51.505 – Foundations of Cybersecurity

Week 9 - Secure Channel & Randomness

Created by **Pawel Szalachowski** (2017)
Modified by **Jianying Zhou** (2018)

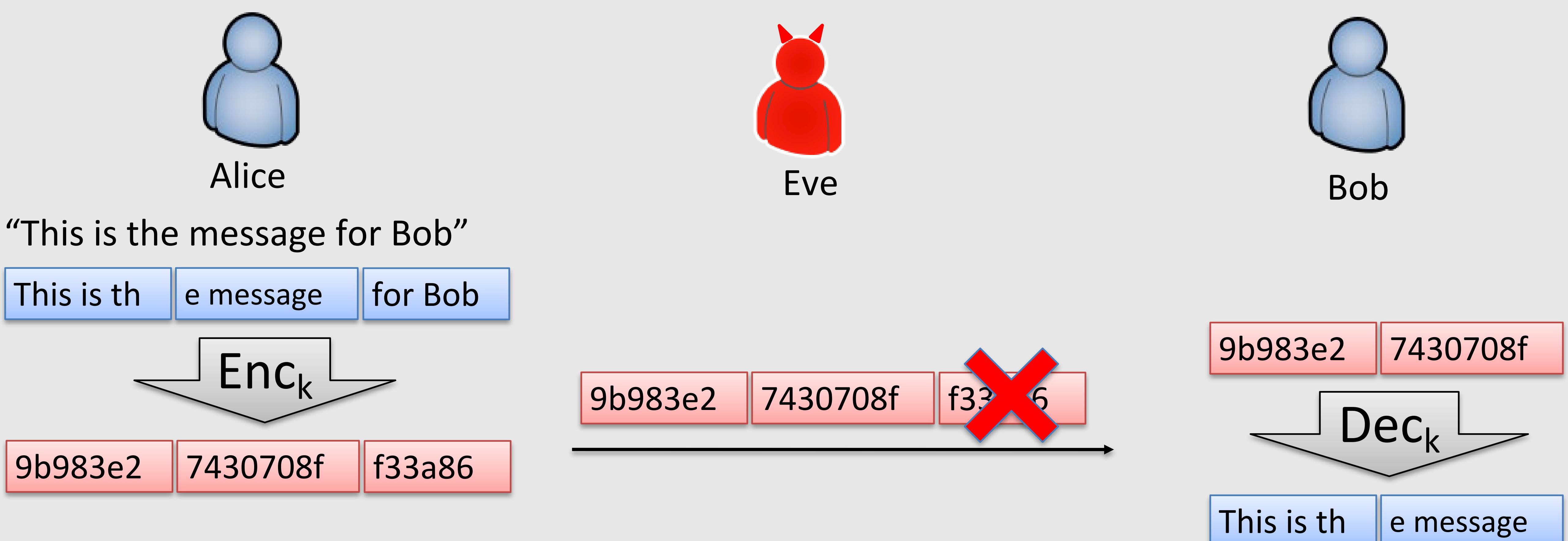
Last updated: 18 Oct 2018

Recap

- Questions on Week 8’s exercises?

CPA-secure Encryption

- CPA-security is about confidentiality, not integrity.
- Secure channel is most useful application of cryptography.



Secure Channel

- Roles
  - ✓ Alice and Bob wants to communicate *securely.*
  - ✓ Eve can eavesdrop, modify, delete, or insert data.
- Key
  - ✓ Only Alice and Bob know a shared key.
  - ✓ Every time the secure channel is initialized, a new key (<u>session key</u>) is generated.
- Messages
  - ✓ Alice and Bob exchange <u>discrete</u> messages.

Security Properties

- Alice sends $m_1, m_2, \ldots$ that are processed by the secure channel algorithms and then sent to Bob. Bob processes the messages through the secure channel algorithms and obtains $m'_1, m'_2, \ldots$

  ✓ **Secrecy:** Eve does not learn anything about the messages (except their timing and size, by *traffic analysis*).

  ✓ **Message Order:** If Eve attacks the channel, the sequence $m'_1, m'_2, \ldots$ received by Bob is a subsequences of $m_1, m_2, \ldots$ and Bob learns exactly which subsequence he received.

  – Subsequence is constructed from the original sequence by removal of zero or more elements.

Authenticated Encryption

- Combination of encryption and authentication
  - ✓ Prevent from eavesdropping and **modifying** adversary
  - ✓ Use existing primitives
- How to combine them?
  - ✓ Use CPA-secure encryption and secure MAC
  - ✓ Use <u>different</u> keys for each primitive (derived from the session key)
  - ✓ <u>Order</u> of authentication and encryption?

Encrypt-and-Authenticate

- Encrypt a message and authenticate the message. Transmit the ciphertext and the tag.
  - ✓ derive K_e and K_a from K
  - ✓ $ctxt = Enc(K_e, msg); tag = Mac(K_a, msg) ; send(ctxt \mathbin{||} tag)$
- Encryption and authentication can be done in <u>parallel</u>.
- Receiver has to first decrypt the ciphertext to check authenticity.
- According to theoretical results it is ***insecure*.**
  - ✓ Attacker sees the tag of the initial message itself, which could lead to a privacy leak.

Authenticate-then-Encrypt

- Authenticate a message then encrypt both the message and the tag. Transmit the ciphertext.
  - ✓ derive K_e and K_a from K
  - ✓ *tag = Mac(K_a, msg) ; ctxt = Enc(K_e, msg || tag) ; send(ctxt)*
- Tag is <u>invisible</u> to Eve.
  - ✓ Eve has no valid (message, authtag) pair.
- Receiver has to first decrypt the ciphertext to check authenticity.

Encrypt-then-Authenticate

- Encrypt a message and authenticate the ciphertext. Transmit ciphertext and the tag.
  - ✓ derive K_e and K_a from K
  - ✓ *ctxt = Enc(K_e, msg); tag = Mac(K_a, ctxt) ; send(ctxt || tag)*
- According to theoretical results it is ***secure.***
- Efficiency: Bob never decrypts bogus messages.
- Eve has valid (message, authtag) pairs.
- May violate Horton Principle (Authenticate what it meant, not what is said) – Week 8.

Which to use?

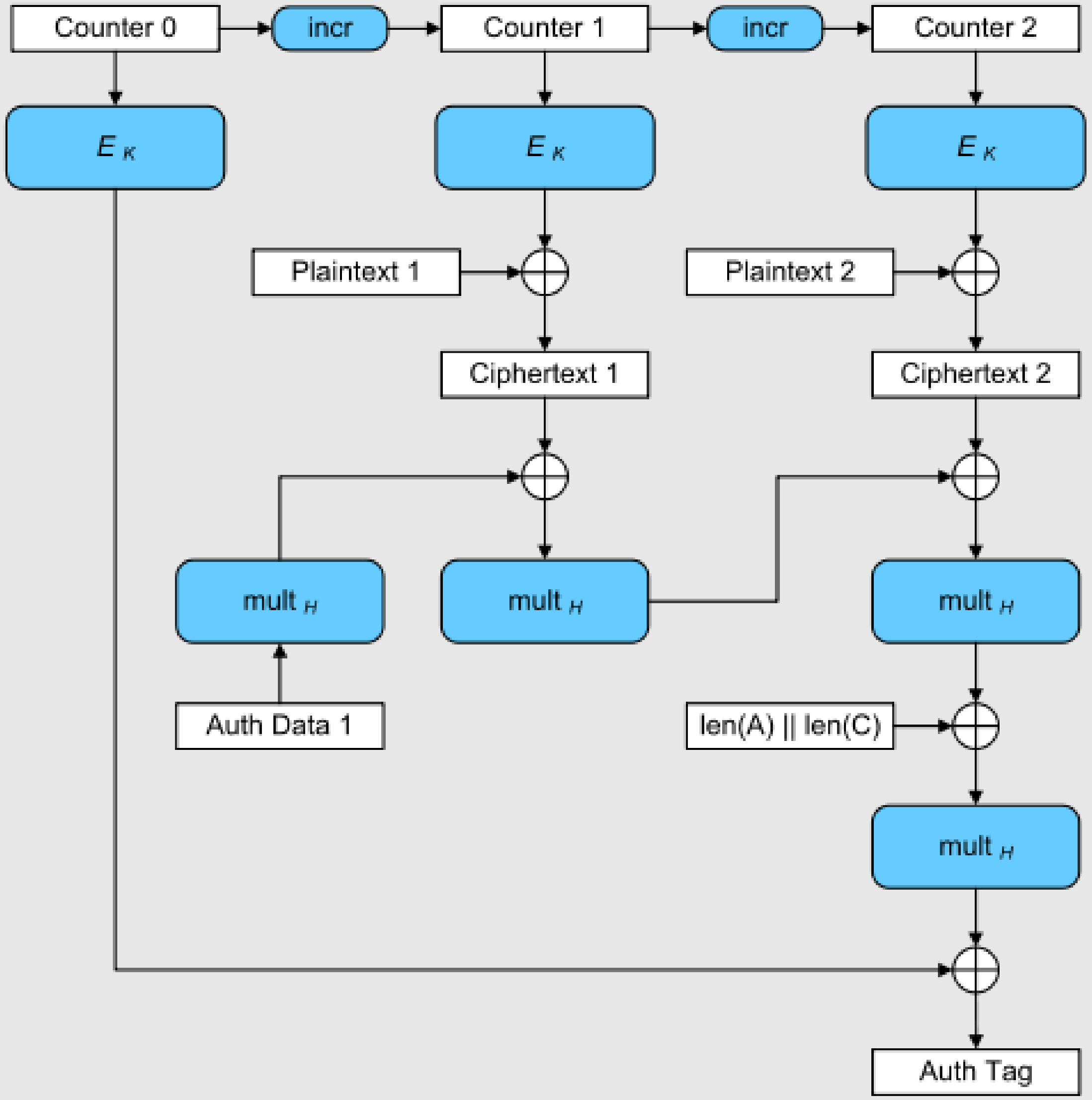
- Encrypt-and-Authenticate
  - ✓ used in SSH
- Authenticate-then-Encrypt
  - ✓ used in SSL/TLS
- **Encrypt-then-Authenticate**
  - ✓ Encryption is CPA-secure and MAC is secure, then the construction is CCA-secure.
  - ✓ used in IPSec

Authenticated Associated Data

- Some parts of a message cannot be encrypted.
  - ✓ e.g., packet headers
- With the encrypt-then-authenticate scheme
  - ✓ encrypt only relevant parts
  - ✓ authenticate the entire message

Alternatives

- Advanced modes of operation of block ciphers – **authenticated encryption**
  - ✓ OCB (Offset Codebook Mode)
    - Very efficient
    - Limited adoption due to patent
  - ✓ CCM (Counter with CBC-MAC)
    - Combine CBC-MAC with CTR mode encryption
    - Used in IEEE 802.11i, IPsec, TLS 1.2, BLE, …
  - ✓ GCM (Galois/Counter Mode)
    - Combine CTR mode encryption with Galois mode of authentication
    - Used in IEEE 802.11ad, IPsec, TLS 1.2, SSH, …



Secure Channel Design

- Message Numbers
- Encryption
- Authentication
- Initialization
- Sending/Receiving
- Message Order

Message Numbers

- Replay protection (Bob can efficiently reject replays)
- Can be used for deriving IVs for the encryption algorithm
- Order preserving (must increase monotonically and be unique)
- Usually implemented as a **counter** (starting from 1)
  - ✓ 32-bit is enough for most applications, up to 2^{32} -1.
  - ✓ Counter cannot be reused thus, with the last value the session has to be re-established.
  - ✓ Do not have to be encrypted (authentication-only is OK).

Encryption

- CPA-secure
  - ✓ CBC mode with IV
  - ✓ CTR mode with nonce and counter
  - ✓ ...
- Padding if necessary

Authentication

- Secure MAC
  - ✓ HMAC
  - ✓ CMAC
- MAC has to be computed over the metadata and actual content, such that an adversary cannot modify both.

Initialization

- From the main session K, derive an encryption key
  - ✓ For two-way communication a key per direction can be derived.
- From the main session K, derive an authentication key
  - ✓ For two-way communication a key per direction can be derived.
- Reset counters
  - ✓ Usually two counters are used (for sending and receiving).

Sending

- Pad message (if needed)
- Using the encryption key, encrypt the message to be protected
- Using the authentication key, authenticate the ciphertext and metadata (additional authenticated data)
- Send the metadata, ciphertext, and the tag
- Increment counter

Receiving

- Check message order
- Using the authentication key, verify the message
- Using the encryption key, decrypt the message
- Remove padding (if needed)
- Increment counter

Message Order

- Reordering may happen during transmission.
- It is application-specific.
  - ✓ Some applications accept reordered messages.
- In some cases receiver can itself do ordering by buffering.
  - ✓ Again, application specific.

Randomness

Generating Randomness

- Informally, random data is **unpredictable** to the attacker
- Applications
  - ✓ Key material
  - ✓ Initialization vectors
  - ✓ Nonces
  - ✓ Salts
- Problems
  - ✓ Lack of initial randomness
  - ✓ Backdoors (e.g., Dual_EC_DRBG)
  - ✓ Bugs (e.g., Debian SSH)

Entropy

- (Introduced before)
- Measure of randomness
- x-bit string that is completely random has x bits of entropy

Real Randomness

- What is really *random*?
- Computers are deterministic, thus randomness is taken externally.
  - ✓ keyboard, mouse, microphone, network traffic, ...
  - ✓ other I/O interruptions
  - ✓ external randomness devices
- Problems
  - ✓ Quality (entropy) is hard to measure.
  - ✓ Availability (taken externally, may not be available at any time, e.g. keystroke)

Pseudorandomness

- Pseudorandom Number Generator (PRNG)

  ✓ Numbers are generated deterministically, from a random seed.

  ✓ Address the *availability* issue:
    - Only use real random data to seed a PRNG.
    - Generate immediately as many pseudorandom bits as needed.

  ✓ If a PRNG is used, the protocol is only secure as long as PRNG is not broken.

Cryptographically Secure PRNG

- Good statistical properties (must hold for any PRNG)
- **The next-bit test:** *given the first k bits of a random sequence, there is no polynomial-time algorithm that can predict the (k+1)th bit with probability of success non-negligibly better than 50%.*
- **State compromise extensions**: *if part or all of its state has been revealed (or guessed correctly), it should be impossible to reconstruct the stream of random numbers prior to the revelation.*
- Usually, build on cryptographic primitives or hard mathematical problems.

Key Derivation Function (KDF)

- Related problem: derive secret key(s) from a secret value
  - ✓ The secret value can be a master key, password, or passphrase.
- Many ways assuming a Pseudorandom Function (PRF)
  - ✓ newKey = SHA-256(MasterKey || "NewSessionKey")
  - ✓ newKey = HMAC-SHA-256(MasterKey, "NewSessionKey")
  - ✓ Password-Based Key Derivation Function

Attacks to PRNG

- Security of PRNG relies on
  - ✓ How to get a random seed?
  - ✓ How to keep the random seed secret in a real-world situation?

- At any point of time, PRNG has an *internal state*.
  - ✓ Ensure the next PRNG request does not return the same random data.

- Attacks:
  - ✓ The attacker acquires the internal state. Then it can get all subsequent output of PRNG and update of internal state.
  - ✓ The same PRNG state is used more than once, e.g., 2 VMs are booted from the same state and read the same seed file from disk.

PRNG Design Example

- Initialization [*Initialization*]
  - ✓ Set key K and counter C to zero. $(K,C) \leftarrow (0,0)$
  - ✓ Output: G as generator state, $G \leftarrow (K,C)$
- Reseed [*Reseed(G,s)*]
  - ✓ Input: G as generator state, s as a new seed
  - ✓ Output: $K \leftarrow$ SHA-256($K||s$); $C \leftarrow C+1$

PRNG Design Example

- Generate Blocks [*GenerateBlocks(G,k)*]
  - ✓ Input: *G as generator state, k* as number of random blocks to generate
  - ✓ Output: *r* as pseudorandom string of 16*k* bytes

    for $i = 1, \ldots, k$ do {$r \leftarrow r \,||\, E(K,C)$; $C \leftarrow C+1$}

    return *r*

- Generate Random Data [*PseudoRandomData(G,n)*]
  - ✓ Input: *G* as generator state, *n* as number of bytes of random data to generate
  - ✓ Output: *r* as pseudorandom string of *n* bytes

    r ← first-n-bytes *(GeneateBlocks(G, n/16))*

Key Points

- Secure channel
  - ✓ Design of secure channel
  - ✓ Combination of encryption and authentication
  - ✓ Authenticated encryption (CCM, GCM)
- Randomness
  - ✓ Entropy
  - ✓ Cryptographically secure PRNG
  - ✓ Key derivation function

Exercises & Reading

- Classwork (Exercise Sheet 9): due on Fri Nov 9, 10:00 PM
- Homework (Exercise Sheet 9): due on Fri Nov 16, 6:59 PM
- Reading: FSK [Ch7, Ch9]

End of Slides for Week 9