אבטחת תקשורת - מטלה 2

מגישים: רועי חשאי ויתיר גרוס

חלק א - Padding Oracle Attack 2
הקדמה 2
שלב 1 2
שלב 2 2
שלב 3 2
שלב 4 3
התקפת POA 6
חלק 2 - DNSSec 8
תהליך הפתרון 8

חלק א - Padding Oracle Attack

הקדמה

בדוח זה נסביר כל חלק מהסקריפט מהו עושה נדגים את הלוגיקה ואז נריץ את ההתקפה על מס׳ דוגמאות

שלב 1

```
from Cryptodome.Cipher import DES
import sys
import subprocess
```

תחילה נייבא את הספריות המתאימות. נשים לב שאלו שאנו מייבאים את DES אך ורק כדי לדעת את גודל הבלוק במקום להשתמש במס׳ 8 כhardcoded, אין בסקריפט זה ביצוע של הצפנה ישירה וישנו רק שימוש באורקל כדי לפצח את הסייפר כמו שנראה כדלקמן.

שלב 2

ראשית, נבנה פונק Xor אשר בהמשך תשמש אותנו לחישוב הבתים במתקפה. פונק אשר מקבלת שלושה מספרים ומחזירה את תוצאות הXor בין שלושתם.

```
def xor(a, b, c):
    """
    input: a,b,c (int)
    output: a XOR b XOR c (bytes)
    """
    return bytes([a ^ b ^ c])
```

נשים לב שיש פה עניין של לקבל טיפוס int ולהחזיר byte לפי הדרישה של התרגיל ולכן נחזיר מערך בתים עם בייט בודד שהוא יהיה התשובה כדי לקבל תשובה בבתים ולא כint. עניין של טיפוסים והמרות של פייטון. באמצעות פונק זו אנחנו נחלץ את הP_i ואת הX_j המתאימים כדי לשבור את הסייפרטקסט

שלב 3

נכתוב כעת את הפונק אשר תקרא לאורקל, נראה אותה

```
def call_oracle(ciphertext, iv):
    result = subprocess.run(
        [sys.executable, "oracle.py", ciphertext.hex(), iv.hex()],
        stdout=subprocess.PIPE,
        stderr=subprocess.DEVNULL
    )
    try:
        return result.stdout.strip() == b'1'
    except:
        return False
```

פונק זו נועדה לקרוא לאורקל אשר ממומש בסקריפט אחר. נביט על הסקריפט של האורקל ונסביר אותו.
הפונק המרכזית בסקרפיט oracle.py היא כמובן oracle. נדגיש כי סקריפט זה מקבל כCommand line
args את הסייפר ואת הiv כדרוש וקוראת מקובץ בשם key.txt את המפתח.

```
def oracle(ciphertext, key, iv):
    cipher = DES.new(key, DES.MODE_CBC, iv)
    decrypted_text = cipher.decrypt(ciphertext)
    try:
        unpad(decrypted_text, DES.block_size)
        return True
    except ValueError:
        return False
```

הפונק oracle היא פונק אשר מחזירה אמת או שקר לאם יש ריפוד חוקי לטקסט. כלומר היא מקבל סייפרטקסט מוצפן מבצעת את הפענוח. לה כמובן יש את המפתח ומחזירה אמת או שקר רק אם הפענוח יצר מחרוזת אשר יש לה ריפוד חוקי. מה זה ריפוד חוקי? בשיטת PKCS7 ראינו שמרפדים בכמות בתים שמתחלקת ב8 וכמות הבתים שריפדנו זה מה שכותבים בבתים שרופדו. ולכן Hello World\x05\x05\x05\x05\x05 יהיה ריפוד חוקי אכן הוספנו 5 בתים ויש 5 בתים שכתוב בהם הערך 5.
לעומת Hello World!\x05\x05\x05\x05 רואים שלכאורה יש 5 בתים של ריפוד אבל בפועל יש רק 4 בתים שיש בהם את הערך 0x05. וזה לא חוקי.
הפונק שכתבתנו קודם כל מפענחת את הטקסט(ולה מותר כי היא לא מחזירה את הטקסט מפוענח או משהו בסגנון), מנסה לבצע את הunpadding ולהוריד את הpad ותצליח רק במקרה שהוא חוקי. כלומר בדיוק ההתנהגות שאנחנו רוצים. התהליך מדפיס 0/1 ובהתאם למה שהיא הדפיסה הסקריפט הראשי יודע להגיד האם הריפוד חוקי או לא שכמובן 1 יגיד חוקי ו0 לא.

שלב 4

כעת נתבונן על הפונק המרכזית שתשבור את ההצפנה ותבצע את המתקפה ונסביר כל חלק בו.

```
def padding_oracle_attack(ciphertext, iv):
    blocks_cnt = len(ciphertext) // DES.block_size
    plaintext_blocks = []
    for i in range(blocks_cnt):
        block_start = i * DES.block_size
        block_end = block_start + DES.block_size
        block_ciphertext = ciphertext[block_start:block_end]
        if i == 0:
            C_prev = iv
        else:
            C_prev = ciphertext[block_start - DES.block_size:block_start]
        XJ = bytearray(8)
        plaintext_block = bytearray(8)
        for byte_idx in reversed(range(8)):
            padding_val = 8 - byte_idx
            # Step 1: Brute-force byte at byte_idx in XJ
            for byte_val in range(256):
                XJ[byte_idx] = byte_val
                if call_oracle(block_ciphertext, XJ):
                    print(f"[+] Found valid byte {byte_idx} in block {i}: {hex(byte_val)}")
```

```
                break
        # Step 2: Recover plaintext byte at byte_idx in P2
        plaintext_block[byte_idx] = xor(padding_val, C_prev[byte_idx], XJ[byte_idx])[0]
        # Step 3: Update all known bytes to match new padding value
        next_pad = padding_val + 1
        for j in range(8 - byte_idx):
            pos = 7 - j
            XJ[pos] = xor(next_pad, C_prev[pos], plaintext_block[pos])[0]
    plaintext_blocks.append(bytes(plaintext_block))
return b''.join(plaintext_blocks)
```

ראשית, אנחנו מתחילים עם החלק הבא:

```
blocks_cnt = len(ciphertext) // DES.block_size
plaintext_blocks = []
for i in range(blocks_cnt):
```

בחלק זה נבין מהו כמות הבלוקים שיש בטקסט. הרי ההתקפה היא מפענחת בלוק בלוק אחד אחרי השני(ניתן להפוך את זה להיות מקבילי אך זה אינו דרישה ולכן נעשה זאת סדרתי) ולכן נחשב קודם כל כמה בלוקים יהיה עלינו לפצח. לאחר מכן נבצע לולאה על כל בלוק.
נעבור הלאה אל הלולאה בתוך כל בלוק ובלוק:

```
for i in range(blocks_cnt):
    block_start = i * DES.block_size
    block_end = block_start + DES.block_size
    block_ciphertext = ciphertext[block_start:block_end]
    if i == 0:
        C_prev = iv
    else:
        C_prev = ciphertext[block_start - DES.block_size:block_start]
```

ישנם חישובי של ההיסטים של הבלוק הנוכחי מתוך כלל הסייפרטקסט. ואז ישנו הבדל בין הבלוק הראשון לבין השאר. כי עבור הבלוק הראשון הבלוק שלפניו הוא הIV ולעומת כל שאר הסייפר הבלוק שלפני הוא הCi-1 ולכן נצטרך להבדיל את זה כבר פה כדי שבהמשך כאשר נעשה את הXor נוכל להתשמש בזה.
נעבור הלאה:

```
    XJ = bytearray(8)
    plaintext_block = bytearray(8)
    for byte_idx in reversed(range(8)):
        padding_val = 8 - byte_idx
```

בחלק זה אנחנו נאתחל את הXj להיות כרגע בלוק של אפסים. זה הבלוק שעליו נעשה את האיטרציות את הbroute force ונגדיל את ערכו ב1 עד אשר נמצא ערך שעבורו האורקל מחזיר אמת. כמו כן נשים את הplaintext כדי שנוכל לשמור את הבלוקים של הטקסט שאותם נחשוף.
כזכור את המתקפה אנחנו נבצע בייט בייט. ולכן אנחנו נכנס לעוד לולאה אשר חושפת בייט בייט החל מהבייט הכי ימני כלומר ה8(או אינדקס 7) של הבלוק ועד הבייט ה1(או 0). ישנו חישוב בנוסף לערך שאותו אנחנו רוצים למצוא בכל איטרציה. כי כאשר האורקל יגיד לנו אמת באיטרציה הראשונה זה כאשר

הבייט המתאים בP'2 הוא 0x01 ובאירטציה הבאה זה יגיד אמת כאשר שני הבתים הימניים הם 0x02 וכך הלאה. ולכן זה מידע שחשוב שנדע אותו למהלך הלולאה, שוב כדי להשתמש בו בXor בהמשך.

נעבור הלאה:

```
# Step 1: Brute-force byte at byte_idx in XJ
for byte_val in range(256):
    XJ[byte_idx] = byte_val
    if call_oracle(block_ciphertext, XJ):
        print(f"[+] Found valid byte {byte_idx} in block {i}: {hex(byte_val)}")
        break
```

כעת בחלק זה נבצע את הbroute force על Xj. נזכיר שאנחנו נקרא לאורקל ובכל פעם כאשר האורקל יחזיר אמת אנחנו נעצור ונוכל מתוך האמת לחשב את הערך של הטקסט.
במצב הזה כדי לקבל מהאורקל אמת נצטרך שלאחר הפענוח ערך הבייט האחרון יהיה 0x01 (במקרה ומדובר באיטרציה הראשונה של הבייט) כדי שהטקסט יחשב טקסט עם ריפוד חוקי. ולכן כמו שראינו בכיתה כאשר נבדוק את כל האפשרויות לערך של בייט נקבל בטוח שתוצאות הxor של אחד מהם(הxor המגיע מהפענוח של DES) יניב טקסט שהבייט האחרון שלו הוא בעל הערך 0x01 כלומר ריפוד חוקי ויחזיר עליו אמת. כמובן שהדבר הנ"ל תקף גם עבור 0x02 בסיבוב הבא. brute force יתן בוודאות אחד כזה מכיוון שעוברים על כל האפשרויות של הבייט. אנחנו רק צריכים להבין מיהו הבייט של Xj אשר נותן את הריפוד החוקי הזה.

נעבור הלאה:

```
# Step 2: Recover plaintext byte at byte_idx in P2
plaintext_block[byte_idx] = xor(padding_val, C_prev[byte_idx], XJ[byte_idx])[0]
```

לאחר שאנו יודעים מהו הערך של הבייט החיפשנו נוכל לגלות את הערך של הבייט byte_idx בטקסט! וזה כמובן מבלי לפענח את הטקסט עם המפתח הסודי. כל שעלינו לעשות הוא להשתמש בנוסחה שראינו בכיתה שכרגע היא משוואה בנעלם אחד

$$P_i[x] = P_2'[x] \oplus C_{i-1}[x] \oplus X_j[x]$$

את הערך של P'2 אנחנו יודעים כי מזה שהאורקל החזיר אמת זה כנראה כי הבייט האחרון הוא 0x01 (במקרה ומדובר בבייט הראשון שמנסים לפצח שהוא הכי ימני בבלוק, אחרת זה יעלה כמו שתיארתי, כלומר אם מנסים לפצח את הבייט השני מימין שכאשר האורקל יחזיר אמת נידע שזה יהיה כאשר הפענוח הניב בשני הבתים הימניים ביותר את הערך 0x02, איטרציה שלישית בכל שלושת הבתים הימניים ביותר 0x03 וכך הלאה) את Ci-1 אנחנו גם מכירים כי יש לנו את הסייפרטקסט המקורי ואת Xj אנחנו יודעים כי עשינו עליו את הbroute force ולכן כל מה שנותר הוא לגלות את Pi שזה הבייט הbyte_idx בטקסט המקורי!

נמשיך:

```
# Step 3: Update all known bytes to match new padding value
next_pad = padding_val + 1
for j in range(8 - byte_idx):
    pos = 7 - j
    XJ[pos] = xor(next_pad, C_prev[pos], plaintext_block[pos])[0]
```

ראשית נבצע המרה לאינדקס של הלולאה לשקף את הבייט שאותו מתקנים ומקבעים ע״י
pos = 7 - j
כי קודם עוברים על הבייט השמיני ואז שביעי וכך הלאה..
כעת כדי להמשיך את המתקפה לפאזות הבאות עלינו לבצע את התיקונים בכל הבתים שאמנם פיצחנו את הטקסט עבורם אך נצטרך לתקן את הבתים המתאימים אליהם בXj. כמו שתיארנו לעיל

0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x01 => oracle will return true
0x00 0x00 0x00 0x00 0x00 0x00 0x02 0x02 => oracle will return true
0x00 0x00 0x00 0x00 0x00 0x03 0x03 0x03 => oracle will return true
...

נשים לב שלאחר שפיצחנו את הטקסט בבייט השמיני אנחנו צריכים להכין את הXj ולקבע את הבייט הימני ביותר כדי שבbroute force של הבייט הבא משמאל נקבל אמת. זה יקרה כאשר הימני ביותר יהיה 0x02 ולאחריו בbrute force נקבל שוב 0x02. ולכן אנחנו בלולאה הזאת נעבור בייט בייט ונתקן מה שצריך לתקן בעצם נקבע אותו למה שבאירטרציה הבאה יתן אמת כאשר נשאל את האורקל את השאילתות.
לשם כך נשתמש באותה משוואה שהצגנו לעיל רק עם העברת אגפים קטנה כדי שנוכל לחלץ את Xj. כלומר כך

```
X_j[x] = P'_2[x] ^ C_i-1[x] ^ P_2[x]
```

ואז בסה״כ מקבלים שבאיטרציה הראשונה של הלולאה הפנימית חושפים את הבייט הראשון(מימין) ואז צריך לקבע בלוק אחד מימין לערך 0x02. לאחר מכן בפאזה הבאה לאחר מציאת הבייט השני מימין בטקסט נצטרך לקבע את ה2 בתים הימניים בXj בערך 0x03. ואז 3 בתים ימניים ביותר בערך 0x04 וכך הלאה עד לחשיפת הבלוק כל הבלוק. כאשר נסיים על הבלוק הנוכחי נעבור לבלוק הבא וככה נחשוף את כל הטקסט.

התקפות POA

נריץ כעת את הקוד עם הסייפר `You can't break me, I'm a padding oracle!`.
קודם כל צריך לרפד את הטקסט באופן חוקי, אנחנו מניחים שהריפוד הוא חוקי ותקין. כי בתור תוקף אנחנו רואים סייפרטקסט כלומר תוצאה של הצפנה חוקית. ונתייחס כיאלו אנחנו יירטנו את הסייפר. אז הריפוד וביצוע ההצפנה נעשו באופן חוקי.
אז נקבל ריפוד של

```
b"You can't break me, I'm a padding oracle!\x07\x07\x07\x07\x07\x07\x07"
```

ההצפנה של הטקסט יהיה

ff201a1db9d4314f1f7a7c996c25e6070c41be20362db832e16ffd6c3a7853b20ac400fc03fef7c49b7778bfc6c93a48

נשמור את המפתח שאיתו ביצענו את ההצפנה בקובץ key.txt את המפתח

poaisfun

נפעיל את הסקריפט עם הסייפרטקסט מלעיל יחד עם iv שכולו אפסים. כלומר נריץ:

```
(myenv38) roeehashai@roeeh ex2 $ python ex1.py
ff201a1db9d4314f1f7a7c996c25e6070c41be20362db832e16ffd6c3a7853b20ac400fc03fef7c49b7778bfc6c
93a48 0000000000000000
```

נקבל את הפלט(הוספנו הדפסות לצורך debugging כמובן שאינם נמצאים שם בקובץ המוגש)

```
ex2 — -zsh — 85×20
[+] Found valid byte 1 in block 3: 0x66
[+] Found valid byte 0 in block 3: 0x65
[+] Found valid byte 7 in block 4: 0xd6
[+] Found valid byte 6 in block 4: 0x3d
[+] Found valid byte 5 in block 4: 0x18
[+] Found valid byte 4 in block 4: 0x5f
[+] Found valid byte 3 in block 4: 0x1b
[+] Found valid byte 2 in block 4: 0x94
[+] Found valid byte 1 in block 4: 0x48
[+] Found valid byte 0 in block 4: 0x8e
[+] Found valid byte 7 in block 5: 0xc2
[+] Found valid byte 6 in block 5: 0xf2
[+] Found valid byte 5 in block 5: 0xfa
[+] Found valid byte 4 in block 5: 0x0
[+] Found valid byte 3 in block 5: 0xfe
[+] Found valid byte 2 in block 5: 0x1
[+] Found valid byte 1 in block 5: 0xc4
[+] Found valid byte 0 in block 5: 0x23
You can't break me, I'm a padding oracle!
(myenv38) roeehashai@roeeh ex2 $
```

כדרוש!

כעת נריץ עוד מס התקפות על הטקסט Hello World, אז לאחר ריפוד והצפנה עם אותו המפתח poaisfun נקבל את הסייפר:

33aaa3017e45337bd36342b3920be656

כמו כן נריץ עוד מתקפה על הטקסט RoeeHash שעבורו הסייפר יהיה לאחר הצפנה בהקסה כמובן

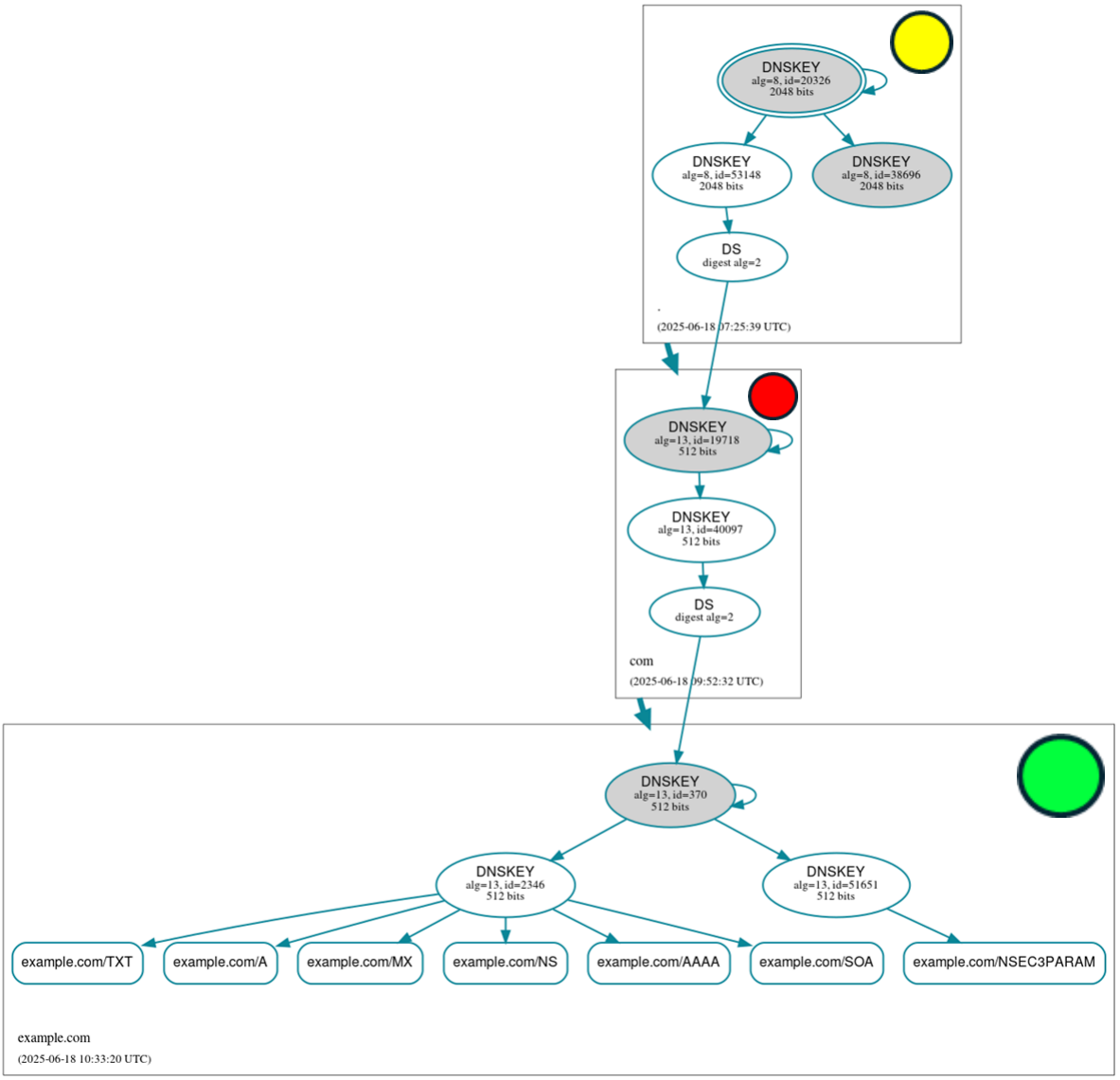
28874d534e9844fcb8bd2c770ce045ea

ונקבל את הפלטים

```
ex2 — -zsh — 85×20
(myenv38) roeehashai@roeeh ex2 $ python ex1.py ff201a1db9d4314f1f7a7c996c25e6070c41be
20362db832e16ffd6c3a7853b20ac400fc03fef7c49b7778bfc6c93a48 0000000000000000
You can't break me, I'm a padding oracle!
(myenv38) roeehashai@roeeh ex2 $ python ex1.py 33aaa3017e45337bd36342b3920be656 00000
00000000000
Hello World
(myenv38) roeehashai@roeeh ex2 $ python ex1.py 28874d534e9844fcb8bd2c770ce045ea 00000
00000000000
RoeeHash
(myenv38) roeehashai@roeeh ex2 $
```

חלק 2 - DNSSec

להלן תרשים אשר התקבל בDNSviz עבור example.com:



תהליך הפתרון

שלב 1 - ROOT מלבן עליון:

רואים בתרשים שלוש רשומות DNSKEY עם מפתחות של 2048 ביט, אלגוריתם 8 (RSA/SHA-256):

- id=20326 המפתח המרכזי הפעיל (KSK)
- id=35431, id=58509 מפתחות ישנים/חדשים לגיבוי או החלפה
- המפתח id=20326 חותם על ה-DS של אזור com. (שנמצא למטה)
- תאריך עדכון: 2025-06-18 07:25:39 UTC

מה זה אומר: זו נקודת ההתחלה, הרמה היחידה שמקבלת אמון מובנה אצל כל מערכת הפעלה (trust anchor). כלומר ישנו מפתח KSK שכולם מכירים מראש. הKSK של ה־root חותם על רשומת DS שמכילה את ה hash על ה KSK public של .com. ברגע שיש חתימה תקפה מהמפתח הזה - אפשר להאמין לרשומות של .com. כלומר שרת הROOT **מאשר** ש"המפתח הבא" (KSK של .com) באמת שייך ל-.com. ואם החתימה תקפה, הלקוח יודע שאפשר לסמוך על מה שיגיע מ-.com.

שלב 2 - COM מלבן אמצעי:

רואים בתרשים שני מפתחות DNSKEY, שניהם באלגוריתם 13 (ED25519, עם 512 ביט):

- id=19718 מפתח KSK
- id=40997 מפתח ZSK
- ה KSK חותם על ה ZSK וכך שאנחנו יודעים שהוא אותנטי.
- רשומת DS שמכילה את ה hash על ה KSK public של example.com נחתמה על ידי .com על באמצעות הZSK (שהוא id=40997)
- תאריך עדכון: 2025-06-18 09:52:32 UTC

מה זה אומר: עכשיו אנחנו סומכים על המפתחות של .com כי החתימה שלהם נבדקה מול ה-root. ה-KSK כאן חותם על ה-ZSK, וזה מאשר את זהות המפתח שמולו יבדקו הרשומות. ברגע שה-ZSK חתום על הרשומות (כמו DS ל־example.com אצלנו) אנחנו יכולים לסמוך גם עליהן.

שלב 3 - example.com מלבן תחתון:

רואים בתרשים שלושה מפתחות DNSKEY, (כולם ED25519 alg=13):

- id=370 ה־KSK של example.com.
- id=2546, id=15651 שני ZSK של example.com.
- ה KSK חותם על ה ZSK שלו.
- ה ZSK חותמים על כל רשומות הDNS שהם RRSet כפי שהוסבר בהנחיות התרגיל. והן: example.com/A, TXT, MX, NS, AAAA, SOA, NSEC3PARAM
- תאריך עדכון: 2025-06-18 10:33:20 UTC

מה זה אומר: קיבלנו DS מרמת .com שחתום ע"י המפתחות של .com, והוא מכיל את ה hash על ה KSK public של example.com ולכן מאמתים אותו, ה-KSK חותם על ZSK שמאמת את כל שאר הרשומות.

עכשיו הלקוח יודע שהתשובה אמינה אם ה-ZSK שחתם עליה הוא חלק משרשרת אמון שמתחילה ב-root, עוברת דרך ה-DS של .com, ומגיעה עד ה-KSK של example.com. כל עוד כל החתימות והטביעות אצבעות מתאימות זו לזו, התשובה אותנטית.

זרימת התהליך בפועל:

1. הלקוח (resolver) שואל מי ה־A של example.com
2. ה-Resolver פונה לשרתי ה-Root כדי לברר איפה השרתים של .com.
3. ה-Root עונה עם ה-NS של .com וגם עם רשומת ה-DS של .com. ה-DS הזה מכיל את ה-hash של ה-KSK של .com, והוא חתום ע"י ה-KSK של ה-Root (Trust Anchor).

4. ה-Resolver מאמת שה-DS של .com אכן תואם למפתח ה-KSK של .com ושנחתם ע"י ה-Root.
5. פנייה ל-.com: אחרי שאימתו את המפתחות של .com, ה-Resolver פונה לשרתי ה-.com ושואל מה ה-NS של example.com, ומה ה-DS של example.com?
6. השרתים של .com עונים עם ה-DS של example.com וגם עם ה-NS שלו. ה-DS הזה חתום ע"י ה-ZSK של .com, וה-Resolver מוודא את החתימה ע"י מפתחות ה-.com. בנוסף, ה-Resolver מוודא שטביעת האצבע שב-DS מתאימה ל-KSK הציבורי של example.com.
7. פנייה ל-example.com: ה-Resolver פונה לשרתים של example.com ומבקש את רשומת ה-A.
8. השרת עונה עם הרשומה וגם עם חתימות ה-RRSIG שלה. במקביל, ה-Resolver מקבל את ה-DNSKEY של KSK + ZSK) example.com).
9. עכשיו ה-Resolver מאמת: שה-KSK של example.com מתאים ל-DS שכבר אושר מ-.com. שה-KSK חותם נכון על ה-ZSK. שה-ZSK אכן חותם נכון על הרשומות (A, RRSIG וכו').

אם כל השלבים אומתו מה-Root ועד ה-ZSK של example.com - ה-Resolver יודע שהתשובה אותנטית. כלומר, אף תוקף לא שינה את המידע בדרך, ויש שרשרת אמון רציפה מה-root ועד הרשומה שביקשנו.ולא שונה על ידי תוקף מאזין או חיצוני כלשהו. וכמו ששאלנו בתרגיל שתמיד אפשר להגיד לה הפתח שמקבלים "אבל מי אמר שהוא אמין?" ולכן גם עליו נחתום ואז ושוב ניתן לשאול את אותה שאלה. אז כאן אנחנו רואים שאנחנו מתחילים ממפתח עם אמון מובנה וכל תהליך שרשרת החתימות מתחיל ממנו ולכן ניתן לקבל אמון!