

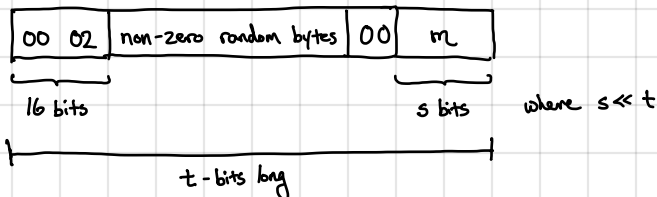
In practice: Most widely-used standard for RSA encryption is PKCS1 (by RSA labs)

↳ 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

PKCS1 padding:
(mode 2)



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

Decryption: Compute $m_{\text{pad}} \leftarrow c^d$ 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 $m \leftarrow \mathbb{Z}_N^*$ 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...

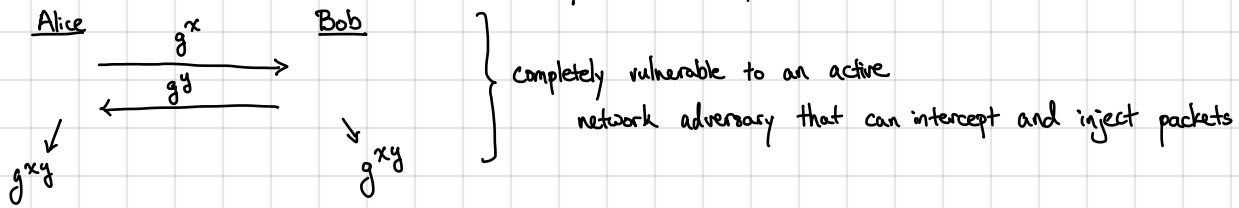
Take-away: PKCS1 is not CCA-secure which is very problematic for key exchange

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

New standard: Optimal Asymmetric Encryption Padding (OAEP) [1994] } standardized in PKCS1 version 2.0

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

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

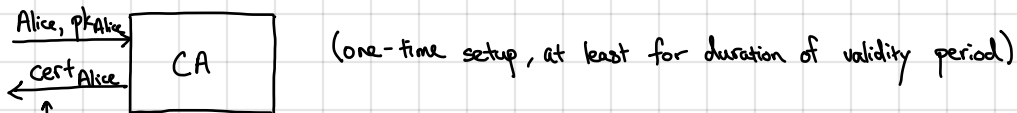


In addition, should guarantee that one compromised session should not affect other honest sessions

- 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



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

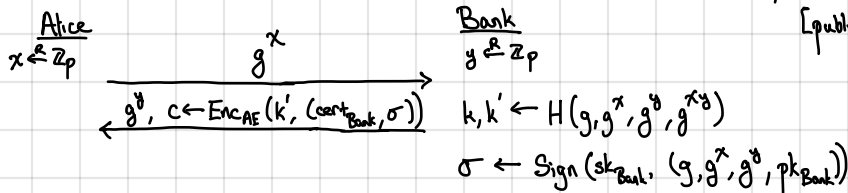
- Certificates typically have the following format (X509):

- 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

← 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:



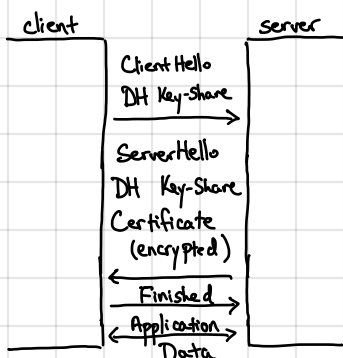
derive $k, k' \leftarrow H(g, g^x, g^y, g^{xy})$
 check σ is signature on (g, g^x, g^y, pk_{Bank})
 under pk_{Bank} is the public key identified by $cert_{Bank}$

} intuition: $cert_{Bank}$ identifies server as Bank (with pk_{Bank})
 σ binds the session parameters (g, g^x, g^y) to the public key identified by $cert_{Bank}$

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

"one-sided AKE" - most common mode on the web

↳ Basis of TLS 1.3 handshake ("one-sided" AKE) **ALWAYS USE TLS 1.3 - Don't invent your own AKE protocol!**



ClientHello: List of supported ciphersuites (e.g., AES-GCM-128, AES-GCM-256)

Possible TLS extensions

ServerHello: Chosen ciphersuite

Application layer secured using unidirectional keys $k_{A \rightarrow B}$ and $k_{B \rightarrow A}$

older systems / foreign systems may prefer different ciphers / older versions of TLS vulnerable to cipher downgrade attacks