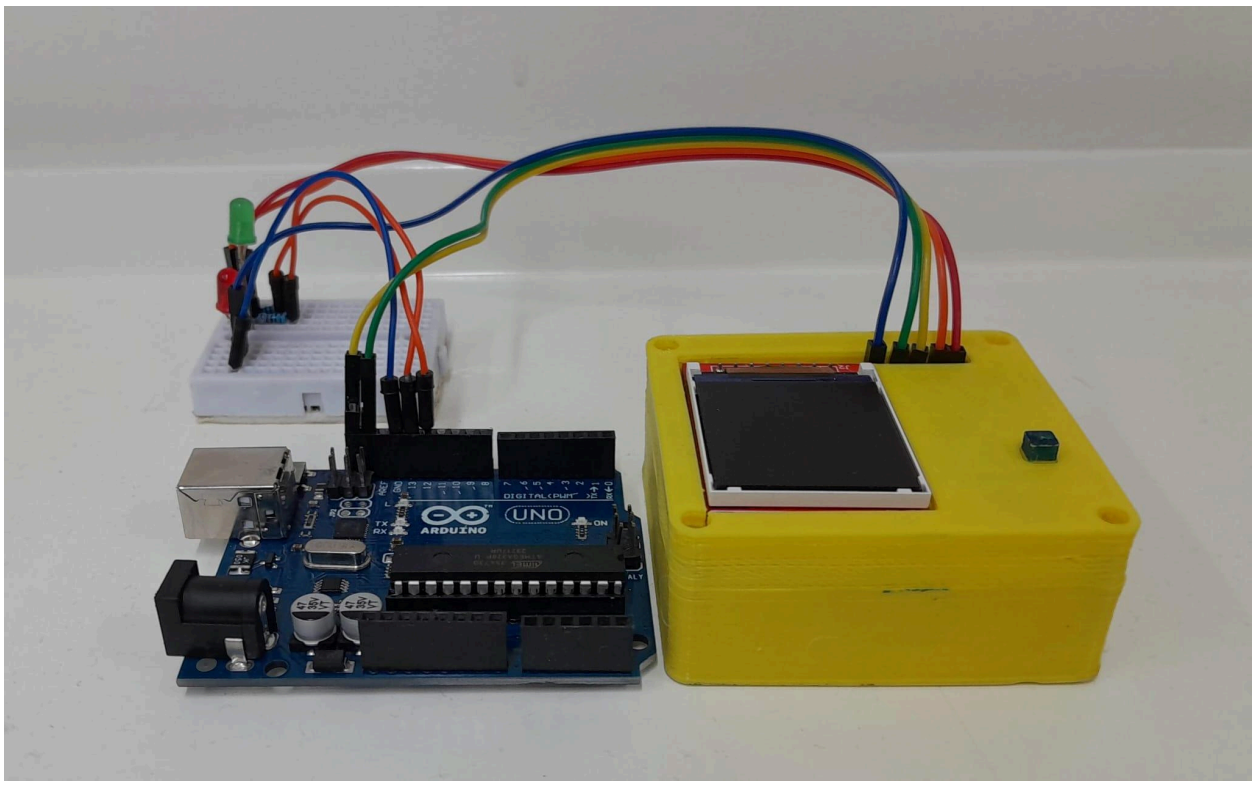
בס"ד

Time Side Channel Attack



מגישים: אבישי גונן ואריאל זקן

מרצה מנחה: מר אריה הנל

המרכז האקדמי לב, הפקולטה למדעי המחשב

תשפד, סמסטר ב'

Abstract

Time Side Channel Attack (TSCA) is a well-known attack from the Side Channel Attack family. These attacks exploit phenomena arising from the code's execution, not directly due to vulnerabilities in the code itself, but rather due to side effects produced by the code. In TSCA, the attack measures the algorithm's execution time, and from this measurement, it can deduce information about the secret data.

Our project comprises three parts, all related to TSCA, where we implement and demonstrate the attack.

1. **Part One**: We create a controller that receives passwords and checks if they match the stored secret password. If the password is correct, it returns true; otherwise, it returns false. The controller will be implemented in a naive manner, checking character by character. If an incorrect character is found, it returns false immediately. By measuring the response time for various passwords, we can deduce the correct password.
2. **Part Two**: Building on the first part, this time we introduce a random delay between each character check to disrupt the ability to perform TSCA. Here, we will perform TSCA using statistical analysis.
3. **Part Three**: We perform TSCA on a controller implementing an ECC-based message signing server (ECDSA). We will develop an attack to find the secret key used by the server to sign messages.

Detailed explanations of all topics mentioned in the abstract, such as TSCA, ECC, ECDSA, and more, will be provided in the following sections.

תוכן עניינים

Abstract **2**
תוכן עניינים **2**
מבוא **5**
חלק I **6**
הקדמה 6
המתקפה 6
מימוש 6
הקורבן 6
התוקף 7
חלק II **8**
הקדמה 8
המתקפה 8
מימוש 8
הקורבן 8
התוקף 9
I2C task 9
TFT task 9
TSCA task 9
אופטימיזציה 9
חלק III **11**
הקדמה 11
תיאוריה 12
קריפטוגרפיה בעקומות אליפטיות - ECC 12
אלגוריתם חתימה דיגיטלית בעקומות אליפטיות - ECDSA 13
אנוטציות 13
בעיית ה HNP 13
בעיית ה CVP 14
המתקפה 14
שלב א 14
שלב ב 15
שלב ג 15
מימוש 16
הקורבן 16
התוקף 16
עיבוד הנתונים 16
הקשר אבטחה **18**
מודל איומים 18
הנחות אבטחה 18

היקף ומגבלות 18
מדריך משתמש 19
Part1 19
Part2 19
Part3 19
עבודות עתידיות ושיפורים 21
שיפור לחלקים I & II 21
שיפור לחלק III 21
נוסחאות שלמות ((Complete Formulas 21
תיקון אורך הביטים ((Fixing the Bit-Length 21
נונסים ארוכים ((Long Nonces 21
סיכום 23
מקורות 24

מבוא

כיום ישנה מודעות לא חזקה מספיק לגבי Time Side Channel Attacks למיניהן, ובפרט למתקפות כאלו על אלגוריתמים שמתבססים על Eliptic Curve Cryptographic, וכאשר יש חוסר מודעות ממילא אנשים גם לא דואגים להגן בפני המתקפות הללו, מה שמשאיר מקומות רבים עם חולשה מסוכנת ביותר.

Elliptic Curve Cryptographic נחשב כיום לאחד ממנגנוני ההצפנה הכי חזקים ואמינים שקיימים בשוק, הוא נמצא בכל תחום כמעט - מהצפנה באמצעות אלגורתימי TLS, שמעבירים את רוב התעבורה המוצפנת כיום באינטרנט, ועד התחום הכי לוהט כיום, ה bitcoin.
ה ECC מחליף את ה RSA בהרבה תחומים כיוון שהוא יותר יעיל ונחשב גם ליותר חזק, לכן, במידה ויצליחו לפרוץ את האלגוריתם הזה, או לפחות לשבור מימושים שונים שלו שמשתמשים בהם היום בשוק, יווצר סיכון עצום לרוב מוחלט של המשתמשים ברשת כיום, סיכום שקשה לכמת אותו במספרים.

אנחנו בחרנו את הפרוייקט הזה כיוון שאנחנו מאוד אוהבים חומרה, ורצינו לממש פרוייקט על בקרים ולראות את היכולות של כל בקר ובקר.
כמו כן, אלגוריתם ההצפנה ECC הוא אלגוריתם מרתק, עם עולם מתמטי שלם שעומד מאחוריו, והרגשנו צורך להרחיב את האופקים שלנו ולצלול לתוך העולם הזה, בכדי שנוכל להבין יותר טוב את אופן הפעולה של אחד מין אלגוריתמי ההצפנה שהכי בשימוש כיום בעולם.

חלק I

הקדמה

בחלק בראשון של הפרוייקט נרצה כזכור להתמקד במקרה קלאסי של TSCA שהוא השוואה של מערכים איבר איבר בלולאה ויציאה מתי שהאיברים לא שווים, אלגוריתמי השוואה אלה נפוצים בקודנים של בניינים ושערים ובמשרדים.
אנו נרצה להדגים כיצד מדידה של הזמן ממתי שמכניסים סיסמא עד שחוזרת תשובה נותנת לשחזר את הסיסמא בזמן ליניארי לגודל הסיסמא (בניגוד לברוטפורס שכידוע הוא אקספוננציאלי)

המתקפה

ההתקפה שלנו מניחה שיש לנו דרך לשלוח ניסיונות של סיסמאות לקודן באופן שיטתי, שניתן לקבל חיווי האם הסיסמא נכונה או לא, ושניתן למדוד ברזולוציה את הזמן שלקח לבדוק את הסיסמא וכן בשביל הפשטות אנו יודעים מה האורך של הסיסמא.

ההתקפה על המערכת בחלק הראשון היא על הקוד הבא:

```
bool check_pass()
{
  // this is the vulnerable part of the code
    for (int i = 0; i < PASS_SIZE; i++){
        if(curr_pass[i]!=password[i])
            return false;
    }
    return counter <= PASS_SIZE;
}
```

ההתקפה שלנו הולכת ככה:
1) שלח 10 ניסיונות מהצורה הבאה X000…0 כך ש X הוא מספר עולה מ 0 עד 9 בהתאמה ותזמן כל אחד מהניסיונות.
2) קח את הספרה שהיא הכי **איטית** a, זו הספרה הראשונה בסיסמא
3) שלח 10 ניסיונות מהצורה הבאה aX000…0 ותזמן כל ניסיון.
המשך כך עד שתגיע לסיסמא המלאה!

ההתקפה עובדת מכיוון שב10 נסיונות הראשונים, אחד מהניסיונות שעשינו התו הראשון שלו בבדאות חלק מהסיסמא ולכן לפי האלגוריתם שראינו הלולאה תעשה עוד איטרציה ויווצר הפרש זמנים מדיד ולכן זו הספרה הנכונה, והדבר נכון גם לגביי המשך הספרות מכיוון שאנחנו תמיד משרשרים מקדימה את התווים הנכונים.

מימוש

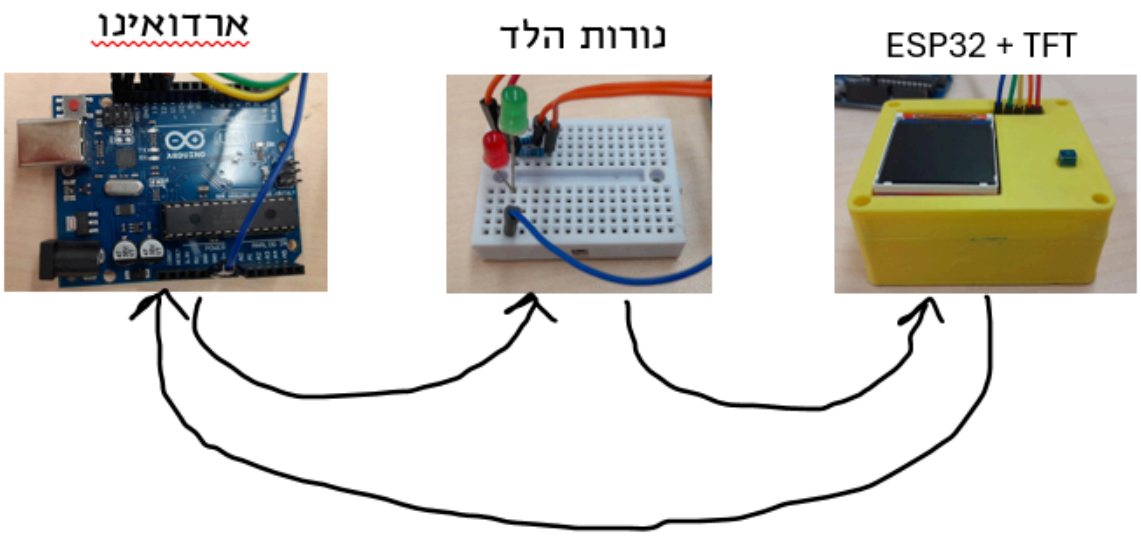
הקורבן

את קודן הפגיע אנחנו מימשנו באמצעות Arduino uno עם קודן שמתקשר עם הארדואינו באמצעות פרוטוקול I2C עם הסיפריה הרשמית של adafruit.

לארדואינו מחברות 2 נורות לד ,ירוקה ואדומה שאחת מבשרת שהסיסמא נכונה והשנייה מבשרת על סיסמא לא נכונה בהתאמה.
וכמובן שגם מממש את הפונקציה הפגיעה שלנו שבודקת האם הסמא שלנו נכונה.

<u>התוקף</u>

בהתוקף שלנו הוא ESP32 שמדמה את הפרוטוקול של הקודן שלנו דרך I2C ומתחזה אליו. וכך ניתן לשלוח הודעות באופן שיטתי לתוך הארדואינו. וכן ניתן למדוד את הזמנים באמצעות 2 חוטים שמחוברים אל הלדים[1].



[1] המימוש פה הוא ממש כמו חלק ב ולכן אנחנו חסכנו פה בהסברים על המימוש, ההבדל בין חלק א ל ב הוא כמות האיטרציות לכל ספרה של סיסמא.
וזה מכיוון שבכל מקרה אנחנו עושים ניתוח סטטיסטי מכיוון הרעש בבדיקות.

חלק II

הקדמה

אם בחלק הראשון ביצענו מתקפה בסיסית, עכשיו אנחנו עולים שלב. עכשיו לבקר יש שכבת הגנה נוספת, יש הוספת delay רנדומלי בתוך הפונקציה הפגיעה. נסביר על כך במתקפה.

המתקפה

בחלק השני יש את אותו עיצוב בדיוק. ההבדל היחידי הוא שעכשיו ישנה שכבת הגנה נוספת בבקר באלגוריתם של בדיקת הסיסמא הנכונה. אם באלגוריתם הקודם הסתמכנו על האלגוריתם שהבאנו לעיל, עכשיו אנחנו נוסיף delay של זמן רנדומלי, כלומר:

```
bool check_pass()
{
    for (int i = 0; i < PASS_SIZE; i++){ // this is the vulnerable part of
the code
        delay(random_time) // this is a pseudo for rand time
        if(curr_pass[i]!=password[i])
            return false;
    }
    return counter <= PASS_SIZE;
}
```

במקרה הזה נבצע מתקפה שמתבבסת על ניתוח סטטיסטי. לכן, גם בתצורה הזאת נצליח לפרוץ ולגלות את הסיסמא.

בניגוד לחלק א עכשיו יש לנו זמן רנדומלי בין כל איטרציה, ההתקפה עדיין יכולה לעבוד מכיוון שבמקום לשלוח רק 10 נסיונות נצטרך לשלוח יותר נסיונות נבדוק איזה ספרה הייתה הכי הרבה פעמים הכי הרבה זמן וכך נבחר את הספרה הבאה.. וההמשך הוא בדיוק כמו חלק א.

כמובן שאנחנו עדיין בגדר הליניאריות אך אולי לסיסמאות קצרות אולי כבר עדיף לעשות ברוטפורס.(עשינו לזה אופטימיזציה בהמשך!)

מימוש

בחלק הזה המימוש בארדואינו נשאר כמעט אותו דבר, בעוד המימוש ב ESP32 נהיה קצת יותר מסובך, נשתדל להסביר את כל החלקים של הקוד של ה ESP32.

הקורבן

המימוש בארדואינו יהיה אחד לאחד כמו בשלב הקודם, אלא שפה התווסף גם הזמן הרנדומלי בתוך הפונקציה הפגיעה.

בס"ד

התוקף

אנחנו משתמשים בפרוטוקול I2C בשביל התקשורת בין ה ESP32 לארודאינו. לאחר הרבה בדיקות הבנו שהפרוטוקול נכנס בתוך מדידת הזמן, ופוגע בה, לשם כך, חילקנו את המשימה של תקשורת ה I2C ושאר המשימות לשתי ליבות שונות שנמצאות בתוך ה ESP32.

בסה"כ, יש לנו פה 3 tasks, המשימה של ה I2C, המשימה של ה TFT לצורך התצוגה, והמשימה של המתקפה עצמה, ה TSCA.

את החלוקה לליבות הגדרנו בקובץ ה main, ובעצם כל משימה עסקה בנושא אחר, נסביר בפירוט על כל task.

```
xTaskCreatePinnedToCore(
    I2CTask::i2cTask,
    "i2cTask",
    10000,
    NULL,
    1,
    NULL,
    0
);
```

```
xTaskCreatePinnedToCore(
    TFT::TFTTask,
    "TFT",
    5000,
    NULL,
    20,
    &(TSCA::tftTaskHandle),
    1
);
```

```
xTaskCreatePinnedToCore(
    TSCA::TSCALoop,
    "TSCA",
    5000,
    NULL,
    100,
    NULL,
    1
);
```

I2C task

המשימה הזו אחראית על להקים את התקשורת של I2C וכן להתחזות לקודן, כידוע לESP32 יש 2 ליבות ומכיוון שאנחנו צריכים לתזמן מדידות של ננו שניות החלטנו שאת ניהול האינטרפטים של I2C וכל מה שקשור אליו נשים בליבה מיוחדת (core 0)

TFT task

המשימה הזו אחראית על כל הנושא של מסך הTFT והיא רצה במקביל להתקפה ומהווה חלון להתקפה עצמה.

TSCA task

המשימה העיקרית של המערכת, היא אחראית על ההתקפה עצמה ועל המדידות.

אופטימיזציה

בשלב הזה החלטנו לעשות אופטימיזציה לאלגוריתם שלנו. עד היום מה שעשינו היה זה ככה: עבור כל תו בסיסמא (נניח שגודל הסיסמא הוא N), נשלח את כל התווים האפשריים (10 ספרות), ונעשה את זה C פעמים וניקח את הממוצע.

כלומר, מספר האיטרציות שייקח לנו הוא כנ"ל: $N * C * 10$.

אולם, אם נשים לב, במידה ונישאר לנו רק תו אחד, אם ננסה לפענח אותו בברותפורס, ייקח לנו בממוצע $10/2 = 5$ ניסיונות, בעוד לפי האלגוריתם שלנו ייקח לנו $C * 5 = C / 2 * 10$ בממוצע, וברור שזה הרבה פחות יעיל... לכן, נסמן את ב x את מספר התווים שעליהם אנחנו מנסים לעשות ברותפורס, וניצור פונקציה שתלויה ב x ומחשבת את מספר האיטרציות שנידרש לעשות בממוצע.

לאחר מכן, נגזור את הפונקציה ונחפש מתי הנגזרת מתאפסת, ככה נוכל לגלות מה הערך של x שייתן לנו בממוצע את היעילות הכי טובה.

נבנה את הפונקציה:

מספר הניסיונות שנצטרך לעשות עבור x תווים בברותפורס: $10^x/2$.

מספר הניסיונות שנצטרך לעשות עבור שאר ה $N - x$ התווים האחרים: $((N - x) * C * 10)$

לכן, $f(x) = 10^x/2 + ((N - x) * C * 10)$

נגזור, ונקבל: $f'(x) = 10^x * ln10 - 10 * C$.

נשווה ל 0 על מנת למצוא נקודת מינימום: $f'(x) = (10^x * ln10)/2 - 10 * C = 0$

$$10^{x-1} = (2 * C) / ln10$$

$$x - 1 = log10((2 * C) / ln10)$$

$$x = log10((2 * C) / ln10) + 1$$

לכן, עבור הערך x הזה נקבל את הזמן הריצה הכי אופטימלי! יש לשים לב שאין פה תלות ב N, כלומר, באורך הסיסמא, אלא רק במספר האיטרציות שאנחנו עושים לכל תו.

בקוד, זה מחושב כך:

```
const uint8_t NUM_OF_BRUTEFORCE_DIGITS =
round(log10((NUM_OF_ATTEMPTS_FOR_DIGIT*2)/log(10))+1);
const uint32_t NUM_OF_BRUTEFORCE_ITERAIONS = pow(10, NUM_OF_BRUTEFORCE_DIGITS);
const uint32_t NUM_OF_ITERATIONS = NUM_OF_BRUTEFORCE_ITERAIONS + (NUM_OF_ATTEMPTS_FOR_DIGIT *
(PASS_SIZE-NUM_OF_BRUTEFORCE_DIGITS) * 10);
const float PROGRESS_INTERVAL = 100.0 / NUM_OF_ITERATIONS;
```

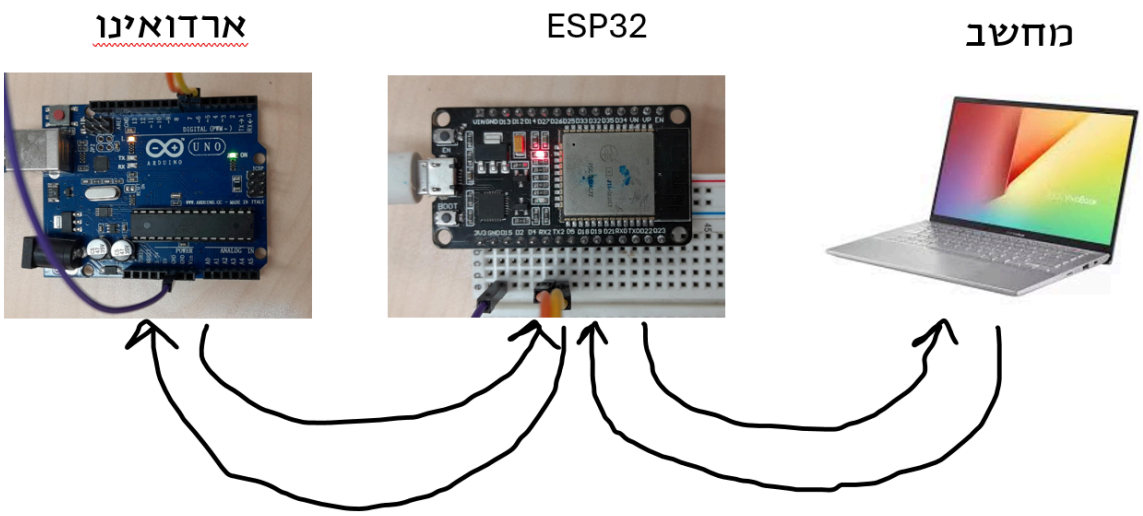
חישבנו את ה PROGRESS_INTERVAL בשביל ה TFT, כפי שהוזכר מקודם, ב TFT task.

חלק III

הקדמה

עכשיו, לאחר שראינו איך מבצעים TSCA על חלק 1 וחלק 2, וחשנו מה זה TSCA, נראה איך מבצעים אותה על מימוש של ECC.

בקצרה, מה שנעשה זה לממש מתקפה על בקר שחותם על הודעות בשיטת ECDSA.
הבקר יהיה ארדואינו, והתוקף יהיה ה ESP32. במתקפה הזאת אנחנו נשלח ל ESP32 הודעה כלשהיא מהמחשב דרך ה serial, ומספר כלשהו של חתימות אותן אנחנו רוצים לאסוף מהבקר. ה ESP32 ישלח את ההודעה הזאת לארדואינו בשביל שהוא יחתום עליה, וימדוד את הזמן שלקח לחתום על ההודעה, וכן ייקח את החתימה שהבקר ייצר.
באופן הזה ה ESP32 יעשה כמספר הפעמים שהמחשב ביקש, ולבסוף יישלח לו את הכל, את החתימות + הזמן שלקח להן. המחשב יירשום את הכל לתוך קובץ אחד גדול, ובעזרת קוד שהוא מריץ יצליח לעשות ניתוח כלשהוא ולגלות את המפתח הפרטי של הבקר שחתם על הסיסמאות.



האלגוריתם הפגיע שבו השתמשנו נקרא Montgomery multiplication, זה בעצם אלגוריתם יעיל לחישוב מכפלות, שעובר ביט ביט ובהתאם לערך בודק מה צריך לעשות.

```
// Function that implements montgomery algorithm
Point EllipticCurve::scalarMultiplication(Point P, uint32_t k){
    Point R0 = P;
    byte l = key_length(k);
    for(int8_t j = l-2; j >= 0; j--)
    {
        R0 = doublingPoint(R0);
        if(bitRead(k, j)){ // if k[j] == 1
            R0 = addPoint(R0, P);
        }
        delay(5);
    }
    return R0;
}
```

כמו כן, השתמשנו ב minerva attack, על מנת לבצע את המתקפה על החתימות שקיבלנו עם הזמן, ולגלות את המפתח הפרטי.

עכשיו, לאחר הטיזר שניתן והסבר כללי, נתחיל ללמוד את התיאוריה שעומדת מאחורי המתקפה. לשם כך, נסביר נושא נושא, ובעז"ה בסוף נתפור את הכל למתקפה אחת גדולה.

תיאוריה

קריפטוגרפיה בעקומות אליפטיות - ECC

מהי עקומה אליפטית? עקומה אליפטית היא קבוצה של נקודות במישור שמקיימות את המשוואה:

$$y = x^3 + ax + b$$

כאשר a ו-b הם קבועים שמתאימים לקריפטוגרפיה. בעקומות אלה ישנן תכונות מתמטיות מיוחדות שמאפשרות שימוש בטכניקות קריפטוגרפיות יעילות.

הפעולות הבסיסיות:

- **חיבור נקודות (Point Addition)** - חיבור של שתי נקודות על העקומה נותן נקודה נוספת על העקומה.
- **כפל סקלרי (Scalar Multiplication)** - כפל נקודה במספר סקלרי (שלם) הוא חיבור חוזר של הנקודה עם עצמה.

הצפנה ואבטחת מידע: באמצעות כפל סקלרי ניתן ליצור מערכת מפתחות פרטיים וציבוריים. אם נבחר מפתח פרטי d שהוא מספר שלם, ניתן לחשב את המפתח הציבורי על ידי כפל הנקודה הבסיסית G (נקודה קבועה על

העקומה) במפתח הפרטי: $dG = Q$. החישוב של d מ-Q הוא קשה מאוד מבחינה מתמטית, מה שמבטיח את הבטיחות של המערכת.

אלגוריתם חתימה דיגיטלית בעקומות אליפטיות - ECDSA

שלבי האלגוריתם:

1. **יצירת חתימה:**
   - בוחרים מפתח פרטי d.
   - בוחרים מספר אקראי k עבור כל חתימה.
   - מחשבים נקודה $kG = P$ וממנה את r, שהיא הקואורדינטה ה-x של P.
   - מחשבים s באמצעות המשוואה $s = k^{-1}(z + rd) \bmod n$, כאשר z הוא המסר המיועד לחתימה ו-n הוא סדר הנקודה הבסיסית G.
2. **אימות חתימה:**
   - מחשבים $u_1 = zs^{-1} \bmod n$ ו- $u_2 = rs^{-1} \bmod n$.
   - מחשבים את הנקודה $P = u_1G = u_2Q$.
   - אם r שווה לקואורדינטה ה-x של P, החתימה מאומתת.

מה כוללת החתימה: החתימה ב-ECDSA כוללת שני מספרים: r ו-s. שניהם נחוצים לאימות החתימה.

על מנת שנוכל להבין איך אפשר להתקיף דבר שכזה, נתחיל בלהגדיר כמה הגדרות ובעיות.

אנוטציות

נגדיר לפעמים את y mod q בתור $\lfloor y \rfloor_q$ בשביל הנוחות וכן בגלל שזה הנוטציה שהשתמשו בה במאמר המקורי.
וכן נגדיר את פונקצית מרחק הבא:

$$|y|_q := \min_{a \in \mathbb{Z}} |y - aq|$$

שמחזירה את המרחק בין y לבין המכפלה הכי קרובה של q.

בעיית ה HNP

בעיית ה Hidden Number מוגדרת כך:
בהינתן d אי שוויונות מהסוג הבא:

$$|k_i - a_i|_q < q/2^{l_i} \quad , \quad k_i = \lfloor \alpha t_i - u_i \rfloor_q$$

בס"ד

כאשר ידועים האיברים t_i, q, u_i, l_i, a_i בהתאמה לכל משוואה i מ 1 עד d

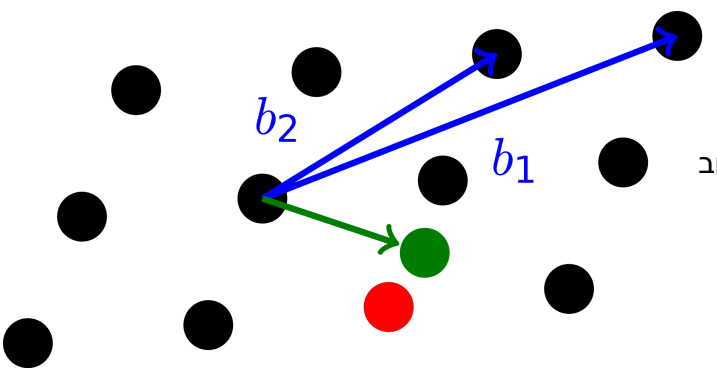
מצא את המספר הנעלם α.

מכיוון שבבעיה שלנו אנחנו מניחים שa הוא 0 יוצא לנו שהבעייה מצטמצמת ל:

$$|\alpha t_i - u_i|_q < q/2^{l_i}$$

בעיית ה CVP

בעיית הClosest vector מוגדרת כך:
בהינתן מטריצה כך שהשורות שלה הן בסיס B וכן ווקטור u, רוצים למצוא את הווקטור v (שכל איבריו מספרים שלמים) כך שהכפלת כל איבר בv בכל שורה של B וסכימתן תיתן את הווקטור הכי קרוב לu.



או במילים אחרות בהינתן אין סוף נקודות במרחב L_v שמוגדרות על ידי הווקטור **השלם** v וכן על ידי המטריצה B

$$b_1 * v_1 + b_2 * v_2 + ... + b_n * v_n = L_v$$

כך שb_i הוא השורה הi של המטריצה B וכן v_i הוא האיבר הi של v (וכמובן שv_i הוא מספר שלם (שלילי או חיובי))

מצא את הווקטור השלם v_{min} כך ש $L_{v_{min}}$ הוא הווקטור הכי קרוב לווקטור u.

בתמונה מוויקיפדיה אנו רואים את הנקודות השחורות (L_v) וכן הווקטורים הכחולים שהם השורות של B, הווקטור u בירוק והתשובה של הבעייה באדום ($L_{v_{min}}$).

אנחנו יודעים ומניחים שיש דרך סטנדרטית לפתור את הבעייה הזו.

המתקפה

המתקפה עובדת ככה.

שלב א

נשלח N הודעות[2] אל החותם, נקבל את החתימות בחזרה על כל הודעה והודעה, ונמדוד את הזמן שלקח לו לחתום עליהן. את התוצאה של השלב הזה נשמור בקובץ אחד שעליו נבצע עיבוד בהמשך.

[2] בפרקטיקה אנחנו שולחים את אותה ההודעה בגלל שזה לא באמת משנה בגלל הK הרנדומלי אנחנו עדיין נקבל מספיק חתימות.

שלב ב

נמיין את כל החתימות לפי זמני החתימה שלהן, וניקח את d החתימות המהירות ביותר.

שלב ג

נעשה פה הפסקה קצרה, נפתח קצת משוואות ותיאוריה, ואחרי זה נמשיך.
תזכורת:

- מחשבים נקודה $kG = P$ וממנה את r, שהיא הקואורדינטה ה-x של P.
- מחשבים s באמצעות המשוואה $s = k^{-1}(z + rd) \bmod n$, כאשר z הוא המסר המיועד לחתימה ו-n הוא סדר הנקודה הבסיסית G.

מה כוללת החתימה: החתימה ב-ECDSA כוללת שני מספרים: r ו-s. שניהם נחוצים לאימות החתימה.

נסמן את ה nounce ב k, ולאחר פיתוח מהמשוואה שראינו למעלה, נקבל: $k = s^{-1}(xr + z) \bmod n$

כיוון שלקחנו את d החתימות הקצרות, אנחנו מניחים ש 3 הביטים הראשונים, ה MSB, הם 0.

ולכן, מתקיים האי שוויון שבתמונה, כיוון ש n זה המספר של המודולו, ואנחנו יודעים ש l הביטים הראשונים הם 0, אזיי חלוקה של n ב 2^l ממילא תהיה יותר גדולה מ k.

$$\lfloor k_i \rfloor_n < n/2^{l_i}$$

לאחר מכן, נציב את k כפי שראינו לעיל:

$$|s_i^{-1}(xr_i + H(m_i))|_n < n/2^{l_i}$$

לבסוף, נגיע למשוואה הסופית על ידי כך שנפרק את הסוגריים.

$$|x\lfloor s_i^{-1} r_i \rfloor_n + \lfloor s_i^{-1} H(m_i) \rfloor_n|_n < n/2^{l_i}$$

אם ככה, על ידי הסימונים הבאים: $q = n, \alpha = x,\ t_i = \lfloor s_i^{-1} r_i \rfloor_n \text{ and } u_i = \lfloor -s_i^{-1} H(m_i) \rfloor_n$

נגיע למשוואה של HNP!

אם ככה, כל מה שנשאר לנו זה לפתור את בעיית ה HNP שקיבלנו, ואז אנחנו אמורים לקבל את ה private key, את ה α הסודי.

לכן, מה שנעשה עכשיו זה להרכיב את ה lattice שפותר את בעית CVP:

מבנה המשוואה הוא ככה:

$$|\alpha t_i - u_i|_n < n/2^{l_i},$$

נבנה את ה lattice הבא:

$$B = \begin{pmatrix} 2^{l_1}n & 0 & 0 & \dots & 0 & 0 \\ 0 & 2^{l_2}n & 0 & \dots & 0 & 0 \\ & \vdots & & & \vdots & \\ 0 & 0 & 0 & \dots & 2^{l_d}n & 0 \\ 2^{l_1}t_1 & 2^{l_2}t_2 & 2^{l_3}t_3 & \dots & 2^{l_d}t_d & 1 \end{pmatrix}$$

הווקטור u שהוא הווקטור מהבעיה של ה CVP יהיה:

$$\mathbf{u} = (2^{l_1}u_1, \ldots, 2^{l_d}u_d, 0)$$

והנקודה הכי קרובה תהיה מהצורה:

$$\mathbf{v} = (2^{l_1}t_1\alpha, \ldots, 2^{l_d}t_d\alpha, \alpha),$$

לכן, נעשה רדוקציה לבעית SVP בצורה הבאה:

$$C = \begin{pmatrix} B & 0 \\ \mathbf{u} & n \end{pmatrix}$$

וזהו. נפתור את הבעיה ונמצא את α מתוך v.

חשוב להדגיש... הצלחנו למצוא את המפתח רק בגלל שקיבלנו מספיק מידע מהחתימות!
אם לא היינו יודעים מה הביטים הכי חשובים של k ,ההנחות שלנו לא תקפות וזה בדיוק כמו לנחש מפתחות.

מימוש

את חלק 3 מימשנו בצורה הבאה: יש מיקרו בקר שחותם על hash כל שהוא שנותנים לו ומחזיר את החתימה דרך serial.
אנחנו מחברים לו את הESP32 בשביל לתזמן את החתימות, הESP שולח מספר חתימות ושולח את המידע למחשב. המחשב מקבל את המידע ויוצר קובץ מידע מתאים, ואז ניתן לקחת את הקובץ ולהריץ עליו את הסקריפט פייתון שממש את ההתקפה.

הקורבן

המיקרו בקר שחותם על ההודעות הוא Arduino Uno , מימשנו בC++ אלגוריתם פשטני של חישובים של הכפלה וחיבור של נקודות בשביל לממש את הסרבר וכן את היכולת לחתום, לבדוק ולקבל את המפתח הציבורי מהארדואינו.
בשביל מידע נוסף על פקודות לארדואינו ניתן לראות את הקוד המצורף.

התוקף

התוקף הוא הESP32 שמחובר לארדואינו ומהווה חוצץ בין המחשב לבין הארדואינו. נותנים לESP פקודה של כמה חתימות לתזמן, והESP מחזיר את החתימות על מסר קבוע של 1234 כיוון שזה לא משנה וכן את הזמן שלקח לכל חתימה. המחשב מקבל את החתימות ומעבד אותן לקובץ מידע. כמובן המחשב מבקש מהESP את המפתח הציבורי על מנת שנוכל לוודא שהמפתח שמצאנו בהתקפה הוא באמת המפתח על ידי זה שהמחשב עצמו יצור מפתח ציבורי על ידי המפתח הפרטי שמצא ואז ישווה בין השניים.

```
public key: 635 171
r | s | time[ms]
409 863 46
977 879 14
821 121 53
454 1038 52
56 316 35
287 557 53
57 959 52
196 555 41
524 459 48
391 491 52
986 618 53
329 4 47
916 99 52
133 901 53
337 387 47
519 273 41
```

עיבוד הנתונים

לאחר שהשגנו את קבצי הנתונים אנחנו יכולים להתחיל במתקפה, אנחנו ממיינים את החתימות לפי הזמנים מוצאים כפילויות ולוקחים d כלשהוא של החתימות הכי מהירות.

לפי התיאוריה אנחנו מניחים שה3 MSB של K הם 0 אבל לפעמים צריך לשנות אותו מכיוון שזה לא תמיד המצב[3].

אחרי זה אנחנו יצרנו את המטריצה של CVP, החלטנו לפתור את הבעייה על ידי המרה לבעייה אחרת (SVP) ולהריץ אלגוריתם ידוע של LLL reduction.

ההמרה ל SVP היא לא קריטית ויכולנו לפתור את CVP אפילו באיטרציה בלולאה או גרדיאנט דיסנט, אך החלטנו לא להמציא את הגלגל ולהשתמש בקוד מוכן.

```
def build_svp_lattice(signatures, curve):
    """Build the basis matrix for the SVP lattice using `signatures`."""
    dim = len(signatures)
    B = IntegerMatrix(dim + 2, dim + 2)  # the SVP matrix
    n = curve.calc_order()  # curve order

    for i in range(dim):
        li = 3# geom_bound(i, dim) + 1  # Assume number of MSB bits with zero
        ri = signatures[i][0][0]  # ri
        si = signatures[i][0][1]  # si
        ti = (curve.modularInverse(si, n) * ri) % n  # ti
        ui = -((curve.modularInverse(si, n) * hash_to_change) % n)  # ui

        B[i, i] = (2 ** li) * n  # (2^li)*n
        B[dim, i] = (2 ** li) * ti  # (2^li) * ti
        B[dim + 1, i] = (2 ** li) * ui + n  # (2^li) * ui + n

    B[dim, dim] = 1
    B[dim + 1, dim + 1] = n
    return B
```

בתכלס אנחנו ישר בנינו את המטריצה של SVP שהרדוקציה היא:

$$C = \begin{pmatrix} B & 0 \\ \mathbf{u} & n \end{pmatrix}$$

כאשר B זה המטריצה של CVP ו u זה הווקטור שרוצים למצוא את הווקטור הכי קרוב אליו.
לאחר שבנינו את המטריצה אנחנו מריצים את LLL ואז מקבלים את ההמועמדים לפיתרון בעמודה השנייה לפני הסוף.

בודקים אם אחד מהמפתחות המועמדים זה המפתח הנכון על ידי יצירה של המפתח הציבורי והשוואה. אם כן פרצנו! אם לא אז כנראה שצריך לשנות את ההנחות שלנו או את d.
אם גם זה לא עבד אז המתקפה שלנו פשוט נכשלת... (מה לעשות)

[3] קיימת במאמר המקורי שיטה של אקסלרציה שמחשבת על ידי חישוב סטטיסטי את מספר הביטים המשמעותיים שהם 0 שמחליפים את המספר השרירותי 3. אך זה רק אקסלרציה ולא משפיע על הנכונות של ההתקפה.

הקשר אבטחה

בפרויקט זה אנו שואפים לטפל בפגיעות של אלגוריתמים קריפטוגרפיים למתקפות תזמון צד ערוץ (TSCA). ההקשר האבטחתי מוגדר על ידי האלמנטים הבאים:

מודל איומים

הפרויקט שלנו נועד בעיקר להגן מפני מתקפות תזמון שבהן התוקף מודד את הזמן שנדרש לביצוע פעולות קריפטוגרפיות כדי להפיק מידע סודי. במיוחד אנו מתמקדים ב:

- **מתקפות תזמון על אימות סיסמאות:** הגנה מפני מתקפות שמסיקות את הסיסמה הנכונה בהתבסס על הזמן שנדרש לאמת כל תו.
- **מתקפות תזמון על חתימות ECC:** הגנה מפני מתקפות על פעולות חתימה (ECDSA) מבוססות ECC על ידי מדידת זמן יצירת החתימה.

הנחות אבטחה

אנו מניחים את הדברים הבאים לגבי הסביבה והמשתמשים:

- **סביבת הפעלה בטוחה:** החומרה ומערכת ההפעלה בטוחות וללא פגיעויות.
- **גישה מבוקרת:** רק אנשים מורשים יכולים לגשת למערכת, והמערכת מוגנת מפני גישה פיזית לא מורשית.
- **התנהגות משתמשים:** המשתמשים מקפידים על נהלי אבטחה, כולל שימוש בסיסמאות חזקות וייחודיות ואי שיתוף מידע רגיש.

היקף ומגבלות

מה מכוסה:

- **אימות סיסמאות:** אנו משתמשים בשיטת אימות נאיבית, תו אחר תו, ומודדים את הזמן שנדרש להגיב לסיסמאות שונות. על ידי ניתוח זמני תגובה אלה, נוכל להסיק את הסיסמה הנכונה.
- **הפחתת מתקפות תזמון באמצעות השהיות רנדומליות:** הוספת השהיות רנדומליות בין בדיקות תווים כדי לשבש את דפוסי התזמון, וביצוע ניתוח סטטיסטי כדי לבצע עדיין TSCA.
- **אבטחת חתימות ECC:** מימוש שרת חתימות (ECDSA) מבוסס ECC והדגמה כיצד ניתן להשתמש ב-TSCA כדי למצוא את המפתח הסודי.

מה לא מכוסה:

- **מתקפות פיזיות:** הפרויקט שלנו לא עוסק במתקפות פיזיות שבהן לתוקף יש גישה ישירה לחומרה.
- **קריפטואנליזה מתקדמת:** בעוד אנו מתמודדים עם מתקפות תזמון, מתקפות קריפטוגרפיות מתקדמות יותר, כגון ניתוח הספק דיפרנציאלי, הן מחוץ להיקף הפרויקט.

מדריך משתמש

Part1

בתור התחלה, נעתיק את ה remote repo מה github:

```
git clone https://github.com/avishaigonen123/Time-side-channel-attack
```

ניכנס לתיקייה:

```
cd '.\Time side channel attack\'
```

לאחר מכן, ננווט לתיקייה של השלב המתאים:

```
cd Part1
```

בתייקיה הזו ישנן 2 תיקיות, תיקיה לארדואינו, שזה הקוד ה c שהמשתמש יצטרך לקמפל ולטעון על הארדואינו, ותיקיית ESP32, שזה הקוד c שהמשתמש יצטרך ולקמפל ולטעון על ה ESP32.

כמו כן, חשוב לחבר את החוטים בין הארדואינו לפי הצבעים! (כחול לכחול, אדום לאדום וכו.)

אם נרצה לשנות את הסיסמא, נצטרך ללכת אל הקוד שנמצא בקובץ main.cpp, שנמצא בתוך תיקיית src ב Arduino. שמה נצטרך לשנות את ה PASS_SIZE בשורה 9, וכן לשנות את ה password בשורה 21.

כמו כן, נצטרך ללכת אל הקוד שנמצא בקובץ TimeAttack.h בתיקיית src ב ESP, ושמה בשורה 8 לשנות את ה PASS_SIZE.

Part2

בשלב 2 זה בדיוק אותו הדבר, ההבדל היחידי הוא שנכנסים לתיקיית Part2 ולא ל Part1.

גם ההסברים על שינוי הסיסמא הם אותו הדבר.

Part3

בשלב הזה ישנה ספרייה שבה משתמש ה script ב python, ה fpylll, שזמינה רק ב linux. לכן, יש לפתוח linux, ניתן פשוט לפתוח wsl.
לאחר מכן, נעשה clone וניכנס לתיקייה:

```
git clone https://github.com/avishaigonen123/Time-side-channel-attack
cd '.\Time side channel attack\Part3\'
```

נטעין את הקוד של הארדואינו על הארדואינו שלנו, ניכנס לתיקיית arduino, וכן לתיקיית ESP32, ונטעין בהתאם.
לאחר מכן, נחבר את החוטים בהתאם לטבלה:

Arduino uno	ESP32
Pin 6	Tx2
Pin 7	Rx2
GND	GND

עכשיו נתקין ספריות נחוצות באמצעות pip install למען הרצת המערכת:

```
pip install fpylll
```

במידה ויש דרישה לעוד ספריות, יש להתקין אותן.

לאחר שסיימנו להתקין את כל הספריות הנחוצות, ניכנס לתיקיית Attack:

```
cd Attack
```

ושמה, ניכנס לקובץ data_generator.py.
נוכל לשנות את מספר החתימות שנשלחות לקורבן על ידי שינוי הערך של N, בשורה 10.
נריץ את ה script, חשוב לוודא שה port בשורה 75 תואם ל port של ה serial במחשב, ל ESP32.
בסיום השלב הזה אמור להיווצר קובץ בתיקיית data, שמכיל N חתימות + הזמן שלהן, וכן את ה public_key בראש הקובץ.

עכשיו ניכנס לקובץ my_lattice_attack.py, ושמה נשנה את הפרמטרים של האליפטה בהתאם למה שאיתו עבדנו, בשורות 11-14. כמו כן, נצטרך לשנות את ה hash של ההודעה בשורה 18, במידה והוא השתנה.

לבסוף, נריץ את הקובץ, ניתן לו את המסלול לקובץ ה data שייצרנו, וזהו, המתקפה עובדת.
לדוג:

```
agonen@MyPC:~/Time-side-channel-attack/Part3$ /bin/python3
/home/agonen/Time-side-channel-attack/Part3/Attack/my_lattice_attack.py
give me path to your data file:
/home/agonen/Time-side-channel-attack/Part3/Attack/data/new_test_2_1.txt
d = 20
Private key recovered: 18
```

עבודות עתידיות ושיפורים

שיפור לחלקים I & II

לאחר שראינו שהמתקפה הסטטיסטית שלנו דורשת לבדוק כמה וכמה נסיונות לספרה, ישנה ההסתברות שניחשנו לא נכון, ויוצא שאנחנו לא נדע את זה עד שננסה את כל הספרות והדבר לוקח זמן.
אנחנו יכולים לשפר את ההתקפה על ידי זה שאם ההתפלגות של הנסיונות היא "שטוחה" כלומר נראה שיש הסתברות שווה לכל ספרה אז ניתן להסיק שטעינו בניחוש הקודם שלנו ולנסות שוב ולחסוך בזמן.

שיפור לחלק III

כפי שכבר ראינו, כל הבסיס של המתקפה יושב על זה שניתן להדליף מידע על ה nounce. במידה ופירצת המידע הזו תיסתם, אזי גם כל המתקפה תיפול.

בשלב זה נבחן שיפורים אפשריים שניתן לבצע כדי לחזק את האבטחה ולהתגבר על החולשות שהתגלו. אלו כמה מהשיפורים שנציע:

נוסחאות שלמות (Complete Formulas)

אחת מהסיבות העיקריות לחולשה היא השימוש בנוסחאות תוספת לא שלמות, שגורמות להבדלים נמדדים בזמן הריצה. נוסחאות שלמות מתנהגות באותו אופן לכל הקלטים ומונעות דליפות מידע. נוסחאות אלו איטיות יותר, אך הן מפשטות את האלגוריתם ומונעות דליפות מידע.

שיפור: להשתמש בנוסחאות שלמות במקום הנוסחאות החלקיות, ולהתחיל את לולאת הכפל הסקלרי מביט קבוע מבלי לחשב את אורך הסקלר באופן מפורש.

תיקון אורך הביטים (Fixing the Bit-Length)

גם בשימוש בנוסחאות חלקיות, אפשר למנוע את החולשה על ידי תיקון אורך הביטים של ה-nonces כך שיהיו באותו אורך.

אחת השיטות היא להוסיף כפולה מתאימה של סדר הקבוצה ל-nonce כך שהלולאה תרוץ תמיד אותו מספר פעמים.

$$\hat{k} = \begin{cases} k + 2n & \text{if } \lceil \log_2(k+n) \rceil = \lceil \log_2 n \rceil \\ k + n & \text{otherwise.} \end{cases}$$

במצב הזה, האורך של k יהיה תמיד $log_2 n + 1$, ואז תמיד נרוץ אותו זמן.

שיפור: להוסיף את הערך המתאים לסקלר כדי לוודא שהלולאה תרוץ תמיד אותו מספר פעמים, ובכך למנוע דליפת מידע דרך זמני הריצה.

נוננסים ארוכים (Long Nonces)

הגדלת אורך ה-nonces ושימוש בהם ללא הפחתה מודולו סדר הקבוצה יכולים להקשות על התקפות ערוץ צד.

שיפור: להשתמש ב-nonces ארוכים בהרבה ולבצע את הכפל הסקלרי עליהם מבלי להפחית מודולו סדר הקבוצה.

שיפורים אלה יוכלו לסייע בשיפור האבטחה של המערכת ולמנוע דליפות מידע באמצעות התקפות צד ערוץ.

סיכום

בפרויקט זה הצגנו את עולם מתקפות ה-Side Channel Attack, ובפרט את מתקפת ה-Time Side Channel Attack (TSCA). התמקדנו ביישום מתקפות אלו על רכיבי embedded, והראינו כיצד הן פועלות בפרקטיקה.

בשלב הראשון, ביצענו מתקפה בסיסית על אלגוריתם שבודק סיסמאות, והוכחנו שניתן לפרוץ אותו באמצעות TSCA. בשלב השני, הוספנו זמני רנדומלי לאלגוריתם על מנת להקשות על המתקפה, והראינו שבאמצעות ניתוח סטטיסטי ניתן לשבור גם את האלגוריתם המוגן.

בשלב האחרון, עברנו ליישום מתקפה על ECDSA, בה הצגנו את עקרונות ההצפנה והחתימה, ומימשנו בקר שחותם על הודעות. באמצעות דליפת מידע מהחתימה, הצלחנו להמיר את הבעיה ל-HNP, CVP, ו-SVP, ומצאנו את הסיסמא הפרטית של הבקר, מה שהוכיח את הפגיעות של האלגוריתם.

לסיום, הצגנו מספר דרכים להגנה על האלגוריתמים הפגיעים, כולל שימוש בנוסחאות שלימות, תיקון אורך הביטים, רנדומיזציה של הסקלר והגדלת אורך הנונסים. דרכים אלו מסייעות להקשות על תוקפים לנצל את חולשות ה-TSCA.

פרויקט זה הדגים את הסכנות הטמונות במתקפות TSCA והציג דרכים להתמודד עמן. המתקפות וההגנות שהצגנו מדגישות את החשיבות בהבנת האיומים והצורך בשיפור מתמיד של מנגנוני האבטחה.

מקורות

Timing attack - Wikipedia
Elliptic-curve cryptography - Wikipedia
Elliptic Curve Digital Signature Algorithm - Wikipedia
Montgomery modular multiplication - Wikipedia
Lattice problem - Wikipedia

Minerva
Minerva: The curse of ECDSA nonces
GitHub - crocs-muni/minerva: Artifacts for the "Minerva: The curse of ECDSA nonces" paper at CHES 2020

Elliptic Curve scalar multiplication ($\mathbb{F}_{\square}$)
https://chatgpt.com/
https://claude.ai/