CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 3
September 9, 2015

Polyalphabetic Substitution Ciphers

Polyalphabetic ciphers

Recall: A *polyalphabetic substitution cipher* allows a different substitution to be applied to a plaintext letter, depending on the letter's position i in the message.

The Vigenère cipher presented last time is a simple example.

The key is the tuple $(r, k_0, \ldots, k_{r-1})$.

The i plaintext letter is encrypted using the Caesar cipher with key k_s, where $s = i \bmod r$.

Vigenère example

Suppose $k = (3, 5, 2, 3)$ and $m =$ "et tu brute".

Plaintext	`ettub rute`
Sub-key	`52352 3523`
Ciphertext	`jvwzd uzvh`

Rotor machines

Rotor machines are mechanical polyalphabetic cipher devices that generalize Vigenère ciphers, both in having a very large value of r and in their method of generating the substitutions from the letter positions.

They were invented about 100 years ago and were used into the 1980's.

See Wikipedia page on rotor machines for a summary of the many such machines that have been used during the past century.

The German Enigma machines

- Enigma machines are rotor machines invented by German engineer Arthur Scherbius.
- They played an important role during World War 2.
- The Germans believed their Enigma machines were unbreakable.
- The Allies, with great effort, succeeded in breaking them and in reading many top-secret military communications.
- This is said to have changed the course of the war.

Image from Wikipedia

How a rotor machine works

- Uses 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 a lamp to light, indicating the corresponding ciphertext character.

The operator types the message one character at a time and writes down the letter corresponding to the illuminated lamp.

The same process works for decryption since $E_{k_i} = D_{k_i}$.

Keystream generation

The encryption permutation.

- 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).

Keystream generation (cont.)

Let $\rho_k(x) = (x + k) \bmod 26$. Then rotor in position k implements permutation $\rho_k \pi \rho_k^{-1}$. (Note that $\rho_k^{-1} = \rho_{-k}$.)

Several rotors stacked together 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

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

Changing the permutation

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 repeating the same permutation.

Thus, a rotor machine is implements a polyalphabetic substitution cipher with a very long period.

Unlike a pure polyalphabetic cipher, the successive permutations until the cycle repeats are not independent of each other but are related by equation (1).

This gives the first toehold into methods for breaking the cipher (which are far beyond the scope of this course).

History

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". Nice descriptions may be found at http://en.wikipedia.org/wiki/Enigma_machine and http://www.quadibloc.com/crypto/intro.htm.

Vernam cipher

The *Vernam cipher* (one-time pad) is an *information-theoretically secure* cryptosystem.

This means that Eve, knowing only the ciphertext, can extract absolutely no information about the plaintex other than its length.

We will explore the concept of information-theoretic security later.

Exclusive-or on bits

The Vernam cipher is based on *exclusive-or* (XOR), which we write as $\oplus$.

- $x \oplus y$ is true when exactly one of x and y is true.
- $x \oplus y$ is false when x and y are both true or both false.

Exclusive-or is just sum modulo two if 1 represents true and 0 represents false.

$$x \oplus y = (x + y) \bmod 2.$$

XOR is associative and commutative. 0 is the identity element.

$$k \oplus 0 = 0 \oplus k = k$$

XOR is its own inverse.

$$k \oplus k = 0$$

Informal description

The one-time pad encrypts a message m by XORing it with the key k, which must be as long as m.

Assume both m and k are represented by strings of bits. Then ciphertext bit $c_i = m_i \oplus k_i$.

Note that $c_i = m_i$ if $k_i = 0$, and $c_i = \neg m_i$ if $k_i = 1$.

Decryption is the same, i.e., $m_i = c_i \oplus k_i$.

The one-time pad cryptosystem formally defined

$\mathcal{M} = \mathcal{C} = \mathcal{K} = \{0,1\}^r$ for some length r.

$E_k(m) = D_k(m) = k \oplus m$, where $\oplus$ is applied to corresponding bits of k and m.

It works because

$$D_k(E_k(m)) = k \oplus (k \oplus m) = (k \oplus k) \oplus m = 0 \oplus m = m.$$

Security

Like the 1-letter Caesar cipher, for given m and c, there is *exactly one* key k such that $E_k(m) = c$ (namely, $k = m \oplus c$).

For fixed c, m varies over all possible messages as k ranges over all possible keys, so c gives no information about m.

It will follow that the one-time pad is information-theoretically secure.

What more is there to prove?

Importance of the Vernam cipher

It is important because

- it is sometimes used in practice;
- it is the basis for many *stream ciphers*, where the truly random key is replaced by a pseudo-random bit string.

Attraction of one-time pad

The one-time pad would seem to be the perfect cryptosystem.

- It works for messages of any length (by choosing a key of the same length).
- It is easy to encrypt and decrypt.
- It is information-theoretically secure.

In fact, it is sometimes used for highly sensitive data.

Drawbacks of one-time pad

It has two major drawbacks:

1. The key k must be as long as the message to be encrypted.
2. The same key must never be used more than once. (Hence the term "one-time".)

Together, these make the problem of key distribution and key management very difficult.

Why the key cannot be reused

If Eve knows just one plaintext-ciphertext pair (m_1, c_1), then she can recover the key $k = m_1 \oplus c_1$.
This allows her to decrypt all future messages sent with that key.

Even in a ciphertext-only situation, if Eve has two ciphertexts c_1 and c_2 encrypted by the same key k, she can gain significant partial information about the corresponding messages m_1 and m_2.

In particular, she can compute $m_1 \oplus m_2$ without knowing either m_1 or m_2 since

$$m_1 \oplus m_2 = (c_1 \oplus k) \oplus (c_2 \oplus k) = c_1 \oplus c_2.$$

How knowing $m_1 \oplus m_2$ might help an attacker

Fact (important property of $\oplus$)

For bits b_1 and b_2, $b_1 \oplus b_2 = 0$ if and only if $b_1 = b_2$.

Hence, blocks of 0's in $m_1 \oplus m_2$ indicate regions where the two messages m_1 and m_2 are identical.

That information, together with other information Eve might have about the likely content of the messages, may be enough for her to seriously compromise the secrecy of the data.

Cryptanalysis

Breaking the Caesar: A brute force attack

We saw last time an example of breaking the Caesar cipher using a brute force attack.

Brute force attack means trying every possible key to see which one "works".

Determining which is the correct key is the problem.

For our Caesar cipher example, there were only 26 possible keys; hence only 26 possible decryptions of the given ciphertext `HWWXE UXWH`, only one of which "makes sense".

Breaking the Caesar cipher: Extending these ideas

The longer the correct message, the more likely that only one key results in a sensible decryption.

For example, suppose the ciphertext were "EXB JXQ".
We saw two possible keys for "JXQ" — 3 and 23.
Trying them both we get:

$$k = 3: \quad D_3(\text{EXB JXQ}) = \text{BUY GUN}.$$

$$k = 23: \quad D_{23}(\text{EXB JXQ}) = \text{HAE MAT}.$$

Latter is nonsense, so we know $k = 3$ and the message is "BUY GUN".

Breaking the Caesar cipher: Conclusion

Let n be the message length.

- $n = 1$: The Caesar cipher is information-theoretically secure!
- $n > 1$: The Caesar cipher is only partially secure or completely breakable, depending on message length and redundancy present in the message.

How long is long enough for a brute force attack to succeed?

There is a whole theory of redundancy of natural language that allows one to calculate a number called the "unicity distance" for a given cryptosystem. If a message is longer than the unicity distance, there is a high probability that it is the only meaningful message with a given ciphertext and hence can be recovered uniquely, as we were able to recover "BUY GUN" from the ciphertext "EXB JXW" in the example. See [Sti06, section 2.6] for more information on this interesting topic.

Trying all keys

A *brute force attack* can be attempted against any cryptosystem.

It tries all possible keys k. It works against the Caesar cipher because the key space is so small.

For each k, Eve computes $m_k = D_k(c)$ and tests if m_k is meaningful. If exactly one meaningful m_k is found, she knows that $m = m_k$.

Given long enough messages, the Caesar cipher is easily broken by brute force—one simply tries all 26 possible keys to see which leads to a sensible plaintext.

What is long enough?

Automating brute force attacks

With modern computers, it is quite feasible for an attacker to try millions ($\sim 2^{20}$) or billions ($\sim 2^{30}$) of keys.

The attacker also needs an automated test to determine when she has a likely candidate for the real key.

How does one write a program to distinguish valid English sentences from gibberish?

One could imagine applying all sorts of complicated natural language processing techniques to this task. However, much simpler techniques can be nearly as effective.

Random English-like messages

Consider random messages whose letter frequencies are similar to that of valid English sentences.

For each letter b, let p_b be the probability (relative frequency) of that letter in normal English text.

A message $m = m_1 m_2 \ldots m_r$ has probability $p_{m_1} \cdot p_{m_2} \cdots p_{m_r}$.

This is the probability of m being generated by the simple process that chooses r letters one at a time according to the probability distribution p.

Determining likely keys

Assume Eve knows that $c = E_k(m)$, where m was chosen randomly as described above and k is uniformly distributed.

Eve easily computes the 26 possible plaintext messages $D_0(c), ..., D_{25}(c)$, one of which is correct.

To choose which, she computes the conditional probability of each message given c, then picks the message with the greatest probability.

This guess will not always be correct, but for letter distributions that are not too close to uniform (including English text) and sufficiently long messages, it works correctly with very high probability.

How long should the keys be?

The DES (Data Encryption Standard) cryptosystem (which we will talk about next week) has 56-bit keys for a key space of size 2^{56}.

A special DES Key Search Machine was built as a collaborative project by Cryptography Research, Advanced Wireless Technologies, and EFF. (Click here for details.)

This machine was capable of searching 90 billion keys/second and discovered the RSA DES Challenge key on July 15, 1998, after searching for 56 hours. The entire project cost was under $250,000.

Now, 15+ years later, the same task could likely be done on a commercial cluster computer such as Amazon's Elastic Compute Cloud (EC2) at modest cost.

What is safe today and into the future?

DES with its 56-bit keys offers little security today.

80-bit keys were considered acceptable in the past decade, but in 2005, NIST proposed that they be used only until 2010.

Triple DES (with 112-bit keys) and AES (with 128-bit keys) will probably always be safe from brute-force attacks (but not necessarily from other kinds of attacks).

Quantum computers, if they become a reality, would cut the effective key length in half (see Wikipedia "key size"), so some people recommend 256-bit keys (which AES supports).

Cryptography before computers

Large-scale brute force attacks were not feasible before computers.

While Caesar is easily broken by hand, clever systems have been devised that can be used by hand but are surprisingly secure.

Attacks on any monoalphabetic ciphers

The Caesar 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 $|\mathcal{K}| = 26! > 2^{74}$, large enough to be moderately resistant to a brute force attack.

Nevertheless, monoalphabetic ciphers can be readily broken using letter frequency analysis, given a long enough message.

This is because *monoalphabetic ciphers preserve letter frequencies*.

How to break monoalphabetic ciphers

Each occurrence of a in m is replaced by $k(a)$ to get c.
Hence, if a is the most frequent letter in m, $k(a)$ will be the most frequent letter in c.

Eve now guesses that a is one of the most frequently-occurring letters in English, i.e., 'e' or 't'.
She then repeats on successively less frequent ciphertext letters.

Of course, not all of these guesses will be correct, but in this way the search space is vastly reduced.

Moreover, many wrong guesses can be quickly discarded even without constructing the entire trial key because they lead to unlikely letter combinations.

Why can't one break the one-time pad?

For the one-time pad on n-bit messages and keys, there are 2^n possible keys.

For any fixed ciphertext c, every n-bit message is a possible decryption of c.

This completely masks all letter frequency information from the ciphertext.

References


Douglas R. Stinson.
Cryptography: Theory and Practice.
Chapman & Hall/CRC, third edition, 2006.
ISBN-10: 1-58488-508-4; ISBN-13: 978-58488-508-5.