CSE107: Intro to Modern Cryptography

https://cseweb.ucsd.edu/classes/sp22/cse107-a/

Emmanuel Thomé

May 17, 2022

Lecture 14a

PKI and session-key exchange

Public Key Infrastructure (PKI)

Session key exchange

Plan

Public Key Infrastructure (PKI)

Session key exchange

The public key setting

Bob's secret key is $sk[B]$ and its associated public key is $pk[B]$. The public key setting **assumes** Alice is in possession of $pk[B]$.

$A^{pk[B]}$ $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ B

$$C \xleftarrow{\$} \mathcal{E}_{pk[B]}(M) \xrightarrow{\quad C \quad} M \leftarrow \mathcal{D}_{sk[B]}(C)$$

$$\mathcal{V}_{pk[B]}(M,\sigma) \xleftarrow{\quad M,\sigma \quad} \sigma \xleftarrow{\$} \mathcal{S}_{sk[B]}(M)$$

Now Alice can encrypt a message M under $pk[B]$ to get a ciphertext C that B can decrypt using $sk[B]$.

Bob can sign a message M using $sk[B]$ to get signature σ that Alice can verify using $pk[B]$.

But how does Alice get $pk[B]$?

But who exactly are Alice and Bob?

Typically, as in most uses of TLS, Bob is a server. Its identity B is an associated domain name or ip address, for example $B = \texttt{google.com}$.

Alice is a client, also with an associated ip address.

How does A get B's public key?

How about: B runs a prescribed key-generation algorithm $\mathcal{K}$ to generate $(pk[B], sk[B])$. It sends $(B, pk[B])$ to A.

$$\xleftarrow{B, pk[B]} \qquad (pk[B], sk[B]) \xleftarrow{\$} \mathcal{K}$$

Entity-in-the-middle attack

A

$\xleftarrow{B, pk[E]}$ $\quad (pk[E], sk[E]) \xleftarrow{\$} \mathcal{K}$

Adversary E

Entity-in-the-middle attack

A Adversary E

$$\xleftarrow{B, pk[E]} \quad (pk[E], sk[E]) \xleftarrow{\$} \mathcal{K}$$

So:

$$C \xleftarrow{\$} \mathcal{E}_{pk[E]}(M) \quad \xrightarrow{C} \quad M \leftarrow \mathcal{D}_{sk[E]}(C)$$

$$\mathcal{V}_{pk[E]}(M, \sigma) \quad \xleftarrow{M, \sigma} \quad \sigma \xleftarrow{\$} \mathcal{S}_{sk[E]}(M)$$

Adversary E can decrypt ciphertexts intended for B and can forge B's signatures. Adversary effectively becomes B.

PKI, CAs and certificates

Goal: A gets an **authentic** copy of B's public key, meaning if pk claims to come from B, then A has a proof to that effect.

Popular Solution: The PKI (Public Key Infrastructure).

Certificate authority: Trusted entity that provides the above proof.

Certificate: The proof

Note: There are other ways to reach the goal: B could post its public key on its Facebook; post it on its personal or corporate webpage; include it as an attachment in its emails; put it on a keyserver like openpgp SKS; hand it to A in person; ... (what do you think of these methods?)

Let's Encrypt

Some other certificate authorities

Rank	Issuer	Usage	Market share
1	IdenTrust	20.4%	39.7%
2	Comodo	17.9%	34.9%
3	DigiCert	6.3%	12.3%
4	GoDaddy	3.7%	7.2%
5	GlobalSign	1.8%	3.5%
6	Certum	0.4%	0.7%
7	Actalis	0.2%	0.3%
8	Entrust	0.2%	0.3%
9	Secom	0.1%	0.3%
10	Let's Encrypt	0.1%	0.2%
11	Trustwave	0.1%	0.1%
12	WISeKey Group	$< 0.1\%$	0.1%
13	StartCom	$< 0.1\%$	0.1%
14	Network Solutions	$< 0.1\%$	0.1%

Certificate process

- B generates $(pk, sk) \stackrel{\$}{\leftarrow} \mathcal{K}$ by running a key-generation algorithm $\mathcal{K}$
- B sends its identity B, and pk, to CA
- CA does identity check to ensure pk is B's
- B proves knowledge of sk to CA
- CA issues certificate to B
- B sends certificate to A
- A verifies certificate and extracts B's public key pk

RSA Key generation with `openssl`

Generate a 3072-bit RSA key, output it to `key.pem`:

```
$ openssl genrsa -out key.pem 3072
Generating RSA private key, 3072 bit long modulus (2 primes)
........................++++
.................................++++
e is 65537 (0x010001)
```

Extract the public key from the key pair, which can be used in a certificate

```
$ openssl  rsa -in key.pem -outform PEM -pubout -out public.pem
writing RSA key
```

EC Key generation with `openssl`

Generate an EC (256-bit) private key with the elliptic curve `prime256v1`, output it to `key.pem`:

```
$ openssl ecparam -name prime256v1 -genkey -noout -out key.pem
```

Extract the public key from the key pair, which can be used in a certificate

```
$ openssl ec -in key.pem  -pubout -out public.pem
read EC key
writing EC key
```

Checks

B sends its identity B (domain name, ip address, email address, ...) and its public key pk to the certificate authority (CA).

Upon receiving (B, pk) the CA performs some checks to ensure pk is really B's key.

Example: If B is a domain name, then the CA sends B a challenge and checks that it can put it on the webpage of the domain name.

Example: If B is an email address, then the CA sends an email to that address with a link for B to click to verify that it owns the address.

Example: If B is a passport or driver's license, the CA may be able to verify it physically, out of band.

Proof of knowledge of secret key: The CA might have B sign or decrypt something under sk to ensure that B knows sk. This ensures B has not copied someone else's public key.

In openssl: certificate requests

```
$ openssl req -newkey rsa:3072
Generating a RSA private key
..............++++
...++++
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
[...]
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
[...]
-----BEGIN CERTIFICATE REQUEST-----
MIIDijCCAfICAQAwRTELMAkGA1UEBhMCQVUxEzARBgNVBAgMClNvbWUtU3RhdGUx
ITAfBgNVBAoMGEludGVybmV0IFdpZGdpdHMgUHR5IEx0ZDCCAaIwDQYJKoZIhvcN
AQEBBQADggGPADCCAYoCggGBAMJxl8Y0INj9UwmMoFqj4/azaE0wG4DSmDl0Wp8u
uc0ox9QQ20d7LI7cSZ0MiXF7U50AQl6VwbdPdU14BCSlfG9vP23kgirz6T4Tu0cG
7Yj82Lwquc0mhbhhdcPYooLbmxk6xu1/QQhz+9eLYZmLfE+n7MzdmRxrsLeIwPFs
IQHQo1StyD02A2JbyA1VVB8GpXe2Jj+vRTT5pWc1Qq5DBTvHb4I0ydekswb3hP6j
GoavlHATPlPostesQCuCGFxjAn1npxdePaNe11IMCqPQ2UiTlssg3KfbueAW0dzS
dl9cly0DclDh3Emriv2mtRS+SBYN6VptTqc1Uu7DwzE0hbVQkFa10fPaNwafJTL0
j+4LBbcywEoHD9baA1ZRUr80Dn3SXsY4fTqXqSR2S5mEK1K7GoEmp917kg2mITR0
o8eoUbERDEWC1h6IHAw9C4u2M7fPbln46AtbbRA0pkbCBTi9IqtsPQvE67XiH3uX
1hvCPsUfrD9sYMMjTxg/fsKCUwIDAQABoAAwDQYJKoZIhvcNAQELBQADggGBAEbL
0BbMJQ6gxG5MGA7UwDs7J4I2uZEo9YMzgRClqxzi9Xuh7BU8JNkL4hD5XgtHAn6A
ZAEgWKp0TUbqcLZBVRmiAU+nNH+ZgTIcJ7ZJySSQnI+XCi3UsGTus6cpCudikuf9
HoTuil6CI7geUFr5U57olCvbWFNPm+e0fZcVt3iUixwUrMrNLipraMsJwQx703oR
9AohE13QtvJLJysk+XI6X4B9bmm+CJ/YyoMxG+BGNP/0i5WgGyQMg0iKp4nhozC2
Wzqx0E5ECmVzSGuInF7hnIJT1UDUsEgwpjgMMZfA9L/NaZj7f4+KD309CjUbZzGw
9zVqjbCuM0weyMNzGTBLuuXvNQVpXPr5jRs8iVwFW+eQaaWAXQH40x75/DYMo3HH
NFm8drk+lhBzERwdHIHg7127x9epzfxkPwz1h1QMBooQKAqNLQ06c3DqvfRyda7z
q/IBpoZpM35PeqsYS8IUCceIjwYAM5GmpQZ4Fsb9VgW1bmK/fUBlUf9TGJhUwQ==
-----END CERTIFICATE REQUEST-----
```

key + identity

=

certificate request

In openssl: certificate requests

```
$ openssl req -newkey ec:<(openssl ecparam -name prime256v1)
Generating an EC private key
writing new private key to 'privkey.pem'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
[...]
Country Name (2 letter code) [AU]:
State or Province Name (full name) [Some-State]:
[...]
-----BEGIN CERTIFICATE REQUEST-----
MIH/MIGnAgEAMEUxCzAJBgNVBAYTAkFVMRMwEQYDVQQIDApTb21lLVN0YXRlMSEw
HwYDVQQKDBhJbnRlcm5ldCBXaWRnaXRzIFB0eSBMdGQwWTATBgcqhkjOPQIBBggq
hkjOPQMBBwNCAAStLsJ/rKbthNWCo01RF3OOZvCVZK0yS60TmJ0o0lHl/PXWqLlD
fLNPTGEmuEVdD8IkcH7INzfkz3VEO0AYsbHZoAAwCgYIKoZIzj0EAwIDRwAwRAIg
ZzQRe1h42I6Olazt+qN2ymx20qe81kJraCEtAkBmp4ICIA6kAIL9wBnBlpk2+v73
L1cN0E9k+eFszl89yVJSaazG
-----END CERTIFICATE REQUEST-----
```

key + identity

=

certificate request

Elliptic curve keys and certificate requests are shorter than for RSA, because the DLog problem is very hard on elliptic curves, and 256-bit keys are safe enough.

Certificate Issuance

Once CA is convinced that pk belongs to B, it forms a certificate

$$\mathrm{CERT}[B] = (\mathrm{CERTDATA}, \sigma),$$

where σ is the CA's signature on $\mathrm{CERTDATA}$, computed under the CA's secret key $sk[\mathrm{CA}]$, and $\mathrm{CERTDATA}$ contains:

- B's public key pk, and its type (RSA, EC, ...)
- Identity B of B
- Name of CA
- Expiry date of certificate
- ...

The certificate $\mathrm{CERT}[B]$ is returned to B.

Certificate usage

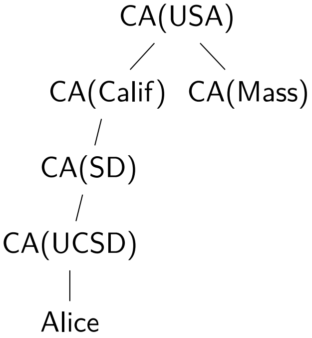
B can send $\mathrm{CERT}[B]$ to A, who is assumed to have the CA's public key $pk[\mathrm{CA}]$, and now will:

- Parse $\mathrm{CERT}[B]$ as $(\mathrm{CERTDATA}, \sigma) \leftarrow \mathrm{CERT}[B]$
- Check that $\mathcal{V}_{pk[\mathrm{CA}]}(\mathrm{CERTDATA}, \sigma) = 1$
- Extract $(pk, B, \mathrm{expiry}, \ldots) \leftarrow \mathrm{CERTDATA}$
- Check certificate has not expired
- Check that B is the desired identity
- ...

If all is well, A accepts the certificate and is ready to use the public key pk therein.

How does A **get** $pk[\mathrm{CA}]$**?** CA public keys are embedded in software such as your browser, or, on Apple, in the keychain.

Certificate hierarchies



CERT[Alice]

CERT[CA(USA) : CA(Calif)]
CERT[CA(Calif) : CA(SD)]
CERT[CA(SD) : CA(UCSD)]
CERT[CA(UCSD) : Alice]

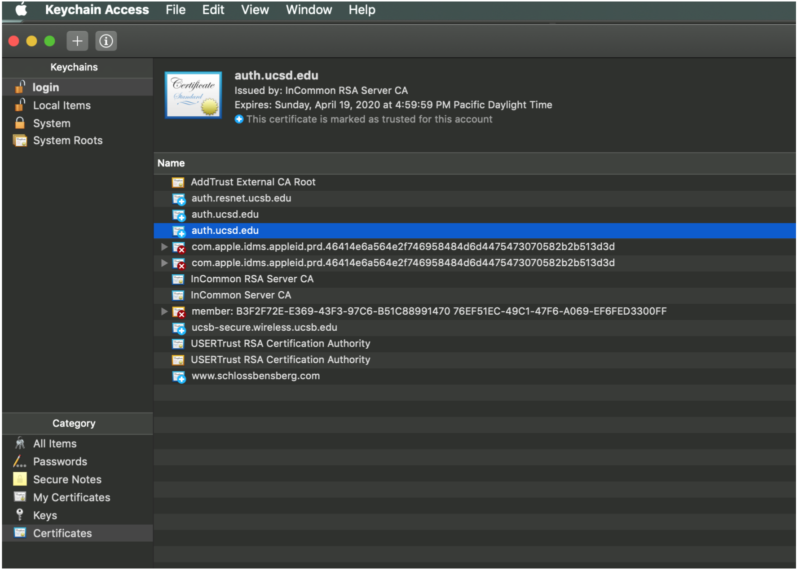
$$\mathrm{CERT}[X : Y] = (\,(pk[Y], Y, \ldots), \mathcal{S}_{sk[X]}((pk[Y], Y, \ldots))\,)$$

To verify $\mathrm{CERT}[\mathrm{Alice}]$ you need only $pk[\mathrm{CA}[\mathrm{USA}]]$.

Why certificate hierarchies?

- It is easier for CA(UCSD) to check Alice's identity (and issue a certificate) than for CA(USA) since Alice is on UCSD's payroll and UCSD already has a lot of information about her.
- Spreads the identity-check and certification job to reduce work for individual CAs
- Browsers need to have fewer embedded public keys. (Only root CA public keys needed.)

Certificates on Mac: `keychain`



A particular certificate

Revocation

Suppose B wishes to revoke its certificate $\mathrm{CERT}[B] = (\mathrm{CERTDATA}, \sigma)$, perhaps because its secret key sk, corresponding to the pk in $\mathrm{CERTDATA}$, was compromised. Then:

- B sends $\mathrm{CERT}[B]$ and revocation request to CA, signed under sk
- CA verifies the signature under pk
- CA puts $(\mathrm{CERT}[B], \mathrm{RevocationDate})$ on its Certificate Revocation List (CRL)
- This list is disseminated.

Before A accepts B's certificate, A should check that it is not on the CRL.

The OCSP (Online Certificate Status Protocol) is one way to do this.

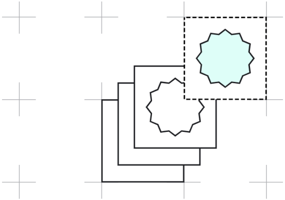
Revocation Issues

- May 13: B's secret key compromised
- May 16: B's $\mathrm{CERT}[B]$ revoked
- May 17: A sees CRL

$\mathrm{CERT}[B]$ might be used in the May 13-16 range, compromising security.

In practice, CRLs are large and revocation is a problem.

Certificate transparency (link)



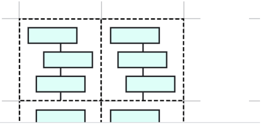
Who watches the watchers?

Historically, user agents determined if CAs were trustworthy through audits by credentialled third parties. But these tended to look at operational practices and historical performance rather than technical correctness. Such audits can’t catch everything. Before CT, there could be a significant time lag between a certificate being wrongly issued, and a CA doing something about it.

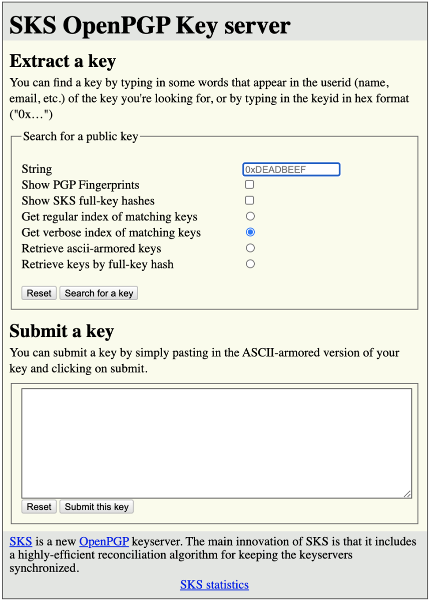
That's where
Certificate Transparency
comes in.

Independent, reliable logs

CT depends on independent, reliable logs because it is a distributed ecosystem. Built using Merkle trees, logs are publicly verifiable, append-only, and tamper-proof.



PGP SKS keyservers



SKS OpenPGP Key server

Extract a key

You can find a key by typing in some words that appear in the userid (name, email, etc.) of the key you're looking for, or by typing in the keyid in hex format ("0x...")

Search for a public key

String: 0xDEADBEEF
Show PGP Fingerprints
Show SKS full-key hashes
Get regular index of matching keys
Get verbose index of matching keys
Retrieve ascii-armored keys
Retrieve keys by full-key hash

Reset | Search for a key

Submit a key

You can submit a key by simply pasting in the ASCII-armored version of your key and clicking on submit.

Reset | Submit this key

SKS is a new OpenPGP keyserver. The main innovation of SKS is that it includes a highly-efficient reconciliation algorithm for keeping the keyservers synchronized.

SKS statistics

Plan

Public Key Infrastructure (PKI)

Session key exchange

Session key exchange

A large part of secure communication over the Internet is through protocols like TLS (`https`).

Here, public-key cryptography is not used to directly secure data.

Rather, public-key cryptography is used in a *session-key exchange* that provides (client) A and (server) B with a shared (symmetric) *session key* K.

Data is then secured under K using an authenticated encryption scheme $\mathcal{AE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$:

A^K $\qquad\qquad\qquad\qquad$ B^K

$$M \leftarrow \mathcal{D}_K(C) \quad \xleftarrow{\quad C \quad} \quad C \xleftarrow{\$} \mathcal{E}_K(M)$$

Session key exchange

Why session keys, as opposed to directly securing data with public-key cryptography?

- One reason is performance: symmetric cryptography is more efficient than asymmetric cryptography.

  We mentioned that as one of the arguments in favor of hybrid encryption.

- More fundamentally, it reflects the Internet architecture in which A and B will engage in multiple, sometimes concurrent communication sessions.

  The session key exchange paradigm gives each such session a *fresh* session key, making its security independent of that of other sessions.

Recall Diffie-Hellman Key Exchange

Let $G = \langle g \rangle$ be a cyclic group of order m in which the CDH problem is hard. Let **H**: $\{0,1\}^* \rightarrow \{0,1\}^k$ be a hash function.

A

$$x \xleftarrow{\$} \mathbb{Z}_m;\ X \leftarrow g^x \quad \xrightarrow{A, X}$$

$$L \leftarrow Y^x \quad \xleftarrow{B, Y} \quad y \xleftarrow{\$} \mathbb{Z}_m;\ Y \leftarrow g^y$$

$$L \leftarrow X^y$$

B

$$Y^x = (g^y)^x = \underbrace{g^{xy}}_{L} = (g^x)^y = X^y$$

This enables A and B to agree on the common k-bit key $K = \mathbf{H}(L) = \mathbf{H}(g^{xy})$.

So is this a suitable session key exchange protocol? Are we done?

DH Key Exchange is secure under Passive Attack

A

$x \stackrel{\$}{\leftarrow} \mathbb{Z}_m;\ X \leftarrow g^x \quad \xrightarrow{A,X}$

$L \leftarrow Y^x \quad \xleftarrow{B,Y} \quad y \stackrel{\$}{\leftarrow} \mathbb{Z}_m;\ Y \leftarrow g^y$

$L \leftarrow X^y$

B

A passive adversary is one that observes the communication, acquiring $X = g^x$ and $Y = g^y$, and wants to compute $K = H(g^{xy})$. But to do so requires solving the CDH problem, which is here assumed hard.

DH Key Exchange is secure under Passive Attack

A

$x \xleftarrow{\$} \mathbb{Z}_m;\ X \leftarrow g^x \quad \xrightarrow{A,X}$

$L \leftarrow Y^x \quad \xleftarrow{B,Y} \quad y \xleftarrow{\$} \mathbb{Z}_m;\ Y \leftarrow g^y$

$L \leftarrow X^y$

B

A passive adversary is one that observes the communication, acquiring $X = g^x$ and $Y = g^y$, and wants to compute $K = H(g^{xy})$. But to do so requires solving the CDH problem, which is here assumed hard.

However, the problem of authenticity remains (recall that KEMs assumed authenticity of ek! ($ek =$ the public key).

DH Key Exchange is insecure under Active Attack

Entity-in-the-middle attack:

A

$x \xleftarrow{\$} \mathbb{Z}_m;\ X \leftarrow g^x$ $\xrightarrow{A,X}$

$L \leftarrow Y^x$ $\xleftarrow{B,Y}$ $y \xleftarrow{\$} \mathbb{Z}_m;\ Y \leftarrow g^y$

$L \leftarrow X^y$

E

- Adversary E impersonates B.
- A thinks it shares $K = \mathbf{H}(L)$ with B, but in fact A shares K with E.

If A now encrypts, under K, a message intended for B, then E can decrypt the ciphertext and recover the message.

DH in itself does not solve the session key exchange problem.

Session key exchange requirements

We consider the unilateral, public-key setting. Here B has a certificate $\mathrm{CERT}[B]$ and corresponding public and secret keys $pk[B], sk[B]$. A is not assumed to have a certificate or corresponding keys.

This is the most common setting for TLS, where B is a server like `google.com` and A is a client.

The session key exchange should result in a session key K, known to both A and B, and satisfying:

- Authenticity: A really shares K with B, not some other entity
- Secrecy: The adversary does not know K.

This must hold even if the adversary knows session keys of other sessions and is active, meaning in complete control of the communication.

These basic requirements are supplemented by various others including forward secrecy, anonymity, ...

Session key exchange secrecy

Secrecy: The adversary E cannot distinguish the true session key K from a random string of the same length.

Suppose the protocol terminates and a party X outputs a session key K. Now we let

$K_1 \leftarrow K$; $K_0 \xleftarrow{\$} \{0,1\}^{|K|}$; game returns K_0 or K_1. Adversary must tell which.

Then the adversary's advantage should be small.

This must hold even if the adversary has obtained the session key of all other instances except the one partnered with X, and when the adversary is active, in charge of all communication.

Warning: This is not a formal definition, just a glimpse of it. The IND-CCA notion for KEMs comes closer.

Session key exchange landscape

Session-key exchange is a subtle problem.

Easy to specify protocols, hard to get them right. One very hard aspect is that an active adversary may have multiple concurrent sessions.

Many security requirements, many proposed protocols, many attacks.

Definitions and provable security treatment started in the mid 1990s and continued well into the 2000s.

Today, standards look for proof-based support.

The TLS 1.3 session key exchange protocol is based on the Sigma (sign-and-mac) protocol of [Kr03].

Plan

Session key exchange

- Key exchange protocols

Protocol KE1

A

B

$A \to B$: A, R_A

$B \to A$: $B, R_B, \mathrm{CERT}[B], \mathsf{Sig}_B(R_A \| R_B)$

A: $C \xleftarrow{\$} \mathsf{Enc}_B(L)$

$A \to B$: $C, \mathsf{MAC}_M(R_A \| R_B \| C)$

R_A, R_B, called *nonces*, are randomly chosen by the parties.

$\mathsf{Sig}_B(X)$ is B's signature on X, computed under $sk[B]$ and verifiable under the $pk[B]$ that is in $\mathrm{CERT}[B]$.

L is randomly chosen by A. Session key is $K = \mathbf{H}_1(L)$ and MAC key is $M = \mathbf{H}_2(L)$ where $\mathbf{H}_1, \mathbf{H}_2$ are public hash functions.

$\mathsf{Enc}_B(L)$ is encryption of L under B's public key $pk[B]$. Decryption uses $sk[B]$.

Identity mis-binding attack on KE1

A accepts B and thinks it shares K with B.

But B accepts E and thinks it shares K with E.

This is viewed as a problem, even though E does not know K, because there is a mis-binding of identities.

A good definition would view this as a successful attack.

A good protocol should ensure that if A accepts B with K, then B either accepts A with K, or accepts nobody with K or a key related to K.

Protocol KE2

Identity mis-binding is circumvented by inclusion of identities in the signature and the MAC, and addition of a MAC from the server:

A $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ B

$C \xleftarrow{\$} \mathrm{Enc}_B(L)$

$$\xrightarrow{\quad A, R_A \quad}$$

$$\xleftarrow{\quad B, R_B, \mathrm{CERT}[B], \mathsf{Sig}_B(A \| B \| R_A \| R_B) \quad}$$

$$\xrightarrow{\quad C, \mathsf{MAC}_M(0 \| A \| B \| R_A \| R_B \| C) \quad}$$

$$\xleftarrow{\quad \mathsf{MAC}_M(1 \| A \| B \| R_A \| R_B) \quad}$$

Session key is $K = \mathbf{H}_1(A \| B \| R_A \| R_B \| L)$ and MAC key is $M = \mathbf{H}_2(A \| B \| R_A \| R_B \| L)$.

KE2 is not forward secure

$C \xleftarrow{\$} \mathsf{Enc}_B(L)$

A, R_A $\longrightarrow$

$B, R_B, \mathrm{CERT}[B], \mathsf{Sig}_B(A \| B \| R_A \| R_B)$ $\longleftarrow$

$C, \mathsf{MAC}_M(0 \| A \| B \| R_A \| R_B \| C)$ $\longrightarrow$

$\mathsf{MAC}_M(1 \| A \| B \| R_A \| R_B)$ $\longleftarrow$

C_B $\longleftarrow$

$C_B \xleftarrow{\$} \mathsf{Enc}_K(X)$

Apr. 17: Adversary E records above flows.
May. 13: E compromises B's system and obtains $sk[B]$
May. 17: B revokes $\mathrm{CERT}[B]$, and thus $pk[B]$

However, at any time after May. 13, E can obtain session key K and decrypt C_B to obtain X via: $K \leftarrow \mathsf{Dec}_{sk[B]}(C)$; $X \leftarrow \mathsf{Dec}_K(C_B)$.

This is a violation of what's called forward secrecy.

Forward secrecy

Definition (Forward Secrecy)

Forward secrecy asks that exposure of $sk[B]$ does not allow recovery of session keys K exchanged prior to the time of exposure.

FS is achieved using the DH key exchange inside the session key exchange protocol.

Forward secrecy is considered necessary in modern session key exchange, and is present in the TLS 1.3 protocol.

Session-key exchange protocols using DH for forward secrecy are often called authenticated DH key exchange protocols.

Protocol KE3

Let $G = \langle g \rangle$ be a cyclic group of order m in which the CDH problem is hard.

A, g^a $\longrightarrow$

$\longleftarrow$ $B, g^b, \mathrm{CERT}[B], \mathsf{Sig}_B(A \| B \| g^a \| g^b), \mathsf{MAC}_M(1 \| A \| B \| g^a \| g^b)$

$\mathsf{MAC}_M(0 \| A \| B \| g^a \| g^b)$ $\longrightarrow$

Here $a, b \stackrel{\$}{\leftarrow} \mathbb{Z}_m$ are chosen by A, B, respectively, and g^a, g^b play the role of nonces.

$\mathsf{Sig}_B(X)$ is B's signature on X, computed under $sk[B]$ and verifiable under the $pk[B]$ that is in $\mathrm{CERT}[B]$.

Let $L = g^{ab}$ be the DH key. Then session key is $K = \mathbf{H}_1(A \| B \| g^a \| g^b \| L)$ and MAC key is $M = \mathbf{H}_2(A \| B \| g^a \| g^b \| L)$ where $\mathbf{H}_1, \mathbf{H}_2$ are as before.

Protocol KE3

A, g^a →

← $B, g^b, \mathrm{CERT}[B], \mathrm{Sig}_B(A \| B \| g^a \| g^b), \mathrm{MAC}_M(1 \| A \| B \| g^a \| g^b)$

$\mathrm{MAC}_M(0 \| A \| B \| g^a \| g^b)$ →

There is no public-key encryption used here, only signatures.

Compromise of $sk[B]$ only gives E the ability to forge signatures. Even given $sk[B]$, it cannot recover the DH key $L = g^{ab}$ from a prior exchange, and thus cannot distinguish from random the session key $K = \mathbf{H}_1(A \| B \| g^a \| g^b \| L)$.

Accordingly this provides forward secrecy.

This is roughly the core of the unilateral session-key exchange in the TLS 1.3 handshake.