Weaver
Fall 2020

CS 161
Computer Security

Discussion 3

Cryptography I

Question 1 ***Block Cipher and Entropy Potpourri*** ()

(a) Explain how an adversary can always win the IND-CPA game against a deterministic encryption algorithm. Given identical plaintext, a deterministic encryption algorithm will produce identical ciphertext.
(Corollary: AES block cipher and ECB mode of operation, which are deterministic, are IND-CPA insecure.)

Solution: An adversary can provide two plaintexts A and B to be encrypted. Adversary gets back X, which is an encryption of either A or B. Then, the adversary requests an encryption of A again and compares it with X. If two are the same, X is the encryption of A, and vice versa.

(b) 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.

(c) What are good possible sources of entropy for key generation for a block cipher? Assume a hardware noise generator is a good source of entropy on it's own (these usually incorporate physical sources for randomness).

- The computer's clock time (assumed in seconds)
- The Parent Process ID $\oplus$ my Process ID $\oplus$ time
- Hardware noise generator $\oplus$ a vector of 0s
- Hardware noise generator $\oplus$ time
- Hardware noise generator $\wedge$ a vector of 0s
- Hardware noise generator $\wedge$ time

($\oplus$ and $\wedge$ denote bitwise XOR and AND respectfully)

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, 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, 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.
- No, the output will always be the 0 vector
- No, even though the HNG is a good source of entropy, the bitwise AND removes entropy since it is not injective. Essentially, the output will be biased towards 0

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

(a) Suppose you have access to function R that takes a 128-bit seed s and integers n, m as input. R outputs the n^{th} (inclusive) through m^{th} (exclusive) bits produced by the a pseudorandom generator $PRNG$ when it is seeded with seed s.

$$R(s, n, m) = PRNG(s)[n : m]$$

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.

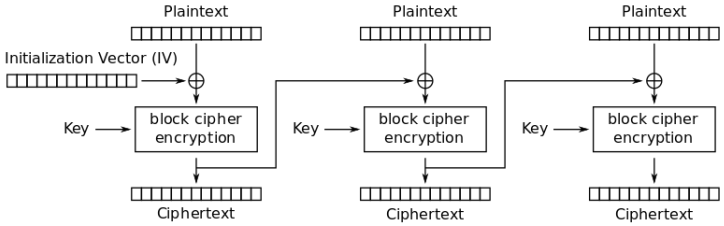
Nonce and counter are encrypted with key K to produce the random stream that for a given element of the plaintext P_i is XORed with P_i to produce the ciphertext C_i. In summary, CTR is defined as:

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

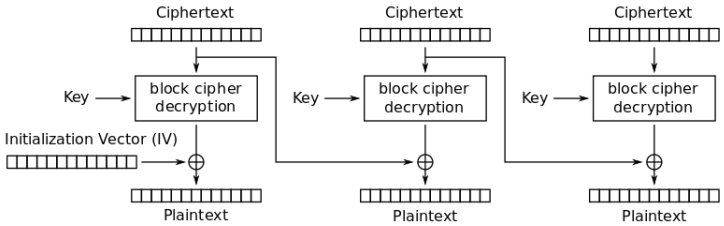
where $||$ denotes concatenation.

Question 3 ***Block cipher security*** ()

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) What happens if two messages are encrypted with the same key and nonce? What can the attacker learn about the two messages just by looking at their ciphertexts?

> **Solution:** If IV is reused in AES-CBC, the attacker can determine if two messages have identical prefix, up to but not including the first block containing the difference. This is because the nth plaintext block affects the input to nth input to the block cipher, and any difference in the plaintext block results in a completely different block cipher output.
>
> 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) If the random number generator used for IV creation is sabotaged, an attacker may be able to predict IVs used to encrypt future data. If this is the case, can an attacker win the IND-CPA game against AES-CBC mode of operation?

Specifically, an attacker provides one-block long plaintext messages P_1 and P_2 to an oracle, which encrypts one of the plaintexts using IV_1.

Can an attacker determine the plaintext used for the first encryption by requesting encryptions of a few chosen plaintexts?

Assume the attacker knows IV_n for any n.

Solution: Yes. Mallory asks Alice for the encryption of $m_1 \oplus IV_1 \oplus IV_2$. 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.