**Unit 2: Applications of Symmetric Cryptography**

Applications of Symmetric Ciphers

=================================

[ Recap ]

Unit 1 we learned about symmetric ciphers; for the rest of the course, we are nearly always going to view our symmetric ciphers are a black boxes that provide encrypt and decrypt functions. In this unit, we’re going to look at ways to use symmetric ciphers to solve problems.

We assume:

k -> c

m

Even if the cipher is perfect, we have no security if the key k is predictable. Ideally, k is randomly selected from |K|.

**Generating Random Keys**

What is randomness? This is a deep philosophical question. I’m not much of a philosopher, so let’s try a quiz instead. Which of these is the *most* random?

[]

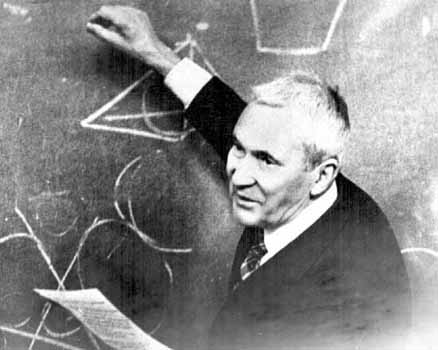
[]

[]

[]

Definition of “Randomness”

**Kolmogorov complexity Andrey “**kohl-moh-**GOH-**rawf”

****

**complexity of a sequence s is:**

**K(s) = length of shortest possible description of s**

**~ length of shortest possible program that can produce s**

[Quiz]

For a given sequence s, is there a way (in theory) to compute K(s)?

[ ] We can compute K(s) = len(“print ” + s)

[ ] There are a finite number of programs of any length n. So run,

n = 1

while True:

for p in all\_programs\_of\_length(n):

if p() == s: return n

n = n + 1

[ ] Yes, but its very difficult.

[ ] No.

[ answer ]

Showing (a) doesn’t work isn’t enough to prove there isn’t a way to solve this.

I’m not going to go through a full proof that K(s) is uncomputable, but the Berry paradox should give you some good idea why it this is the case:

“What is the smallest integer that cannot be defined in 10 words?”

Is there a way to use statistical tests to determine if a sequence is random?

[ ] Yes

[ ] Yes, but we don’t know the right statistical tests yet.

[ ] No, statistical tests can only show lack of randomness.

No way to prove randomness – only statistical tests to prove non-randomness!

Ideal test:

For numbers in range 0...2^n-1, an observer with the first m – 1 numbers, cannot guess the mth with probability better than 1/2^n.

“What is the sequence with smallest K(s) that passes all statistical tests?”

Physically random events

ask user to make some? - show GPG key generation

doesn’t work well to TLS!

/dev/random

Pseudo-Random Number Generation: Algorithms for Producing "Random"

Numbers

"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." - John von Neumann

unpredictability:

**Storing an Encrypted File**

**=====================**

First application we’re going to talk about is storing and encrypted file:

[ draw picture ]

Straight forward solution:

divide file into blocks: m = m\_0 m\_1 … m\_n-1

store c = E\_k(m\_0) || E\_k(m\_1) || …

E = block cipher (e.g., AES: 128-bit blocks = 16 bytes)

[quiz] Assuming cipher is perfect, what does someone who captures the file, but not the key, learn from intercepted message?

[] Nothing at all

[x] If m\_i == m\_j, attacker learns that blocks i and j are equal.

[] If m\_i == 0, attacker learns the value of k.

[x] Length of message

E\_k(x) is deterministic: so if two message blocks are identical E\_k(m) = E\_k(m)

128-bit blocks: only 16 characters

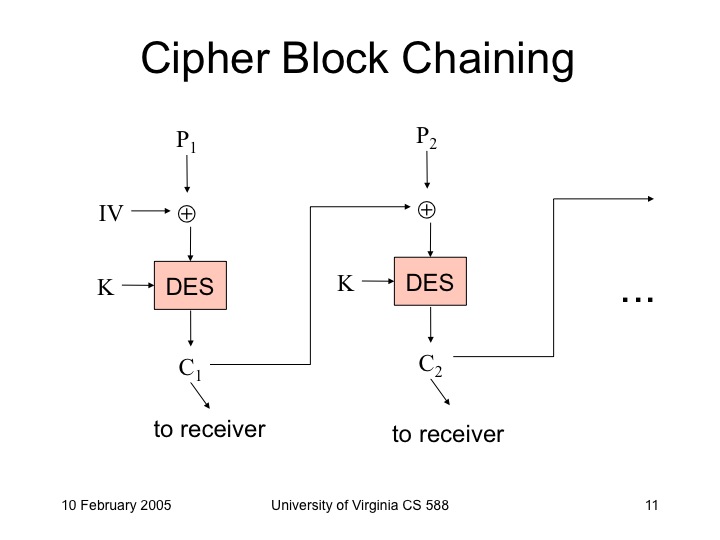
Other problems:

- E might not be perfect --- this gives attacker access to lots of ciphertext encrypted with same key

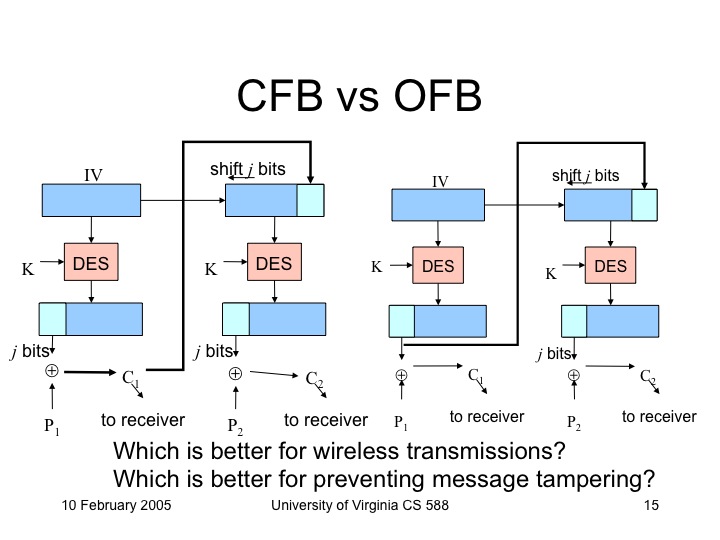
- Attacker can move blocks around, potentially changing meaning of message in a predictable way.

This is called: “Electronic Codebook Mode” – its actually what we just used for the PRNG!

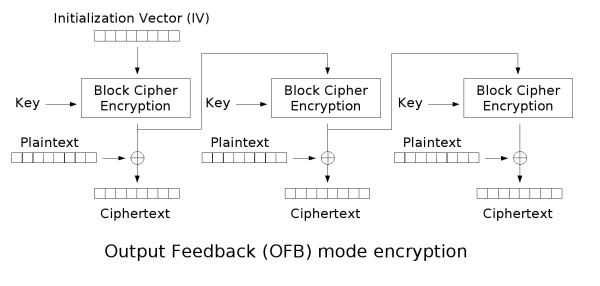
**Cipher Block Chaining**

****

Cipher feedback mode vs. Output feedback mode



THIS is wrong! OFB as NIST recommends is below:



**CTR Mode**

CFB: IV need not be secret, but should be unpredictable (resist dictionary attacks)

OFB: IV must be unique

[quiz]

which is best if you want to take advantage of many processors to quickly encrypt a large file?

CTR, CBC

[quiz]

which ones makes the final encrypted output depend on all the plaintext?

OFB, CFB

=> this property is useful for making cryptographic hash functions!

**CFB can decrypt in parallel! < good HW question**

**Padding**

**Key**

cipher modes

**Cryptographic Hash Functions**

First, I’m going to motivate crypto hash functions with a simple example.

Alice and Bob want to toss a coin to decide who’s turn it is to charge the robo-dog. [draw]

But, they aren’t in the same place so need to do the coin toss over the phone. Can they do it fairly?

Try 1:

Alice: I’ll toss the coin, you call it.

Bob: I call “tails”.

Alice: Tosses the coin. Sorry, it was “edge”.

Try 2:

Alice: x <- {0, 1} k <- random key

m = E\_k(x)

Is x = 0 or x = 1?

Bob: guess = 1

Alice: you’re wrong, k = 01100101001010

How hard would it be for Alice to cheat?

() Impossible

() She would need to find a key k’, where E\_k’(‘0’) = E\_k(‘1’)

(x) She would need to find a pair of keys, k\_0 and k\_1 such that E\_k0(‘0’) == E\_k1(‘1’) or E\_k0(‘1’) == E\_k1(‘0’)

() It would be as easy as cheating in the first game.

This might be hard…depending on what E is.

**Cryptographic Hash Function**

different from encryption: no need to decrypt!

h = H(x)

Key properties we need:

* **Pre-image resistance:** given h, it is hard to guess
* **Weak collision resistance:** given h, it is hard to find an x’ such that H(x’) = h
* **Strong collision resistance:** it is hard to find any pair, x, y, such that H(x) = H(y)
* **Compression:** | H(x) | << | x |

Birthday “Paradox”

Application: User Authentication

Passwords, USENIX Password scheme,

one-time passwords S/Key

**Strong Collision Resistance**

It is hard to find a pair of values,

encrypting passwords

merkle trees

HW2 Ideas

- Importance of padding: identifying web sites by size of encrypted traffic