סטגנוגרפיה, או - על ביטים לא חשובים שחשובים

מאת ליאור פולק (מבוסס על מאמר מהבלוג: blog.liorp.dev)

הקדמה

יש שיאמרו שהצפנה היא המקצוע השני הכי עתיק בעולם; יש בידינו עדויות לשימוש בצפנים עוד מאז המאה ה-20 לפני הספירה, בקברו של חנומותפ השני במצרים.

אפילו מקור המילה עצמה - הצפנה, והפעולה ההפוכה לה - פענוח - מקורם בתנ"ך, בכינויו של לא אחר מאשר יוסף (צפנת-פענח).

דוגמה לצופן ותיק ופשוט הוא צופן אתב"ש. זהו צופן החלפה - צופן בו מוחלפות האותיות הראשונות בסדר האלפבית באותיות האחרונות בו - אל"ף מוחלפת באות ת"ו, בי"ת מוחלפת באות שי"ן, גימ"ל מוחלפת באות רי"ש וכן הלאה. כך, המילה "אבא" הופכת למילה "תשת", ובאנגלית המילה "mom" הופכת למילה "nln".

עם ההתפתחויות הטכנולוגיות במרוצת הדורות, פותחו הצפנות נוספות מסוגים שונים. אחת מהן נחשבת לידועה במיוחד - זוהי מערכת ההצפנה אניגמה (מיוונית: תעלומה, חידה), אשר שומשה בידי הגרמנים הנאצים במלחמת העולם השנייה על מנת להעביר מסרים צבאיים. האניגמה נחשבה כבלתי ניתנת לפיענוח, עד אשר לכדו האנגלים מערכת אניגמה, ולאחר עבודה מתמטית משמעותית ופורצת דרך, הצליחו לפענח את כל סודות התקשורת של הגרמנים במלחמת העולם השנייה. על הצוות שהצליח לפצח את האניגמה נמנה אלן טיורינג, שהיה לאבי מדעי המחשב המודרניים.

[אניגמה, מתוך ויקיפדיה, נחלת הכלל]

קריפטוגרפיה מודרנית 101

האם אי פעם חשבתם כיצד זה אפשרי לרכוש מוצרים בצורה בטוחה באינטרנט? כיצד ניתן לשלוח מייל בצורה מאובטחת מאדם לחברו, בלי שמאזין לקו השידור יוכל לקרוא את תוכנו? או כיצד ניתן לממש הצבעה אלקטרונית חשאית? כל הדברים הללו אפשריים בעולם התקשורת האלקטרונית בימינו בזכות ביסוסם על ענף במתמטיקה שנקרא קריפטוגרפיה (מיוונית, קריפטו - הסתר, גרפיה - כתיבה).

הקריפטוגרפיה המודרנית מושתת על שלוש רגליים:

חשאיות (confidentiality):

הרגל הראשונה מבין השלוש עוסקת בהבנת המידע המוצפן עצמו. במסגרת ההצפנה, המסר הגלוי מועבר למצב מוצפן. כדי לקרוא את תוכן המסר, יש צורך בהליך פענוח שנעשה על ידי המקבל ובמסגרתו המסר המוצפן חוזר להיות גלוי, לרוב בעזרת שימוש במידע משותף לשני הצדדים (כגון מפתח הצפנה). לעיתים מפתח ההצפנה זהה למפתח הפענוח ולעיתים שונה. ישנה חשיבות לעובדה כי אין יכולת פענוח למי שאין ברשותו את המפתח (זאת בדומה למפתח כניסה לבית).

בקריפטוגרפיה מודרנית בשני השלבים האמורים משתמשים השולח והמקבל בפרוטוקולים ובאלגוריתמים קריפטוגרפיים להשגת סודיות, כאשר השיטות עצמן אינן סודיות והן ידועות ומוסכמות מראש, ואילו מפתח ההצפנה עצמו סודי; זאת בשל עקרון קרקהופס, שקובע כי "האויב מכיר את המערכת שבשימוש".

אימות (authentication):

סודיות היא תנאי הכרחי אך לא מספיק. יש צורך בנוסף בפרוטוקול אימות זהויות שנועד למנוע התחזות וכן לספק דרך לדעת מיהו מקור המידע. בעולם הדיגיטלי של ימינו, ישנו שימוש נרחב במנגנון הנקרא חתימה דיגיטלית, כאשר השימוש בו דומה לפונקציה אותה ממלאת חתימה על גבי המחאה.

שלמות (integrity):

יש הבדל גדול בין המשפט "אתמול עברתי ליד בית ראש הממשלה" ובין המשפט "אתמול עברתי ליד ראש הממשלה". רוצה לומר, מסר איננו שווה הרבה אם איננו יודעים את כולו. הבטחת השלמות נעשית בדרך כלל על ידי אלגוריתם אימות שתפקידו להבטיח שהמידע אותנטי, כלומר שלא נעשה בו שינוי זדוני כלשהו על ידי צד שלישי. כמובן, יש לטפל גם במקרים בהם המסר הופל בשלמותו על ידי צד שלישי. יתר על כן, האלגוריתם בנוי כך שכל שינוי, אפילו קל מאוד, יתגלה מיד על ידי הצדדים בשיחה.

להרחבה בנושא, אני ממליץ לקרוא את הערך בויקיפדיה על קריפטוגרפיה וקריפטואנליזה.

סטגנוגרפיה

אחרי החימום הקל, אנחנו מגיעים לנושא המרכזי. ישנן עוד שיטות להסתיר מידע שאינן עושות שימוש במפתח הצפנה. זוהי בעצם הסתרת מסרים באופן שאף אחד זולת המקבל לא יוכל לראותם או לדעת על קיומם; זאת בניגוד לקריפטוגרפיה, שבה קיום המידע עצמו אינו מוסתר, אלא רק תוכנו. שיטות אלו נקראות בשם הכללי סטגנוגרפיה (כמובן מיוונית, סטגנו - חבוי, גרפיה - כתיבה).

באופן די צפוי, סטגנוגרפיה הייתה (ועודנה נמצאת) בשימוש נרחב על ידי ארגוני ביון וסוכני חרש בכל העולם.

סטגנוגרפיה היא בעצם הסיבה בגינה כתבתי את המאמר הזה. כחובב הקאתונים, רציתי לפתח הקאתון משלי. יצא שקראתי הרבה על אתגרים טכנולוגיים פוטנציאליים, שגם יהיו מהנים ומלמדים, וסטגנוגרפיה נשמעה לי כמו קונספט ממש מגניב שאפשר לבנות מעליו תרגיל נחמד.

איך בונים תרגיל שכזה? האסוציאציה הראשונה שעלתה לי בראש היא הסרט "מועדון קרב" (Fight Club). יש שמועיה עיקשת שבין כמה מהפריימים שבסרט הבמאי טמן מסרים סודיים בצורת פריימים שמופיעים לרגע ואז חולפים. מחשבה הובילה למחשבה ומכאן הבנתי שאפשר לעשות דברים דומים עם סוגים שונים של מדיה דיגיטלית, כמו קובצי תמונות גרפיות או קובצי שמע. כך, לדוגמה, ניתן להחליף את הביט הנמוך ביותר בכל בית (או מספר ביטים נמוך בכל פיקסל) בקובץ תמונה כלשהו, כך שהם יכילו את המסר הנסתר.

חשוב מאוד לשים לב שהשפעת שינוי זה על מראה התמונה זניחה, מאחר שרוב התקנים לקובצי תמונה (כמו TIFF או JPG) מאפשרים הדרגתיות גבוהה של צבעים (לרוב 8 ביט, כמו ב-JPEG, מה שמאפשר 256 צבעים שונים), הרבה יותר ממה שהעין האנושית מסוגלת לקלוט.

התמונה המקורית:

[המקור: On-the-internet-nobody-knows-you-are-a-dog]

התמונה המוצפנת (באדיבות המחבר):

לעין הבלתי מזוינת, שתי התמונות נראות זהות - אך למעשה, בתמונה השנייה מוצפן המסר:

My secret message

(עם מפתח ההצפנה המפורסם 12345678).

עבורי, לכתוב מימוש בקוד של קונספטים שאני נתקל בהם זו דרך ממש מהנה ליישם את הנושא שאני לומד, ולכן החלטתי לרשום קוד שעושה את התהליך הזה. התחלתי לכתוב ספריה קטנה בפייתון (1.6KB לאחר כיווץ!) שמבצעת הליך סטגנוגרפי פשוט על תמונות.

השוואה בין הבינארי של שתי התמונות. מימין - הקובץ המוצפן, משמאל - הקובץ המקורי:

348226	03	348204	c0
348227	a7	348205	29
348228	1c	348206	c7
348229	17	348207	85
348230	ba	348208	ae
348231	6e	348209	db
348232	6d	348210	1a
348233	20	348211	08
348234	ac	348212	6b
348235	fd	348213	ff
348236	0f	348214	c3
348237	0b	348215	42
348238	f5	348216	3d
348239	04	348217	41
348240	79	348218	5e
348241	01	348219	00
348242	f8	348220	fe
348243	ff	348221	3f
348244	cd	348222	28
348245	8f	348223	90
348246	7c	348224	5d
348247	88	348225	d9
348248	b8	348226	91
348249	f9	348227	61
348250	4e	348228	2c
348251	d6	348229	59
348252	00	348230	00
348253	00	348231	00
348254	00	348232	00
348255	00	348233	00
348256	49	348234	49
348257	45	348235	45
348258	4e	348236	4e
348259	44	348237	44
348260	ae	348238	ae
348261	42	348239	42
348262	60	348240	60
348263	82	348241	82
348264		348242	

הקוד מורכב משתי מחלקות: Encryptor ו-Decryptor:

ה-Encryptor מאותחל עם מפתח ומסר, ולאחר מכן מבצע הצפנה של המסר בתוך תמונה. מבחינה טכנית, המסר מקודד בפורמט של header שמכיל את אורך ההודעה, ולאחר מכן את ההודעה המוצפנת עצמה. הפלט הזה מחולק לביטים ש"נדחפים" במקום ה-LSB של כל פיקסל בתמונה, עד שכל המסר מוטמע בתוך התמונה:

```python
from base64 import urlsafe_b64encode
from hashlib import md5
from typing import Tuple, Iterator

import imageio
from cryptography.fernet import Fernet

from utils import str2bin


class Encryptor:
    """Responsible for steganography of many images and messages using a given key"""
    def __init__(self, password: str):
        _hash = md5(password.encode()).hexdigest()
        cipher_key = urlsafe_b64encode(_hash.encode())
        self.__encryptor = Fernet(cipher_key)

    def __get_data(self, message: str) -> Tuple[Iterator, str]:
        # Preparing the data for writing: length+data
        encrypted_message = self.__encryptor.encrypt(message.encode())
        data_length = format(len(encrypted_message)*8, '032b')
        data = iter(data_length + str2bin(encrypted_message.decode()))
        return data, data_length

    @staticmethod
    def __check_encoding_capacity(height: int, width: int, data_length: int) -> int:
        encoding_capacity = height * width * 3
        if data_length > encoding_capacity:
            raise ValueError("The data size is too big to fit in this image!")
        return encoding_capacity

    @staticmethod
    def __insert_data_to_image(im, data: str):
        new_im = im.copy()
        height, width, _ = new_im.shape
        modified_bits = 0
        try:
            for i in range(height):
                for j in range(width):
                    pixel = new_im[i, j]
                    for k in range(3):
                        # To replace the LSB with b, where b can be either 0 or 1, you can use (n & ~1)
| b.
                        pixel[k] = (pixel[k] & ~1) | int(next(data))
                        modified_bits += 1
        except StopIteration:
            pass
        return new_im, modified_bits

    def encrypt(self, im, message: str, out_path: str = None) -> Tuple[bytes, float]:
        """
        Encrypts `message` inside `image` and writes output to `out_path`.
        Returns the encrypted image, and loss (percentage of how many bits were changed).
        """
        data, data_length = self.__get_data(message)
        height, width, _ = im.shape
        encoding_capacity = self.__check_encoding_capacity(height, width, int(data_length, 2))

        new_im, modified_bits = self.__insert_data_to_image(im, data)

        imageio.imwrite(out_path, new_im)
        loss_percentage = (modified_bits / encoding_capacity) * 100
        return im, loss_percentage
```

ההצפנה עצמה נעשית באמצעות אלגוריתם Fernet, שבתורו מבוסס על AES. הוא גם יודע לחשב את ההפסד - זאת אומרת, כמה אחוז מהביטים שונו על מנת להצפין את המסר בתוך הביטים האחרונים של ההודעה.

ה-Decryptor מאותחל עם מפתח ואז מבצע את פענוח ההודעות בתוך תמונות:

```python
from base64 import urlsafe_b64encode
from hashlib import md5

from cryptography.fernet import Fernet

from utils import bin2str


class Decryptor:
    """Responsible for decrypting steganography of many images using a given key"""
    def __init__(self, password: str):
        _hash = md5(password.encode()).hexdigest()
        cipher_key = urlsafe_b64encode(_hash.encode())
        self.__encryptor = Fernet(cipher_key)

    @staticmethod
    def __extract_data_from_image(im, data_length, header_length=32) -> str:
        data = ""
        height, width, _ = im.shape
        try:
            for i in range(height):
                for j in range(width):
                    pixel = im[i, j]
                    for k in range(3):
                        # Skip the header
                        if header_length:
                            header_length -= 1
                            continue

                        # Get the lsb
                        data += str(pixel[k] & 1)
                        data_length -= 1
                        if data_length == 0:
                            raise StopIteration
        except StopIteration:
            pass
        return bin2str(data)

    @staticmethod
    def __extract_data_length_from_image(im, header_length=32) -> int:
        data_length = ""
        height, width, _ = im.shape
        try:
            for i in range(height):
                for j in range(width):
                    pixel = im[i, j]
                    for k in range(3):
                        # Get the lsb
                        data_length += str(pixel[k] & 1)
                        header_length -= 1
                        if header_length == 0:
                            raise StopIteration
        except StopIteration:
            pass
        return int(data_length, 2)

    def decrypt(self, im) -> str:
        """Decrypts the hidden message in `im`."""
        data_length = self.__extract_data_length_from_image(im)
        data = self.__extract_data_from_image(im, data_length)
        message = self.__encryptor.decrypt(data.encode()).decode()
        return message
```

לבסוף, אחרי כמה מאבקים עם דחיסת JPEG (מי ידע שאלגוריתם דחיסה lossy אומר שהנתונים יכולים להשתנות לאחר החילוץ?), המימוש עבד, תוך שימוש ב-2 תלויות בלבד - cryptography ו-imageio.

כדי להעלות את החבילה השתמשתי בכלי חביב ומוצלח בשם poetry, וכעת הוא זמין ב-pypi - אפשר להריץ:

pip install pysteg

ולהתנסות בספריה בעצמכם!

הינה דוגמא לשימוש:

```
import imageio
from cryptography.fernet import Fernet

from encrypt import Encryptor
from decrypt import Decryptor


def load_image(path: str):
    return imageio.imread(path)


def load_password(password: str) -> Fernet:
    return Fernet(password.encode())


def main():
    in_path = "./dogcyber.png"
    out_path = "./dogscybersteg.png"
    password = "12345678"
    message = "My secret message!"

    im = load image(in path)

    encryptor = Encryptor(password)
    encryptor.encrypt(im, message, out_path)

    decryptor = Decryptor(password)
    print(decryptor.decrypt(load_image(out_path)))


if __name__ == '__main__':
    main()
```

קוד המקור זמין כאן:

https://github.com/liorp/pysteg

והמאמר מבוסס על גרסה קצרה שמופיעה בבלוג שלי:

https://blog.liorp.dev/development/steganography-aka-encryption-and-cyber/

סיכום

הצפנה היא קונספט מאוד מעניין, וקצרה היריעה של מאמר זה מלכסות אפילו קמצוץ מתכולת התחום המרתק הזה.

לכל מי שחפץ ללמוד עוד בנושא, אני ממליץ על הקורס המצויין של פרופסור דן בונה מאוניברסיטת סטנפורד, שזמין ברשת בחינם באתר קורסרה:

https://www.coursera.org/learn/crypto

כפי שכתבתי, אני ממליץ לכל מי שקורא על קונספטים מעניינים להתנסות בהם בעצמכם על ידי מימוש שלהם, ומי יודע, אולי אפילו יצא לכם מאמר ב-DigitalWhisper ☺

ביבליוגרפיה

ויקיפדיה:

https://en.wikipedia.org/wiki/Cryptography

מכון ויצמן:

https://stwww1.weizmann.ac.il/communication/?p=468