Technologic Papers

המתמטיקה שמאחורי פרצות במנגנון WEP

מאת יהלי הכט ומתן ונגוש

הקדמה

ברב המקרים, כאשר נפרוש רשת, לא נרצה שלכל אחד תהיה אליה הגישה. כאשר החיבור הוא קווי, יהיה לנו קל להיות בטוחים שאף אחד לא יתחבר לרשת ללא רשות, משום שהוא צריך גישה פיזית לחומרה עצמה. אך כאשר מדובר ברשת Wireless העניין הופך להיות מעט מורכב יותר.

בדיוק לטובת כך הומצא אלגוריתם האבטחה WEP (קיצור של Wired Equivalent Privacy) שהחל להיכנס לשימוש בשנת 1999.

אלגוריתם זה הוצג כבר בעבר במגזין זה, מאמר מנגנון הצפנה WEP, אך המאמר שלנו מרחיב את הנושא, בדגש על החולשות והמתקפות נגד WEP, בניגוד למאמר שהוזכר, שבו מוצגות האפשרויות השונות לאבטחת הרשת, מאמר זה יתמקד בלוגיקה ובאלגוריתמים המתמטיים ולא רק במה שמתאפשר לבצע בעקבותם.

כאמור, WEP התיימר לאבטח את הרשת ברמה שקולה לחיבור קווי, הדורש גישה פיזית ומבטיח פרטיות. למרות זאת, כבר בשנת 2001 הצליחו לפצח את האבטחה של האלגוריתם, החוקרים Fluhrer, Mantin and Shamir הכינו את ההתקפה FMS שתפורט בהרחבה בהמשך. לכן פותח אלגוריתם ה- Wi-Fi WPA (Protected Access). למרות שלאלגוריתם זה אבטחה יותר חזקה, גם הוא חשוף לפריצות. אחריו הגיעו גם WPA2 ו-WPA3 שהוכרז ב-2018 וצפוי להחליף את WPA2 בעוד כמה שנים.

בהמשך, מסופר בקצרה על IEEE 802.11, תקן לרשתות מקומיות אלחוטיות, נתמקד באלגוריתם RC4, שבאמצעותו מוצפן המידע ב-WEP, בדרך פעולתו של האלגוריתם, פרצות אבטחה שהתגלו בו ודרכים לפרוץ רשת מבוססת WEP.

IEEE 802.11

IEEE 802.11 הוא סטנדרט לרשתות אלחוטיות מקומיות, שנמצא בשכבת הקו במודל חמשת השכבות. לסטנדרט זה יש מספר גרסאות, אך אנחנו נציג את IEEE 802.11b ששוחררה בשנת 1999 ותומכת במנגנון WEP, אך נמצאת בשימוש מועט. לקריאה נוספת על גרסאות נפוצות:

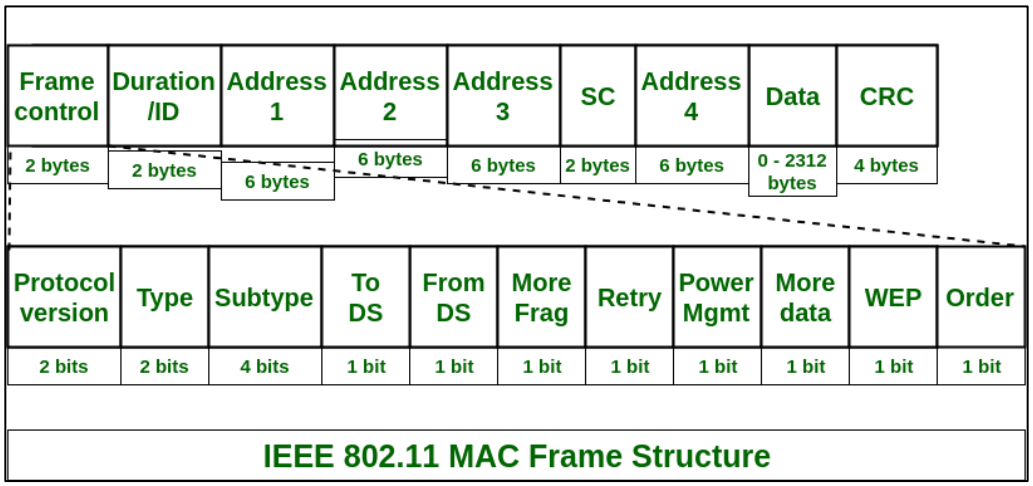
The Most Common Wi-Fi Standards and Types Explained

IEEE Standard	802.11a	802.11b	802.11g	802.11n	802.11ac	802.11ax
Year Released	1999	1999	2003	2009	2014	2019
Frequency	5Ghz	2.4GHz	2.4GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz
Maximum Data Rate	54Mbps	11Mbps	54Mbps	600Mbps	1.3Gbps	10-12Gbps

[מקור: The Most Common Wi-Fi Standards and Types Explained]

בתמונה מופיעים התדרים שבהם משתמש כל תקן, המהירות גלישה מרבית שמתאפשרת והשנה שבה כל תקן נכנס לשימוש

כדי להבין יותר טוב מה ה-IEEE 802.11 מספק לנו, נסתכל על שדות ב-Header שלו והמבנה של ה-Frame.



[מקור: IEEE 802.11 Mac Frame - GeeksforGeeks]

- Frame Control - מטרתו לקבוע את סוג ה-Frame ומידע דרוש. במאמר לא יפורטו הסעיפים ב-Frame Control, למעט הסעיף WEP ולכן מומלץ לקרוא עליהם בקישור למעלה. הסעיף WEP הוא בגודל סיבית אחת - 0 או 1. וקובע האם יש שימוש במנגנון.
- Duration - משפיע על סוג ה-Frame שמופיע בשדה הקודם (לקריאה נוספת)

- Source Address (SA) - כתובת ה-MAC של השולח הראשוני
- Destination Address (DA) - כתובת ה-MAC של היעד הסופי
- Transmitter Address (TA) - כתובת ה-MAC של השולח האחרון, מי ששידר את המידע.
- Receiver Address (RA0 - כתובת ה-MAC של מי שאליו משדרים את המידע
  - לא בהכרח היעד (DA) אלא הישות הבאה במסלול
- SC (Sequence Control) - כמו ה-Frame Control, השדה מחולק:
  - Sequence number (12 bits)
  - Fragment number (4 bits).
  - השדה נועד לשמור על הסדר של הפקטות, כדי שהמידע לא יתבלגן בדרך (כי אם Frame אחד יגיע לפני Frame אחר שמופיע לפניו בסדר המידע, צריך לדאוג שהמידע לא ישורשר לפי זמן ההגעה אלא לפי הסדר הראשוני).
- Data - כולל את המידע שהועבר ב-Frame ואת ה-Headers של השכבות הגבוהות יותר.
- CRC - על ה-CRC Checksum בפרט ועל Checksum בכלל יפורט בהמשך, בהקשר של הצפנה במנגנון WEP.

אלגוריתם ה-RC4

אלגוריתם ה-RC4 הינו צופן זרם, כלומר צופן סימטרי המשמש להצפנת זרמי מידע באורכים משתנים, שפותח בשנת 1987 על ידי מומחה ההצפנה רונלד ריבסט במעבדות RSA, חברה שהקים עם עדי שמיר ולאונרד אדלמן שאיתם פיתח בשנת 1977 את אלגוריתם RSA (מומלץ לקרוא) שאת שמו החברה נושאת. משמעות השם RC4 היא Rivest Cipher 4, אך יש כאלו שמפרשים אותו בתור Ron's Code 4 או Ron's Cipher 4. האלגוריתם הינו אלגוריתם דטרמיניסטי, כלומר אלגוריתם המתנהג בצורה צפויה וניתנת לחיזוי. בהינתן קלט זהה, המכונה עליה רץ האלגוריתם תבצע את אותם צעדים והפלט הסופי יהיה אותו פלט. האלגוריתם אינו נתמך או נמצא בשימוש ברוב מערכות האינטרנט נכון להיום, אך בעבר שימש להצפנה בפרוטוקול SSL 4 (מאוחר יותר הפך אופציונלי ואז הוצא מכלל שימוש כפי שמתואר במסמך RFC 7465 שיצא בפברואר של שנת 2015), הצפנת ה-Seed (עליו נסביר בהמשך) ב-WEP, ועד לשנת 2015 עוד היה בשימוש בדפדפן Internet Explorer ובדפדפן Microsoft Edge.

האלגוריתם מזוהה עם הקלות שבה ניתן ליישם ולהטמיע אותו בתוכנה ובחומרה, אך גם בפרצות האבטחה הרבות שלו שהוציאו אותו משימוש.

לאלגוריתם יש שני חלקים:

1. KSA (Key Scheduling Algorithm)
2. PRGA (Pseudo Random Generation Algorithm)

נסתכל על שלבי הפעולה של האלגוריתם. נתחיל מ-KSA:

כדי שהאלגוריתם יוכל לעבוד, מספק המשתמש מפתח סודי אשר נשמר במערך K שאורכו n. כעת ניצור מערך S שאורכו בדרך כלל 256 אלמנטים, אך לצורך הדוגמה נשתמש במערך שאורכו 10 אלמנטים. את S ניצור באופן פשוט ביותר - באופן עולה. באינדקס 0 יהיה הערך 0, באינדקס 1 יהיה הערך 1 וכן הלאה, עד לאינדקס 9 שערכו 9. כפי שצוין, המערך K הוא מפתח סודי אשר מסופק על ידי המשתמש. הבה נביט במערך S:

Value	0	1	2	3	4	5	6	7	8	9
Index	0	1	2	3	4	5	6	7	8	9

ובמערך K, שהוא מפתח פרטי שהתקבל כקלט מהמשתמש (לצורך הדוגמה נשתמש במפתח רנדומלי):

Value	5	4	7	1	0	6	3	2
Index	0	1	2	3	4	5	6	7

נשים לב כי n = 8 (נזכיר כי n אורכו של K).

נגדיר את הסמן j = 0. כעת נרוץ על המערך S איבר איבר באמצעות הסמן i = 0 שמצביע על האינדקס הנוכחי בכל איטרציה. נבצע את הפעולות הבאות:

נגדיר j = j + S[i] + K[i mod n] ונבצע מודולו עם אורכו של S. במקרה זה נקבל j = 5. נחליף בין ערכי S[i] ו-S[j]. נמשיך עד שנעבור על כל איברי S.

להלן הפונקציה כתובה בשפת python3, כך שתוכלו להריץ אותה על מחשבכם ולצפות בה פועלת (שימו לב כי במקרה זה אורכי S,K יתקבל כקלט מהמשתמש, וכי המערך K ייווצר באופן רנדומלי בידי התוכנה):

```
import random

s = int(input('enter the length of S: '))
k = int(input('enter the length of K: '))
S = list(range(0, s))
K = list(range(0, k))
random.shuffle(K)
print(f'K is {K}')

j = 0
for i in range(0, s):
    j = (j + S[i] + K[i % k]) % s
    S[i], S[j] = S[j], S[i]
print(f'shuffled S is {S}')
```

פלט לדוגמא:

```
enter the length of S: 10
enter the length of K: 8
K is [6, 0, 7, 1, 4, 5, 3, 2]
shuffled S is [3, 7, 0, 2, 8, 4, 9, 5, 6, 1]
```

כעת נסתכל על פעולתו של PGA:

לאחר שאלגוריתם KSA סיים את פעולתו, נשארנו עם מערך S מעורבב באופן פסודו-רנדומלי, כלומר רנדומלי כביכול (מכיוון שמחשב לא מסוגל לייצר רצפים רנדומליים אמיתיים, ניתן לקרוא על כך בלינק לעיל). נסתכל על המערך שיוצא כפלט כשמכניסים את K מהדוגמה הקודמת לאלגוריתם KSA. השורה האמצעית מייצגת את האינדקס, השורה העליונה מייצג את הערך, והשורה התחתונה מציגה את המיקום של שני סמנים i,j:

Value	1	6	9	2	7	3	5	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer	i,j=0									

כפי שניתן לראות, המיקום ההתחלתי של i,j הוא 0. נתחיל את פעולת האלגוריתם על ידי הוספה של 1 לערך של i. כדי לא לחרוג מגבולות האינדקס של S נבצע מודולו בין i לבין האורך של S, במקרה זה אנחנו נישאר עם i = 1, אך אם נגיע למצב שבו i = 10, נשים לב שלא קיים אינדקס כזה, ולכן פעולת המודולו תחזיר אותנו חזרה לערך של i = 0:

Value	1	6	9	2	7	3	5	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer	j=0	i=1								

כעת נוסיף לערך של j את S[i] (הערך של S באינדקס i). נבצע שוב מודולו עם האורך של S כדי לוודא שאיננו חורגים מגבולות המחרוזת. במקרה הזה S[i] = 6 ולכן j = 6 + 0 = 6 ואם נבצע מודולו ונשאר עם הערך j = 6:

Value	1	6	9	2	7	3	5	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer		i=1					j=6			

נחליף בין הערכים של S[i], S[j]:

Value	1	5	9	2	7	3	6	4	8	0
Index	0	1	2	3	4	5	6	7	8	9
Pointer		i=1					j=6			

כעת נקבע t = S[i] + S[j]. נבצע מודולו עם אורך S, ואז נפנה לערך S[t] ונשמור אותו במערך הפלט. במקרה זה t = 5 + 6 = 11. אחרי פעולת מודולו נישאר עם t = 2. לכן האיבר הראשון בפלט שלנו יהיה:

S[t] = 9

כעת נחזור על פעולת האלגוריתם עד שאורך הפלט יהיה כאורך הטקסט שנרצה להצפין. הפלט מאלגוריתם RC4 נקרא Key Stream, וכפי שנסביר לעומק בהמשך, ההצפנה נעשית בפעולת XOR בינו לבין הטקסט שנרצה להצפין.

להלן קטע הקוד המלא בשפת python3 שמקבל את האורך של S, מקבל את K (המפתח הסודי) ומקבל את הטקסט שברצוננו להצפין, ומדפיס כפלט את הטקסט המוצפן והטקסט המפוענח (מעביר תהליך של פענוח מבלי להשתמש בקלט המקורי של המשתמש):

```
try:
    s = int(input('enter the length of S: '))
except:
    s = 255
K = [ord(c) for c in input('enter secret key: ')]
k = len(K)
text = input('enter text to encrypt: ')
stream_len = len(text)
text_dec = [ord(c) for c in text]
S = list(range(0, s))


# KSA:
j = 0
for i in range(0, s):
    j = (j + S[i] + K[i % k]) % s
    S[i], S[j] = S[j], S[i]

# PRGA:
key_stream = []
i, j = 0, 0
while len(key_stream) < stream_len:
    i = (i + 1) % s
    j = (j + S[i]) % s
    S[i], S[j] = S[j], S[i]
    key_stream.append(S[(S[i] + S[j]) % s])

# encoding:
text_enc = [chr(text_dec[i] ^ key_stream[i]) for i in range(stream_len)]
print(''.join(text_enc))

# decoding:
text_dec = [chr(ord(text_enc[i]) ^ key_stream[i]) for i in
range(stream_len)]
print(''.join(text_dec))
```

פלט לדוגמא:

```
enter the length of S:
enter secret key: Secret Key
enter text to encrypt: This is a secret message!
?"?i_8?xdy;??BDdX
This is a secret message!
```

שימו לב כי ההודעה המוצפנת מוצגת ב-UTF8.

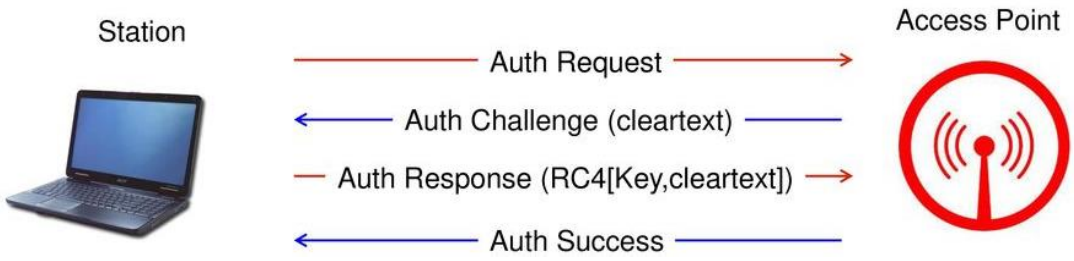
WEP - Wired Equivalent Privacy

אלגוריתם ה-WEP הוא אלגוריתם אבטחה של רשתות נתונים אלחוטיות הפועל מעל שתי השכבות הנמוכות במודל חמשת השכבות, השכבה הפיזית ושכבת הקו. מטרת האלגוריתם היא הצפנה השידור האלחוטי באמצעות סיסמא המוסכמת בין המחשב לנקודת הגישה, כך שתתקבל פרטיות שקולה לזאת של חיבור קווי.

ל-WEP יש מספר גרסאות, שההבדלים ביניהם הוא אורך ה-Seed (גרעין). ה-Seed מורכב מהמפתח הסודי של המשתמש ומחלק נוסף בשם Initialization Vector, או בקיצור IV. בעצם, ה-IV הוא מספר רנדומלי (או במקרה של מחשב, רק סודו-רנדומלי). במידה ולא היה לנו את ה-IV, התוצאה מה RC4 הייתה זהה למפתח, מה שיהפוך התקפות מסוימות (יפורט בהמשך) לפשוטות אפילו יותר.

אורך ה-Seed, בגרסה המקורית של האלגוריתם, הינו 64 סיביות, כאשר 24 סיביות מכילות את ה-IV ו-40 הסיביות האחרות מכילות את המפתח, שנקבע על ידי מנהל הרשת. בגרסאות החדשות יותר, ה-Seed באורך 128 או 256 סיביות. האלגוריתם עובד בשלבים הבאים:

ראשית, מתבצע אימות בין הנקודת גישה והלקוח (אופציונלי):



[מקור: Published by Ethan Montgomery]

- כדי לבצע אימות, הלקוח שולח ל-AP בקשת אימות (authentication).
- ה-AP מחזיר מספר אקראי באורך 128 סיביות.
- הלקוח מצפין את המידע שקיבל באמצעות המפתח המשותף ומחזיר את המידע המוצפן ל-AP.
- ה-AP מפענח את ההודעה ואם המידע ששלח זהה לפענוח הוא מאמת את הלקוח.

ה-AP שולח אתגר, כדי לוודא שיש ללקוח את המפתח האמיתי. למרות זאת, יש דרכים פשוטות לזייף אימות (יפורט בהמשך).

לאחר שהלקוח אומת, התקשורת בין ה-AP ללקוח מוצפנת ומתבצעת כך:

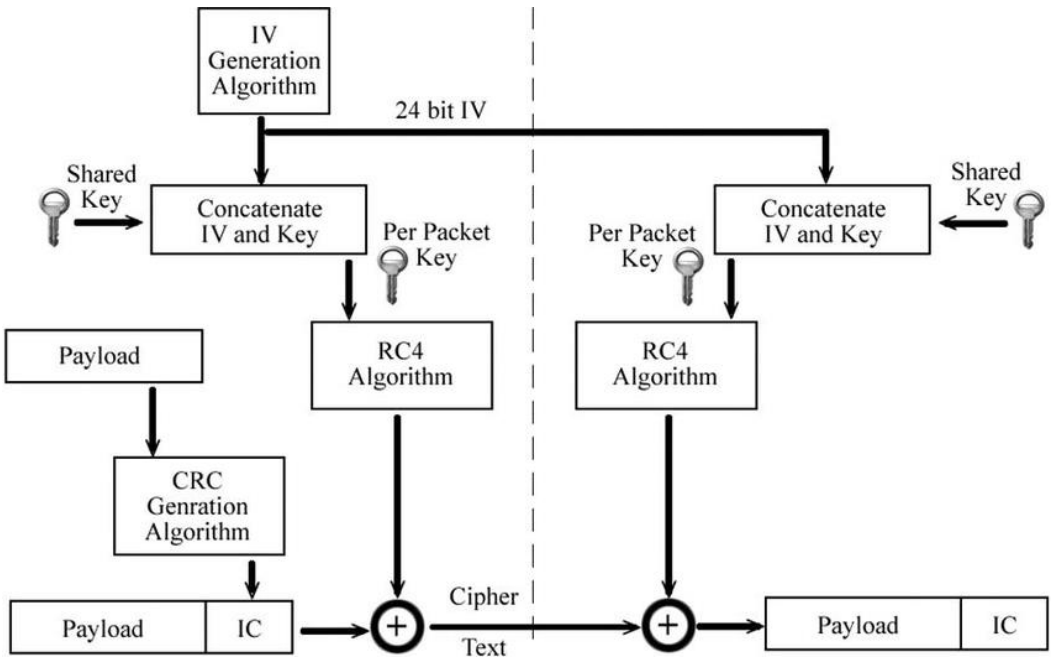
הצפנה:

- לוקחים את ה-Seed שאותו מצפינים באמצעות ה-RC4, ומקבלים רצף סודו-אקראי.
- המידע שישלח משורשר (צֵרוּף מחרוזת בתוכנת מחשב לסופה של מחרוזת אחרת- מילוג) עם ה-Checksum (CRC32) של המידע (מחזיר 32 סיביות שנקבעו על ידי הפלט ומשמש לאימות המידע, המטרה היא לזהות שגיאות בשידור ההודעה), ועובר XOR עם הרצף שהתקבל מה-RC4. זהו ה-Ciphertext, דהיינו המידע המוצפן.
- ל-IV משורשר ה-Ciphertext.

פענוח:

- מוציאים את ה-IV, מייצרים את ה-Seed ובאמצעות RC4 משיגים את ה-keystream (משום ש-RC4 הוא דטרמיניסטי, אותו ה-Seed יוביל לרצף זהה).
- מבצעים ל-Ciphertext את פעולת ה-XOR, מקבלים את ה-data וה-checksum.
- מבצעים על המידע שהתקבל checksum (CRC32) ומוודאים שה-checksum זהה לזה ששורשר ל-data כדי לאמת את המידע. ה-checksum לא משמש בתור חתימה דיגיטלית ולכן השימוש בו לא נועד לאבטחה.

בתרשים ניתן לראות את ההצפנה והפענוח של מנגנון ה-WEP:



[מקור: researchgate.net]

[1] הסימון דמוי גלגל מסמן את הפעולה XOR, שהיא סימטרית, ולכן ביצוע XOR פעמיים עם המחרוזת שהתקבלה מה-RC4 על המידע וה-checksum יוביל לאותו המידע וה-checksum.

[2] לאחר ה-XOR המידע משורשר ל-IV.

CRC

כשאנחנו שולחים מידע ממקום למקום, נרצה לוודא כי המידע, אשר נשלח כראוי, לא מגיע תקול. לכן אנחנו משתמשים בבדיקות Checksum. הבדיקה שנדבר עליה כעת נקראת CRC n-bit, והיא משתמשת בפעולת XOR ובפולינומים יוצרים כדי לייצר מספר שניתן לבדוק בעזרתו את תקינות המידע. ה-CRC n-bit מייצר מספר בן n ביטים, וככל ש-n גדול יותר כך רמת הדיוק תעלה. לכל CRC נקבע פולינום (הפולינום היוצר של CRC n-bit) ממעלה n שמקדמיו, כאשר הם משורשרים, יוצרים רצף בינארי שמשמש לחישוב ה-CRC של רצף מידע כלשהו, כפי שיוסבר בהמשך. לדוגמה, הפולינום היוצר הקבוע של CRC32 הוא:

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

וכאמור הוא יוצר את הרצף הבינארי הבא, שישמש כמחלק:

100000100110000010001110110110111 = 0x104C11DB7

כעת, נעבור לפעולת האלגוריתם: תחילה נקבל רצף מידע, לדוגמה hello. נמיר אותו לרצף בינארי. את הרצף הבינארי נמיר למספר בינארי (כלומר נוריד את כל האפסים בתחילת רצף המידע). כעת, כאשר אנו משתמשים ב-CRC n-bit, נשרשר n אפסים לסופו של הרצף הבינארי. כעת, ניקח את n+1 הביטים הראשונים ברצף המידע, ונבצע פעולת xor עם המחלק (שייצרנו ממקדמיו של הפולינום היוצר). לתוצאה נשרשר את שאר הביטים ברצף המידע כמו שהם. נזיז את המחלק לביט הבא שערכו 1 ברצף המידע ונחזור על פעולת האלגוריתם (או באופן שקול, נוריד את כל האפסים מתחילת הרצף ונחזור על פעולת האלגוריתם). בסופו של דבר נקבל רצף בינארי שאורכו לכל היותר n ביטים.

כדי להמחיש את הפעולה, נבצע CRC-3 על רצף של 14 ביטים.

רצף המידע הינו: 11010011101101

נוסיף לו 3 אפסים בסוף ונקבל: 11010011101101000

הפולינום המוסכם עבור CRC-3 הוא $x^3 + x + 1$, שיוצר את המחלק 1011.

נתחיל את פעולת האלגוריתם:

- Data: 11010011101101000
- Divisor: 1011
- Result Data: 01100011101101000

כאן ביצענו את השלב הראשון באלגוריתם. לקחנו את ארבעת הביטים הראשונים של זרם המידע, וביצענו פעולת xor ביניהם לבין המחלק. שאר הביטים נשארו כמו שהם.

נמשיך את פעולת האלגוריתם:

- Result Data: 01100011101101000
- Shifted Divisor: 01011
- Result Data: 00111011101101000
- Shifted Divisor: 001011
- Result Data: 00010111101101000
- Shifted Divisor: 0001011
- Result Data: 00000001101101000
- Shifted Divisor: 00000001011
- Result Data: 00000000110101000
- Shifted Divisor: 000000001011
- Result Data: 00000000011001000
- Shifted Divisor: 0000000001011
- Result Data: 00000000001111000
- Shifted Divisor: 00000000001011
- Result Data: 00000000000100000
- Shifted Divisor: 000000000001011
- Result Data: 00000000000001100
- Shifted Divisor: 00000000000001011
- Result Data: 00000000000000111

קיבלנו מספר בן 3 ביטים: 111. נצמיד אותו לסופו של זרם המידע המקורי שלנו, ונקבל את זרם המידע 11010011101101111. את זרם המידע החדש נוכל לחלק שוב בפולינום היוצר באותה שיטה בדיוק, ואם נקבל רצף של 0 סימן שזרם המידע תקין ולא השתבש במהלך ההעברה, וכך למעשה הצד השני יכול לבדוק את תקינות המידע.

חולשות ופרצות אבטחה

לאלגוריתם ישנן מספר חולשות מרכזיות, בחלק זה נפרט אודותן.

ניתן לזייף בקלות הודעות אימות

לאלגוריתם WEP קיימות מספר גרסאות: Open System authentication ו-Shared Key Authentication. ההבדל ביניהן הוא האם ה-AP מבצעת אימות. תזכורת:

[מקור: Published by Ethan Montgomery]

למה יש צורך באימות? הרי בכל מקרה המידע מוצפן. ללא אימות, כל אחד יכול להתחבר ולהפיץ את הרשת בפקטות, כדי לפגוע בשירותים שה AP מעניק. כמו כן, תוקף יכול לבצע Replay Attack, בפשטות, לשחזר דבר שכבר קרה. לתוקף יכול לתפוס Frame שאומר לבנק להעביר סכום כסף מהחשבון של אחד המחוברים, ולשלוח אותה שוב...

כדי לזייף הודעת אימות, ניתן להסניף את תעבורת המידע, עד שתימצא בקשת אימות. לאחר מכן, יחזור אתגר ולבסוף הפתרון שלו.

בקלות, ניתן לבודד את ה-Key Stream (לבצע XOR) ולאחר מכן להשתמש ב-Key stream לכל הודעה שתגיע בעתיד, לדוגמא:

challenge= 0x31a1 = 0011 0001 1010 0001

(נשתמש במספר אקראי באורך 16 סיביות בשביל ההדגמה. המספר מוצג בבסיס 16, כלומר בסיס הקסדצימלי):

Encrypted Challenge = 0x3ef0 = 0011 1110 1111 0000

נבצע XOR ונקבל:

Key Stream = 0000 1111 0101 0001

ה- IV קצר מידי

משום שאורך ה-IV הוא 24 סיביות, ל-IV יש 2 בחזקת 24 אפשרויות: 16,777,216 - אפשרויות. ההסתברות לקיום שתי פקטות עם אותו ה-initialization vector מתוך n פקטות: נחשב את ההסתברות למקרה ההופכי - שלכל הפקטות IV ייחודי: (ונחסר)

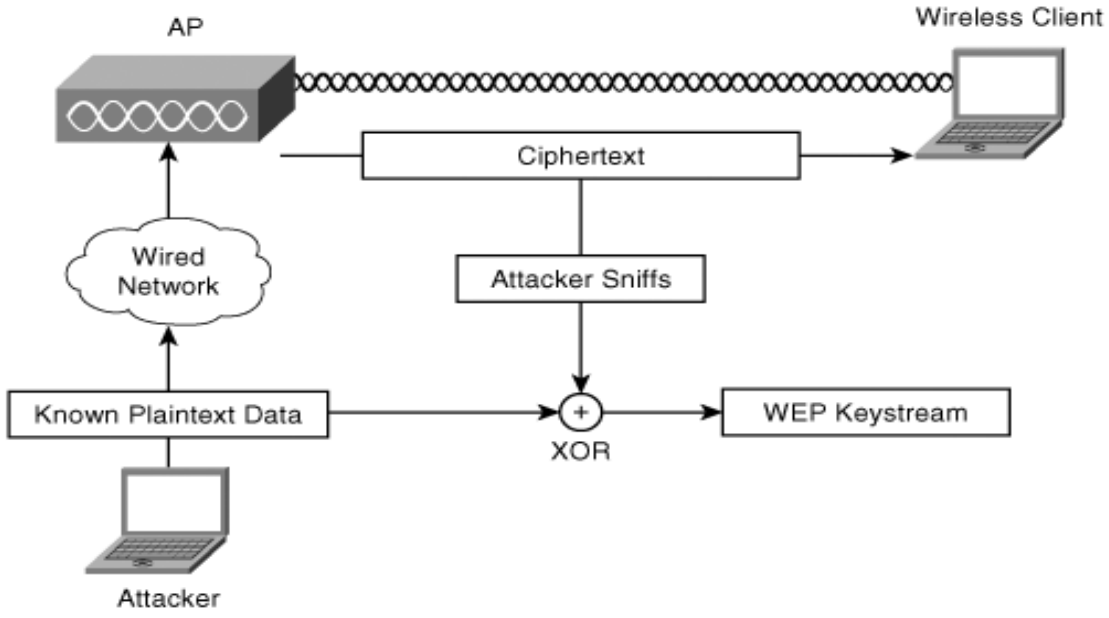
$$\frac{1}{\left(2^{24}\right)^n} \cdot \frac{2^{24}!}{\left(2^{24}-n\right)!} = \frac{2^{24}\left(2^{24}-1\right)\ldots\left(2^{24}-n+1\right)}{2^{24n}}$$

[החישובים נכתבו באמצעות המחשבון symbolab]

(יש 2^{24} אפשרויות ל-IV הראשון, 1 - 2^{24} למספר השני...), למרות שבמבט ראשון, נראה שההסתברות לקיום שתי פקטות היא אפסית, כאשר n = 5000, ההסתברות גדולה מ-50%. ראו - פרדוקס יום ההולדת.

למה זה בעייתי?

אדם שמנסה להשיג גישה ל-AP, יכול להתחבר ל-AP אחר, למשל ה-AP בביתו, לשלוח הודעה, למשל ping, לאחד המחוברים לרשת שהוא רוצה להשיג גישה אליה. הוא בנוסף יסניף את התעבורה וישיג את הפקטה המוצפנת. לאחר ביצוע XOR בין המידע המוצפן שקיבל מההסנפה, למידע הלא מוצפן שנשלח על ידו, המידע יתבטל והתוצאה שתתקבל הינה ה-keystream. כאשר הרצף נתון, ניתן לפענח בקלות כל פקטה שמוצפנת עם אותו ה-IV.



[מקור:etutorials.org]

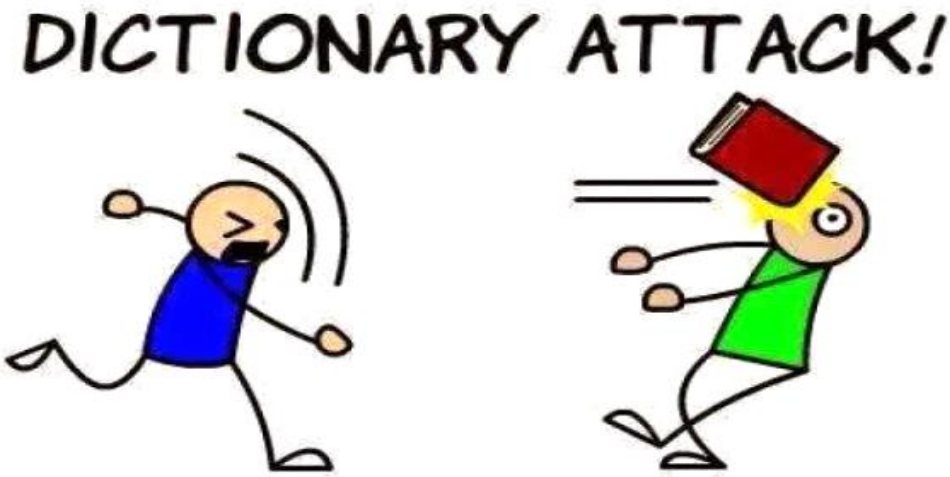
בשיטה זאת ניתן לייצר מילון רצפים (תלוי IV) שיאפשר לפענח לפחות חלק מהפקטות משום שכמות ה-IV האפשריים היא לא כל כך גדולה ואורך רצף מקסימלי הוא 2304 ביטים (ה-MTU של 802.11) לכן, מילון של כל הרצפים ידרוש בערך 38GB, מספר הגיוני בהחלט. כמו כן, ניתן לפענח את המפתח באמצעים שונים. הפתרון המתבקש הוא להאריך את ה-IV, אך הארכת ה-IV תהפוך את התקפת ה-FMS (שתפורט בהמשך) לחזקה יותר, משום שחלק גדול יותר מה- Seed יהיה נתון לפורץ.

ניהול המפתח והאורך שלו

ניהול המפתחות הוא אחד החולשות העיקריות של WEP. לרוב הרשתות המבוססות WEP יש מפתח אחד לכל משתמשי הרשת, מפתח שמוחלף לעיתים רחוקות או בכלל לא, זה בעקבות חוסר הנוחות בהחלפת המפתחות ברשת. כאשר המפתחות לא מוחלפים לעיתים קרובות, רשתות WEP הופכות לקלות לפריצה אפילו יותר.

כמו כן, מפתח באורך 40 סיביות אמנם היה בטוח ב-1997, אך כיום הוא חשוף להתקפות brute force. למפתח WEP 104 יש 2^{104} אפשרויות, מה שמייתר התקפה מסוג זה? לא, כמעט תמיד, יש מחולל סיסמאות, שלוקח סיסמא "נחמדה" כגון password (לא סיסמא מומלצת לכל הזדהות, משום שהיא אחת מהסיסמאות הנפוצות ביותר וקל לנחש אותה) וממיר אותה לסיסמא פחות נחמדה, למשל 20b7e73523.

לכן, ניתן להשתמש ב-Dictionary Attack (בפשטות לקחת מילון של סיסמאות נפוצות, להעביר אותן במחולל ולבדוק אם הן מתאימות)



[מקור: devhumor]

זה לא הכל, למרות שמחולל סיסמאות הוא לא קבוע מראש ושונה אצל ספקים שונים, לעיתים קרובות הספקים השונים משתמשים במחוללים זהים. בעקבות המבנה של המחוללים, הם לא כיסו את כל האפשרויות לסיסמאות WEP.

במקום מעל לטריליון סיסמאות אפשריות, Tim Newsham הציג בכנס BlackHat שישנן רק 2 בחזקת 21 אפשרויות לסיסמא, כלומר בערך 2 מיליון אפשרויות. Newsham 21-bit WEP attack היא התקפת brute force שמפצחת סיסמאות של wep בראוטרים מחלק גדול של הספקים בפחות מדקה.

The KoreK chopchop attack

קורק (השם בדוי) פרסם התקפה נוספת, בשם KoreK Key Recovery Attack, התקפה שמבוססת על רעיונות דומים ל-FMS ומאפשרת לגלות את המפתח, אך היא מודרנית יותר (פורסמה בשנת 2004).

ההתקפה Chopchop שקורק פרסם (בשנת 2004), מאפשרת לפענח את ה-m ביטים האחרונים של המידע, עם 128m חזרות בממוצע ומאמץ חישובי זניח.

ההתקפה משתמשת במימוש לקוי של ה-CRC ומניחה שקיים אורקל, שבודק האם ה-checksum בפקטה מוצפנת הוא נכון או לא.

בניגוד להתקפה FMS, שהיא ההתקפה הראשונה שמפענחת את המפתח, ההתקפה לא תוצג במלואה, אלא רק הרעיונות המרכזיים שלה. לקריאה נוספת, ראו:

https://www.aircrack-ng.org/doku.php?id=chopchoptheory

הסימן ⊕ ייצג את פעולת ה-XOR אלא אם כן צויין אחרת, נסמן את המידע הלא מוצפן בתור D, את ה-checksum בתור J ו- I ואת keystream בתור K. כמו כן, נסמן 5X = I3+D (תזכורת, הכוונה ל-XOR)

Frame 1:

```
___ DATA ____     ___ICV ___
D0 D1 D2 D3 D4 D5 J3 J2 J1 J0
⊕  ⊕  ⊕  ⊕  ⊕  ⊕   ⊕  ⊕  ⊕  ⊕
K0 K1 K2 K3 K4 K5 K6 K7 K8 K9
=  =  =  =  =  =  =  =  =  =
S0 S1 S2 S3 S4 S5 S6 S7 S8 S9
```

Frame 2:

```
___ DATA ___   ____ICV ____
D0 D1 D2 D3 D4 I3 I2 I1 I0
⊕  ⊕  ⊕  ⊕  ⊕   ⊕  ⊕  ⊕  ⊕
K0 K1 K2 K3 K4 K5 K6 K7 K8
=  =  =  =  =  =  =  =  =
R0 R1 R2 R3 R4 R5 R6 R7 R8
```

אנחנו ננסה להגיע מ-Frame1 ל-Frame2 באמצעות הניחוש של X (נתייחס אל X בתור נתון).

מתקיים ש:

R0 - R4 = S0 - S4 (בהתאמה)

$R5 = I3 \oplus K5 = I3 \oplus (D5 \oplus D5) \oplus K5 = (I3 \oplus D5) \oplus (D5 \oplus K5) = X \oplus S5$

את I2 – I0 ניתן לגלות באמצעות J3 – J1, ניתן לגלות לאחר חזרה בשלבים של האלגוריתם CRC, עם שימוש ב-X.

כדי לבדוק שה-X שנבחר נכון, תוקף יבדוק האם ה-checksum של ה-Frame שיוצר נכון, במידה וכן, ה-X שנבחר נוחש נכון, אחרת, צריך לנסות שוב. איך ניתן לבדוק זאת?

1) תוקף יכול להתחבר לרשת עם תחנה A, משום שניתן לזייף בקלות אימות. התוקף ישלח את הפקטה שאותה הוא רוצה לבדוק, מ-A ליעד המקורי. אם ה-checksum נכון, ההודעה תימסר ויהיה ניתן להסניף אותה. אחרת, ההודעה תושלך על ידי ה-AP.

2) דרך נוספת, היא לשלוח את ההודעה עם תחנה שלא מחוברת ל-AP. במידה וה-checksum נכון, ה-AP ישלח הודעת שגיאה ויבקש ממנו להתחבר לרשת, אחרת, ה-AP יפיל את ההודעה.

מה השגנו בעצם?

באמצעות X, ניתן להשיג את J0, שתלוי רק בו (בעקבות המבנה של CRC):

$S9 = J0 \oplus K9$

$K9 = J0 \oplus S9$

ומשום שגם S9 וגם J0 נתונים, השגנו ביט אחד של ה-keystream. על ה-Frame שיצרנו ניתן לבצע את התהליך שוב פעם, כדי לקבל ביט נוסף...

ההתקפה נקראת בקיצור FMS Attack. היא מנצלת מימוש לקוי של ה-RC4 ב-WEP. נסמן את ה-keystream בתור מערך של ביטים X, למשל X[0] הוא הביט הראשון ב-keystream. תיבת התמורה תסומן באות S והמערך K ייצג את הסיסמא (בדומה להדגמת ה-RC4).

נניח שאנחנו יודעים את ה-l ביטים הראשונים בK, ונרצה לגלות את K[l], הביט ה-l+1 כמו כן, X[0] נתון. (משום שבתוכן המוצפן נכללים גם ה-headers של השכבות הגבוהות יותר, ניתן לגלות יחסית בקלות את הביט הראשון של המידע הלא מוצפן ולבצע XOR).

כמו כן, נדרש ש- l=>2 אך משום שה-IV נתון, שלושת הביטים הראשונים של הסיסמא נתונים.

נסמן Slבתור תיבת התמורה לאחר l פעמים, כלומר כשב- KSA מתקיים ש: i = l - 1. את ההופכית נסמן בתור (S^)-1. אנחנו יודעים את S l משום שניתן לבצע את ההתחלה של KSA, מאותה הסיבה אנחנו יודעים את ערכו של J (ואת ערכו של i) במיקום ה-l.

נדרוש את התנאים הבאים (נשתמש רק בסיסמאות המקיימות את התנאים כי הם ברי בדיקה): (הסיסמאות שמקיימות את התנאים הבאים ב-l נקראות resolved condition).

1. $S_l[1] < l$
2. $S_l[1] + S_l[S_l[1]] = l$
3. $S_l^{-1}[X[0]] \neq 1$
4. $S_l^{-1}[X[0]] \neq S_l[1]$

נסתכל על X[0], השוויון הבא מתקיים (פשוט להסתכל ב-PRGA):

X[0] = S[S[1] + S[S[1]]]

RC4 KSA : For $i = 0$ to $N - 1$ do: $S[i] = i$ $j = 0$ For $i = 0$ to $N - 1$ do: $j = j + S[i] + K[i \bmod kLength]$ $\text{Swap}(S[i], S[j])$	**RC4 PRGA :** $i = j = 0$ **Loop :** $i = i + 1$ $j = j + S[i]$ $\text{Swap}(S[i], S[j])$ Return: $S[S[i] + S[j]]$	$\text{Swap}(S[x], S[y])$ $t = S[x]$ $S[x] = S[y]$ $S[y] = t$

[מקור: ויקיפדיה - RC4]

לכן, אם מתקיים ש:

If neither Sl [1], Sl [Sl [1]] nor S(l +1)[l] did participate in any further swaps in the rest of the RC4-KSA.

אזי:

X[0] = S[S[1] + S[S[1]]] = S[l] = S(l +1)[l]

משום ש Sl[J]מוחלף עם [(l + 1)]Sl[J], נובע ש:

Sl[J(l + 1)] = S(l +1)[l]

ולפי הנוסחה של J:

[Sl[J(l + 1)] = Sl[Jl + K[l] + Sl [l]

נקבל ש:

X[0] = S(l +1)[l] = Sl[J(l + 1)] = Sl[Jl + K[l] + Sl [l]]

X[0] = Sl[Jl + K[l] + Sl [l]]

ולבסוף:

$$\mathcal{F}_{fms}(\mathsf{K}[0],\ldots,\mathsf{K}[l-1],\mathsf{X}[0]) = \mathsf{S}_l^{-1}[\mathsf{X}[0]] - j_l - \mathsf{S}_l[l]$$

היא פונקציה שמחזירה את K[l], במידה וההנחה מתממשת. אחרת, היא מחזירה מספר שהוא בערך רנדומלי.

מהו הסיכוי שיתקיים התנאי?

הסיכוי שבכל פעם J לא יהיה אחד משלושת הערכים הוא (n-3)/n כאשר n הוא האורך של S (לרוב n=256). לכן, הסיכוי שההנחה מתממשת הוא:

$$\frac{(n-3)^{n-l}}{n^{n-l}}$$

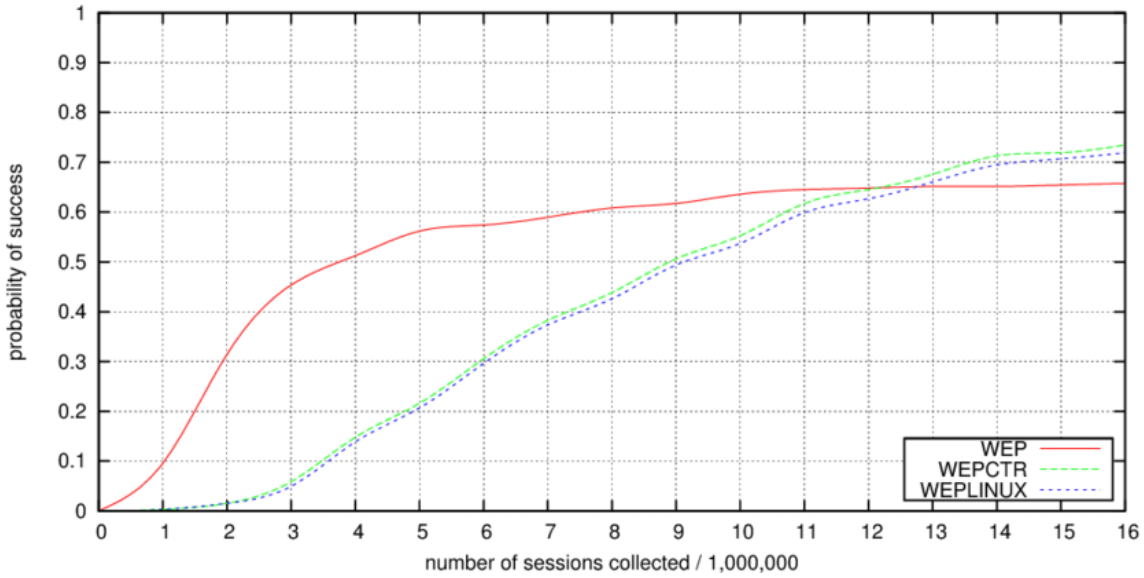
כאשר n=256 ובנוסף l = 3 , הסיכוי הינו בערך 5.07%.

מימוש FMS

תוקף יצבור כמות לא מבוטלת של IV ואת ה-X[0] שמתאים להם, הוא יבדוק אילו סיסמאות מכילות את המצב resolved condition, כאשר שלושת הביטים הראשונים של הסיסמא ידועים (משום שהם ה-IV).

על כל אחד מהם, תתבצע הפונקציה $\mathcal{F}_{fms}$ והתוצאה תישמר. כאשר יש מספיק IV, ניתן להניח שהביט הבא של המפתח התקבל.

התוקף יבצע שוב את אותה הפעולה, אך כשידועים ארבעה ביטים של המפתח... עד שהתוקף קיבל סיסמא שבסבירות יחסית גבוהה היא הסיסמא הנכונה.



[מקור: https://eprint.iacr.org/2007/471.pdf]

הגרף מייצג את סיכויי ההצלחה של FMS attack בתוכנה aircrack-ng, כשזמן העיבוד מוגבל לשלוש דקות. למרות שאחוזי ההצלחה אינם גבוהים במיוחד, הם גבוהים מספיק כדי לאיים על הבטיחות של ה-AP.

התקפות יותר עדכניות כמו KoreK (key recovery attack) משיגות אחוזי הצלחה גבוהים יותר ודורשות פחות מידע מוצפן. ההתקפה שפיתח KoreK למשל, משיגה בערך 97% הצלחה, עם עשירית מכמות ההפעלות. בניגוד להתקפה FMS שמשיגה פחות מ-80% הצלחה, גם כשכמות ההפעלות היא גבוהה מאוד (זמן עיבוד נוסף משפר מאוד את התוצאה של FMS באופן מאוד משמעותי).

סיכום

רשתות אלחוטיות הן דבר שכיח ונמצאות בכמעט כל בית, מסעדה, משרד... אמנם רשתות אלו מאפשרות ניידות ונוחות מירבית, אך יש להן גם חסרונות. המידע מועבר במרחב פתוח, באמצעות גלי מיקרו למשל. לאור זאת, המידע המועבר חשוף להאזנה של כל מי שנמצא בסביבת המשדר.

כדי למנוע חשיפת מידע רגיש יש לנקוט אמצעי הגנה ואבטחה על הרשת האלחוטית. לשם כך נכללים בתקנים השונים אמצעי הצפנה, כמו WEP, שנועדו להבטיח שרק הצד המשדר והקולט יוכלו לפענח את התוכן המשודר. מנגנון ההצפנה WEP הוא מנגנון הצפנה יחסית פשוט, שנועד לאבטח רשתות אלחוטיות כדי להעניק פרטיות שקולה לזאת של חיבור קווי.

כפי שהוצג, המנגנון כולל כמה חולשות וכשלים מבניים, המאפשרים להשיג את המפתח לרשת או לחלופין לפענח את המידע ללא המפתח הצפנה. נכון לעכשיו, מומלץ להשתמש ב- WPA2, אלגוריתם הצפנה חדש יותר ומאובטח יותר. אולם גם ב- WPA2 ישנן בעיות אבטחה ולכן כדאי להשתמש בגרסה עדכנית של המנגנון. זאת ועוד, WPA3 כבר הוכרז וצפוי להחליף את WPA2 בשנים הבאות. כיום, השימוש ב- WEP רק מעניק אשלייה של אבטחה וכלים מודרניים מאפשרים לפרוץ לרשת מבוססת WEP בדקות וחברות שלא עדכנו את האבטחה נחשפו לפריצות, למשל TJ Maxx שנגנבו ממנה פרטי אשראי של 94 מיליון לקוחות.

על עצמנו

יהלי הכט, בן 15 מרעננה, לומד בכיתת מחוננים בחטיבת השרון (רעננה) ומשתתף בתוכנית אודיסאה, מרכז מדעני העתיד, באוניברסיטת תל אביב לנוער. במסגרת תכנית אודיסאה, שבה אנחנו לומדים במסלול סייבר בשנה א', התבקשנו לכתוב מאמרים בנושא הקשור לרשתות תקשורת בהנחייתו של שלומי בוטנרו. בעקבות החיבה שלנו למתמטיקה, בחרנו לכתוב על WEP, מנגנון שאמנם האבטחה שלו ניתנת לפריצה על ידי כלים אוטומטיים כבר לא מעט שנים, אך למרות זאת, מרתק ללמוד על הפרצות, גם אלו שלא הוכנסו למאמר לבסוף .

מתן ונגוש, בן 15 מכפר סבא, לומד בכיתת מחוננים בחטיבת הביניים על שם משה שרת בכפר סבא ועולה לכיתת המחוננים בבית הספר התיכון על שם יצחק רבין בכפר סבא, לומד לתואר ראשון במדעי המחשב בתכנית אודיסיאה, מרכז מדעני העתיד, באוניברסיטת תל אביב לנוער.

במהלך הלימודים בתכנית אודיסאה, נדרשנו לכתוב מאמרים, כאשר הנושא הכללי היה רשתות תקשורת. בחרנו לכתוב את המאמר על פרוטוקול האבטחה WEP בגלל העניין והאתגר המתמטי שהוא עורר אצלנו, ורצינו ללמוד עליו עוד ולשתף את המידע שצברנו.

מקורות

IEEE 802.11 WEP (Wired Equivalent Privacy) Concepts and Vulnerability

- https://pdfs.semanticscholar.org/beb2/90fb1db86715c5ddb256245bff2ca1bb1dec.pdf

What's Wrong With WEP?

- http://www.opus1.com/www/whitepapers/whatswrongwithwep.pdf

WEP Keystream and Plaintext Recovery

- http://etutorials.org/Networking/Wireless+lan+security/Chapter+6.+Wireless+Vulnerabilities/WEP+Keystream+and+Plaintext+Recovery/

Attacks on the WEP protocol

- https://eprint.iacr.org/2007/471.pdf

What is RC4?

- https://paginas.fe.up.pt/~ei10109/ca/rc4.html

Computer Network | RC4 Encryption Algorithm

- https://www.geeksforgeeks.org/computer-network-rc4-encryption-algorithm/

RC4 cipher is no longer supported in Internet Explorer 11 or Microsoft Edge

- https://support.microsoft.com/en-us/help/3151631/rc4-cipher-is-no-longer-supported-in-internet-explorer-11-or-microsoft