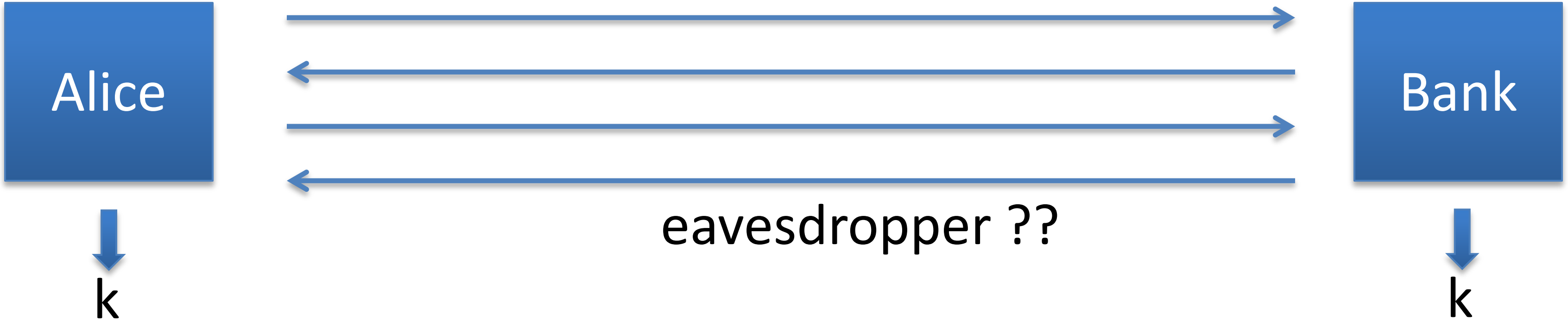
Auth. Key Exchange

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

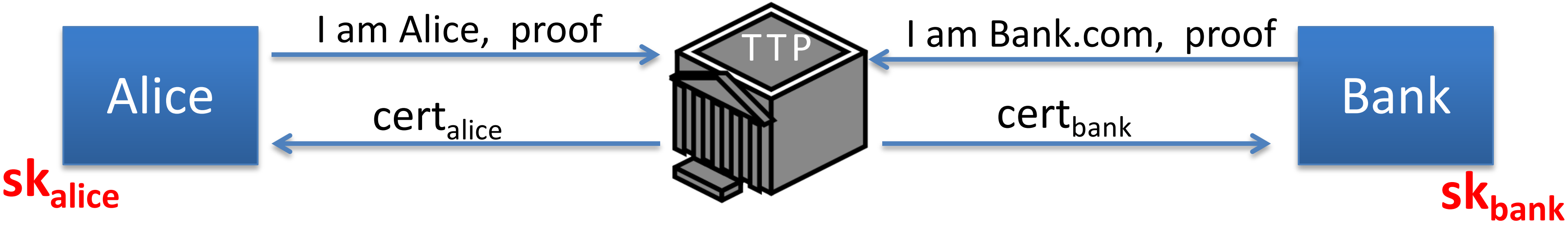
Moreover, some users are honest and others are corrupt

- 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:



Alice

I am Alice, proof

$\text{cert}_{\text{alice}}$

sk_{alice}

TTP

I am Bank.com, proof

$\text{cert}_{\text{bank}}$

Bank

sk_{bank}

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

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

  Benefit: security using only symmetric crypto

AKE: syntax

Bank

sk_{alice}

$cert_{alice}$

pk_{CA}

Alice

k , Bank or ⊥

Alice

sk_{bank}

$cert_{bank}$

pk_{CA}

Bank

k , Alice or ⊥

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 $\mathbf{sk}_{\mathbf{bank}}$ at time T then all sessions with Bank from time t<T remain secret.
- **HSM security**: if adv. queries an HSM holding $\mathbf{sk}_{\mathbf{bank}}$ 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

$\mathbf{cert_{bank}}$: contains pk_{bank} . Bank has **$\mathbf{sk_{bank}}$** .

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

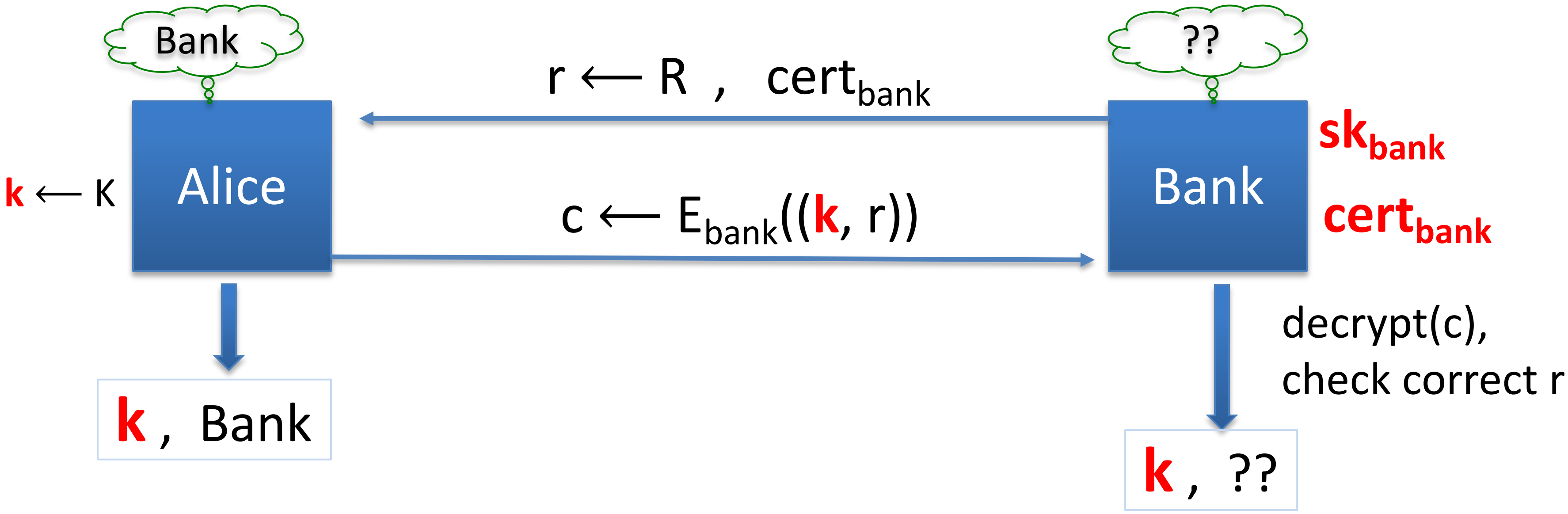
- Recall: from $E_{bank}((m,r))$ adv. cannot build $E_{bank}((m,r'))$ for $r' \neq r$

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

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

Protocol #1

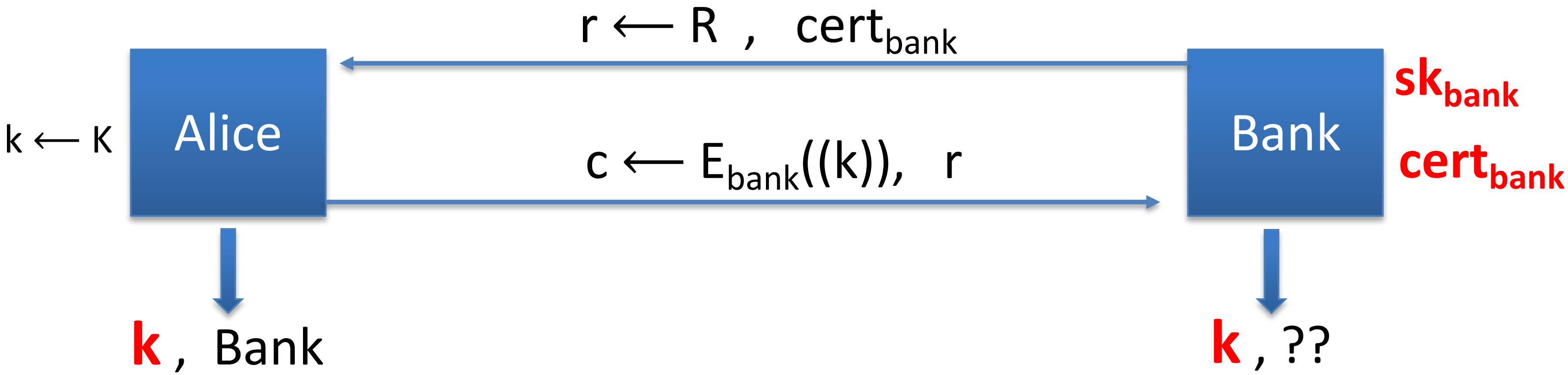
Simple one-sided AKE protocol



"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



$r \leftarrow R$, $\text{cert}_{\text{bank}}$

sk_{bank}

$k \leftarrow K$ Alice

$c \leftarrow E_{\text{bank}}((k))$, r

Bank

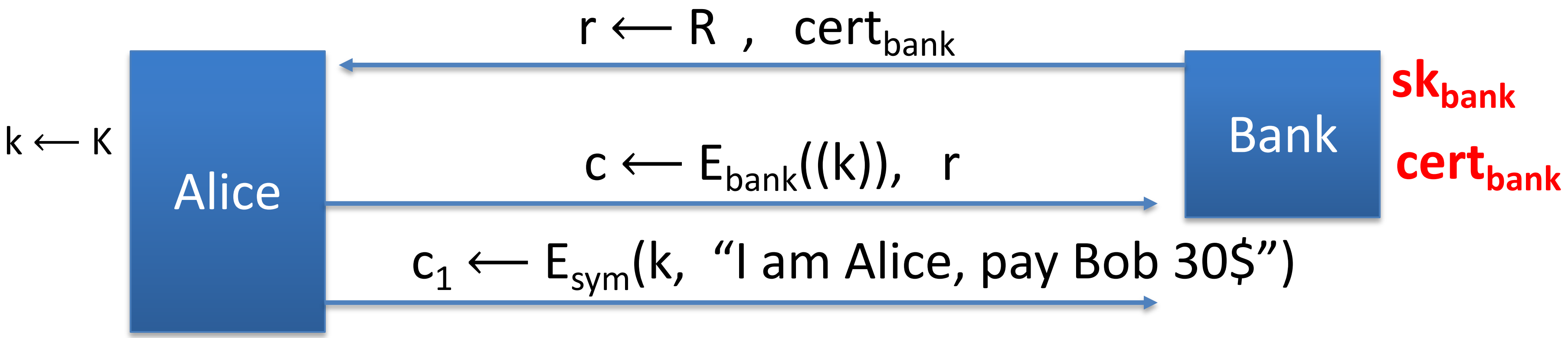
$\text{cert}_{\text{bank}}$

k , Bank

k , ??

Problem: replay attack

Replay attack



$r \leftarrow R$, $\text{cert}_{\text{bank}}$

$k \leftarrow K$

Alice

$c \leftarrow E_{\text{bank}}((k))$, r

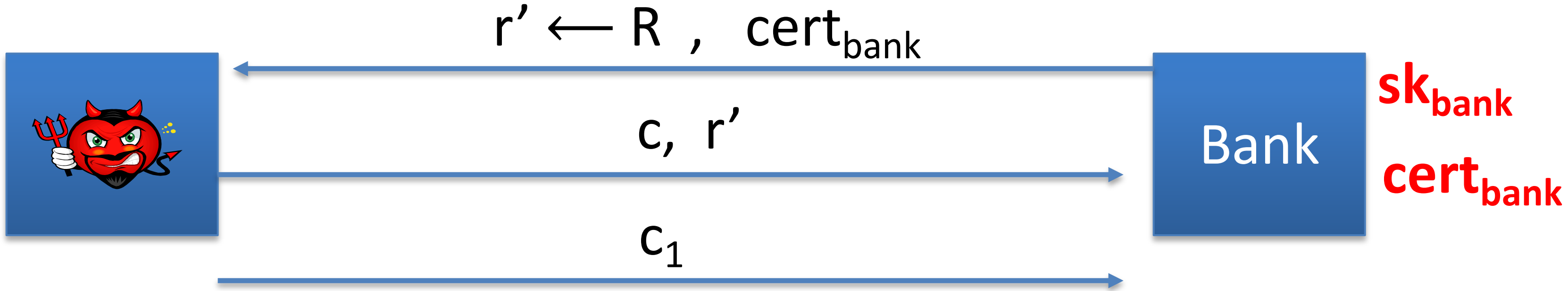
$c_1 \leftarrow E_{\text{sym}}(k, \text{“I am Alice, pay Bob 30\$”})$

Bank

sk_{bank}

$\text{cert}_{\text{bank}}$

Later:



$r' \leftarrow R$, $\text{cert}_{\text{bank}}$

c, r'

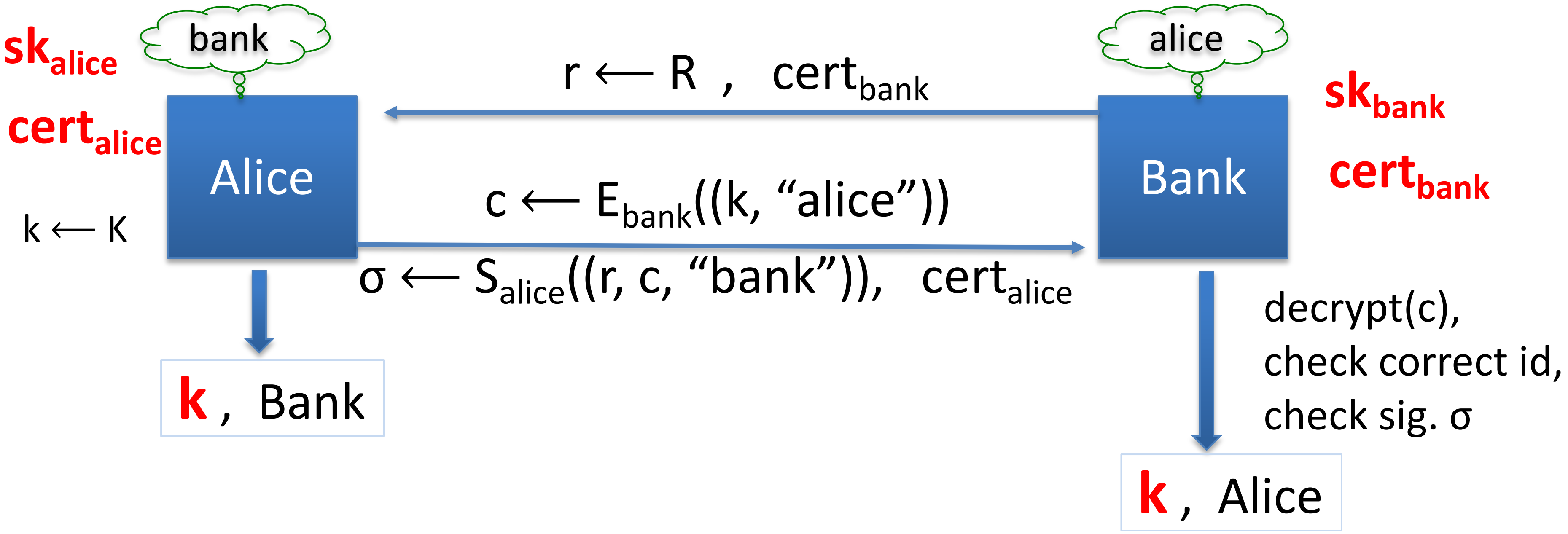
c_1

Bank

sk_{bank}

$\text{cert}_{\text{bank}}$

Two-sided AKE (mutual authentication)



sk_{alice}

$cert_{alice}$

Alice

$k \longleftarrow K$

$r \longleftarrow R$, $cert_{bank}$

$c \longleftarrow E_{bank}((k, \text{“alice”}))$

$\sigma \longleftarrow S_{alice}((r, c, \text{“bank”}))$, $cert_{alice}$

Bank

sk_{bank}

$cert_{bank}$

decrypt(c),
check correct id,
check sig. σ

k , Bank

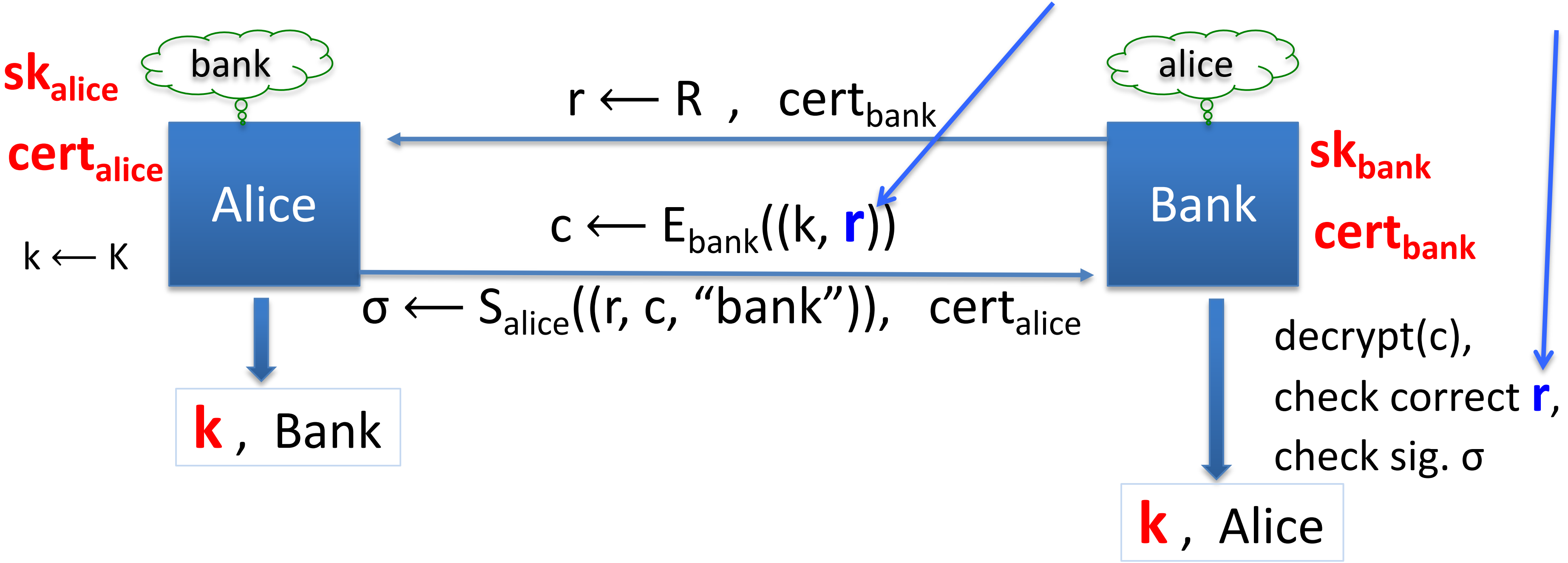
k , Alice

“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:



sk_{alice}

$cert_{alice}$

bank

Alice

$k \longleftarrow K$

$r \longleftarrow R$, $cert_{bank}$

$c \longleftarrow E_{bank}((k, \mathbf{r}))$

$\sigma \longleftarrow S_{alice}((r, c, \text{“bank”}))$, $cert_{alice}$

alice

Bank

sk_{bank}

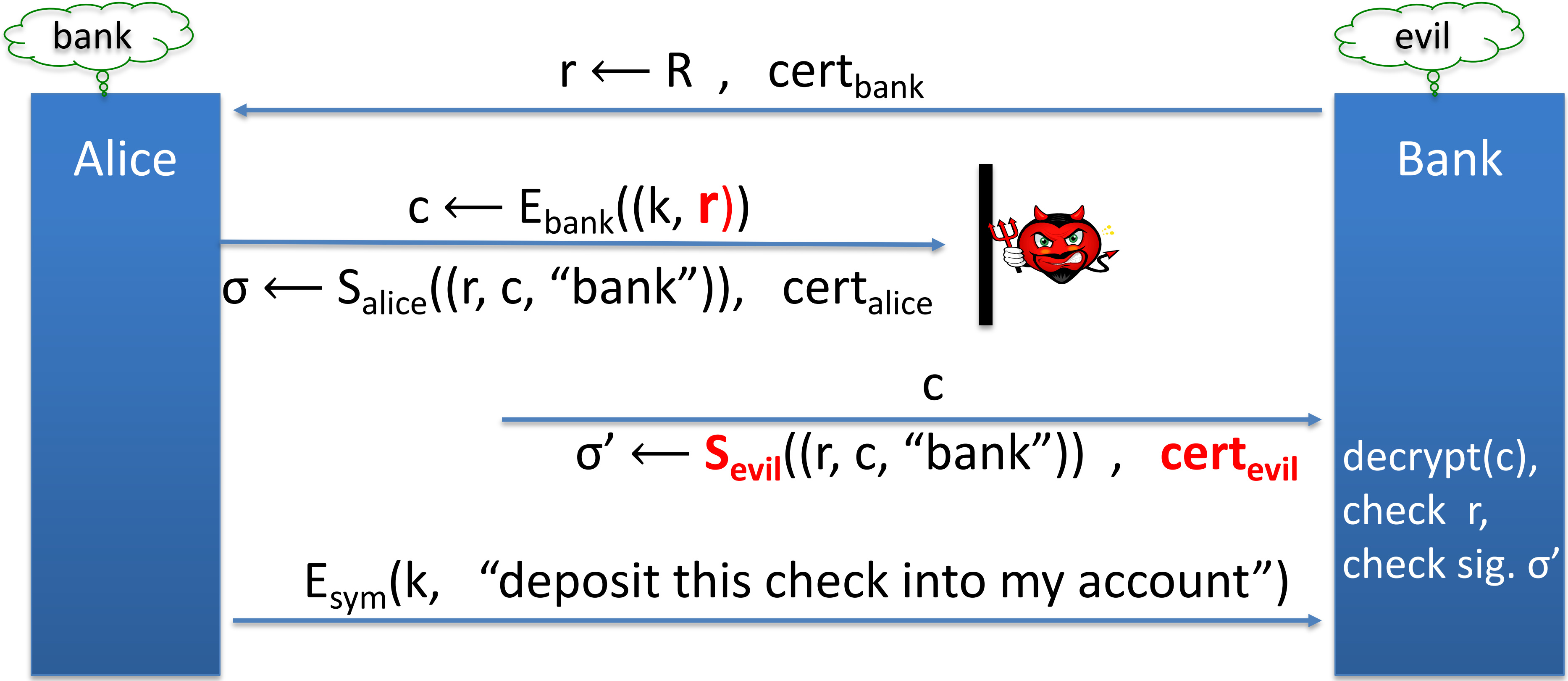
$cert_{bank}$

decrypt(c),
check correct **r**,
check sig. σ

k , Bank

k , Alice

Attack: identity misbinding



bank

evil

Alice

Bank

$r \leftarrow R$, $\text{cert}_{\text{bank}}$

$c \leftarrow E_{\text{bank}}((k, \mathbf{r}))$

$\sigma \leftarrow S_{\text{alice}}((r, c, \text{“bank”}))$, $\text{cert}_{\text{alice}}$

c

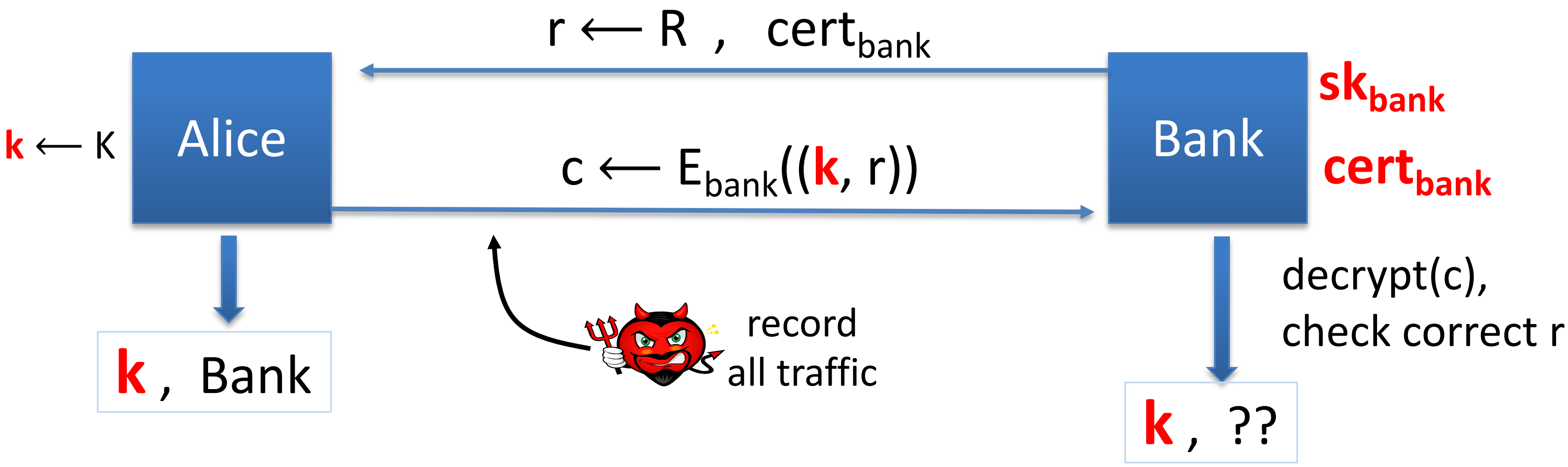
$\sigma' \leftarrow \mathbf{S_{\text{evil}}}((r, c, \text{“bank”}))$, $\mathbf{cert_{\text{evil}}}$

decrypt(c),
check r,
check sig. σ’

$E_{\text{sym}}(k,$ “deposit this check into my account”)

Problem: no forward secrecy

Recall the one-sided AKE:



$r \longleftarrow R$, $\text{cert}_{\text{bank}}$

$\mathbf{k} \longleftarrow K$

Alice

$c \longleftarrow E_{\text{bank}}((\mathbf{k}, r))$

Bank

$\mathbf{sk}_{\mathbf{bank}}$

$\mathbf{cert}_{\mathbf{bank}}$

k , Bank

record all traffic

decrypt(c),
check correct r

k , ??

Suppose a year later adversary obtains $\mathbf{sk}_{\mathbf{bank}}$

⇒ can decrypt all recorded traffic

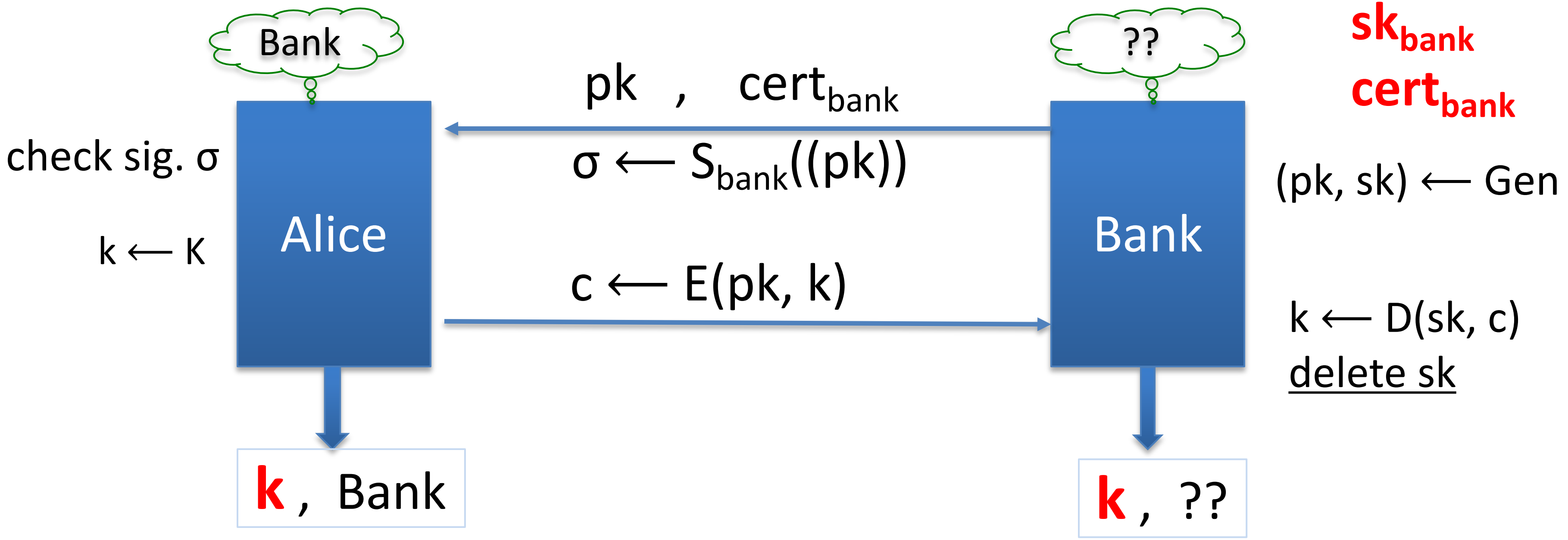
Same attack on the two-sided AKE

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



Bank

??

$\mathbf{sk_{bank}}$

$\mathbf{cert_{bank}}$

pk , $\text{cert}_{\text{bank}}$

$\sigma \longleftarrow S_{\text{bank}}((\text{pk}))$

check sig. σ

$k \longleftarrow K$

Alice

Bank

$(\text{pk}, \text{sk}) \longleftarrow \text{Gen}$

$c \longleftarrow E(\text{pk}, k)$

$k \longleftarrow D(\text{sk}, c)$

<u>delete sk</u>

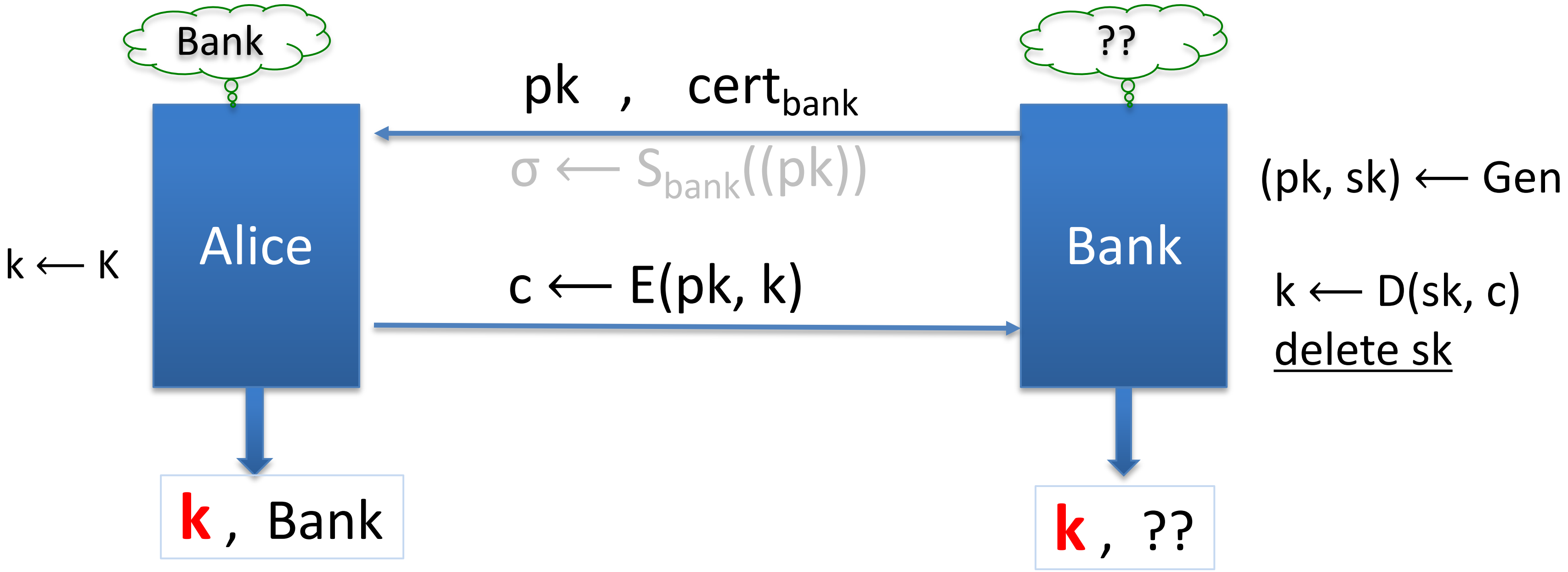
k , Bank

k , ??

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

Compromise of Bank: past sessions are unaffected

Insecure variant: do not sign pk



Bank

??

pk , $\text{cert}_{\text{bank}}$

$\sigma \longleftarrow S_{\text{bank}}((pk))$

$(pk, sk) \longleftarrow \text{Gen}$

$k \longleftarrow K$

Alice

Bank

$c \longleftarrow E(pk, k)$

$k \longleftarrow D(sk, c)$

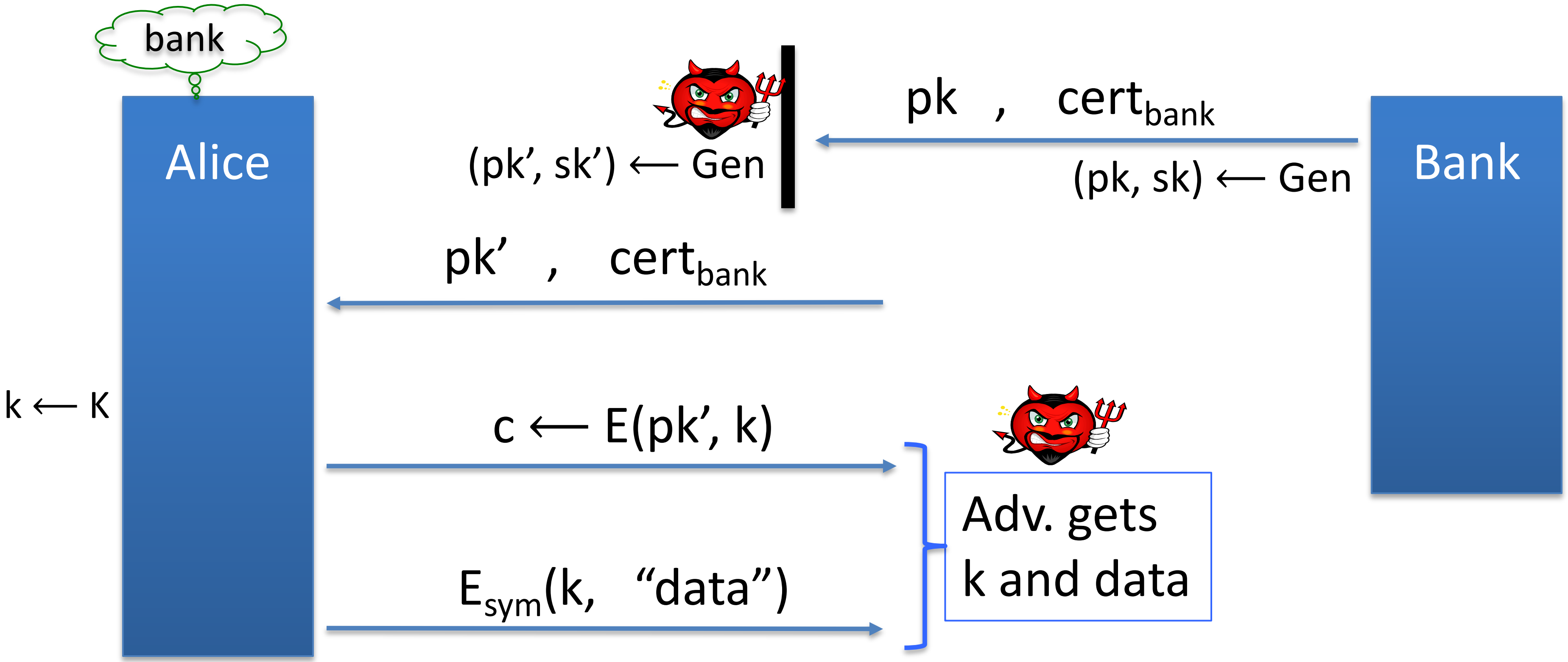
delete sk

k , Bank

k , ??

Attack: complete key exposure

Attack: key exposure



bank

Alice

Bank

$pk \;,\; cert_{bank}$

$(pk, sk) \longleftarrow Gen$

$(pk', sk') \longleftarrow Gen$

$pk' \;,\; cert_{bank}$

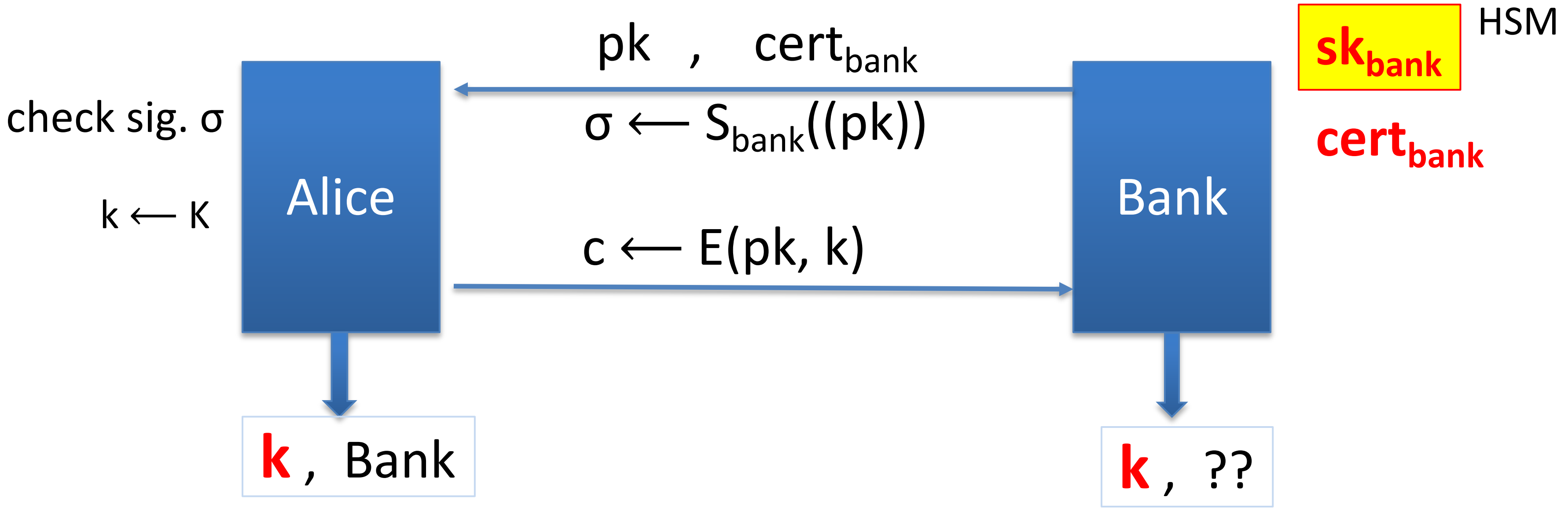
$k \longleftarrow K$

$c \longleftarrow E(pk', k)$

$E_{sym}(k, \text{“data”})$

Adv. gets
k and data

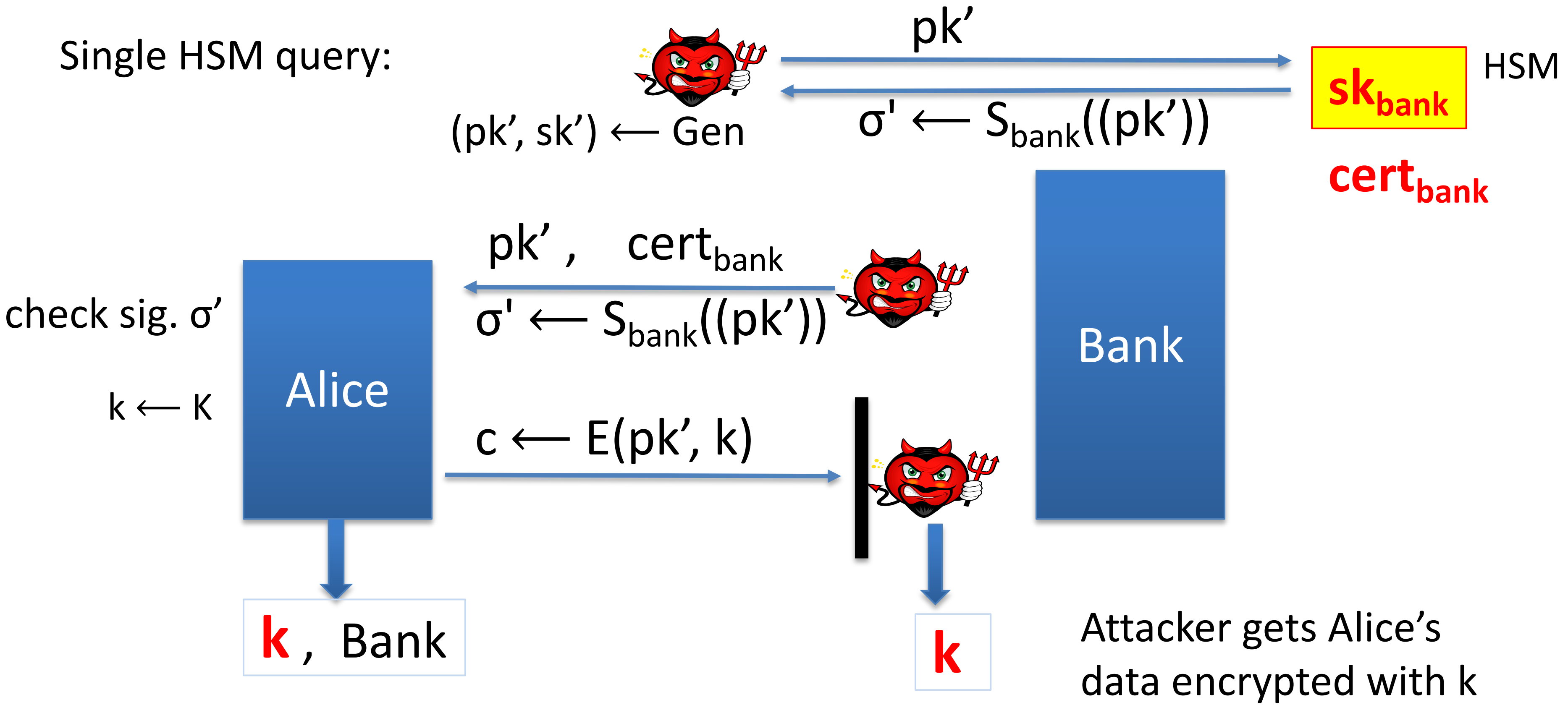
Problem: not HSM secure



Suppose attacker breaks into Bank and queries HSM <u>once</u>

⇒ complete key exposure <u>forever</u> !

Problem: not HSM secure



Single HSM query:

pk'

$(pk', sk') \longleftarrow Gen$

$\sigma' \longleftarrow S_{bank}((pk'))$

sk_{bank} HSM

$cert_{bank}$

check sig. σ'

$k \longleftarrow K$

Alice

pk' , $cert_{bank}$

$\sigma' \longleftarrow S_{bank}((pk'))$

$c \longleftarrow E(pk', k)$

Bank

k , Bank

k

Attacker gets Alice’s data encrypted with k

Protocol #3: HSM Security

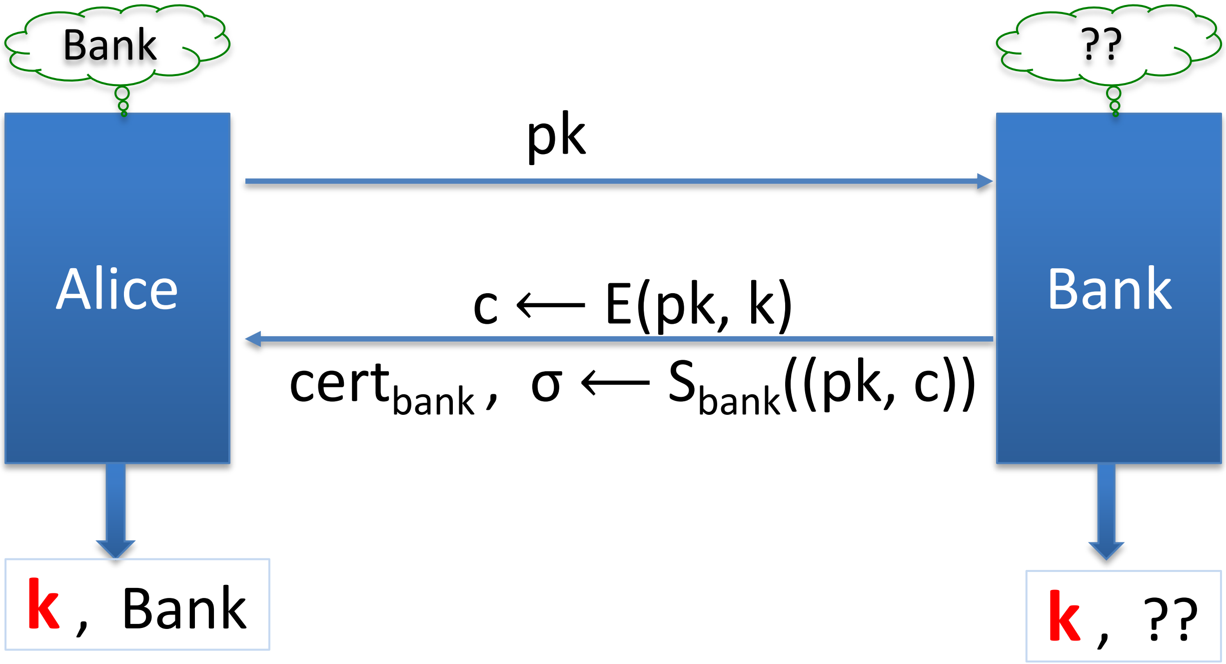
Forward secrecy, and

n queries to HSM should compromise at most n sessions

Simple HSM security (one-sided)

$\boxed{sk_{bank}}$

$cert_{bank}$



$(pk, sk) \longleftarrow Gen$

pk

$k \longleftarrow K$

$c \longleftarrow E(pk, k)$

$cert_{bank}$, $\sigma \longleftarrow S_{bank}((pk, c))$

check sig. σ

$k \longleftarrow D(sk, c)$

delete sk

k , Bank

k , ??

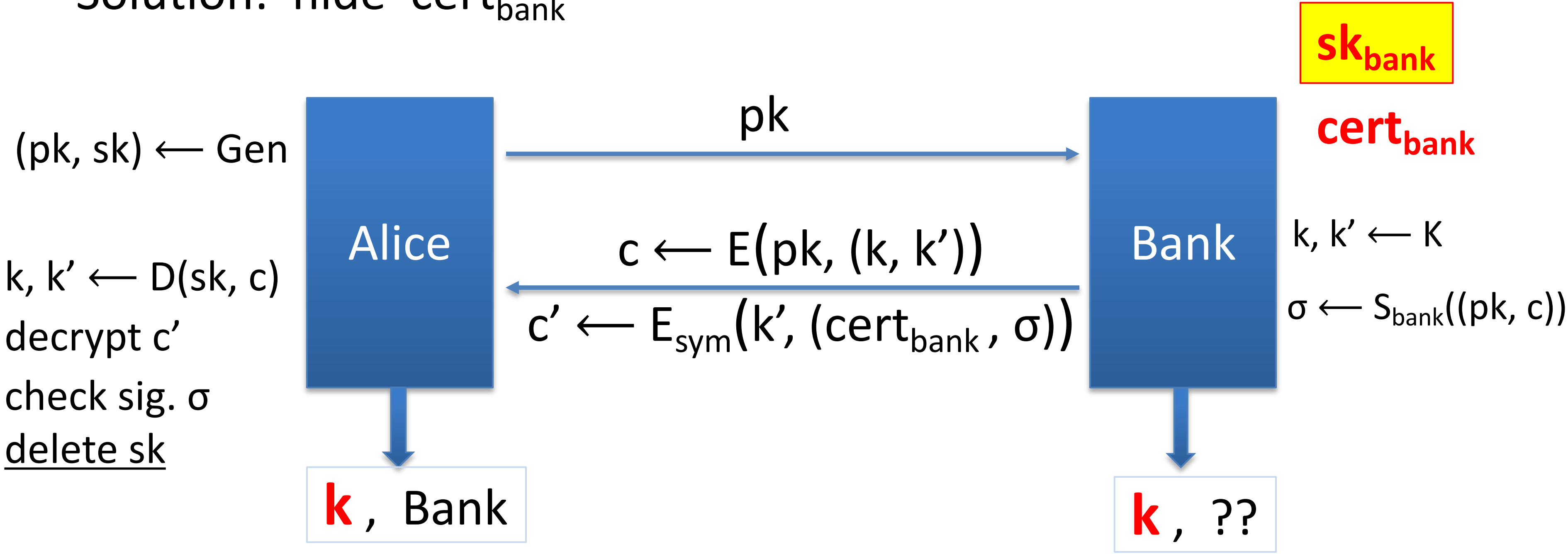
Main point: HSM needed to sign ephemeral pk from client

⇒ 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 $\text{cert}_{\text{bank}}$



sk_{bank}

$\text{cert}_{\text{bank}}$

$(pk, sk) \longleftarrow \text{Gen}$

Alice

pk →

← $c \longleftarrow E(pk, (k, k'))$

$c' \longleftarrow E_{sym}(k', (\text{cert}_{\text{bank}}, \sigma))$

Bank

$k, k' \longleftarrow K$

$\sigma \longleftarrow S_{\text{bank}}((pk, c))$

$k, k' \longleftarrow D(sk, c)$

decrypt c’

check sig. σ

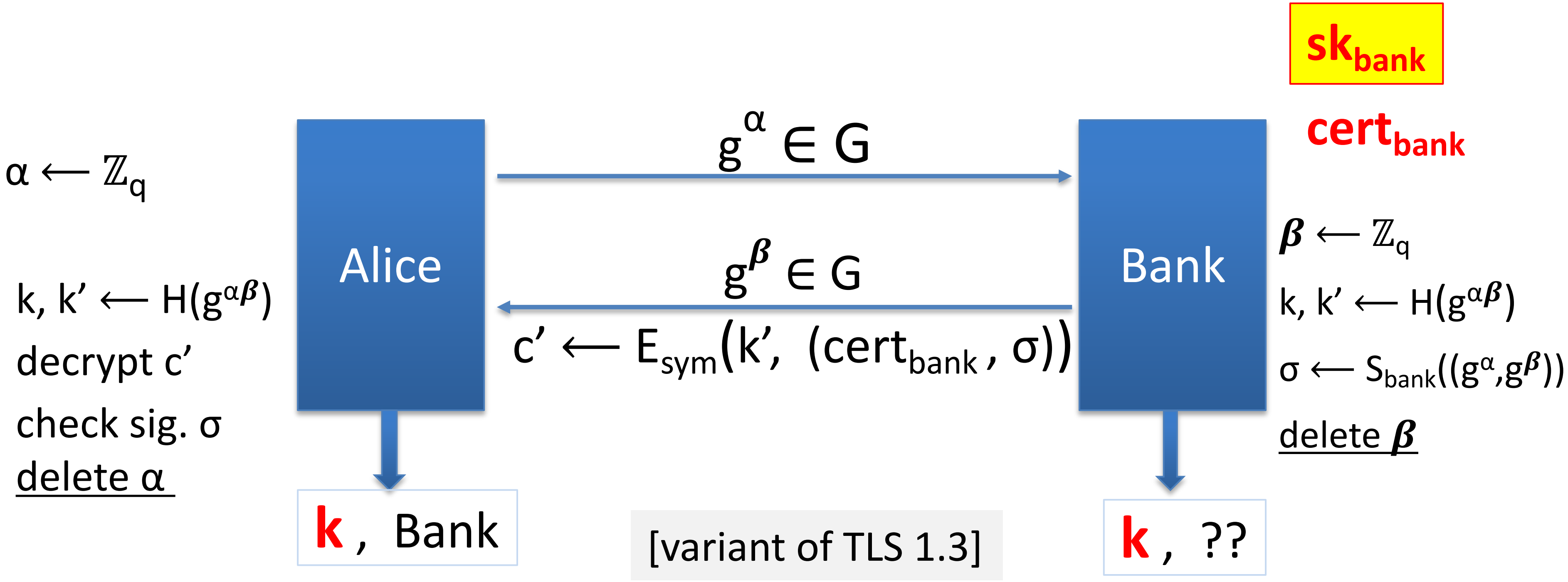
delete sk

k , Bank

k , ??

Using Diffie-Hellman: DHAKE (simplified)

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



sk_{bank}

$cert_{bank}$

$\alpha \leftarrow \mathbb{Z}_q$

Alice

$g^{\alpha} \in G$

Bank

$\beta \leftarrow \mathbb{Z}_q$

$g^{\beta} \in G$

$c' \leftarrow E_{sym}(k', \ (cert_{bank}, \sigma))$

$k, k' \leftarrow H(g^{\alpha\beta})$

decrypt c'

check sig. σ

delete α

$k, k' \leftarrow H(g^{\alpha\beta})$

$\sigma \leftarrow S_{bank}((g^{\alpha}, g^{\beta}))$

delete β

k , Bank

[variant of TLS 1.3]

k , ??

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

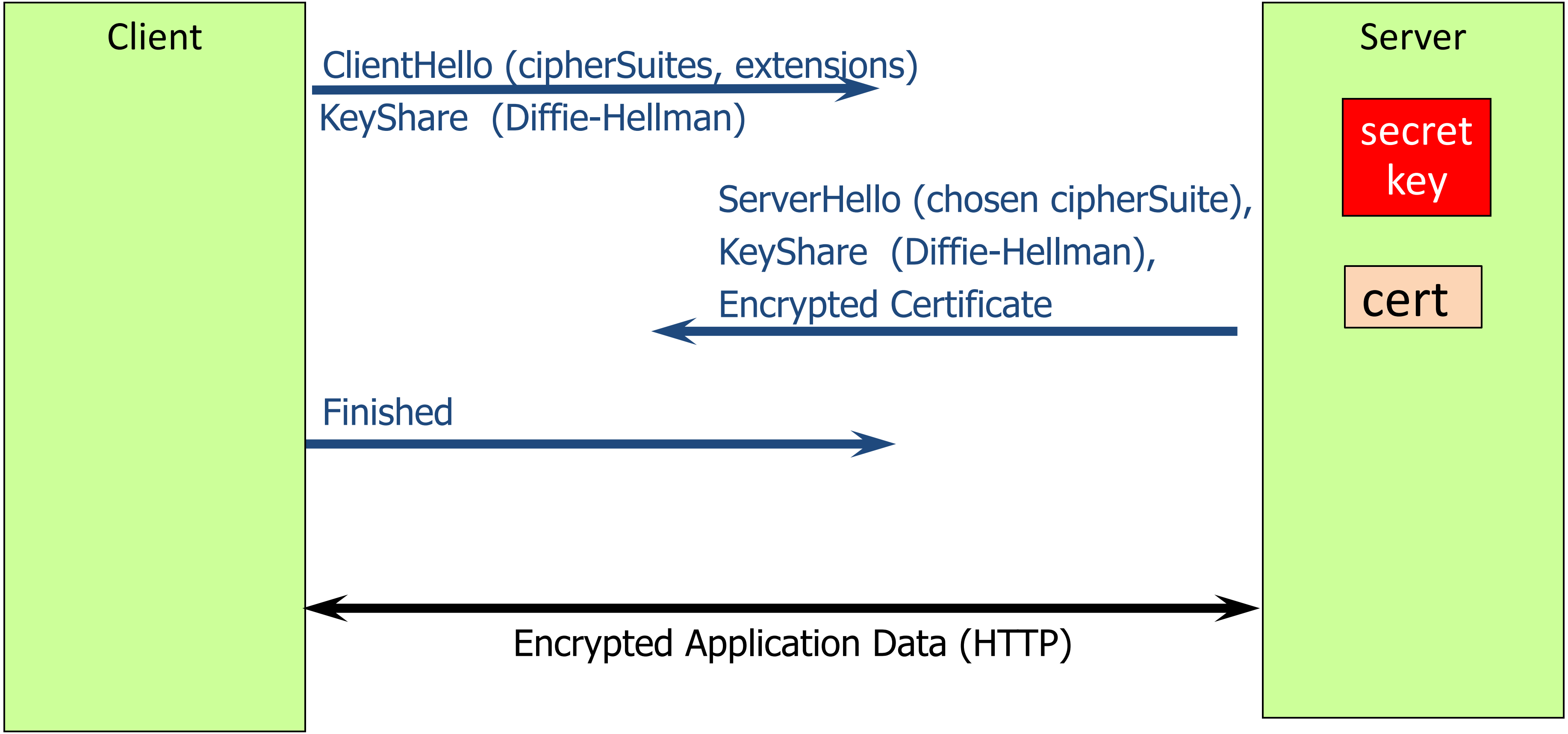
Generate unidirectional keys: $k_{b\longrightarrow s}$ and $k_{s\longrightarrow b}$

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)



Client

Server

secret key

cert

ClientHello (cipherSuites, extensions)

KeyShare (Diffie-Hellman)

ServerHello (chosen cipherSuite),

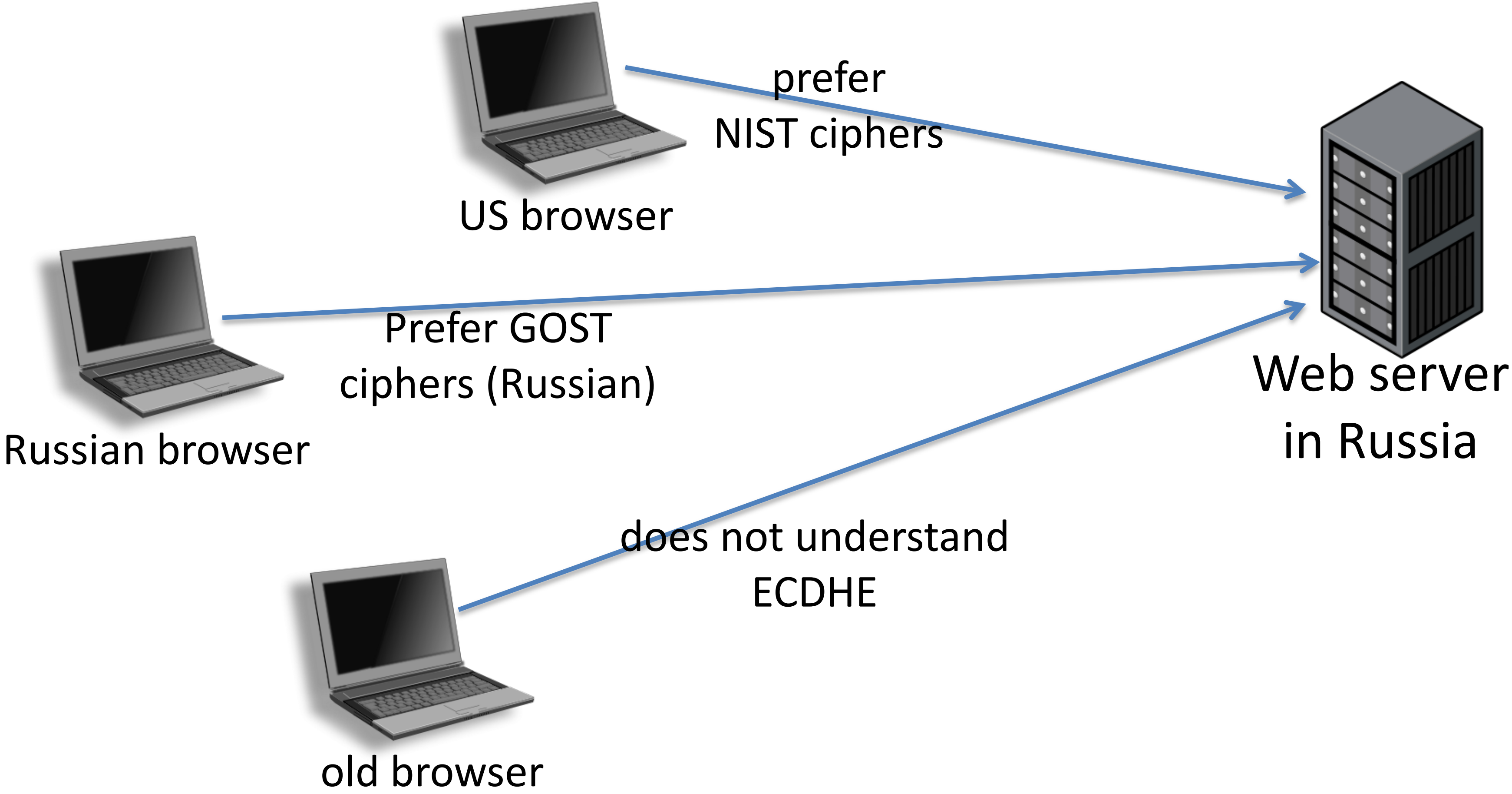
KeyShare (Diffie-Hellman),

Encrypted Certificate

Finished

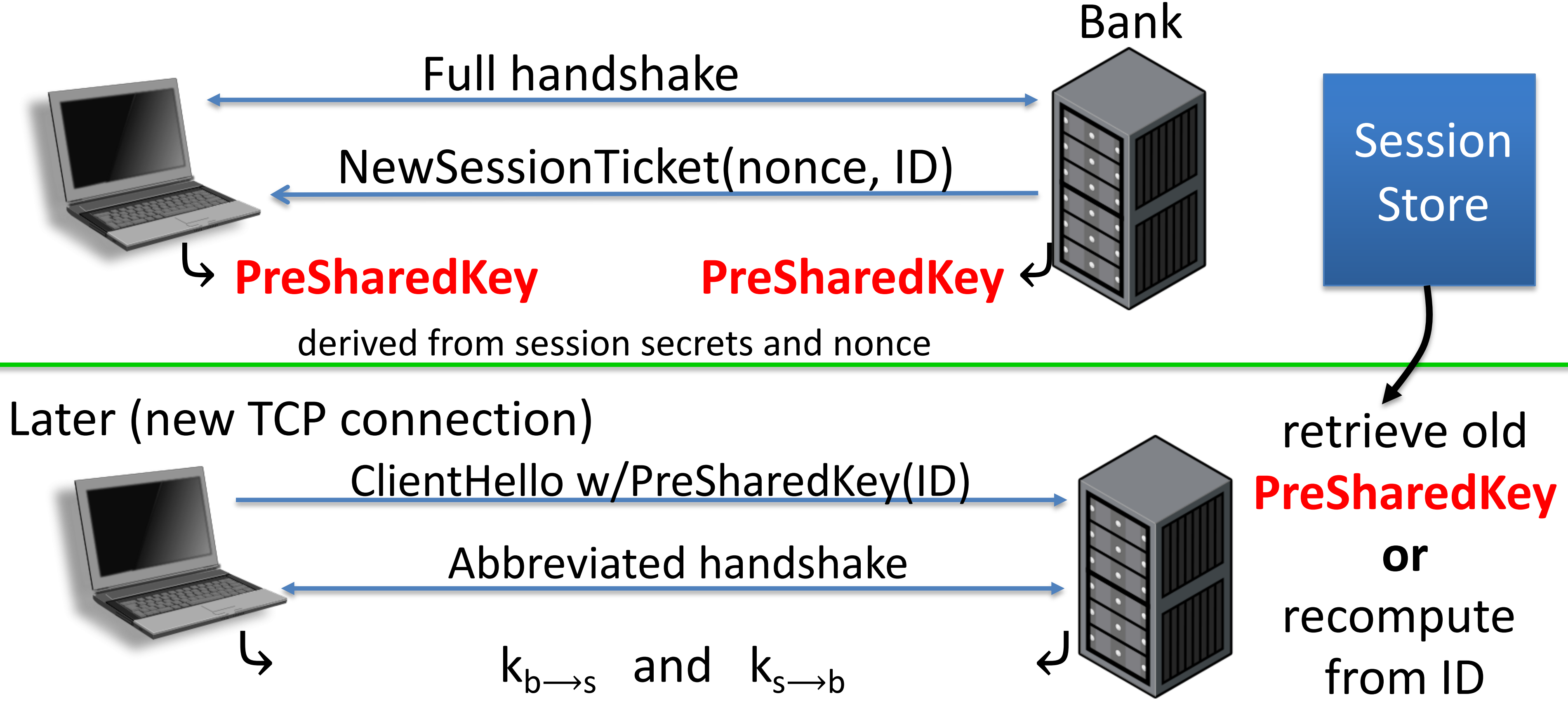
Encrypted Application Data (HTTP)

The need for negotiating ciphers



Session setup from pre-shared keys

Goal: reduce # of full handshakes



Bank

Full handshake

NewSessionTicket(nonce, ID)

↳ **PreSharedKey** **PreSharedKey** ↲

derived from session secrets and nonce

Session Store

Later (new TCP connection)

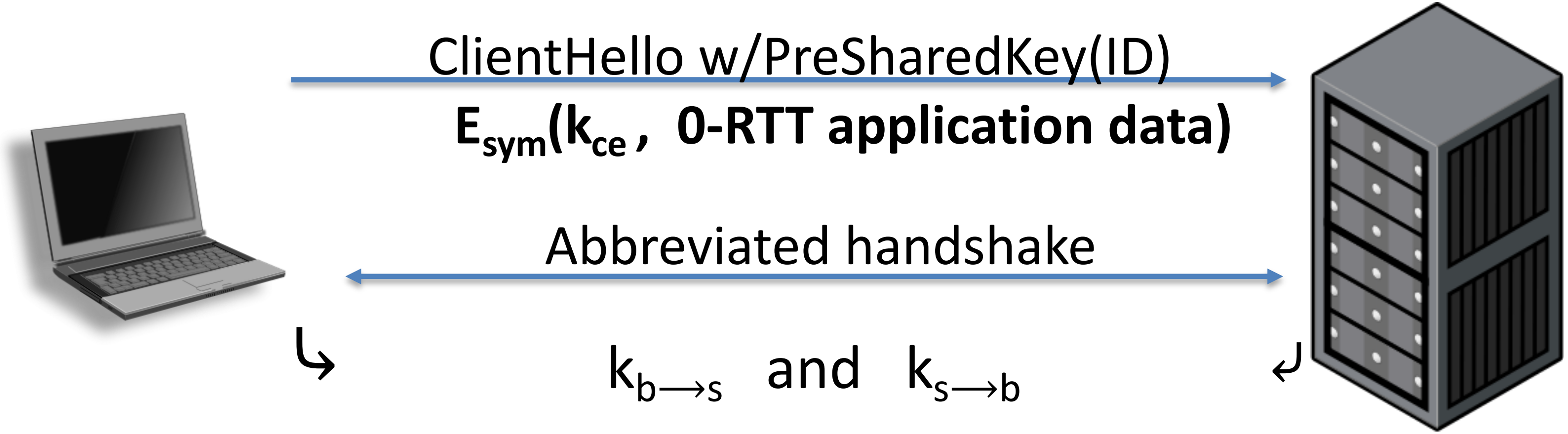
ClientHello w/PreSharedKey(ID)

Abbreviated handshake

↳ $k_{b\longrightarrow s}$ and $k_{s\longrightarrow b}$ ↲

retrieve old **PreSharedKey** **or** recompute from ID

PSK 0-RTT



ClientHello w/PreSharedKey(ID)

$E_{sym}(k_{ce},$ 0-RTT application data)

Abbreviated handshake

↳ $k_{b\longrightarrow s}$ and $k_{s\longrightarrow b}$ ↲

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