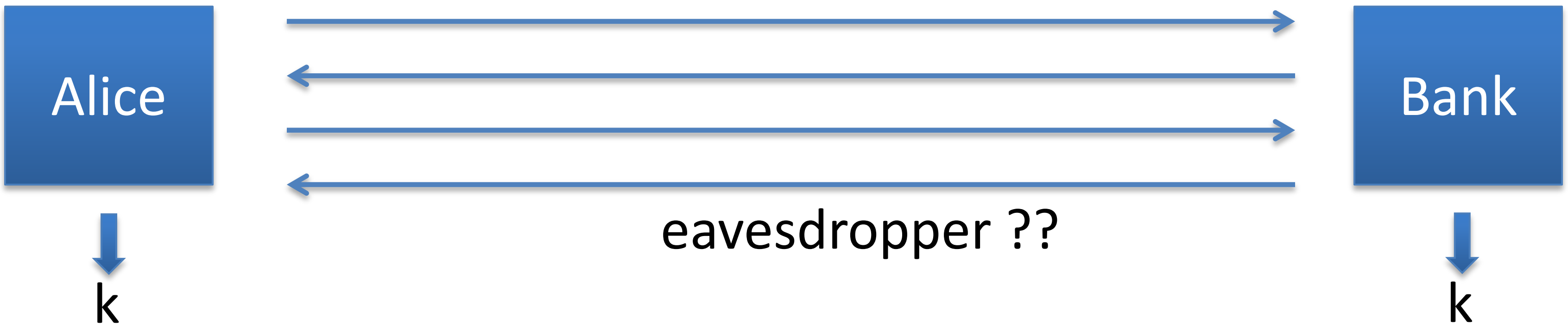
Auth. Key Exchange

Review: key exchange

Alice and Bank want to generate a secret key

- 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

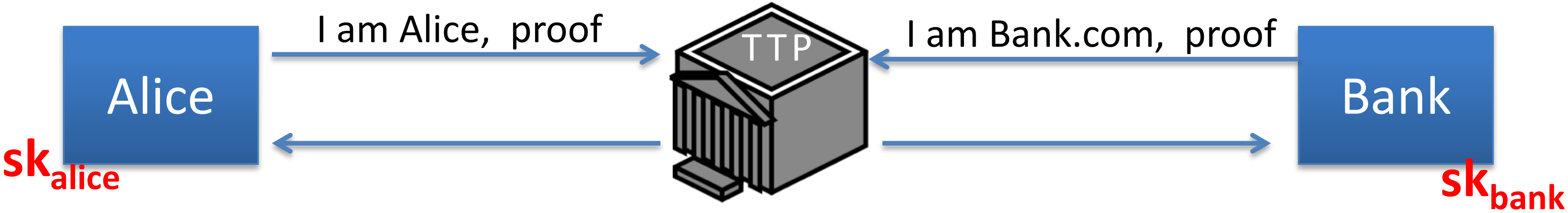
Moreover, some users are honest and others are corrupt

- Corrupt users are controlled by the adversary
  - Key exchange with corrupt users should not “affect” other sessions
- Adversary may corrupt an honest user at time T
  - We want sessions established at time t < T to remain “secure”

Trusted Third Party (TTP)

All AKE protocols require a TTP to certify user identities.

Registration process:



Two types of TTP:

- **Online TTP**: actively participates in <u>every</u> key exchange (Kerberos)

  Benefit: security using only symmetric crypto

- **Offline TTP (CA)**: contacted only during registration (… not quite true)

AKE: syntax

Followed by Alice sending E(k, "data") to Bank

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)**

One-sided AKE

Security: authenticity for Alice and secrecy for Alice

- Bank has no guarantees for identity of peer (no consistency)
- Commonly used on the Web (often followed by ID protocol)

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} .

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

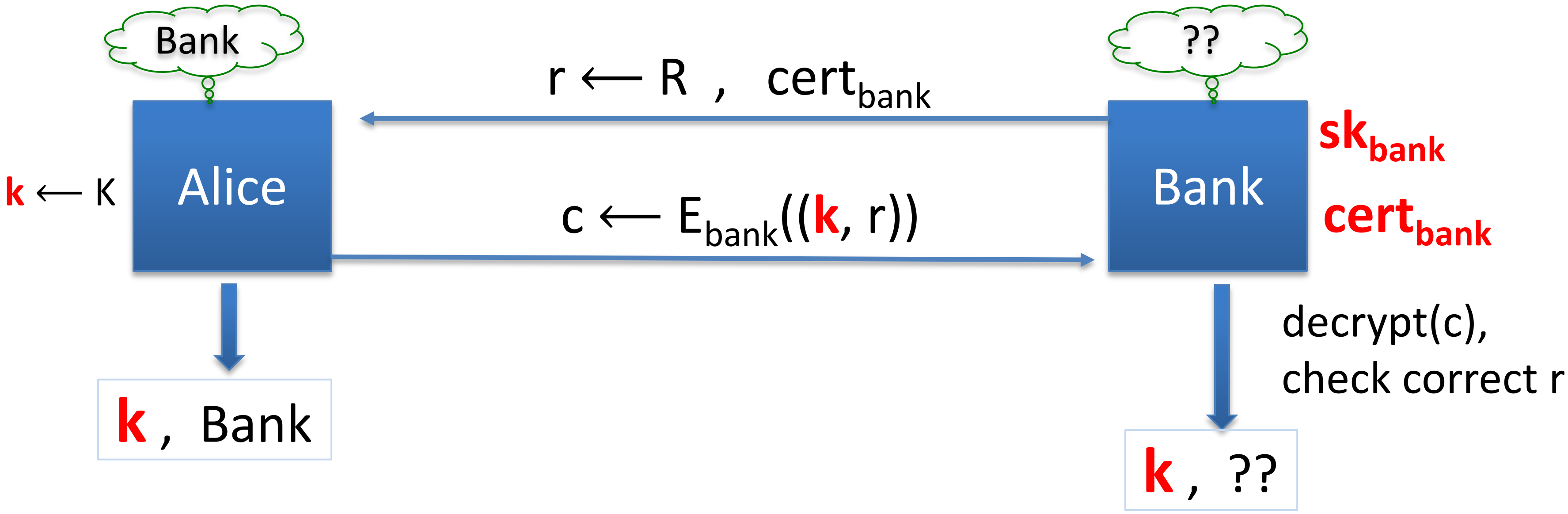
- Recall: from $E_{bank}((m,r))$ adv. cannot build $E_{bank}((m,r'))$ for $r' \neq r$

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

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

Protocol #1

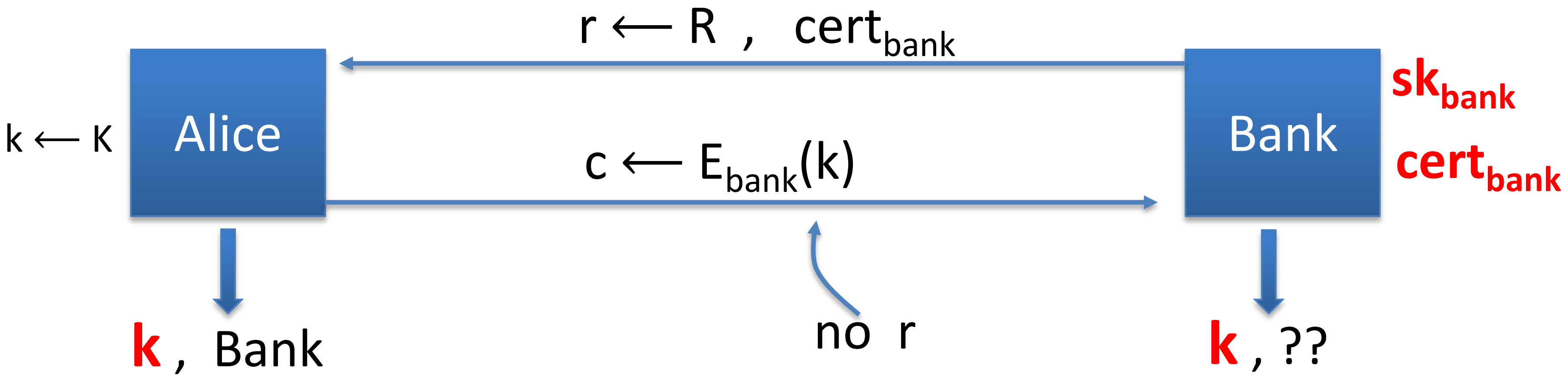
Simple one-sided AKE protocol



“Thm”: this protocol is a secure one-sided AKE

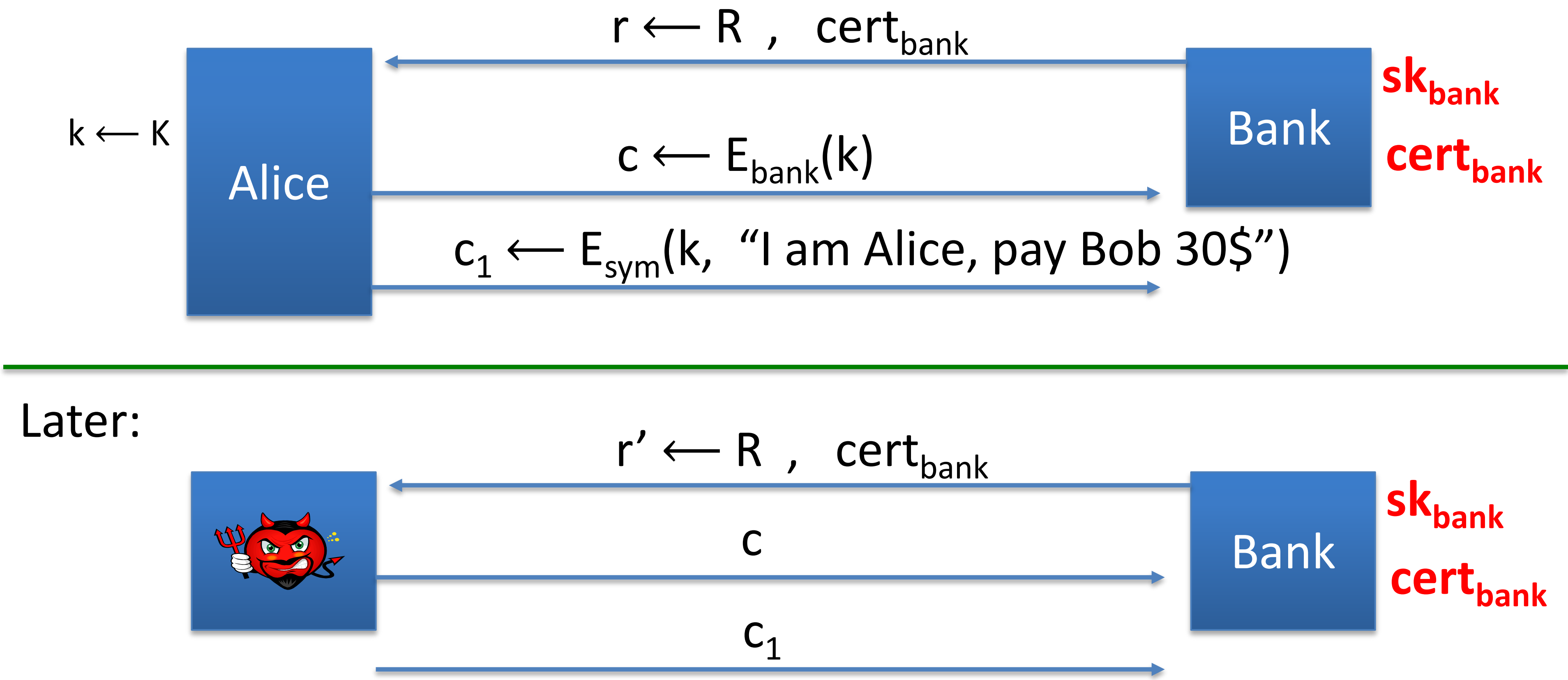
Informally: if Alice and Bank are not corrupt then we have (1) secrecy for Alice and (2) authenticity for Alice

Insecure variant 1: r not encrypted



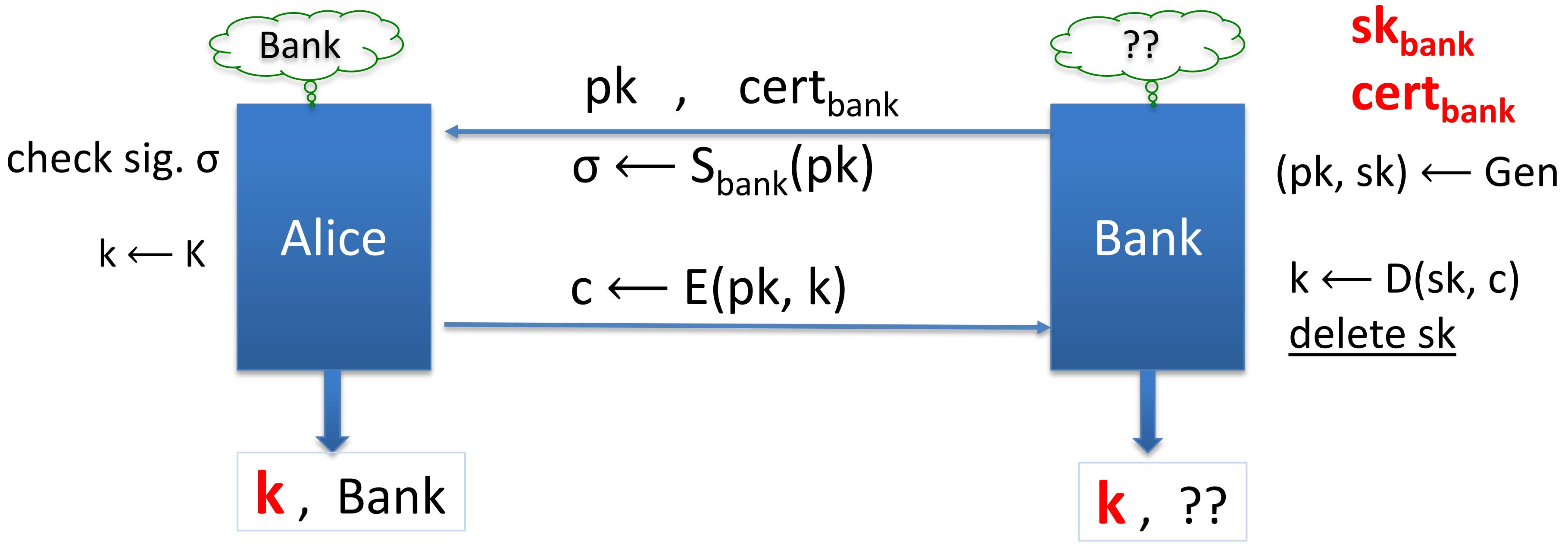
Problem: replay attack

Replay attack



Protocol #2

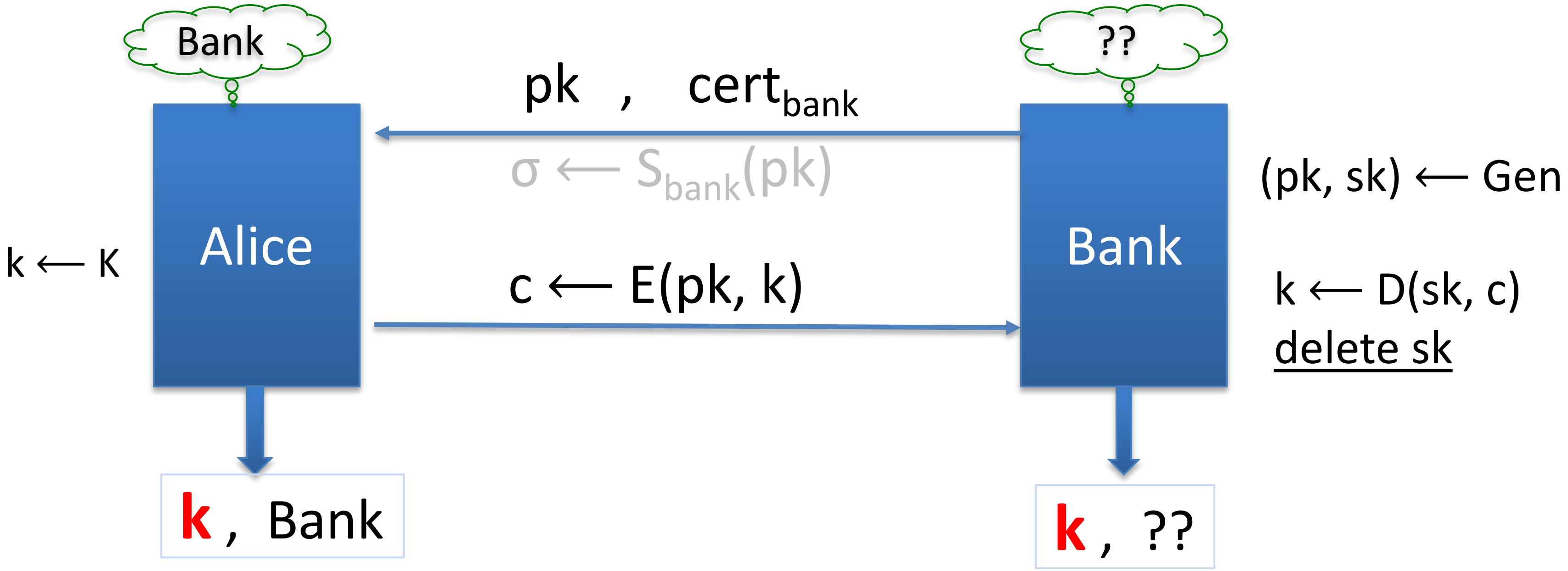
Simple one-sided AKE with forward-secrecy



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

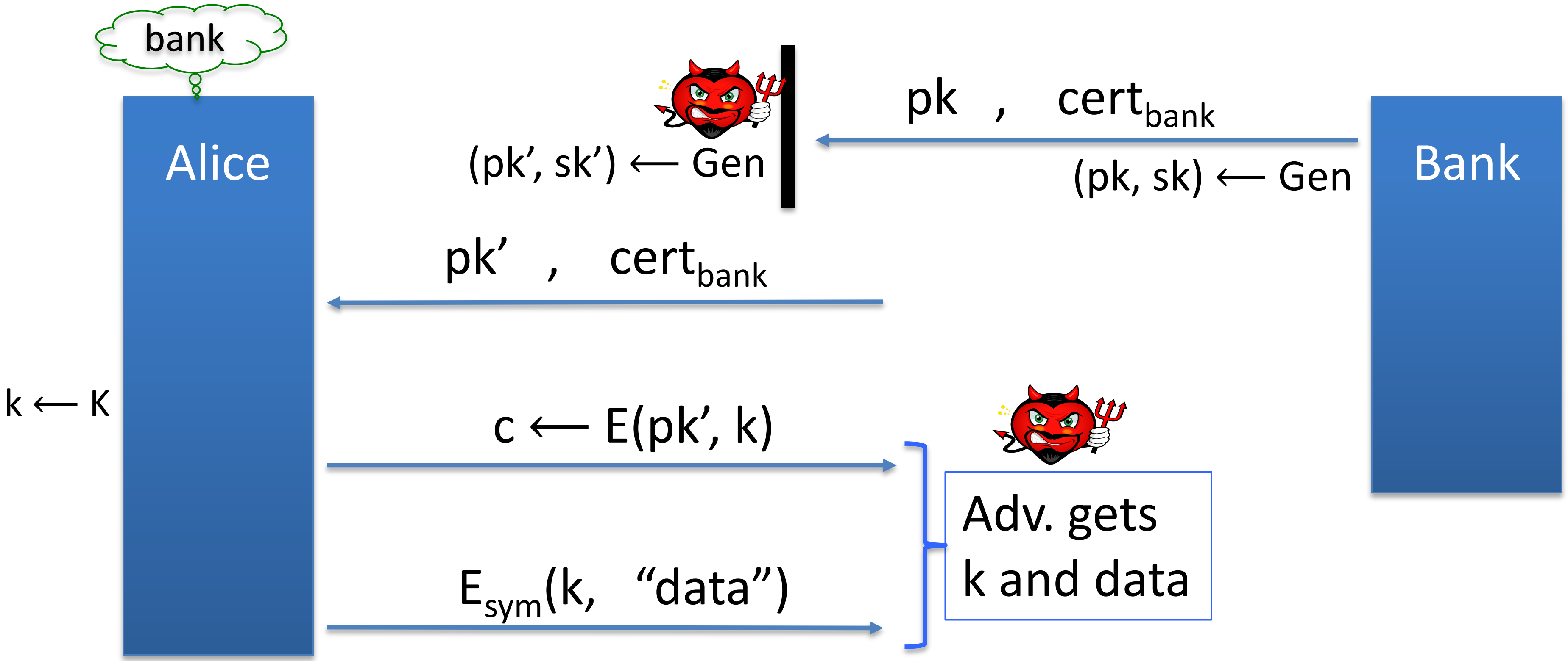
Compromise of Bank: past sessions are unaffected

Insecure variant: do not sign pk



Attack: complete key exposure

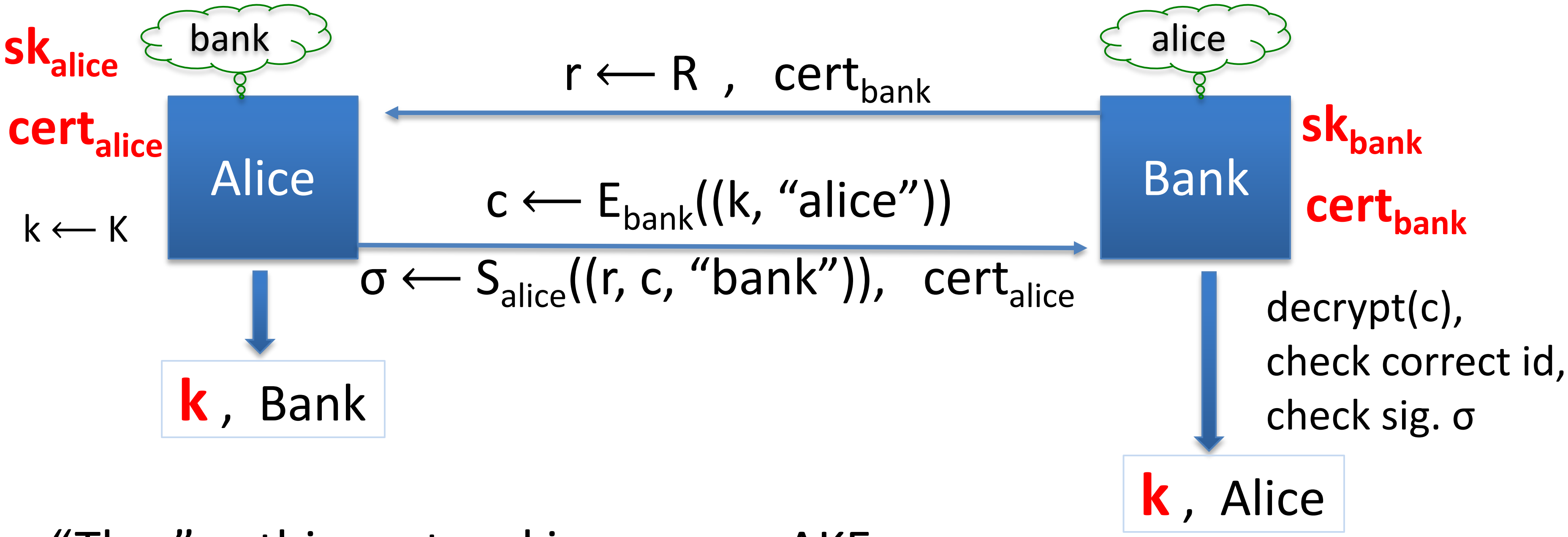
Attack: key exposure



Two-sided AKE

For now: no forward secrecy

Two-sided AKE (mutual authentication)



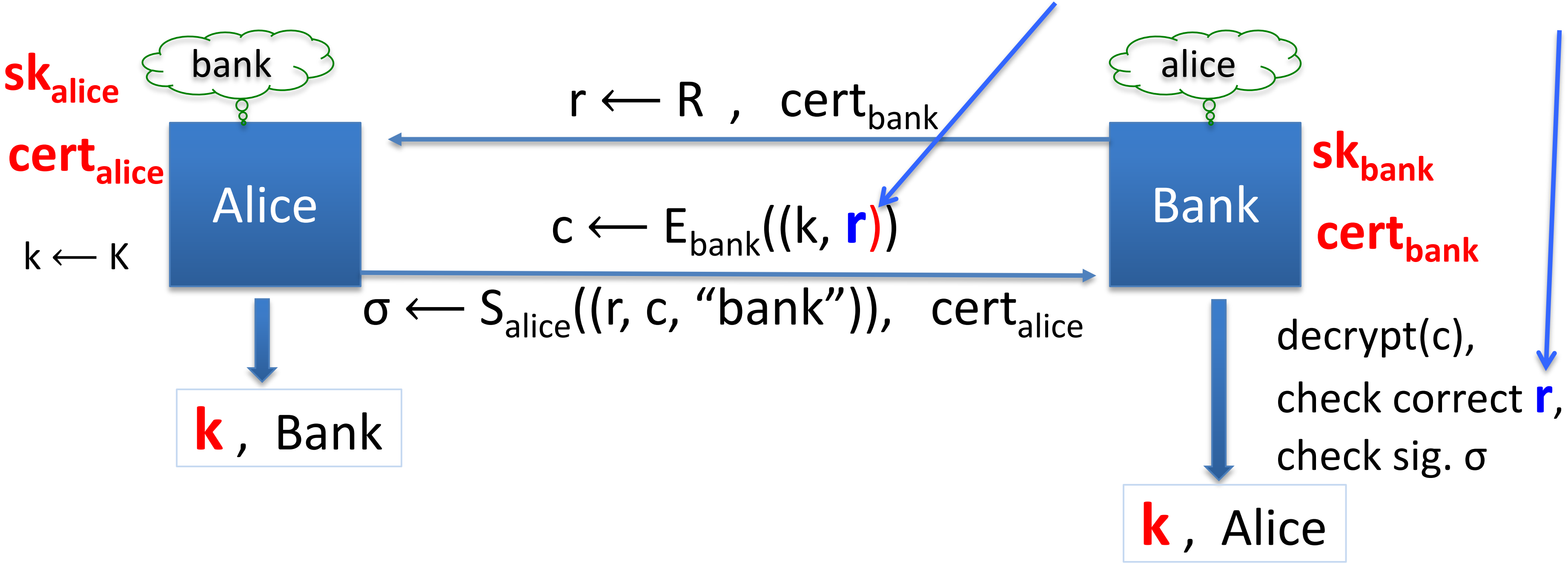
“Thm”: this protocol is a secure AKE

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

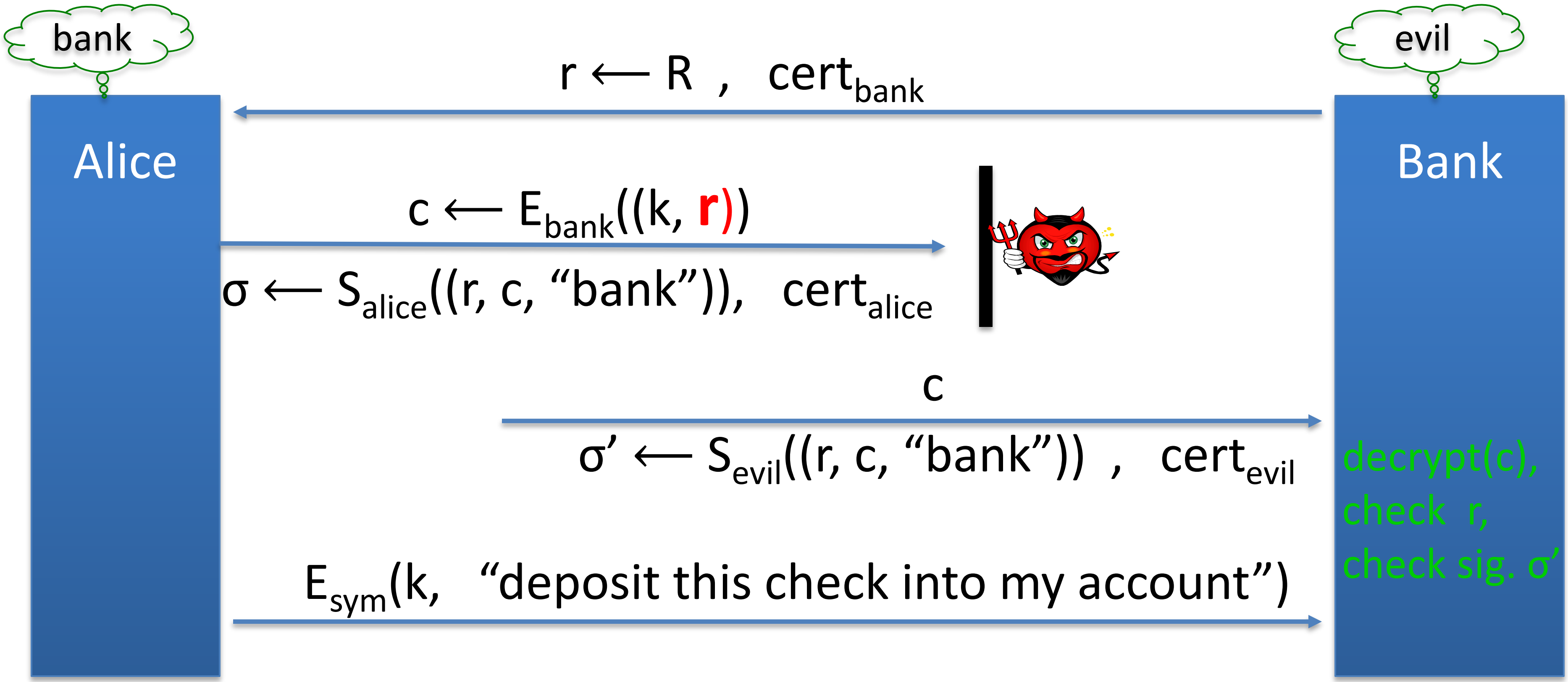
Insecure variant: encrypt r instead of "Alice"

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

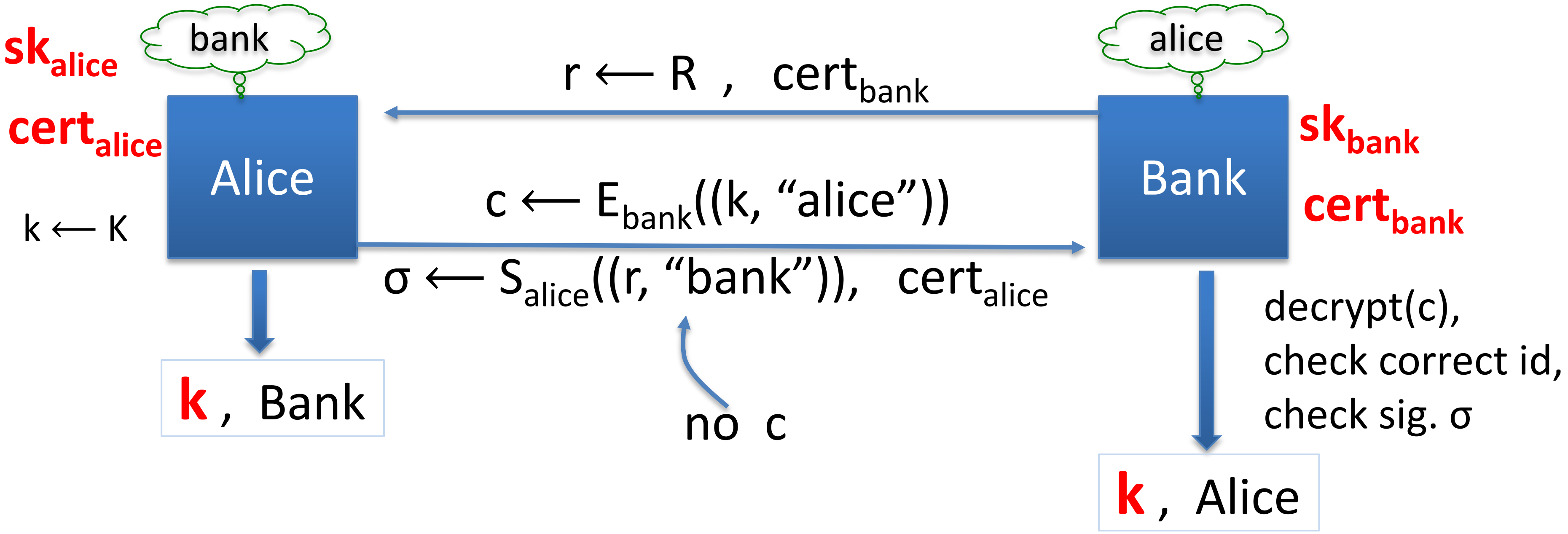
Example:



Attack: identity misbinding

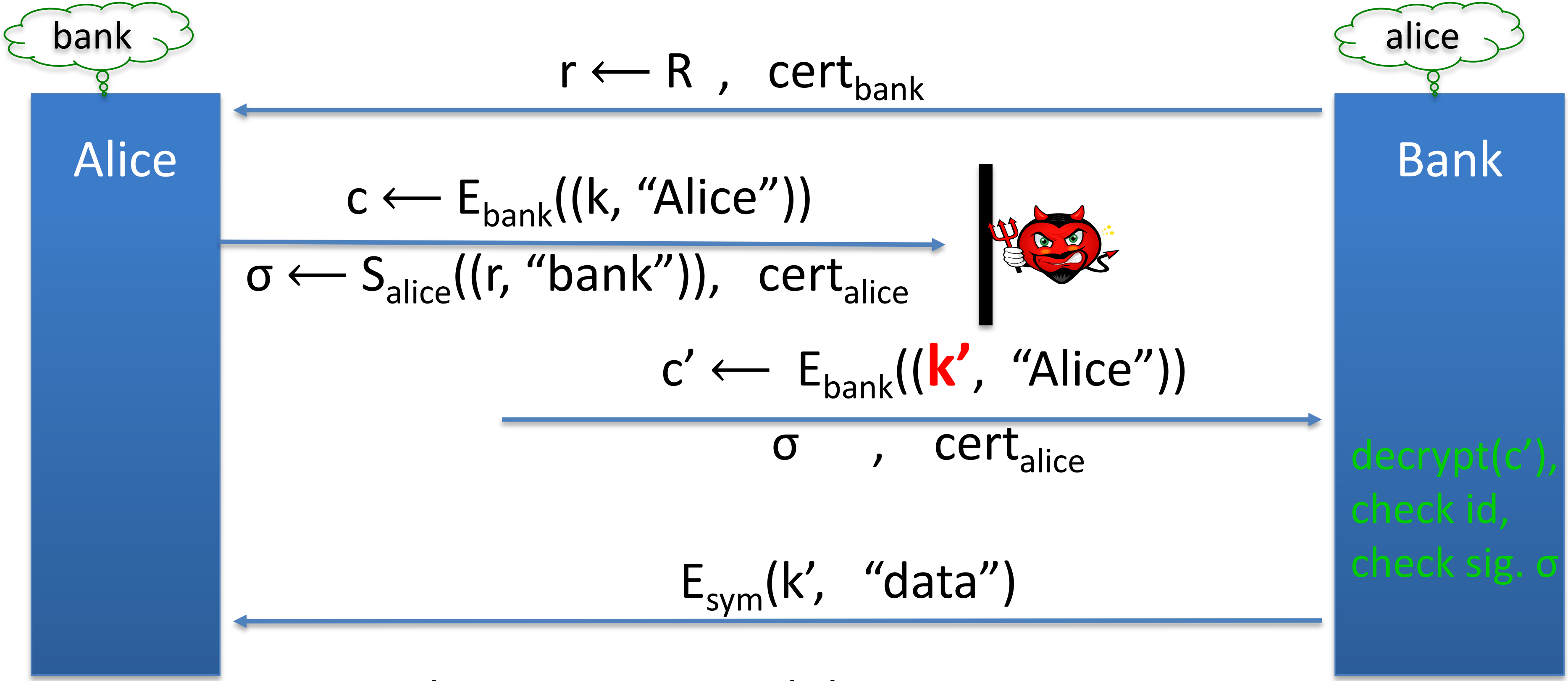


Insecure variant: do not sign c



Attack: key exposure

Attack: key exposure



Adversary can read data

Many more AKE variants

Two-sided AKE with forward secrecy:

AKE with end-point privacy:

- Goal: certificates are not visible to adversary (TLS 1.3)

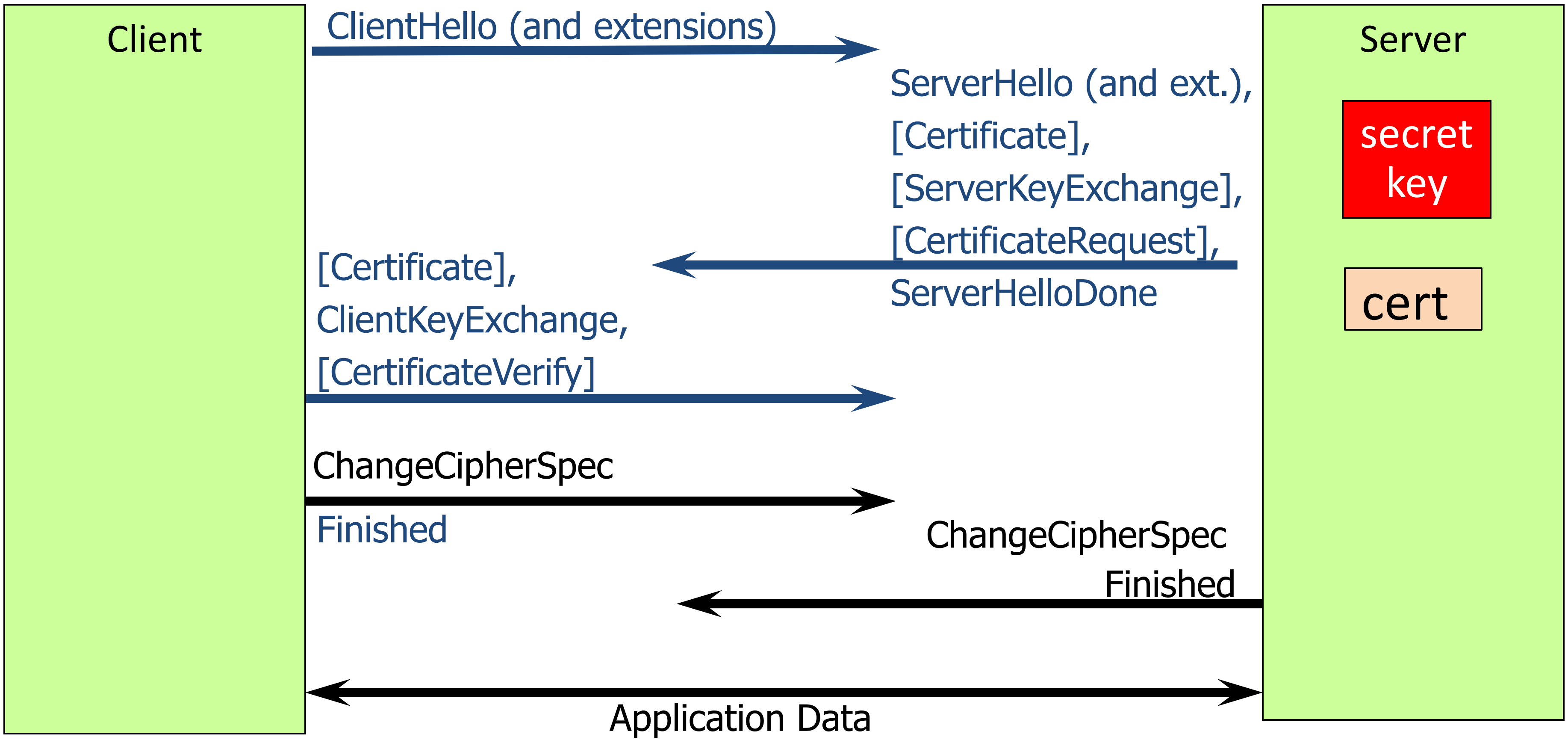
AKE based on a shared secret between Alice and Bank:

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

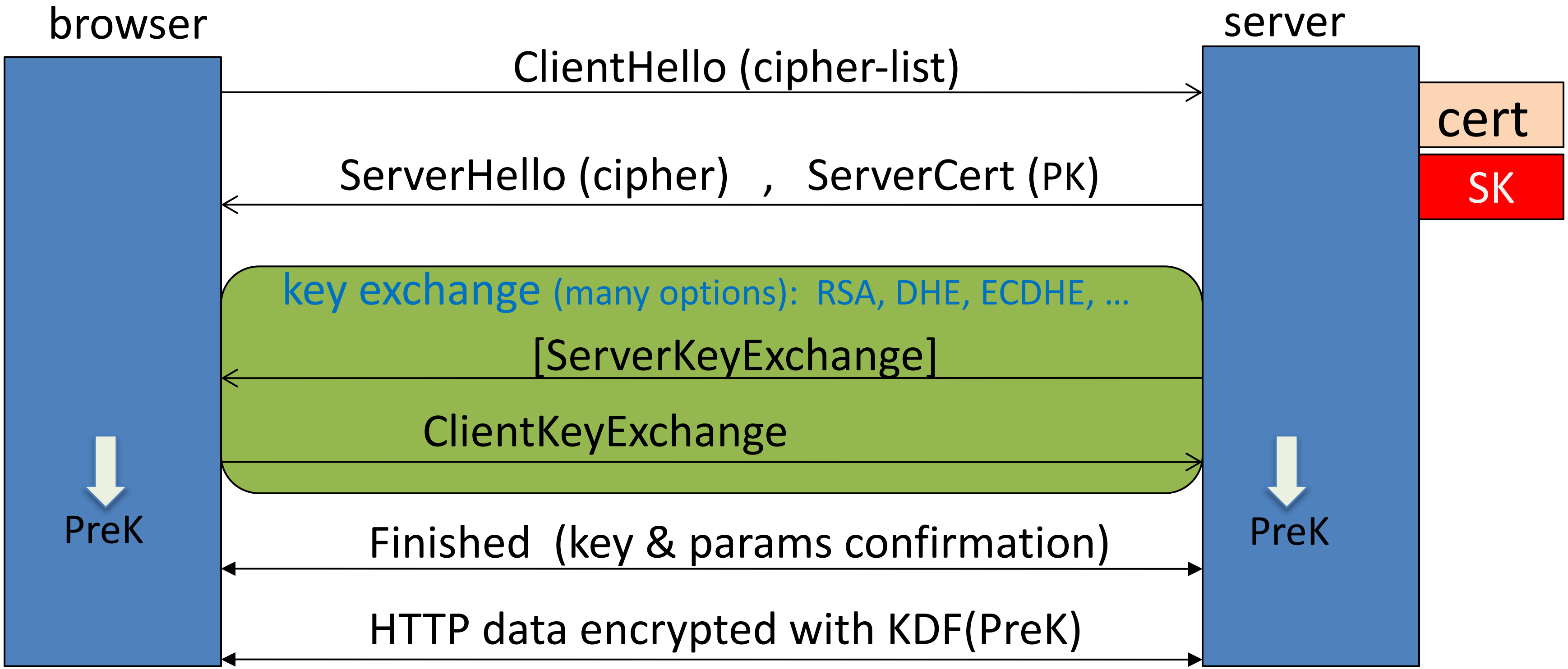
Auth. key exchange

TLS v1.2 key exchange

TLS session setup (handshake)



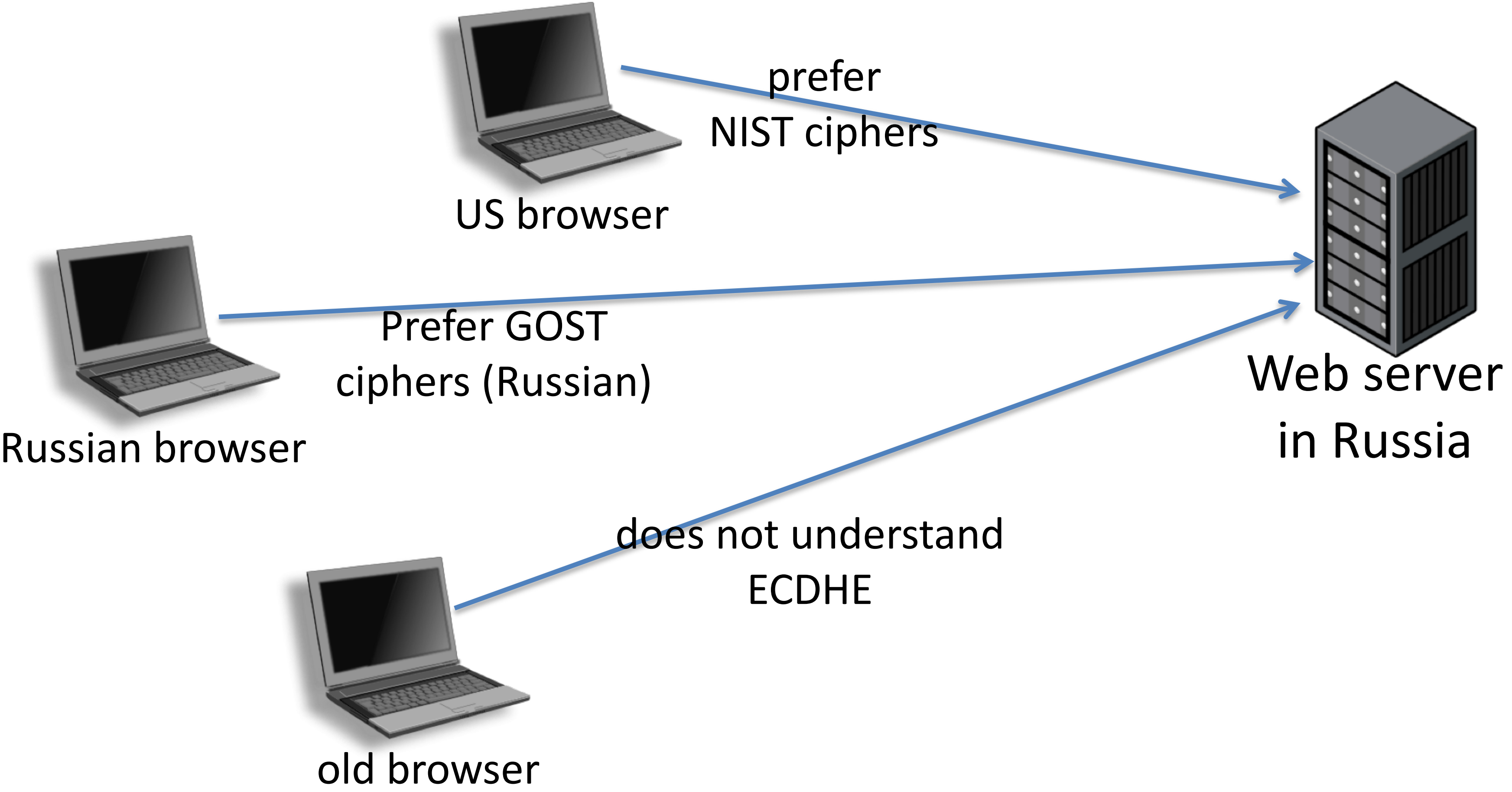
Brief overview of SSL/TLS



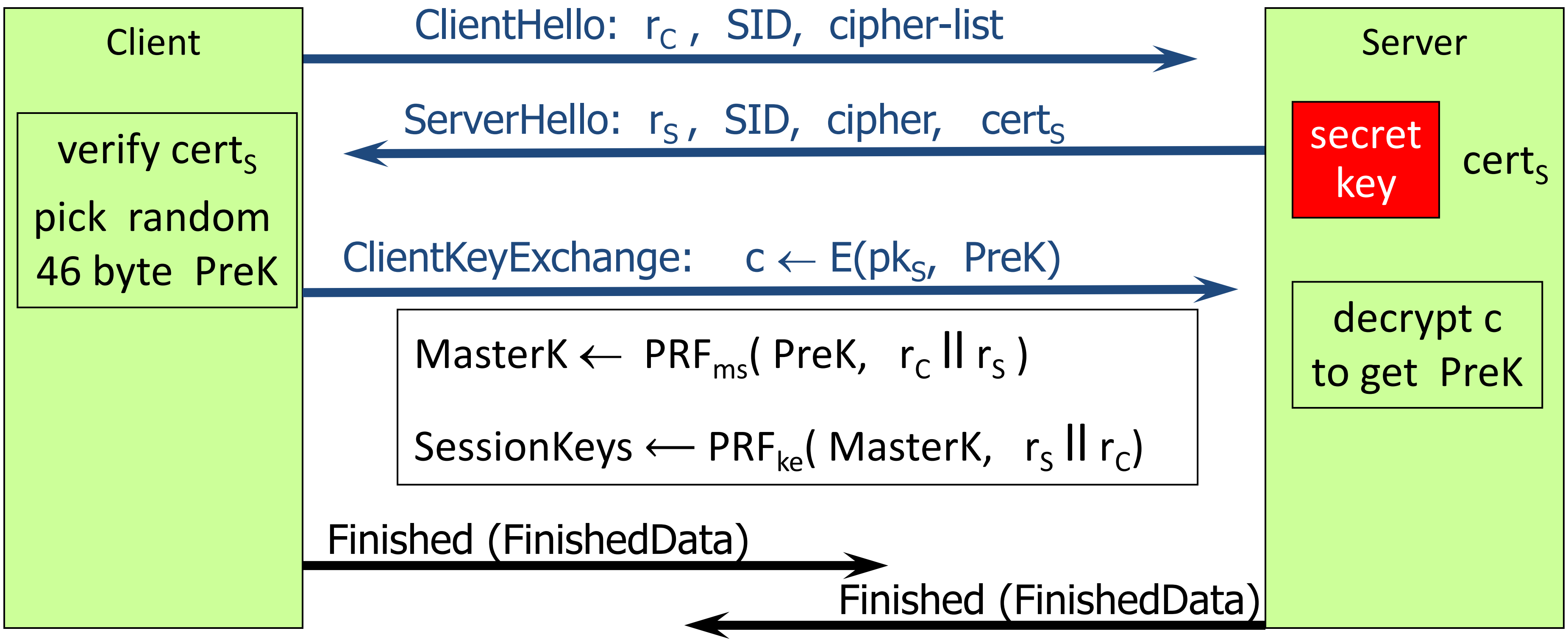
In this diagram: one sided authentication (no client authentication)

Dan Boneh

The need for negotiating ciphers



Abstract TLS: RSA exchange (simplified)



Key Confirmation: FinishedData = PRF$_{vd}$(MasterK, hash(HandshakeMessages))

Properties

r_C , r_S: prevent replay of of old session

RSA key exchange: no forward secrecy

– Compromise of server secret key exposes old sessions

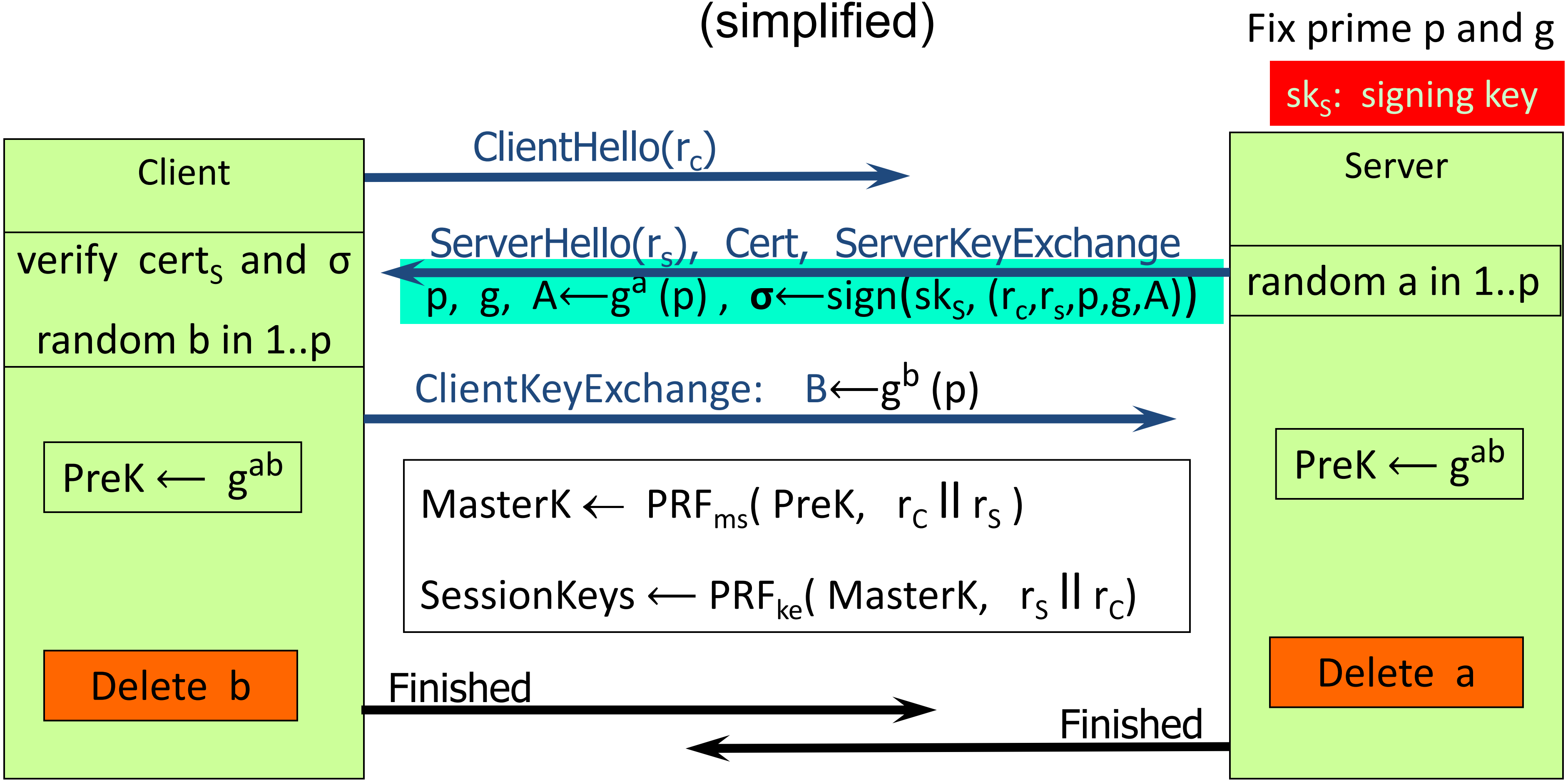
– Costly RSA decryption on server, easier RSA enc. on client

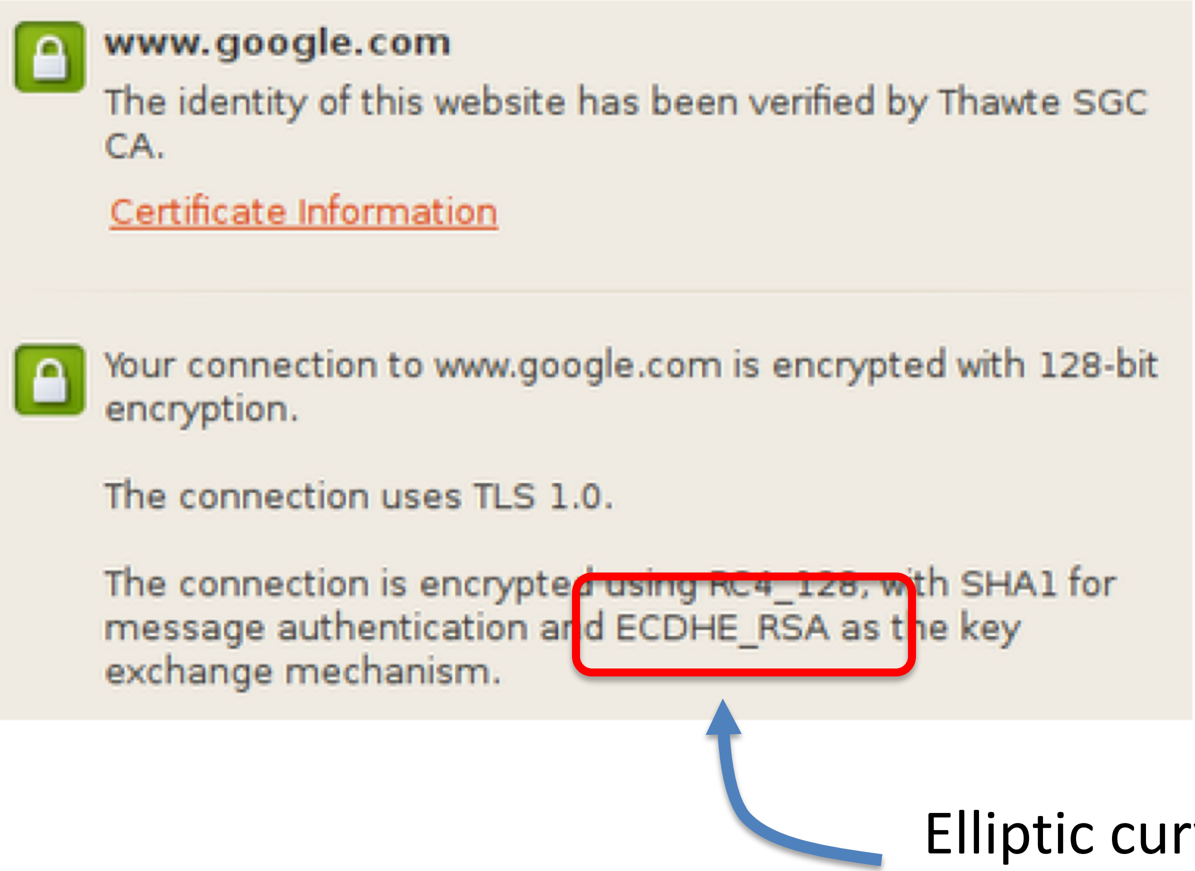
One sided identification:

- Browser identifies server using server-cert
- Server has no guarantees about client's identity
  - TLS has support for mutual auth. (client needs sk_C and $cert_C$)

TLS key exchange with forward-secrecy (DHE)

(simplified)





Elliptic curve Diffie-Hellman

Prefer ECDHE over DHE

Performance: RSA vs. forward-secrecy

Cost of crypto operations on server per handshake:

- RSA key exchange: one RSA-2048 decryption (deprecated in TLS 1.3)
- ECDHE: Diffie-Hellman in group G with generator $g \in G$

1. One exp. to compute $A \longleftarrow g^a \in G$
2. One sig. on Diffie-Hellman parameters (G,g,A)

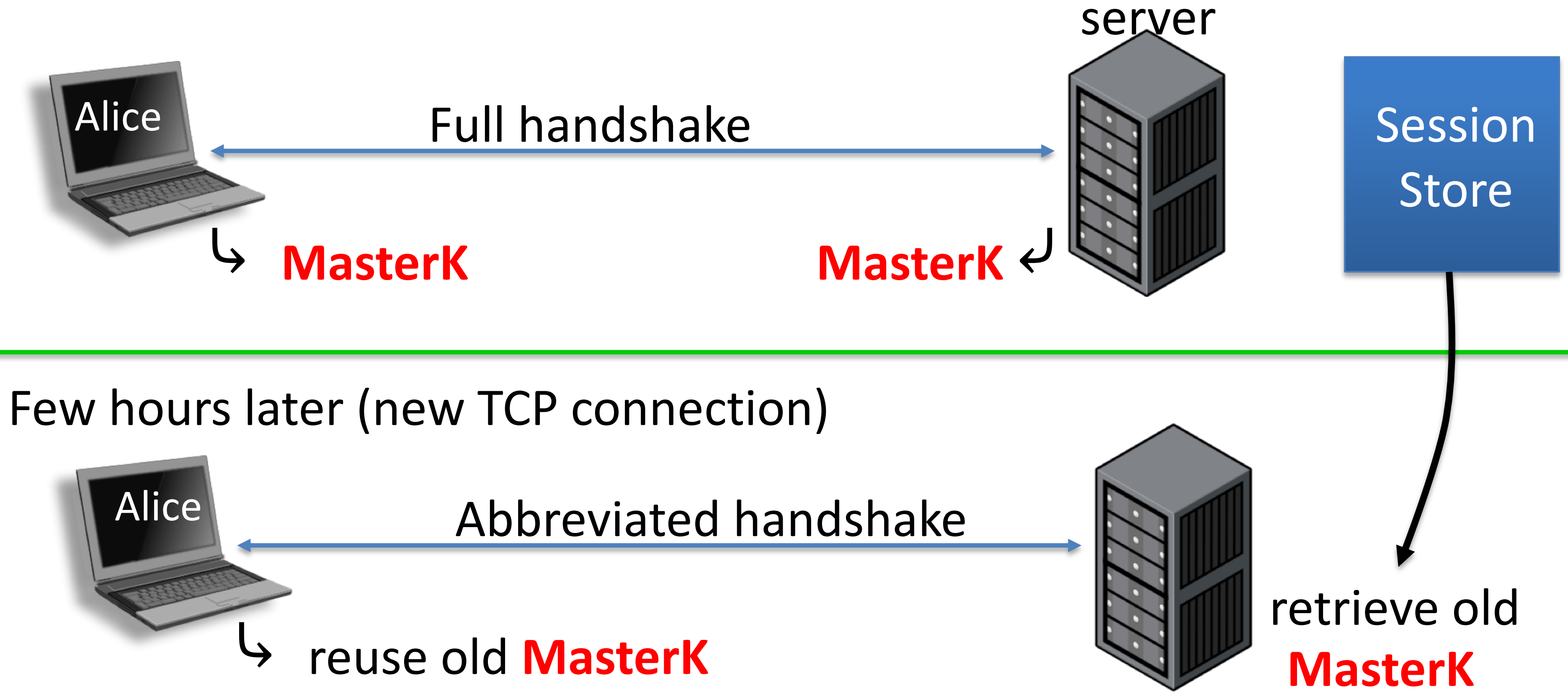
must be done for every handshake

3. One exp. to compute DH secret: $PreK \longleftarrow g^{ab} \in G$

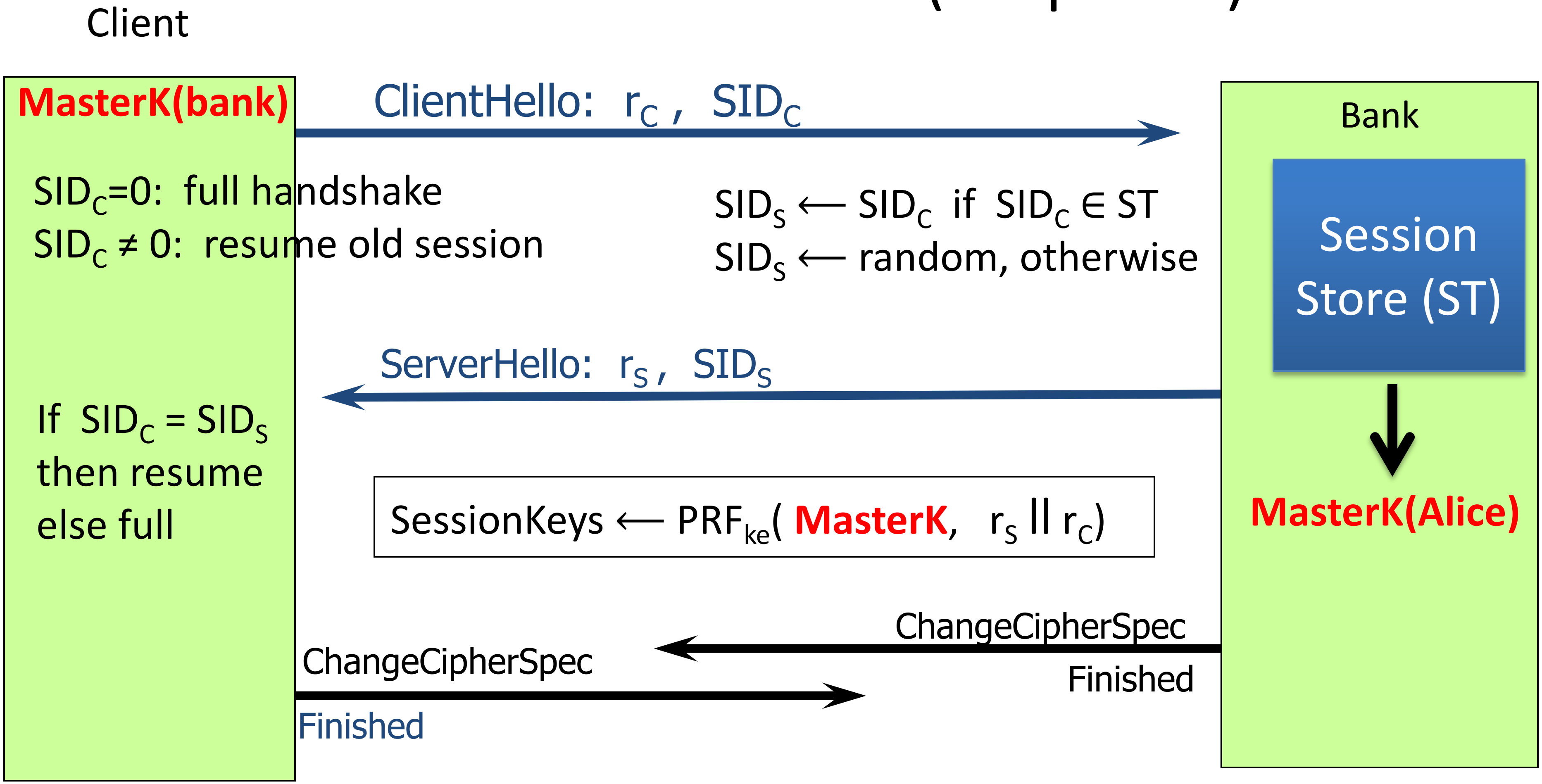
Server support (2014): RSA (99.9%) , DHE (60%) , ECDHE(18%)

Session Resume

Goal: reduce # of full handshakes



Session resume (simplified)



THE END