In practice: Most widely-used standard for RSA encryption is PKCS1 (by RSA labs)
- Has shorter ciphertexts if we are encrypting a single Z^2 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

PKCS1 padding:

<table>
<thead>
<tr>
<th>00 02  non-zero random bytes  00  m</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 bits</td>
</tr>
<tr>
<td>8 bits where s = t</td>
</tr>
<tr>
<td>t bits key</td>
</tr>
</tbody>
</table>

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

In SSL 3.0: during the handshake, server decrypts client's message and checks if resulting \( m_{\text{pad}} \) 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 conversation

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

- TLS 1.0: fix is to set \( m \in \mathbb{Z}^n \) if decryption over 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) \[1994\] \{Standardized in PKCS1\}
- Can be shown to be CCA-secure in random oracle model \version{2.0}
Now that we have digital signatures, let's revisit the question of key exchange (with active security)

\[ \text{Alice} \quad g^x \quad \rightarrow \quad \text{Bob} \quad g^y \]

\[ g^{xy} \quad \left\{ \begin{array}{l}
\text{completely vulnerable to an active network adversary that can intercept and inject packets}
\end{array} \right. \]

In addition, should guarantee that one compromised session should not affect other honest sessions
- Alice ↔ Eve should not compromise security of Alice ↔ Bob

Authenticated key exchange (AKE): provides security against active adversaries
- Requires a "root of trust" (certificate authority) → we need some binding between keys and identities

\[
\begin{align*}
\text{Alice, plain} & \quad \xrightarrow{\text{cert Alice}} \quad \text{CA} \\
\text{the certificate binds Alice's public key } pk_{\text{Alice}} \text{ to Alice's identity} & \\
\text{Certificates typically have the following format (X509):} & \\
\text{- Subject (entity being authenticated)} & \\
\text{- Public key (public key for subject for signature scheme)} & \\
\text{- CA: identity of the CA issuing the certificate} & \\
\text{- Validity dates for certificate} & \\
\text{- CA's signature on certificate} & \xleftarrow{\text{the browser and operating system have a set of hand-rolled certificate authorities and their respective public keys (usually several hundred authorities)}} \\
\end{align*}
\]

Basic flow of Diffie-Hellman based AKE:

\[
\begin{align*}
\text{Alice} & \quad g^x \quad \rightarrow \quad \text{Bank} \quad g^y \quad \rightarrow \quad \text{Bank} \quad g^z \\
\text{g}^y \quad \xleftarrow{\text{CS-Enc}(k', \text{cert}_{\text{Bank}})} & \quad k, k' \leftarrow H(y, g^x, g^y, g^z) \\
\sigma & \leftarrow \text{Sign}((k', \text{cert}_{\text{Bank}}), (y, g^x, g^y, pk_{\text{Bank}})) \\
\sigma \text{ derives } k' & \leftarrow H(y, g^x, g^y, g^z) \\
\text{check } \sigma \text{ is signature on } (y, g^x, g^y, pk_{\text{Bank}}) & \\
\text{under } pk_{\text{Bank}} \text{ is the public key identified by cert}_{\text{Bank}} & \text{ intuition: cert}_{\text{Bank}} \text{ identifies server as Bank (with } pk_{\text{Bank}}) \quad \sigma \text{ binds the session parameters } (g, y^x, g^z) \text{ to the public key identified by cert}_{\text{Bank}}
\end{align*}
\]

End of protocol: Alice knows she is talking to Bank (but not vice versa!)

> Basic of TLS 1.3 handshake ("one-sided" AKE) ALWAYS USE TLS 1.3 — Don't invent your own AKE protocol!

<table>
<thead>
<tr>
<th>client</th>
<th>server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client Hello</td>
<td>Client Hello: list of supported ciphersuites (e.g., AES-GCM-128, AES-GCM-256)</td>
</tr>
<tr>
<td>DH Key-Share</td>
<td>Possible TLS extensions</td>
</tr>
<tr>
<td>Server Hello</td>
<td>Server Hello: Chosen ciphersuite</td>
</tr>
<tr>
<td>Certificate (encrypted)</td>
<td>Application layer secured using unidirectional keys kn→B and kB→A</td>
</tr>
<tr>
<td>Finished Application Data</td>
<td>older systems/foreign systems may prefer different cipher suites</td>
</tr>
<tr>
<td></td>
<td>older versions of TLS vulnerable to cipher downgrade attacks</td>
</tr>
</tbody>
</table>