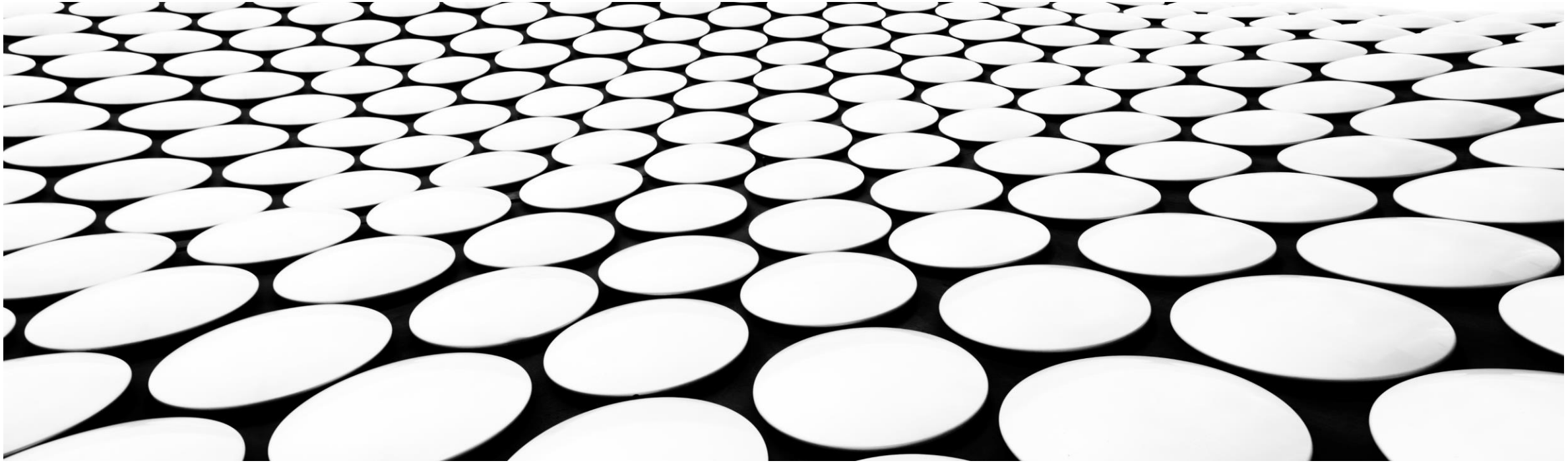
יסודות הקריפטוגרפיה
(מיקוד: אבטחת מידע)



מבוא

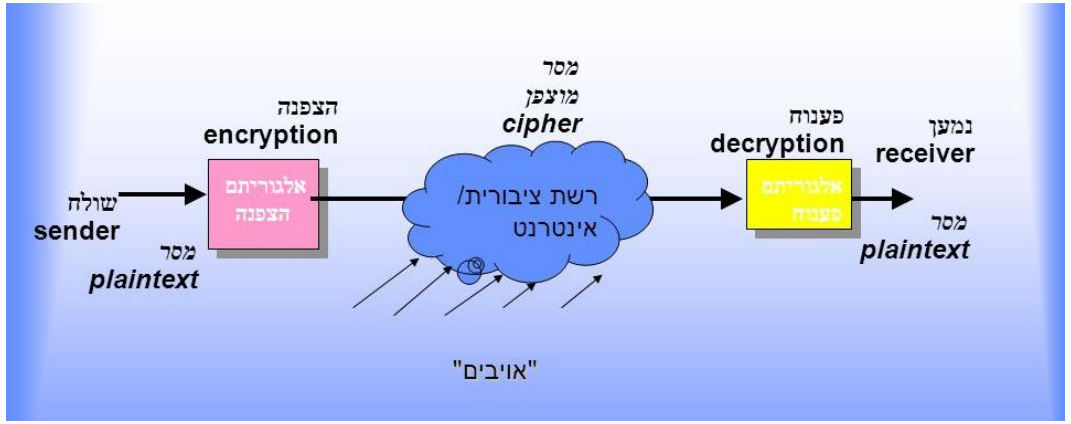
- הסבר
  - **תורה** העוסקת בעקרונות ובשיטות של הצפנת מידע. העברת מידע למצב שבו הוא קריא רק לאנשים מסוימים
  - מיוונית: kryptós – הסתרה, graphein – כתיבה
  - מבוסס על משפחה של אלגוריתמים, פרוטוקולים, וטכניקות
  - איפה משתמשים כיום בקריפטוגרפיה?
- מטרות הקריפטוגרפיה
  - סודיות (confidentiality)
  - שלמות / אמינות (integrity)
  - אימות (authenticity)

מה לא נלמד

- מתימטיקה (רק קצת עקרונות בלבד)
- כל ההיסטוריה
- פוליטיקה וחוקיות
- שימושים מתקדמים
- הצפנה קוונטית
- כל הפרמטרים האפשריים, כל מצבי עבודה אפשריים, כל השילובים האפשריים, כל היכולות האפשריים, וכו וכו
- לא נדבר על הכול – ויש הרבה

הגדרות כלליות

- **קריפטולוגיה:** תורת ההסתרה.
- **Plaintext:** המסר המקורי.
- **Ciphertext:** המסר המוצפן.
- **הצפנה:** התהליך מ- Plaintext ל- Ciphertext.
- **פענוח:** התהליך מ- Ciphertext ל- Plaintext.
- **צופן:** שיטת ההצפנה הסודית (האלגוריתם).
- **מפתח:** מידע קריטי שמשתמש בו אלגוריתם ההצפנה, הידוע לשולח ולמקבל בלבד.
- **קריפטואנליזה:** ניסיון לגלות את ה- Plaintext או את מפתח ההצפנה או את שניהם.



הצפנה = Encryption, צופן = Cipher

נושאים בקריפטוגרפיה

- Hash גיבוב
- הצפנה
  - סימטרי
  - א-סימטרי
- מספרים רנדומליים
- מפתחות ציבוריים
- החלפת מפתחות
- חתימות דיגיטליות
- פרוטוקולים AES, DES...

הצפנה
(Cipher = צופן , Encryption = הצפנה)

תקשורת מחשבים - הנחות עבודה

- תקשורת בין מחשבים היא גלויה לכולם.
- לא ניתן לדעת בוודאות מי עומד בצד השני של הקו
- לא ניתן להבטיח שאין צד שלישי שמאזין לתקשורת ומקבל אליו את כל המידע שעובר.

עקרונות אבטחת מידע

- בניגוד למה שחושבים, מידע מוצפן אינו מידע בטוח, כדי שמידע יהיה בטוח דרושים להתקיים שלושה דברים :

1. **הצפנה המידע הנשלח, סודיות (Confidentiality) -** כך שרק השולח ומקבל ההודעה יוכלו להבין את תוכנה. וחשוב לא פחות – אלגוריתם ההצפנה חייב להיות **חזק**, אחרת ההצפנה תישבר.
2. **שלמות/אמינות המידע שהתקבל (Integrity)** – השולח והמקבל רוצים לוודא שההודעה לא שונתה במהלך שליחתה בתווך בו עברה, ואם שונתה נוכל לדעת על זה מיד.
3. **אוטנטיות מקור המידע שהתקבל** (**Authenticity**) – כל צד יוכל לאמת את זהותו של הצד האחר

CIA = **Confidentiality, Integrity, Authenticity**

מה זה אומר הצפנה חזקה

הצפנה צריכה להיות חזקה, זאת אומרת שאם התוקף יודע:

- את האלגוריתם
- חלקים מהטקסט המקורי
- חלקים מהטקסט המוצפן

הוא **לא יוכל לגלות** את מפתח ההצפנה שיפענח את כל הטקסט

תוצאה פרקטית: חוזק ההצפנה כחוזק המפתח

האם אפשר ליצור אלגוריתם הצפנה שלא ניתן לפרוץ?
<u>בתאוריה</u>: לא.
<u>במציאות:</u>
אם עלות שבירת הצופן עולה על הערך של המידע
והזמן הדרוש כדי לשבור את הצופן עולה על זמן הרלוונטיות של המידע
אז האלגוריתם נחשב לחסין פריצות.

חוקי יסוד בהצפנה

- אין להמציא לבד אלגוריתם או פרוטוקול (Don’t Roll Your Own)
  - כל אחד יכול להמציא אלגוריתם הצפנה **שהוא עצמו** לא מסוגל לשבור, הרבה יותר קשה להמציא אלגוריתם **שאף אחד** לא יכול לשבור
- יש להשתמש בספריות קיימות ובדוקות ואין לממש לבד אלגוריתם (אפילו אחד מוכר לכם היטב)
- לא לשכוח להגן על המפתח הפרטי! (הגורם האנושי הוא החוליה החולשה בכל הצפנה)

קריפטואנליזה מהי

- התהליך הנדרש כדי למצוא את הטקסט המפורש או המפתח. קימות שלושה דרכים עיקריות:

1. **Brute Force -** לנסות את כל האופציות שקיימות למפתח.
אם המפתח ארוך מדי, עלות משאבי המחשוב שתידרש לפיצוח הופכת ללא כדאי, מה שהופך את הצופן "שביר" באופן מעשי.

2. **ניצול חולשות -** של האלגוריתם או של המפתח
(למשל אם המפתח נוצר מכיל רק ספרות או רק אותיות אנגליות קטנות, אם יש חזרות של אות מסוימת יותר מאחרות וכו').

3. **"הנדסת אנוש" -** ניצול חולשות אנושיות כדי לקבל את המידע על ההצפנה במרמה

בהמשך תקבלו תרגיל בפיצוח מפתח, כאשר ייתן לכם סוג של רמז על תכולת המפתח

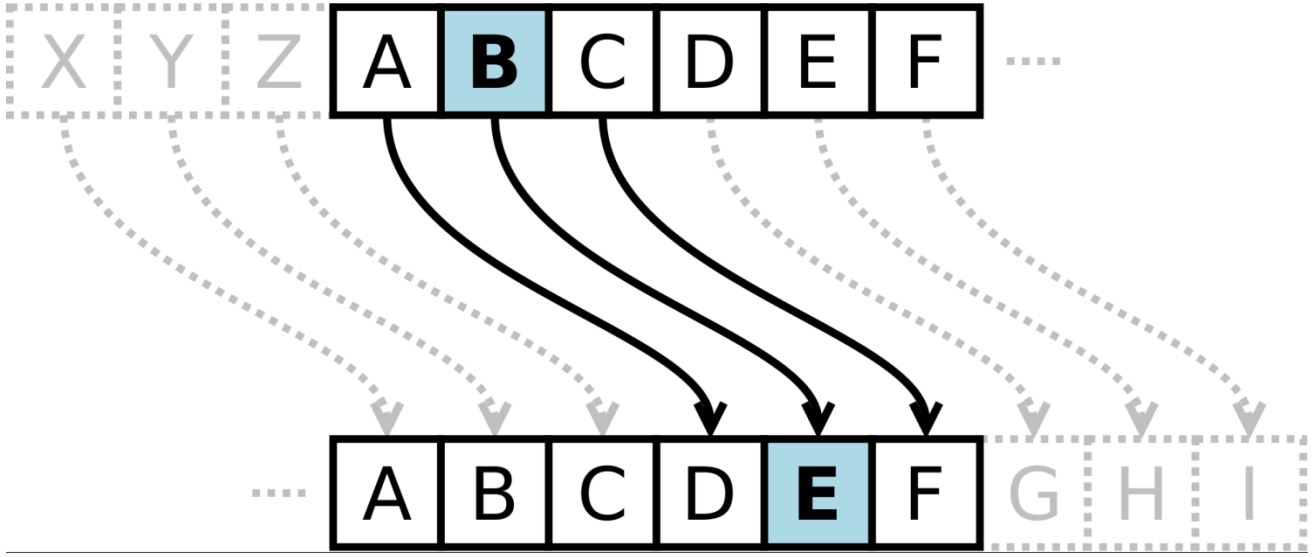
סוגי הצפנה, אלגוריתמי הצפנה ופרוטוקולי הצפנה

- סוגי הצפנות
  1. הצפנה פשוטה (נראה רק לצורך המחשה מדוע אינה טובה, וגם ננסה לשפר)
  2. פונקציית גיבוב (Hashing Function) - הצפנה חד כיוונית, קל להצפין קשה לגלות.
  3. הצפנה סימטרית
  4. הצפנה אסימטרית
- אלגוריתמי הצפנה למידע : Diffie-Hellman, RSA, AES, DES, PGP (Pretty Good Protection) שהוא חזק יותר מ AES . האלגוריתמים משתמשים בקומבינציה של סוגי ההצפנות מעלה כדי ליצור מסגרת הצפנה טובה יותר
- פרוטוקולי הצפנה : SSL, TLS

סוגי הצפנה

צופן קיסר – הצפנה פשוטה

- צופן קיסר (Caesar Cipher) – בהצפנה זו יש הזזה מעגלית של מספר אותיות קבוע. הקשר בין אות בטקסט המקורי לאות בטקסט המוצפן הוא 1 ל-1.



בציור צופן קיסר עם הזזה של 3

דוגמת קוד : CheaserCipher.py

שיטת ההחלפה – הצפנה פשוטה

בשיטה זו, השולח והמקבל מחליטים על שיטת הצפנה משלהם, לדוגמא החלפת האותיות ע"פ הטבלה הבאה.

א	ב	ג	ד	ה	ו	ז	ח	ט	י	כ	ל	מ	נ	ס	ע	פ	צ	ק	ר	ש	ת
ק	י	נ	ל	ט	א	ח	פ	צ	ד	ה	ו	ר	ע	ש	מ	ג	ס	ב	כ	ת	ז

Plaintext:

אין לראות את הדברים היטב אלא בלב בלבד

Ciphertext:

קדע ונקאז קז טלינדר טדצי קוק יוי יויל

מאחר שאין כאן חוקיות כלשהיא , יקשה מאוד על המאזין לפענח את השדר.

קריפטואנליזה של שתי השיטות הפשוטות

השיטה	צופן קיסר	צופן החלפה
Brute Force	המפתח הוא מספר אחד בין 1 ל- 25 והקשר בין האתיות בטקסט המוצפן לטקסט הרגיל הוא 1 ל-1. לכן צריך פשוט לנסות 25 אופציות - קל מאוד.	בלתי אפשרי המפתח הוא באורך 26 (הקשר בין אות בטקסט המוצפן לאות בטקסט הרגיל הוא 1 ל-26), מספר האפשרויות למפתח הוא: 26! = 4*10^26.
חולשת האלגוריתם	אלגוריתם פשוט ומפתח קצר.	הסטטיסטיקה של אותיות האנגלית ידועה לכן אפשר לגלות את הטקסט המקורי על ידי ניתוח תדירויות האותיות וניתוח תדירויות של צירופים – קל מאוד.

תרגיל 3 – שיפור חולשות לצופן קיסר

- לצופן קיסר ולשיטת ההחלפה החולשות הבאות:
  - מפתח הזזה קבוע 1-25 קיסר
  - רווח נשאר רווח קיסר
  - אין טיפול בתווים כמו מספרים , נקודה , פסיק ורבים אחרים $, !, *,), (וכדומה...
  - כאשר מילה / אות חוזרת פעמים רבות ניתן לפצח את הקוד – החלפה
  - אין טיפול בקבצי טקסט אם מספר שורות

- כתבו תוכנית לקידוד צופן קיסר משופר, המבוסס על עקרונות קיסר אבל משפר את חולשות צופן קיסר ושיטת ההחלפה שהוזכרו מעלה.
  - יש לכתוב את קוד הצפנה הקורא מקובץ טקסט מצפין אותו מציג את תוצאת ההצפנה וכותב את ההצפנה לקובץ חדש.
  - וקוד/פעולה נוסף המקבל את קובץ המוצפן מפענח אותו ומציג את התוצאה.

ספריות הצפנה בפיתון

- יש ספריות רבות נתמקד בשלושה עיקריות

1. hashlib - סיפריה מובנית לטיפול בפונקציות גיבוב
2. Cryptodome (install as pycryptodome, import Crypto in python) API_Book
3. cryptography (import cryptography)

- ספריות 2, 3 ממשות פונקציות רבות זהות, אולם בצורת ממשק שונה.
- ספריה 2 היא הפופולרית ביותר וברוב המקרים נשתמש בה.
- לספריה 3 יש את מרחב הכיסוי הגדול ביותר אולם זו גם הספרייה המורכבת ביותר לשימוש.
- סביר להניח שתצטרכו את כולם בשלב כזה או אחר.

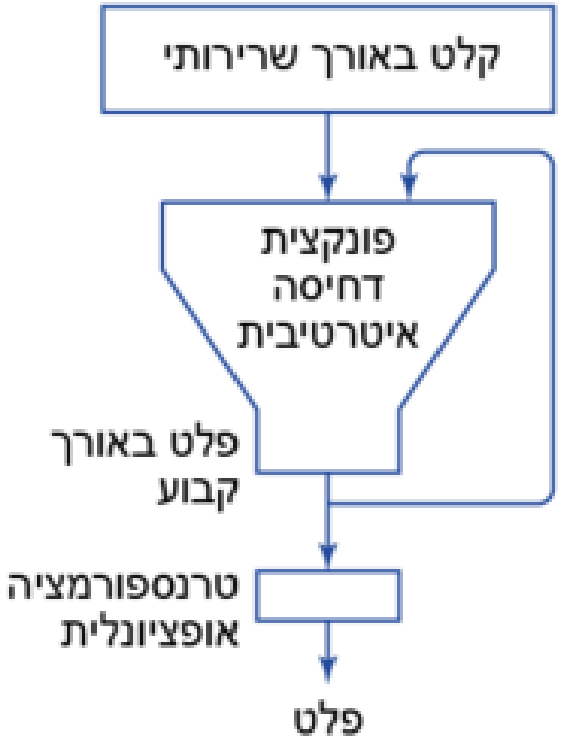
פונקציית גיבוב – Hashing Function

- תזכורת, לאבטחת מידע שלושה עקרונות:
  - סודיות (confidentiality)
  - שלמות / אמינות (integrity)
  - אימות (authenticity)

- **בצופני קיסר והחלפה ראינו שניתן לשפר את הקוד כך שיקשה מאוד לפענח אותו**
- **עדיין אין לנו דרך לדעת האם המידע שהתקבל הוא אכן המידע שנשלח**
- **לצורך זה וצרכים נוספים, נכיר את פונקציית הגיבוב** – Hashing function

פונקציית גיבוב קרטוגרפית - Hashing Function

- פונקציה חד-כיוונית **הממירה קלט באורך כלשהו לפלט באורך קבוע וידוע מראש**.
- מתוכננת כך, שכל שינוי קטן בקלט יגרום לשינוי משמעותי בפלט. בדרך זו ניתן להתמודד עם בעיית הבטחת שלמות מסרים גדולים, על ידי השוואת הערך המגובב שלהם במקום השוואה ישירה. בשל היותו קטן משמעותית, קל יותר להגן על הערך המגובב מאשר על המסר המקורי.
- פונקציות גיבוב הינה מאבני הבסיס של ההצפנה המודרנית ומשמשות :
  - כחתימות דיגיטליות,
  - קודי אימות, הבטחת שלמות המידע
  - הגנת סיסמאות
- ביישומים **שאינם** קריפטוגרפים הן משמשות לעיתים **כמזהה ייחודי של קובץ** לצורך בדיקת שלמותו או נכונותו וכן לזיהוי קבצים זהים. או שימושיים **לסכמת התחייבויות** (אליס יודעת סוד שבוב מטיל ספק אם יודעת, אליס שולחת את פונקציית הגיבוב של הסוד לבוב, כאשר הסוד מתגלה בוב יכול ליצר ממנו גיבוב ולהשוות למה ששלחה אליס...)



סוגי פונקציות גיבוב

- MD5 = Message Digest משתמש ב 128 bit להצפנה
- SHA-1 = Secure Hashing Algorithm משתמש ב 160 bit להצפנה
- SHA-2 – אלגוריתם מתקדם מזה של SHA1 ותומך 224, 256, 384, 512 ביט להצפנה
- SHA-3 – אלגוריתם דומה ל SHA-2 אבל נחשב בטוח יותר ע"י זה שאינו פגיע להתקפת **length extension attack** בעוד שהקודמים כן חשופים לסוג זה של התקפה.

איך משתמשים בפונקציית גיבוב בפיתון

- קיימים כמה סוגים : MD5, SHA160, SHA256, SHA512 כל אחד "בחוזק" שונה:
  - md5 – 128 bit (MD5) (מיושן וחלש)
  - sha1 (SHA-1) – 160 bit (תלוי בשימוש)
  - sha256 , sha512 (SHA-2) – 256 -512 bit (מודרני וחזק)
  - Sha3_256... - 224 – 512 bit (מודרני חזק ומאובטח יותר)

שימוש עם פיתון:

```
import hashlib

m = hashlib.md5()   - הגדרת האובייקט של ההצפנה
m.update(b"text to makes hashing on")
m.hexdigest() // or m.digest()

When defending m object you may select: haslib.sha160() or hashlib256() or any
length supported in the library!!
```

דוגמת קוד לשימוש בפונקציות הגיבוב השונות: HashBasics.py

פונקציית גיבוב כמנגנון אימות

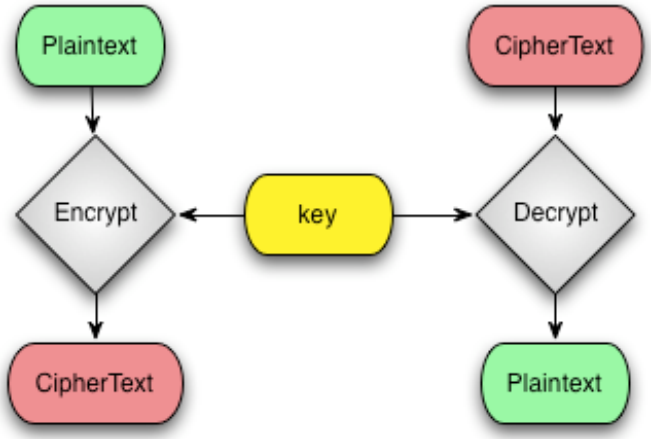
- אימות נתוני תקשורת שנשלחו
- אימות סיסמאות שמתקבלות מול מסד נתונים שבהם הם מאוחסנים כפונקציית גיבוב
- אימות קבצים שנשלחו
- וכו'

דוגמא : HashOnFile.py

זה גם הזמן לקחת פסק זמן ולתרגל:
תרגיל 4 – חישוב מבוזר לשבירת קוד שהוצפן ע"י פונקציית גיבוב של 128bit (MD5)

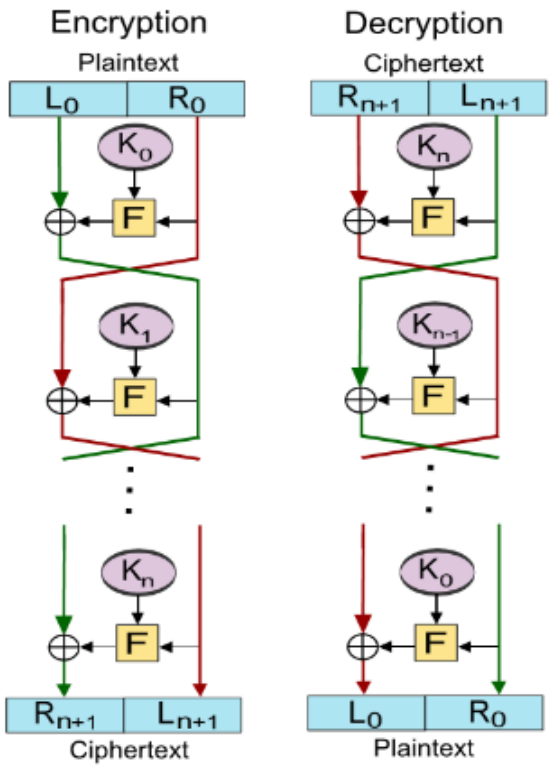
הצפנה סימטרית

- מה זה?
  - הצפנה שמשתמשת במפתח יחיד להצפנה ופיענוח(השולח והמקבל משתמשים באותו מפתח שנשמר בסוד)
  - אלגוריתם להצפנה ולפיענוח הפוכים אבל זהים
  - מפתח סודי / מפתח משותף – רנדומלי חזק וארוך
- למה להשתמש בזה?
  - להגן על סודיות
  - חזק, מהיר, ונחשב ל(כמעט) לא שביר עם מיושם נכון
  - אבל צריך לנהל מפתח(ות) סודי ולהגן עליו
- חולשות – שמירה על המפתחות , ניהול מפתחות רבים בתקשורת מרובת לקוחות



הצפנה סימטרית המשך

- הצפנת ביטים – במקום להצפין אותיות מצפינים ביטים.
- **הצפנת XOR:**
בהצפנה זו בוחרים מפתח שהוא רצף סיביות אקראי ומבצעים פעולת XOR על כל הטקסט.
- ככל שמפתח ההצפנה גדול יותר, יותר קשה לפענח את המסר.
- הצפנה זו שכיחה מאוד בצפנים מורכבים יותר.
- היתרון העיקרי: קל ליישום וזול.
- **חולשת עבר** - בשימוש בהצפנה זו ללא לפי הספר ניתן לפענח את הצפנת ה- XOR באמצעות ניתוח תדירות הופעת תווים (אותו קטע בטקסט מתפרש לאותו קטע מוצפן (חולשה זו כבר לא קיימת כיום באלגוריתמים המתקדמים)



אלגוריתמים המשתמשים בהצפנה סימטרית

- מיושן וחלש:
  - **DES** – 56 ביט, משנת 1975
  - **3DES** – 168 ביט (אבל שווה ל-112 ביט בלבד)
  - **RC4** – 40-2048 ביט
- מודרני וחזק:
  - **AES** – 128, 192, או 256 ביט
- AES = Advanced Encryption Standard

אנחנו נתמקד באלגוריתם AES שהו הסטנדרט "דה-פקטו" כיום ונחשב לבלתי "פריץ"

הצפנת AES (Advanced Encryption Standard = AES

- זוהי ההצפנה שבה כולם משתמשים כיום
- **עד כה** נחשבת לבלתי ניתנת לשבירה
- זוהי הצפנת **בלוקים** סימטרית , צופן בלוקים פועל על מחרוזת סיביות באורך קבוע (מקבל בלוק גלוי ומחזיר בלוק מוצפן) , **בסימטרית** הכוונה שאותו מפתח משמש להצפנה ולפיענוח.
- ב AES גודל הבלוק הוא 128bit וגודל המפתח בטווח של 128-256bits
- ל AES קיימים כמה אופני הפעלה (Mode of Operation), כל אופן הפעלה מתייחס לאלגוריתם שמגדיר כיצד להפעיל צופן בלוקים סימטרי דטרמיניסטי להצפנת **טקסט-גלוי באורך שעולה על אורך הבלוק שהצופן מסוגל להצפין בבת אחת**
- **ההסבר על איך עובד אלגוריתם AES אינו בסקופ של התוכנית השנה ,אבל מי שרוצה יכול לצפות בסרטון הבא:** How AES Works **שמסביר בצורה יפה איך האלגוריתם עובד**

אופני הפעלה עיקריים ב-AES

- חמשת הבסיסים ביותר הינם : CFB, CBC,ECB OFB ו-CTR
- אנחנו נתמקד רק בשניים מהם : CBC ו- CTR (GCM) שנחשבים המאובטחים ביותר
  - **CBC = Cipher-Block Chaining (חיסרון עיקרי לא ניתן לחישוב מקבילי)**
  - **CTR = Counter mode (יתרון עיקרי , ניתן לחישוב מקבילי)**
  - **GCM** - Galois/Counter Mode צופן סימטרי במצב מונה ופונקציית גיבוב אוניברסלית

- **הסבר מפורט על כל אופני ההפעלה AES: אופני הפעלה AES**

דוגמאות קוד לשימוש בהצפנת AES

CBC Mode

```
from Crypto.Cipher import AES
from Crypto.Util import Counter
Import hashlib
# ENCODING
 plain_text = "this is my text to encrypt"
 password = "1234".encode()
 key = hashlib.sha256(password).digest()
 mode = AES.MODE_CBC
 IV = '0123456789ABCDEF'.encode()  # should be 16 bytes
 cipher = AES.new(key, mode, IV)
 padded_message = pad_message(plain_text.encode())
 encryped_message = cipher.encrypt(padded_message)
 return encryped_message

# DECODING
encryped_message="(*(79&*(&)&9&(&"
password = "1234".encode()
 key = hashlib.sha256(password).digest()
 mode = AES.MODE_CBC
 IV = '0123456789ABCDEF'.encode()  # should be 16 bytes
 cipher = AES.new(key, mode, IV)
 dec_text = cipher.decrypt(encryped_message)
 dec_text.rstrip(b'0')
 return dec_text
```

CTR Mode

```
from Crypto.Cipher import AES
from Crypto.Util import Counter
Import hashlib
# ENCODING
def encrypt(plain_text):
    password = "1234".encode()
    key = hashlib.sha256(password).digest()
    mode = AES.MODE_CTR
    cipher = AES.new(key, mode, counter=Counter.new(128))
    encryped_message = cipher.encrypt(plain_text.encode())
    return encryped_message

# DECODING
def decrypet(cipher_text):
    password = "1234".encode()
    key = hashlib.sha256(password).digest()
    mode = AES.MODE_CTR
    cipher = AES.new(key, mode, counter=Counter.new(128))
    decrypted_text = cipher.decrypt(cipher_text)
    return decrypted_text
```

AES.py – דוגמאת קוד

תרגיל 5 כיתה (השלמה בבית)

- קחו קובץ אקסל או קובץ תמונה כלשהו, הצפינו אותו בעזרת AES במוד CTR ושימרו הנתונים בקובץ חדש
- כתבו תוכנית המקבלת את הקובץ המוצפן מפענחת אותו וכותבת את התוצאה לקובץ חדש.
- בדקו שהקובץ החדש שקיבלתם אכן עובד וניתן לפתוח אותו בעזרת תוכנית אקסל או עורך תמונות

- כתבו תוכנית עם שרת ולקוח מעל האינטרנט (Socket), כאשר **הלקוח מבקש קובץ** מהשרת, השרת מצפין את הקובץ ושולח את הקובץ המוצפן ע"י AES ללקוח, הלקוח מפענח את הקובץ שהתקבל, מאמת שאכן מה שנשלח הוא מה שהתקבל ושומר את הקובץ. (עשו על קובץ בינארי כל שהו , לא טקסט)

סיכום ביניים

- הבנו מהי פונקציית גיבוב ולמה אפשר להשתמש בה (חתימה דיגיטלית כדי לבדוק אמינות המידע שהתקבל (Integrity) , הצפנה חד כיוונית), תרגיל 4 ניסיתם לפצח קוד גיבוב שניתן לכם
- למדנו מהי **הצפנה סימטרית** (מפתח יחיד גם להצפנה וגם לפיענוח), התמקדנו **בהצפנת AES** (Advanced Decryption Standard) והבנו את מודי ההפעלה של הצפנת AES (תרגיל 5 הצפנת קובץ)
- נכון להיום הצפנת AES נחשבת לבלתי פריצה.
- אז איפה הבעיה...סימטריות המפתח

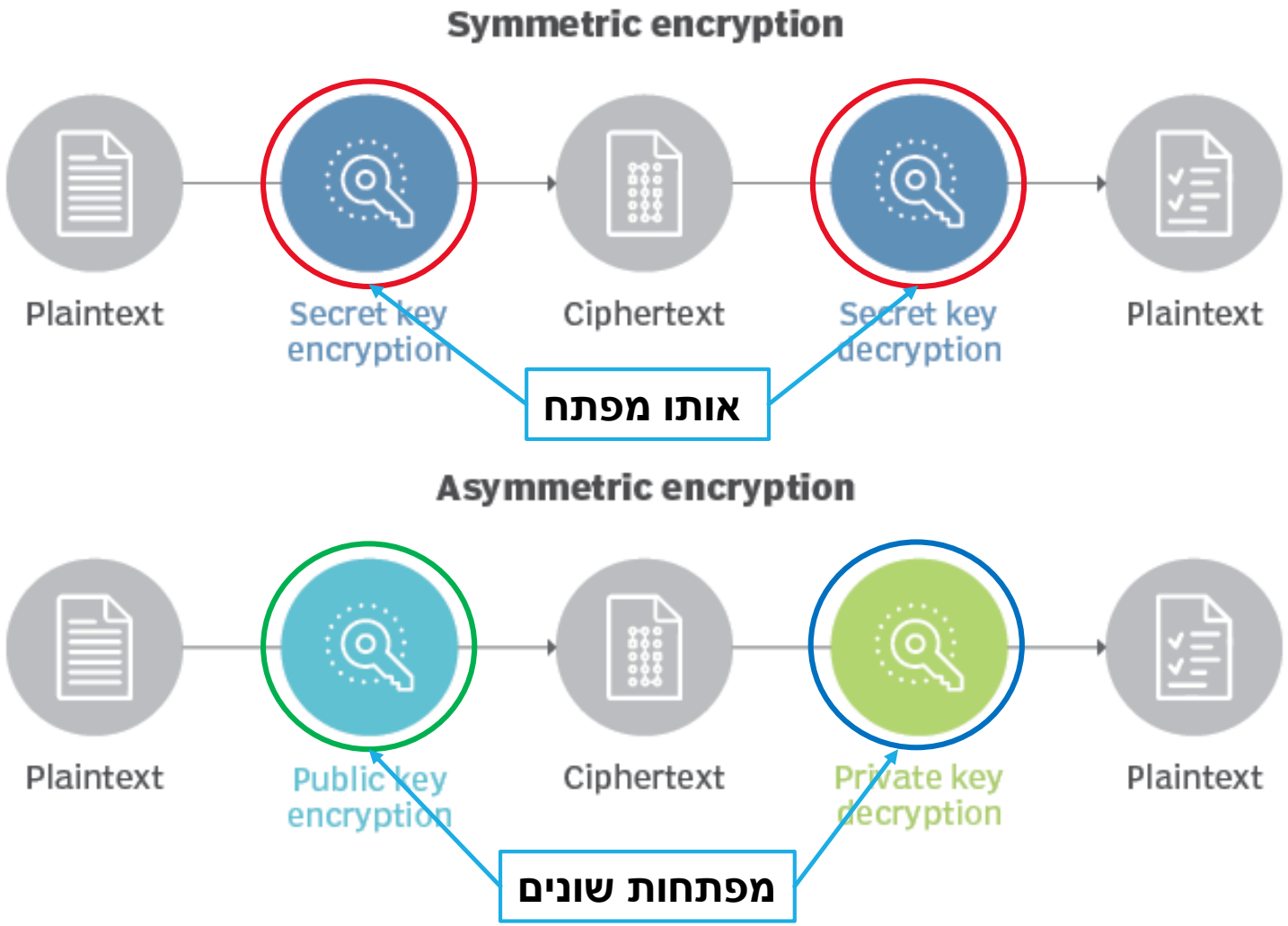
בעיות בהצפנה סימטרית:

1. איך להעביר את המפתח בצורה סודית?
2. ניהול מפתחות מרובים ללקוחות מרובים
3. איך לוודא שהאדם ששלח את ההודעה הוא אותו האדם שאתה חושב – התקפת MitM

- חסרונות אילו הביאו לפיתוחה של **הצפנה אסימטרית**.

הצפנה אסימטרית
Asymmetric Encryption

הצפנה סימטרית לעומת הצפנה אסימטרית



פרוטוקולים עיקריים בהצפנה סימטרית ואסימטרית

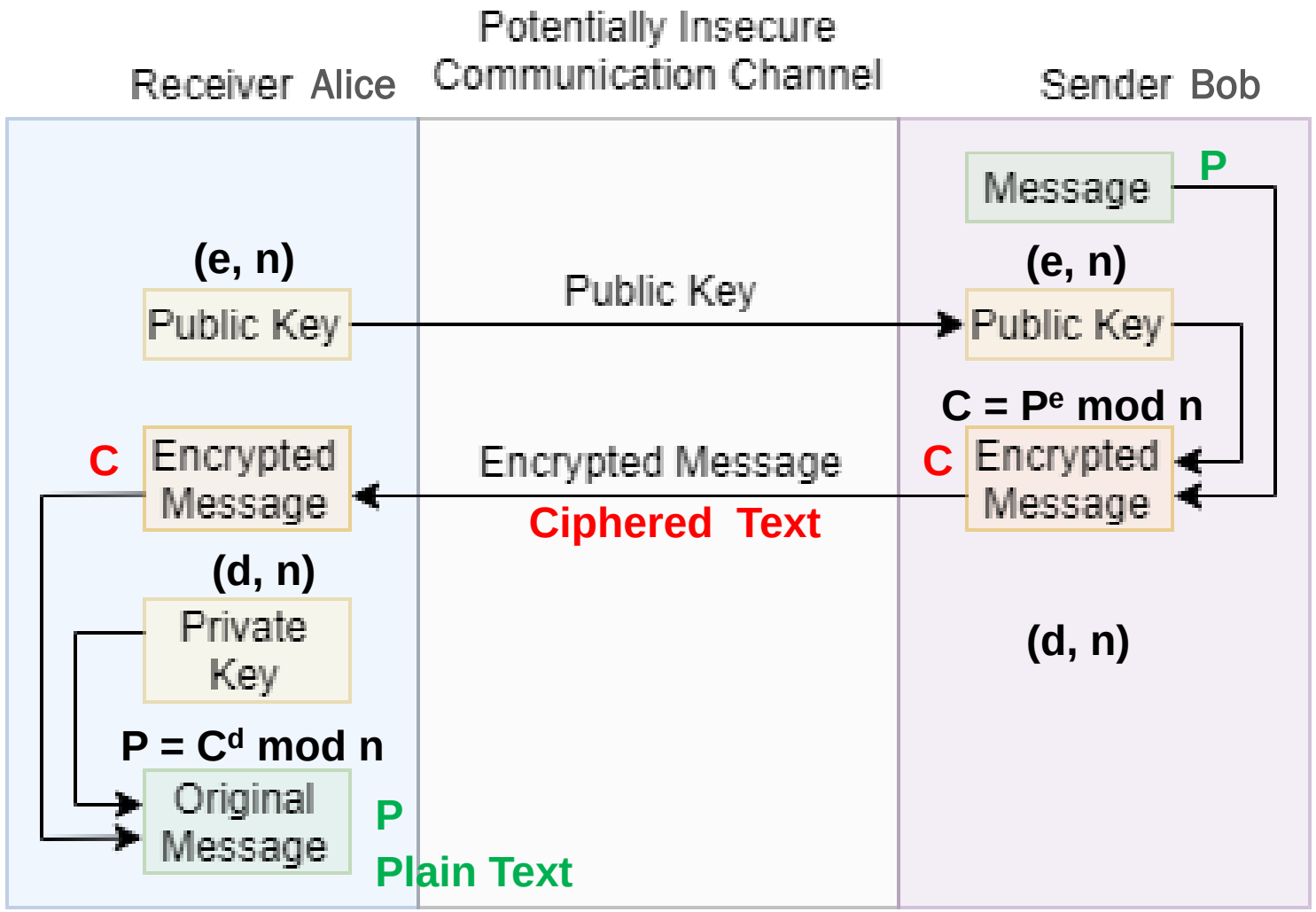
- **הצפנה סימטרית**
  - **DES (לא בשימוש יותר)**
  - **AES = Advanced Encryption Standard בשימוש נרחב כיום ונחשב ללא פריץ**

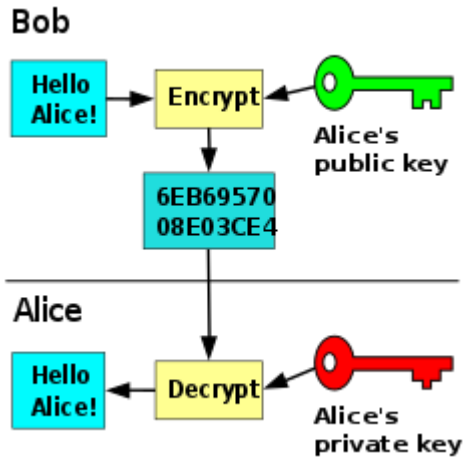
- **הצפנה אסימטרית:**
  - Rivest–Shamir–Adleman - **RSA** (כיום בשימוש פוחת עקב גילוי חולשה ופריצה אליו)
  - **ECC** – Elliptic Curve Cryptography
  - **ECRSA –** Elliptic Curve RSA
  - **DHE –** Deffie Hellman
  - **ECDHE –** Elliptic Curve Deffie Hellman

איפה אפשר לראות במה משתמשים בכל אתר: בדפדפן F12 לשונית Security כאשר נמצאים באתר מוצפן.

סרטון הסבר קצר על עקרונות הצפנה סימטרית ואסימטרית

תהליך הצפנה של אלגוריתם אסימטרי RSA





נגדיר:

Encryption with:
PublicKey = (e,n)

Decryption with:
PrivateKey = (d,n)

עקרונות RSA

- אלגוריתם RSA מבוסס על מספרים גדולים n, p, q, e, d שמקשים מאוד על מציאתם גם עם מחשבים חזקים.
- מכיוון שכדי לפענח הודעה M מוצפנת, צריך לחשב $(M^e)^d$ וקשה למצוא את d אפילו אם יודעים מהם n = p*q ו- e
- האלגוריתם ניזון מהעובדה המתמטית הבאה שמתקיים תמיד ש :
- $(M^e)^d \equiv M \mod n$ $\qquad ((M^e)^d = n * k + M)$
- and $(M^d)^e \equiv M \mod n$
- e, d הם מפתחות פרטי וציבורי כל מה שמוצפן עם מפתח e (ציבורי) ניתן לפתיחה עם מפתח d (פרטי) **ולהיפך**

חישוב מפתחות באלגוריתם RSA צעד אחרי צעד

- שלב 1 : בחר שני מספרים ראשוניים <u>גדולים</u> p ו- q באורכים שווים כך ש- $p \neq q$
- שלב 2 : חשב את $n = p * q$
- שלב 3 : חשב את בפונקציה פאי $\Phi(n) = (p-1)*(q-1)$ אילו מספר המספרים הראשוניים הקטנים מ n (coprime number to n)
- שלב 4 : בחר מספר e המקיים את התנאים הבאים : $1 < e < \Phi(n)$ וגם

  $GCD(e, \Phi(n)) = 1$ וגם $GCD(e, n) = 1$ e is a coPrime to n and $\Phi(n)$,

1. P = 13 , q = 11
2. N = 13 * 11 = 143
3. $\Phi(n) = (13-1) * (11-1) = 120$
4. Select e = 17 , GCD(17, 120) = 1 && GCD(17,143)=1 , **e** is the public key to this algorithm

חישוב מפתחות באלגוריתם RSA צעד אחרי צעד

- שלב 5: חשב את d כך שיתקיים : $e * d = 1 \bmod \Phi(n)$

  או בכתיבה אחרת : $e * d = k * \Phi(n) + 1$ (אריתמטיקה מודולרית)

Finding d: (in general **$a \equiv b \pmod{n}$ => $a = k * n + b$, k is integer**)

5. $e * d \equiv 1 \bmod \Phi(n)$, $e = 17$ rewriting **$e * d = \Phi(n) * i + 1$**
   - $17 * d \bmod 120 = 1$
   - $d = ((\Phi(n) * i)+1) / e$ should be whole number (לקחת לאו דווקא את הראשון)
     - $(120 * 1 + 1) / 17 = 7.11$
     - $(120 * 2 + 1) / 17 = 14.17$
     - …
     - $(120 * 33 + 1) / 17 = 233 \Rightarrow d = 233$

חישוב מפתחות באלגוריתם RSA צעד אחרי צעד

- שלב 6: נגדיר את המפתח ציבורי = (e,n) ואת המפתח פרטי = (d,n)
- שלב 7: הצפנה באופן הבא:
- $C = P^e \bmod n$, where $P<n$

  C = Cipher Text, P = Plain Text, e = Encryption key , n = block size
- שלב 8 : פיענוח בצד השני ע"י $P = C^d \bmod n$

- Result:

6. PublicKey = (17,143), PrivateKey=(233,143)
7. Encryption: **P='X'** -> $C = P^e \bmod n = 88^{17} \bmod 143$ -> **C = 121** **את זה שולחים לצד השני**
8. Decryption: $P = C^d \bmod n = 121^{233} \bmod 143 = 88$ -> **P = 88 = 'X'**

https://umaranis.com/rsa_calculator_demo.html

הצגה אחרת של התהליך ההצפנה והפיענוח ב RSA

בוב

Alice Public key = (e, n) = (17, 143)
Alice Private Key = (d, n) = (233, 147)

שלב א' -

נניח ש**בוב** רוצה לשלוח לאליס הודעה המכילה אות אחת בלבד X.

לפי קוד ASCII מתקבל המספר 88 העשרוני, ולכן ההודעה M היא המספר 88.

שלב ב' - הצפנת ההודעה

כדי להצפין את ההודעה בוב מחפש את המפתח הציבורי של אליס ומגלה שהוא:N=143 ו-e=17.

ולכן הוא מצפין את ההודעה ומקבל: $C=88^{17}(\text{mod } 143)$

החישוב נותן C=121 . וזו היא ההודעה המוצפנת שבוב מעביר לאליס.

הצגה אחרת של התהליך ההצפנה והפיענוח ב RSA

Alice Public key = (e, n) = (17, 143)
Alice Private Key = (d, n) = (233, 147)
C = 121 p = 13 q = 11

אליס

אבל לאליס יש את המפתח הפרטי שלה. והיא היחידה שיודעת את ערכי p ו- q ולכן יכולה לחשב את ערכו של מספר מיוחד d שהוא המפתח הפרטי שלה.

$e * d \equiv 1 \pmod{(p-1)*(q-1)}$

$17 * d \equiv 1 \pmod{12*10}$

$17 * d \equiv 1 \pmod{120}$ => $(120 * i) + 1 / 17$, $i = 33$ => $d = 233$

$d=233$

חישוב d אינו פשוט אבל משתמשים באלגוריתם של אוקלידס.

בשלב האחרון נחשב את M :

$M=C^d \pmod{147}$

$M=121^{233} \pmod{147}$

$M=88$

וזה שווה X לפי קוד ASCII.

"חוזק" פרוטוקול RSA

- יכולת הפיצוח של הקוד תלויה בקושי למצו את p ו-q מתוך n , ומציאת e מתוך Φ(n) וחישוב d מ- (e, Φ(n)) . כזכור n = p*q ו- p, q הינם מספרים ראשוניים גדולים מאוד.
- בהיסטוריה פוצחו הקודים עבור אורך n של:
  - 700 bit ב 2007 , מספר עם 210 ספרות (700 * log2)
  - 768 bit ב 2009, מספר עם 230 ספרות
- כיום משתמשים ב 2048 bit מינימום -> שזה מספר עם 617 ספרות...
- מהירות הפענוח של RSA גדלה ביחס של פי 4 לאורך המפתח ונחשבת **איטית בסדרי גודל יחסית ל AES, ולכן לא משתמשים בה להצפנה של אינפורמציה רבה**
- חיסרון גדול ב-RSA: הודעות זהות מוצפנות זהה – לכן מאפשר פיצוח בעזרת אנליזת חזרות
- RSA מאפשר אימות זהות (נראה גם איך בהמשך)

שבירת RSA בעזרת מחשב קוונטי

- **מחשב קוונטי** הוא מכונה המעבדת נתונים תוך שימוש ישיר בתכונות של מכניקת הקוונטים כגון סופרפוזיציה ושזירה קוונטית.
- מחשב קוונטי שונה ממחשב רגיל, הוא משתמש בקיוביט (ביט קוונטי) במקום ביט כיחידת המידע הבסיסית, והפעולות הבסיסיות שניתן לבצע על קיוביטים שונות מהשערים הלוגיים העומדים בבסיסו של מחשב קלאסי.
- ישנן בעיות שמחשב קוונטי מסוגל לפתור **ביעילות גבוהה** יותר מאשר האלגוריתם המיטבי האפשרי עבור מחשב קלאסי רגיל.

שבירת RSA בעזרת מחשב קוונטי

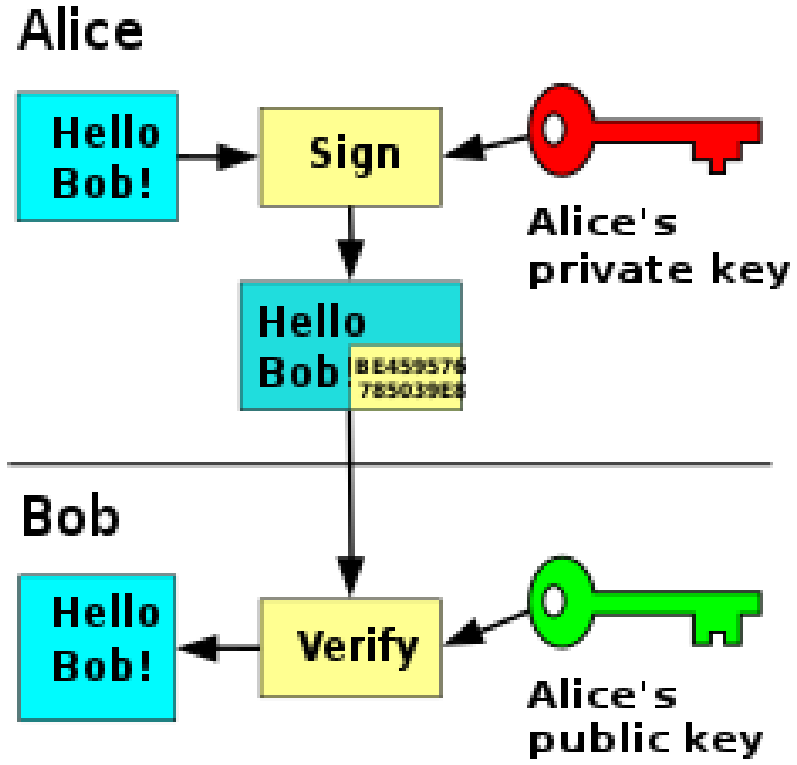
- בשנת 1994 פיתח מדען המחשב פיטר שור את **אלגוריתם שור** לפירוק לגורמים באמצעות חישוב קוונטי שיש לו סיבוכיות ריצה טובה בהרבה מכל האלגוריתמים ה"רגילים" הידועים כיום.
- בעזרתו ניתן לגלות מה- n שבמפתח הפומבי את ה- p וה- q (ע"י פירוק לגורמים), ואז ניתן לחשב בקלות את $\phi(n) = (p-1)(q-1)$ ואז כבר ניתן לקבל את המפתח הפרטי d מתוך המפתח הפומבי e (צריך פשוט להפוך את e מודולו $\phi(n)$).
- **כל זה נכון לגדלים מסוימים של n נכון להיום עם השינויים שהונהגו בפרוטוקול אין אפשרות גם עם מחשב קוונטי לפצח**

חתימה דיגיטלי ואימות זהות

- RSA מספק : אימות זהות, ואמינות המידע (אמינות: שמה שהשולח שלח אכן הגיע ולא שום דבר אחר, אימות וידוי מי השולח)
- התהליך:
  - h = H(M) , h is hash signature of M
  - $(h^d)^e = (h^e)^d = h \bmod n$

דוגמת בוב ואליס:

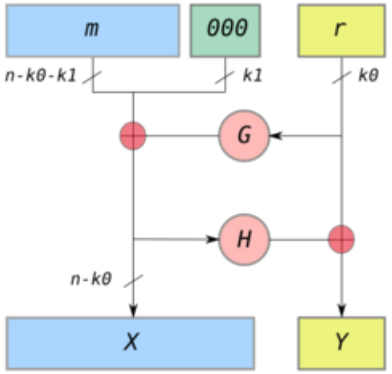
- אליס שולחת הודעה לא מוצפנת לבוב וחותמת את ההודעה עם המפתח הפרטי שלה d
- לכל מי שיש את המפתח **הציבורי** e של אליס יכול לפענח את החתימה של אליס
- **הודעה שניתן לפענח באמצעות המפתח הציבורי של אליס הייתה חייבת להישלח מאליס!**
- אמינות נובעת מצירוף חתימה למידע ובדיקתו מול חתימת המידע שהגיע.



RSA כיום

- בעיקר משמש בהצפנה משולבות יחד עם AES , כאשר RSA משמש להצפנת המפתח המשותף של הצפנת AES ו- AES משמש להצפנה המידע עצמו
- כדי להתגבר על הבעיה שהודעות זהות מקודדות זהה משתמשים באלגוריתם RSA משופר המוגדר ב PKCS#1 ומגדיר את Optimal Asymmetric = OAEP Encryption Padding המוסיף אינפורמציה אקראית (padding) למפתחות באופן הבא:

- To encrypt M choose random number r and encode M as:
  - M’ = (X=M ⊕ G(r) , Y = r ⊕ H(X)) , H_1, H_2 are Hash function s
  - Encryption: done with **$(M')^e$ mod n**
  - **Decryption: given M’ = (X,Y) ,** r = Y ⊕ H_2(X), and **M = X ⊕ H_1(r)**



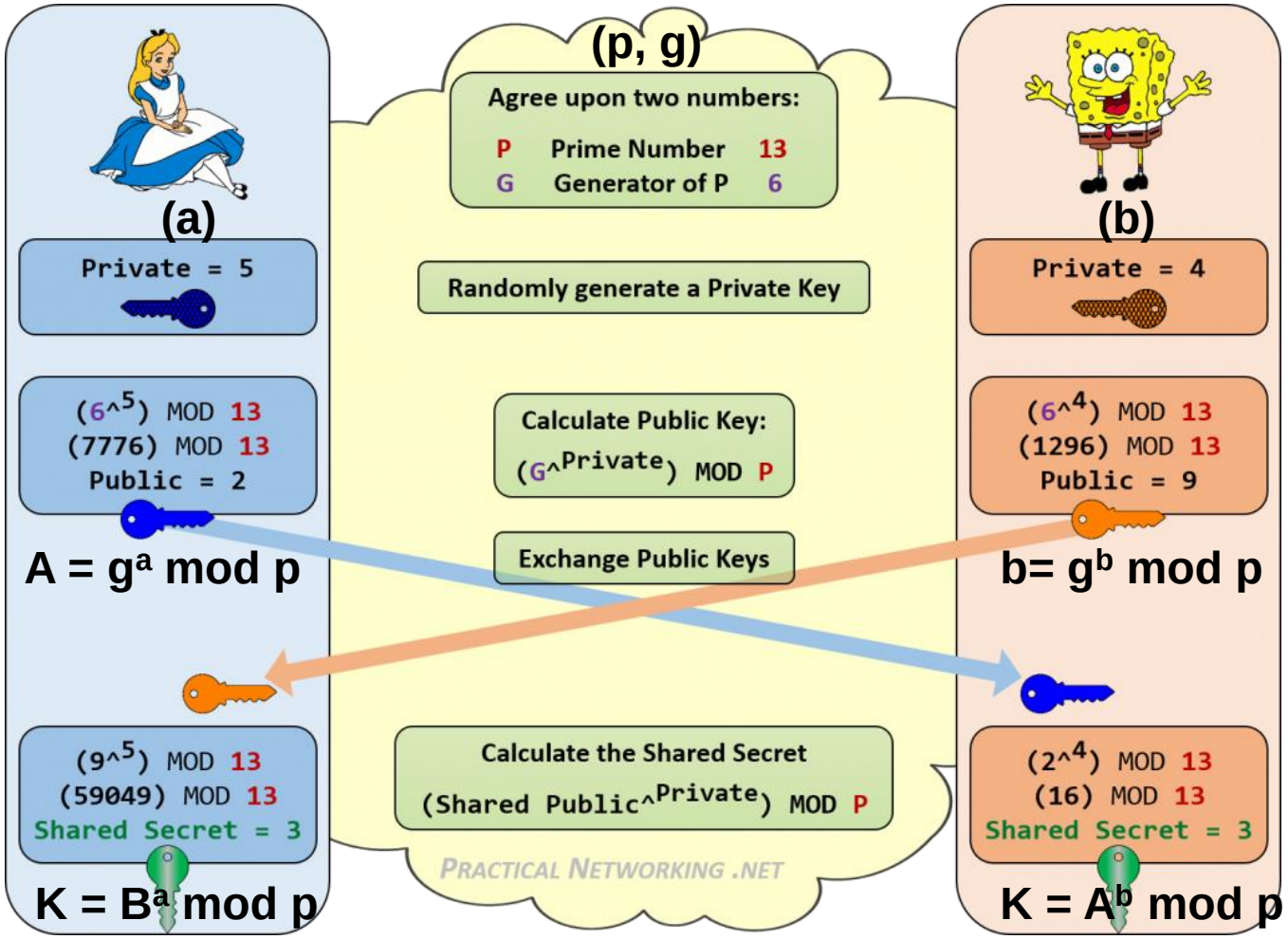
דוגמת קוד, יצירת מפתחות והצפנת RSA : RSA.py

שילוב בין הצפנה סימטרית לא-סימטרית

- אליס רוצה לתקשר עם בוב באמצעות מפתח סודי משותף (סימטרי).
- היא יוצרת את המפתח.
- היא שולחת את המפתח באמצעות השיטה הא-סימטרית (מצפינה עם המפתח הציבורי של בוב ובוב מפענח עם המפתח הפרטי שלו).
- כעת אליס ובוב יכולים לתקשר עם המפתח המשותף **שלהם** באמצעות הצפנה סימטרית כי אין סכנה שמישהו גילה את המפתח מלבדם.
- **היתרון של שיטה זו הוא חיסכון בזמן כי הצפנת טקסט שלם ופענוחו בשיטה הא-סימטרית לוקח יותר זמן מאשר השיטה הסימטרית.**

Diffie-Helman Key Exchange
Asymmetric Encryption

אלגוריתם Diffie-Hellman



מוטיבציה:

- **פרוטוקול מוצפן כך ששני הצדדים יסכימו על מפתח הצפנה משותף**

חולשה עיקרית :

- **אי יכולת לבצע אימות של זהות הצד השני ובכך חושפת את התקשרות להתקפת MitM (Man-in-the-Middle)**

$$K=A^b \bmod p = (g^a \bmod p)^b = g^{ab} \bmod p = (g^b \bmod p)^a = B^a \bmod p$$

אלגוריתם מהיר למציאת מודולו של מספרים גדולים

```
static int powerMod(int bas, int exp, int mod)
{
        if (mod == 1) return 0;
        var result = 1;
        bas = bas % mod;
       while (exp > 0)
      {
           if (exp % 2 == 1)   //odd number
                result = (result * bas) % mod;
         exp = exp / 2;          //divide by 2
           bas = (bas * bas) % mod;
       }
      return result;
}
```

https://umaranis.com/rsa_calculator_demo.html

- PKI library references:
  1. https://www.cryptosys.net/pki/python-pki-xref.html
  2. https://www.cryptosys.net/pki/python.html
- PKI lib = cryptosyspki
  - Pip install cryptosyspki (or update with pip install –upgrade cryptosyspki