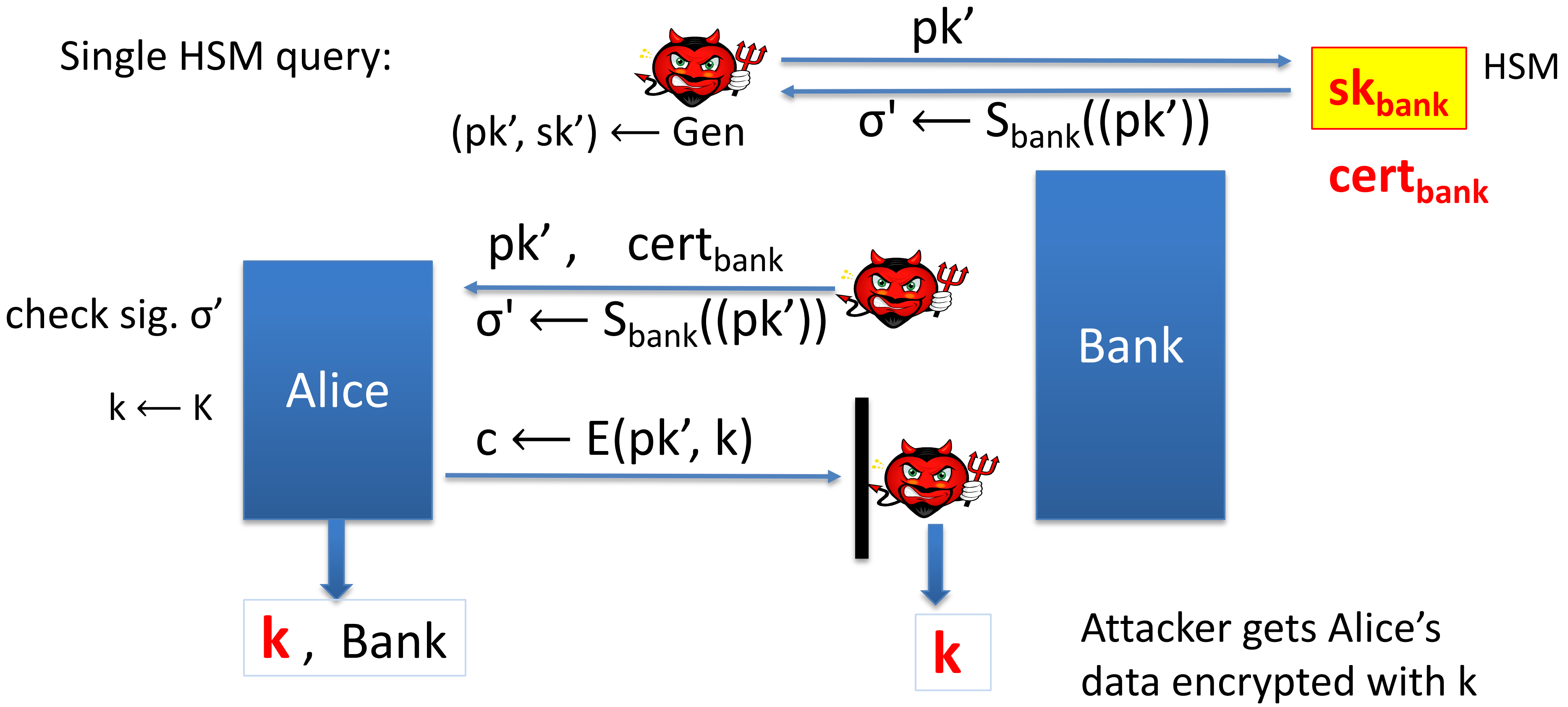
# Problem: not HSM secure




Suppose attacker breaks into Bank and queries HSM once

⇒ complete key exposure forever !

# Problem: not HSM secure

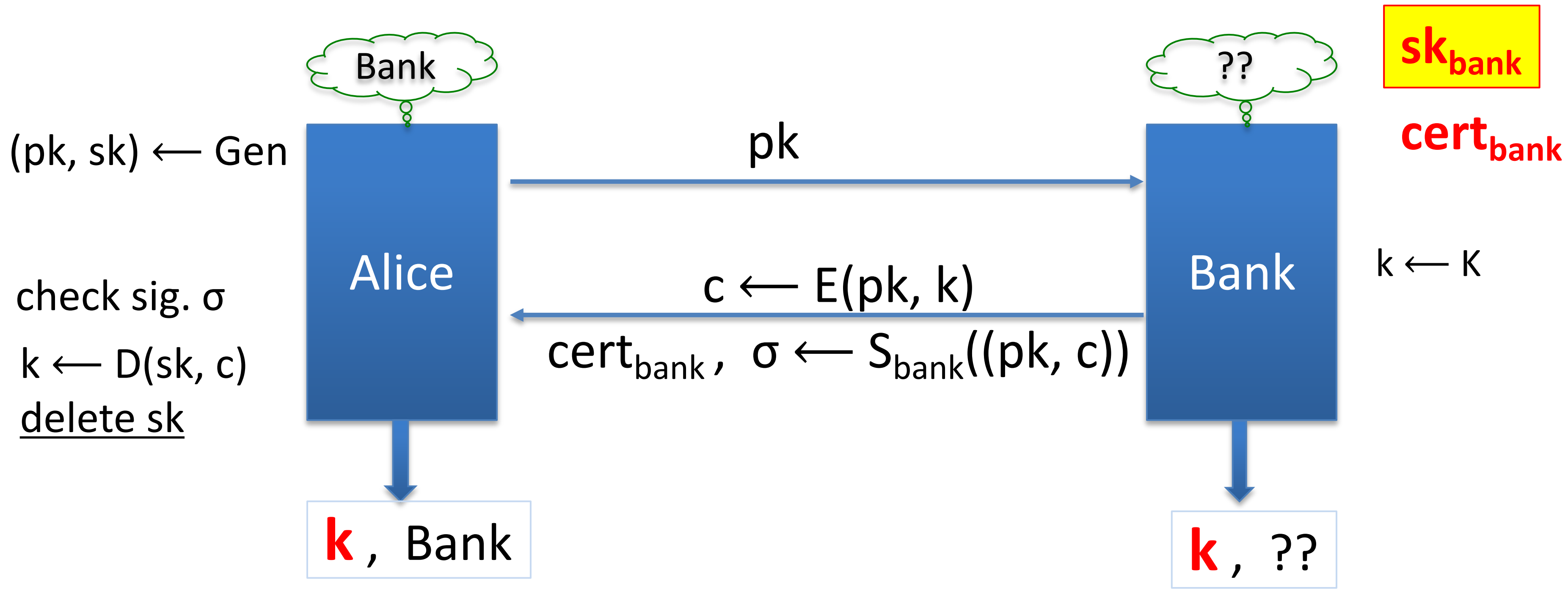


# Protocol #3:  HSM Security

Forward secrecy, and

n queries to HSM should compromise at most n sessions

# Simple HSM security (one-sided)



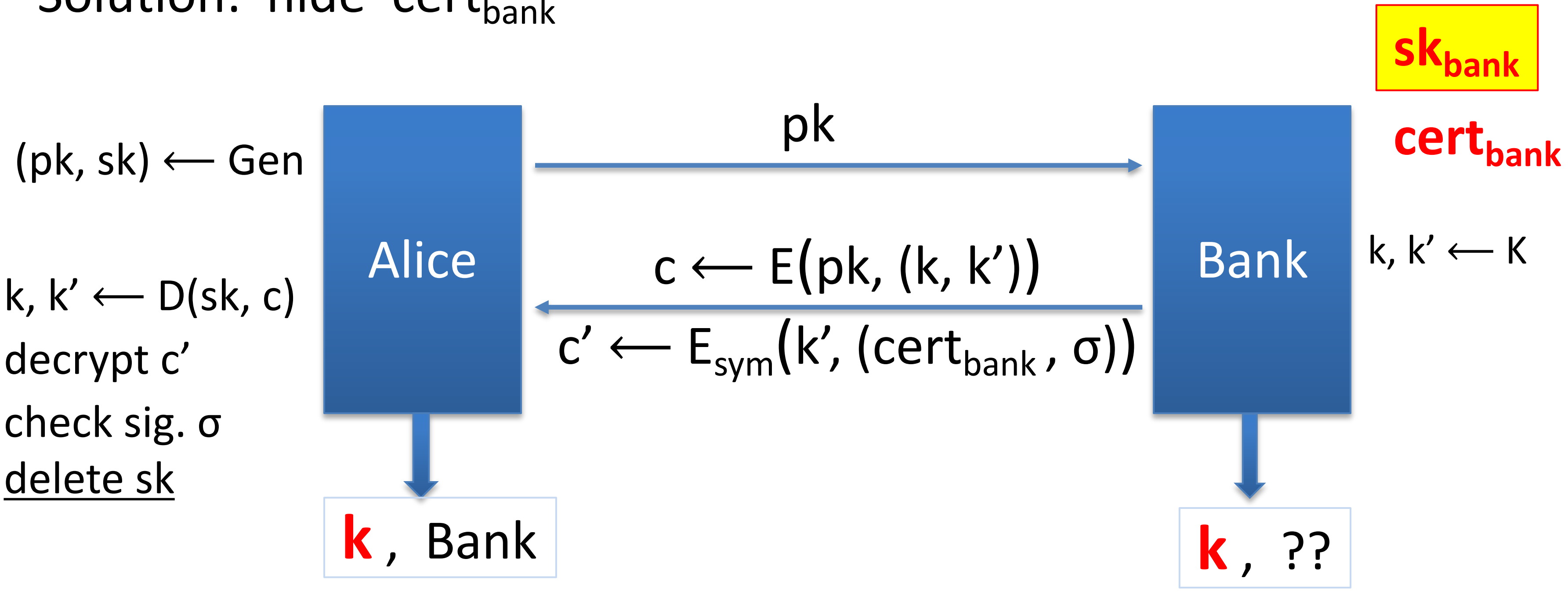

Main point: HSM needed to sign ephemeral pk from client

⇒ past access to HSM will not compromise current session

# Final variant: end-point privacy

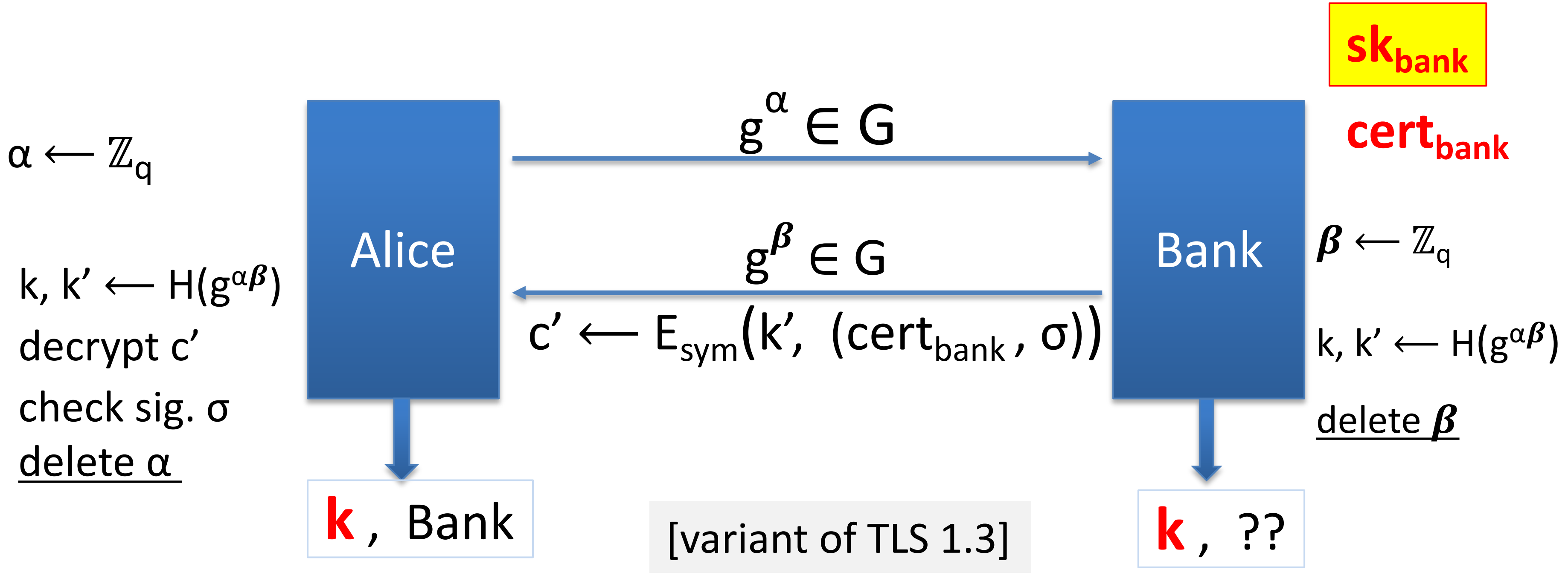
Protocol #3: eavesdropper learns that Alice wants to talk to Bank.

Solution: hide $cert_{bank}$



# Using Diffie-Hellman: DHAKE (simplified)

We can use Diffie-Hellman instead of general public-key encryption



# Many more AKE variants

AKE based on a pre-shared secret between Alice and Bank:

- High entropy pre-shared secret: ensure forward secrecy
- Password: ensure no offline dictionary attack (PAKE)

Deniable:

- Both sides can claim they did not participate in protocol
- In particular, parties do not sign public messages


# Auth. key exchange

# TLS 1.3 Session Setup

RFC 8446 (Aug. 2018)

# TLS 1.3 Session Setup

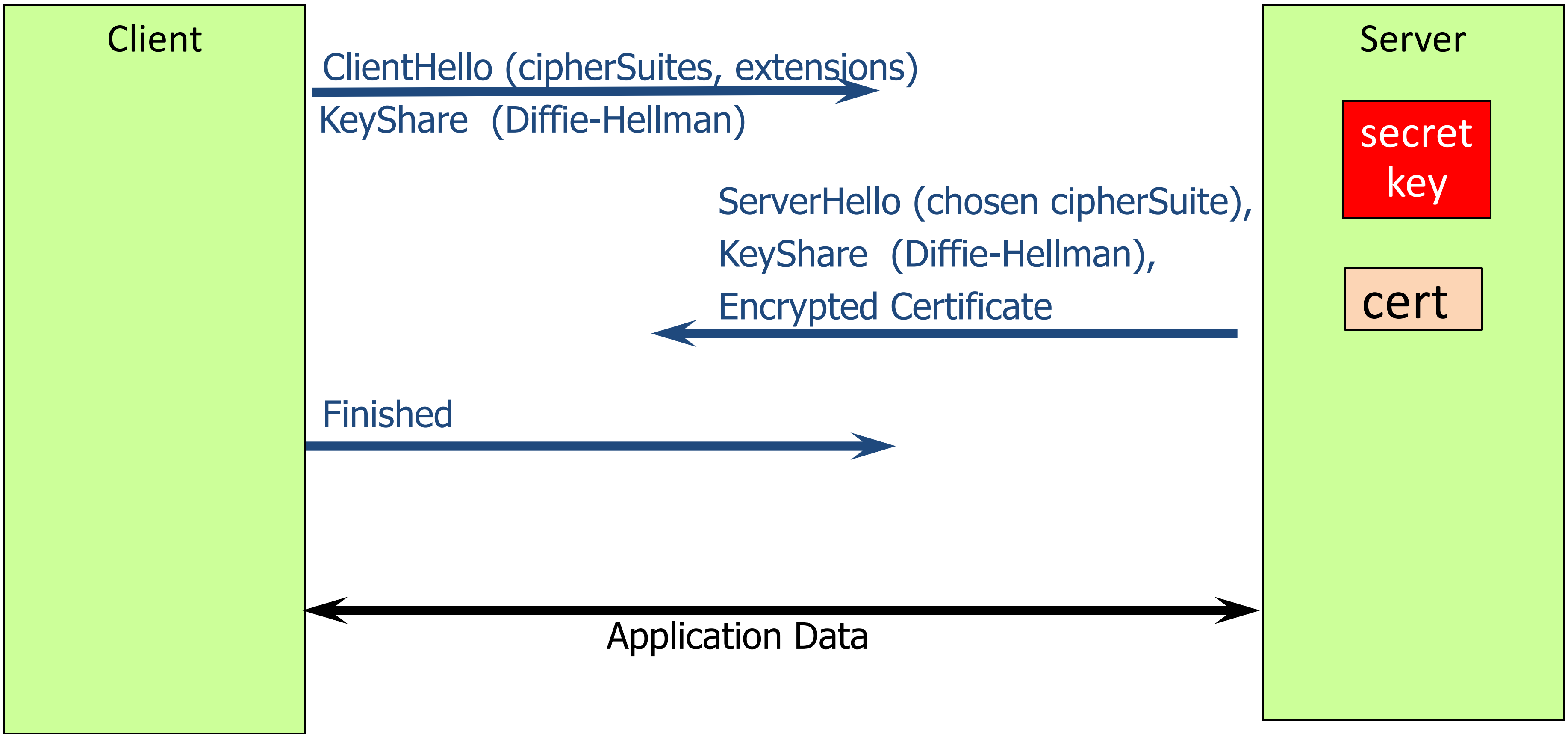
Generate unidirectional keys: $k_{b\longrightarrow s}$ and $k_{s\longrightarrow b}$

Security goals:

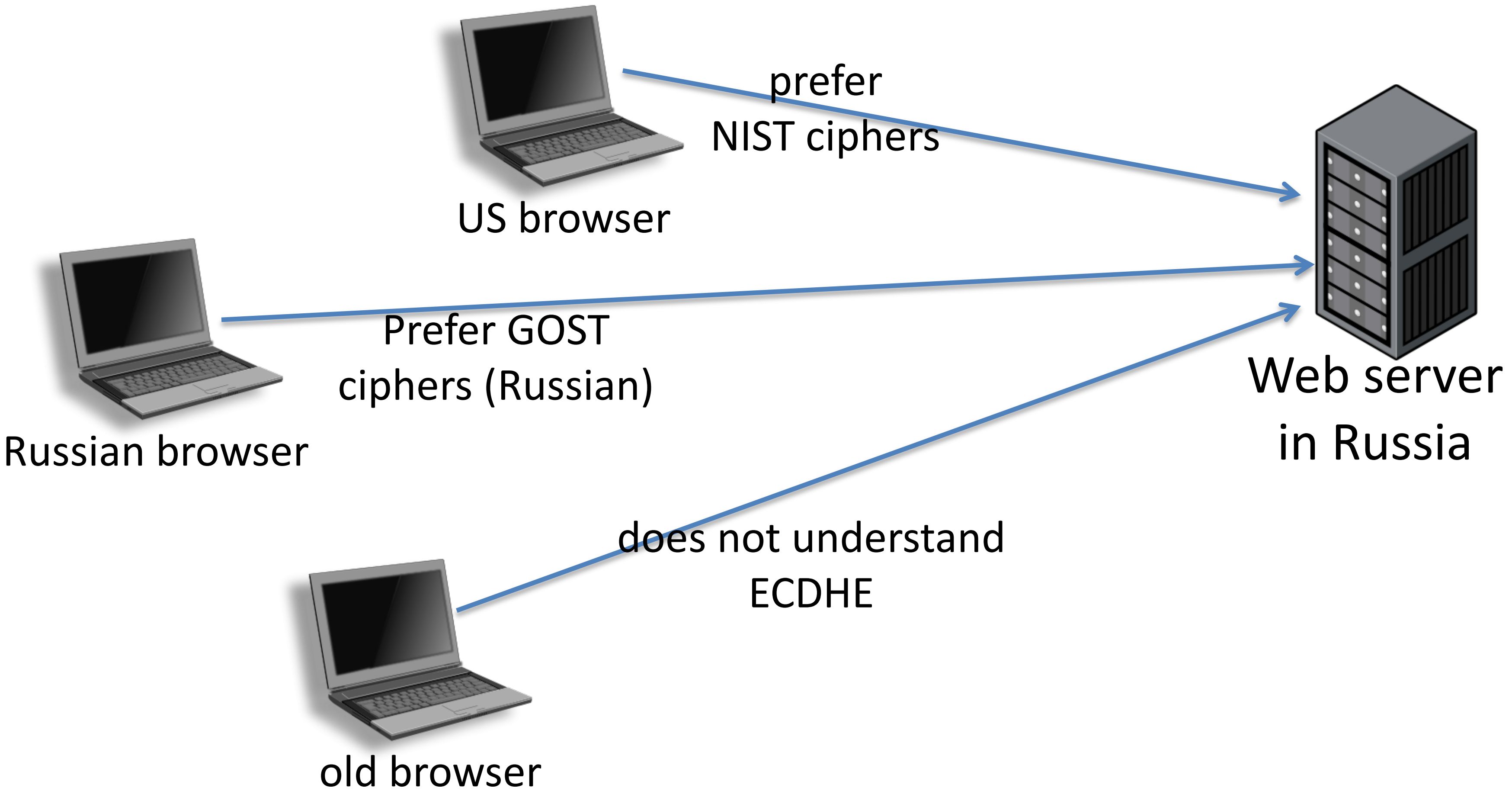
- Support for one-sided and two-sided AKE
- HSM security (including forward secrecy and static security)
- End-point privacy against an eavesdropper

Protocol is related to the Diffie-Hellman protocol DHAKE above

# TLS 1.3 session setup (full handshake, simplified)

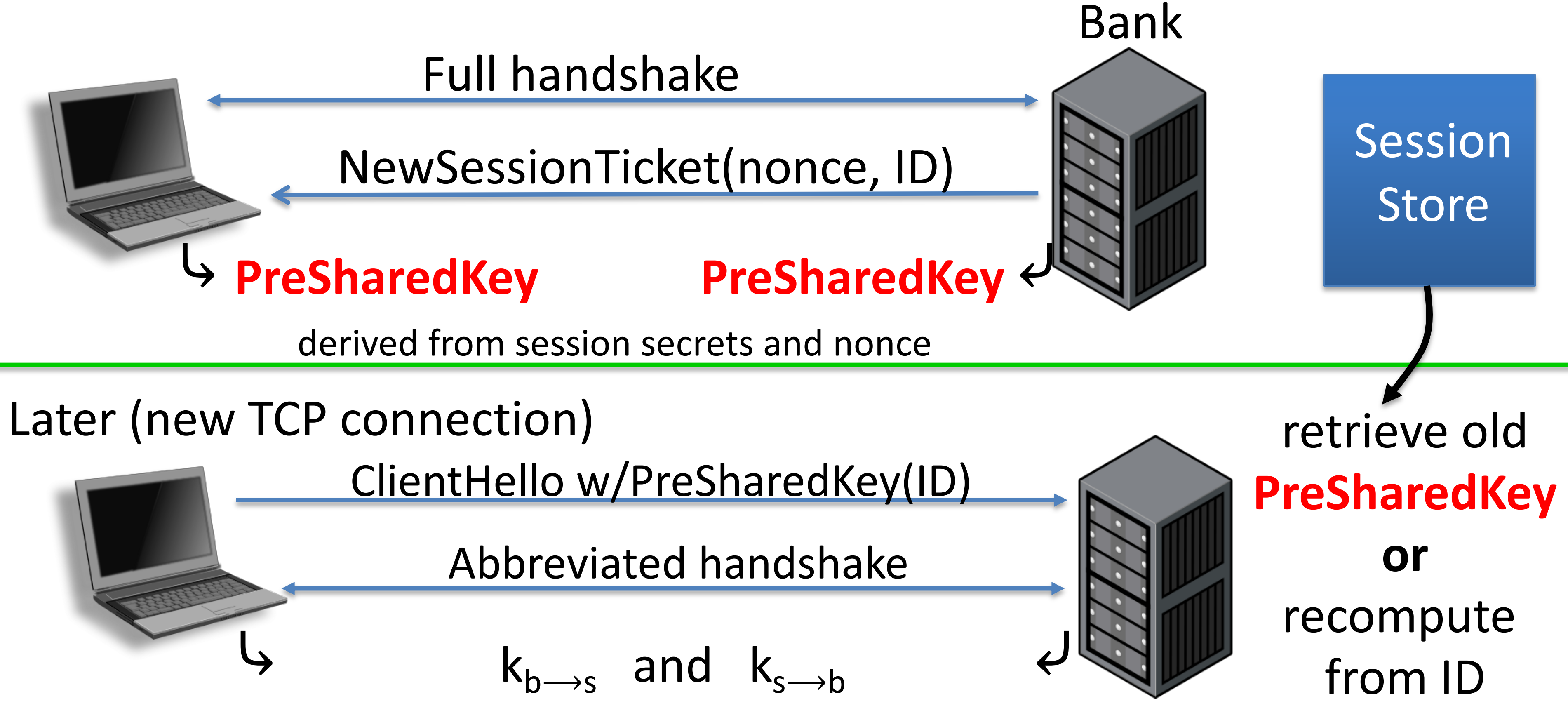


# The need for negotiating ciphers

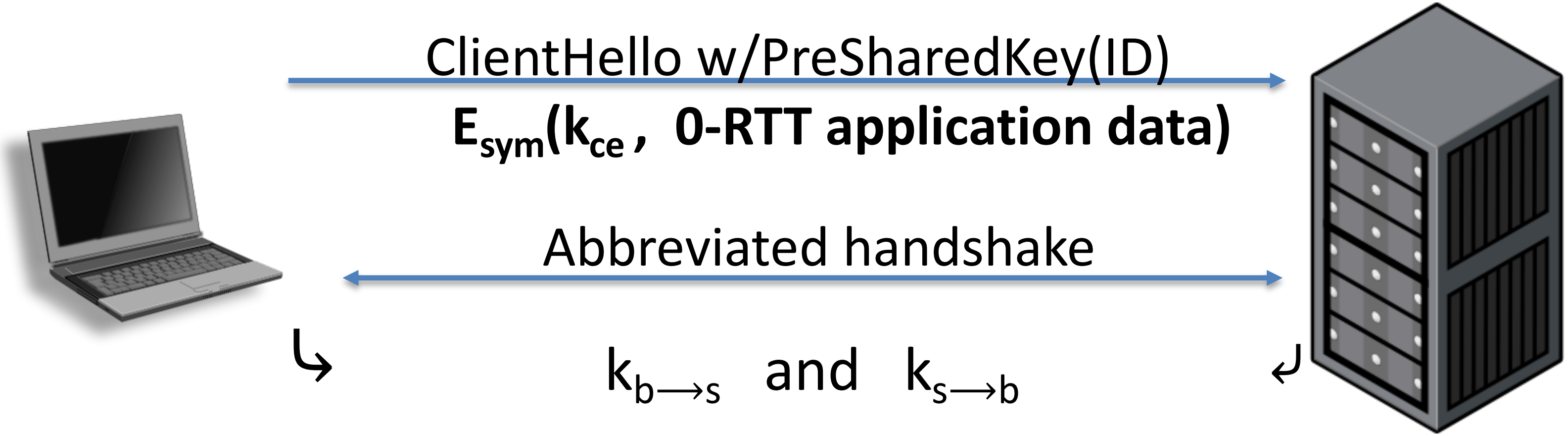


# Session setup from pre-shared keys

Goal: reduce # of full handshakes



# PSK 0-RTT




$k_{CE}$ : client early key-exchange key.
derived from PSK (and other ClientHello data)

Problem: 0-RTT app data is vulnerable to replay.

# THE END