Lecture 20

MAC-then-Encrypt or Encrypt-then-MAC?

- Method 1: Encrypt-then-MAC
  - First compute Enc(K_1, M)
  - Then MAC the ciphertext: MAC(K_2, Enc(K_1, M))
- Method 2: MAC-then-encrypt
  - First compute MAC(K_2, M)
  - Then encrypt the message and the MAC together: Enc(k_1, M || MAC(K_2, M))
- Which is better?
  - In theory, both are secure if applied properly
  - MAC-then-encrypt has a flaw: You don’t know if tampering has occurred until after decrypting
    - Attacker can supply arbitrary tampered input, and you always have to decrypt it
    - Passing attacker-chosen input through the decryption function can cause side-channel leaks
- **Always use encrypt-then-MAC** because it’s more robust to mistakes

TLS 1.0 “Lucky 13” Attack

- TLS: A protocol for sending encrypted and authenticated messages over the Internet
- TLS 1.0 uses MAC-then-encrypt: $\mathrm{Enc}(k_1, M \,||\, \mathrm{MAC}(k_2, M))$
  - The encryption algorithm is AES-CBC
- The Lucky 13 attack abuses MAC-then-encrypt to read encrypted messages
  - Guess a byte of plaintext and change the ciphertext accordingly
  - The MAC will error, but the time it takes to error is different depending on if the guess is correct
  - Attacker measures how long it takes to error in order to learn information about plaintext
  - TLS will send the message again if the MAC errors, so the attacker can guess repeatedly
- Takeaways
  - Side channel attack: The algorithm is proved secure, but poor implementation made it vulnerable
  - Always encrypt-then-MAC
- https://medium.com/@c0D3M/lucky-13-attack-explained-dd9a9fd42fa6

Authenticated Encryption: Summary

- Authenticated encryption: A scheme that simultaneously guarantees confidentiality and integrity (and authenticity) on a message
- First approach: Combine schemes that provide confidentiality with schemes that provide integrity and authenticity
  - MAC-then-encrypt: Enc(K_1, M || MAC(K_2, M))
  - Encrypt-then-MAC: MAC(K_2, Enc(K_1, M))
  - Always use Encrypt-then-MAC because it's more robust to mistakes

Digital Signature

Digital Signatures

- NIST FIPS PUB 186-4 - the result of a cryptographic transformation of data that, when properly implemented, provides a mechanism for verifying origin authentication, data integrity, and signatory non-repudiation
- Based on asymmetric keys

Digital Signatures

- Asymmetric cryptography is good because we don’t need to share a secret key
- Digital signatures are the asymmetric way of providing integrity/authenticity to data
- Assume that Alice and Bob can communicate public keys without David interfering

Digital Signatures: Definition

- Three parts:
  - KeyGen() → *PK*, *SK*: Generate a public/private keypair, where *PK* is the verify (public) key, and *SK* is the signing (secret) key
  - Sign(*SK*, *M*) → *sig*: Sign the message *M* using the signing key *SK* to produce the signature *sig*
  - Verify(*PK*, *M*, *sig*) → {0, 1}: Verify the signature *sig* on message *M* using the verify key *PK* and output 1 if valid and 0 if invalid
- Properties:
  - **Correctness**: Verification should be successful for a signature generated over any message
    - Verify(*PK*, *M*, Sign(*SK*, *M*)) = 1 for all *PK*, *SK* ← KeyGen() and *M*
  - **Efficiency**: Signing/verifying should be fast
  - **Security**: Same as for MACs except that the attacker also receives PK
    - Namely, no attacker can forge a signature for a message