Lecture Notes, Week 2

1 Some Classical Encryption Methods

1.1 Monoalphabetic ciphers

The Caeser cipher uses only the 26 rotations out of the 26! permutations on the alphabet. The *monoalphabetic cipher* uses them all. A key k is an arbitrary permutation of the alphabet. $E_k(m)$ replaces each letter a of m by $k(a)$ to yield c. To decrypt, $D_k(c)$ replaces each letter b of c by $k^{-1}(b)$.

The size of the key space is $|\mathcal{K}| = 26! > 2^{74}$, which is too large for a successful brute force attack. However, monoalphabetic ciphers can be readily broken using letter frequency analysis, given a long enough message. Because each occurrence of a letter a in the message is replaced by the same letter $k(a)$, the most frequently-occurring letter of m will correspond to the most frequently-occurring letter of c. While Eve might not know what the most frequently-occurring letter of m is, if the message is long enough and she knows that it is English, then it is quite likely that the most frequently-occurring letter in m is one of the most frequently-occurring letters in English, i.e., 'e' or maybe 't'. She can then assume that the most frequent letter b_1 in c is 'e', the next most frequent letter b_2 is 't', and so forth. Of course, not all of these guesses will be correct, but the number of likely candidates for each ciphertext letter is greatly reduced. Moreover, many wrong guesses can be quickly discarded even without constructing the entire trial key because they lead to unlikely letter combinations.

1.2 Playfair cipher

A cipher that encrypts two letters at a time is much harder to break than a monoalphabetic cipher since it tends to mask the letter frequencies. The *Playfair* cipher, popularized by L. Playfair in England circa 1854, is one such example.[1] Here, the key is a 5×5 matrix of letters. Pairs of plaintext letters are located in the matrix and used to produce a corresponding pair of ciphertext letters.

For example, consider the key matrix that might result from the passphrase, "CRYPTOGRAPHY REQUIRES STRONG KEYS":

C	R	Y	P	T
O	G	A	H	E
Q	U	I/J	S	N
K	B	D	F	L
M	V	W	X	Z

The matrix is constructed by writing the passphrase into the matrix cells from left to right and top to bottom, omitting any letters that have previously been used. It is filled out with the letters of the alphabet that do not occur in the passphrase, in alphabetical order. (In carrying out this process, "I"

[1] See Memezes et. al., "Handbook of Applied Cryptography", p. 239–240, for details.

and "J" are identified, so we are effectively working over a 25-character alphabet.) Note that each letter of the alphabet occurs exactly once in the resulting matrix.

A message to be encrypted is first broken up into pairs of letters. For example, the message "MEET ME AT THE SUBWAY" would be broken into the pairs "ME" "ET" "ME" "AT" "TH" "ES" "UB" "WA" "YX". Note that we have padded the message with a trailing "X" in order to make its length even, and spaces have been suppressed. Now, each pair of plaintext letters is encrypted. For example, the pair "AT" would be encrypted by taking the rectangle with "A" and "T" as its corners and then using the letters from the other two corners as the cipher text. In this example, the encryption of "AT" is "EY". Decryption is by a similar procedure. The cipher also contains rules for handling various special cases such as when both plaintext letters occur in the same row or the same column or both. In decrypting, one also must resolve the I/J ambiguity and figure out when to remove the padding character.

1.3 Hill cipher

The *Hill cipher* is another example of a cipher that encrypts groups of letters at once, thereby tending to mask letter frequencies. It is based on linear algebra. The key is, say, a non-singular 3×3 matrix K. The message m is divided into vectors m_i of 3 letters each. Encryption is just the matrix-vector product Kv. Decryption is the same using K^{-1}.

Unfortunately, the Hill cipher succumbs to a known plaintext attack. Given three linearly independent vectors m_1, m_2, and m_3 and the corresponding ciphertexts $c_i = Km_i$, $i = 1, 2, 3$, it is straightforward to solve for K.

1.4 Polyalphabetic ciphers

Another way to strengthen monoalphabetic ciphers is to use different substitutions for different letter positions. For example, one might choose 10 different alphabet permutations $k_1, \ldots, k_{10}$, use k_1 for the first letter of m, k_2 for the second letter, and so forth, repeating this sequence after every 10 letters. While this is much harder to break than monoalphabetic ciphers, it turns out that letter frequency analysis can still be used. Every 10th letter is encrypted using the same permutation, so the submessage consisting of just those letters still exhibits normal English language letter frequencies.

1.5 Transposition techniques

All of the methods discussed so far are based on letter substitution. Another technique is to rearrange the letters of the plaintext. For example, one might write a plaintext message into a matrix by rows and then read it out by columns. While transposition does not seem like a very powerful technique by itself, when used in combination with substitution techniques it can be quite effective. Most practical symmetric cryptosystems are built by composing together many stages of substitutions and transpositions.

1.6 Rotor machines

Rotor machines had an important history during the Second World War. The Germans believed their Enigma machine was unbreakable, but the Allies, with great effort, succeeded in breaking it and in reading many of the top-secret military communications. This is said to have changed the course of the war.

The basic idea of a rotor machine is to use electrical switches to create a permutation of 26 input wires to 26 output wires. Each input wire is attached to a key on a keyboard. Each output wire is

attached to a lamp. The keys are associated with letters just like on a computer keyboard. Each lamp is also labeled by a letter from the alphabet. Pressing a key on the keyboard causes one of the lamps to light, which indicates the ciphertext character corresponding to the key pressed. The operator types the message one character at a time and writes down for each letter the corresponding lamp. To decrypt, one could switch inputs and outputs obtain the inverse permutation, type in the ciphertext, and read out the plaintext.

What I have described so far is just an electro-mechanical device for implementing a monoalphabetic cipher. However, rotor machine gain their power by changing the permutation after each letter. Each rotor is individually wired to produce some random-looking fixed permutation π. Several rotors stacked together produce the composition of the permutations implemented by the individual rotors. In addition, the rotors can rotate relative to each other, implementing in effect a rotation permutation (like the Caeser cipher uses). Let $\rho_k(x) = x + k \bmod 26$. Then rotor in position k implements permutation $\rho_k \pi \rho_k^{-1}$. Several rotors could be stacked together to implement the composition of the permutations computed by each. For example, three rotors implementing permutations π_1, π_2, and π_3, placed in positions r_1, r_2, and r_3, respectively, would produce the permutation

$$\begin{aligned} & \rho_{r_1} \cdot \pi_1 \cdot \rho_{-r_1} \cdot \rho_{r_2} \cdot \pi_2 \cdot \rho_{-r_2} \cdot \rho_{r_3} \cdot \pi_3 \cdot \rho_{-r_3} \\ = \quad & \rho_{r_1} \cdot \pi_1 \cdot \rho_{r_2 - r_1} \cdot \pi_2 \cdot \rho_{r_3 - r_2} \cdot \pi_3 \cdot \rho_{-r_3} \end{aligned} \tag{1}$$

After each letter is typed, some of the rotors change position, much like the mechanical odometer used in older cars. The period before the rotor positions repeat is quite long, allowing long messages to be sent without ever repeating the same permutation. Thus, a rotor machine is much like a polyalphabetic substitution cipher, but with a very long period. However, unlike a pure polyalphabetic cipher, the successive permutations until the cycle repeats are not independent of each other but are related to each other by (1). This gives the first toehold into methods for breaking the cipher (which are far beyond the scope of this course).

Several different kinds of rotor machines were built and used, both by the Germans and by others, some of which work somewhat differently from what I described above. However, the basic principles are the same. The interested reader can find much detailed material on the web by searching for "enigma cipher machine" and "rotor cipher machine". One nice description may be found at `http://home.ecn.ab.ca/~jsavard/crypto/jscrypt.htm`.

1.7 Steganography

Steganography, hiding one message inside another, is an old technique that is still in use. For example, a message can be hidden inside a graphics image file by using the low-order bit of each pixel to encode the message. The visual effect of these tiny changes is probably too small to be noticed by the user. The message can be hidden further by compressing it or by encrypting it with a conventional cryptosystem. Unlike conventional cryptosystems, where we assume the attacker knows everything about the cryptosystem except for the secret key, steganography relies on the secrecy of the method of hiding for its security. If Eve does not even recognize the message as ciphertext, then she is not likely to attempt to decrypt it.

2 Symmetric Cryptosystems

Symmetric (one-key) cryptosystems fall into two broad classes, *block ciphers* and *stream ciphers*. A block cipher takes as inputs a key and a fixed-length plaintext block and produces a fixed-length ciphertext block as output. Most of the ciphers we have been discussing so far are of this type. A

stream cipher on the other hand process a stream of characters in an on-line fashion, emitting the ciphertext characters as it goes. The rotor machines are examples of stream ciphers.

2.1 Stream ciphers

A stream cipher has two components, the cipher that is used to encrypt a given character, and a keystream generator that produces a different key to be used for each successive letter.

A simple stream cipher can be built from the XOR cryptosystem used in the one-time pad. However, rather than using a random key as long as the message, we instead generate the keystream on the fly using a state machine. A *keystream generator* consists of three parts: an internal state, a next-state generator, and an output function. The next-state generator and output functions can both depend on (original) key. At each stage, the state is updated and the output function applied to obtain the next component of the keystream. Like a one-time pad, one must use different key for each message; otherwise the system is easy to break.

To be secure, the keystream generator must be a good pseudorandom sequence generator. Any regularities in the output of the keystream generator will give an attacker information about the plaintext. In particular, if the attacker is ever able to figure out the internal state of the keystream generator, then she will be able to predict all future outputs of the generator and decipher the remainder of the ciphertext. It turns out that the linear congruential pseudorandom number generators typically found in software libraries are quite insecure. After observing a relatively short sequence of outputs from the generator, one can solve for the state and correctly predict all future outputs. For the simple XOR cipher to be secure, a *cryptographically strong* pseudorandom number generator must be used. Even so, the fact that a different key must be used for each message sent makes it problematic in practice. It's at least an improvement to make the next-state generator depend on the current plaintext or ciphertext characters so that the generated keystreams will diverge on different messages, even if the key is the same.

2.2 Block ciphers

A block cipher operates on an entire *block* of plaintext to produce a *block* of ciphertext. So does a stream cipher, but with a stream cipher the block size is very small (typically one bit or one byte), whereas a block cipher operates on fairly long blocks, e.g., 64-bits for DES, 128-bits for Rijndal (AES). With a stream cipher, security rests with the keystream generator, for the ciphers used to encrypt the individual characters are all subject to known-plaintext attacks. On the other hand, block ciphers can be designed to resist known-plaintext attacks and can therefore be pretty secure, even if the same key is used to encrypt a succession of blocks, as is often the case.

Of course, the length messages one wants to send are rarely exactly the block length. To use a block cipher to encrypt long messages, one first divides the message into blocks of the right length, padding the last partial block according to a suitable padding rule. Then the block cipher is used in some *chaining mode* to encrypt the sequence of resulting blocks. A chaining mode tells how to encrypt a sequence of plaintext blocks $m_1, m_2, \ldots, m_t$ to produce a corresponding sequence of ciphertext blocks $c_1, c_2, \ldots, c_t$, and conversely, how to recover the m_i's given the c_i's.

Some standard chaining modes are:

- *Electronic Codebook Mode (ECB)* – apply cipher to each plaintext block. That is, $c_i = E_k(m_i)$ for each i. This becomes in effect a monoalphabetic cipher, where the "alphabet" is the set of all possible blocks and the permutation is defined by E_k. To decrypt, Bob computes $m_i = D_k(c_i)$.

- *Cipher Block Chaining Mode (CBC)* – encrypt the XOR of the current plaintext block with the previous ciphertext block to produce the current ciphertext block. That is, $c_i = E_k(m_i \oplus c_{i-1})$. To get started, we take $c_0 = \text{IV}$, where IV is a fixed *initialization vector* which we assume is publicly known. To decrypt, Bob computes $m_i = D_k(c_i) \oplus c_{i-1}$.

- *Cipher-Feedback Mode (CFB)* – XOR the current plaintext block with the encryption of the previous ciphertext block. That is, $c_i = m_i \oplus E_k(c_{i-1})$, where again, c_0 is a fixed initialization vector. To decrypt, Bob computes $m_i = c_i \oplus E_k(c_{i-1})$. Note that Bob is able to decrypt without using the block decryption function D_k. In fact, it is not even necessary for E_k to be a one-to-one function (but using a non one-to-one function might weaken security).

- *Output Feedback Mode (OFB)* – the encryption function is iterated on an *initial vector (IV)* to produce a stream of block keys, which in turn are XORed with the successive plaintext blocks to produce the successive ciphertext blocks. (This is similar to a simple keystream generator.) That is, $c_i = m_i \oplus k_i$, where $k_i = E_k(k_{i-1})$ is a *block key*. k_0 is a fixed initialization vector IV. To decrypt, Bob can apply exactly the same method to the ciphertext to get the plaintext, that is, $m_i = c_i \oplus k_i$, where $k_i = E_k(k_{i-1})$.

- *Propagating Cipher-Block Chaining Mode (PCBC)* – encrypt the XOR of the current plaintext block, previous plaintext block, and previous ciphertext block. That is, $c_i = E_k(m_i \oplus m_{i-1} \oplus c_{i-1})$. Here, both m_0 and c_0 are fixed initialization vectors. To decrypt, Bob computes $m_i = D(c_i) \oplus m_{i-1} \oplus c_{i-1}$.

Remarks

1. Both CFB and OFB are closely related to stream ciphers since in both cases, c_i is m_i XORed with some function of stuff that came before stage i. Like a one-time pad and other simple XOR stream ciphers, OFB becomes insecure if the same key is ever reused, for the sequence of k_i's generated will be the same. CFB, however, avoids this problem, for even if the same key k is used for two different message sequences m_i and m'_i, it will not generally be the case that $m_i \oplus m'_i = c_i \oplus c'_i$; rather, $m_i \oplus m'_i = c_i \oplus c'_i \oplus E_k(c_{i-1}) \oplus E_k(c'_{i-1})$, and the dependency on k does not drop out.

2. The different modes differ in their sensitivity to data corruption. With ECB and OFB, if Bob receives a bad block c_i, then he cannot recover the corresponding m_i, but all good ciphertext blocks can be decrypted. With CBC and CFB, he needs both good c_i and c_{i-1} blocks in order to decrypt m_i. Therefore, a bad block c_i renders both m_i and m_{i+1} unreadable. With PCBC, a bad block c_i renders m_j unreadable for all $j \geq i$.

3. Other modes can be easily invented. We see that in all cases, c_i is computed by some expression (which may depend on i) built from $E_k()$ and $\oplus$ applied to blocks $c_1, \ldots, c_{i-1}$, $m_1, \ldots, m_i$, and the initialization vectors. Any such equation that can be "solved" for m_i (by possibly using $D_k()$ to invert $E_k()$) is a suitable chaining mode in the sense that Alice is able to produce the ciphertext and Bob is able to decrypt it. Of course, the resulting security properties depend heavily on the particular expression chosen.