INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

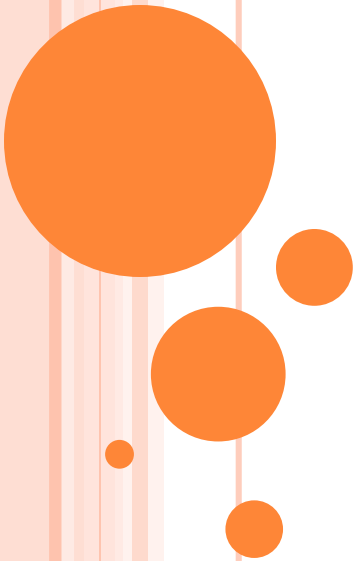
UMR IRISA

Proxying over TLS: Breaking and Fixing Keyless SSL

With P.-A. **Fouque** (UR1),
K. **Bhargavan** (Inria de Paris)
I. **Boureanu** (Univ. of Surrey)
B. **Richard** (**Orange**)

EMSEC
embedded security & cryptography

Authenticated Key Exchange



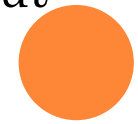
Secure Channels

- Goal:

Secure communication over insecure channels

- Insecure channels:
  - The Internet (HTTP://)
  - Mobile networks (2G, 3G, 4G…)
  - Bluetooth
  - Radio Frequency Channels

- "Secure" channels:
  - Messages exchanged over this channel could be intercepted, but not read by active 3rd parties (Man-in-the-Middle attackers)



Some Classic Crypto

- Encryption:

  Hiding messages from all unauthorized users

  - Authorized user: user who has "decryption" key.
  - Plenty of choice:
    - Symmetric-key: block ciphers, stream ciphers
    - PKE schemes: RSA, ElGamal, Paillier…

- Message authentication:

  Only authorized users may modify messages

  - MACs and signatures:
    - MACs: symmetric key, signer and verifier share same key
    - Signatures: public key, private key used for signing
      public key used for verification

Constructing a Secure Channel

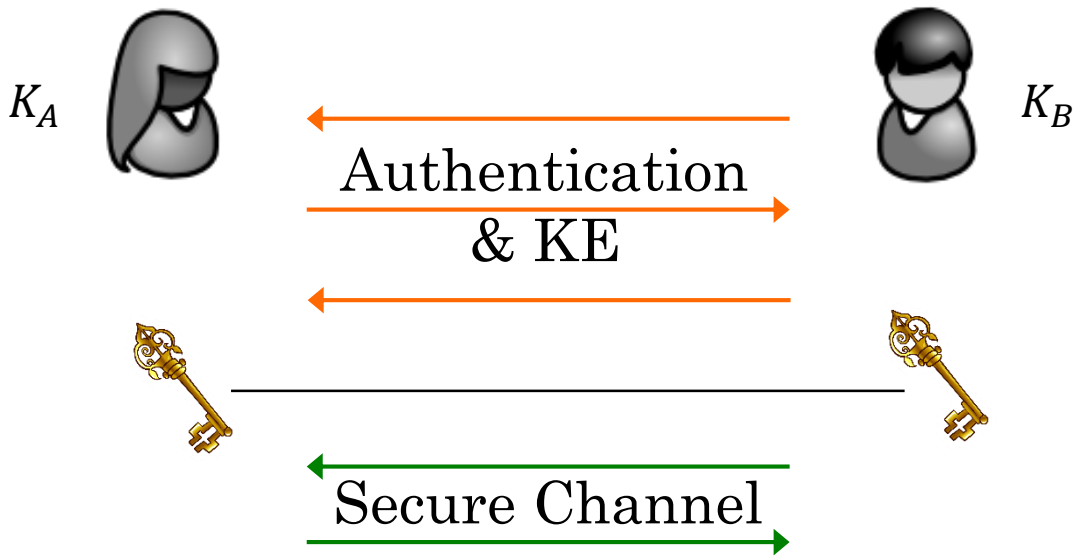
- Our original goal:

  Secure communication over insecure channels

- How to achieve it:
  - Encryption and authentication can help
  - … but users will need keys!

- (Authenticated) Key-Exchange:
  - Alice and Bob exchange data across insecure channel
  - At the end they derive a set of keys, usable for authenticated encryption
  - By using authenticated encryption on their messages, they construct the secure channel

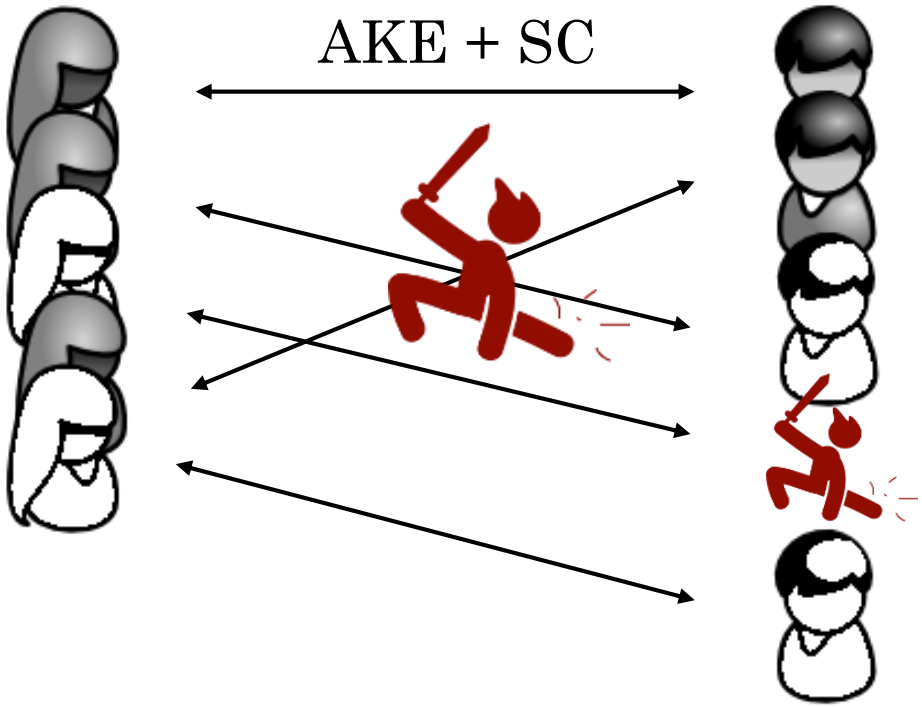
Typical 2-Party AKE



- Symmetric-key AKE: $K_A = K_B = sk$
- Public-key AKE: $K_A = (sk_A, pk_A); K_B = (sk_B, pk_B)$

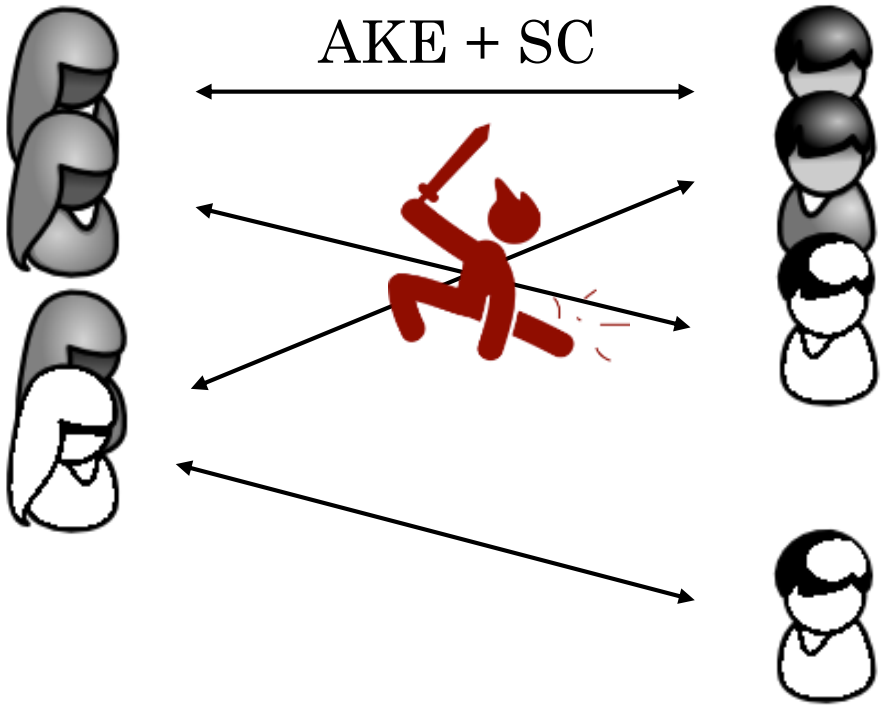
Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties



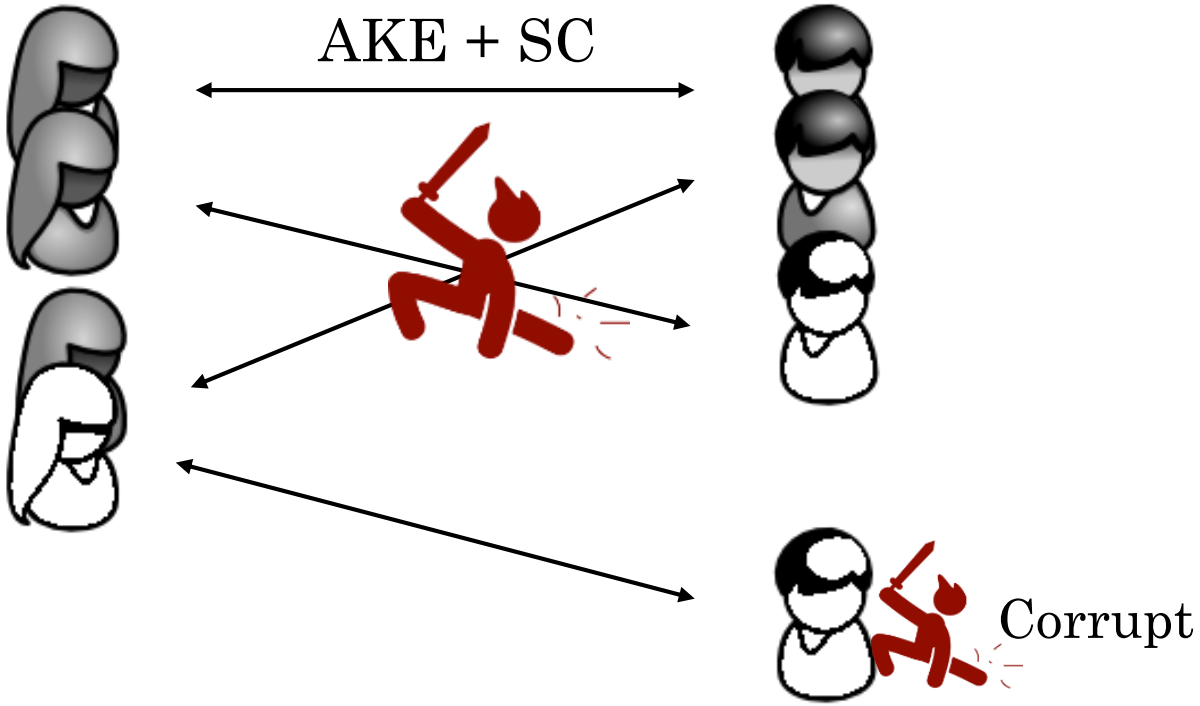
Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties



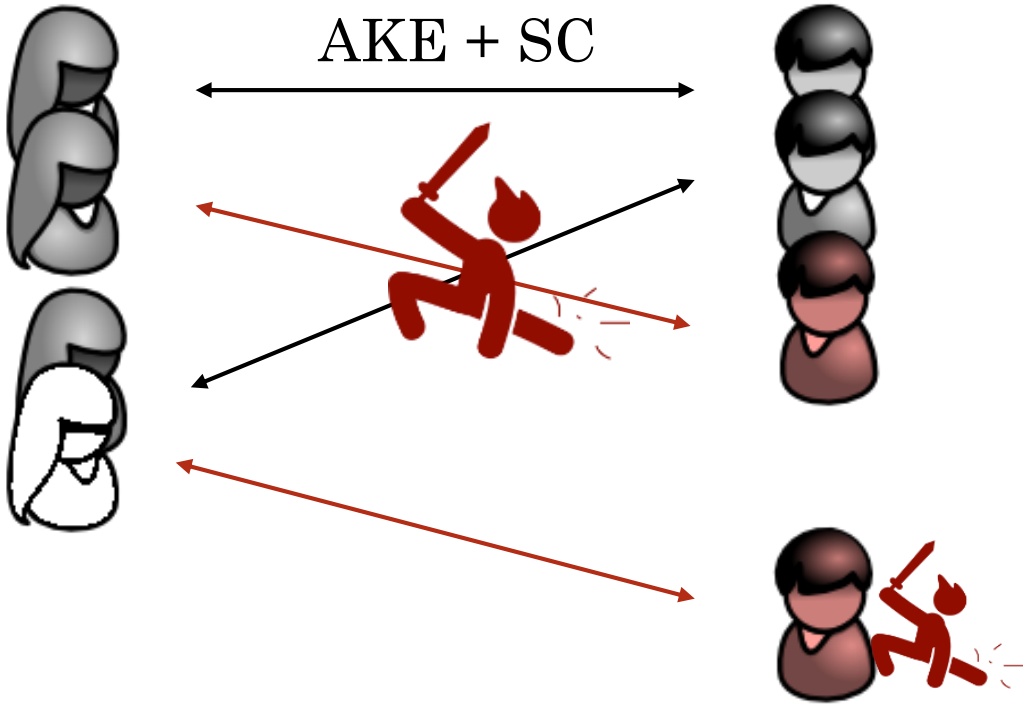
Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys



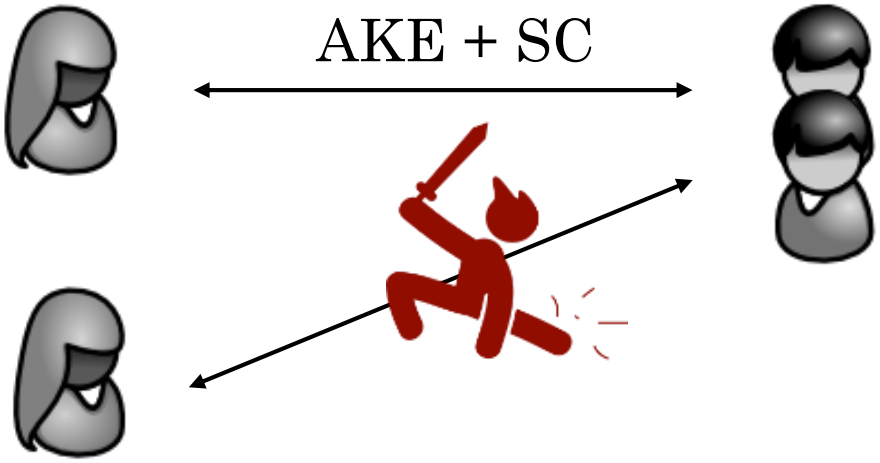
Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys



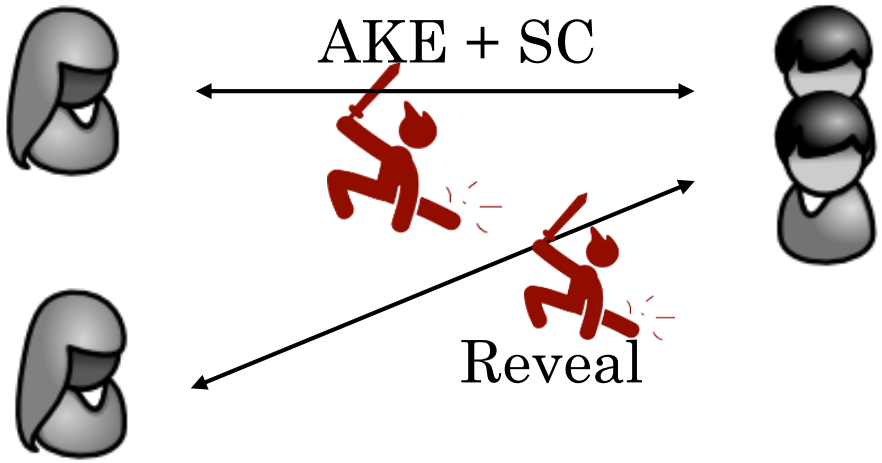
Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys



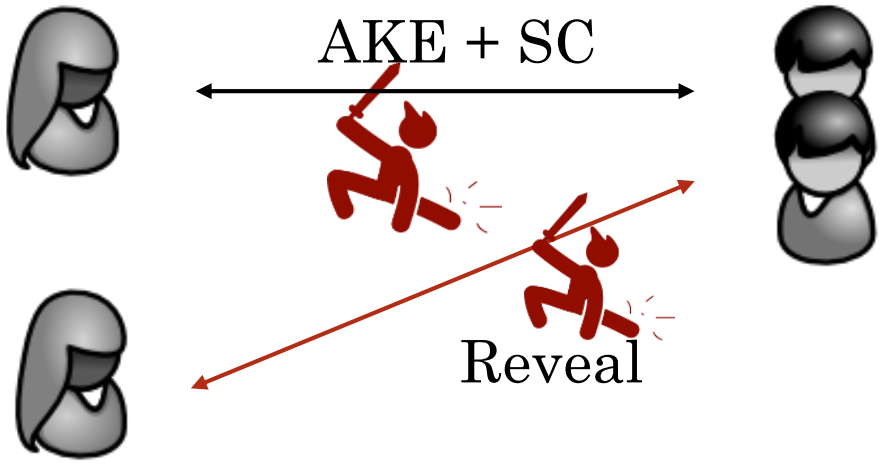
Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys
  - Can reveal computed session keys



Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys
  - Can reveal computed session keys

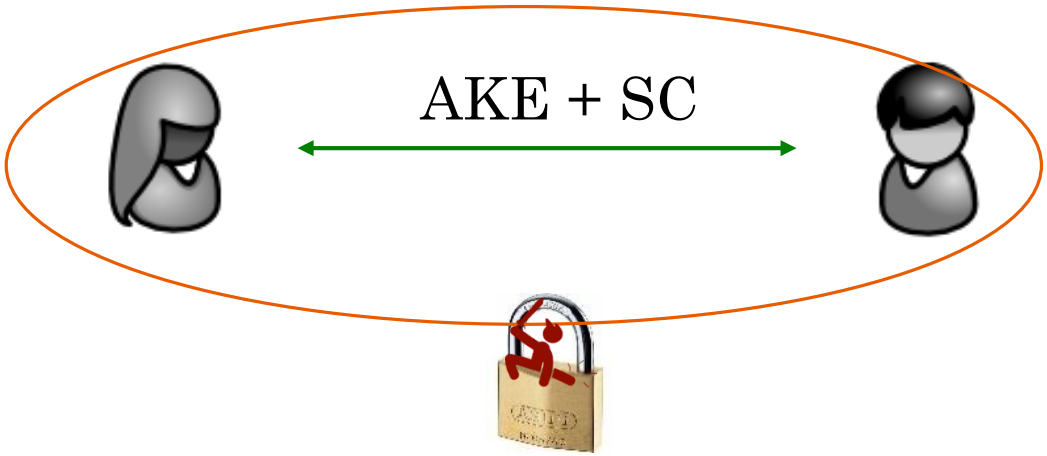


Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys
  - Can reveal computed session keys

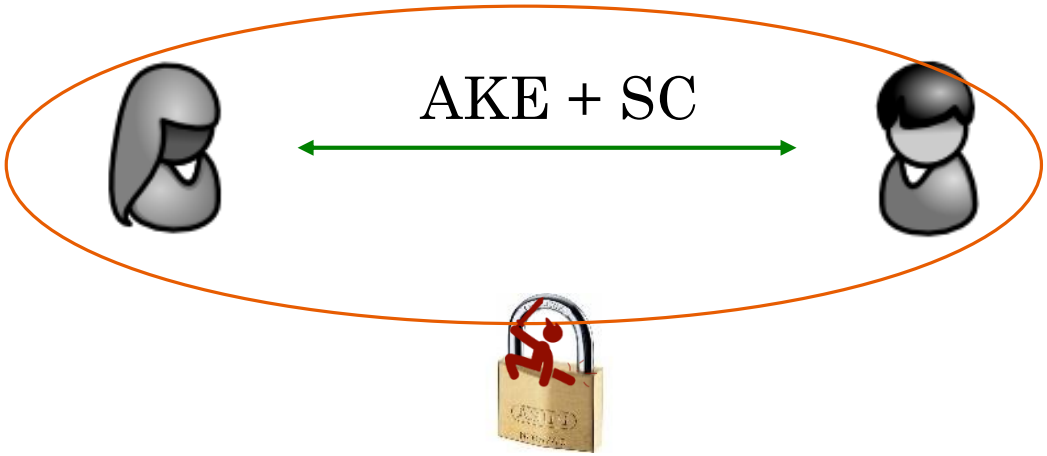
Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys
  - Can reveal computed session keys



Security of AKE

- Meet the adversary:
  - A Man-in the Middle, aims to break channel security
  - Can interact in multiple sessions of many parties
  - Can corrupt parties to learn long-term keys
  - Can reveal computed session keys
  - Forward-secrecy: if the adversary corrupts a user, it cannot break the security of past sessions



Synopsis

- AKE protocols address a fundamental goal:

Secure communication over insecure channels

- Typical 2-step structure:
  - Authentication & KE: the two parties derive session keys
  - Secure Channel: use session keys to secure communication

- Secure AKE:
  - Unilateral/mutual authentication
  - The established, untainted channels are secure
  - Forward-secrecy: even if long-term keys are compromised, past sessions are secure

But this security only holds for 2 party protocols

Real-world AKE

- In practice, ensures:
  - Secure Internet browsing (TLS/SSL)
  - Secure emailing
  - Mobile services (AKA)
  - Payments
  - Personal identification (ID cards/passports)

- Usually Alice and Bob are a Client and a Server
- Security of protocol only proved for 2-party use
  - Yet sometimes, handshakes are proxied, by semi-trusted third parties

Is the resulting protocol still secure?

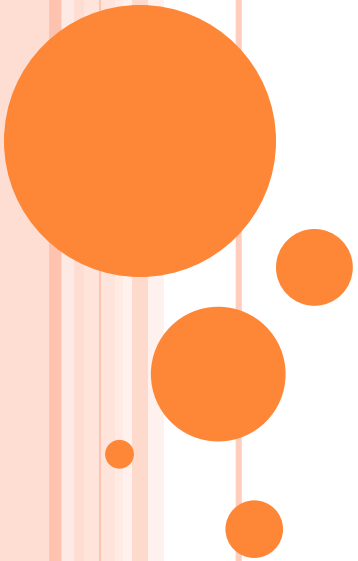
Proxying

- Client-Server connections are rarely direct:
  - Routing
  - Firewalls
  - Content delivery networks
  - Cross-operator services

- The proxying party is only semi-trusted
  - Trusted to perform its task: it must authenticate to the other party, and it will know the established session keys
  - Untrusted to know the client/server's long-term secrets (privacy)
  - Untrusted to know session keys of sessions it is not involved in

- Example: CDNs and Keyless SSL

EXAMPLE 2 : KEYLESS SSL



Content Delivery Networks (CDNs)

- HTTP: client retrieves webpage content from server
  - Clients expect speed
  - Physical distance makes for slow traffic
  - Solution: contract a provider that uses physically-close edge servers to cache and deliver the sought content
- Content delivery networks:
  - Edge server receives request and either forwards cached content, or requests content and sends it

Easy for HTTP, hard for HTTPS!

TLS/SSL

- HTTPS uses TLS/SSL (version 1.2 hopefully)
  - A 2-party AKE protocol which is provably secure
- TLS/SSL is public-key AKE protocol, providing:
  - server-to-client authentication
  - optional client-to-server authentication
  - forward secrecy (some modes only)
- Server authentication done by means of PKI:
  - Server has certified public key
  - During the protocol, the corresponding secret key is used
  - Session resumption: shortcut requiring no secret key, using a pre-established secret

THE 2-PARTY TLS PROTOCOL

Client

Server

$sk_S, pk_S, Cert_S$

N_C, ConfigList, ExtList →

← N_S, Config, Ext

← $KE_S = (pk_S, Cert_S)$

Choose pmk
Set: $msk = HMAC_{pmk}(N_C, N_S)$
$KE_C = Enc_{pk_S}(pmk)$
$(K_C, K_S) = HMAC_{msk}(N_C, N_S)$
$CFin = HMAC_{msk}(H[N_C \dots KE_C])$

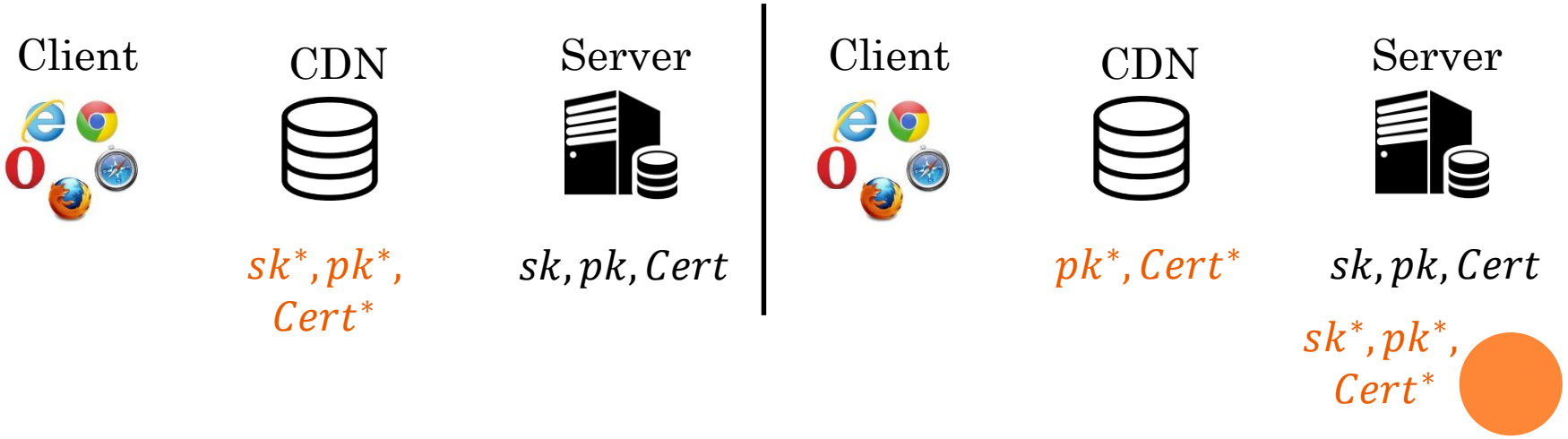
KE_C, $\{CFin\}_{K_C}$ →

Decrypt KE_C to get pmk
Compute msk, (K_C, K_S)
Decrypt $\{CFin\}_{K_C}$
$SFin = HMAC_{msk}(H[N_C \dots CFin])$

← $\{SFin\}_{K_S}$

CDN & Keyless SSL

- PKI for proxied TLS:
  - Client queries "www.abc.com".
  - Server/CDN agree on CDN representing www.abc.com.
  - Classic: the CDN has an (sk, pk, Cert) tuple for www.abc.com
  - Keyless: the CDN has (pk, Cert), but not sk
    - The CDN needs to query the server to get the key



KEYLESS SSL

pk, Cert

sk, pk, Cert

N_C, ConfigList, ExtList →

← N_S, Config, Ext

← $\text{KE}_S = (pk_S, \text{Cert}_S)$

Choose pmk
Set: $msk = \text{HMAC}_{pmk}(N_C, N_S)$
$KE_C = \text{Enc}_{pk_S}(\text{pmk})$
$(K_C, K_S) = \text{HMAC}_{msk}(N_C, N_S)$
$\text{CFin} = \text{HMAC}_{msk}(H[N_C \ldots KE_C])$

$\text{KE}_C, \{\text{CFin}\}_{K_C}$ →

KE_C →

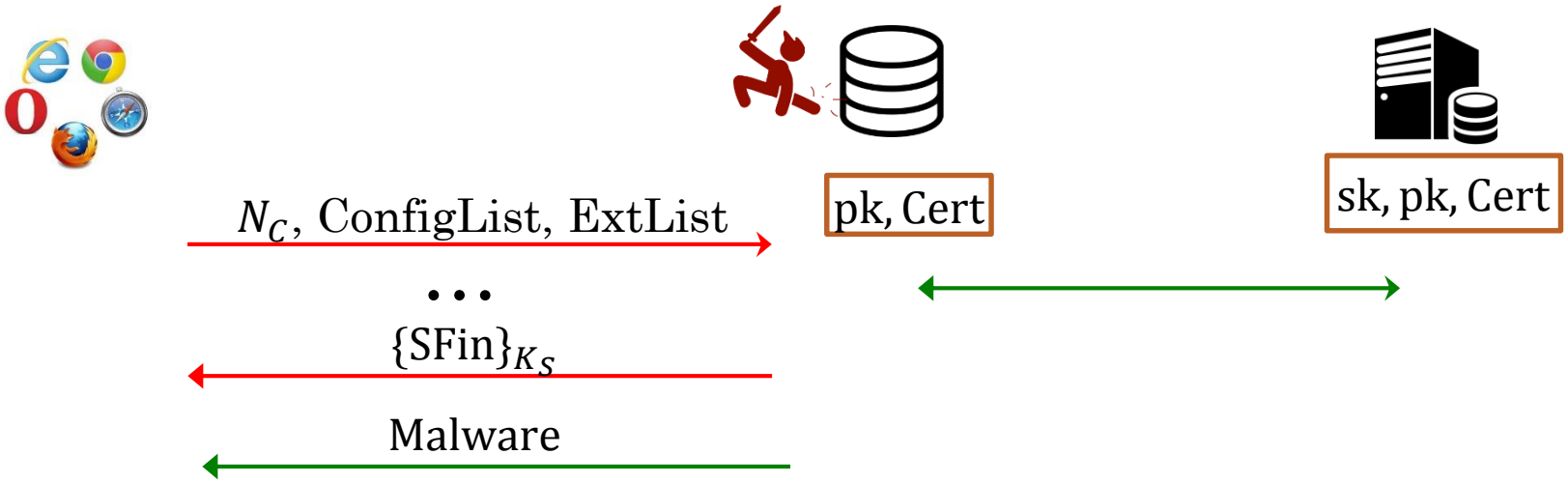
← pmk

Get msk, (K_C, K_S), check CFin
$\text{SFin} = \text{HMAC}_{msk}(H[N_C \ldots \text{CFin}])$

← $\{\text{SFin}\}_{K_S}$

A problem of Accountability

- Any malicious behaviour of CDN is on behalf of server
  - … and the server doesn't even know it!



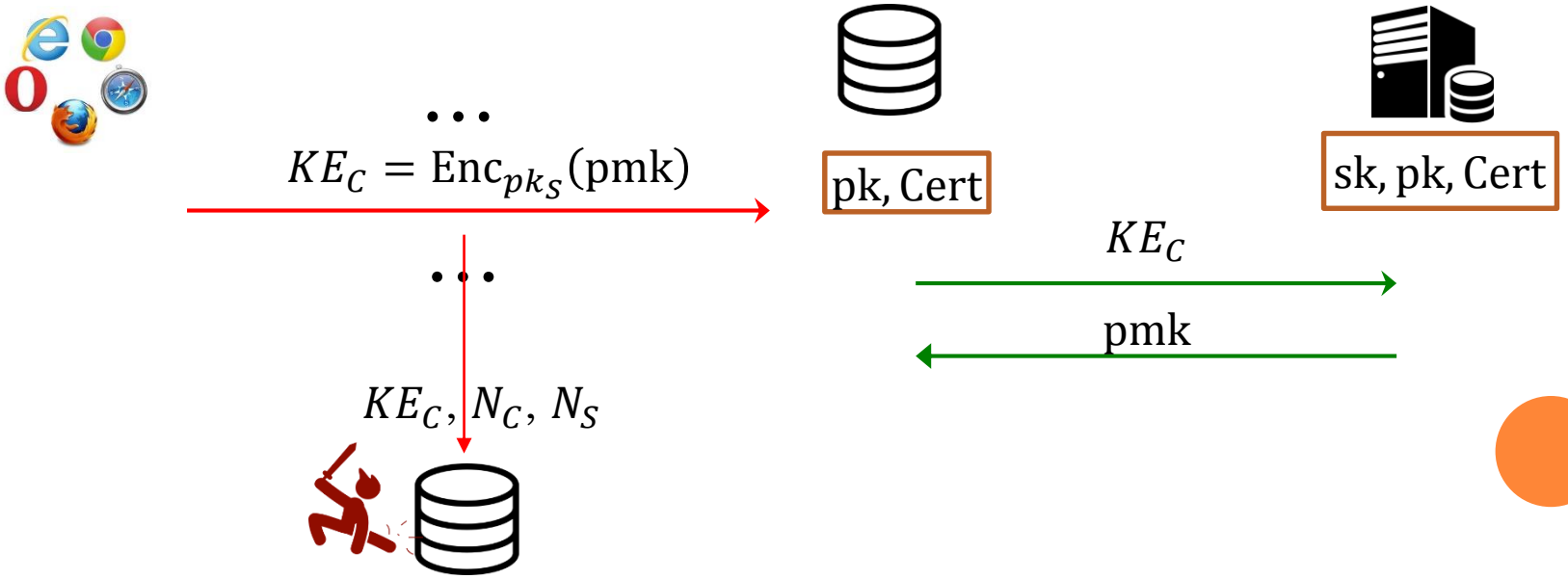
A server knowing the key could, however, monitor CDN behaviour!

A Big Security Risk

Bad, but normal

- Malicious CDN compromises all its sessions
  - Unfortunately it can also compromise any other session that is being run, anywhere on Earth

Perfect means of mass surveillance!

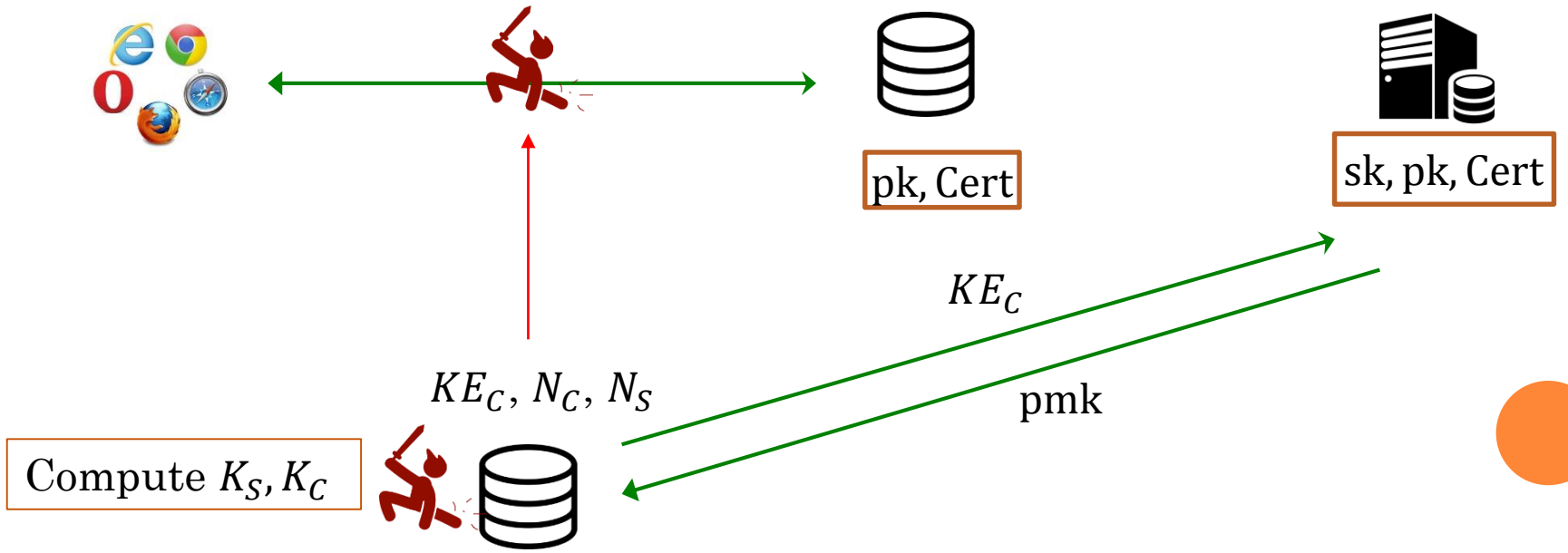


A Big Security Risk

- Malicious CDN compromises all its sessions
  - Unfortunately it can also compromise any other session that is being run, anywhere on Earth

Bad, but normal

Perfect means of mass surveillance!

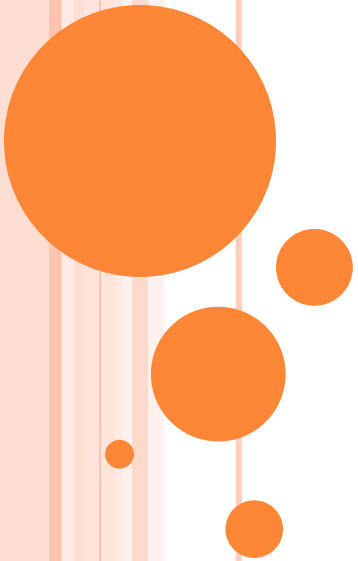


Other risks

- A single CDN has many clients:
  - Same entity holds session keys for many sessions
  - Thus, CDNs become a perfect tool of mass surveillance
  - Forward secrecy may depend on more than just one key

- In addition:
  - Mismatching nonces can cause authentication problems
  - Proxying over TLS 1.2 achieves at most TLS 1.2 security
    - Which is not much
  - No immediate construction of Keyless TLS 1.3
    - We do present one in our work

Fixing Keyless SSL



Proxied AKE infrastructure

- Three-party system:
  - Client, server, Middleware (MW)
  - Server owns contents $\omega_1, \ldots \omega_n$
    - Each ω_i associated with $(sk_i^S, pk_i^S, \mathrm{Cert}_i^S)$
  - MW agrees on contract with Server such that:
    - MW can later cache ω_i and send it to clients
    - MW has its own credentials (sk_{MW}, pk_{MW})
    - For each contracted ω_i, MW gets $(pk_i^{S,MW}, \mathrm{Cert}_i^{S,MW})$, maybe $sk_i^{S,MW}$

sk_{MW}, pk_{MW}

$\omega_j, pk_j^{S,MW}, \mathrm{Cert}_j^{S,MW}$

$sk_j^{S,MW}$

$\omega_1, \ldots \omega_n$

$sk_i^S, pk_i^S, \mathrm{Cert}_i^S \quad \forall i$

Secure Proxied AKE

- Composition of two 2-party channels (C-MW & MW-S)
  - C-MW is always unilaterally authenticated
  - MW-S is always mutually authenticated
  - C-S (direct) is always unilaterally authenticated

- We defined four security notions:
  - Authentication
  - Channel security

  } Adapted from 2-party case

  - Accountability: if MW impersonates S, then S knows key
  - Content soundness: MW cannot deliver uncontracted content

Main technical difficulty: session partnering

Partnering and Security

- Protocol is executed by parties
  - Each execution is a party instance
  - Party instances execute protocol sessions, which have sid's
  - Each party instance keeps track of:
    - Session ID sid – e.g. randomness and values used in key-computation
    - Partner ID pid

      For CDN, pid could be server, while partner is MW

    - Computed session key set K
    - Some more technical stuff (reveal bit, channel bit, etc.)

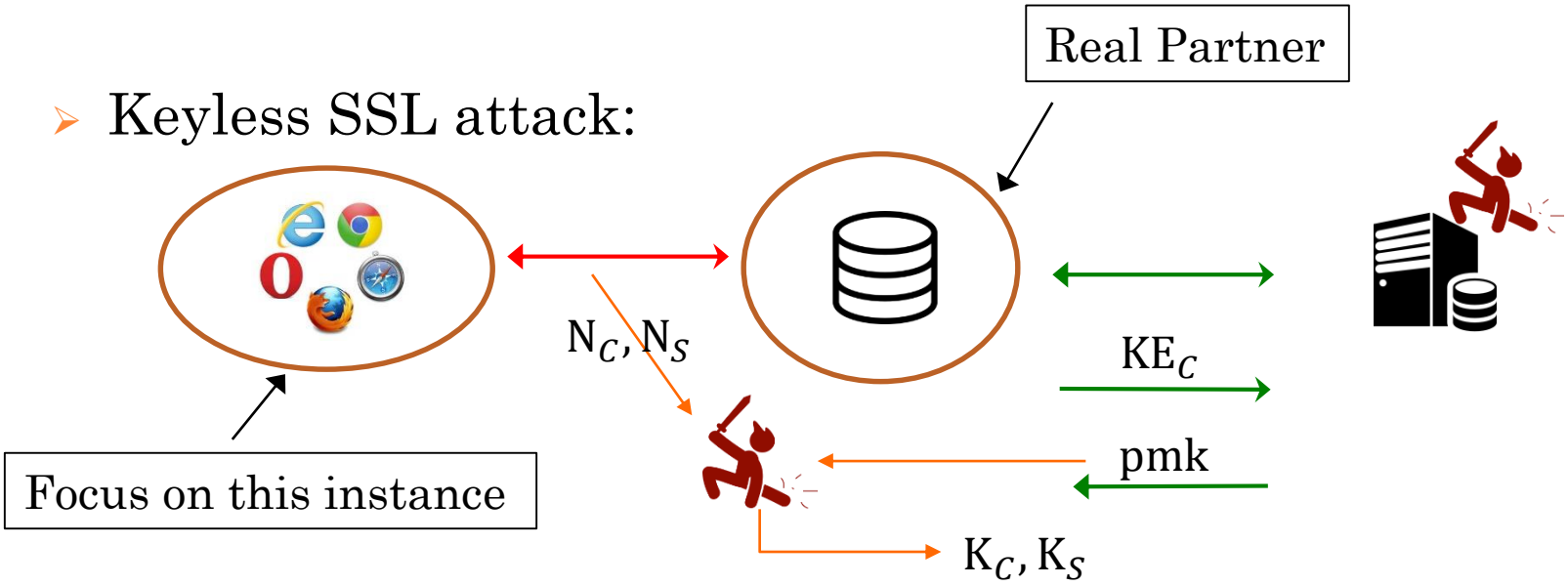
- 2-Partnering: 2 instances are partnered if they share sid's

  Partnering defines which sessions can be secured

Partnering and Security (cont'd)

- 2-Party authentication and channel security:
  - Only guaranteed if:
    - Party not corrupted
    - Partner (pid) not corrupted
    - Session key (of sid) not revealed
    - Session key of partner not revealed

- Keyless SSL attack:



3-Partnering

- 2 cases:
  - Client is aware of MW (essentially 2-partnering)
  - Client is unaware of MW (CDN/Keyless SSL)

- If client is unaware of MW:
  - If MW needs S (Keyless SSL): four instances partnered:
    - Client instance, MW1 instance, MW2 instance, S instance
  - If MW is autonomous (like in CDN): 2 instances partnered:
    - Client + MW1 instance
    - Partnering extends on 3 parties though (corrupting S is bad)

- Using 3-partnering this way allows us to re-use 2-party security definitions for auth. + secure channel

Accountability & Content Soundness

- Accountability
  - CDNs allow MW to impersonate S, with S's accord
  - It is in S's interest not to care beyond that
  - However, client has no way of distinguishing MW & S
  - Solution:
    - Either make client aware of the MW
    - Or make sure MW cannot hurt client (by auditing secure channel)

- Content Soundness
  - MW only allowed to know some contents (by contact)
  - Later, MW will contact S and ask to cache contents
  - S must make sure only allowed contents are sent
    - Currently not done: most S's just check "legitimate source port"

KEYLESS SSL CHANNEL INSECURITY

N_C, ConfigList, ExtList

pk, Cert

sk, pk, Cert

N_S, Config, Ext

$\text{KE}_S = (pk_S, \text{Cert}_S)$

Choose pmk
Set: $msk = \text{HMAC}_{pmk}(N_C, N_S)$
$KE_C = \text{Enc}_{pk_S}(\text{pmk})$
$(K_C, K_S) = \text{HMAC}_{msk}(N_C, N_S)$
$\text{CFin} = \text{HMAC}_{msk}(H[N_C \ldots KE_C])$

Out of context data

$\text{KE}_C, \{\text{CFin}\}_{K_C}$

KE_C

pmk

Get msk, (K_C, K_S), check CFin
$\text{SFin} = \text{HMAC}_{msk}(H[N_C \ldots \text{CFin}])$

$\{\text{SFin}\}_{K_S}$

KEYLESS TLS 1.2 V1

N_C, ConfigList, ExtList

pk, Cert

sk, pk, Cert

N_S, Config, Ext

$\text{KE}_S = (pk_S, \text{Cert}_S)$

Choose pmk
Set: $msk = \text{HMAC}_{pmk}(N_C, N_S)$
$KE_C = \text{Enc}_{pk_S}(\text{pmk})$
$(K_C, K_S) = \text{HMAC}_{msk}(N_C, N_S)$
$\text{CFin} = \text{HMAC}_{msk}(H[N_C \ldots KE_C])$

$\text{KE}_C, \{\text{CFin}\}_{K_C}$

$[N_C, \ldots, \{\text{CFin}\}_{K_C}]$

Check validity

pmk

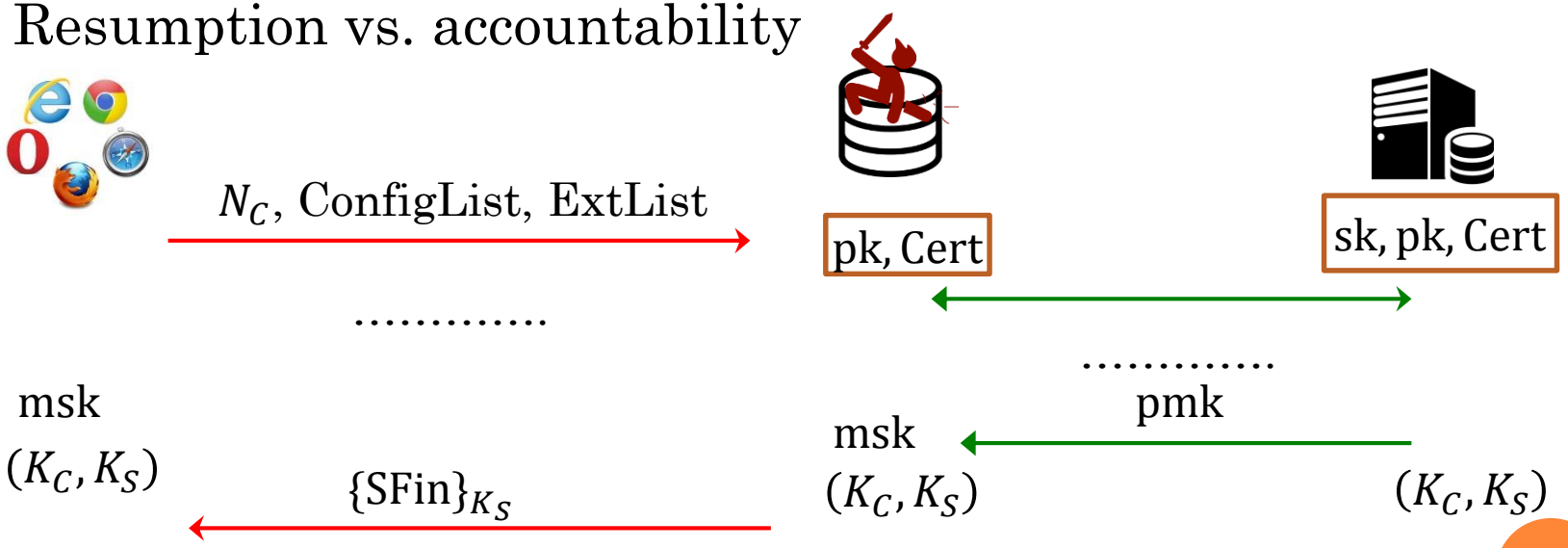
Get msk, (K_C, K_S), check CFin
$\text{SFin} = \text{HMAC}_{msk}(H[N_C \ldots \text{CFin}])$

$\{\text{SFin}\}_{K_S}$

Introducing Session Resumption

- Shorter handshake allows computation of session keys related to a full handshake
  - Given *msk*, just need new tuple of nonces

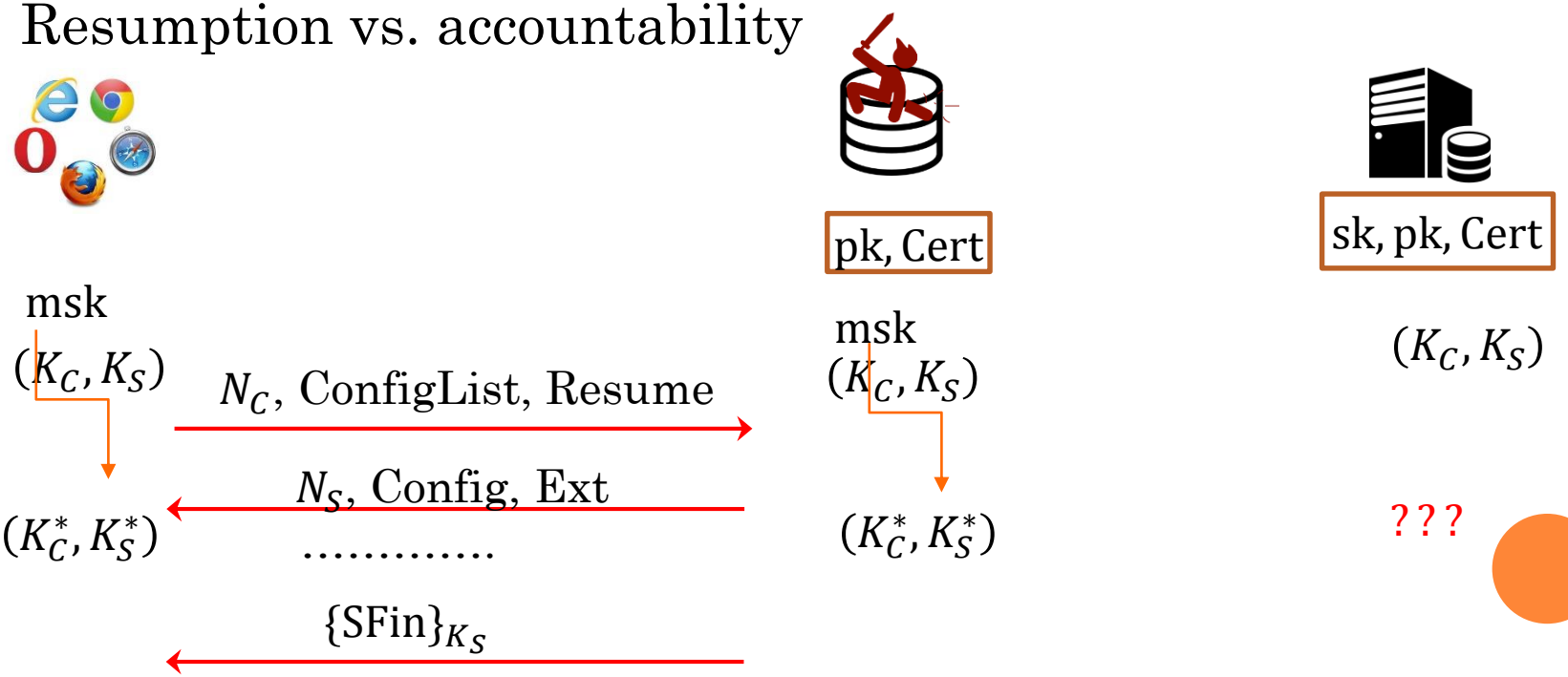
- Resumption vs. accountability



Introducing Session Resumption

- Shorter handshake allows computation of session keys related to a full handshake
  - Given *msk*, just need new tuple of nonces

- Resumption vs. accountability



KEYLESS TLS 1.2 V2

N_C, ConfigList, ExtList

N_S, Config, Ext

$\text{KE}_S = (pk_S, \text{Cert}_S)$

pk, Cert

sk, pk, Cert

Choose pmk
Set: $msk = \text{HMAC}_{pmk}(N_C, N_S)$
$KE_C = \text{Enc}_{pk_S}(\text{pmk})$
$(K_C, K_S) = \text{HMAC}_{msk}(N_C, N_S)$
$\text{CFin} = \text{HMAC}_{msk}(H[N_C \ldots KE_C])$

MW chooses all parameters

Cross-protocol, Auth. attacks

$\text{KE}_C, \{\text{CFin}\}_{K_C}$

$[N_C, \ldots, \{\text{CFin}\}_{K_C}]$

Check validity

(K_C, K_S, SFin)

Get msk, (K_C, K_S), check CFin
$\text{SFin} = \text{HMAC}_{msk}(H[N_C \ldots \text{CFin}])$

$\{\text{SFin}\}_{K_S}$

KEYLESS TLS 1.2

pk, Cert

Request

N_S

sk, pk, Cert

N_C, ConfigList, ExtList

N_S, Config, Ext

$\text{KE}_S = (pk_S, \text{Cert}_S)$

Choose pmk
Set: $msk = \text{HMAC}_{pmk}(N_C, N_S)$
$KE_C = \text{Enc}_{pk_S}(\text{pmk})$
$(K_C, K_S) = \text{HMAC}_{msk}(N_C, N_S)$
$\text{CFin} = \text{HMAC}_{msk}(H[N_C \ldots KE_C])$

Unavoidable

S runs entire protocol!

KE_C, $\{\text{CFin}\}_{K_C}$

$[N_C, \ldots, \{\text{CFin}\}_{K_C}]$

Check validity

(K_C, K_S, SFin)

Get msk, (K_C, K_S), check CFin
$\text{SFin} = \text{HMAC}_{msk}(H[N_C \ldots \text{CFin}])$

$\{\text{SFin}\}_{K_S}$

Security of Keyless TLS 1.2

- Authentication and Secure Channel:
  - 3-partnering: no relevant session/secret keys given away
  - N_S generated honestly
  - KE_C given in full context, allows S (honest) to prevent attacks by MW corruption
- Accountability:
  - MW forwards Nonces, encrypted CFin, and KE_C
    - S can verify these are correct nonces from CFin and KE_C
  - S sends directly the session keys and encrypted SFin
    - No session resumption
- Content soundness:
  - One certificate per content per MW for each S

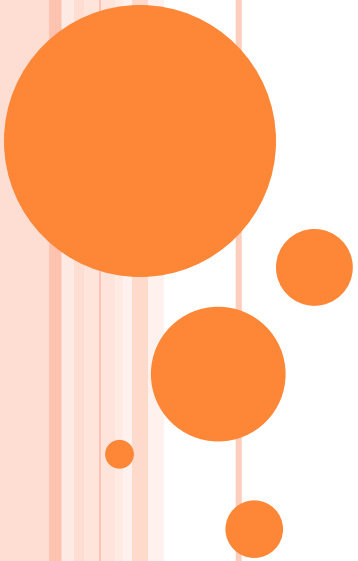
Some more results

- Original Keyless SSL in DHE mode:
  - Problematic for original Keyless SSL (cross-protocol attack)
  - No accountability, no content soundness
  - Unfortunately fixed Keyless TLS 1.2 DHE has same drawbacks as TLS 1.2 RSA (big PKI, lots of server-side computation)
- Keyless TLS 1.3:
  - Did not exist in original CloudFlare proposal
  - We propose a version that does not support resumption
  - … but it is more efficient (MW not simple spectator!)
  - Lighter PKI
- Some tradeoffs: accountability vs limited resumption

In Perspective

Proxying: pros and cons

- Proxying through CDNs:
  - Uses a cache-then-deliver strategy to improve efficiency in content delivery over HTTPS://
  - Provide such services transparently to clients
  - A single CDN can serve many content owners simultaneously
- … but unfortunately CDNs:
  - Were not designed with client-security and privacy in mind
  - Provide an ideal target for mass surveillance since a lot of information passes through a single CDN!
  - Do not allow clients to make informed decisions based on whether they communicate with a CDN or the server directly
- Our take: the client should know!



Thanks! Questions?