

Міністерство освіти і науки України
Національний авіаційний університет
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

Лабораторна робота № 2.1
з дисципліни «Захист інформації в комп'ютерних системах»
на тему «Ознайомлення з криптографічними алгоритмами»

Виконав:
студент ФККПІ
групи СП-425
Клокун В. Д.
Перевірила:
Супрун О. М.

Київ 2019

1. МЕТА РОБОТИ

Ознайомитись з основними поняттями криптографії. Ознайомитися з поняттями дайджеста повідомлення та деякими засобами його отримання та перевірки.

2. ЗАВДАННЯ РОБОТИ

Створити програмні реалізації простих криптографічних алгоритмів. Ознайомитись з утилітами для отримання та перевірки дайджестів повідомлень. Ознайомитися з алгоритмами отримання дайджестів та на їх основі створити аналогічну утиліту самостійно.

3. ХІД РОБОТИ

3.1. Реалізація алгоритмів шифрування

Щоб виконати завдання, необхідно розробити програмний модуль, який реалізує алгоритми шифрування: заміною, перестановкою, гамуванням та аналітичним перетворенням. Також необхідно реалізувати розшифрування повідомлень, зашифрованих реалізаціями кожної з вищезазначених категорій. Розроблюємо програмний модуль, який реалізує алгоритми шифрування із завдання (лістинг А.1). Запускаємо розроблений програмний модуль і бачимо результати шифрування різними алгоритмами (рис. 1).

Бачимо, що розроблені реалізації дійсно шифрують задане повідомлення і вірно розшифровують його, тому необхідні алгоритми шифрування різних категорій були правильно реалізовані.

3.2. Знайомство із засобами командного рядка Windows для хешування

Завдання лабораторної роботи вимагає ознайомитись із програмами для операційної системи Windows, призначеними для обчислення хеш-сум за допомогою командного рядка, а саме HashConsole, HashUtils і rhash.

Щоб ознайомитись із програмами, спробуємо обчислити хеш-суму певного файлу. Для цього запускаємо програми із необхідними параметрами і бачимо результат (рис. 2).

Усі програми дали очікуваний результат, тому ми переконались, що вони вірно встановлені і справно працюють.

```
C:\Windows\system32\cmd.exe
>python y04601-infosec-lab-02-01-solution.py
Process: Encryption
Input: the quick brown fox jumps over the lazy dog.
Key: ['a': 'm', 'b': 'x', 'c': 'y', 'd': 'z', 'e': 'a', 'f': 'b', 'g': 'c',
'h': 'd', 'i': 'e', 'j': 'f', 'k': 'g', 'l': 'h', 'm': 'i', 'n': 'j', 'o': 'k',
'p': 'l', 'q': 'm', 'r': 'n', 's': 'o', 't': 'p', 'u': 'q', 'v': 'r', 'w': 's',
'x': 't', 'y': 'u', 'z': 'v', ' ': ' ', '.': '.']
Output: pda0mqeyg0xmkjs0bkt0fqllo0kran0pda0hvvu0zkc.

Process: Decryption
Input: pda0mqeyg0xmkjs0bkt0fqllo0kran0pda0hvvu0zkc.
Key: ['a': 'm', 'b': 'x', 'c': 'y', 'd': 'z', 'e': 'a', 'f': 'b', 'g': 'c',
'h': 'd', 'i': 'e', 'j': 'f', 'k': 'g', 'l': 'h', 'm': 'i', 'n': 'j', 'o': 'k',
'p': 'l', 'q': 'm', 'r': 'n', 's': 'o', 't': 'p', 'u': 'q', 'v': 'r', 'w': 's',
'x': 't', 'y': 'u', 'z': 'v', ' ': ' ', '.': '.']
Output: the quick brown fox jumps over the lazy dog.

Process: Encryption
Input: the quick brown fox jumps over the lazy dog.
Key: S
Output: tub j ldhirfuotaoccomvhzg kwxpeey.q n sr

Process: Decryption
Input: tub j ldhirfuotaoccomvhzg kwxpeey.q n sr
Key: S
Output: the quick brown fox jumps over the lazy dog.

Process: Encryption
Input: the quick brown fox jumps over the lazy dog.
Key: [32, 114, 133, 132, 141, 85, 251, 181, 209, 185, 163, 17, 178, 161, 254,
246, 91, 188, 95, 18, 217, 17, 111, 225, 184, 69, 5, 254, 66, 48, 126, 47, 94,
64, 154, 86, 114, 72, 227, 119, 49, 184, 241, 30]
Output: [84, 26, 224, 184, 252, 32, 146, 6, 186, 153, 193, 99, 221, 214, 144, 2
14, 61, 211, 39, 42, 179, 180, 2, 145, 203, 28, 186, 136, 39, 66, 94, 91, 54, 37
, 186, 58, 19, 59, 154, 87, 65, 7, 150, 48]



Process: Decryption
Input: [84, 26, 224, 184, 252, 32, 146, 6, 186, 153, 193, 99, 221, 214, 144, 2
14, 61, 211, 39, 42, 179, 180, 2, 145, 203, 28, 186, 136, 39, 66, 94, 91, 54, 37
, 186, 58, 19, 59, 154, 87, 65, 7, 150, 48]
Key: [32, 114, 133, 132, 141, 85, 251, 181, 209, 185, 163, 17, 178, 161, 254,
246, 91, 188, 95, 18, 217, 17, 111, 225, 184, 69, 5, 254, 66, 48, 126, 47, 94,
64, 154, 86, 114, 72, 227, 119, 49, 184, 241, 30]
Output: the quick brown fox jumps over the lazy dog.

Process: Encryption
Input: the quick brown fox jumps over the lazy dog.
Key: S
Output: ym3vznhp0gt[skkt]k0zruxxtfjwkyms0qf0aitl3

Process: Decryption
Input: ym3vznhp0gt[skkt]k0zruxxtfjwkyms0qf0aitl3
Key: S
Output: the quick brown fox jumps over the lazy dog.
```

Рис. 1: Результат шифрування повідомлення розробленим програмним модулем

3.3. Знайомство з графічними засобами перевірки хеш-сум

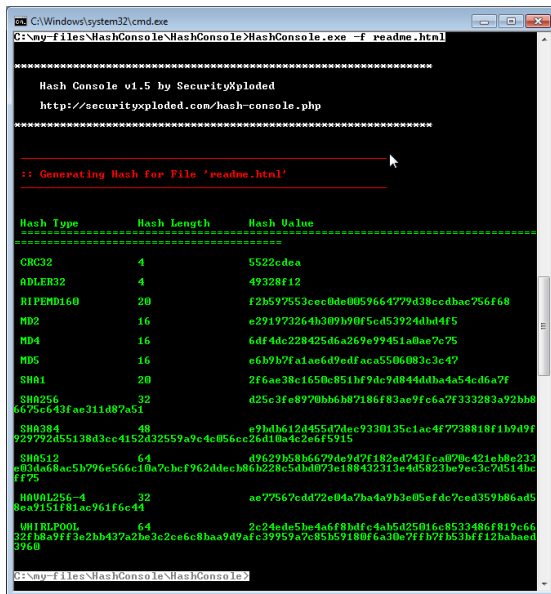
Для обчислення та перевірки хеш-суми за допомогою графічного інтерфейсу, використаємо відповідну можливість програми-архіватора 7-Zip. Відкриваємо Провідник, виділяємо файл, хеш-суму якого необхідно обчислити, відриваємо контекстне меню, натиснувши праву клавішу миші, та обираємо пункт **CRC SHA**  , щоб обчислити хеш-суми файлу за допомогою усіх доступних алгоритмів. В результаті бачимо обчислені хеш-суми (рис. 3).

Як бачимо, незважаючи на те, що обчислення хеш-сум не є основним призначенням архіватора 7-Zip, він все одно надає цю можливість.

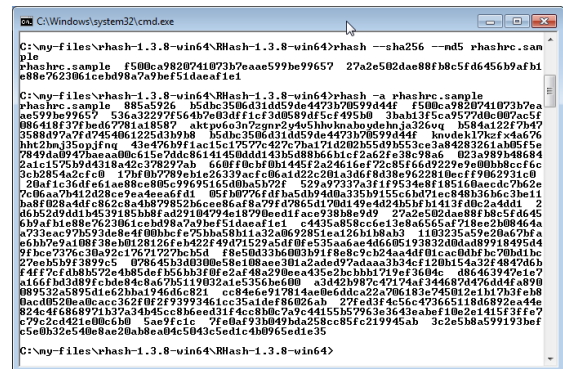
3.4. Знайомство із засобами обчислення хеш-сум командного рядка GNU/Linux

В операційній системі GNU/Linux за замовчуванням встановлені інструменти для обчислення хеш-сум, зокрема `md5sum`, `sha1sum` та `sha256sum`. Використаємо їх для обчислення хеш-сум певного файлу. Для цього запускаємо їх, вказуємо ім'я бажаного файлу як параметр і бачимо результат (рис. 4).

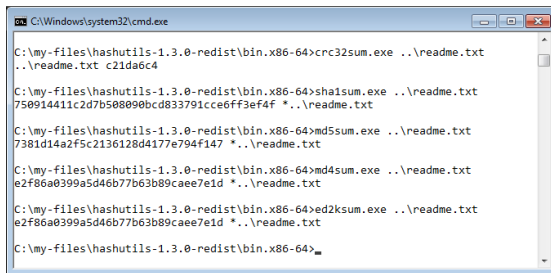
Як бачимо, обрані засоби повернули правдоподібні значення, тому можна сказати, що вони справно працюють.



а)



б)



в)

Рис. 2: Результат обчислення хеш-сум файлів за допомогою запропонованих програм: а — HashConsole, б — RHash, в — hashutils

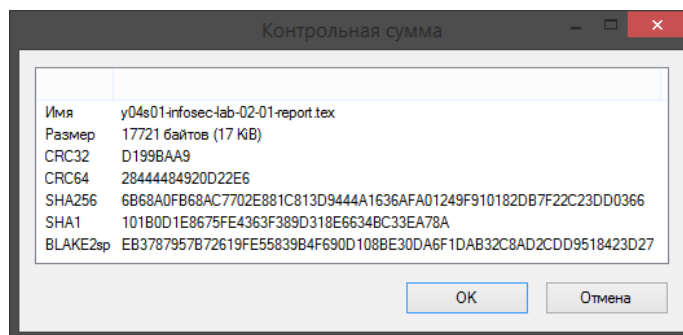
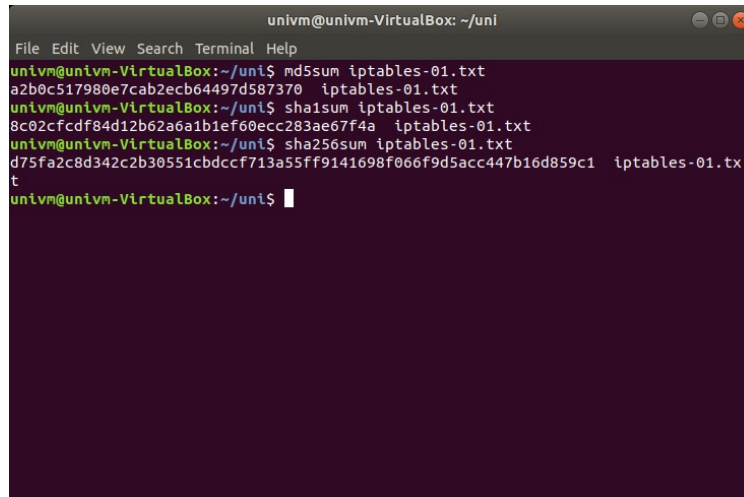


Рис. 3: Результат обчислення хеш-сум за допомогою програми 7-Zip



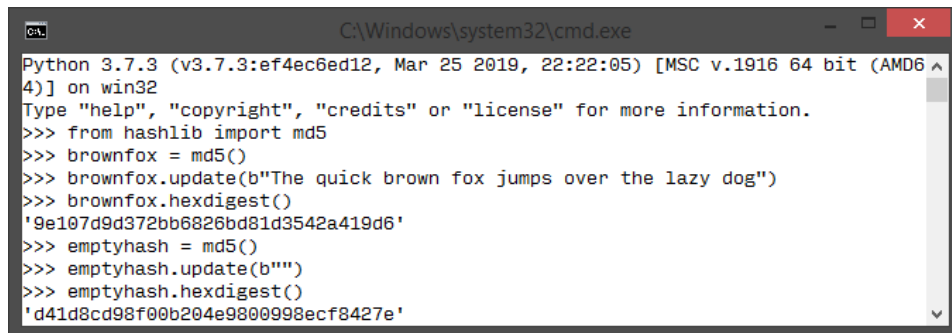
```
univm@univm-VirtualBox: ~/uni
File Edit View Search Terminal Help
univm@univm-VirtualBox:~/uni$ md5sum iptables-01.txt
a2b0c517980e7cab2ecb64497d587370  iptables-01.txt
univm@univm-VirtualBox:~/uni$ sha1sum iptables-01.txt
8c02cfd84d12b62a6a1b1ef60ecc283ae67f4a  iptables-01.txt
univm@univm-VirtualBox:~/uni$ sha256sum iptables-01.txt
d75fa2c8d342c2b30551cbdccf713a55ff9141698f066f9d5acc447b16d859c1  iptables-01.tx
t
univm@univm-VirtualBox:~/uni$
```

Рис. 4: Обчислення хеш-суми файлу за допомогою засобів командного рядка операційної системи Ubuntu

3.5. Знайомство з алгоритмом MD5

Знайомимось із алгоритмом MD5. Це алгоритм обчислення хеш-суми, побудований на основі структури Меркла—Дамгора, складається з 4 раундів обчислення і повертає 128-бітне значення. У 2013 році дослідники представили атаку, яка зламує стійкість алгоритму MD5 до колізій за 2^{18} часу.

Щоб перевірити коректність реалізації алгоритму MD5, обчислимо хеш-суми еталонних повідомлень: порожнього символного рядка «» та фрази «The quick brown fox jumps over the lazy dog» (рис. 5).



```
C:\Windows\system32\cmd.exe
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from hashlib import md5
>>> brownfox = md5()
>>> brownfox.update(b"The quick brown fox jumps over the lazy dog")
>>> brownfox.hexdigest()
'9e107d9d372bb6826bd81d3542a419d6'
>>> emptyhash = md5()
>>> emptyhash.update(b'')
>>> emptyhash.hexdigest()
'd41d8cd98f00b204e9800998ecf8427e'
```

Рис. 5: Обчислення хеш-сум еталонних повідомлень

Значення хеш-сум обох вхідних рядків співпадають з еталонними, отже можна стверджувати, що алгоритм MD5 реалізований правильно.

4. ВИСНОВОК

Виконуючи дану лабораторну роботу, ми ознайомились з основними поняттями криптографії, а також поняттями дайджеста повідомлення та деякими засобами його отримання та перевірки.

А. ПОЧАТКОВИЙ КОД ПРОГРАМНОЇ РЕАЛІЗАЦІЇ МОДУЛЯ ШИФРУВАННЯ

Лістинг А.1: Початковий код програмного модуля для шифрування повідомлення

```
1  #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4  """
5  This script provides ciphers for lab 02.01 for NAU's Information Security
6  course.
7  """
8
9  import math
10 import random
11 from itertools import zip_longest
12
13
14 class SubstitutionEncryptor(object):
15
16     def __init__(self):
17         pass
18
19     def encrypt(self, message, key):
20         return "".join([key[s] for s in message])
21
22     def decrypt(self, message, key):
23         dec_key = {v: k for k, v in key.items()}
24         return self.encrypt(message, dec_key)
25
26
27 class TranspositionEncryptor(object):
28
29     def __init__(self):
30         pass
31
32     @staticmethod
33     def split_message(message, key):
34         offset = key
```

```

35         for i in range(0, len(message), offset):
36             yield message[i:i + offset]
37
38     @staticmethod
39     def transpose(matrix):
40         return list(map(list, zip_longest(*matrix)))
41
42     @staticmethod
43     def flatten(matrix):
44         return [item for row in matrix for item in row]
45
46     def encrypt(self, message, key):
47         res_matrix = self.flatten(
48             self.transpose(list(self.split_message(message, key)))
49         )
50         res = "".join((c for c in res_matrix if c is not None))
51         return res
52
53     def decrypt(self, message, key):
54         row_len = math.ceil(len(message) / key)
55         decrypted = self.encrypt(message, key=row_len)
56         return decrypted
57
58
59     class GammaEncryptor(object):
60         BYTE_MAX = 255
61
62         def __init__(self):
63             self.rng = random.SystemRandom()
64
65         @staticmethod
66         def xor(ba1, ba2):
67             return [b1 ^ b2 for b1, b2 in zip(ba1, ba2)]
68
69         def encrypt(self, message, key):
70             res = self._execute_round(message, key)
71             return res
72
73         def decrypt(self, message, key):
74             res = self.encrypt(message, key)
75             return res
76
77         def _execute_round(self, message, key):
78             if isinstance(message, str):
79                 message = message.encode()
80             if isinstance(key, str):
81                 key = key.encode()

```

```

82
83         if len(key) < len(message):
84             raise ValueError(
85                 "The key should be at least as long as the message."
86             )
87         return self.xor(message, key)
88
89     def get_key_for_message(self, message):
90         return [self.rng.randint(0, self.BYTE_MAX) for _ in
91                 ↵ message.encode()]
92
93     class AnalyticalEncryptor(object):
94         def __init__(self):
95             pass
96
97         def shift_enc(symbol, key=1, *args, **kwargs):
98             return chr(ord(symbol) + key)
99
100        def shift_dec(symbol, key=1, *args, **kwargs):
101            return chr(ord(symbol) - key)
102
103        @staticmethod
104        def t01_cipher_substitution(message, subst_func, *args, **kwargs):
105            """Implements an analytical cipher for task 04.
106
107            Args:
108                message (str): a plaintext message
109                subst_func (func): a function that performs substitution on an
110                                individual symbol.
111            Returns:
112                Ciphertext.
113            """
114            ciphertext = None
115
116            ciphertext = "".join([
117                subst_func(symbol, *args, **kwargs)
118                for symbol in message
119            ])
120
121            return ciphertext
122
123        def encrypt(self, message, key, subst_func=shift_enc):
124            res = self.t01_cipher_substitution(
125                message,
126                subst_func=subst_func,
127                key=key

```



```

128         )
129         return res
130
131     def decrypt(self, message, key, subst_func=shift_dec):
132         res = self.t01_cipher_substitution(
133             message,
134             subst_func=subst_func,
135             key=key
136         )
137         return res
138
139
140     def print_process(msg_in, key, msg_out, process):
141         print(
142             "Process: {}\n".format(
143                 "Input:  {}\n".format(
144                     "Key:   {}\n".format(
145                         "Output: {}\n".format(
146                             process, msg_in, key, msg_out
147                         )
148                     )
149                 )
150             )
151
152     def main():
153         # Task 01: substitution cipher
154         plaintext = "the quick brown fox jumps over the lazy dog."
155         key = {
156             "a": "w",
157             "b": "x",
158             "c": "y",
159             "d": "z",
160             "e": "a",
161             "f": "b",
162             "g": "c",
163             "h": "d",
164             "i": "e",
165             "j": "f",
166             "k": "g",
167             "l": "h",
168             "m": "i",
169             "n": "j",
170             "o": "k",
171             "p": "l",
172             "q": "m",
173             "r": "n",
174             "s": "o",

```

```

175         "t": "p",
176         "u": "q",
177         "v": "r",
178         "w": "s",
179         "x": "t",
180         "y": "u",
181         "z": "v",
182         " ": "0",
183         ".": ",",
184     }
185
186     se = SubstitutionEncryptor()
187     ciphertext = se.encrypt(plaintext, key)
188     print_process(plaintext, key, ciphertext, process="Encryption")
189
190     decrypted = se.decrypt(ciphertext, key)
191     print_process(ciphertext, key, decrypted, process="Decryption")
192
193     # Task 02: transposition cipher
194     td = TranspositionEncryptor()
195     key = 5
196     ciphertext = td.encrypt(plaintext, key)
197     print_process(plaintext, key, ciphertext, process="Encryption")
198
199     decrypted = td.decrypt(ciphertext, key)
200     print_process(ciphertext, key, decrypted, process="Decryption")
201
202     # Task 03: gamma encryption
203     ge = GammaEncryptor()
204     key = ge.get_key_for_message(plaintext)
205     ciphertext = ge.encrypt(plaintext, key)
206     print_process(plaintext, key, ciphertext, process="Encryption")
207
208     decrypted = ge.decrypt(ciphertext, key)
209     print_process(ciphertext, key, "".join(map(chr, decrypted)),
210                  process="Decryption")
211
212     # Task 04: analytic encryption
213     ae = AnalyticalEncryptor()
214     key = 5
215     ciphertext = ae.encrypt(plaintext, key)
216     print_process(plaintext, key, ciphertext, process="Encryption")
217
218     decrypted = ae.decrypt(ciphertext, key)
219     print_process(ciphertext, key, decrypted, process="Decryption")
220

```

```
221 if __name__ == "__main__":  
222     main()
```
