טכנו"צ סייבר – תרגיל 2 – "Encryption Engine"

טעינת פקודות מוצפנות

מטרת העל

בתרגיל הקודם (1) בנינו נוזקה בסיסית – קוד פייתון ש"מתלבש" על המשחק Chicken Invaders ומאפשר לנו לבצע פעולות זדוניות.

בתרגילים הבאים (3-4) נרצה לבנות "פיקוד מרוחק" ששולט בנוזקה (או נוזקות, אם פגענו בכמה מחשבים) ממחשב אחר. לא נרצה שהפקודות הזדוניות שלנו יתגלו, לכן בתרגיל הזה נכין תשתית הצפנת מידע עבור מערכת תקשורת בין הפיקוד לבין הנוזקות.

השיטה

מערכת ההצפנה תעבוד באופן הבא:

1) לפיקוד המרוחק (שישלוט בנוזקה בהמשך) יהיו שני מפתחות public, private, שישמשו להצפנת RSA אסימטרית.
2) הנוזקה שלנו תחזיק בעצמה במפתח הפומבי (public), שיהיה שמור באחד הקבועים שלה.
3) בכל פעם שאחד מהצדדים ירצה לשלוח הודעה, עליו:
   a) להגריל מספר בין 1 לבין MAX_CBC_KEY (קבוע נתון), שישמש כמפתח הצפנה סימטרי בשיטת CBC שלמדתם בהרצאות.
   b) להצפין את המפתח הסימטרי באמצעות המפתח הפומבי (עבור הנוזקה) או הפרטי (עבור הפיקוד). **הערה חשובה** – אנחנו מניחים שאי אפשר לגנוב את המפתח הציבורי ומתייחסים אליו כ"סודי" גם כן.
   c) לבצע תהליך "עטיפה" (padding) להודעה: הצפנת ה CBC תעבוד רק אם אורך המידע שלנו יתחלק באורך הבלוק, לכן נוסיף ל data בהתחלה את התו "~" כמה שצריך כדי להשלים את האורך של המידע למספר המתחלק בגודל בלוק (נתון

טכנו"צ סייבר – תרגיל 2 – "Encryption Engine"

בתור הקבוע BLOCK_SIZE). נניח הנחה מקלה שהמידע עצמו בחיים לא יתחיל ב"~" כך שלא נתבלבל בין padding לבין מידע אמיתי.

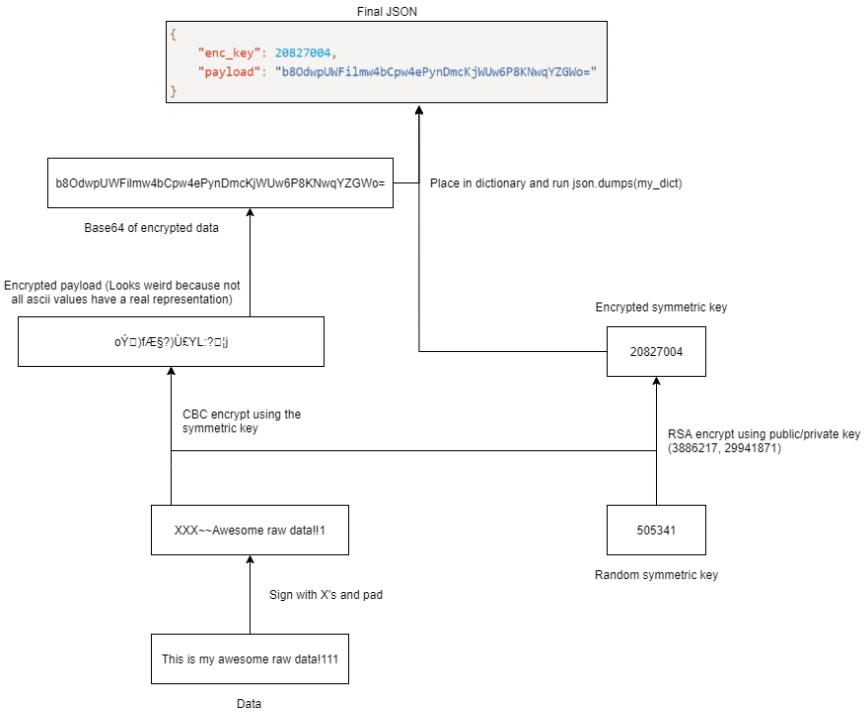
d) לחתום את ההודעה: נוסיף בלוק שלם של התו "X" בתחילת המידע (לפני הpadding של ה"~"). בהמשך נשתמש בבלוק זה כדי לוודא שפענוח ההצפנה עבד.

e) להצפין את ההודעה באמצעות המפתח הסימטרי.

f) לעטוף את הכל במילון (dictionary), שיכיל את המפתח הסימטרי המוצפן ("enc_key") ואת ההודעה המוצפנת ("payload"). המידע עלול להכיל תווים לא חוקיים בטבלת ASCII לאחר ההצפנה (למשל למספר 7 אין באמת ייצוג של תו ב ASCII), לכן ראשוית נמיר את payload לייצוג של Base64 (שבו כל התווים האפשריים הם בעלי ייצוג טקסטואלי "יפה"). את המילון הסופי נמיר לפורמט שנקרא JSON.[1]

g) להחזיר את המילון העטוף (זהו "פיקוד").

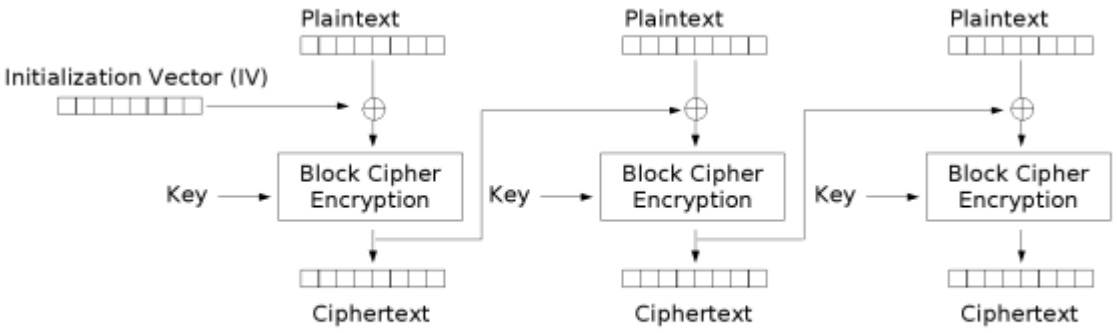
המחשה של שיטת ההצפנה ניתן לראות בתרשים הבא (במקרה זה BLOCK_SIZE = 3):



ולמי ששכח איך CBC נראה, ההצפנה עובדת כך:

[1] בהמשך נלמד שצריך להמיר את המידע שלנו לstring/בינארי על מנת שנוכל לשלוח אותו ברשת, JSON זה מנגנון המאפשר להמיר מילונים ומערכים לstring שכזה, וכן לבצע המרה בחזרה.

טכנו"צ סייבר - תרגיל 2 - "Encryption Engine"



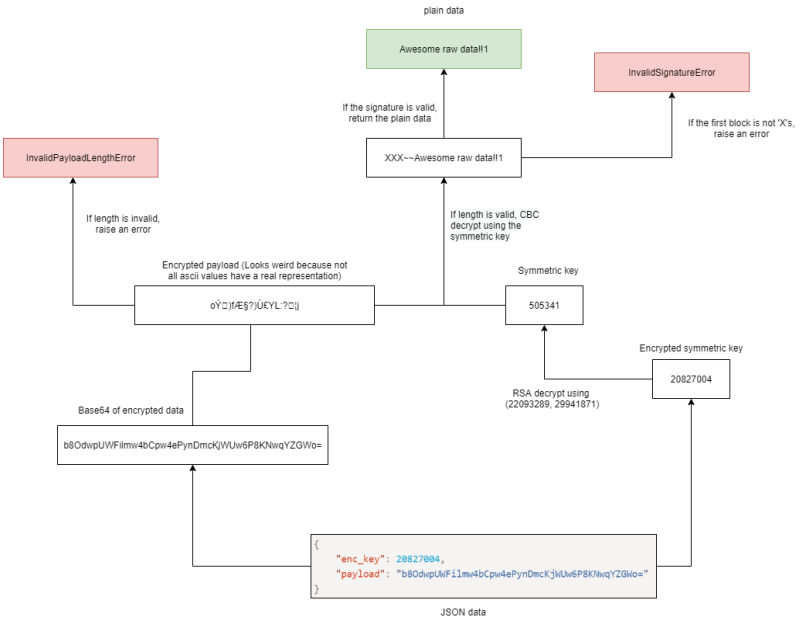
Cipher Block Chaining (CBC) mode encryption

(הBlock Cipher Encryption יהיה פשוט XOR של הבלוק עם המפתח).

4) בכל פעם שאחד מהצדדים ירצה לפענח הודעה, עליו:
   a) להמיר את ה JSON בחזרה למילון (באמצעות הפונקציה json.loads).
   b) לקרוא מהמילון את המפתח הסימטרי המוצפן, ולפענח אותו באמצעות המפתח הפרטי (עבור הנוזקה).
   c) להמיר את ה payload בחזרה מ- Base64.
   d) לוודא את תקינות ה payload (אורכו מתחלק בBLOCK_SIZE). אם ההודעה לא תקינה, יש לזרוק שגיאת InvalidPayloadLengthError (נמצאת ב ex2_utils).
   e) להשתמש במפתח הסימטרי המפוענח כדי לפענח את ההודעה המוצפנת.
   f) לוודא את תקינות החתימה (מתחילה בבלוק של "X"). אם החתימה אינה תקינה, יש לזרוק שגיאת InvalidSignatureError (נמצאת ב ex2_utils).
   g) להחזיר את תוכן ההודעה ללא ה X וללא padding.

וכמובן, לא נשאיר אתכם בלי תרשים שימחיש את כל העסק (משלים את התרשים הקודם):

טכנו"צ סייבר - תרגיל 2 - "Encryption Engine"



חומרי עזר

בתרגיל מסופקים לכם מספר חומרי עזר. נמנה אותם כאן:

1) קובץ שלד לקוד שלכם ex2_skeleton.py, המכיל מימושים ריקים והסברים לכל הפונקציות שעליכם לממש.

2) קובץ פונקציות עזר ששמו ex2_utils.py. הקובץ מכיל הרבה פונקציות וקבועים שימושיים (הimports לדברים הרלוונטים נמצאים כבר בex2_skeleton.py).

3) **הערה**: עליכם להתקין את הספריה pycryptodome על מנת שחומרי העזר יעבדו.

שלבי התרגיל

1) עליכם לממש את הפונקציה generate_key_pair. פונקציה זו אינה מקבלת אף פרמטר, ויוצרת שני מפתחות הצפנה (פומבי ופרטי) של אלגוריתם RSA. המפתח הפומבי יהיה הטאפל (e, n), והמפתח הפרטי יהיה הטאפל (d, n).

לשם כך אתם מוזמנים להשתמש בשלושת פונקציות העזר:

a) פונקציה שמייצרת מספר ראשוני גדול get_random_prime. שימו לב שהמספרים p ו q צריכים להיות **שונים**.
b) פונקציה שבודקת האם שני מספרים זרים is_coprime.
c) פונקציה שמחשבת הופכי-כפלי של מספר תחת מודולו מסויים get_multiplicative_inverse.

טכנו"צ סייבר – תרגיל 2 – "Encryption Engine"

2) עליכם לממש את הפונקציה use_key. הפונקציה מקבלת מספר ומפתח, ומצפינה/מפענחת את המידע באמצעות המפתח באלגוריתם RSA. אנחנו עובדים כאן עם מספרים גדולים, מאוד גדולים. על כן, שימו לב **לשתי נקודות משמעותיות**:

a) לא נוכל לחשב חזקה בכלים סטנדרטים, כי אחרת נגלוש מגבולות ה integer (המספרים עצומים!). נחשוב ביחד על דרכים אלטרנטיביות לחשב חזקה שכזו:

i) הפתרון הפשוט (**שלא נממש בתרגיל**) יהיה לחשב חזקה בלולאה של כפל, כשבכל פעם נבצע מודולו n. זה עובד כיוון ש:

$$(x * y) \bmod n = (x \bmod n) * (y \bmod n)$$

ii) הפתרון הפשוט לצערנו לא יתכנס חישובית למספרים כמו $78213^{33242059}$. לכן **בתרגיל נממש** דרך משוכללת יותר לחשב את $m^e \bmod n$.

מתמטיקאים, החלק הבא הוא במיוחד בשבילכם:

(1) נמצא את הייצוג הבינארי של e (מוזמנים לקרוא על bin בפייתון). נניח שאורכו A ביטים.

(2) נחשב את:

$$m \bmod n,\ m^2 \bmod n,\ m^4 \bmod n,\ m^8 \bmod n \ldots\ m^{2^{A-1}} \bmod n$$

באופן איטרטיבי: בכל פעם נכפול את האיבר הקודם בעצמו ונבצע מודולו. נחזור על תהליך זה A פעמים, כלומר נרצה רשימה של A חזקות הב2 הראשונות (כולל $2^0 = 1$) של m תחת מודולו n.

(3) כעת נרוץ על הייצוג הבינארי של e, ונשמור משתנה צובר (כפלי) שמאותחל ל1. בכל פעם שניתקל ב"1" בביט הi של הייצוג הבינארי, נכפול את הצובר בחזקה הi מהרשימה (כלומר m^{2^i}) ונבצע מודולו n. לבסוף נחזיר את הצובר.

(4) למה זה עובד? ניקח דוגמה: $3^5 \bmod 7$. ביטוי זה שווה ל $(3^4 \bmod 7 * 3^1 \bmod 7) \bmod 7$ שזה מה שאנחנו מחשבים עם הצובר הכפלי. כדי להכליל לכל שני מספרים נרצה פשוט לפרק את החזקה לייצוג בינארי, ולכפול איברים מהקבוצה $\{m^{2^i} \mid i \in N,\ i \le ceil(log_2 e)\}$ זה בזה בהתאם לייצוג (תוך הפעלת מודולו בכל פעם).

(5) למה זה מהיר? לחשב את כל החזקות עולה לנו $log_2 e$ (אורך הייצוג הבינארי של e). פעולות כפל של מספרים שגודלם חסום ב n. להשיג את הצובר הכפלי עולה לנו גם כן $log_2 e$ פעולות. זאת אל מול הפתרון הקודם שעבד ב e פעולות כפל של מספרים שגודלם חסום ב n.

(6) מתמטיקאים, מקווים שנהנתם מהאלגוריתם האחרון, כי כותב התרגיל פחות.

טכנו"צ סייבר - תרגיל 2 - "Encryption Engine"

הערה: ניתן לבצע את האלגוריתם הזה ללא שימוש בפונקציה bin ובלי שום מעבר לייצוג טקסטואלי של המספר e אלא רק באמצעות אופרטורים בינארים. מוזמנים לממש את האלגוריתם גם בצורה זו.

b) כפל של שני מספרים בודדים גם הוא עלול לגלוש מגבולות ה integer של פייתון (שהוא 32 ביטים). בשביל להתגבר על זה, לפני הכפל נמיר את שני המספרים לסוג np.int64, ולאחר המודולו נחזיר אותם להיות int (שכן אחרי המודולו נקבל מספר שקטן מn).

3) עליכם לממש שתי פונקציות - cbc_encrypt וcbc_decrypt, המקבלות מידע (נקי או מוצפן בהתאמה) ומתפתח, ומצפינות/מפענחות אותו בשיטת CBC. נזכיר שהצפנת הבלוק מבחינתנו היא XOR עם המפתח.

4) עליכם לממש את הפונקציה encrypt_data שתקבל מידע ומפתח אסימטרי ותצפין את המידע באלגוריתם שהצגנו. את החלק של הJSON והBase64 מימשנו בשבילכם והוא מופיע בשלד.

5) עליכם לממש את הפונקציה decrypt_data שתקבל מידע מוצפן ומפתח אסימטרי ותפענח את המידע באלגוריתם שהצגנו (כל עוד הוא תקין וחתום היטב, אם לא תזרקו את השגיאה הרלוונטית). את החלק של הJSON והBase64 מימשנו בשבילכם והוא מופיע בשלד.

שאלה Q1

עליכם לענות על שאלה זו ולצרף את התשובה לתרגיל.

למעשה, משהו במימוש מנגנון האבטחה שלנו גורם לכך שההצפנה הסימטרית שלנו היא אמנם מסוג CBC, אבל בקלות אפשר להפוך אותה להצפנה מסוג EBC, מבלי לדעת מה המפתח. מצאו את הבעיה והסבירו כיצד תוקף יכול להשתמש בה כדי "לקלף" את אלמנט הCBC.

שאלה Q2

עליכם לענות על שאלה זו ולצרף את התשובה לתרגיל.

חשבו על בעיות אבטחתיות נוספת במנגנון ההצפנה שבנינו. מנו לפחות בעיה אחת **משמעותית**.
שאלות מכווינות: מה יקרה אם תהיה יותר מנוזקה אחת בעולם? מה יקרה אם נוזקה תתגלה?

בונוס מעשי

חשבו כיצד ניתן לשפר את אלגוריתם הצפנה סימטרי שהשתמשנו בו. פרטו כיצד השיפור/ים מטפלים בחלק מהבעיות מהסעיפים הקודמים או בכולם. ממשו את האלגוריתם בפונקציות נפרדות בשם advance_encrypt ו advance_decrypt כאשר החתימות של הפונקציות יהיו זהות למקבילות ה-cbc שלהן. פרטו על התהליך בקובץ ה README.

טכנו"צ סייבר - תרגיל 2 - "Encryption Engine"

טיפים לתרגיל

1) **תתחילו אותו מוקדם**. אמנם מדובר בב 150 שורות קוד שעליכם לכתוב, זה לא תרגיל פשוט בכלל, והוא לא טריוויאלי.
2) **בדיקות**: מסופקים לכם כמה שימושים בסיסיים ב main של ה skeleton, אבל תמשיכו לשחק עם זה - תוודאו שאתם מגיעים לכל מקרי הקצה המעניינים (שגיאות למיניהן וכו').
בנוסף, מומלץ לבדוק את הקוד שלכם עבור BLOCK_SIZE שאינו 3. **תשנו רק את הקבוע PRIME_NUMBER_BITS**, באופן הבא:
   a) כדי לבדוק את BLOCK_SIZE = 3 (דיפולט), תשתמשו ב 13 עבור PRIME_NUMBER_BITS
   b) כדי לבדוק את BLOCK_SIZE = 2, תשתמשו ב 9 עבור PRIME_NUMBER_BITS
   c) כדי לבדוק את BLOCK_SIZE = 1, תשתמשו ב 5 עבור PRIME_NUMBER_BITS.
3) **תרגישו בנוח להיעזר בסגל הקורס** לכל דבר ועניין בתרגיל - הוא גם לא פשוט, וזו גם השנה השנייה שהוא קיים :)

הגשה

עליכם להגיש את הקוד שלכם, בשם ex2_sol.py, עטוף ב**ex2_[FullName].zip**. אין לצרף את קובץ העזר. בנוסף, עליכם להגיש קובץ README שבו תסבירו את התשובה לשאלות Q1 ו Q2 ולבונוס המעשי במידה ועשיתם אותו.
בהצלחה!