CS 558 – Network Security Notes

Note Taker: David Wang
Date: 1/24/17
Lecture #: 2

Going back to ancient times…

Cryptography techniques have existed since then (but all of those techniques can be broken now). For now, we'll be working on authentication and encryption in the coming weeks.

If you want to send a secure message to someone – what's the goal?
Encryption – to make sure only the parties intended can read the message, providing confidentiality of the message itself.
Authentication – providing confirmation that the message came from/for the intended person (often mixed up with "integrity", even while both authentication and integrity are necessary)

- Examples: ancient Egyptian seals on papyrus papers + a king's wax seal on a roll of paper + your own signatures when you sign something

Integrity, on the other hand, is to make sure that the message was unchanged en route to its intended parties.

(*Note:* To ensure authentication and integrity – "Message Authentication Code" (MAC), or something called digital signatures are used.)

Begin Goldberg's "channeling" of 237…

One of the oldest-known ciphers is called the "Caesar Cipher", from Julius Caesar's own time. Known as an example of a ***"shift cypher"***, it's an encryption technique that basically replaces each letter of a given message with the k^{th} letter that occurs after the given letter's position in the alphabet.

Example:

- "BR ZKR KBR PBRRNR" – the cipher text
- How do we map the cipher text to the plain text?
- Trial and error – can figure out that the 'R' in the cipher text actually corresponds to the letter 'E' – meaning the shift in the alphabet is actually 13 positions forward in the alphabet (using modulo of course, to wrap around the end of the alphabet)
- So, if we shifted every letter in the cipher text forward 13 positions in the English alphabet, what do we get as the plaintext message?
- "HE ATE THE CHEESE" – the plaintext
- In breaking the encrypted message this way, we've actually just committed a ***"plaintext recovery attack"***

Why was the above not very secure? Because you can already clearly see that 'E' is the most common letter in the English alphabet, and that so was 'R' in the example used above. This is known as ***"frequency analysis"***, by figuring out how to correspond the given structure of the encrypted text to the structure of a known language. We began with this example as an introduction to thinking about what makes a "good" encryption scheme.

In an encryption scheme, we have a ***"sender"*** and a ***"receiver"*** (typically always known as **Alice** and **Bob**, respectively). Consider a ***"symmetric encryption scheme"*** – where we have a secret key that is "symmetric", in the sense that both Alice and Bob have the key. This key allows both Alice to encrypt a message, and Bob to decode the encryption. Now, we have the grounds for talking in formal definitions:

Given:

- Plaintext message *m*
- Cipher text (i.e. encrypted message) *c*
- Encryption function, dependent upon some variable k, that takes in a message and returns its encrypted version: $Enc_k(m) = c$
- Decryption function, dependent upon some variable k, that takes in an encrypted message and returns its plaintext: $Dec_k(c) =$ *some message m'* (m-prime)
    - We denote m' vs. m since we don't know if the decryption was actually successful in recovering the plaintext of the original message
- An encryption scheme must have both encryption and decryption functions (otherwise, what's the point of encrypting something such that no one can recover it)

We define an ***encryption scheme*** as having two properties: 1) ***correctness***, and 2) ***security***.
Formally, we define "correctness" as such that $Dec_k(Enc_k(m)) = m$ (that is, if we decrypt the cipher text $c = Enc_k(m)$ returned from the encryption function that encrypted the original message m, we should be able to recover the original message m – so, m' == m.)

A ***hash function*** takes in an input space (of messages) that could be very large (like say a space of 2^{3000} combinations, that we call "3000-***bit-string***" messages). The input-space is 2^{3000} possible messages, and a hash function will take any input message of 3000 bits in length and encode it/map to a smaller output space of, let's say, 160-bit-string messages (so hash(some message of length 3000 bits) = some encrypted bit-string of length 160 bits).

Another example: take the hash function SHA256 –
Bit-strings of {0, 1}* (* = arbitrary length)
(maps → to)
Bit-strings of $\{0, 1\}^{256}$

In other words, SHA256 as a hash-function maps bit-strings of arbitrary length to bit-strings of length 256. For example, SHA256("Sharon") = some bit-string of length 256. So, SHA256 takes in an input of arbitrary length and maps it to an output space of 2^{256} possible bit-strings – such that each input maps to a unique (each possible input has a corresponding output in the output space), consistent (repeating the hash-function on a given input will always map to the same thing) output (since it is a function, by definition).

However, by itself, a hash function cannot be an encryption scheme – simply because it's impossible that every output maps to a unique input (given that the input space is larger than the output space). This is intuitive – because the input space is larger than the output space, you could encrypt a message given a hash function, but you cannot recover/decrypt the cipher text into the original message (i.e. decode the message, since the output bit-string you get could map to any number of possible messages in the input space). Thus, an encryption scheme must allow anyone who has the key to both encrypt a message *and* decode an encryption to recover the original message – this is an example of the "correctness" property.

Intuition: given one example of an attack – plaintext recovery (given cipher text → learn plaintext) – then intuitively, there should be another type of attack someone could commit: ***key recovery*** (which is simply to learn key); furthermore, intuitively, key recovery seems like a harder task for an attacker than plaintext recovery.

Another example of an attack: ***Cipher text distinguishability attack*** – given cipher c, decide if it corresponds to message m_1 or m_0

The task: given an *adversary A*, who is asked to choose 2 messages, m_1 or m_0 (and $m_0 != m_1$), who then asks for the encryption cipher *c* of one of the messages, the adversary must decide which message (m_1 or m_0) was encrypted.
The adversary asks for the *challenger* to choose a uniform random bit *b* (meaning that the bit could be 0 or 1 with equal probability, akin to just flipping a coin), and the challenger sends back an encrypted cipher text $c^* = Enc_k(m_b)$ (i.e., adversary doesn't know which message m_1 or m_0 the challenger actually chose to encrypt – thus, from the perspective of the adversary, it is simply m_b) – all the adversary has to do is choose the correct message that the challenger encrypted (adversary outputs *b'* (b-prime), which may or may not equal the original bit b). This distinguishability attack, too, is easier to break than the key recovery.

So far, we are building up to a definition that allows the following: an encryption scheme is considered ***"broken"*** if it cannot withstand any of a set of specified attacks. For A, the hardest task to break an encryption scheme is "key recovery", and the easiest task is the "distinguishability attack". From the viewpoint of security experts who want to build a secure system, we want to ensure that our encryption scheme is capable of preventing even the easiest, lamest attack, because if an attacker tries that attack and sees that it's been covered by our system, it implies that harder attacks are also warded against. (Fun fact: SuperPhish was also a "key recovery" attack)

(*Note:* To be clear, so far, we have the scheme itself, and the definition of security. The example so far of the distinguishability attack does not actually detail the scheme itself, but is only used to allow you to better understand the definition of security.)

Another example to detail an actual encryption scheme: the ***one-time pad*** – an encryption scheme for just one bit

- Require key *k* to be a random bit (so k as a bit = 0 w/ 50% chance, 1 with 50% chance)
- Both Alice and Bob know k
- $Enc_k(m)$ (where the message passed into our encryption function, m, is also 1 bit in length) = $(m \oplus k)$ = cipher text *c*

Bit Table Review for ⊕ (XOR):

k	m	(k ⊕ m)
0	0	0
1	0	1
0	1	1
1	1	0

- Alice encrypts m to get encrypted message c, and sends that to Bob.
- Conversely for Bob, in the one-time pad, $Dec_k(c) = (c \oplus k) = m$
- This demonstrates the "correctness" property – since $Dec_k(Enc_k(m)) == m$ (can check that this is indeed the case)

- As for the "security" property of the one-bit, one-time pad ("going back to a definition set by Shannon in the 1940's"): for all adversaries, play this game with Pr[b' = b] = ½ (b-prime in that the adversary guesses a bit b' that does or does not match the original bit the challenger chose to encrypt the message – m_b from the adversary's perspective) – this is to make it such that the adversary cannot do better than just guessing randomly

So, we've identified an **actual** encryption scheme with the one-time pad – it is secure (because of the Pr[…] = ½ proof to be demonstrated), and both correct, given the $Dec_k(Enc_k(m)) == m$ – so it satisfies the properties specified by our definition of an encryption scheme.

(*Note:* What makes the probability actually random in our example is due to the bit b and the key – randomness is over b and key k. But the messages themselves still have structure! So an encryption scheme is still bad if, for messages passed in (since actually choosing which messages to encrypt may not be, itself, random), the scheme then produces an encrypted cipher that still reveals some structure given the predictability of the chosen messages. A good encryption scheme supports a truly random probability if it's still ½ even when the messages passed in share a lot in common. (An example is that, for war, if there were only ever like 3 messages that got sent, such as "Attack at dawn", etc., etc., then your encryption scheme doesn't matter much.))

Proof that one-time pad satisfies the "definition" of security ("cryptographically secure"):

→ Observe that *k* is a random variable:
k = {1 with probability ½, 0 with probability ½}

→ By definition of one-time pad's encryption scheme:
$Pr[(k \oplus m_0) = c^*] = Pr[Enc_k(m_0) = c^*]$
(Here, k is a random variable from the perspective of an adversary; m_0 is NOT a random variable since the adversary chose it; c* is NOT a random variable as well, since it depended upon m_0 which the adversary chose.)

→ The event that the challenger chose to encrypt message m_0 with bit b = 0:
$Pr[Enc_k(m_0) = c^*] = Pr[b = 0]$
(Or otherwise stated, the event that the adversary chose the correct message m_0 that corresponded to the actual bit b = 0 that the challenger chose to encrypt m_0.)

→ By simple algebra:
$Pr[(k \oplus m_0) = c^*] = Pr[(m_0 \oplus m_0 \oplus k) = (c^* \oplus m_0)]$
$Pr[(m_0 \oplus m_0 \oplus k) = (c^* \oplus m_0)] = Pr[k = (m_0 \oplus c^*)]$

→ Therefore, since k is a random variable, the probability that $k = (m_0 \oplus c^*) = ½$
(No matter what the adversary does, no supercomputer, no algorithm, nothing short of breaking into the system and stealing the actual key, the probability that k = 0 or 1, based on the returned value of $(m_0 \oplus c^*)$ given to the adversary by the challenger, will always be ½.)

To see why exactly, follow the same logic with bit b = 1:
$Pr[b = 1]$ = *(same logic as before)* $= Pr[k = (m_1 \oplus c^*)] = ½$
– this shows that:
$Pr[k = (m_0 \oplus c^*)] = ½ = Pr[k = (m_1 \oplus c^*)]$,
meaning the adversary can't guess if either message 1 or 0 was the one encrypted, given c* – and thus, the adversary cannot guess with probability greater than ½, and hence has no information on what to do and must guess randomly.

(*Note:* It's called the "one-time pad" because you can't use the same one-time pad key to encrypt different messages – if you did, you could automatically eliminate the key from the equation:

Given cypher $c_1 = (m_1 \oplus k)$

Given cypher $c_2 = (m_2 \oplus k)$

If you simply:

$(c_1 \oplus c_2)$,

you get

$(m_1 \oplus m_2)$

– and that means that if you have c_1 and c_2 and m_1, you can figure out someone else's original message m_2. Thus, at least in the case of the one-time pad, you must use a unique key for every message you want to encrypt.)