CS 5430

Block Cipher Modes and Asymmetric-key Encryption

Prof. Clarkson

Spring 2017

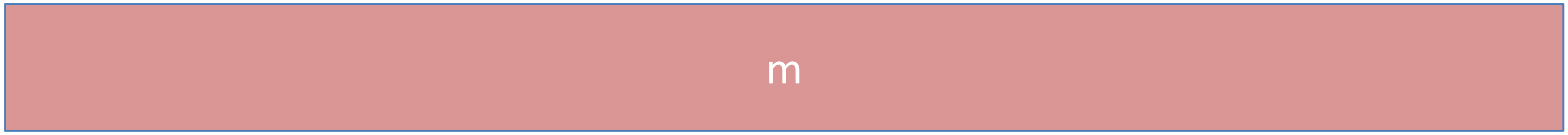
Review: block ciphers

- Encryption schemes:
  - Enc(m; k): encrypt message m under key k
  - Dec(c; k): decrypt ciphertext c with key k
  - Gen(len): generate a key of length len
- Defined for a particular block length
  - DES: 64 bit blocks
  - AES: 128 bit blocks
  - Messages must have exactly that length
- Every pair of principals must share a key
  - O(n^2) key distribution problem

BLOCK CIPHER MODES

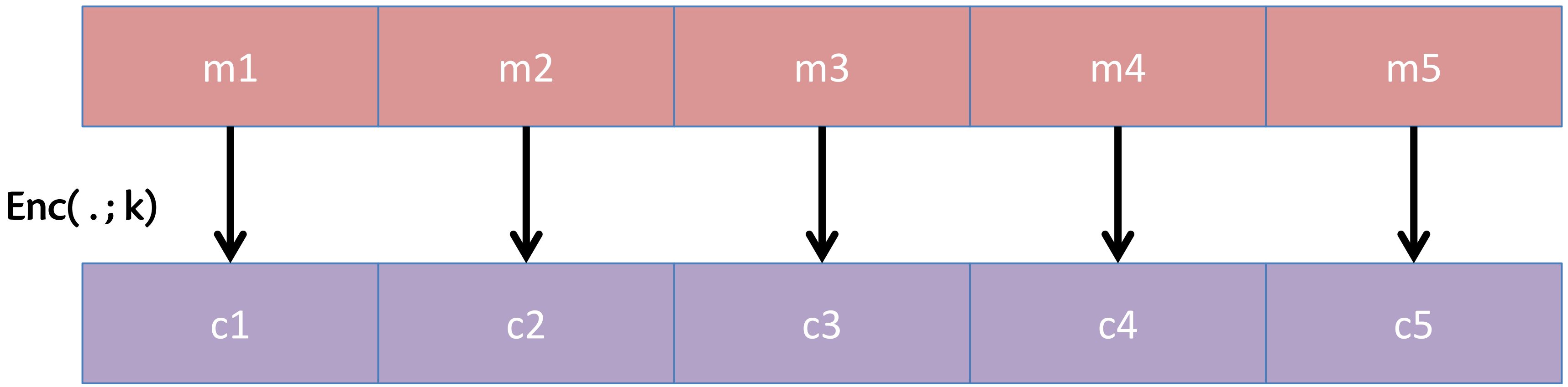
The obvious idea…

- Divide long message into short chunks, each the size of a block
- Encrypt each block with the block cipher

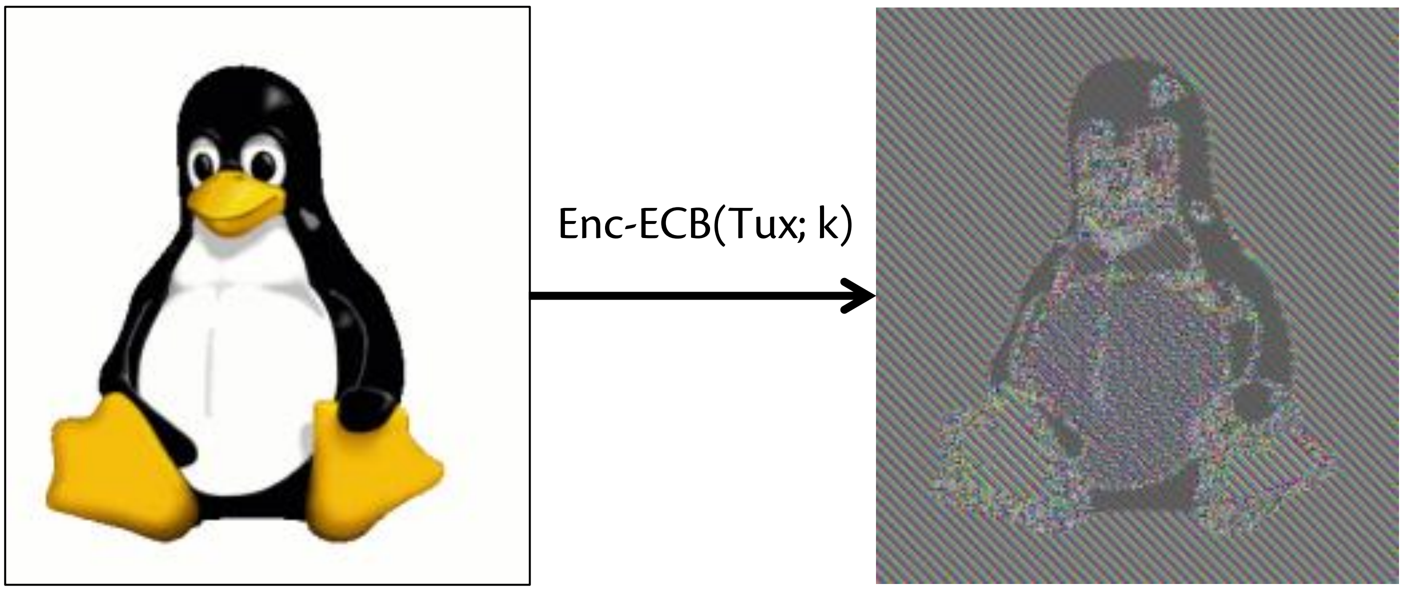


The obvious idea…

- Divide long message into short chunks, each the size of a block
- Encrypt each block with the block cipher



...is a bad idea

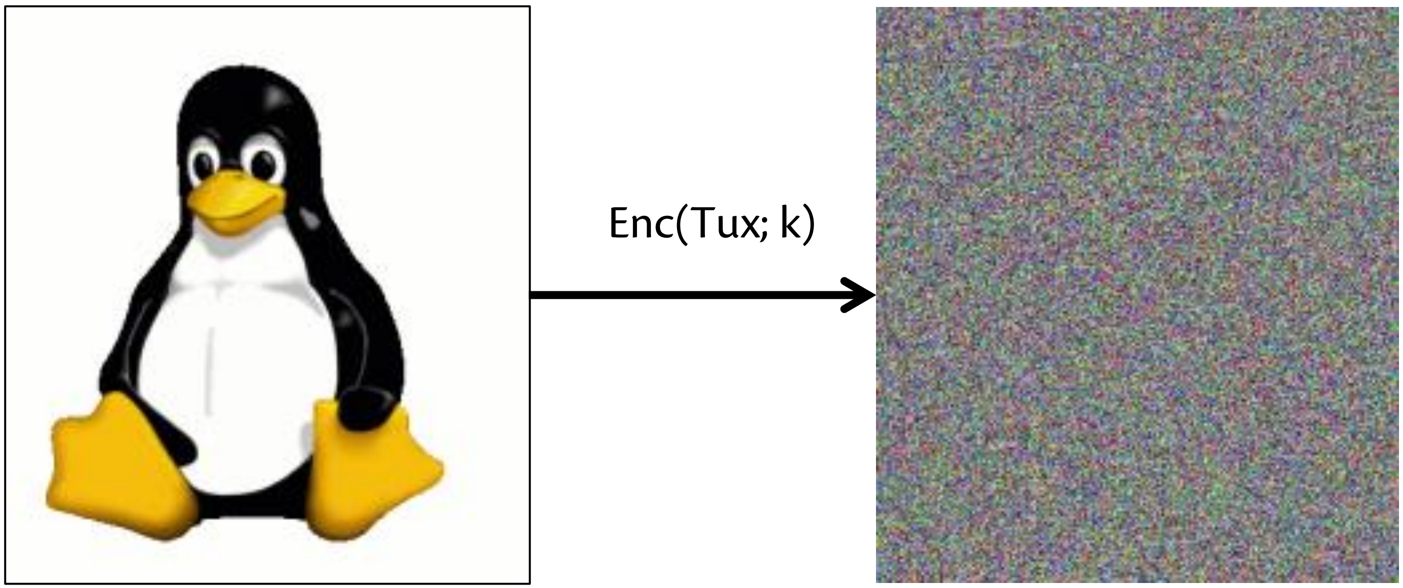


Called *electronic code book* (ECB) mode

Good modes

- Cipher Block Chaining (CBC) mode
  - idea: XOR previous ciphertext block into current plaintext block
- Counter (CTR) mode
  - idea: derive one-time pad from increasing counter
- (and others)
- With both:
  - every ciphertext block depends in some way upon previous plaintext or ciphertext blocks
  - so even if plaintext blocks repeat, ciphertext blocks don't
  - so *intra-message* repetition doesn't disclose information

Good modes

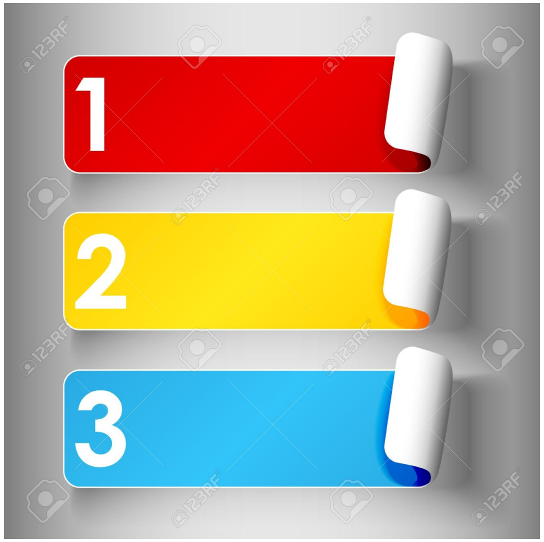


but what if you encrypt Tux twice under the same key?

Good modes

- Problem: block ciphers are *deterministic*: inter-message repetition is visible to attacker
- Both CBC and CTR modes require an additional parameter: a *nonce*
  - Enc(m; nonce; k)
  - Dec(c; nonce; k)
  - CBC calls the nonce an *initialization vector* (IV)
- Different nonces make each encryption different than others
  - Hence inter-message repetition doesn't disclose information

Nonces



A nonce is a number used once

Must be

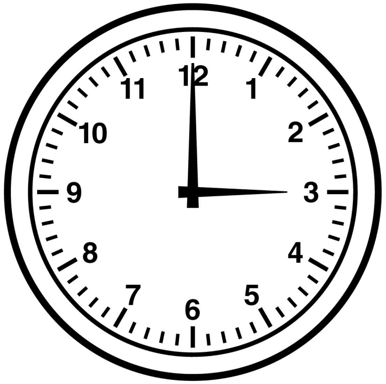
- **unique:** never used before in lifetime of system

and/or (depending on intended usage)

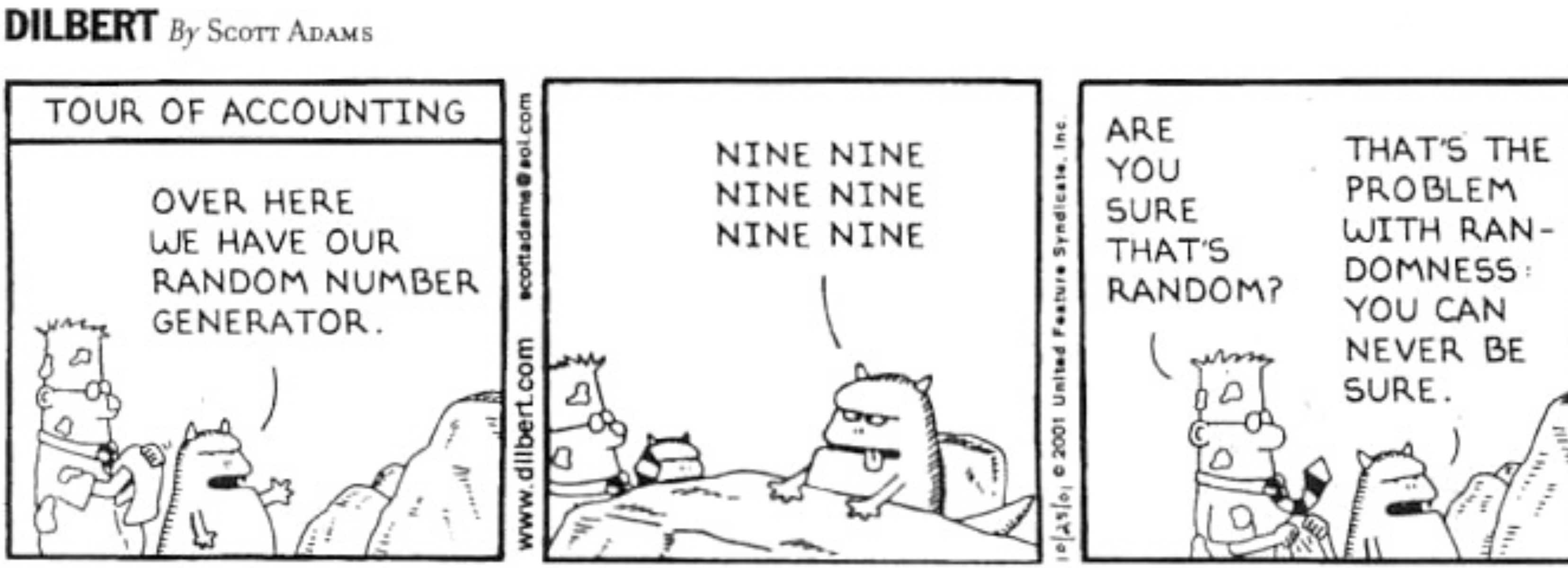
- **unpredictable:** attacker can't guess next nonce given all previous nonces in lifetime of system

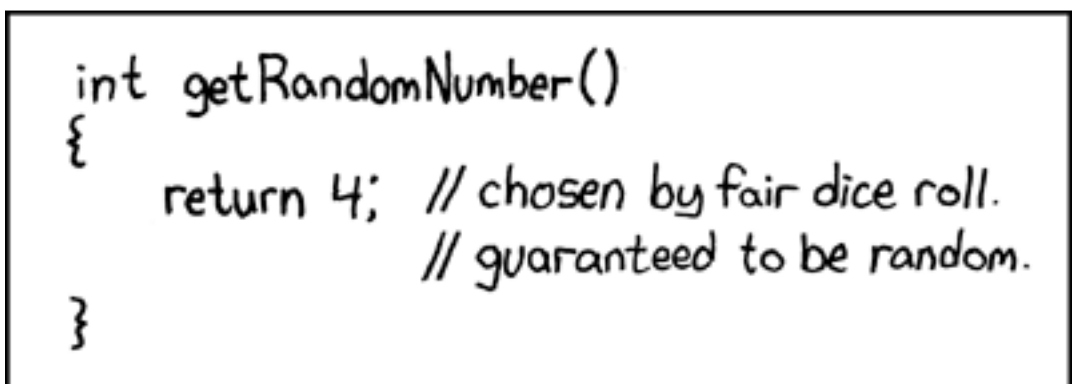
Nonce sources

- **counter**
  - requires state
  - easy to implement
  - can overflow
  - highly predictable
- **clock:** just a counter
- **random number generator**
  - might not be unique, unless drawn from large space
  - might or might not be unpredictable
  - generating randomness:
    - standard library generators often are not cryptographically strong, i.e., unpredictable by attackers
    - cryptographically strong randomness is a black art



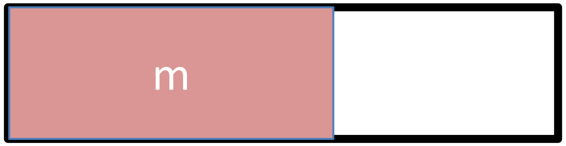
Random comics





Padding

What if the message length isn't *exactly* a multiple of block length? End up with final block that isn't full:



Non-solution: pad out final block with 0's (not reversible)

Solution: Let B be the number of bytes that need to be added to final plaintext block to reach block length. Pad with B copies of the byte representing B. Called *PKCS #5 or #7 padding.*

Block modes

Now we know how to encrypt messages of arbitrary length!

But we still have the quadratic key distribution problem...

ASYMMETRIC-KEY ENCRYPTION

Key pairs

- Instead of sharing a key between pairs of principals...
- ...every principal has a pair of keys
  - **public key:** published for the world to see
  - **private key:** kept secret and never shared

Key pairs

Terminology breakdown!

- private keys aren't necessarily personally-identifying
- symmetric-key crypto sometimes called "secret key" even though private keys also kept secret

Protocol to exchange encrypted message

```
1. A:   c = Enc(m; K_B)
2. A -> B:   c
3. B:   m = Dec(c; k_B)
```

key pair: **`(K_B, k_B)`**

- public key written with uppercase letter
- private key written with lowercase letter

Public keys

```
0. B:   (K_B, k_B) = Gen(len)
1.  ...
```

- All public keys published in "phonebook"
- So A can lookup B's key to send message
- Length of phonebook is O(n)
- So quadratic problem reduced to linear!

RSA

[Rivest, Shamir, Adleman 1977]

Shared Turing Award in 2002: *ingenious contribution to making public-key cryptography useful in practice*

RSA modulus

- Encryption and decryption are big integer operations modulo a large number called the *modulus*
  - Size of modulus bounds the size of keys and messages
  - Common modulus sizes: 1024, 2048, … bits
- Modulus is itself a product of two large primes
  - One way to break RSA would be to efficiently factor such numbers
  - Largest challenge broken so far is 768-bit modulus [2010]
  - Shor's algorithm factors in polynomial time on a quantum computer
    - largest factorization so far is of the number 56153 (i.e., 16 bits)
    - motivates work on *post-quantum cryptography*

Textbook RSA is insecure

- *Deterministic*: given same plaintext and key, always produces the same ciphertext
- Several other attacks, too [Katz & Lindell 2008, section 10.4.2]
- **Solution:** incorporate a nonce in the message before encrypting
  - Called *padding* but *encoding* might be a better term
  - Don't implement yourself; use OAEP implementation in your crypto library (Optimal Asymmetric Encryption Padding)

Elgamal

Taher Elgamal [1985]

Elgamal

- Like RSA:
  - Big integer operations modulo a large number
  - Common modulus *(group)* sizes: 1024, 2048, … bits
- Unlike RSA:
  - Key size can be much smaller than group size, which can speed up some operations
  - Elgamal encryption is *probabilistic:*
    - Given same plaintext and key, different calls to Enc produce different ciphertexts with high probability
    - Choice of a nonce is built-in to algorithm instead of part of padding
  - Factoring isn't relevant
    - One way to break Elgamal is by taking discrete logarithms

Key lengths

Again, various recommendations for strength summarized at https://www.keylength.com/en/

Problems of length

- Asymmetric encryption uses big integers, not byte arrays
  - all messages must be encoded as integers
  - modulus dictates maximum integer that can be encrypted
  - big integer operations are slow
    - say, **1 to 3 orders of magnitude slower** than block ciphers
- So the problems we had before crop up again...
  - what if message length is too short?
    - actually that's okay: a small integer is still an integer
  - what if message length is too long?
    - in theory could use block modes like with symmetric encryption
    - in practice, that's too inefficient...

HYBRID ENCRYPTION

Hybrid encryption

- Assume:
  - Symmetric encryption scheme (Gen_S, Enc_S, Dec_S)
  - Asymmetric encryption scheme (Gen_A, Enc_A, Dec_A)
- Use asymmetric encryption to establish a shared session key
  - Avoids quadratic problem, assuming existence of phonebook
  - Session key will be short, so avoids inefficiency
- Use symmetric encryption to exchange long plaintext encrypted under session key
  - Gain efficiency of block cipher and mode

Protocol to exchange encrypted message

```
0.  B: (K_B, k_B) = Gen_A(len_A)
1.  A: k_s = Gen_S(len_S)
       c1 = Enc_A(k_s; K_B)
       c2 = Enc_S(m; k_s) //mode
2.  A -> B: c1, c2
3.  B: k_s = Dec_A(c1; k_B)
       m = Dec_S(c2; k_s)
```

Session keys

- If key compromised, only those messages encrypted under it are disclosed
- Used for a brief period then discarded
  - *cryptoperiod:* length of time for which key is valid
  - in this case, for a single (long) message
  - not intended for reuse in future messages
  - only intended for unidirectional usage:
    - A->B, not B->A
    - why? A chose the key, not B

Encryption

- We can now protect confidentiality of messages against Dolev-Yao attacker
  - efficiently, thanks to hybrid of symmetric and asymmetric encryption
  - assuming existence of phonebook of public keys
- But what about integrity…?

Upcoming events

- [today] A1 due, A2 out
- [Mon] Feb Break

Few false ideas have more firmly gripped the minds of so many intelligent men than the one that, if they just tried, they could invent a cipher that no one could break. – David Kahn