Nick Weaver
Fall 2018

CS 161
Computer Security

Discussion 3

Week of September 10, 2018: Cryptography I

Question 1 ***Activity: Cryptographic security levels*** **(20 min)**

Say Alice has a randomly-chosen symmetric key $K \in \{0,1\}^{128}$ (that is, a 128-bit key) that she uses to encrypt her messages to Bob.

Eve is very suspicious of these messages and would like to brute-force guess the key. She does this by getting a pair (M, C) where she knows that C is Alice's encryption of M. She keeps guessing keys k until $E_k(M) = C$.

(a) **Probability review.** How many attempts does Eve expect to have to try in order to guess Alice's key, if she guesses keys completely at random (with repetition)? What about if she guesses in order (without repetition)?

(b) Eve sits down at her computer and starts brute-forcing the key. If her computer can attempt 1 billion keys per second, how much time does Eve expect to wait?

How long of a time is this?

(c) Eve decides to enlist the help of her friend Ed, who works at the NSA and has access to a cluster of 1,000,000 servers[1] running in parallel that can each guess 10 billion keys per second.

Now how long will Eve be waiting? How much faster is this?

(d) Alice starts getting worried about Eve and decides to increase the key size to 256 bits. Bob claims this is pointless since the key is only twice as big as before, and so Eve needs only double as much time as before. Is he right?

(e) **Bonus.** The quantum computing *Grover's algorithm* lets you brute force a function using only $\mathcal{O}(N^{1/2})$ evaluations, instead of the $\mathcal{O}(N)$ required in classical computing.

If Eve gets a quantum computer, now how many attempts does Eve have to try for a 128 bit key? How much faster is this?

If we wanted to increase key size to combat this, how much of an increase do we need? Should we be concerned about possible future quantum computing attacks against symmetric-key cryptography?

Solution:

[1] This is estimated to be around the number of servers that Google has. https://what-if.xkcd.com/63/

(a) If Eve guesses at random, we get a geometric probability distribution, with $p = 2^{-128}$ (the probability of correct guess). The expected value is $1/p$, so Eve needs to make 2^{128} guesses.

Even if Eve guesses in a systematic way, like counting up from 0, she still needs to make approximately 2^{127} guesses on average, which is still almost as many as 2^{128} (just a factor of 1/2 away).

(b) 2^{128} nanoseconds, which is 3.4×10^{29} seconds, or 1.0×10^{22} years, or 8×10^{11} times the age of the universe.

(c) Even with this massive power increase (10 million times faster), Eve will still expect to wait 3.4×10^{22} seconds, or 1.0×10^{15} years, or $80,000$ times the age of the universe.

(d) No. Doubling the size of the key means Eve now has to make 2^{256} guesses, which is the *square* of 2^{128}, not double (twice 2^{128} is 2^{129}).

(e) This attack shortens the needed number of attempts to $2^{128/2} = 2^{64}$. This is much shorter: only 584 years at 1 ns per attempt (and merely 30 minutes using the setup in part (c))! But the attack can be entirely mitigated by just doubling the key size (like in part (d)). For this reason we mostly don't worry about quantum attacks on symmetric-key crypto.

Question 2 ***PRNGs and stream ciphers*** **(10 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 n^{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.

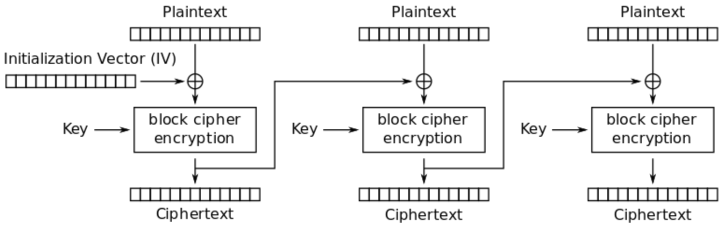
Nonce and counter are encrypted with key K to produce the random stream that for a given element of the plaintext P_i is XORed with P_i to produce the ciphertext C_i. In summary, CTR is defined as:

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

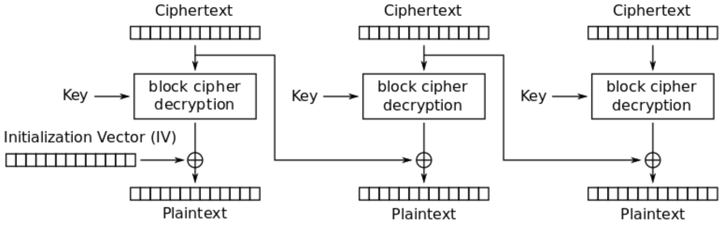
where || denotes concatenation.

Question 3 *Block cipher security and modes of operation* (15 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.