SSI Lecture 4
Public Key Infrastructure

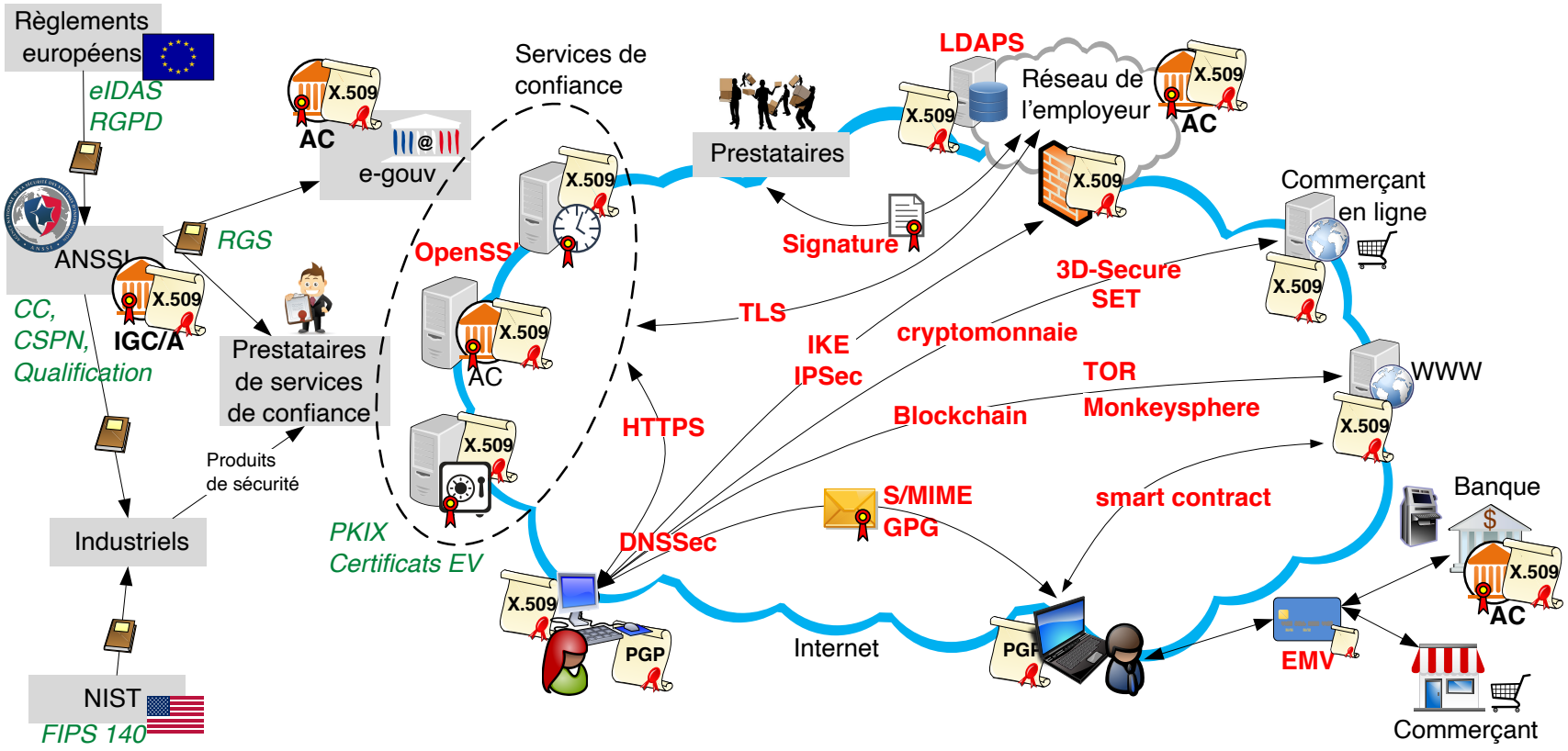
Pascal Lafourcade

2022-2023

Outline

Outline

Public Key Infrastructure



Applications

- IPSec
- SSL/TLS
- DNSSec
- TOR
- IKE
- HTTPS
- LDAPS
- OTR
- S/MIME
- SET
- EMV
- 3d-Secure
- Bitcoin

MIME to S/MIME

Multipurpose Internet Mail Extensions

- RFC822 authorize only text emails
- MIME allows several content types and multi-part messages

S/MIME : Secure/Multipurpose Internet Mail Extensions
is supported in several mail agents.

S/MIME Functionalities

- enveloped data: encrypted content and associated keys
- signed data: encoded message + signed digest
- compressed-data: compression of message
- signer-info: include information like used algorithms, date of signature, certificats ...

It gives confidentiality, integrity of data but also non-repudiation and authentication.

S/MIME Cryptographic Primitives

- hash functions: SHA-1 & MD5
- digital signatures: DSS & RSA
- session key encryption: ElGamal & RSA
- message encryption: Triple-DES, RC2/40 and others

uses X.509 v3 certificates, with several well-known CA (like Verisign)
If the client is not compatible, the data appears as an attached file.

PKI : Public Key Infrastructure

- Using public keys
- Establish a symmetrical session key
- Trust
- Certificates
- Certification Authority
- Chain of Trust

PKI

Problem: how to agree securely on a symmetric key?

- Face-to-face key exchange O(n^2) keys
- Key exchange via a trusted third party (TTP) Kerberos 5

Idea : Public-key encryption solves the problem of key exchange.
How to ensure the authenticity of other people's public keys?

- Face-to-face key exchange O(n)
- Key exchange via a trusted third party (TTP)

Other solution is : Key certificates.

How to share a key?

Several solutions :

- Protocol by Diffie-Hellman (Attack Man-In-the-Middle)
- Kerberos with Trusted party and symmetric keys
- Public Key Infrastructure (PKI)

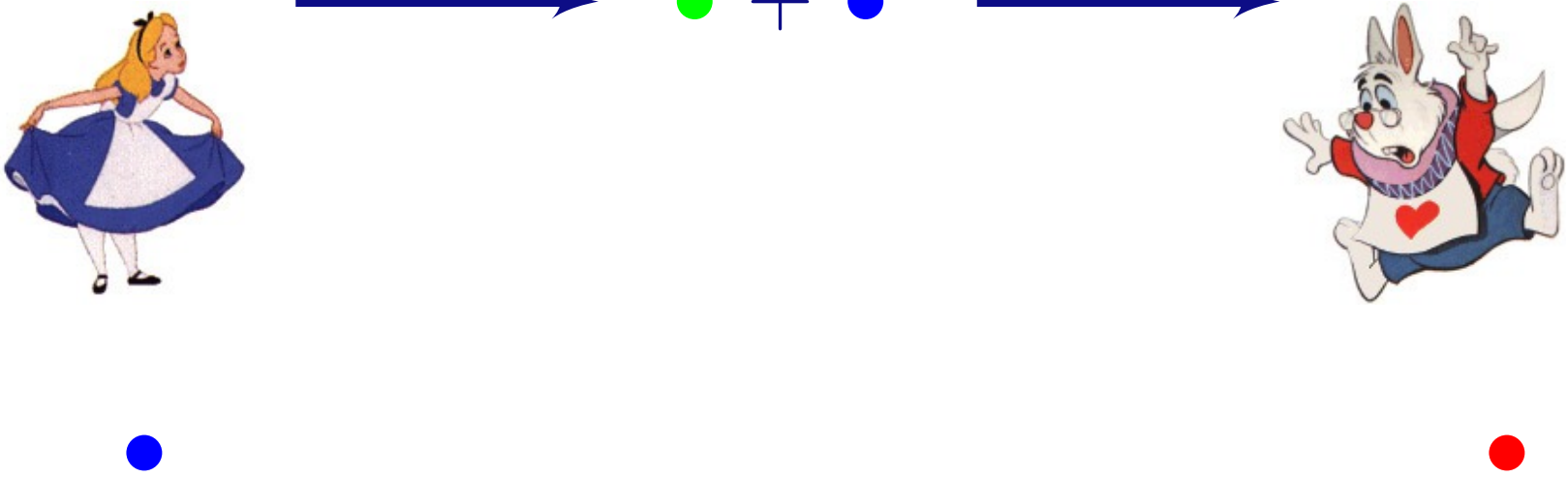
Outline

Diffie Hellman (1976)

- is public

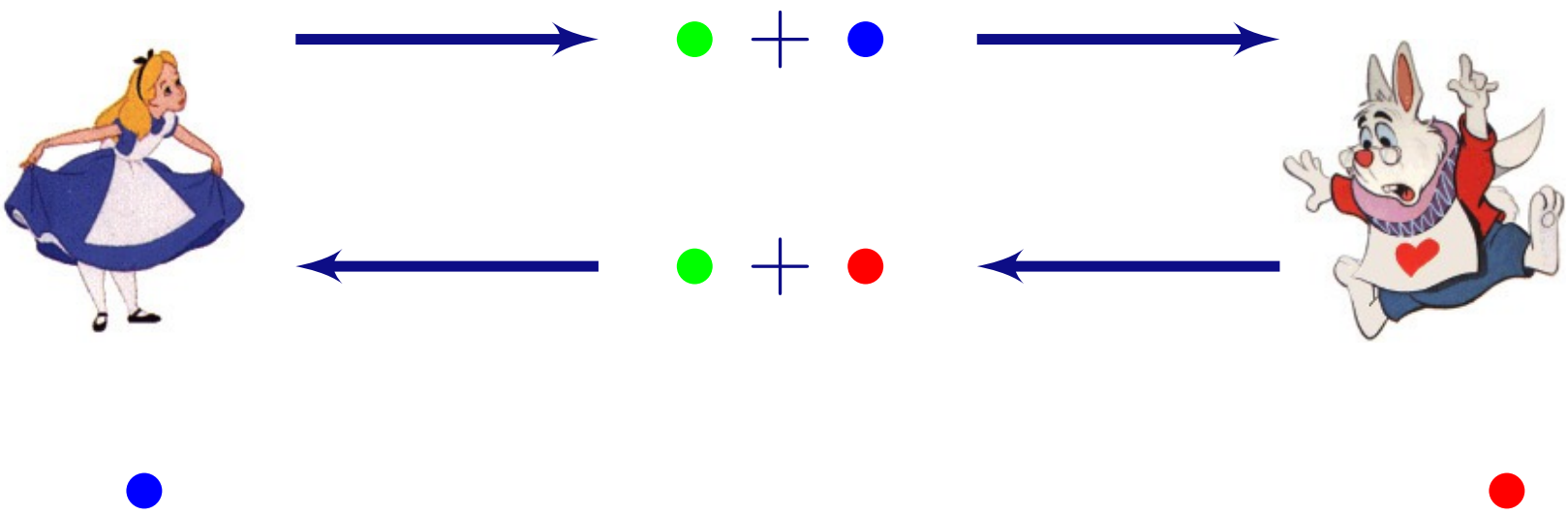
Diffie Hellman (1976)

- is public

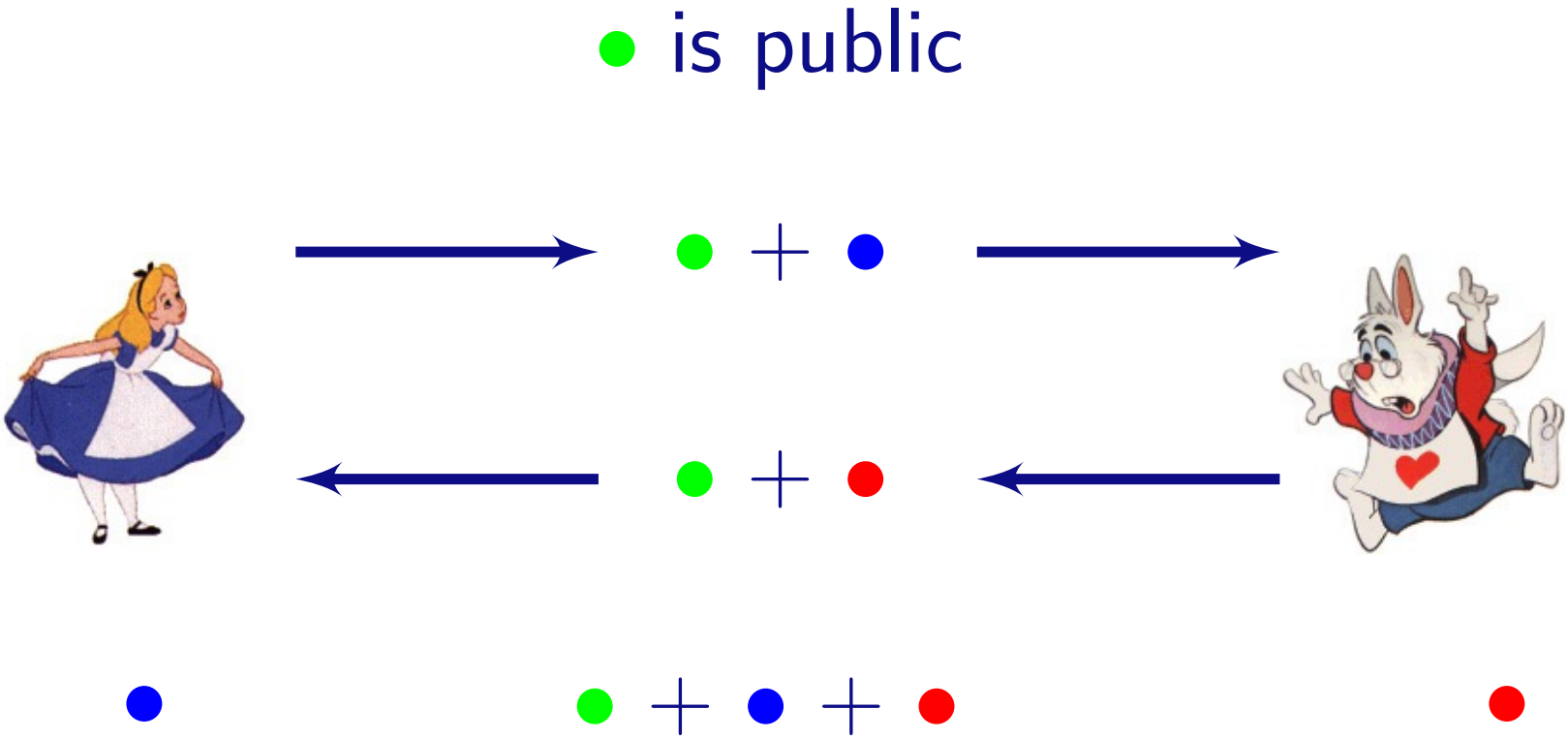


Diffie Hellman (1976)

- is public



Diffie Hellman (1976)



Diffie Hellman (1976)

● is public

● + ●

● + ●

● + ● + ●

- ● $g =$ ●
- ● $a =$ ●
- ● $b =$ ●

$$(g^a)^b = g^{ab} = (g^b)^a$$

The Diffie-Hellman protocol with authentication

- A picks a and sends $A, g^a, Sig_A(g^a)$
- B picks b and sends $A, g^b, Sig_B(g^b)$

Verification of signatures
Shared key: $(g^a)^b = g^{ab} = (g^b)^a$
But authentication is not there!
Charlie can replay the message sent by Alice to succeed the protocol.

The Diffie-Hellman protocol vulnerable to UKS attack

UKS: Unknonw Key Share

- A picks a and sends to B A, g^a.
- B picks b and sends to A $A, g^b, Sig_B(g^b, g^a)$
- A sends to B $A, Sig_A(g^a, g^b)$

Verification of signature and shared key: $(g^a)^b = g^{ab} = (g^b)^a$

But!

Charlie can be in the middle and change Alice name A by C.
Charlie does not learn the key and cannot open the messages.
From B point of view all messages are comming from C. In case of a bank it might be problematic and credit a wrong bank account.

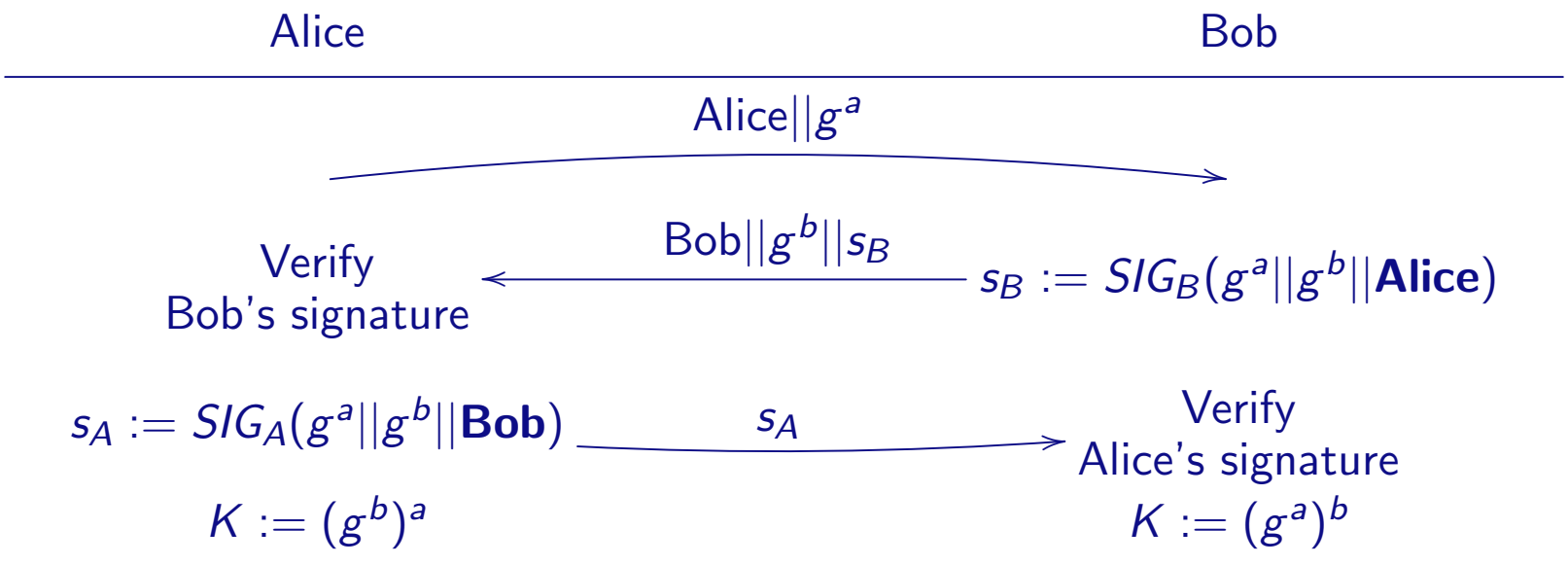
The Diffie-Hellman

To avoid UKS attack, we need to add the names in the signature !

- A picks a and sends to B A, g^a.
- B picks b and sends to A $A, g^b, Sig_B(A, B, g^b, g^a)$
- A sends to B $A, Sig_A(A, B, g^a, g^b)$

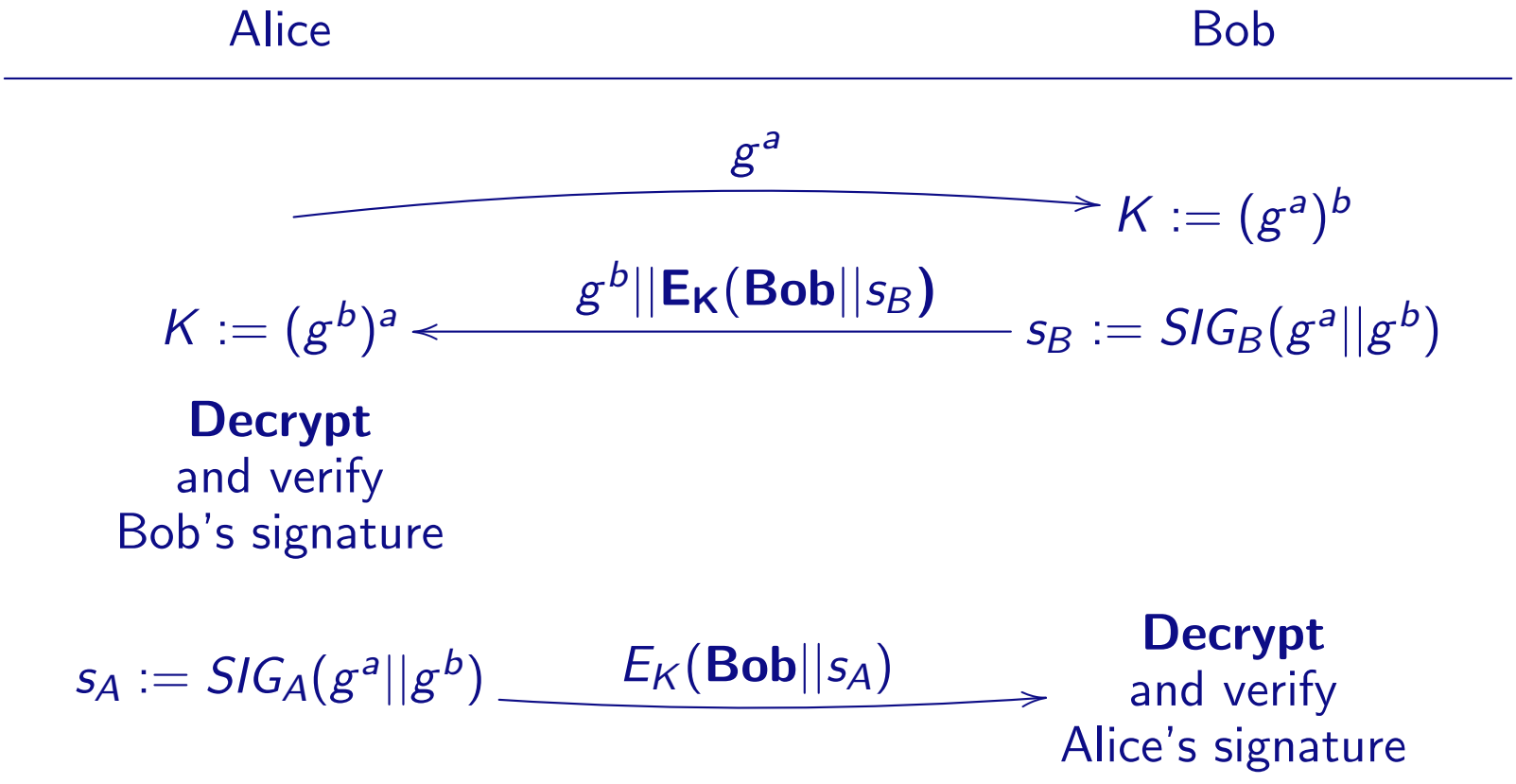
If Charlie changes the name, Alice will notice it and stop the communication.
Perfect Forward Secrecy: If Alice and Bob delete the ephemeral secret a and b then even if the secret key of Alice and Bob are compromised previous messages cannot be compromised.

Protocol ISO-9706

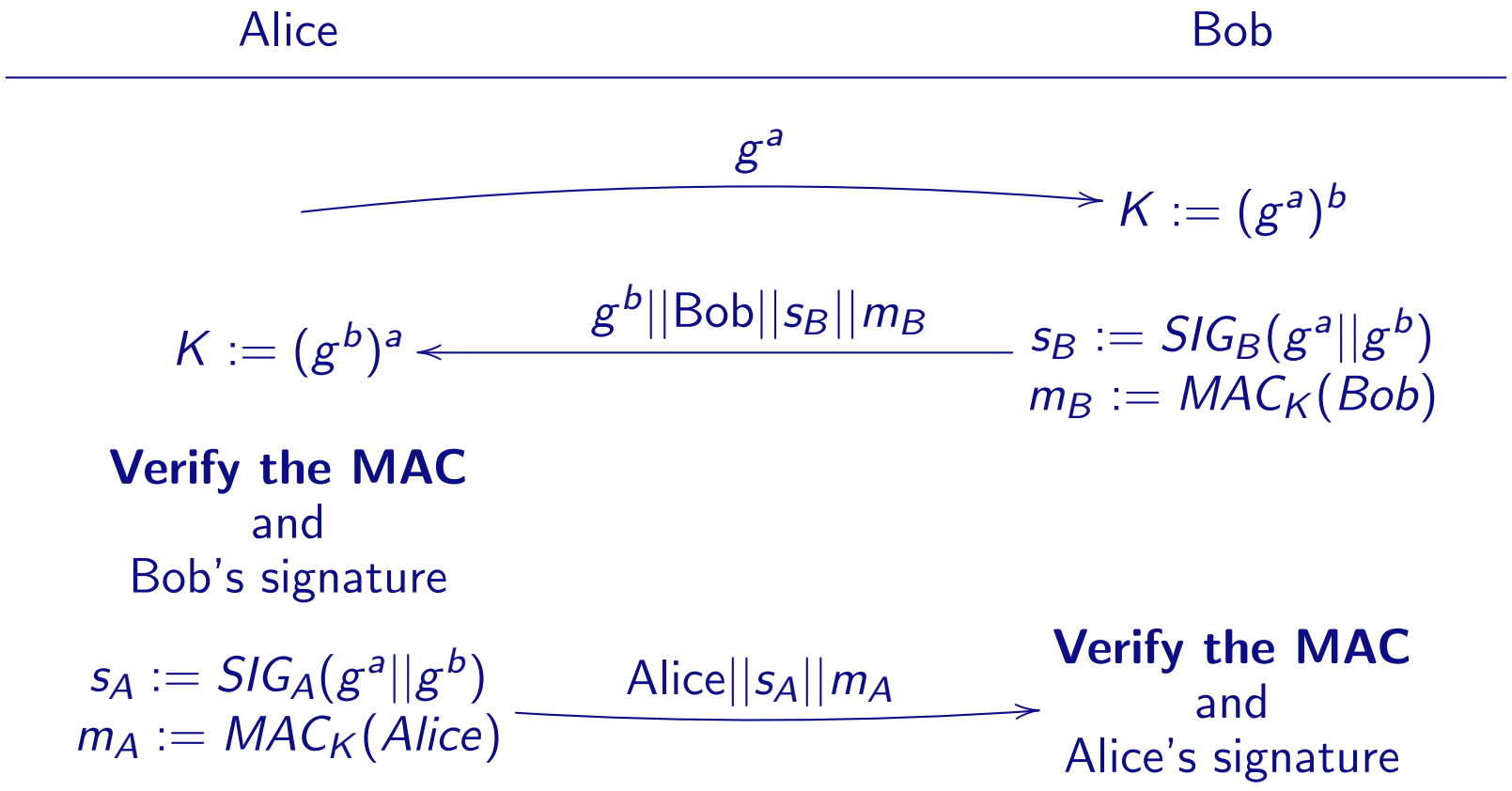


Do not protect the identity of the initiator

Protocol Station to station

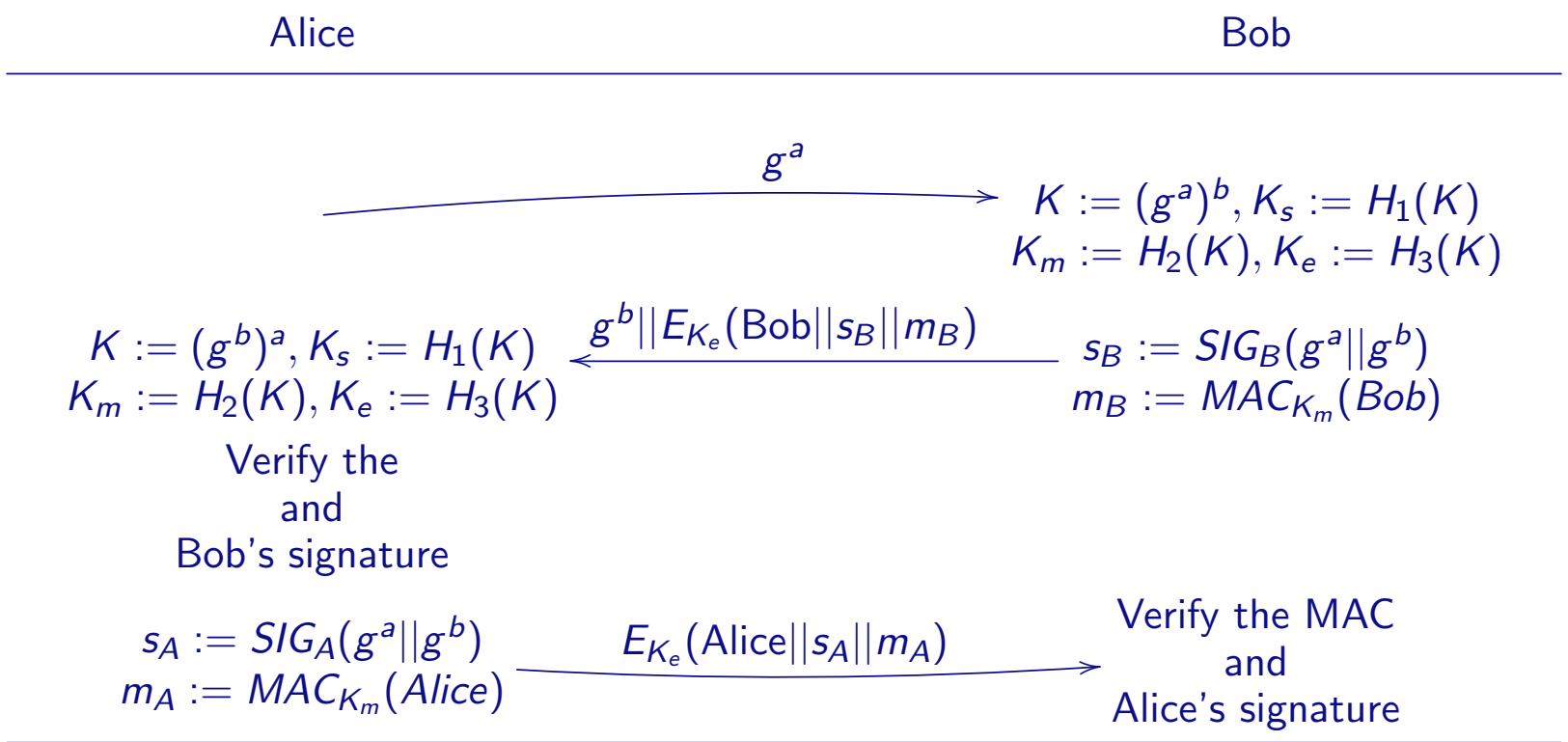


Protocol Sigma

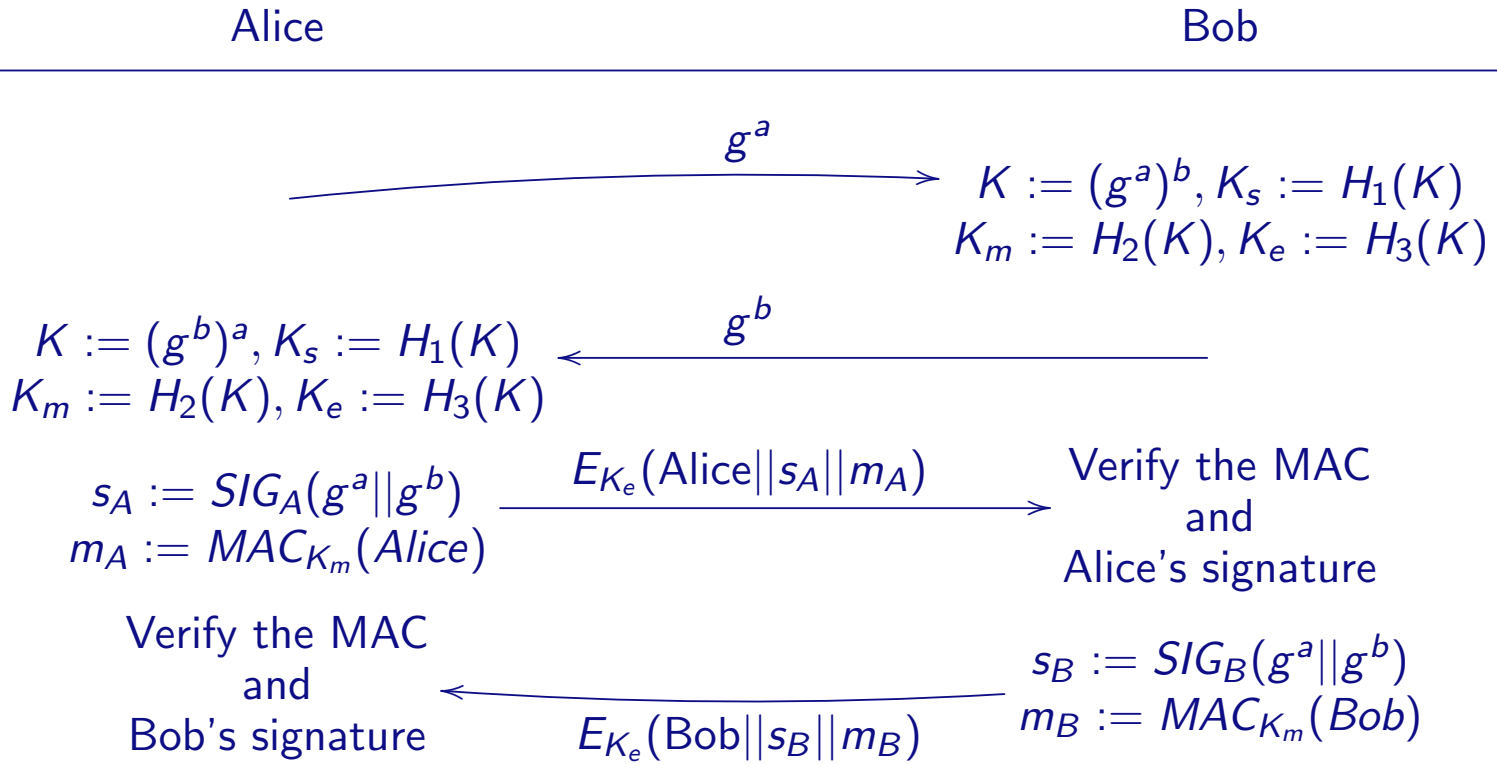


Alice and Bob have to give their identity. To avoid that we have SIGMA-I to preserve identity of the Initiator and SIGMA-R to preserve the identity of the Responder.

Protocol Sigma-I



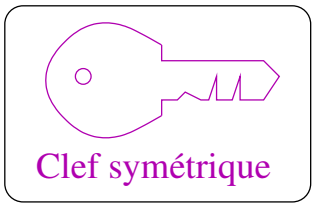
Protocol Sigma-R



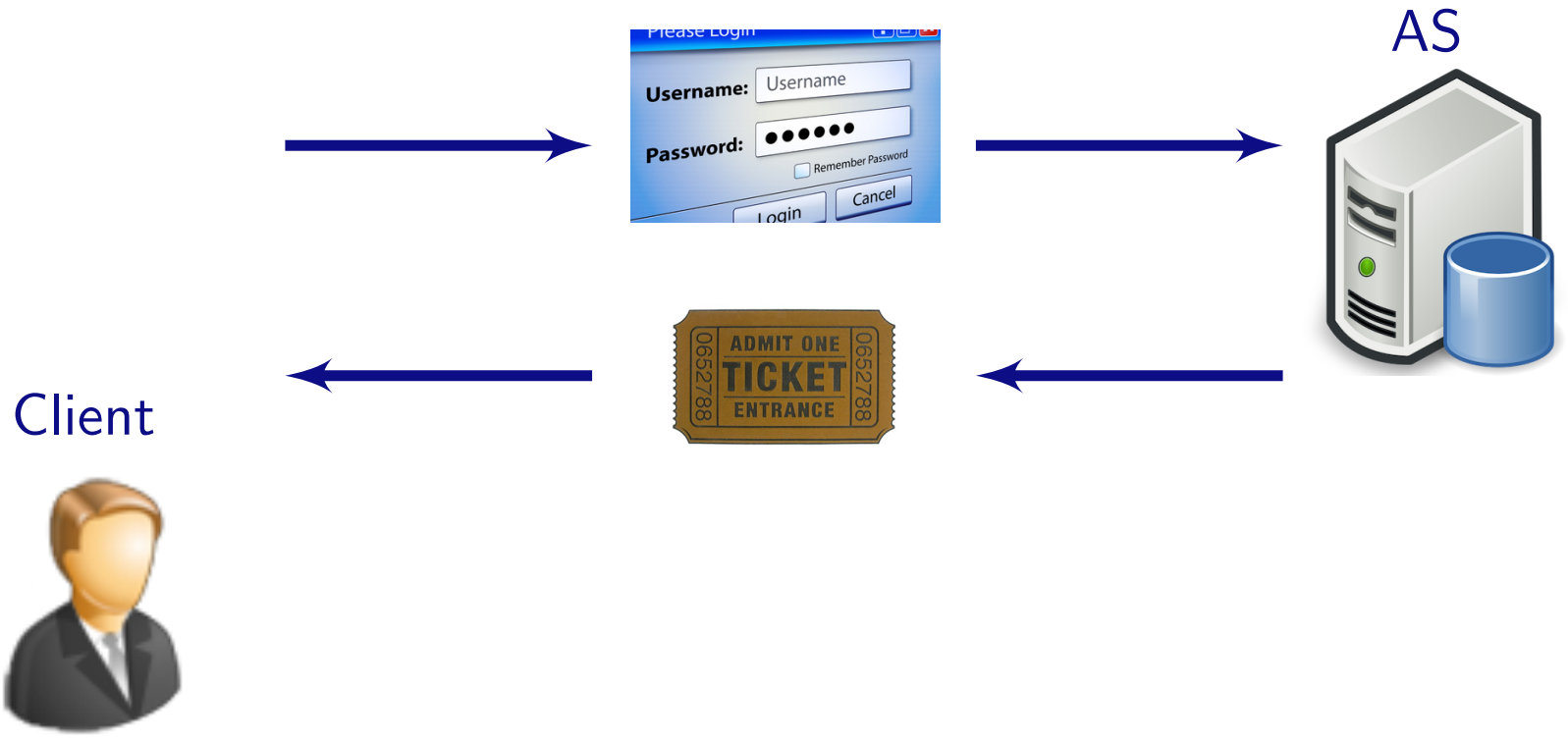
Outline

Kerberos V5

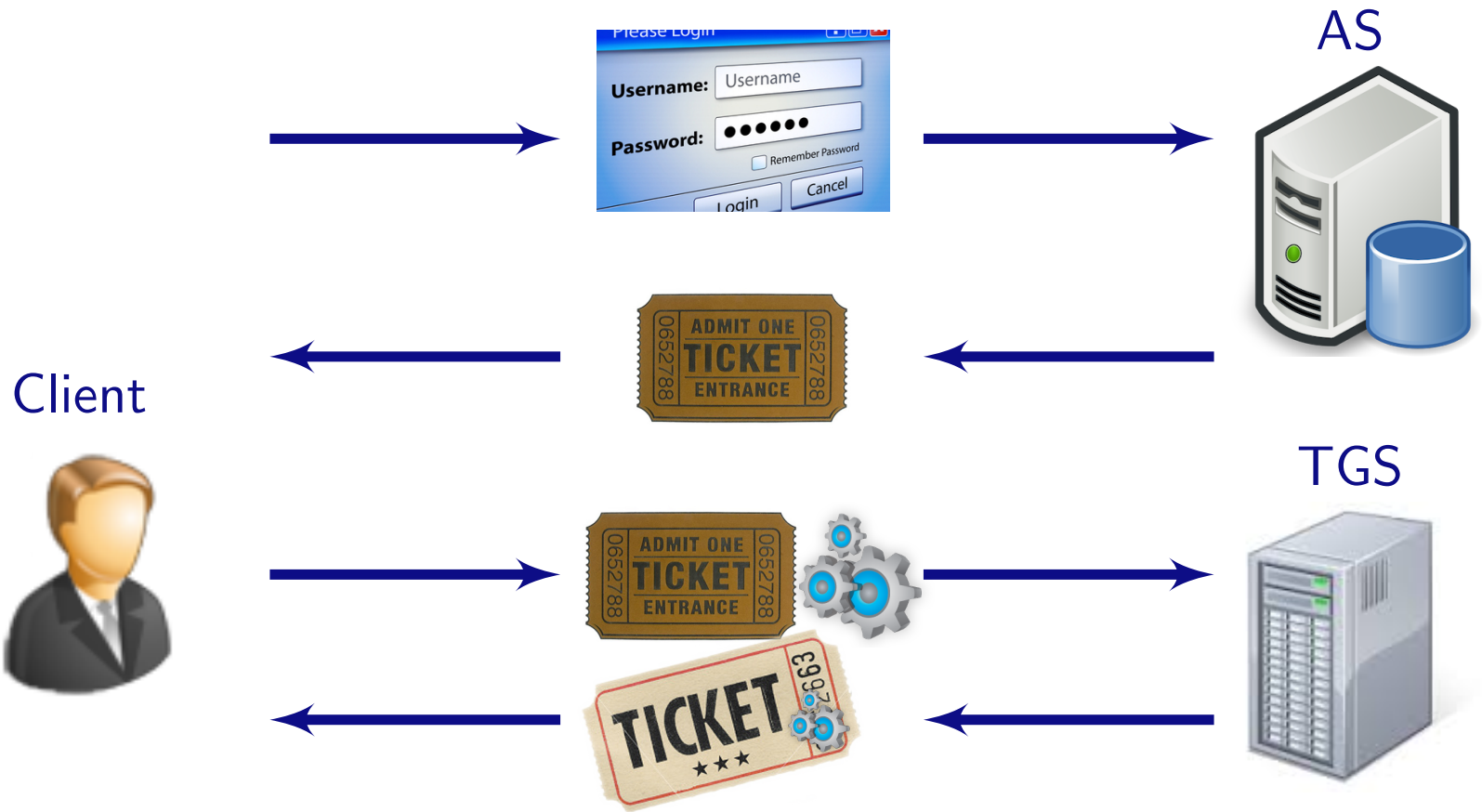
- a trusted third party: AS (Authentication Server)
- symmetrical encryption (private keys)
- tickets: TGS (Ticket Granting Service)
- passwords



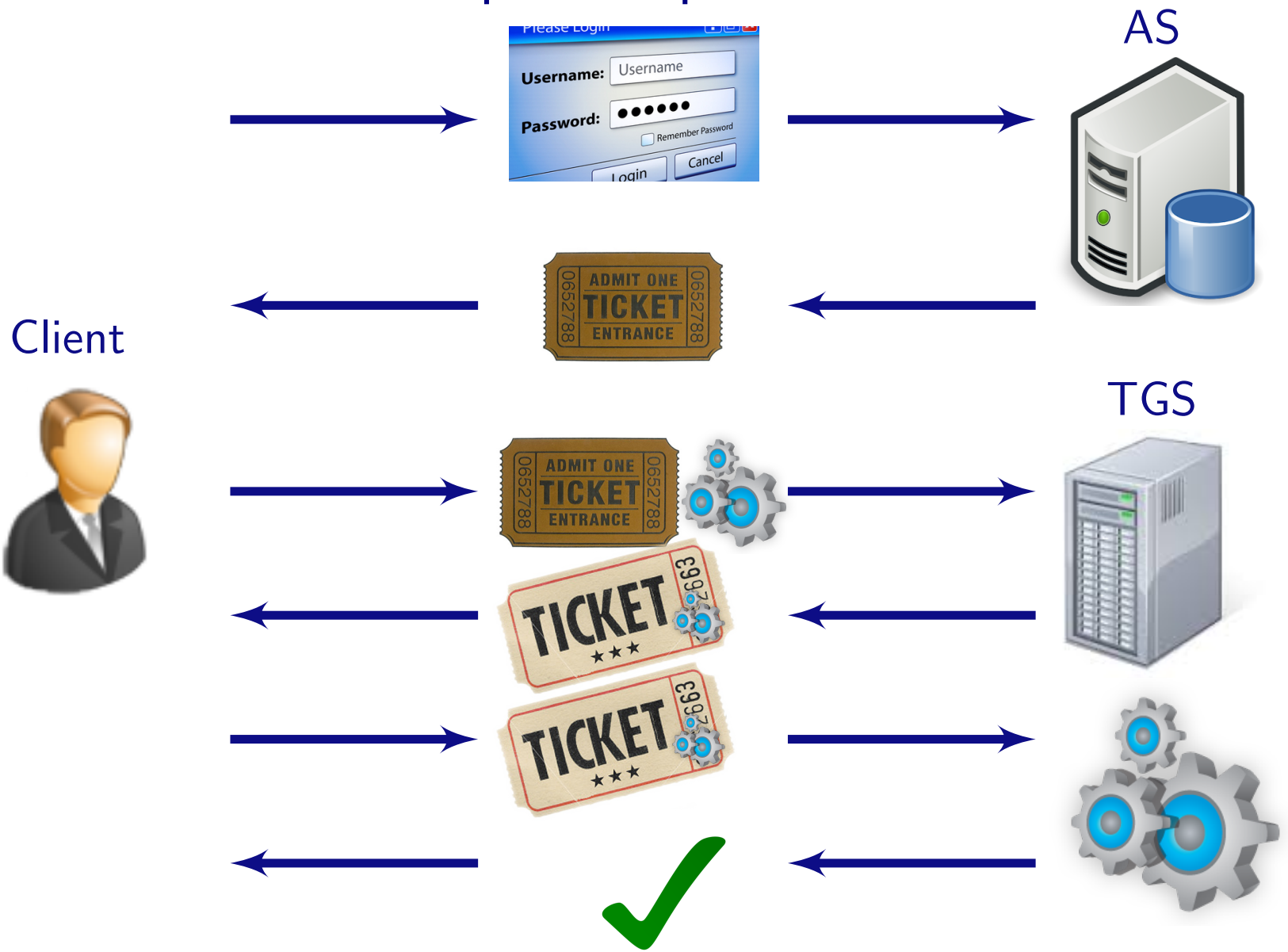
Kerberos V5: Principe in 3 phases



Kerberos V5: Principe in 3 phases



Kerberos V5: Principe in 3 phases



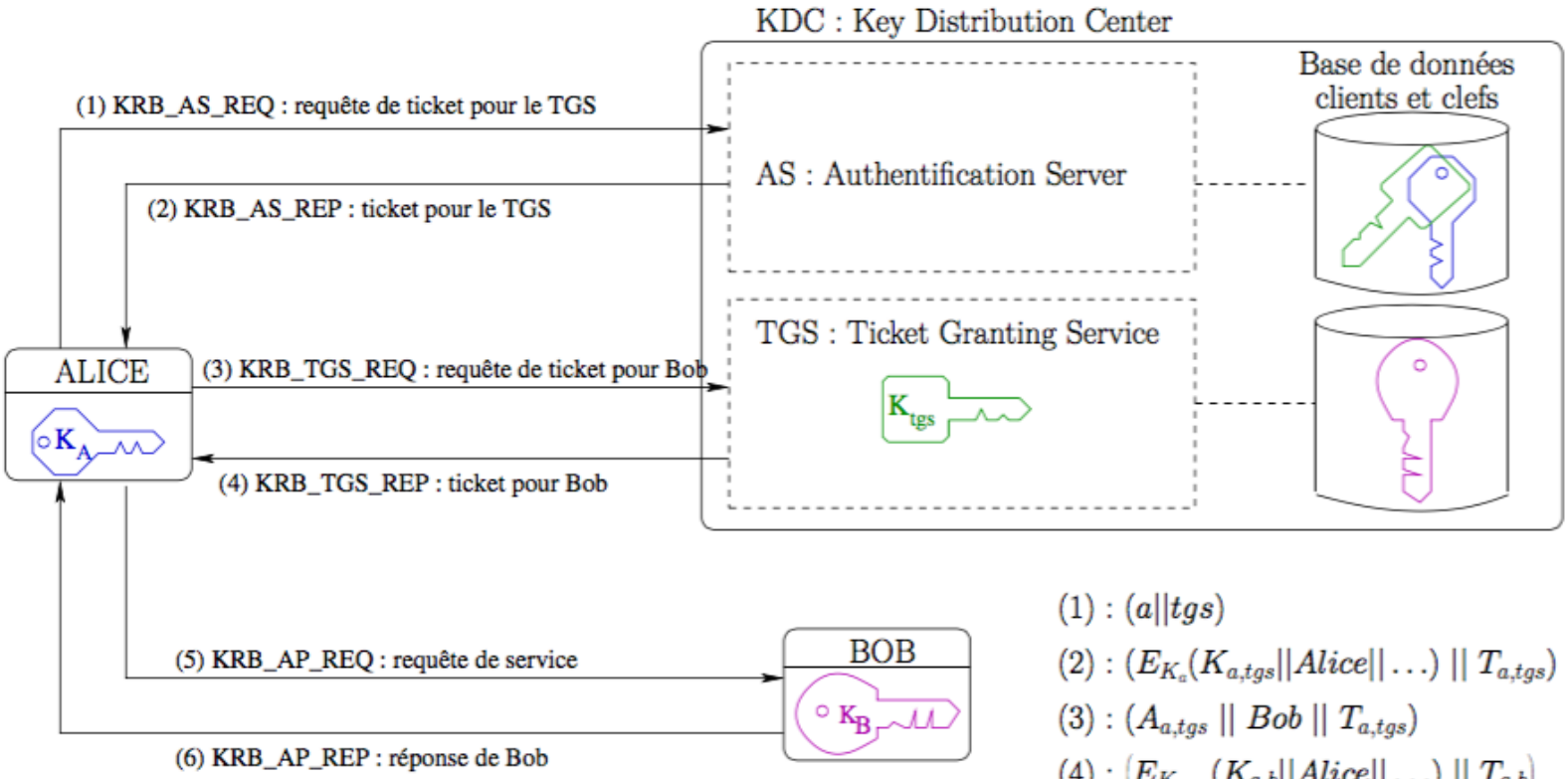
Kerberos V5 Notations

One ticket for Alice for the service s
$T_{a,s} := (id_s \;||\; E_{K_s}(id_a||t||t_{end}||K_{a,s}))$

- Identity of Alice id_a ;
- Date of request t ;
- Date of the enf of the validity of ticket t_{end} ;
- Session key $K_{a,s}$

Authentifiant $A_{a,s}$ for Alice associated to service s :
$A_{a,s} := E_{K_{a,s}}(id_a||t)$
a for Alice and tgs for Ticket Grant Service

Kerberos V5



(1) : $(a||tgs)$

(2) : $(E_{K_a}(K_{a,tgs}||Alice||\ldots) \,||\, T_{a,tgs})$

(3) : $(A_{a,tgs} \,||\, Bob \,||\, T_{a,tgs})$

(4) : $\left(E_{K_{a,tgs}}(K_{a,b}||Alice||\ldots) \,||\, T_{a,b}\right)$

(5) : $(A_{a,b} \,||\, T_{a,b})$

(6) : $\left(E_{K_{a,b}}(t+1)\right)$

```
EncKDCRepPart   ::= SEQUENCE {
            key               [0] EncryptionKey,
            last-req          [1] LastReq,
            nonce             [2] UInt32,
            key-expiration    [3] KerberosTime OPTIONAL,
            flags             [4] TicketFlags,
            authtime          [5] KerberosTime,
            starttime         [6] KerberosTime OPTIONAL,
            endtime           [7] KerberosTime,
            renew-till        [8] KerberosTime OPTIONAL,
            srealm            [9] Realm,
            sname             [10] PrincipalName,
            caddr             [11] HostAddresses OPTIONAL
    }
```

Outline

The concept of key certficate

Main idea

- Alice trusts Bob and knows his public key
- Bob has signed asserting that Carol's key is K
- Then Alice may be willing to believe that Carol's key is K.

Definition

A key certificate is an assertion that a certain key belongs to a certain entity, which is digitally signed by an entity (usually a different one).

Two Kinds of PKI

Hierarchical PKI

- Certificate Authoirities are different of users
- X.509 (PKIX)

Non-Hierarchical PKI

- Each user manages his own trust network
- Pretty Good privacy (PGP) and P2P based PKI

Others : SDSI, SPKI

Example "https" for gmail

- Gmail sends to your browser its public key and a certificate signed by a certificate authority "Thawte Consulting (Pty) Ltd." to prove that this key really is gmail's key.
- Your browser will verify Thawte's signature on gmail's key using the public key of this reputable key certificate authority, stored in your browser.
- Hence your browseer trust Gmail.

Trust chains

Example

A1 has signed asserting that A2's key is K2
A2 has signed asserting that A3's key is K3
. . .
A18 has signed asserting that A19's key is K19
A19 has signed asserting that A20's key is K20
A20 has signed asserting that B's key is K

- If I know A1's key, and I trust A1, A2,..., A20, then I am willing to believe that B's key is K.
- A1 states that A2's key is K2. So, if A2 signs an assertion X, then A1 states that A2 states X. Thus, in the situation above, A1 states that A2 states that A3 states ... that A19 states that A20 states that B's key is K.

Definition PKI

PKI is an infrastructure build of certificates and servers to create, manage and publish certificate to allow autenticity certified by an authority.

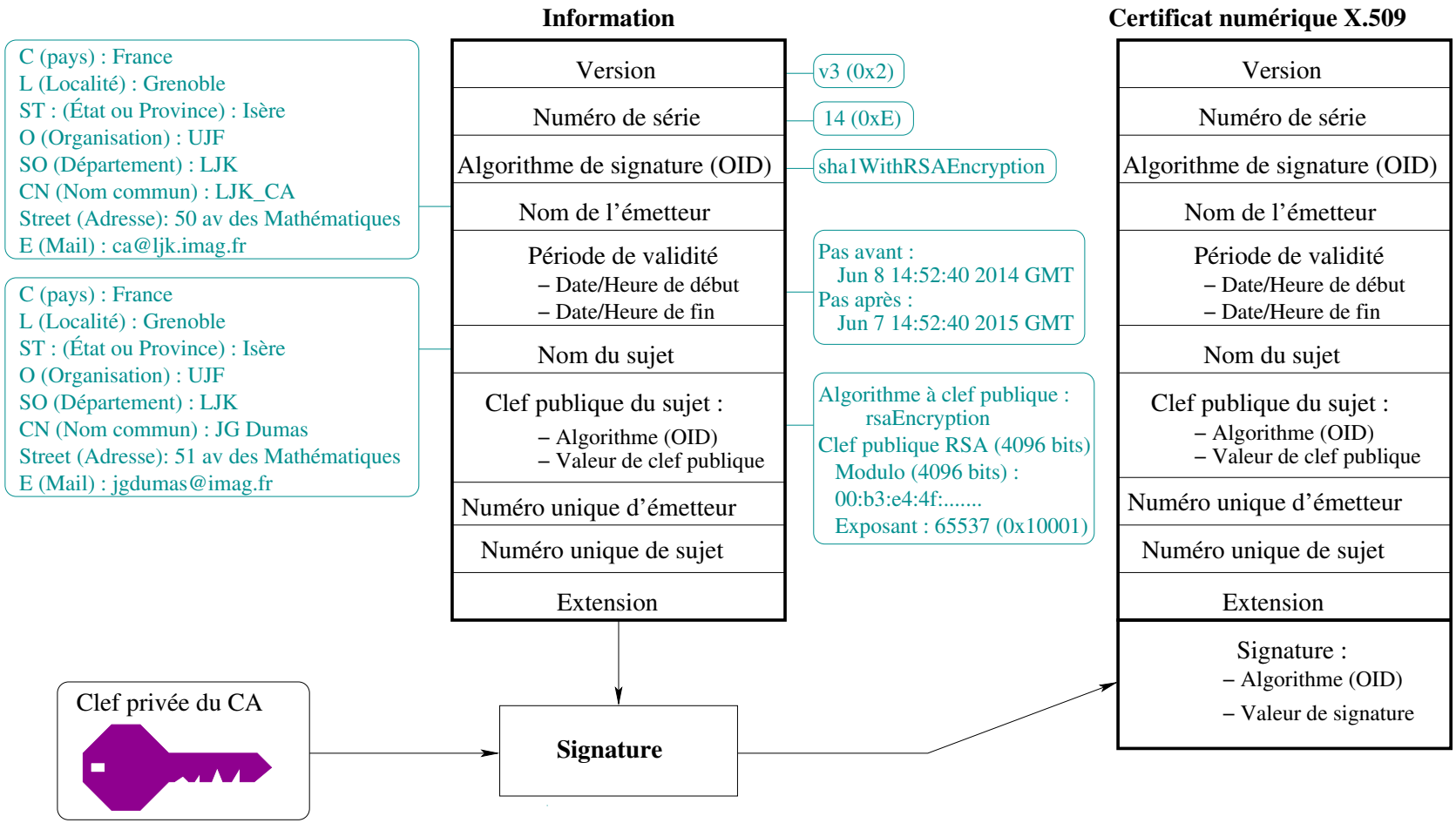
X.509 certificates used by SSL/TLS, SET, S/MIME, IPSec ...

X.509 components:

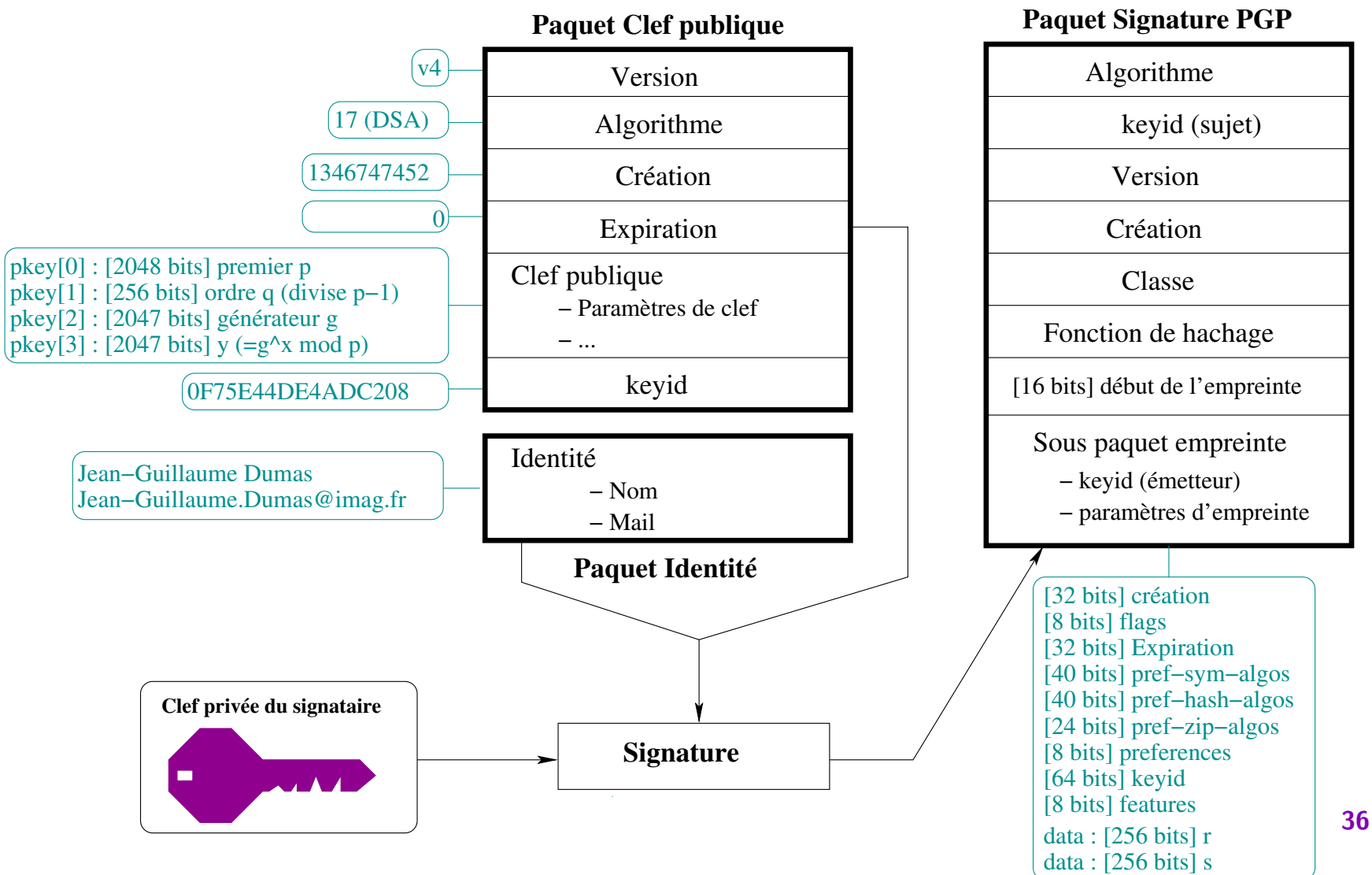
- Version
- Serial number
- Signature algorithm identifier
- Issuer name
- Period of validity
- Subject name
- Subject public-key and algorithm
- Issuer unique identifier
- Subject unique identifier
- Extensions
- Signature

X.509 certificates are typically issued by a certificate authority (CA). Anyone can be a CA, but not all CA's are trusted by everyone

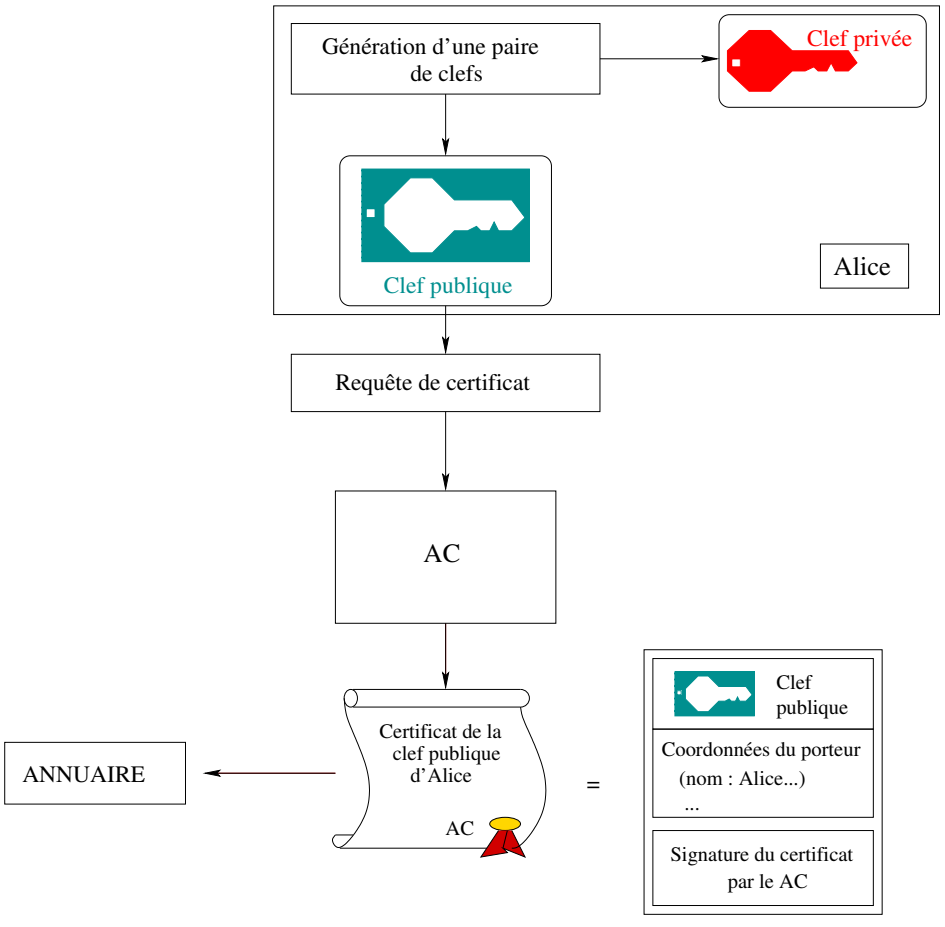
Certificat X509



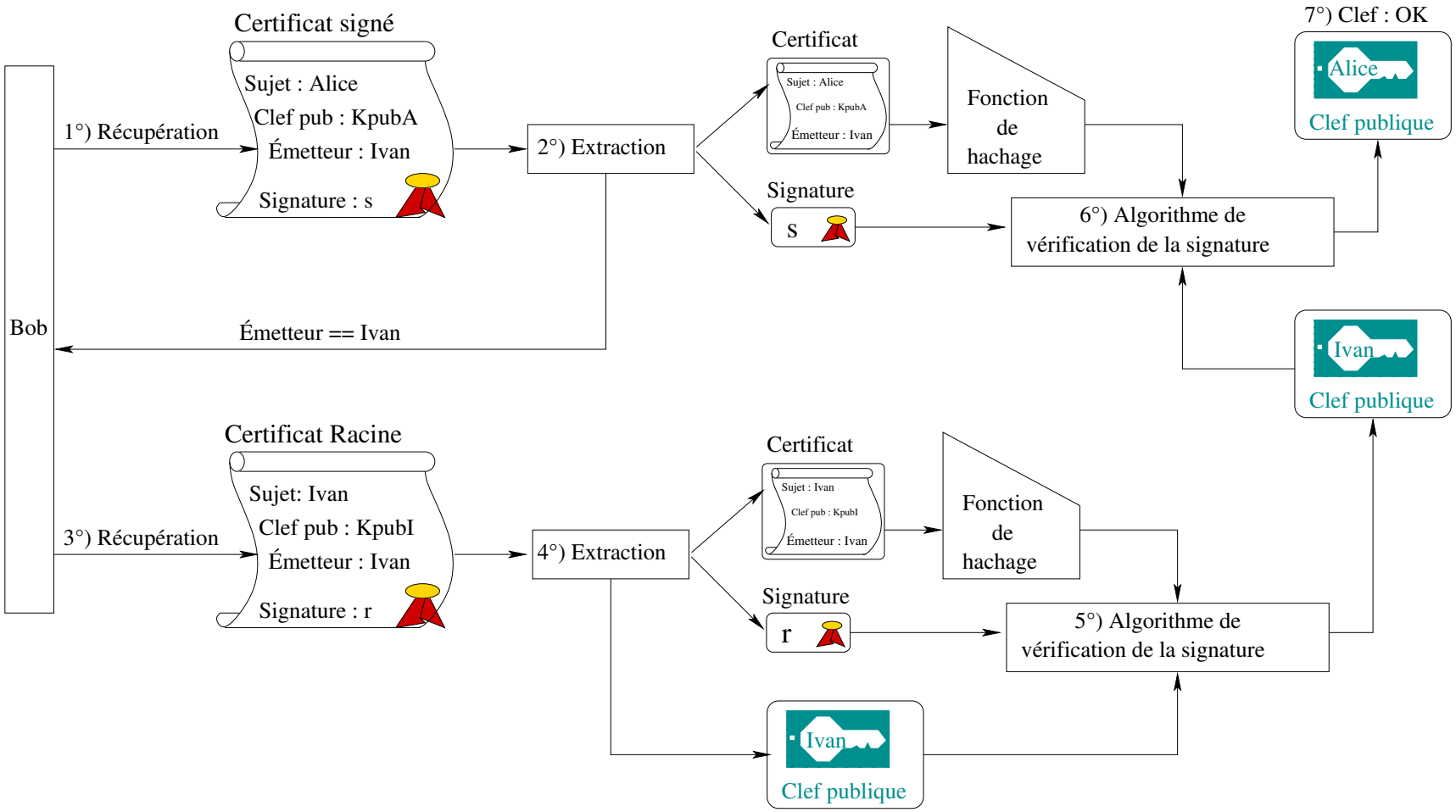
Certificat PGP



Public Key Infrastructure (PKI)

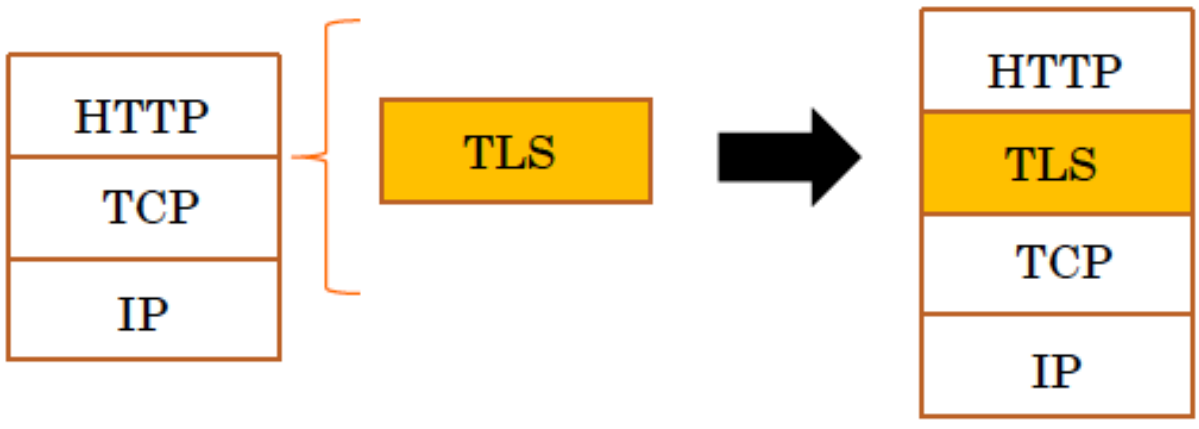


Verification



Outline

SSL/TLS Protocol



Protocole SSL/TLS

Properties

- Server *authentication*
- Data *confidentiality*
- *Integrity* and *authentication* of data
- Client *authentication* (optionnal)

- HTTP (80) $\rightarrow$ HTTP*S* (443)
- IMAP (143) $\rightarrow$ IMAP*S* (993)
- POP3 (110) $\rightarrow$ POP3*S* (995)

History

- *SSL* (*Secure Sockets Layer*), designed by Netscape
  - SSL 1.0 (1994) : theoretical protocol, never used
  - SSL 2.0 (1995-2011) : first version used
  - SSL 3.0 (1996-. . . , RFC 6101)
- *TLS* (*Transport Layer Security*), designed by IETF (*Internet Engineering Task Force*)
  - TLS 1.0 (1999-. . . , RFC 2246)
  - TLS 1.1 (2006-. . . , RFC 4346)
  - TLS 1.2 (2008-. . . , RFC 5246)
  - TLS 1.3 (2017 -. . . ,)
- *Compatibility* of HTTPS servers in (source : SSL Pulse)

	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3
2016	8,4 %	26,8 %	98,2 %	71,9 %	74,2 %	0 %
2020	0,8 %	4,6 %	53,5 %	61,7 %	98,4 %	32,8 %

Cryptography in TLS 1.2

Server [client] *Authentification*

- Public Key : RSA, DSS, ECDSA
- Secret key : PSK (*Pre-Shared Key*), SRP (*Secure Remote Password*)
- No authentication : ANON

Key Exchange

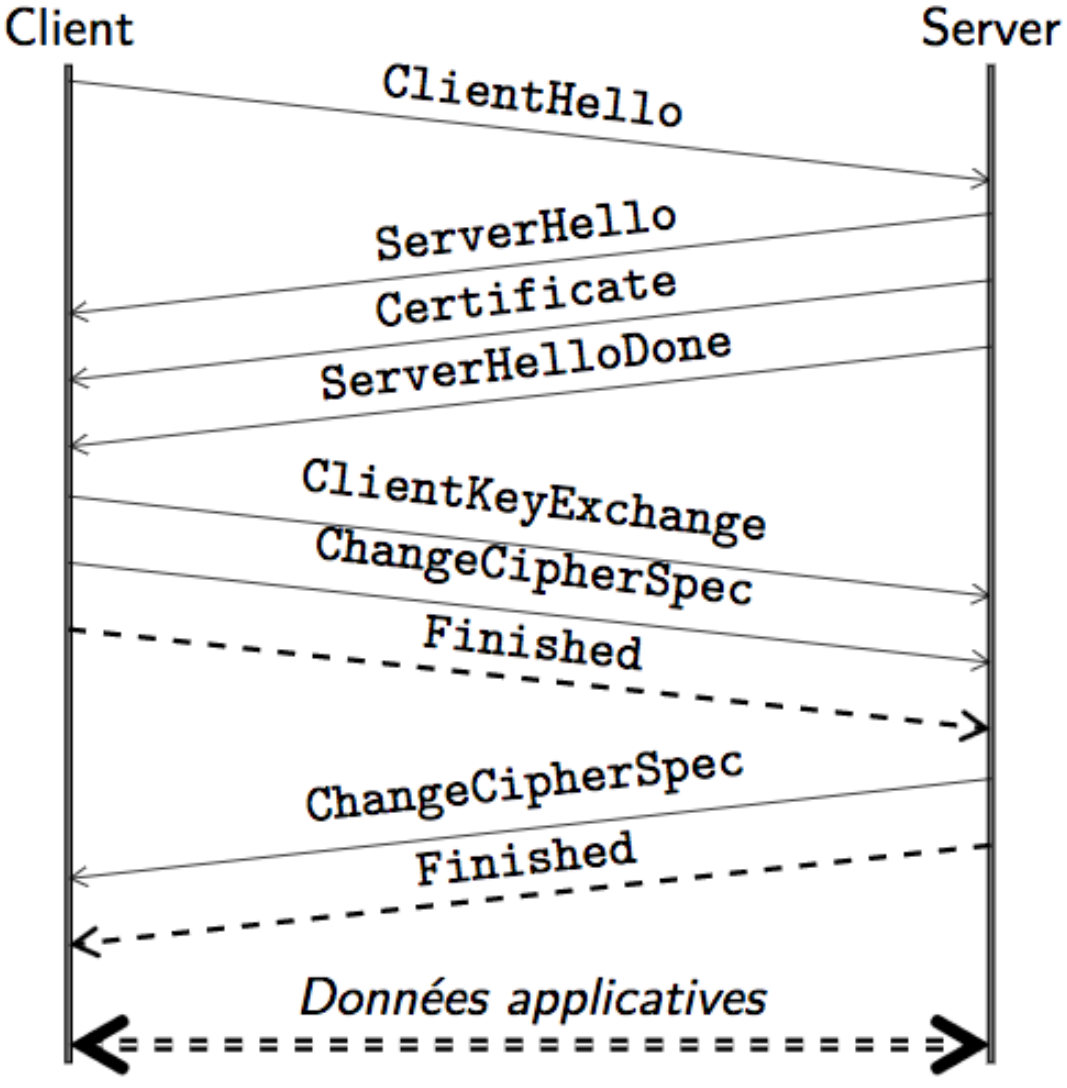
- Public Key (same on the server) : RSA
- Diffie-Hellman statique : DH, ECDH
- share secret : PSK, SRP
- Diffie-Hellman éphemeral (different secret keys each time) ; *forward secrecy*) : DHE, ECDHE

Cryptography in TLS 1.2

- *Encryption* :
  - Mode: AES-CBC, 3DES-CBC, DES-CBC, etc.
  - Stream cipher: RC4
  - No encryption: NULL
- *Integrity* and *authentication*:
  - HMAC : HMAC-MD5, HMAC-SHA1, HMAC-SHA256, etc.
  - AEAD: AES-CCM, AES-GCM, etc.

Example of *cipher suite* :
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

High level


Client
Server
ClientHello
ServerHello
Certificate
ServerHelloDone
ClientKeyExchange
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
Données applicatives

TLS 1.2

5 protocols

- *Handshake*, to establish a secure client/server connexion
- *Change Cipher Spec*, to use a new key
- *Alert* $\Rightarrow$ Warning ou Fatal
- *Application Data*
- *TLS Record* (Symmetric encryption and MAC)

Handshake SSL/TLS simple

1. *Client* $\rightarrow$ *Serveur*
   - `ClientHello` : version supported, *cipher suites*
2. *Serveur* $\rightarrow$ *Client*
   - `ServerHello` : version supported, *cipher suite* chosen
   - `Certificate` (opt.) : server public key in a certificat X.509
   - `ServerKeyExchange` (opt.) : Diffie-Hellman partial key
   - `ServerHelloDone`

Handshake SSL/TLS simple

3. *Client* $\rightarrow$ *Serveur*
   - `ClientKeyExchange` : *PreMasterKey* encrypted by server public key, or Diffie-Hellman partial key
   - `ChangeCipherSpec`
   - `Finished` : message encrypted and authenticated using a transcript of the *handshake*
4. *Serveur* $\rightarrow$ *Client* (If `Finished` of the client is valid)
   - `ChangeCipherSpec`
   - `Finished` : message encrypted and authenticated using a transcript of the *handshake*
5. If `Finished` of the server is valid, then connexion is established.

Public Key Infrastructure

Allow to trust a public key !

- Certificat : *signature of a public key* by a trusted third party.

Two possible infrastructures

- Hierarchic : *Certificate authorites* (SSL/TLS et X.509, EMV)
- Decentralised : *Web of Trust* (PGP, GnuPG)

Certification Authorities

- *Root CA* : certifying *public key of other CA*
- *Other CA* : certifying server keys

Chain of trust :

Root CA $\xrightarrow{\text{sign}}$ *CA 1* $\xrightarrow{\text{sign}}$ *CA 2* $\xrightarrow{\text{sign}}$ *Server*

Authentication : server sends 3 certificats

- Its own *public key*, *signed* by CA 2
- *Public key* of CA 2, *signed* by CA 1
- *Public Key* of CA 1, *signed* by Root CA

Verification : If the client knowns and trusts the public key of Root CA, then he can verify all certificats.

Certification Authorities

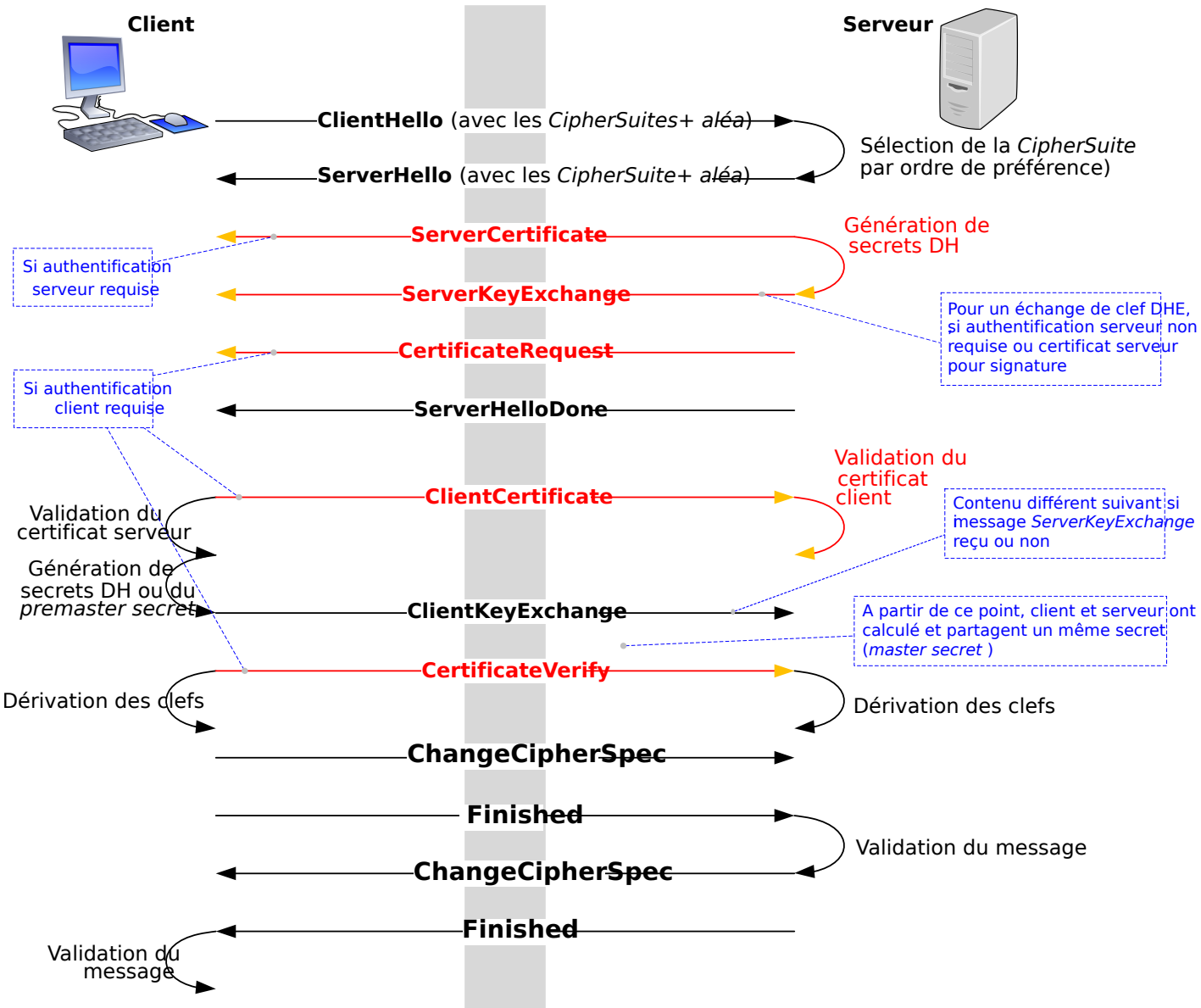
List of *Trusted Root CA* are in the browsers

- $>$ 80 *Root CA* in Firefox
- Need to trust the browser!

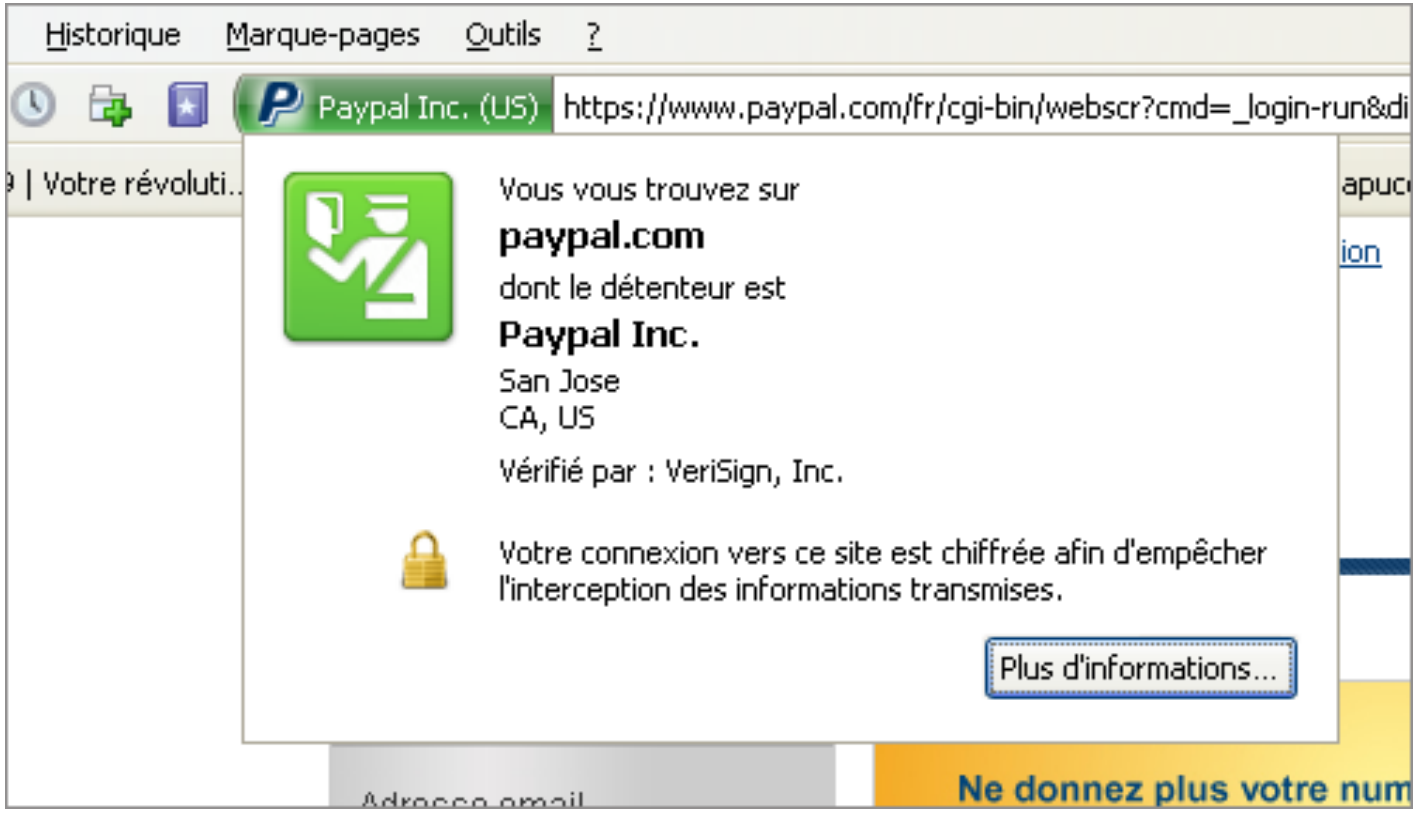
Attention! CA security (root and others)

- If *private key* is compromise : *false certificats* like *DigiNotar* in 2011 : 500 false certificats, including `*.google.com`
- Regular *security audits*
- CRL (*Certificate Revocation List*) ou OCSP (*Online Certificate Status Protocol*)

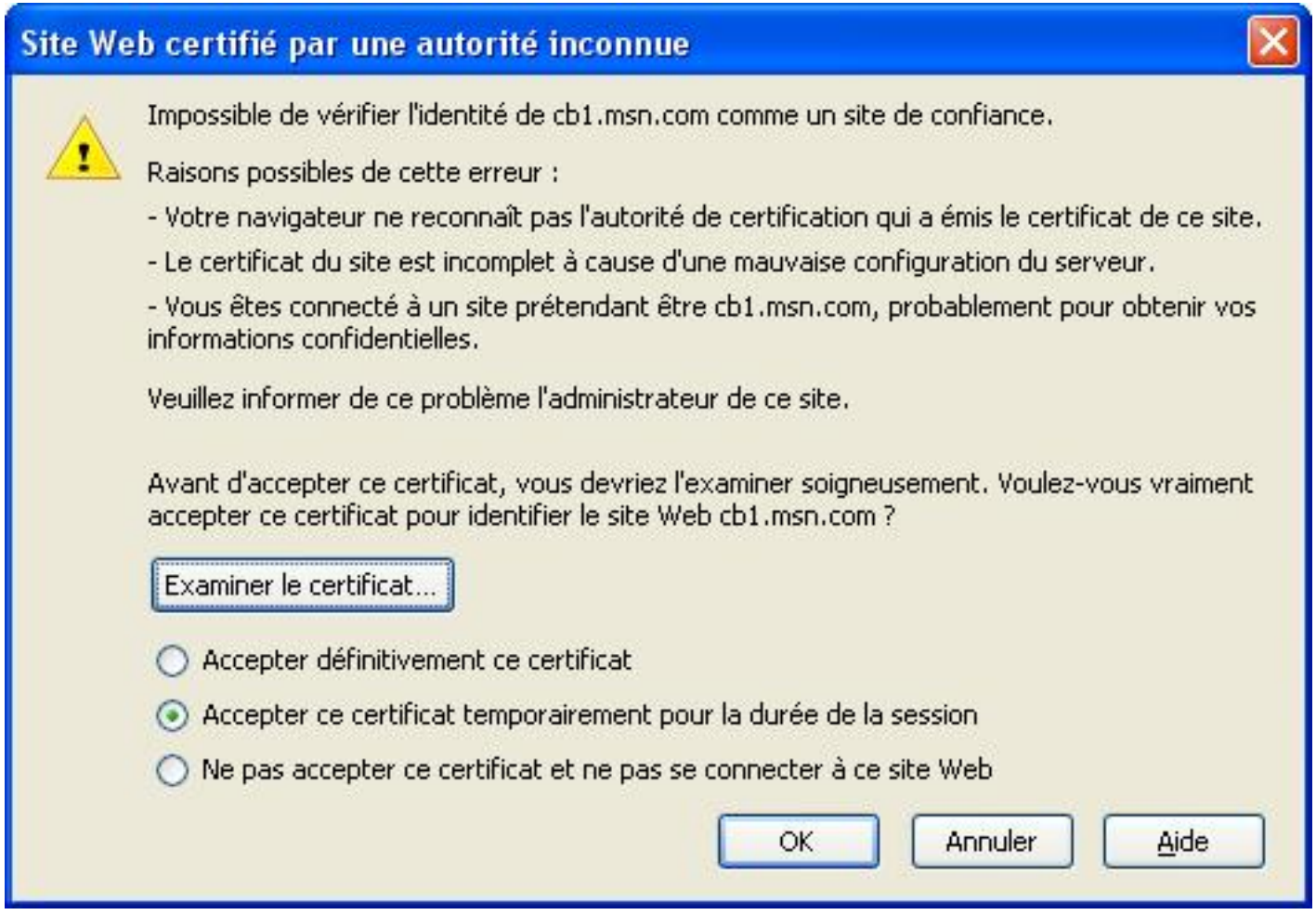
Application : TLS Handshake



Application : TLS



Application :



AC est inconnue du magasin de certificats.

Application :

Cette connexion n'est pas certifiée

Vous avez demandé à Iceweasel de se connecter de manière sécurisée à **static.ak.facebook.com**, mais nous ne pouvons pas confirmer que votre connexion est sécurisée.

Normalement, lorsque vous essayez de vous connecter de manière sécurisée, les sites présentent une identification certifiée pour prouver que vous vous trouvez à la bonne adresse. Cependant, l'identité de ce site ne peut pas être vérifiée.

Que dois-je faire ?

Si vous vous connectez habituellement à ce site sans problème, cette erreur peut signifier que quelqu'un essaie d'usurper l'identité de ce site et vous ne devriez pas continuer.

Sortir d'ici !

Détails techniques

static.ak.facebook.com utilise un certificat de sécurité invalide.

Le certificat n'est valide que pour les noms suivants :
a248.e.akamai.net , *.akamaihd.net , *.akamaihd-staging.net

(Code d'erreur : ssl_error_bad_cert_domain)

Je comprends les risques

Outline

TLS 1.2 Handshake (AKE)

Client (C): Pick N_C and KE_C
C $\rightarrow$ S : N_C, ciphers, ext
Server (S): Pick N_S and has (PK_S, SK_S)
S $\rightarrow$ C : N_s, $Cert_S$, ciphers, ext
where $Cert_S = \{S, PK_S\}_{CA}$
Client checks $Cert_S$, computes pmk
$msk = HMAC(pmk; N_C||N_S)$
$K_C||K_S = HMAC(msk; 0||N_C||N_S)$
$Fin_c = HMAC(msk; 1||\tau)$ where τ is the transcript of previous messages
C $\rightarrow$ S : $KE_C||\{Fin_c\}_{K_C}$
S computes pmk, msk, $K_C||K_S$, checks Fin_c, computes
$Fin_S = HMAC(msk; 2||\tau)$
S $\rightarrow$ C : $\{Fin_S\}_{K_S}$
Checks Fin_s
How to compute pre-master key (pmk) ? 3 modes

TLS 1.2 RSA for computing pre-master key (pmk)

Most used.
Client (C): Pick N_c and KE_C
$C \rightarrow S : N_C$
Server (S): Pick N_s and has (PK_S, SK_S) (RSA Key)
$S \rightarrow C : N_s$, $Cert_S$
Client checks $Cert_S$, choose $pmk \in_R \{0,1\}^{8*48}$
$KE_C = RSA_{PK_S}(pmk)$
$C \rightarrow S : KE_C$
S finds pmk by decrypting with SK_S associated to PK_S.

TLS 1.2 DH for computing pre-master key (pmk)

Client (C): Pick N_c
C $\rightarrow$ S : N_C
Server (S): Picks N_s and $KE_S = g^{ke_S} \mod p$ (DH Public Key)
S $\rightarrow$ C : N_s, $Cert(KE_S)$, KE_S
Client checks $Cert(KE_S)$, choose $ke_C \in_R \{0, q-1\}$
$KE_C = g^{ke_C} \mod p$, $pmk = KE_S^{ke_C} \mod p$
C $\rightarrow$ S : KE_C
S computes $pmk = KE_C^{ke_s} \mod p$.

TLS 1.2 DHE (Ephemeral) for pre-master key (pmk)

Client (C): Pick N_c
C $\rightarrow$ S : N_C
Server (S): Picks N_s and $KE_S = g^{ke_S} \mod p$ (Fresh DH Public Key)
S $\rightarrow$ C : N_s, G, $Cert(KE_S)$, KE_S
Client checks $Cert(KE_S)$, choose $ke_C \in_R \{0, q-1\}$
$KE_C = g^{ke_C} \mod p$, $pmk = KE_S^{ke_C} \mod p$
C $\rightarrow$ S : KE_C
S computes $pmk = KE_C^{ke_s} \mod p$.

Key recognition

Runs of TLs are sessions and have session IDs.

- If Client has seen before. reuse key material (msk)
- Use sID instead of N_C and N_S
  $C \rightarrow S : N_C, sID$
  $S \rightarrow C : N_S, sID, \{Fin_S\}_{K_S}$
  $K_C||K_S = HMAC(msk_{sID}; 0||N_C||N_S)$
  $Fin_c = HMAC(msk_{sID}; 1||\tau)$ where τ is the transcript of previous messages
  $C \rightarrow S : \{Fin_C\}_{K_C}$

Summary

Session Freshness

- Nonces N_C and N_S involved in key derivation
- Prevent replay attacks

Server Authentication

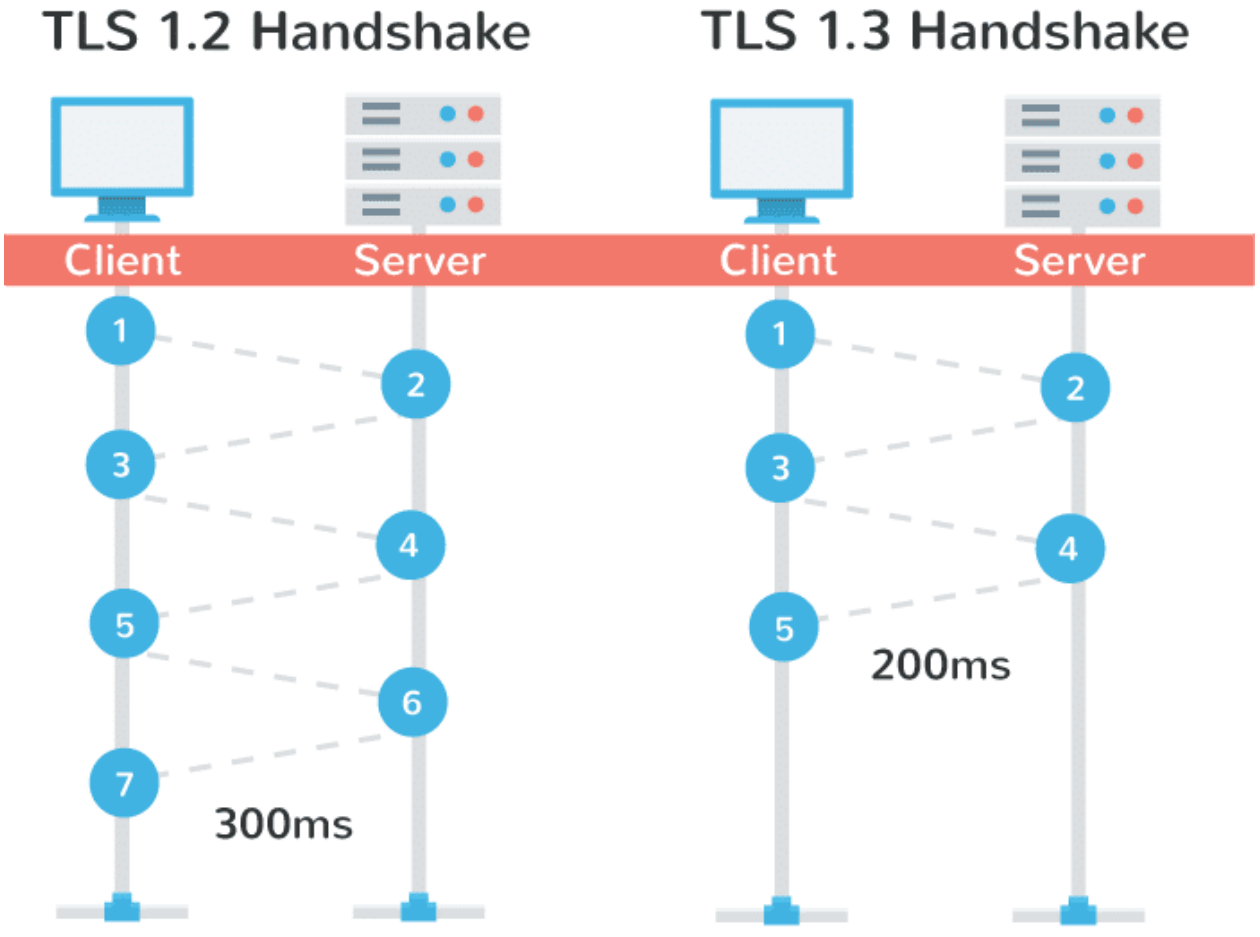
- Certificate enseures only server shares key with client
- Unilateral : anyone can exchange keys with server

Key Confirmation

- Last message: auhtenticated encryptuon with session keys
- Both parties are sure they computed the same keys

Outline

Aperçu


TLS 1.2 Handshake
Client
Server
1
2
3
4
5
6
7
300ms
TLS 1.3 Handshake
Client
Server
1
2
3
4
5
200ms

TLS 1.3

- Clean up: Remove unused or unsafe features
- Security: Improve security by using modern security analysis techniques
- Privacy: Encrypt more of the protocol
- Performance: Our target is a 1-RTT handshake for naive clients; 0-RTT handshake for repeat connections
- Continuity: Maintain existing important use cases

`https://tlswg.github.io/tls13-spec/`

TLS 1.3 removes obsolete and insecure features

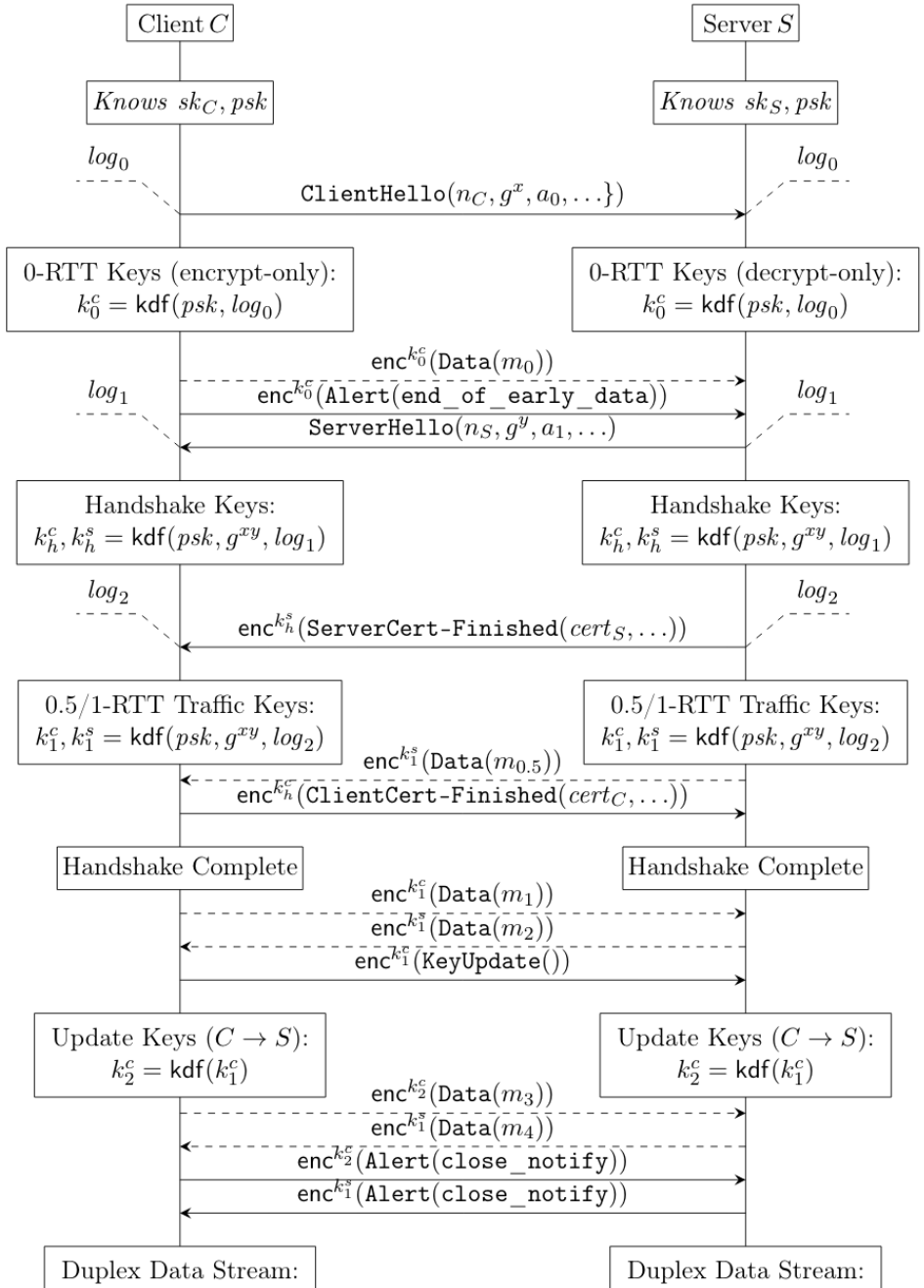
- SHA-1
- RC4
- DES
- 3DES
- AES-CBC
- MD5
- Arbitrary Diffie-Hellman groups — CVE-2016-0701
- EXPORT-strength ciphers – Responsible for FREAK and LogJam

TLS 1.3 1-RTT handshake: 12 messages in 3 flights, 16 derived keys, then data exchange.

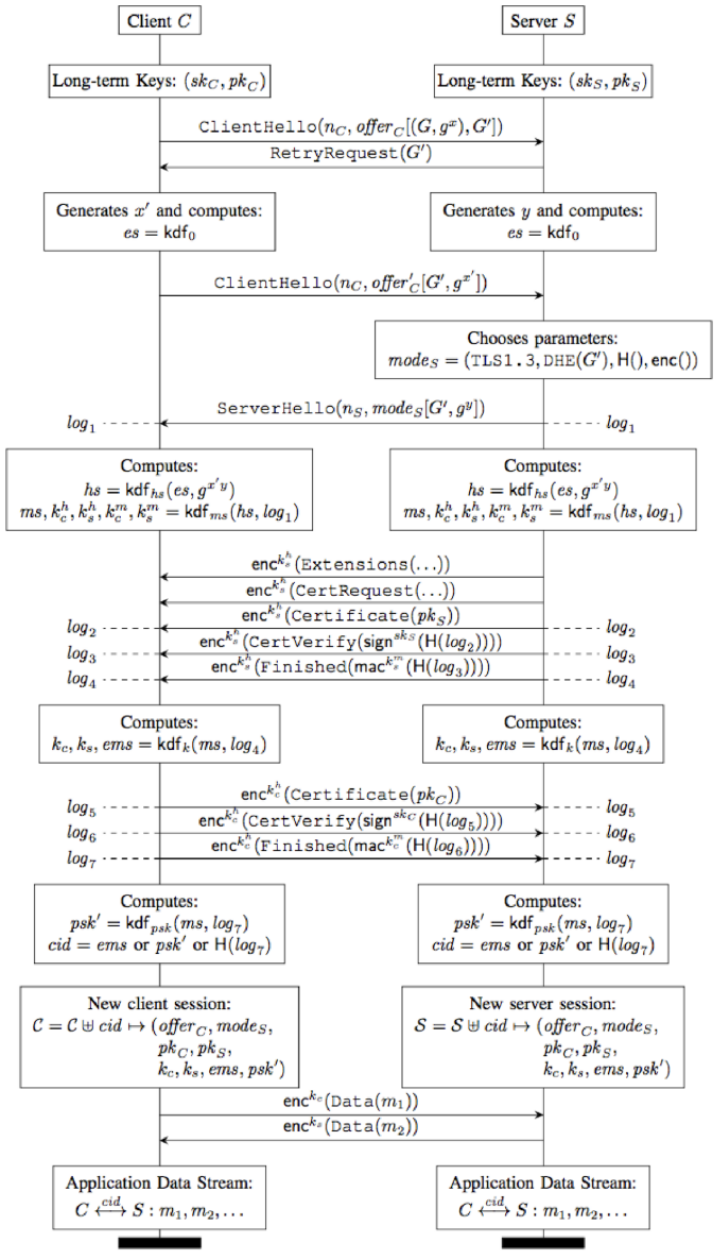
TLS 1.3

$C \rightarrow S : CHello, PK_C$, where $CHello = N_C||ciphers||ext$, supported by C
$S \rightarrow C : SHello, PK_S$ where $CHello = N_S||ciphers||ext||sub$, supported by S include in $CHello$.
$S \rightarrow C : \{Cert_S\}_{htk}$
$S \rightarrow C : \{CVf\}_{htk}$
$S \rightarrow C : \{Fin_S\}_{htk}$
$C \rightarrow S : \{Fin_C\}_{htk}$

TLS 1.3


Client C
Server S
Knows sk_C, psk
Knows sk_S, psk
log_0
log_0
ClientHello$(n_C, g^x, a_0, \ldots\})$
0-RTT Keys (encrypt-only):
$k_0^c = \mathsf{kdf}(psk, log_0)$
0-RTT Keys (decrypt-only):
$k_0^c = \mathsf{kdf}(psk, log_0)$
$\mathsf{enc}^{k_0^c}(\mathtt{Data}(m_0))$
log_1
log_1
$\mathsf{enc}^{k_0^c}(\mathtt{Alert}(\mathtt{end\_of\_early\_data}))$
$\mathtt{ServerHello}(n_S, g^y, a_1, \ldots)$
Handshake Keys:
$k_h^c, k_h^s = \mathsf{kdf}(psk, g^{xy}, log_1)$
Handshake Keys:
$k_h^c, k_h^s = \mathsf{kdf}(psk, g^{xy}, log_1)$
log_2
log_2
$\mathsf{enc}^{k_h^s}(\mathtt{ServerCert\text{-}Finished}(cert_S, \ldots))$
0.5/1-RTT Traffic Keys:
$k_1^c, k_1^s = \mathsf{kdf}(psk, g^{xy}, log_2)$
0.5/1-RTT Traffic Keys:
$k_1^c, k_1^s = \mathsf{kdf}(psk, g^{xy}, log_2)$
$\mathsf{enc}^{k_1^s}(\mathtt{Data}(m_{0.5}))$
$\mathsf{enc}^{k_h^c}(\mathtt{ClientCert\text{-}Finished}(cert_C, \ldots))$
Handshake Complete
Handshake Complete
$\mathsf{enc}^{k_1^c}(\mathtt{Data}(m_1))$
$\mathsf{enc}^{k_1^s}(\mathtt{Data}(m_2))$
$\mathsf{enc}^{k_1^c}(\mathtt{KeyUpdate}())$
Update Keys $(C \to S)$:
$k_2^c = \mathsf{kdf}(k_1^c)$
Update Keys $(C \to S)$:
$k_2^c = \mathsf{kdf}(k_1^c)$
$\mathsf{enc}^{k_2^c}(\mathtt{Data}(m_3))$
$\mathsf{enc}^{k_1^s}(\mathtt{Data}(m_4))$
$\mathsf{enc}^{k_2^c}(\mathtt{Alert}(\mathtt{close\_notify}))$
$\mathsf{enc}^{k_1^s}(\mathtt{Alert}(\mathtt{close\_notify}))$
Duplex Data Stream:
Duplex Data Stream:

TLS



Key Derivation Functions:

$\text{hkdf-extract}(k, s) = \text{HMAC-H}^k(s)$

$\text{hkdf-expand-label}_1(s, l, h) =$
$\text{HMAC-H}^s(len_{\text{H}()} \| \text{“TLS 1.3,”} \| l \| h \| \texttt{0x01})$

$\text{derive-secret}(s, l, m) = \text{hkdf-expand-label}_1(s, l, \text{H}(m))$

1-RTT Key Schedule:

$\text{kdf}_0 = \text{hkdf-extract}(0^{len_{\text{H}()}}, 0^{len_{\text{H}()}})$

$\text{kdf}_{hs}(es, e) = \text{hkdf-extract}(es, e)$

$\text{kdf}_{ms}(hs, log_1) = ms, k_c^h, k_s^h, k_c^m, k_s^m$ where
$ms = \text{hkdf-extract}(hs, 0^{len_{\text{H}()}})$
$hts_c = \text{derive-secret}(hs, \text{hts}_c, log_1)$
$hts_s = \text{derive-secret}(hs, \text{hts}_s, log_1)$
$k_c^h = \text{hkdf-expand-label}(hts_c, \text{key}, \text{“”})$
$k_c^m = \text{hkdf-expand-label}(hts_c, \text{finished}, \text{“”})$
$k_s^h = \text{hkdf-expand-label}(hts_s, \text{key}, \text{“”})$
$k_s^m = \text{hkdf-expand-label}(hts_s, \text{finished}, \text{“”})$

$\text{kdf}_k(ms, log_4) = k_c, k_s, ems$ where
$ats_c = \text{derive-secret}(ms, \text{ats}_c, log_4)$
$ats_s = \text{derive-secret}(ms, \text{ats}_s, log_4)$
$ems = \text{derive-secret}(ms, \text{ems}, log_4)$
$k_c = \text{hkdf-expand-label}(ats_c, \text{key}, \text{“”})$
$k_s = \text{hkdf-expand-label}(ats_s, \text{key}, \text{“”})$

$\text{kdf}_{psk}(ms, log_7) = psk'$ where
$psk' = \text{derive-secret}(ms, \text{rms}, log_7)$

PSK-based Key Schedule:

$\text{kdf}_{es}(psk) = es, k^b$ where
$es = \text{hkdf-extract}(0^{len_{\text{H}()}}, psk)$
$k^b = \text{derive-secret}(es, \text{pbk}, \text{“”})$

$\text{kdf}_{0RTT}(es, log_1) = k_c$ where
$ets_c = \text{derive-secret}(es, \text{ets}_c, log_1)$
$k_c = \text{hkdf-expand-label}(ets_c, \text{key}, \text{“”})$

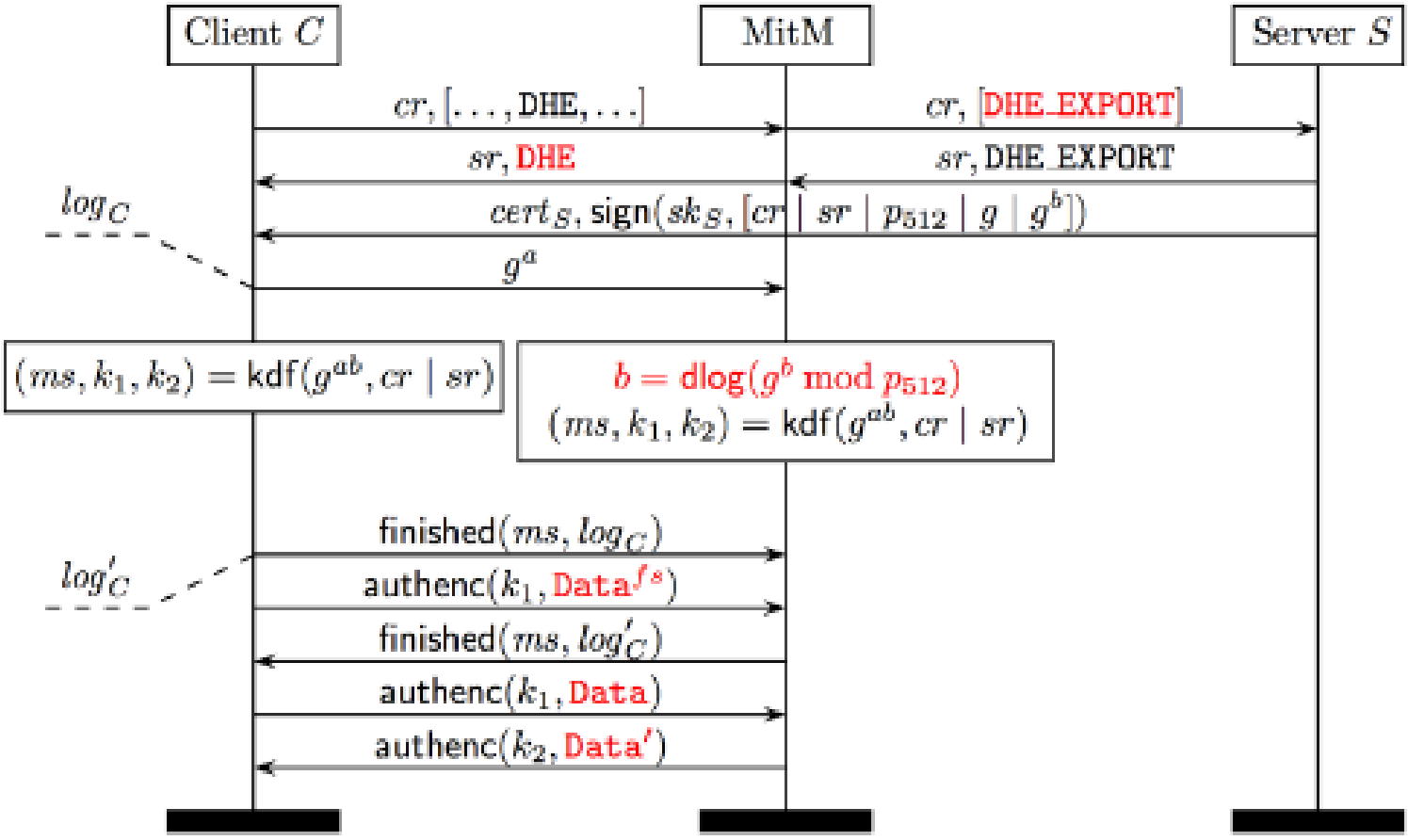
Outline

Attacks cryptography Specification Implentation randomness

- 1995 : Downgrade to SSLv2 during the negociation
- 1998 : Bleichenbacher atttack on PKCS#1 v1.5
- 2002 : Bad interpretation of X.509 (IE)
- 2008 : Overpass validation of OpenSSL certificats
- 2009 : Collision MD5 on concrete certifiats
- 2009 : Attack on renegociation
- 2009 : confusion of nuls characters in the certificats
- 2011 : BEAST Browser Exploit Against SSL/TLS, (IV implicit in CBC mode)
- 2011 : Bad interpretation of the extension X.509 (iOS)
- 2012 : Mining your Ps and Qs (no randomness in RSA key generation)
- 2012 : CRIME, Compression Ratio Info-leak Made Easy
- 2013 : Lucky 13 (oracle of padding CBC) + biais statistic on RC4
- 2014 : goto fail Apple
- 2014 : Overpass validation of GnuTLS certificats
- 2014 : Triple Handshake (renegociation and ressumption session)
- 2014 : Heartbleed and EarlyCCS
- 2014 : POODLE
- 2015 : FREAK and LogJam
- 2016 : DROWN

FREAK attack [BDFKPSZZ 2015]

Implementation flaw ; use fast 512-bit factorization to downgrade modern browsers to broken export-grade RSA
Logjam : Active downgrade to export DH



TLS 1.2 AES CBC

CBC: $C_i = E_K(P_i \oplus C_{i-1}), C_0 = IV$
$P_i = D_K(C_i) \oplus C_{i-1}, C_0 = IV$
M has to be pad using $PKCS_7$: Pad with n bytes each equal to $n : 1, 22, 333, 4444$, etc ... if $n = 0$ then add a full block og 16.

Padding Oracle

if padding is incorrect $\Rightarrow$ error message.

TLS 1.2 AES CBC

Last-bit Attack
Consider $C_1'||C_2$ where $C_1' = r_1||r_2||\ldots||r_{15}||l_{16}$
We try all values for l_{16}
If Padding oracle answer yes then $P_2[16] = 01$ we deduce $D_K(C_2)[16] = l_{16} \oplus P_2[16]$.
else try another value of l_{16}

TLS 1.2 AES CBC

Previous Last-bit Attack
Consider $C_1'||C_2$ where $C_1' = r_1||r_2||\ldots||r_{14}||l_{15}||l_{16}$
We try all values for l_{15} and l_{16} such that $P_2[16] = 02$ it means $l_{16} \oplus c_2[16] = 02$, so $l_{16} = c_2[16] \oplus 02$
If Padding oracle answer yes then $P_2[15] = 02$ we deduce $D_K(C_2)[15] = l_{15} \oplus P_2[15]$.
else try another value of l_{15}
And so on ...

TLS 1.2 AES CBC

Apply previous method for all bits of last messsages.
Then to all block.
Only for IV you need to perform a brute force attack which is resaonable.

Outline

Definition

Malware

A set of instructions that run on your computer and make your system do something that an attacker wants it to do.

Virus (Fred Cohen 1983)

A program that can infect other programs by modifying them to include a, possibly evolved, version of itself.

Polymorphic : uses a polymorphic engine to mutate while keeping the original algorithm intact (packer)
Methamorpic : Change after each infection

Classification

Propagation (codes auto-reproductor)

- Virus: human-assisted propagation (e.g., open email attachment)
- Worm: automatic propagation without human assistance.

Concealment

- Logic Bombs : wait an event to attack
- Rootkit: modifies operating system to hide its existence
- Trojan: provides desirable functionality but hides malicious operation

Botnet:

- Programs running autonomously and controlled remotely
- Can be used to spread out worms, mounting DDoS attacks

Backdoors

- Dual EC DRBG (Dual Elliptic Curve Deterministic Random Bit Generator)
  Published by NSA in June 2006, and withdrawn in 2014
- Cisco NSA Backdoor : Prime Collaboration Provisioning (PCP) by Cisco has hardcoded password.
- Hardware backdoor

History Virus

- 1972 sci-fi novel "When HARLIE Was One" features a program called VIRUS that reproduces itself
- First academic use of term virus by PhD student Fred Cohen in 1984, who credits advisor Len Adleman with coining it
- In 1982, high-school student Rich Skrenta wrote first virus released in the wild: Elk Cloner, a boot sector virus
- (c)Brain, by Basit and Amjood Farooq Alvi in 1986, credited with being the first virus to infect PCs

History Worms

- First worms built in the labs of John Shock and Jon Hepps at Xerox PARC in the early 80s
- CHRISTMA EXEC written in REXX, released in December 1987, and targeBng IBM VM/CMS systems was the first worm to use e-mail service
- The first internet worm was the Morris Worm, written by Cornell student Robert Tappan Morris and released on November 2, 1988

Virus Phases

- Dormant phase
- Propagation phase.
- Triggering phase.
- Action phase.

Viruses

- Encrypted virus :
  - Decryption engine + encrypted body
  - Randomly generate encryption key
  - Detection looks for decryption engine
- Polymorphic virus
  - Encrypted virus with random variations of the decryption engine (e.g., padding code)
  - Detection using CPU emulator.
- Metamorphic virus
  - Different virus bodies
  - Approaches include code permutation and instrucBon replacement
  - Challenging to detect

Cryptolocker

2017 Wannacry, 150 countries and $>$ 300000 computers infected

Signatures: A Malware Countermeasure

Scan compare the analyzed object with a database of signatures

- A signature is a virus fingerprint
  - E.g.,a string with a sequence of instructions specific for each virus
  - Different from a digital signature
- A file is infected if there is a signature inside its code
- Fast pattern matching techniques to search for signatures
- All the signatures together create the malware database that usually is proprietary

White/Black Listing

Maintain database of cryptographic hashes for

- Operating system files
- Popular applications
- Known infected files
- Compute hash of each file
- Look up into database
- Needs to protect the integrity of the database

Heuristic Analysis

- Useful to identify new and “zero day” malware
- Code analysis:
  - Based on the instrucBons, the antivirus can determine whether or not the program is malicious, i.e., program contains instruction to delete system files,

  Execution emulation
  - Run code in isolated emulation environment
  - Monitor actions that target file takes
  - If the actions are harmful, mark as virus
- Heuristic methods can trigger false alarms

Online AntiVirus

Software Online

- Free browser plug-in
- Authentication through third party certificate (i.e. VeriSign)
- No shielding
- Software and signatures update at each scan
- Poorly configurable
- Scan needs internet connection
- Report collected by the company that offers the service

Offline AntiVirus

Software Offline

- Paid annual subscription
- Installed on the OS
- Software distributed securely by the vendor online or a retailer
- System shielding
- Scheduled software and signatures updates
- Easily configurable
- Scan without internet connection
- Report collected locally and may be sent to vendor

Quarantine

- A suspicious file can be isolated in a folder called quarantine :
  - E.g ,. if the result of the heuristic analysis is positive and you are waiting for db signatures update
- The suspicious file is not deleted but made harmless:
  - the user can decide when to remove it or eventually restore for a false positive
  - Interacting with a file in quarantine it is possible only through the antivirus program
- The file in quarantine is harmless because it is encrypted
- Usually the quarantine technique is proprietary and the details are kept secret

Static vs. Dynamic Analysis

Static Analysis

- Checks the code without trying to execute it
- Quick scan in white list
- Filtering: scan with different antivirus and check if they return same result with different name
- Weeding: remove the correct part of files as junk to better identify the virus
- Code analysis: check binary code to understand if it is an executable, e.g., PE
- Disassembling: check if the byte code shows something unusual

Static vs. Dynamic Analysis

Dynamic Analysis

- Check the execution of codes inside a virtual sandbox
- Monitor:
  - File changes
  - Registry changes
  - Processes and threads
  - Networks ports

Virus Detection is Undecidable

Theorem by Fred Cohen (1987)

Virus abstractly modeled as program that eventually executes infect Code where infect may be generated at runtime

Proof by contradiction similar to that of the halting problem.
Suppose isVirus (P) determines whether program P is a virus
Define new program Q as follows:
Q: if (not isVirus (Q)) then Q infects else Q stops
Running isVirus on Q achieves a contradiction, two cases

- isVirus(Q) is true $\Rightarrow$ Q does nothing
- isVirus(Q) is false $\Rightarrow$ Q infects

Other Undecidable Detection Problems

- Detection of a virus:
  - by its appearance
  - by its behavior
- Detection of an evolution of a known virus
- Detection of a triggering mechanism
  - by its appearance
  - by its behavior
- Detection of a virus detector
  - by its appearance
  - by its behavior
- Detection of an evolution of
  - a known virus
  - a known triggering mechanism
  - a virus detector

Outline

Things to bring home

1. Chain of trust, certificat
2. Diffie/Hellman
3. Kerberos
4. Secure communication: TLS
5. Malwares

Thanks for you attention

Questions