In practice: Most widely-used standard for RSA encryption is PKCS1 (by RSA labs)

↳ Has shorter ciphertexts if we are encrypting a single $\mathbb{Z}_N$ element (no need for KEM + symmetric component]

(helpful if PKE just used to encrypt short token or metadata)

General approach: suppose N is 2048 bits and we want to encrypt 256-bit messages

↳ we will first apply a randomized padding to m to obtain a 2048-bit padded message

PKCS 1 padding:
(mode 2)

00 02	non-zero random bytes	00	m
16 bits			s bits

where $s \ll t$

t-bits long

Encryption: Compute $m_{pad} \leftarrow \text{PKCS}(m)$ and set $c \leftarrow m_{pad}^e$ [i.e., directly apply RSA trapdoor permutation to padded message]

Decryption: Compute $m_{pad} \leftarrow c^d$ and recover m from m_{pad}

In SSL v3.0: during the handshake, server decrypts client's message and checks if resulting m_{pad} is well-formed (i.e., has valid PKCS1 padding) and rejects if not

↳ scheme is vulnerable to a chosen-ciphertext attack!

↳ allows adversary to eavesdrop on connection

Devastating attack on SSL 3.0 and very hard to fix: need to change both servers + clients!

↳ TLS 1.0: fix is to set $m \xleftarrow{R} \mathbb{Z}_N^*$ if decryption ever fails and proceed normally (never alert client if padding is malformed) — setup fails at a later point in time, but hopefully no critical information is leaked...

Take-away: PKCS1 is not CCA-secure which is very problematic for key exchange

↳ Absence of security proof should always be troubling...

New standard: Optimal Asymmetric Encryption Padding (OAEP) [1994]

↳ Can be shown to be CCA-secure in random oracle model

} Standardized in PKCS1 version 2.0

Now that we have digital signatures, let's revisit the question of key exchange (with active security)

Alice		Bob
	$\xrightarrow{g^x}$	
	$\xleftarrow{g^y}$	
g^{xy}		g^{xy}

completely vulnerable to an active network adversary that can intercept and inject packets

In addition, should guarantee that one compromised session should <u>not</u> affect other <u>honest</u> sessions

- Alice $\longleftrightarrow$ Eve should not compromise security of Alice $\longleftrightarrow$ Bob

Authenticated key exchange (AKE): provides <u>security</u> against active adversaries

- Requires a "root of trust" (certificate authority) $\longrightarrow$ we need some binding between <u>keys</u> and <u>identities</u>

$$\text{Alice} \underset{\text{cert}_{\text{Alice}}}{\overset{\text{Alice}, pk_{\text{Alice}}}{\rightleftarrows}} \boxed{\text{CA}}$$

(one-time setup, at least for duration of validity period)

the certificate binds Alice's public key pk_{Alice} to Alice's identity

- Certificates typically have the following format (X509):
  - Subject (entity being authenticated)
  - Public key (public key for subject for signature scheme)
  - CA: identity of the CA issuing the certificate
  - Validity dates for certificate
  - CA's signature on certificate $\longleftarrow$ the browser and operating system have a set of hard-coded certificate authorities and their respective public keys (usually several hundred authorities)

[public-key infrastructure (PKI)]

Basic flow of Diffie-Hellman based AKE:

Alice		Bank
$x \xleftarrow{R} \mathbb{Z}_p$		$y \xleftarrow{R} \mathbb{Z}_p$
	$\xrightarrow{g^x}$	
	$\xleftarrow{g^y,\ c \leftarrow \text{Enc}_{AE}(k', (\text{cert}_{\text{Bank}}, \sigma))}$	$k, k' \leftarrow H(g, g^x, g^y, g^{xy})$
		$\sigma \leftarrow \text{Sign}(sk_{\text{Bank}}, (g, g^x, g^y, pk_{\text{Bank}}))$
derive $k, k' \leftarrow H(g, g^x, g^y, g^{xy})$		session key k'

check σ is signature on $(g, g^x, g^y, pk_{\text{Bank}})$ under pk_{Bank} is the public key identified by $\text{cert}_{\text{Bank}}$

<u>intuition:</u> $\text{cert}_{\text{Bank}}$ identifies server as Bank (with pk_{Bank})
σ binds the session parameters (g, g^x, g^y) to the public key identified by $\text{cert}_{\text{Bank}}$

<u>End of protocol</u>: Alice knows she is talking to Bank (but <u>not</u> vice versa!)
"one-sided AKE" — most common mode on the web

$\hookrightarrow$ Basis of TLS 1.3 handshake ("one-sided" AKE) ALWAYS USE TLS 1.3 – Don't invent your own AKE protocol!

client		server
	ClientHello, DH Key-Share $\longrightarrow$	
	$\longleftarrow$ ServerHello, DH Key-Share, Certificate (encrypted)	
	Finished $\longrightarrow$	
	$\longleftrightarrow$ Application Data	

<u>ClientHello</u>: List of supported ciphersuites (e.g., AES-GCM-128, AES-GCM-256)
Possible TLS extensions

older systems / foreign systems may prefer different ciphers

older versions of TLS vulnerable to cipher downgrade attacks

<u>Server Hello</u>: Chosen ciphersuite

Application layer secured using unidirection keys $k_{A \to B}$ and $k_{B \to A}$