

Exercises for the lecture

Cryptography

Solutions to Exercise Sheet 1

for the exercise on October 21th, 2026

Exercise 1.1

Assume that an attacker knows that a user's message is either "cablecar" or "superman". The user encrypts the message using a shift cipher. Show how the attacker can determine the user's message with the help of the resulting ciphertext, or explain why this is not possible.

Solution: Since the key space has only size 26, we can trivially brute-force the cipher by hand (as seen in the lecture). Let us mention some structural attacks that are also possible.

- Two positions in the message carry equal letters if and only if the same positions in the ciphertext carry equal letters. Thus equal letters are preserved in a shift cipher, which is an invariant. Thus, if the 2nd and 7th letter of the ciphertext coincide, then the 2nd and 7th letter of the message also need to coincide, so the message is "cablecar" and cannot be "superman".
- More generally, letter differences (distances) are preserved in a shift cipher. If the shift required to turn the first letter into the second letter of the ciphertext equals 24 (i.e., 2nd letter minus 1st letter as numbers), then the message was "cablecar" because a shift of 24 turns "c" into "a", whereas the corresponding shift from "s" to "u" for "superman" is 2.
- Shift ciphers also leave the letter distribution invariant (ignoring the labels), but we have little ciphertext material available in this scenario.

Exercise 1.2

Consider the following cipher. Given an English word of length n as key, write the message row-by-row below the letters of the key (see illustration). Thus there are as many columns as there are letters in the key and as many rows as necessary to accommodate the message. If the message length is not a multiple of the key length, then pad the message by spaces. Next, we re-arrange the columns such that the letters of the key occur in alphabetic order (keeping ties in the order that they occur in the key). The ciphertext is obtained by reading column 1 top-to-bottom followed by the letters of column 2 top-to-bottom and so on until all columns have been read.

	K	E	Y		E	K	Y	
	P	L	A		L	P	A	
	I	N	T		N	I	T	
	E	X	T		X	E	T	

Message: PLAINTEXT → Ciphertext: LNXPIEATT

- (a) Specify the key space K for this cipher and formalize this cipher as a simplistic encryption scheme $(M, C, \text{enc}, \text{dec})$ for K .
- (b) Verify correctness of this encryption scheme.
- (c) Is this encryption scheme secure? If not, how would you approach breaking this system?

Solution:

- (a) We can model the key space K as the set of all permutations on n elements; i.e.,

$$K = \{k \mid k: [n] \rightarrow [n]\} \quad \text{with} \quad [n] = \{1, \dots, n\} .$$

For the provided example, the permutation $\pi': [3] \rightarrow [3]$ would be

$$\pi'(1) = 2 \quad \pi'(2) = 1 \quad \pi'(3) = 3 .$$

(The first column turns into the second column, the second column turns into the first column, and the last column remains.) Assuming an English message with alphabet $A = \{'a', \dots, 'z', '\#\}'$ we obtain the message space $M = \{m \in A^* \mid 0 = |m| \bmod n\} = C$, which is also the ciphertext space. Encryption of a message $m \in M$ of length ℓn (padded to length multiple of n) is given by

$$\begin{aligned} \text{enc}_\pi(m_1 \cdots m_{\ell n}) &= m_{0n+\pi^{-1}(1)} m_{1n+\pi^{-1}(1)} \cdots m_{(\ell-1)n+\pi^{-1}(1)} & (\pi^{-1}(1)\text{-column}) \\ &\dots \\ &m_{0n+\pi^{-1}(n)} m_{1n+\pi^{-1}(n)} \cdots m_{(\ell-1)n+\pi^{-1}(n)} . & (\pi^{-1}(n)\text{-column}) \end{aligned}$$

Once again, let us verify this formula on the provided example message $m = \text{PLAINTEXT}$.

$$\begin{aligned} \text{enc}_{\pi'}(P_1 L_2 A_3 I_4 N_5 T_6 E_7 X_8 T_9) &= m_{0n+2} m_{1n+2} m_{2n+2} \\ &\quad m_{0n+1} m_{1n+1} m_{2n+1} \\ &\quad m_{0n+3} m_{1n+3} m_{2n+3} \\ &= m_2 m_5 m_8 m_1 m_4 m_7 m_3 m_6 m_9 = L_2 N_5 X_8 P_1 I_4 E_7 A_3 T_6 T_9 . \end{aligned}$$

In other words, encryption is given by the permutation $\Pi: [\ell n] \rightarrow [\ell n]$ given by

$$\Pi(in + j) = (\pi(j) - 1)n + i + 1$$

for every $0 \leq i < \ell$ and $1 \leq j \leq n$. This allows us to formulate decryption simply as $\text{dec}_\pi: [\ell n] \rightarrow [\ell n]$, which is given by

$$\text{dec}_\pi(c_1 \cdots c_{\ell n}) = c_{\Pi^{-1}(1)} \cdots c_{\Pi^{-1}(\ell n)} .$$

- (b) Correctness is now easy. We need to verify

$$\text{dec}_\pi(\text{enc}_\pi(m)) = m .$$

Utilizing the identified permutations, we obtain

$$\begin{aligned} \text{dec}_\pi(\text{enc}_\pi(m)) &= \text{dec}_\pi(m_{\Pi(1)} \cdots m_{\Pi(\ell n)}) \\ &= m_{\Pi(\Pi^{-1}(1))} \cdots m_{\Pi(\Pi^{-1}(\ell n))} = m_1 \cdots m_{\ell n} = m \end{aligned}$$

- (c) The size $n!$ of the key space is overall reasonable if the length n of key phrase is large, so brute force becomes unreasonable pretty quickly. Obviously a single common English word as key severely limits its length (longest word in dictionaries is usually “pneumonoultramicroscopicsilicovolcanoconiosis” of length 45 yielding

$$119,622,220,865,480,194,561,963,161,495,657,715,064,383,733,760 \cdot 10^9$$

permutations, so roughly $120 \cdot 10^{54}$ permutations). A modern desktop processor (AMD Ryzen 9950x3D) can achieve 240,000 MIPS and assuming that one permutation can be processed per instruction (absurd, but simplifies discussion and only distorts result by trivial constant) we obtain a run-time of roughly $5 \cdot 10^{44} \text{ s} \approx 4 \cdot 10^{38} \text{ a}$.

However, our formalization is overly optimistic once the key length n exceeds the alphabet size 27 (26 letters and #) and the many repetitions of letters in actual words yield the same problem (English has a [common] vocabulary size of roughly 300,000 words, but already $10! = 3,628,800$, so already at length 10, there are an order of magnitude more permutations than words that could encode them). Additionally, the average word length in English is only 6 letters, so brute forcing such a key can be done in $3 \cdot 10^{-9} \text{ s} = 3 \text{ ns}$. Even assuming that we need to try all 300,000 possible permutations (limit from the common English vocabulary), we only require $2 \cdot 10^{-6} \text{ s} = 2 \mu\text{s}$. Overall, brute-forcing this encryption scheme seems very reasonable, although our formalization might indicate that it should not be possible.

Assuming that we know the key length n , we can also utilize knowledge of digrams (2-letter combinations) and trigrams (3-letter combinations) to direct the search for the correct permutation.

Exercise 1.3

Develop a simplistic encryption scheme $(M, C, \text{enc}, \text{dec})$ and the key space K for it such that $\text{enc}_k(\text{dec}_k(c)) \neq c$ for some $c \in C$ and $k \in K$. Mind that the scheme need not be secure.

Solution: Our example of Exercise 1.2 obviously relies on bijective functions. However, we can suitably adapt that scheme, so let K and $(M, C, \text{enc}, \text{dec})$ be the encryption scheme of Exercise 1.2. We adjust the ciphertext space to $C' = \{c\# \mid c \in C\}$. In other words, we attach a final separator # to all ciphertexts. We similarly adjust encryption to $\text{enc}'_k: M \rightarrow C'$ given by

$$\text{enc}'_k(m) = \text{enc}_k(m) \#$$

for every $m \in M$, so we also just append a separator # to the standard encryption. The decryption proceeds normally on the prefix without the final separator, if the ciphertext ends with # and otherwise returns just the empty string ε . So $\text{dec}'_k: C' \rightarrow M$ is given for every $c \in C'$ by

$$\text{dec}'_k(c) = \begin{cases} \text{dec}_k(c') & \text{if } c = c'\# \\ \varepsilon & \text{otherwise.} \end{cases}$$

It is straightforward to verify that this encryption scheme is still correct, since actually produced ciphertexts always end with #. However, the equality $\text{enc}'_k(\text{dec}'_k(c)) = c$ trivially does not hold since all ciphertexts that do not end with # are mapped to the empty word by the decryption, which is in turn mapped to # by the encryption.