The exam will cover topics starting from Stream Ciphers (Lecture 9 from 9.6 onward) to Hashing (Lecture 15, which we will be covering tomorrow).

<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture9.pdf>

<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture10.pdf>

<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture11.pdf>

<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pdf>

<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture13.pdf>

<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture14.pdf>

<https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture15.pdf>

**Stream Ciphers:**

**RC4 Stream Cipher Algorithm:**

9.7 in notes

Other important topics include:

**Random Number Generation**:

INFO FOUND IN LECTURE 10, 10.4 PAGE 26

To be considered truly random, a sequence of numbers must

exhibit the following two properties:

Uniform Distribution: This means that all the numbers in a

designated range must occur equally often.

Independence: This means that if we know some or all the number up

to a certain point in a random sequence, we should not be able to

predict the next one (or any of the future ones).

**10.5 PRNGs, Linear Congruential Generators**:

This algorithm is used for generating pseudorandom numbers for non-security applications

Starting from a seed X0, a sequence of pseudorandom numbers

X0, X1, ...., Xi, ... is generated using the recursion:

Xn+1 = (a · Xn + c) mod m

where

m the modulus m > 0

a the multiplier 0 < a < m

c the increment 0 = c < m

X0 the seed 0 < X0 < m

10.5 Explains why this is not a very secure way of generating numbers (page 29 starts this explanation, pg 31 has a paragraph at the end as well)

When a PRNG sequence is secure:

A pseudorandom sequence of numbers is cryptographically secure if it is difficult for an attacker to predict the next number from the numbers already in his/her possession.

**X9.31**:

10.6

Is there a price to pay for the cryptographic security of ANSI X9.17/X9.31?

Yes, it is a much slower way to generate pseudorandom numbers. That makes this approach unsuitable for many applications that require randomized inputs

**CSRNGs**:

Cryptographically secure pseudorandom number generator

10.6

**TRNGs**:

True random number generator

Works without seeds, unlike PRNGs

Based on the fact that only analog phenomena can be trusted to produce truly random numbers

Entropy source: any source that is capable of yielding a TRULY random stream of 1's and 0's

10.9 discusses software entropy sources

**Primality Testing:**

**11.5 Miller-Rabin Algorithm for Primality Testing:**

<http://homepages.math.uic.edu/~leon/mcs425-s08/handouts/Rabin-Miller-Examples.pdf>

<https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/>

**11.5.3 Goes into more info about the proof of the Miller-Rabin Algorithm**

**11.6 AKS Algorithm for Primality Testing**

**11.6.2 Computational Steps for AKS Algorithm**

**RSA and relevant concepts (e.g. Chinese Remainder Theorem):**

**11.3 Totient Function:**

The totient of a number is the number of positive integers less than or equal to n that are coprime to n

Totient(n) = (p-1)(q-1)

**11.7 Chinese Remainder Theorem:**

<https://www.youtube.com/watch?v=ru7mWZJlRQg>

<https://www.youtube.com/watch?v=EHDEvFuYPRQ>

**12.2 RSA Algorithm**

**Diffie Hellman:**

**13.5 Explanation of this algorithm:**

Lecture 13 page 49 describes the man-in-the-middle attack on the Diffie Hellman algorithm

So basically if I understand correctly the only reason that Diffie Hellman has perfect forward secrecy is because the session key is changed often whereas it is not changed very often (if at all) in RSA.

<https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/#:~:text=Step%201%3A%20Alice%20and%20Bob,exchange%20public%20numbers%20Step%205%3A>

**ECC:**

14.2 Starts this discussion

**Discussion on Certificate Authority:**

<https://piazza.com/class/kkr68mv01575p?cid=247>

**Paradox Problem:**

<https://en.wikipedia.org/wiki/Birthday_problem>

15.5 Discusses this as well

**Topics from student on Piazza:**

- Some small encoding/decoding in RC4

- A mock key distribution simulation with some encoding/decoding

- Calculate next digit in random number sequence

- Perform primality test for large number and show work

- Prove that a primality test is (not) necessary and sufficient

- Calculate a discrete logarithm for a particular multiplicative group

- RSA derivation/encoding/describing how to break it

- Independently generate shared key using RSA or Elliptic Curve with Diffe Hillman

- Calculate sum or multiplication of two large numbers modulo n using CRT

- Something with hashing? Idk. I'm just going to do the homework by Tuesday and kill two birds with one stone there.