Authenticated Encryption: Definition

- **Authenticated encryption** (**AE**): A scheme that simultaneously guarantees confidentiality and integrity (and authenticity, depending on your threat model) on a message
- Two ways of achieving authenticated encryption:
  - Combine schemes that provide confidentiality with schemes that provide integrity
    MAC ?
  - Use a scheme that is designed to provide confidentiality and integrity

Scratchpad: Let’s design it together

- You can use:
  - An encryption scheme: Enc(K, M) and Dec(K, M)
  - An unforgeable MAC scheme (e.g. HMAC): MAC(K, M)
- First attempt: Alice sends Enc(K_1, M) and MAC(K_2, M) ✗
  - Integrity? Yes, attacker can’t tamper with the MAC
  - Confidentiality? No, the MAC is not secure $M \,||\, MAC(K_2\, M)$
- Idea 1: Let’s compute the MAC on the *ciphertext* instead of the plaintext: Enc(K_1, M) and MAC(k$_2$, Enc(K_1, M)) $Enc(K_1, M) \,||\, MAC$
  - Integrity? Yes, attacker can’t tamper with the MAC
  - Confidentiality? Yes, the MAC might leak info about the ciphertext, but that’s okay
- Idea 2: Let’s encrypt the MAC too: Enc(K_1, M || MAC(K_2, M))   ~~HMAC~~
  - Integrity? Yes, attacker can’t tamper with the MAC
  - Confidentiality? Yes, everything is encrypted

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

Enc(k, M) M'

trick Attacker M'

Sender

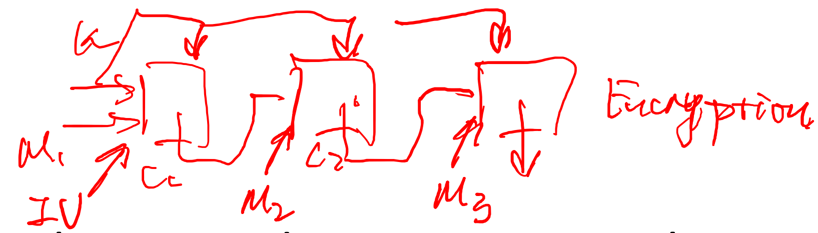
E(K₁, M' || MAC(K₂, M'))

Decrypt

RSA timing attack

TLS 1.0 "Lucky 13" Attack

no exam



- TLS: A protocol for sending encrypted and authenticated messages over the Internet
- TLS 1.0 uses MAC-then-encrypt: Enc(k_1, M || MAC(k_2, M)) bit
  - The encryption algorithm is AES-CBC Cypher Block Chain
- 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

    wrong bit → verification time long
  - Attacker measures how long it takes to error in order to learn information about plaintext

    timing attack
  - TLS will send the message again if the MAC errors, so the attacker can guess repeatedly

sender TLS server verify

randomly delay sending response msg

Attacker

- Takeaways
  - Side channel attack: The algorithm is proved secure, but poor implementation made it vulnerable
  - Always encrypt-then-MAC

Authenticated Encryption: Summary

big data transmission

no group key,

- 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

Computation Time MAC < Symmetric Encryption < Asymmetric Encryption

Defense: Add randomness

timing attack
Big data input

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

no confidentiality

- Based on asymmetric keys

Digital Signatures

H( )

- 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

PKI

Third Authority Party

certificate

Sig(PK, SK)

correct

sender

receiver

private

public. key

sign

verify

correct

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 快?
  - **Security**: Same as for MACs except that the attacker also receives PK
    - Namely, no attacker can forge a signature for a message

without private key

RSA Signature

- KeyGen(): Same as RSA Encryption
  - Randomly pick two large primes, p and q
  - Compute $n = pq$ → large number, public.
    - n is usually between 2048 bits and 4096 bits long
  - Choose e $= \phi(n)$, $\gcd(e, \phi(n)) = 1$
    - Requirement: e is relatively prime to $(p - 1)(q - 1)$
    - Requirement: $2 < e < (p - 1)(q - 1) = \phi(n)$
  - Compute $d = e^{-1} \bmod (p - 1)(q - 1)$ $\phi(n)$
  - **Public key**: n and e → public
  - **Private key:** d

A Short Quiz

- We will have a short quiz on Wednesday, Oct. 30, in class
- A short quiz will cover the materials taught that day.