Popa & Weaver
Spring 2019

CS 161
Computer Security

Discussion 3

Week of February 11, 2019: Cryptography I

Question 1 ***Block Cipher Potpourri*** **(20 min)**

(a) What is the difference between IND-KPA and IND-CPA?

> **Solution:** Assuming Alice has an encryption scheme E and a secret key k, the steps for IND-KPA are as follows:
>
> 1. Mallory sends Alice two distinct plaintext chosen messages $M_1,\ M_2$
> 2. Alice randomly selects M_b between $M_1,\ M_2$ and encrypts it $C = E(M_b, k)$
> 3. Mallory now guesses if C corresponds to M_1 or M_2
>
> If Mallory is able to guess with a probability $> \frac{1}{2}$ then, Alice's scheme is not IND-KPA. The difference between IND-KPA and IND-CPA is that before step 1, Mallory is able to ask Alice to encrypt a polynomially bounded number of messages which she can also do after receiving back C before having to make her guess. The reason that Mallory may only encrypt a polynomial number of messages is that otherwise she could trick Alice into enumerating all possible outputs for a given message, and the game becomes trivial.

(b) Are block ciphers IND-CPA?

> **Solution:** No, as mentioned in lecture, block ciphers alone are **not** IND-CPA because they are deterministic and will always give the same output for the same input. The proposed solution is to create schemes using block ciphers that add entropy to each message such as the IV in CBC (cipher block chaining) or the nonce in CTR (counter) modes. There is a scheme just using a block cipher called ECB (electronc codebook) mode where encryption is done on a block by block basis without incorporating any additional entropy.

(c) What are good possible sources of entropy for key generation for a block cipher?

- The computer's clock time (assumed in seconds)
- The Parent Process ID $\oplus$ my Process ID $\oplus$ time
- Hardware noise generator
- Hardware noise generator $\oplus$ time

- 101010101... $\oplus$ Hardware noise generator

Solution:

- No, a computer clock counts the number of seconds from a given point in time (traditionally the epoch of unix), and because of this, the entropy of such a request is dramatically reduced if you can narrow down the window of time when such a call was made. If you are able to narrow down the year in which a call to time was made, the entropy is reduced to 25 bits, narrowing it down to a month is 22 bits, and narrowing it down to the day is 17 bits.
- No, time as outlined above is not a sufficient source of entropy and with the addition of process IDs remains insufficient. This example was actually inspired by a previous implementation of Netscape's SSL and you can read up on the paper published on its insecurity by our very own David Wagner. https://people.eecs.berkeley.edu/~daw/papers/ddj-netscape.html
- Yes, the hardware implemented (psuedo) random number generators are traditionally very strong sources of entropy in today's computers because they incorporate a physical source for their randomness. Other great examples that have been used are phsyical dice rollers, weather patterns, lava lamps, etc.
- Yes, given a proper source of entropy we can still combine it with a weak source without losing this randomness. This does rely on the fact that we are using a one-to-one function such as XOR, otherwise if we had instead used a bitwise AND or OR, we would have been removing the entropy provided by the hardware.
- Yes, this is just an extrapolation of the previous example. Even with a known value being included with our actual source of randomness, if we remove the 101010101... bitstring, we are still left with enough entropy to provide us with a good key.

(d) Why does a block cipher need to be a permutation?

Solution: A block cipher needs to be one-to-one so that it is invertible, and if it wasn't a permutation then more than one input could result in the same output which means that a ciphertext couldn't be decrypted.

Question 2 ***PRNGs and stream ciphers*** **(20 min)**

(a) Pretend I have given you a pseudo-random number generator R. R is a function that takes a 128-bit seed s, an integer n, and an integer m, and outputs the m^{th} (inclusive) through m^{th} (exclusive) pseudo-random bits produced by the generator when it is seeded with seed s. Use R to make a secure symmetric-key encryption scheme. That is, define the key generation algorithm, the encryption algorithm, and the decryption algorithm.

Solution:

- **Key generation.** Generate a random 128-bit key $K \in \{0,1\}^{128}$.
- **Encryption.** Let j be the latest index we have used from our PRNG. We start with $j := 0$ and maintain the state of j for subsequent encryptions. Let L be the number of bits in message M. Then,

  $$E(K, M) = R(K, j, j + L) \oplus M.$$

  After every encryption, j must be incremented by L.
- **Decryption.** Define j and L as above. We have

  $$D(K, C) = R(K, j, j + L) \oplus C.$$

(b) Explain how using a block cipher in counter (CTR) mode is similar to the scenario described above.

Solution: CTR mode is similar to a stream cipher mode. It uses the key to generate a pseudo-random stream of bits. This random stream is then XORed with the message to form the ciphertext.

In CTR mode, there is no computational dependency between the rounds, which enables an efficient parallel computation. Additionally, the IV is replaced with a nonce and counter.

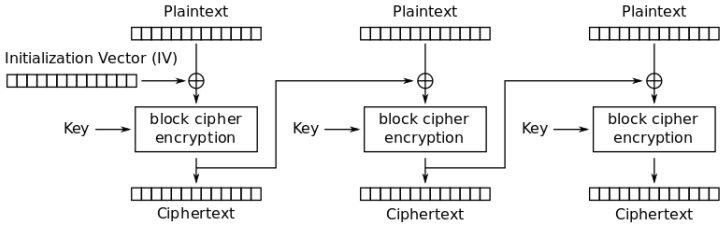
Nonce and counter are encrypted with key K to produce the random stream that for a given element of the plaintext P_i is XORed with P_i to produce the ciphertext C_i. In summary, CTR is defined as:

$$\begin{aligned} R_i &:= E(K, \text{Nonce}||i) \\ C_i &:= P_i \oplus R_i \\ P_i &:= C_i \oplus R_i \end{aligned}$$

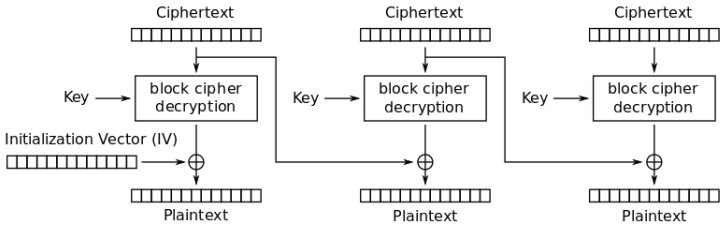
where $||$ denotes concatenation.

Question 3 ***Block cipher security and modes of operation*** **(20 min)**

As a reminder, the cipher-block chaining (CBC) mode of operation works like this:



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

The output of the encryption is the ciphertext concatenated with the IV that was used.

(a) Does the initialization vector (IV) have to be non-repeating? Why?

> **Solution:** Yes, a fundamental criteria for IVs is that they cannot repeat. This prevents CBC from degenerating into a deterministic encryption algorithm (such as ECB mode). In deterministic encryption schemes, if we encrypt the same message multiple times, the ciphertexts will be identical each time. Unfortunately, deterministic encryption schemes can leak a lot of information. Consider the example from lecture where the Linux penguin is encrypted using ECB-mode. Even though all of the colors get mapped to new encrypted values, we can still clearly see the penguin since pixels of the same color share the exact same value after encryption.
>
> To see why CBC-mode with a repeating IV becomes deterministic, consider the simple case of always using an IV of 0 and encrypting the same message twice. In this scenario, the first ciphertext block will always be $E_k(m[0])$, which will be the same value for two identical plaintext messages; this will then propagate to subsequent blocks and cause all of the ciphertext blocks to become equivalent.
>
> When we use non-repeating IVs for CBC-mode, even if we encrypt the same message multiple times, CBC-mode will generate distinct and random-looking ciphertexts each time.

(b) Is a non-repeating IV enough? Imagine you sequentially picked IVs from a list of non-repeating, but publicly-known, numbers, e.g., *A Million Random Digits with*

100,000 Normal Deviates (RAND, 1955).

Say Alice encrypts the one-block long message m_1 with initialization vector IV_1 to get C_1 and encrypts m_2 using IV_2 to get C_2. She gives these to Mallory and challenges her to tell which C came from which m.

Mallory knows that Alice's next IV will be IV_3, and can ask Alice to encrypt messages for her (a *chosen plaintext attack*). Can Mallory distinguish the two ciphertexts?

Solution: Yes. Mallory asks Alice for the encryption of $m_1 \oplus IV_1 \oplus IV_3$. When Alice runs CBC, the output will be the block cipher output for $m_1 \oplus IV_1$. But that's just C_1! So for CBC an IV must also be *unpredictable*, which is to say it has to be kept secret until after the encryption is done.

Thus, IVs for CBC-mode encryption have two necessary criteria: (1) they must not repeat across messages and (2) they must be unpredictable. It turns out we can satisfy both criteria (with high probability) if we just generate a random IV for every message we encrypt.