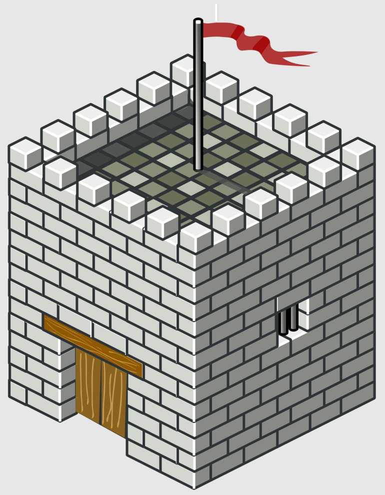
Foundations of Cybersecurity

VIII-Secure Channel and Randomness

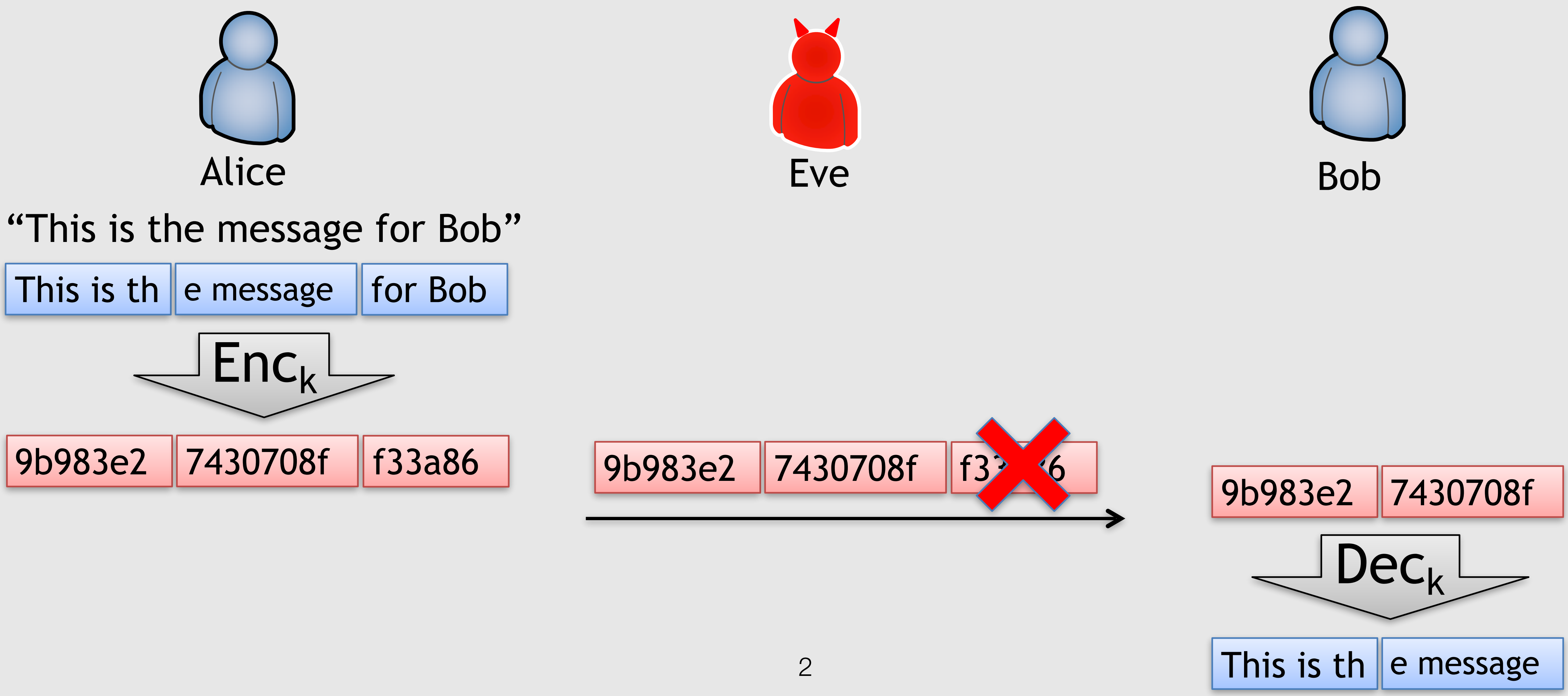


Paweł Szałachowski

2017

CPA-secure Encryption

CPA-security is about *confidentiality,* not *integrity*



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 is generated
- Messages or stream
  - Messages (much more practical and popular)

Security Property

- 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 $m'_1, m'_2, \ldots$
  - Eve does not learn anything about the messages (except their timing and size)
  - 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 the original sequence with removed zero or more elements.)

Authenticated Encryption

- Combination of encryption and authentication
  - Preventing from eavesdropping and **modifying** adversary
  - Using existing primitives
- How to combine them?
  - use CPA-secure encryption and secure MAC
    - use different keys for each primitive (derived from the session key)
  - order of authentication and encryption?

Encrypt-and-Authenticate

- Encrypt a message and authenticate the message. Transmit the ciphertext and the tag
  - derive Ke and Ka from K
  - ctxt = Enc_{Ke}(msg); tag = Mac_{Ka}(msg) ; send(ctxt || tag)
- Encryption and authentication can be done in parallel
- 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 (could lead to a privacy leak)

Authenticate-then-Encrypt

- Authenticate a message then encrypt both the message and the tag. Transmit the ciphertext.
  - derive Ke and Ka from K
  - tag = $Mac_{Ka}(msg)$; ctxt = $Enc_{Ke}(msg \mid\mid tag)$; send(ctxt)
- Tag is *invisible* to Eve
  - she 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 Ke and Ka from K
    - ctxt = Enc_{Ke}(msg); tag = Mac_{Ka}(ctxt) ; send(ctxt ll tag)
- According to theoretical results it is *secure*
- Efficiency: Bob never decrypts bogus messages
- Eve has valid (message, authtag) pairs

Which to use?

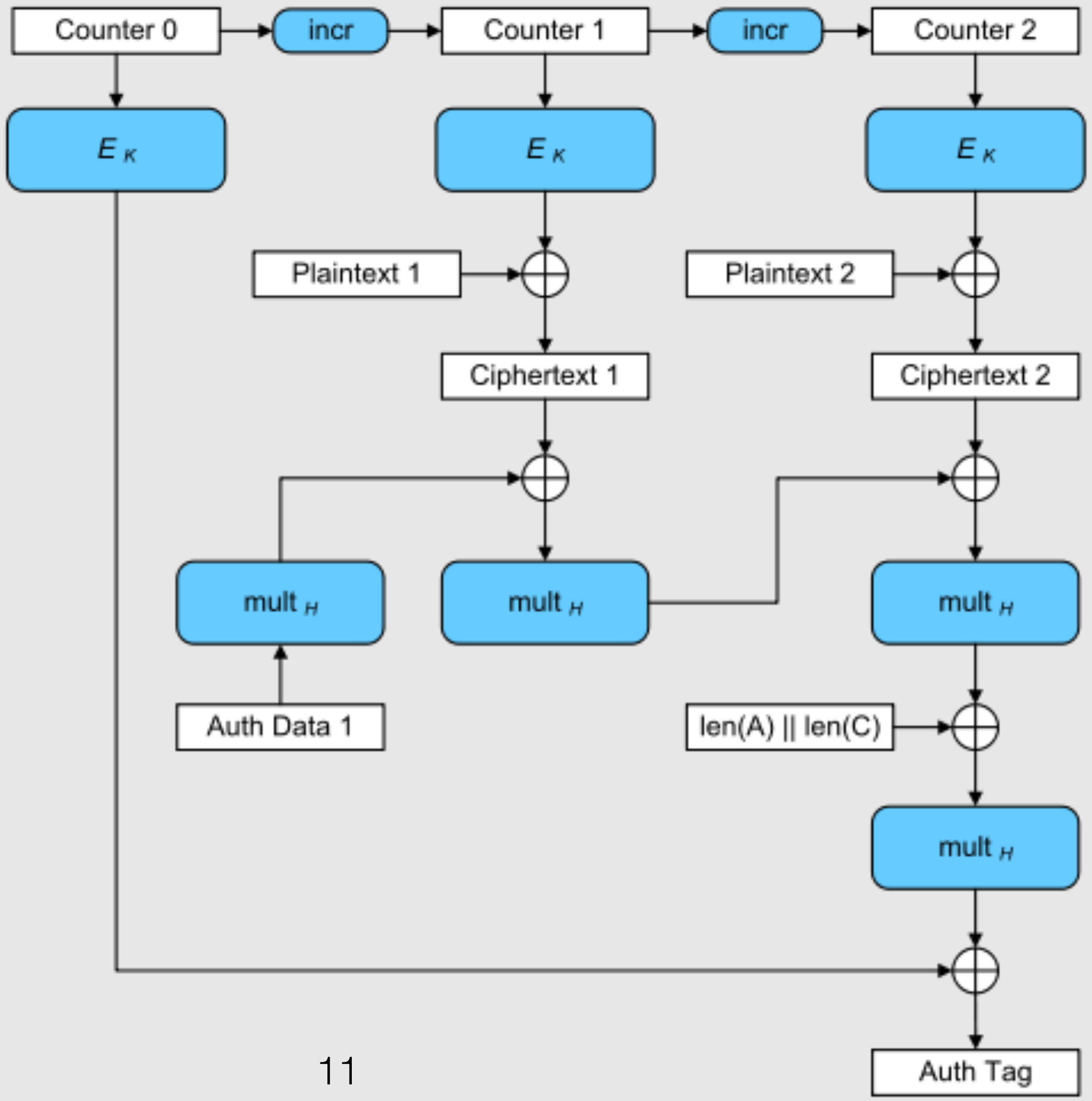
- Encrypt-and-Authenticate
  - used in SSH
- Authenticate-then-Encrypt
  - used in SSL/TLS
- **Encrypt-then-Authenticate**
  - encryption is CPA-secure and MAC is secure, then the construction is CCA-secure
  - used in IPSec

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
  - GCM
  - CCM
  - OCB
  - ...



Secure Channel Design

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

- Unpredictable to the attacker random data
- 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 32 bits of entropy

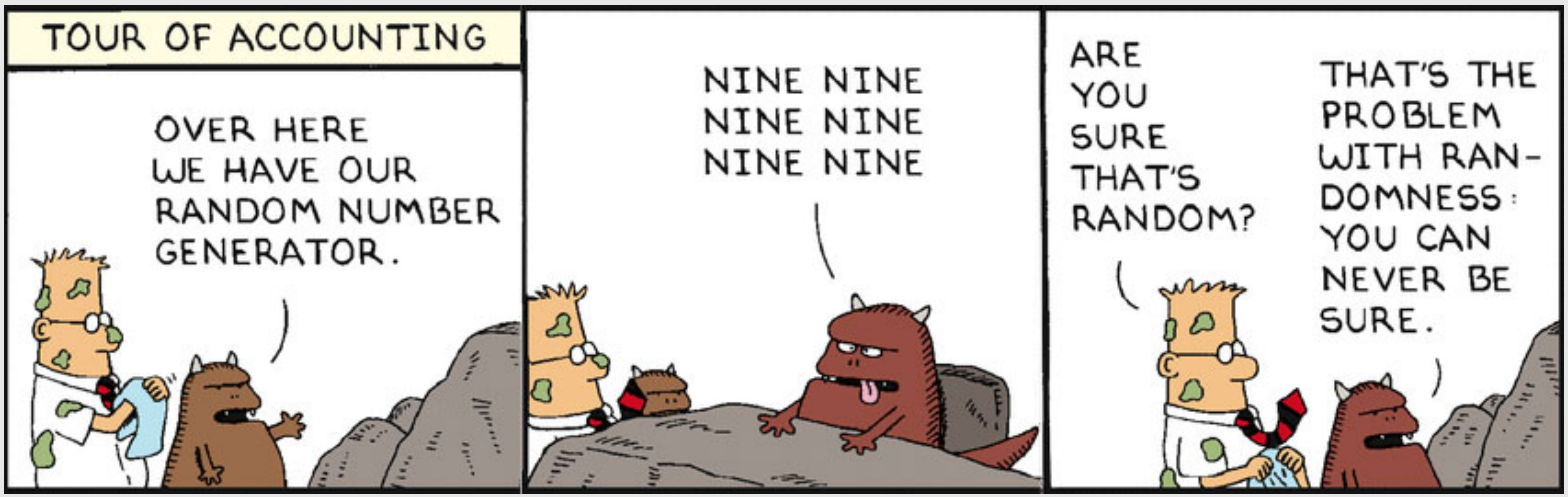
Real Randomness

- 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 (try to read from /dev/random)

Pseudorandomness

- Pseudorandom Number Generator (PRNG)
  - Numbers are generated deterministically, from a random seed
  - If a PRNG is used, the protocol is only secure as long as PRNG is not broken
  - Availability (generate immediately as many pseudorandom bits as needed, try read from /dev/urandom)

Cryptographically Secure PRNG



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*
    - if there is an entropy input while running, it should be infeasible to use knowledge of the input's state to predict future conditions of the CSPRNG state
- 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

Discussion & Classwork