

Time side-channel attack on embedded chip

מאת:

אבי פדר

&&

דניאל יוחנן

יוני 2021

סמסטר ב שנה ג

הפקולטה למדעי המחשב

המרכז האקדמי לב, ירושלים

המרצה: אריה שלמה הנל

| | |
|---------|--|
| 3..... | Summery |
| 4..... | Side-Channel Attack - הקדמה |
| 5..... | Arduino לוח |
| 6..... | לדים (נורות) |
| 8..... | כפתורים |
| 10..... | State Change Detection |
| 12..... | Simple Timing attack - חלק ראשון |
| 16..... | בניית לוח הפריצה |
| 21..... | Timing attack against simple random time-delay insertion - חלק שני |
| 23..... | Basic timing attack demonstration on RSA - חלק שלישי |
| 23..... | RSA מהו |
| 24..... | Repeated Squaring |
| 28..... | הרעיון העומד מאחורי המתקפה |
| 29..... | תשתית למתקפה |
| 30..... | תהליך המתקפה |
| 31..... | בניית מאגר ערכים למתקפה |
| 33..... | ניתוח המאגר |
| 37..... | דוגמת הרצה |
| 39..... | ratio קביעת הערך |
| 42..... | סיכום |
| 43..... | מקורות |

Summery

We were required to prepare a final project as part of our end of course software security project. The theme of the project was based on a choice from a selection of projects.

The project we chose to use is time side-channel attack on an embedded chip. That is, a side channel attack is based on Arduino and Tiva-boards.

We chose this project even though we had no prior knowledge with these particular boards and had never experienced programming these boards and building electronic circuits.

The project deals with a side-channel attack .

To do this is we took advantage of the fact that it is possible to measure both 'external' and 'side' variables in computer system activity (measuring the output of an environmental computer system) and applied an appropriate computational model to those variables, to obtain value information about the state of the computer system.

More specifically we will focus on a timing attack. This relies on the fact that the completion of a computational operation requires a measurable time frame. An attacker who is aware of these parameters, such as the encryption algorithm and the CPU architecture can be used to construct a theoretical model for the calculation process, and compare the model to the time frame that was measured and therefore finding the source input. This is the process that we are going to follow.

הקדמה - Side-Channel Attack

מערכות מחשוב פועלות באינטראקציה הדדית מול הסביבה החיצונית. לדוגמא, מערכת המחשוב עשויה לפלוט רעש לסביבה החיצונית מרכיבים מכניים ו/או שמע המובנים במערכת המחשוב. דוגמה נוספת היא צריכת החשמל של המערכת, ברגעים שונים ובזמן ביצוע פעולות שונות, צריכת החשמל של המערכת גדלה או קטנה.

הוכח בעבר כי מדידה של אותם משתנים חיצוניים הנפלטים לסביבה, יכולה להוביל להסקת מסקנות ואבחון של מצב מערכת המחשוב ברגע נתון.

לשם הפשטה של עקרון זה, ניתן להשתמש בדוגמה מעולם התחבורה. על מנת לדעת אם יש 'פקק' בכביש, ניתן לבחון במישרין את מצב הכביש, ע"י התבוננות ישירה בכביש.

לעומת זאת, ניתן לדעת בצורה עקיפה (אומנם לא באופן וודאי אבל אכן בצורה מספקת) האם יש פקק בכביש, על ידי האזנה לכביש. כאשר ניתן להניח כי אם שומעים רעשי צפירות רבים, ככל הנראה מכוניות 'עומדות' בכביש ולא נוסעות.

על בסיס עקרון זה מבוססת התקפת ערוץ צד (Attack Channel-Side). כלומר אנו ננצל את העובדה כי ניתן למדוד משתנים "חיצוניים" / "צדדיים" בפעילות מערכת מחשוב (מדידת הפלט של מערכת מחשוב לסביבה) ולהחיל מודל חישובי מתאים על אותם משתנים, כדי לקבל מידע לגבי המצב של המערכת.

באופן יותר ספציפי נתמקד במתקפת מידע תזמון. שמסתמכת על כך שלהשלמת פעולה חישובית נדרשת מסגרת זמן שניתנת למדידה. תוקף אשר מודע לפרמטרים, כדוגמת אלגוריתם ההצפנה וארכיטקטורת המעבד אשר נעשה בהם שימוש, יכול לבנות מודל תיאורטי לתהליך החישוב, ולהשוות את המודל למסגרת הזמן אשר נמדדה, ובכך למצוא את קלט המקור. וזה בדיוק מה שנעשה.

את כל הקוד ניתן למצוא [בגיט](#).

לוח Arduino

בשלב הראשון של הפרויקט התחלנו בלמידה על לוח Arduino וכיצד מחברים רכיבים אלקטרוניים ללוח כזה.

מבנה הקוד (בשפת cpp) בלוח Arduino מחולק לשלושה חלקים:

1. הצהרות מוקדמות (לא חובה - לשם הנוחות בלבד).
2. פונקציה בשם setup שרצה פעם אחת כאשר המערכת עולה.
3. פונקציה בשם loop שמתבצעת שוב ושוב כאשר המערכת עובדת.

מבנה הקו נראה כמו בתמונה הבאה:

```
#Some defines

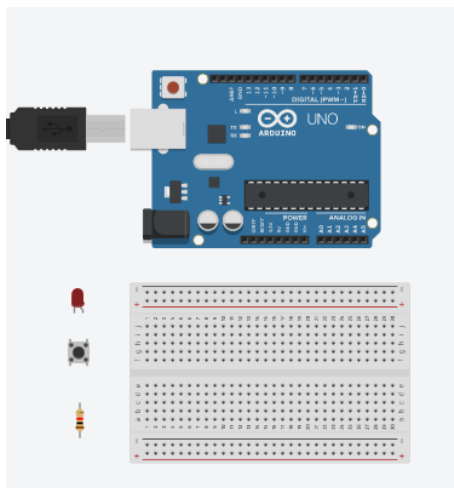
void setup() {
    // put your setup code here, to run once:

}

void loop() {
    // put your main code here, to run repeatedly:

}
```

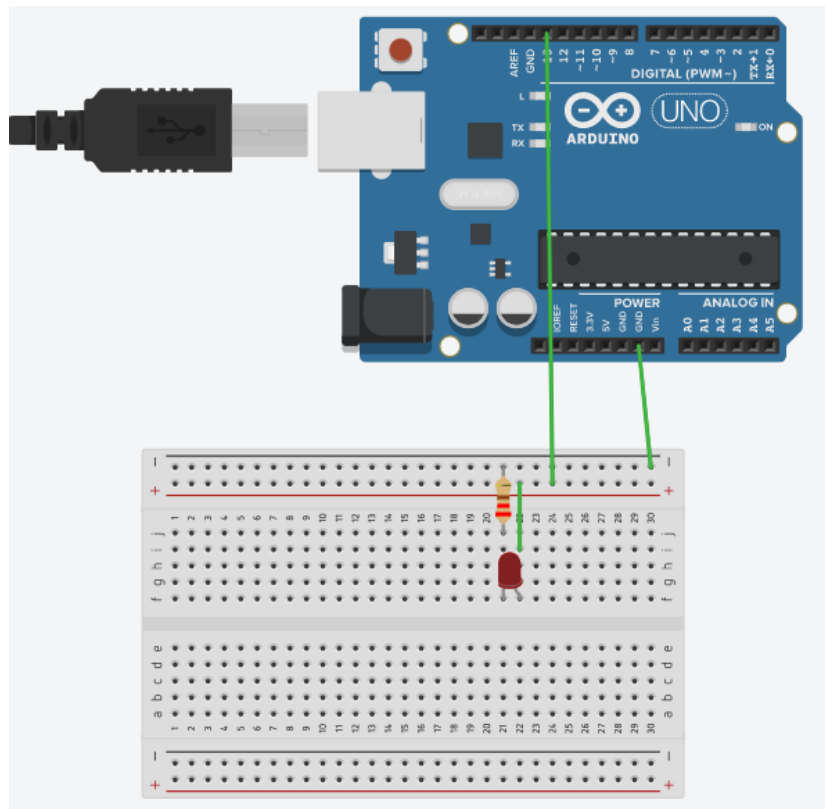
במשך כל בניית המערכת, השתמשנו באתר [הבא](#), שמאפשר לבנות סימולציה של מערכות שונות. ככה התנסנו בשימוש בלוח בצורה נוחה יותר ובלי סיכון של המערכת (טעויות בחיבורים או ברכיבים שונים, כמו נגדים, יכולות לגרום ללוח להישרף).



(צילום מתוך הסימולציה)

לדים (נורות)

באמצעות המדריך [הבא](#) למדנו כיצד לחבר לד (נורה) למערכת. וקיבלנו את הדבר הבא:



הקוד שלנו יראה ככה:

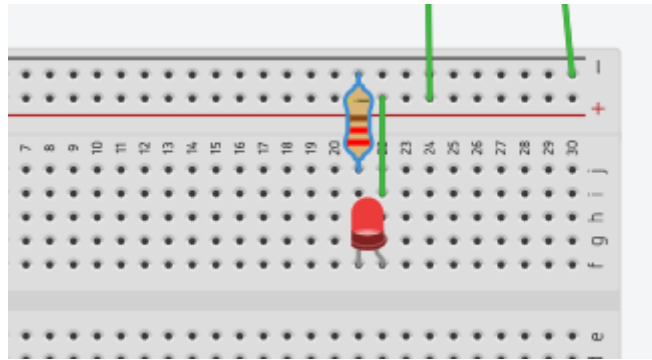
```
1 int led = 13; // the pin the LED is connected to
2
3 void setup() {
4   pinMode(led, OUTPUT) // Declare the LED as an output
5 }
6
7 void loop() {
8   digitalWrite(led, HIGH) // Turn the LED on
9 }
10
```

בשורה 1 אנחנו נצהיר ונקרא לפין מספר 13 בשם led (לפין הזה נחבר את הled שלנו).

בשורות 3-5 יש את פונקציית setup שלנו. היא רצה פעם אחת כאשר המערכת עולה ומגדירה את פין 13 כיציאה (כלומר הוא יוציא מתח וככה נדליק את הled).


בשורות 7-9 יש פונקציית loop שלנו, היא רצה שוב ושוב כאשר המערכת עובדת והיא מורה למערכת להזרים מתח (HIGH) דרך פין 13. כלומר הled יידלק.

כאשר נפעיל את הלוח, הLED האדום ידלק, כפי שניתן לראות בסימולציה:



כמוכן שכאשר אנחנו מחברים רכיבים לערכת, אנחנו צריכים להוסיף נגדים כדי לא לשרוף את הלוח או את הרכיבים. מכיוון שלכל נגד יש התנגדות שונה, שאותה ניתן לזהות באמצעות הפסים הצבעוניים שעל הנגד, נשתמש באתר [הבא](#) ע"מ לבדוק לפי הצבעים של הנגדים מהם הנגדים שברשותנו. ולאתר את הנגד הנכון שעלינו לחבר.

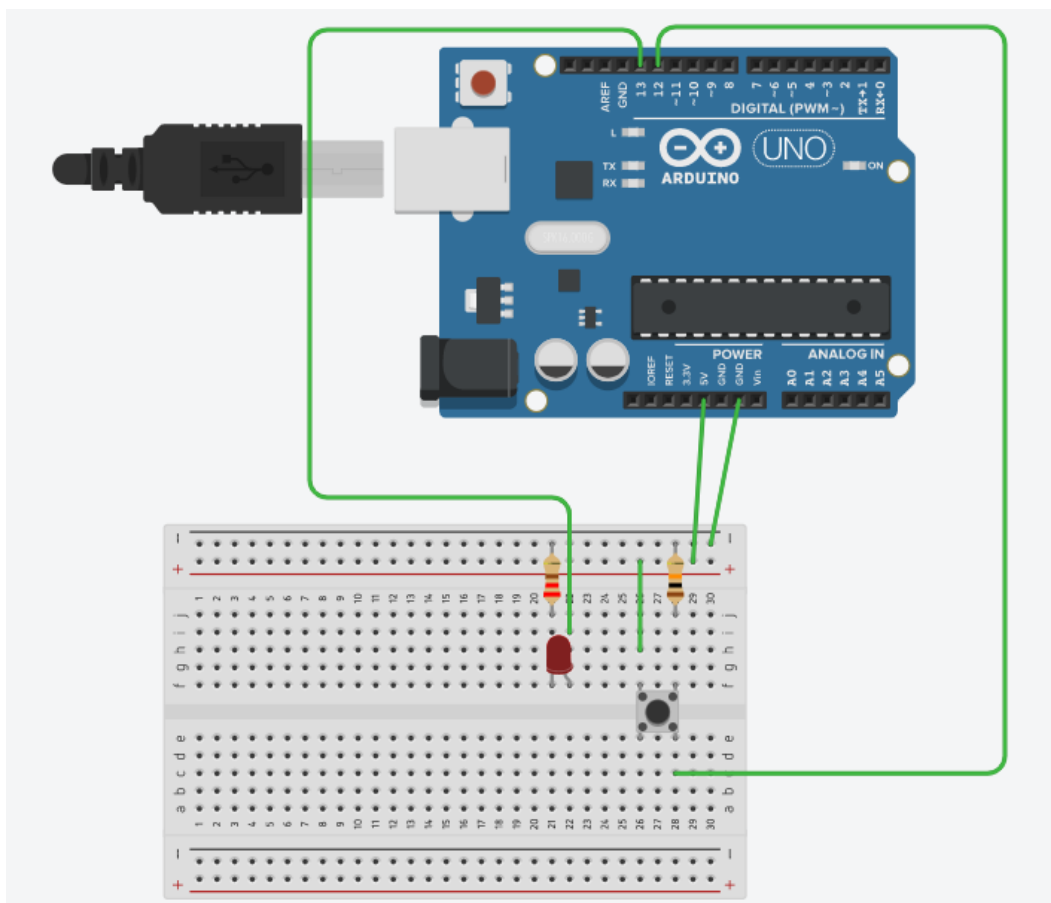
Resistor Colour Code Calculator

| | | | | |
|------------------------|---|----------|----------|---------------------|
| Select # of Bands: | Band #1: | Band #2: | Band #3: | Multiplier |
| Four | 2 - Red | 2 - Red | None | 10 Ω - Brown |
| Tolerance | Temperature Coefficient | | | |
| $\pm 5\%$ (J) - Gold | None | | | |
| Resistance in Ω |  | | | |
| 220 | | | | |

כפתורים

לאחר מכן התחלנו לחקור כיצד כפתור עובד וכיצד לחבר כפתור ללוח שלנו, באתר [הבא](#) מצאנו הסבר על כפתור וכיצד לחבר אותו למערכת. ולאחר מכן למדנו שיטות שונות כיצד לחבר מספר כפתורים למערכת [כאן](#).

כעת נחבר את הכפתור למערכת ונקבל את הצורה הבאה:



הקוד שלנו יראה כך:

```
1  const int buttonPin = 12;    // the number of the pushbutton pin
2  const int ledPin = 13;      // the number of the LED pin
3
4  // variables will change:
5  int buttonState = 0;        // variable for reading the pushbutton status
6
7  void setup() {
8    // initialize the LED pin as an output:
9    pinMode(ledPin, OUTPUT);
10   // initialize the pushbutton pin as an input:
11   pinMode(buttonPin, INPUT);
12 }
13
14 void loop() {
15   // read the state of the pushbutton value:
16   buttonState = digitalRead(buttonPin);
17
18   // check if the pushbutton is pressed. If it is, the buttonState is HIGH:
19   if (buttonState == HIGH)
20     // turn LED on:
21     digitalWrite(ledPin, HIGH);
22
23 }
```


נעבור עליו בקצרה.

בהתחלה יש לנו שתי הצהרות, אחת על מספר הפין של הלד (13) והשנייה על מספר הפין של הכפתור (12). בנוסף קיימת הצהרה נוספת בשם `buttonState` שתסמן את מצב הכפתור שלנו (0 - לא לחוץ, 1 - לחוץ).

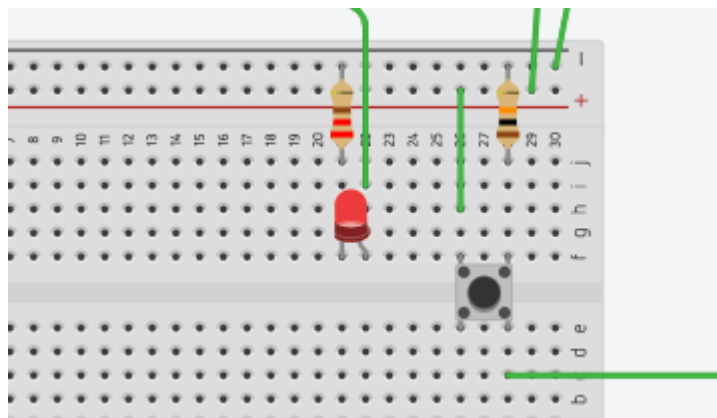
לאחר מכן בשורות 7-12 יש את פונקציית `setup` שלנו. היא מגדירה את פין 13 כיציאה (כלומר הוא יוציא מתח וככה נדליק את הלד) ואת פין 12 ככניסה (כלומר הוא יקבל מתח וככה נזהה את הלחיצה על הכפתור).

לבסוף נראה את פונקציית `loop` שלנו. היא רצה שוב ושוב ומבצעת את הדברים הבאים:

בשורה 16 היא קוראת את מצב הכפתור, אם הוא לחוץ היא שומרת 1 (`HIGH`) במשתנה `buttonState` שלנו כדי לסמן שהכפתור לחוץ.

לאחר מכן בשורה 19 נבדוק האם הכפתור נלחץ. ואם כן, נדליק את הלד.

כעת אם נריץ את הסימולציה ונלחץ על הכפתור נראה שהלד יידלק:



בשלב הלמידה הבא ניסינו להוסיף לקוד תנאי נוסף, שכשאר לוחצים שוב על הכפתור הLED יכבה.

כעת נתקלנו בבעיה. זמן לחיצה של אדם על כפתור, ארוך יותר מאשר כמה סיבובים של פונקציית הloop. ולכן, מכיוון שבכל סיבוב של פונקציית הloop לוח Arduino דוגם מחדש את הכפתור, הלוח מרגיש כאילו הוא מקבל כמה לחיצות ומכבה ומדליק את הLED מספר פעמים.

ע"מ לפתור את הבעיה השתמשנו במנגנון שנקרא State Change Detection (או Edge Detection).

כלומר, אנחנו ניצור מנגנון ש"זוכר" את המצב האחרון של הכפתור ולא מתייחס ללחיצה ארוכה כאל מספר לחיצות.

המנגנון יעבוד בצורה כזאת: אם המנגנון מזהה שהכפתור לחוץ ואתה דגימה נוספת של הכפתור (הרצה נוספת של פונקציית הloop) הוא מזהה שהכפתור עדיין לחוץ, הוא יתייחס לשתי הלחיצות כאל לחיצה אחת. ולכן לא יבצע 2 פעולות. אך אם המנגנון יזהה בין שתי הלחיצות רגע שבו אין לחיצה כלל, הוא יתייחס לשתי הלחיצות כאל שתי לחיצות נפרדות (למעשה - ע"מ שהוא יתייחס לשתי הלחיצות כאל לחיצות נפרדות, חייב להיות ביניהם רגע שבו המנגנון ידגום את הכפתור בהרצה נוספת של פונקציית הloop ויזהה שאין לחיצה כלל).

שילוב המנגנון בקוד שלנו יראה כך:

```

1  const int buttonPin = 12;      // the number of the pushbutton pin
2  const int ledPin = 13;        // the number of the LED pin
3
4
5  // Variables will change:
6  int buttonState = 0;          // current state of the button
7  int lastButtonState = 0;      // previous state of the button
8
9  void setup() {
10     // initialize the button pin as a input:
11     pinMode(buttonPin, INPUT);
12     // initialize the LED as an output:
13     pinMode(ledPin, OUTPUT);
14
15 }
16
17
18 void loop() {
19     // read the pushbutton input pin:
20     buttonState = digitalRead(buttonPin);
21
22     // compare the buttonState to its previous state
23     if (buttonState != lastButtonState && buttonState == HIGH) {
24         // if the state has changed, increment the counter
25         if (digitalRead(ledPin) == LOW) {
26             // if the current state is HIGH then the button went from off to on:
27             digitalWrite(ledPin, HIGH);
28         } else {
29             // if the current state is LOW then the button went from on to off:
30             digitalWrite(ledPin, LOW);
31         }
32     }
33
34     // save the current state as the last state, for next time through the loop
35     lastButtonState = buttonState;
36
37 }
38

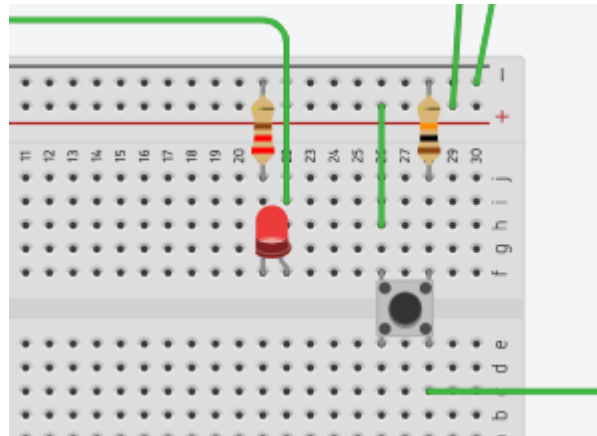
```

נתמקד בהסבר רק של השינויים בקוד.

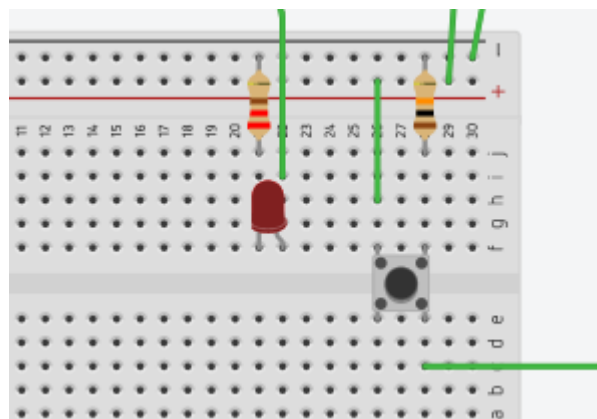
בשורה 7 הוספנו משתנה שתפקידו לנהל את היסטורית המצבים שלנו.

לאחר מכן בשורה 23, נכנס לתוך התנאי רק אם המצב הנוכחי של הכפתור שונה מהמצב הקודם – כלומר רק אם מדובר על שתי לחיצות נפרדות ולא על לחיצה ממושכת (כמובן שמכיוון שהתנאי עוסק בהדלקה וכיבוי של הled כאשר לוחצים על הכפתור, המשך התנאי 'בדוק גם ששינוי המצב של הכפתור הוא ממצב לא לחוץ למצב לחוץ ולא להפך).

כעת מימשנו את המנגנון State Change Detection כמו שצריך. וכפי שניתן לראות בסימולציה, כאשר נלחץ על הכפתור הled תידלק:



וכאשר נלחץ שוב, הled יכבה:

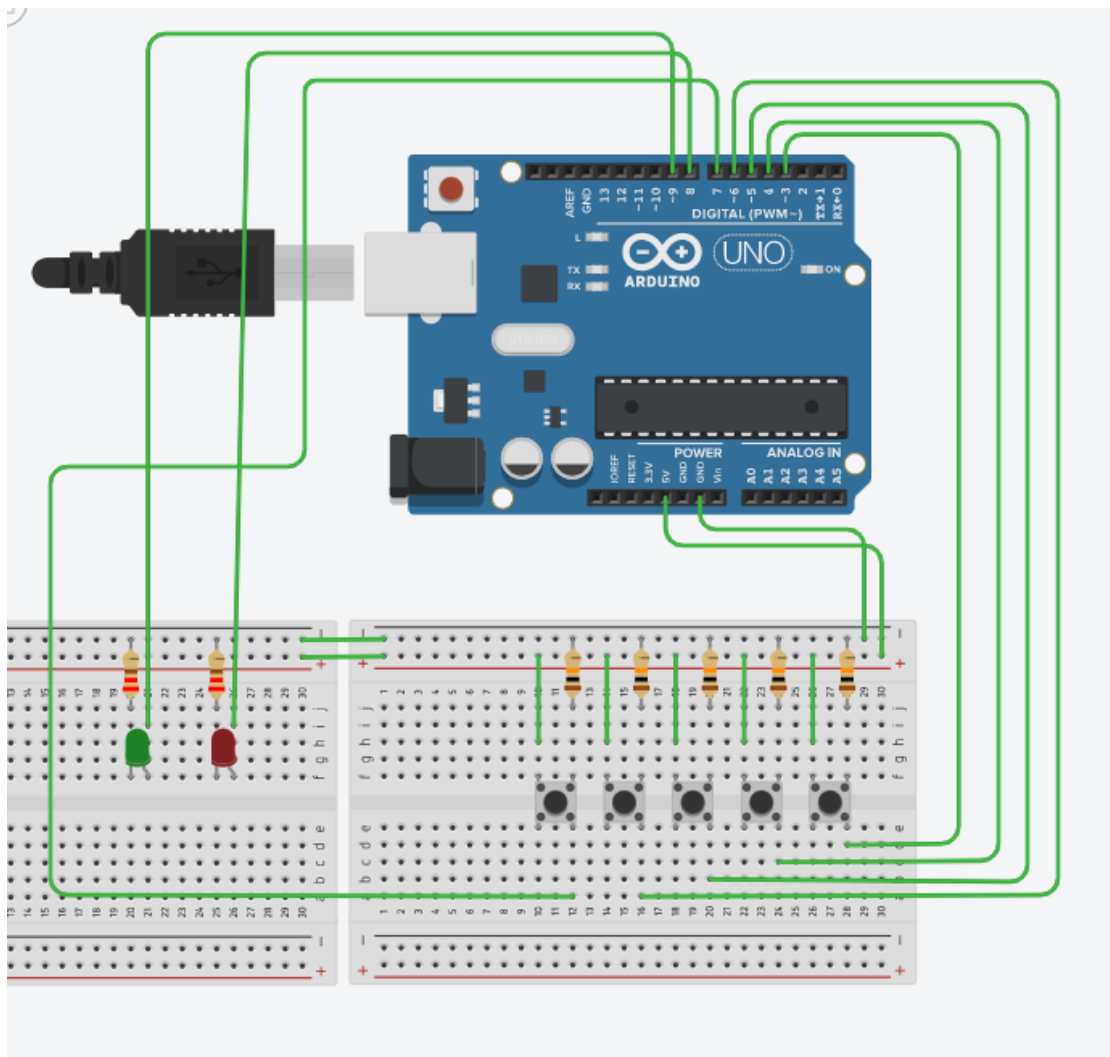


חלק ראשון - Simple Timing attack

כעת נעבור לבניית הפרויקט עצמו. את כל הקוד של החלק הראשון ניתן למצוא כאן. כפי שאמרנו הפרויקט עוסק במתקפת ערוץ צד. מה שמיוחד במתקפות כאלו הוא שאיננו צריכים להיות בתוך המערכת, אלא רק להאזין למערכת. לפעמים נאזין לאותות החשמליים, לפעמים לקולות הבוקעים מהמערכת, לפעמים לטמפרטורה וכך הלאה.

המתקפה שאנחנו נממש נקראת Timing Side Channel Attack ונוגעת לזמנים של המערכת. אנחנו ננתח את הזמן הדרוש למערכת לבצע פעולות מסוימות במהלך הקשת הקוד. ומכיוון שכל איטרציה של המערכת גוזלת זמן עיבוד מסוים. ניתוח של הפרשי הזמן יכול לחשוף את תהליך בקרת הזרימה הפנימי של המערכת וכך לחשוף בפנינו את סיסמה.

בשלב הראשון של הפרויקט נבנה מערכת קוד באמצעות לוח Arduino. המערכת תכיל חמישה כפתורים ושני לדים – ירוק ואדום. כאשר נקיש את הקוד הנכון המערכת תדליק את הנורה הירוקה וכאשר נקיש קוד שגוי המערכת תדליק נורה אדומה. נבנה את המערכת כפי שלמדנו למעלה. המערכת תראה כך:



הקוד שלנו יראה ככה:

```
1 int redLedPin = 8;    // choose the pin for the LED
2 int greenLedPin = 9;
3
4 int input5Pin = 7;    // define push button input pins
5 int input4Pin = 6;
6 int input3Pin = 5;
7 int input2Pin = 4;
8 int input1Pin = 3;
9
10 int code[] = {1,2,3,4,5};
11 int pressHistory[] = {0,0,0,0,0};
12 int counter = 0;
13
14 int lastButtonState[] = {LOW, LOW, LOW, LOW, LOW};
15
16 void setup()
17 {
18     pinMode(redLedPin, OUTPUT);    // declare LED as outputs
19     pinMode(greenLedPin, OUTPUT);
20
21     pinMode(input5Pin, INPUT);    // declare push button inputs
22     pinMode(input4Pin, INPUT);
23     pinMode(input3Pin, INPUT);
24     pinMode(input2Pin, INPUT);
25     pinMode(input1Pin, INPUT);
26 }
27
```

נסביר את החלקים העיקריים בו.

בשורות 1-8 אנחנו מצהירים מראש (לשם הנוחות) על הפינים שבהם נשמש עבור הLEDים והכפתורים.

שורה 10 היא מערך שמחזיק את הקוד שלנו, ובשורה 11 היסטורית הלחיצות הנוכחית של המשתמש. היסטורית הלחיצות נמנית באמצעות מונה שמוגדר בשורה 12.

שורה 13 היא בעצם מנגנון State Change Detection שממומש במערך עבור כל כפתור בנפרד.

לאחר מכן יש את פונקציית setup כפי שראינו כבר בעבר.

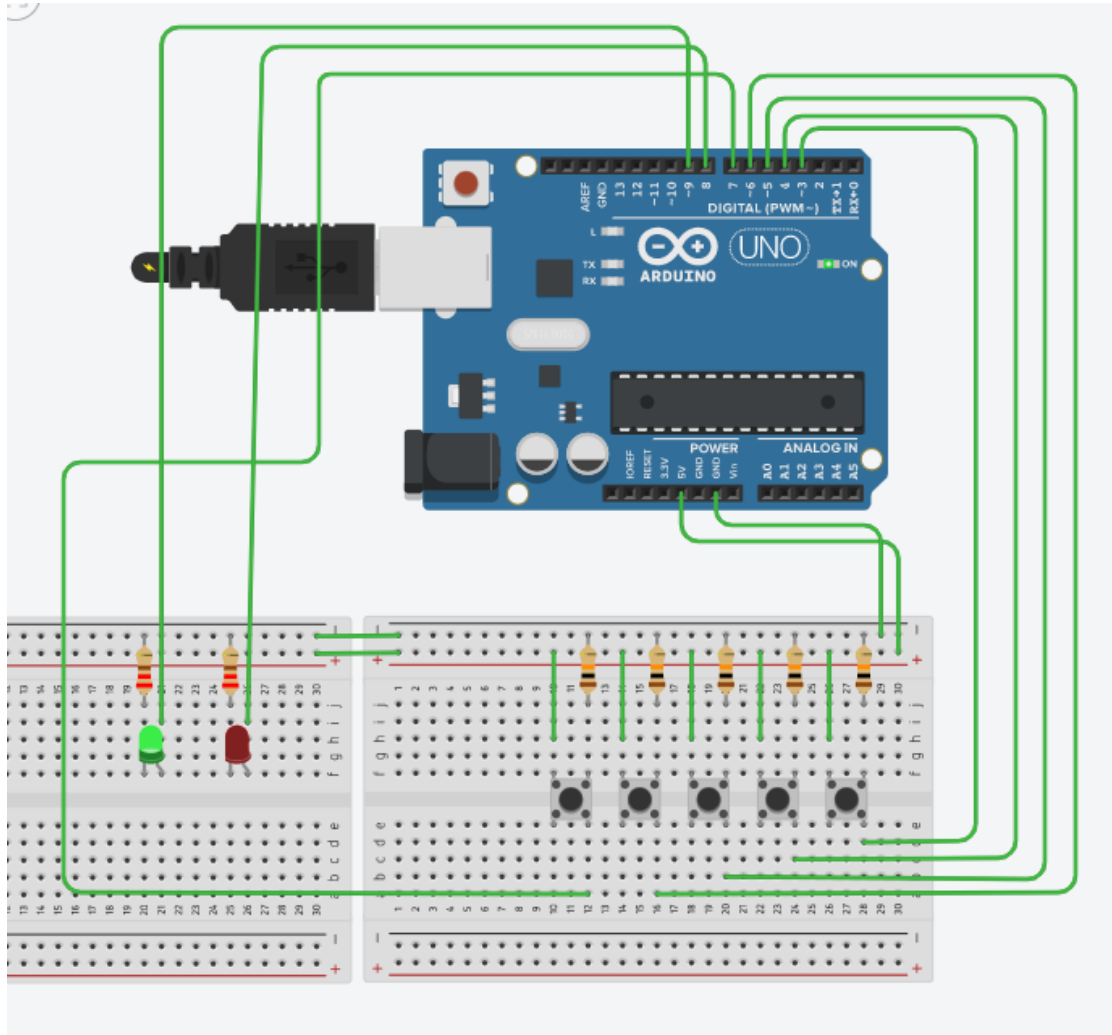
המשך הקוד שלנו יראה כך:

```
28 void loop()
29 {
30     if(counter == 5)
31         checkPass();
32     for( int i = 3; i<= 7; i++)
33         checkPush(i);
34 }
35
36 void checkPush(int pinNumber)
37 {
38     int buttonState = digitalRead(pinNumber); // read input value
39
40     if (buttonState != lastButtonState[pinNumber-3]) {
41         lastButtonState[pinNumber-3] = buttonState;
42
43         if (lastButtonState[pinNumber-3] == HIGH) // check if the input is HIGH (button released)
44         {
45             pressHistory[counter] = pinNumber - 2;
46             counter++;
47             digitalWrite(redLedPin, LOW);
48             digitalWrite(greenLedPin, LOW);
49         }
50     }
51 }
52
53 }
54
55 void checkPass()
56 {
57     for(int i = 0; i<5; i++)
58     {
59         if(pressHistory[i]!=code[i])
60         {
61             counter = 0;
62             digitalWrite(redLedPin, HIGH);
63             return;
64         }
65         delay(300);
66     }
67     digitalWrite(greenLedPin, HIGH);
68     counter = 0;
69 }
```

פונקציית הloop בודקת שוב ושוב עבור כל כפתור האם הוא נלחץ (checkPush(i)) ובנוסף בודקת האם היו חמישה לחיצות. ואם כן היא מפעילה את הפונקציה לבדיקת הסיסמה.

הפונקציה checkPush(int pinNumber) בודקת האם התבצעה לחיצה על הכפתור שנשלח אליה (בשילוב מנגנון State Change Detection) ואם כן היא שומרת את מספר הכפתור במערך היסטורית הלחיצות (שורה 45) ומעלה את המונה של מספר הלחיצות (שורה 46).

כאשר התבצעו 5 לחיצות מופעלת הפונקציה `checkPass()` שבה קיימת (חלק מ) החולשה של המערכת. הפונקציה עוברת בלולאה על כל לחיצה שנשמרה בהיסטורית הלחיצות ומשווה אותה עם הסיסמה השמורה במערכת באמצעות השוואת מחרוזות פשוטה. ברגע שהתגלה תו שגוי, המערכת מודיעה שהסיסמה שגויה ומדליקה נורה אדומה (כמובן שאם הסיסמה נכונה - נדלקת נורה ירוקה).



מכיוון שמערכת ממומשת עם השוואת מחרוזות פשוטה, נמדוד את זמן התגובה של המערכת ונוכל לזהות בכמה תווים צדקנו.

נסביר את השיטה:

נניח שהשוואת כל תו מהסיסמה לוקחת זמן של שניה, נוכל למדוד כמה זמן עבר מהרגע שהקשנו את התו שאחרון של הסיסמה עד שקיבלנו חווי שלילי (נורה אדומה) ולפי זה להסיק בכמה תווים צדקנו.

למשל, אם נקבל נורה אדומה לאחר שלוש שניות נדע שהמערכת השוותה 3 תווים (שניה לכל תו) ושצדקנו בשני התווים הראשונים ורק בשלישי טעינו ולכן שם קיבלנו נורה אדומה. אך אם היינו מקבלים נורה אדומה אחרי 4 שניות, נדע שצדקנו ב3 תווים וכך הלאה.

את הזמנים האלו נמדוד ובאמצעותם נסיק מהי הסיסמה תוך מספר מועט של ניסיונות (ולא Brute force).

המתקפה הזו, שבה אנו מודדים את הזמנים של המערכת, נקראת Timing Side Channel Attack.

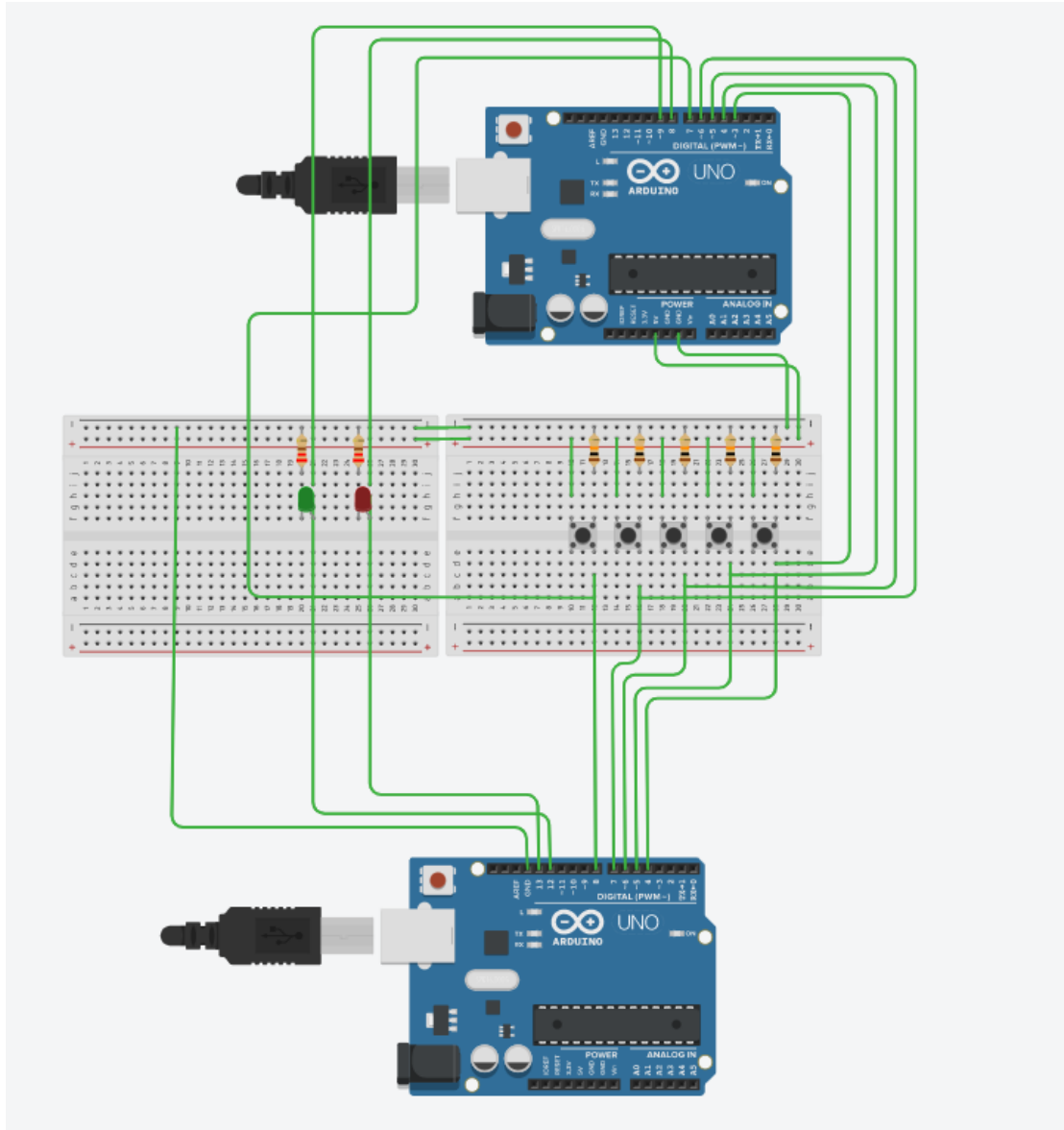
כעת נבנה לוח נוסף שמודד את אותם זמנים.

בניית לוח הפריצה

בשלב הזה, נבנה לוח שמתחבר למגעים של לוח Arduino, את הלוח הזה נממש באמצעות לוח Tiva (בסימולטור נשתמש בעוד Arduino). את הקוד של לוח הפריצה ניתן למצוא [כאן](#).

לוח הפריצה יתחבר למגעים של הכפתורים של הלוח הראשון ע"מ שיוכל "ללחוץ בעצמו" על הכפתורים. בנוסף הוא יתחבר למגעים של הלדים של הלוח הראשון ע"מ שיוכל להרגיש האם הסיסמה שהוכנסה שגויה או נכונה וע"מ שיוכל למדוד את זמני התגובה כפי שהסברנו לפני.

כעת המערכת שלנו תראה כך:



כאשר נפעיל את המערכת, הלוח החדש יזרים לכפתורים זרם (האלגוריתם יוסבר בהמשך) כדי לדמות לחיצות אצל הלוח הראשון.

הלוח החדש ימתין לראות כמה זמן לוקח ללוח הראשון להגיב לכך שהסיסמה שגויה (נורה אדומה) ולפי זה יסיק בכמה לחיצות הוא צדק. ככה עד שימצא את הסיסמה.

הקוד בלוח הנוסף יראה כך:

```
1  int button5 = 8;
2  int button4 = 7;
3  int button3 = 6;
4  int button2 = 5;
5  int button1 = 4;
6  int redLed = 13;
7  int greenLed = 12;
8
9  int buttons[] = {0, button1, button2, button3, button4, button5};
10 int correctPass[] = {-1, -1, -1, -1, -1};
11 int passToCheck[] = {0, 0, 0, 0, 0};
12
13 int numOfCorrectButtons = 0;
14
15 void setup() {
16     pinMode(button1, OUTPUT);
17     pinMode(button2, OUTPUT);
18     pinMode(button3, OUTPUT);
19     pinMode(button4, OUTPUT);
20     pinMode(button5, OUTPUT);
21     pinMode(redLed, INPUT);
22     pinMode(greenLed, INPUT);
23     Serial.begin(9600);
24     delay(5000);
25 }
26
```

כמובן שנתמקד בעיקרי הקוד.

המערך `passToCheck` בשורה 11 תפקידו להחזיק את הסיסמה הבאה שאותה נבדוק.

והמערך `correctPass` בשורה 10 תפקידו להחזיק את תווים של הסיסמה שכבר מצאנו.

חשוב לשים לב שהפעם בפונקציית `setup` נגדיר את הפינים של הפתורים והלדים ההפך. כלומר פין של כפתו יהיה `output` ולא `input` כי נרצה ללחוץ על הכפתור ולא לבדוק האם הוא נלחץ. ופין של לד יהיה `input` ולא `output` כי נרצה לבדוק האם הלד נדלק ולא להדליק אותו.

המשך הקוד יראה כך:

```
40 void loop() {
41   if (digitalRead(greenLed) != HIGH)
42   {
43     int maxTime = 0;
44     int button = 0;
45     for (int i = 1; i <= 5; i++)
46     {
47       generatePass(i);
48       int t1 = millis();
49       enterPass(passToCheck);
50       while (digitalRead(redLed) != HIGH && digitalRead(greenLed)
51       int t2 = millis();
52       int delta = (t2 - t1);
53       Serial.print(i);
54       Serial.print(" : ");
55       Serial.print(delta);
56       Serial.print(", ");
57
58       if (maxTime < delta)
59       {
60         maxTime = delta;
61         button = i;
62       }
63       if(digitalRead(greenLed) == HIGH)
64       {
65         printPass();
66         break;
67       }
68     }
69     if(digitalRead(greenLed) != HIGH)
70     {
71       Serial.println();
72       Serial.print(button);
73       Serial.println();
74       if(numOfCorrectButtons < 5)
75       {
76         correctPass[numOfCorrectButtons] = buttons[button];
77         numOfCorrectButtons++;
78       }
79     }
80   }
81 }
```

כל זמן שהנורה הירוקה לא דולקת (כלומר שלא מצאנו את הסיסמה) נבצע סט של 5 ניסיונות - ניסיון עבור כל כפתור. בכל ניסיון נשתמש בפונקציה generatePass(i) (שורה 47) על מנת לבנות את הסיסמה הבאה שננסה.

הפונקציה generatePass מקבלת את מספר הכפתור שאותו אנו מנסים כעת ומחברת אותו עם הסיסמה שמצאנו עד כה:

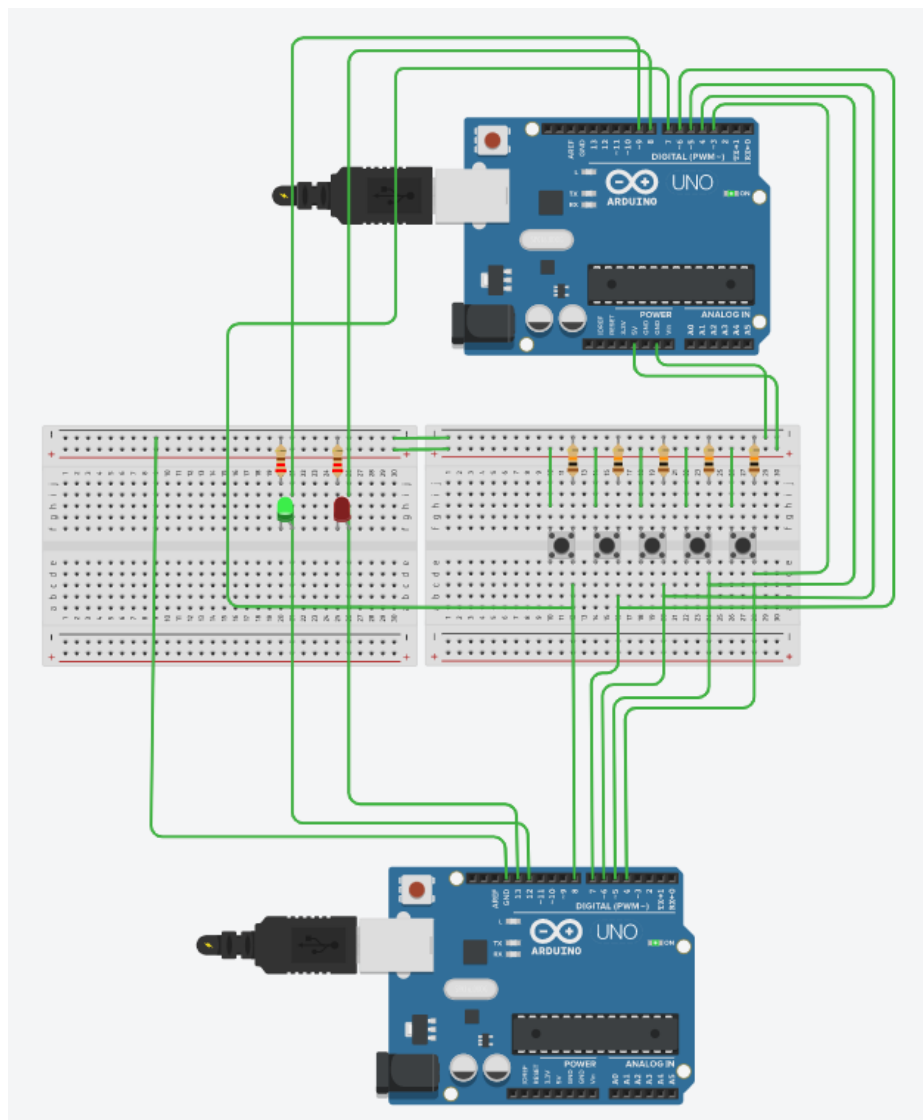
```
27 void generatePass(int buttonToCheck)
28 {
29     for (int i = 0 ; i < 5; i++)
30     {
31         if (correctPass[i] == -1)
32             passToCheck[i] = buttons[buttonToCheck];
33         else
34             passToCheck[i] = correctPass[i];
35     }
36 }
```

לאחר מכן באמצעות הפונקציה enterPass() "נלחץ" על הכפתורים. בכל סיבוב של לחיצות נמדוד את זמני התגובה (שורות 52-48 - בעמוד הקודם) ונשמור את מספר הכפתור שזמן התגובה שלו היה הארוך ביותר (כלומר שצדקנו בכמה שיותר תווים – שורות 62-58)

ככה בכל סיום של פונקציית loop נרכיב תו נוסף בסיסמה שלנו (שורות 77-76) עד שנמצא את כל המחזורות ונדליק את הנורה הירוקה.

כעת נריץ את המערכת וננתח את התוצאות.

כפי שניתן לראות, יש לנו נורה ירוקה. כלומר לח הפריצה שלנו אכן הצליח למצוא את הסיסמה הנכונה:



נביט במסך הסריאלי של לוח הפריצה וננתח את התוצאות שהודפסו:

```
1 : 401, 2 : 642, 3 : 402, 4 : 401, 5 : 402,  
2  
1 : 942, 2 : 642, 3 : 642, 4 : 642, 5 : 642,  
1  
1 : 943, 2 : 942, 3 : 1242, 4 : 942, 5 : 942,  
3  
1 : 1242, 2 : 1542, 3 : 1242, 4 : 1154, 5 : 1241,  
2  
1 : 1543, 2 : 1542, 3 : 1842,  
Password found!!  
The password is:  
2 ,1 ,3 ,2 ,3
```

בכל מלבן כחול, מופיעות תוצאות הרצה של סט של לחיצות על הכפתורים.

כלומר לקחנו את התווים שמצאנו אותם כנכונים עד כה ושרשרנו להם את התו שכעת אנו מנסים.

ניתן לראות שבכל מלבן מופיעים חמישה כפתורים ממוספרים מ1 עד 5 וליד כל אחד מהם את זמן התגובה של המערכת לאותו כפתור.

ככל שזמן התגובה ארוך יותר, ככה צדקנו ביותר תווים כפי שהסברנו קודם. ולכן בשורה השנייה בכל מלבן ניתן לראות שהלוח הפורץ בחר את התוו עם הזמן הכי גדול.

לאחר כ23 ניסיונות (27 שניות) ותוך כדי הבדיקות נלקה הנורה הירוקה ולכן הלוח הפורץ עצר את הפעולה שלו והדפיס את הסיסמה שהוא מצא כפי שניתן לראות בריבוע האדום.

חלק שני -

Timing attack against simple random time-delay insertion

מכיוון שכל הפריצה שלנו מתבססת על זמן התגובה של המערכת לסיסמה המוקשת, ניתן להוסיף דיילי רנדומלי למערכת וככה לשבש את מדידת הזמנים של הלוח הפורץ. מכיוון שה random time-delay יגרום לכך שלפעמים לתו שגוי ייקח יותר זמן מאשר תו נכון.

נעשה זאת בצורה הבאה:

```
54 void checkPass()
55 {
56     delay(random(50,200));
57     for(int i = 0; i<5; i++)
58     {
59         if(pressHistory[i]!=code[i])
60         {
61             counter = 0;
62             digitalWrite(redLedPin, HIGH);
63             return;
64         }
65         delay(100);
66     }
67     digitalWrite(greenLedPin, HIGH);
68     counter = 0;
69 }
```

כפי שרואים, בשורה 56 הוספנו דיילי רנדומלי שאמור לשבש את מדידת הזמנים ולמנוע את גילוי הסיסמה.

אך, כפי שניתן לראות [כאן](#), ניתן לשנות את הקוד בלוח הפורץ ע"מ שיעשה מספר רב של ניסיונות כדי לשבור את הסטייה שנוצרת בעקבות הדיילי הרנדומלי. ולמצוא את הסיסמה בכל זאת.

כלומר במקום לנסות פעם אחת כל סיסמה, ננסה אותה מספר רב של פעמים ולפי ממוצע הזמנים שנקבל, נסיק האם התקדמנו בתו נוסף או לא. בצורה הזאת:

```
80 double calcPassTime()
81 {
82     double sum = 0;
83     for(int i = 0 ; i < 5 ; i++)
84     {
85         int t1 = millis();
86         enterPass(passToCheck);
87         while (digitalRead(redLed) != HIGH && digitalRead(greenLed) != HIGH);
88         if(digitalRead(greenLed) == HIGH)
89             break;
90         int t2 = millis();
91         sum += (t2 - t1);
92     }
93     return sum/5;
94 }
```

ההיגיון שעומד מאחורי השיטה הוא כזה: מכיוון שלכל בדיקה של סיסמה נוסף זמן רנדומלי בין 0 ל- a כלשהו. בביצוע בדיקות רבות על אותה הסיסמה נקבל בממוצע שלכל בדיקה של סיסמה הוספנו $a/2$ זמן, בין אם היא נכונה או לא. כלומר, מקבלים עבור כל הקודים את זמן הריצה האמיתי שלהם בתוספת קבועה לכולם. וככה הלוח הפורץ עדיין יצליח לאתר את הסיסמה.

וזה בדיוק מה שרואים בקוד - כל סיסמה אנו מנסים להכניס 5 פעמים ומחשבים את ממוצע הזמנים של אותם 5 ניסיונות. ככה בין אם הסיסמה נכונה ובין אם לא, הוספנו בערך $a/2$ זמן.

ולפי הזמן הממוצע שקיבלנו, נוכל לקבוע האם צדקנו בסיסמה או לא.

את כל הקוד של החלק השני ניתן למצוא [כאן](#).

חלק שלישי - Basic timing attack demonstration on RSA

את כל הקוד של החלק השלישי ניתן למצוא [כאן](#).

מהו RSA

RSA היא מערכת הצפנת מפתח ציבורי דטרמיניסטית הראשונה שהומצאה והיא עדיין בשימוש נרחב כיום. ב־RSA, כבכל מערכת מפתח ציבורי, מפתח ההצפנה אינו סודי והוא שונה ממפתח הפענוח שנשמר בסוד, לכן היא נקראת אסימטרית. האסימטריה ב־RSA נובעת מהקושי המעשי (והכמעט בלתי אפשרי) שבפירוק לגורמים של מספר פריק, שהוא כפולה של שני ראשוניים גדולים (מדובר על בעיה פתוחה בתורת המספרים).

כלומר, קל יחסית ליצור שני מספרים ראשוניים q , p ולחשב את המכפלה שלהם $N = pq$. אך בהינתן N קשה למצוא את גורמיו p ו- q . ההצפנה משתמשת בערך ציבורי, או במפתח, המופץ וידוע לכל מי שרוצה לשלוח הודעה. הפענוח מתבצע באמצעות מפתח פרטי הנשמר בסוד על ידי הנמען המיועד ולא ניתן להסיק אותו מהמפתח הציבורי. הצפנת כזאת עובדת מבלי לדרוש משני הצדדים המעורבים לשמור על סוד מוסכם - המפתח הפרטי לעולם לא צריך להישלח לשולח.

למרות חוזק מתמטי אדיר זה, מחקרים הראו כי ניתן לשחזר מפתחות פרטיים של RSA מבלי לשבור ישירות את RSA אלא באמצעות Timing attack.

במתקפה מסוג זה התוקף מתבונן בזמן הריצה של אלגוריתם הצפנה ובכך מסיק את הפרמטר הסודי הכרוך בפעולות. למרות שמקובל להסכים כי RSA מוגן מפני התקפה ישירה, הפגיעות של RSA למתקפות תזמון אינה ידועה כל כך ולעתים קרובות מתעלמים ממנה.

קעת נממש מתקפה מסוג זה. אך לפני זה נקדים ונסביר כיצד מממשים מערכת הצפנה של RSA (אצלנו היא תמומש בתוך לוח Arduino) ותוך כדי נגיע למתקפה.

כפי שכתבנו במערכת RSA יש שני מפתחות א-סימטריים, אחד פרטי d ואחד ציבורי e . והכל בעולם מודולו N (על עולם האריתמטיקה המודולרית ניתן לקרוא [כאן](#)), שהוא מכפלת הגורמים שהזכרנו.

בשביל להצפין הודעה m , מערכת ההצפנה מבצע את הפעולה $m^d \bmod(n)$ (חזקה ומודולו) שלה נקרא modular exponent.

את הפעולה modular exponent נבצע באמצעות אלגוריתם Repeated Squaring שהוא שיטה לביצוע modular exponent בצורה יעילה. על השיטה הזאת נפרט כבר, מכיוון ששם המתקפה שלנו מתקיימת.

Repeated Squaring משתמש בפעולות הכפל ומכיוון שמדובר בכפל מספרים גדולים ממש, הוא משתמש בשיטה נוספת שנקראת Montgomery Product, שמבצע פעולות הכפל של מספרים גדולים בצורה יעילה.

Mongomery Product דורש הכנה מוקדמת של מספר פרמטרים. ואותם נקבל מאלגוריתם $nPrime$.

באלגוריתמים $nPrime - 1$ Mongomery Product לא נתעמק כאן, מכיוון שהם רק מבצעים את פעולות הכפל בצורה יעילה ולא משפיעים על מימוש המתקפה (לפחות לא באופן ישיר).

מידע נוסף עליהם ניתן למצוא [כאן](#) ו[כאן](#).

Repeated Squaring

כפי שאמרנו Repeated Squaring הוא שיטה לביצוע modular exponent בצורה יעילה והיא עומדת בבסיס המתקפה שלנו.

בשיטה זו אנו נמיר את d (המפתח הפרטי) למספר בינארי ואז נסרוק את מחזורות הביטים משמאל לימין.

בכל ביט אנו נעלה את m בריבוע (Mongomery Product). ובנוסף, אם הביט הינו 1, נכפיל את הערך שקיבלנו עם ההודעה המקורית (גם פה Mongomery Product).

כמובן שבכל איטרציה (כאשר סורקים את הביטים משמאל לימין) מבצעים הפחתה ($\%n$) במידת הצורך כדי להישאר בעולם המודולו n .

```
// Compute  $y = x^d \pmod N$ 
// where, in binary,  $d = (d_0, d_1, d_2, \dots, d_n)$  with  $d_0 = 1$ 
s = x
for i = 1 to n
    s =  $s^2 \pmod N$ 
    if  $d_i == 1$  then
        s =  $s \cdot x \pmod N$ 
    end if
next i
return s
```

היתרון בשיטה נעוץ בכך שבכל שלב אנו מסתכלים רק על ביט בודד ובכך שבסוף כל שלב אנו מבצעים הפחתה. ככה אנו נשארים תמיד במספרים נמוכים (נמוכים ביחס למספרים שאותם אנו כופלים) שקל לבצע עליהם פעולות.

נראה דוגמה כיצד Repeated Squaring הופך לנו את הפעולה $12^{45} \bmod(40)$ לפעולה פשוטה:

במקום לחשב משהו קצת לא אפשרי כמו

$$12^{45} = 3657261988008837196714082302655030834027437228032$$

ואז לחשב את התוצאה

$$3657261988008837196714082302655030834027437228032 \bmod(40) = 32$$

נפעל בצורה כזאת:

$$12^{45} \bmod(40)$$

נמיר את החזקה למספר בינארי ונסרוק אותו משמאל לימין:

$$(45)_{10} = (101101)_2$$

הביט הראשון הוא 1 ולכן גם נעלה בריבוע וגם נכפיל במספר המקורי:

$$12^2 * 12 = 1728 = 8 \bmod(40)$$

הביט השני הוא 0 ולכן רק נעלה בריבוע:

$$8^2 = 64 = 24 \bmod(40)$$

הביט השלישי הוא 1 ולכן גם נעלה בריבוע וגם נכפיל במספר המקורי:

$$24^2 * 12 = 6912 = 32 \bmod(40)$$

הביט הרביעי הוא 1 ולכן גם נעלה בריבוע וגם נכפיל במספר המקורי:

$$32^2 * 12 = 12288 = 8 \bmod(40)$$

הביט החמישי הוא 0 ולכן רק נעלה בריבוע:

$$8^2 = 64 = 24 \bmod(40)$$

הביט השישי הוא 1 ולכן גם נעלה בריבוע וגם נכפיל במספר המקורי:

$$24^2 * 12 = 6912 = 32 \bmod(40)$$

ניתן לראות שהגענו לתוצאה בצורה פשוטה הרבה יותר.

אבל, ההפחתה הזאת שמבצעים בסוף כל שלב - היא נקודת החולשה של המערכת שננצל במתקפה.

אצלנו במערכת ה-Arduino יקבל ערכים ויצפין אותם.

את הקוד של ההצפנה ב-Arduino ניתן למצוא כאן [והוא יראה כך](#):

```
1 uint64_t MontgomeryProduct(uint64_t a, uint64_t b, uint64_t n, uint64_t nprime, uint64_t r)
2 {
3     //This function calc the montgomery product of numbers a and b
4     uint64_t t = a * b;
5     uint64_t t1 = t % r;
6     uint64_t m = t1 * nprime % r;
7
8
9     uint64_t t_div_r = t / r;
10    uint64_t mn_div_r = (m * n) / r;|
11
12    uint64_t t_apr_r = t % r;
13    uint64_t mn_apr_r = (m * n) % r;
14
15
16    uint64_t u = t_div_r + mn_div_r + 1;
17
18    if (u < n)
19        return u;
20    delay(100);
21    return u - n;
22 }
23
24 uint64_t modexp(uint64_t a, String exp, uint64_t n, uint64_t r, uint64_t k) {
25     //This function encrypt/decrypt message m by rsa protocol
26     uint64_t a_ = (a * r) % n;
27     uint64_t x_ = (1 * r) % n;
28
29     for (int i = exp.length() - 1; i >= 0; i--)
30     {
31         x_ = MontgomeryProduct(x_, x_, n, k, r);
32         if (exp[i] == '1')
33             x_ = MontgomeryProduct(a_, x_, n, k, r);
34     }
35
36     return MontgomeryProduct(1, x_, n, k, r);
37 }
38
39
40 void nPrime(uint64_t n)
41 {
42     //This function calc the r and the k for the montgomery product
43     uint64_t k = (log(n)/log(2)) + 1;
44     uint64_t r = pow(2, k);
45     uint64_t rInverse = ModInverse(r, n);
46     uint64_t nPrime = (r * rInverse - 1) / n;
47     result[0] = nPrime;
48     result[1] = r;
49 }
50
```

נסביר את עיקרי הקוד.

ניתן לראות פונקציה בשם modxp , היא בעצם מצבעת את $m^d \bmod(n)$ רק באמצעות האלגוריתמים שהצגנו.

הפרמטרים המתקבלים הם:

a – ההודעה שאותה נצפין (m).

exp – החזקה (d), שהיא המפתח הפרטי. היא מתקבלת כמחרוזת של ביטים ולא כמספר (בשביל אלגוריתם Repeated Squaring).

n – עולם המודולו.

r, t – הם פרמטרים מקדימים שהזכרנו שמסייעים לפונקציה והם מגיעים מהפונקציה $n\text{Prime}$.

בריבוע האדום ניתן לראות את אלגוריתם Repeated Squaring.

כפי שניתן לראות בקוד, יש עוד המון מתמטיקה מסביב. כל המתמטיקה היא חלק מהאלגוריתמים שהזכרנו והיא לא קשורה ישירות למתקפה. על כל המתמטיקה ניתן לקרוא [פה](#), [פה](#) ו[פה](#).

נניח שאנו נצפין שתי הודעות Y, Z כך שמתקיים $Y^3 < N$ וגם מתקיים $Z^2 < N < Z^3$

כלומר, Y לא יצטרך הפחתה גם אם נעלה אותו בריבוע ונכפיל בערך המקורי שלו (כלומר נעלה בשלישית) מכיוון שהוא ישאר קטן יותר מעולם המודולו n .

ולעומת זאת, Z לא יצטרך הפחתה רק אם נעלה אותו בריבוע. אבל ברגע שנכפיל אותו אח"כ בערך המקורי שלו (כלומר נעלה בשלישית) הוא יצטרך הפחתה מכיוון שהוא יהיה גדול יותר מעולם המודולו n .

כעת לאחר שאנו שולחים את ההודעות להצפנה, ניתן לחלק את המצב לשני מקרים:

במקרה הראשון – ביט החזקה באלגוריתם Repeated Squaring הוא 1, לכן נצטרך להכפיל את ההודעה המוצפנת בהודעה המקורית ולא רק להעלות אותה בריבוע.

במקרה כזה, משך זמן ההצפנה של הודעה Z ייקח יותר זמן. כי כאשר מעלים את הודעה Z בשלישית היא תהיה גדולה מעולם המודולו n .

לעומת זאת, הודעה Y שלא תהיה גדולה מעולם המודולו n גם כאשר מעלים אותה בשלישית, לא תדרוש הפחתה. לכן זמן ההצפנה שלה יהיה קצר יותר.

במקרה שני – ביט החזקה באלגוריתם Repeated Squaring הוא 0, לכן נצטרך רק להעלות את ההודעה המוצפנת בריבוע, בלי להכפיל את ההודעה המוצפנת בהודעה המקורית.

במקרה כזה זמן ההצפנה של שתי ההודעות יהיה שווה מכיוון ששתיהן לא יצטרכו הפחתה.

כעת, כששלחנו את ההודעות להצפנה ואנו יודעים מהם זמני הריצה שלוקח להצפין אותן, נוכל להשוות בין זמני הריצה של Y ו- Z .

אם זמני ההצפנה הם פחות או יותר שווים, נדע שהביט הבא הוא 0 (כי לא הייתה הפחתה להודעה Z ולכן לא לקח לה יותר זמן). ולעומת זאת אם זמני ההצפנה יהיו שונים, נדע שהביט הבא הוא 1 (כי להודעה Z הייתה הפחתה ולכן זמן ההצפנה שלה ארוך יותר).

את התהליך הזה נעשה שוב ושוב עבור כל הביטים במפתח, עד שנמצא את המפתח הפרטי שמתאים למפתח הציבורי שברשותנו

תשתית למתקפה

כדי שנוכל לבצע את המתקפה בצורה יעילה, נצטרך לבצע את הפעולות שהסברנו עם אלפי הודעות. לשם כך נבנה מערכת תקשורת נוחה עם לוח Arduino שלנו - שהוא מערכת ההצפנה שלנו. את התקשורת איתו נבצע באמצעות סקריפט שנכתוב בפיתון (נשתמש בספריה [PySerial](#)). הסקריפט יתקשר עם הפורט הסריאלי של Arduino. באמצעות כך נשלח ללוח הודעות להצפנה ונקבל ממנו את הערכים המוצפנים. במקביל הקוד ימדוד את זמן ההצפנה במיקרו שניות. עקב מגבלות של לוח Arduino לעבוד עם מספרים של 64 ביט, הוא יקבל את ההודעה להצפנה כמחרוזת של ביטים, ימיר אותה למספר, יצפין אותה. ולאחר מכן ימיר אותה בחזרה למחרוזת של ביטים ויחזיר את הצופן. את שלושת הערכים האלו (ההודעה, ההצפנה והזמן) נשמור בתוך קובץ שלאחר מכן נתבסס עליו לביצוע המתקפה כפי שיוסבר בהמשך. הקוד שלנו יראה כך:

```
def write(data):
    while not arduino.writable():
        pass
    arduino.write(data.encode())

def read():
    while not arduino.readable():
        pass
    data = ""
    while data == "":
        data = arduino.readline().decode()
    return data
```

כפי שניתן לראות, יש לנו פונקציה write שמקלת נתונים ומעבירה אותם ללוח. בנוסף יש לנו פונקציה read שממתינה לנתונים שיחזרו מהלוח.

ניתן למצוא את הקוד [כאן](#).

כפי שהסברנו, בכדי שיהיה חומר לניתוח הזמנים ולמציאת המפתח הפרטי, נשלח אלפי הודעות רנדומליות למערכת ההצפנה שלנו ובמקרה שלנו ללוח Arduino, על מנת שיציין אותן באמצעות המפתח הפרטי ויחזיר לנו את הערך המוצפן. כמובן שעבור כל הודעה נמדוד כמה זמן לקח למערכת להציין אותה.

אחרי שיהיה ברשותנו מאגר של אלפי הודעות, ההצפנות של אותן הודעות וזמן ההצפנה שלהן, נוכל לבצע את הניתוח.

בשלב הראשוני ידוע לנו שהביט הראשון של המפתח הפרטי הוא 1 ואנחנו רוצים לגלות כל פעם מה הביט הבא במפתח.

כעת ניקח את רשימת ההודעות ששלחנו ונפעיל על כל אחת מההודעות את האלגוריתם Repeated squaring עד הביט שידוע לנו. עבור כל אחת מההודעות נבדוק האם הביט הבא בהנחה שהוא 1 יש הפחתה או לא (כלומר, האם היה צורך ב% או לא).

כך נחלק בעצם את הרשימה של אלפי ההודעות שלנו לשתי קבוצות:

קבוצה א – כל ההודעות שבהן היה צורך בהפחתה אם הביט הבא הוא 1.

קבוצה ב – שאר ההודעות. כלומר ההודעות שבהן לא היה צורך בהפחתה אם הביט הבא הוא 1.

כעת נחשב את ממוצע זמן הריצה עבור כל קבוצה לפי הזמנים שמופיעים במאגר שלנו.

לאחר מכן נבדוק, אם הפרש הזמנים בין הקבוצות שחילקנו הינו זניח, המשמעות היא שאין באמת הגיון כלשהו בחלוקה שעשינו (כלומר לחלק מההודעות עשינו הפחתה סתם, ללא צורך. ולכן לקח יותר זמן להציין אותן) ולכן ההשערה שלנו שהביט הוא 1 שגויה והביט הבא הוא 0.

אך לעומת זאת, אם הפרש הזמנים בין הקבוצות שחילקנו אינו זניח אלא משמעותי, ניתן להסיק שיש הגיון בחלוקה שביצענו (כלומר ניתן להסיק שבאמת פעלנו נכון כאשר לקבוצה אחת של הודעות עשינו הפחתה ולקבוצה השנייה לא) ולכן באמת הביט הבא הוא 1 כפי שהנחנו.

בניית מאגר ערכים למתקפה

כעת נעבור לפונקציה הראשית.

הפונקציה הראשית תגריל כמה אלפי מספרים בגדלים שונים, מ 1 ועד n.

הסיבה שנגריל רק עד n היא מכיוון שזה עולם המודולו שלנו. לכן אם נגריל יותר מ-n נבצע הפחתה שלא לצורך - מבלי קשר למתקפה.

עבור כל מספר שהתוכנית הראשית תגריל, היא תשלח אותו להצפנה ותמדוד כמה זמן היא ממתנה לתשובה. לאחר מכן היא תכתוב את כל המידע לתוך קובץ שימש אותנו לניתוח הזמנים ומציאת המפתח הפרטי.

הקוד של התוכנית הראשית יראה כך:

```
n = 3839256683
e = 548447537

with open('timeData.txt', 'w') as f1:
    f1.writelines("N,E\n")
    f1.writelines(str(n) + "," + str(e) + "\n")
    f1.writelines("message,signature,duration\n")
    for i in range(1, 9000):
        m = random.randrange(1, n)
        t1 = datetime.now()
        write(longToBytes(int(m)))
        c2 = stringToInt(read())
        t2 = int((datetime.now() - t1).total_seconds() * 1000000)
        f1.writelines(str(m) + "," + str(c2) + "," + str(t2) + "\n")
```

ניתן לראות בקוד שבתוך הלולאה אנו מגרילים ערך כפי שהסברנו לפני, מצפינים אותו ומקבלים את התשובה. בין לבין אנו מודדים זמנים ואת הכל כותבים בסופו של דבר לתוך הקובץ timeData.txt.

סה"כ נקבל בסוף קובץ עם 9000 שורות שיראה כך:

| | |
|----|---------------------------------|
| 1 | N, E |
| 2 | 3839256683, 548447537 |
| 3 | message, signature, duration |
| 4 | 1825426693, 481823779, 2280893 |
| 5 | 2110686704, 3072422398, 2497166 |
| 6 | 1334428208, 1599528745, 2093178 |
| 7 | 1265195384, 2942524450, 2186568 |
| 8 | 2409436678, 1375084233, 2093802 |
| 9 | 401252935, 2464631800, 2186973 |
| 10 | 1695206886, 2565208005, 2690551 |
| 11 | 2933055133, 329494375, 2593669 |

בתחילת הקובץ שמרנו את המפתחת הציבורי ואת המודול N.

כמו כן בכל שורה ניתן לראות את ההודעה, ההצפנה ואת הזמן במיקרו שניות שלקח למערכת להצפין.

כל הנתונים האלו יישמשו את הסקריפט הבא שלנו שינתח את הקובץ וימצא מהו המפתח הפרטי.

זמן ההכנה של מאגר כזה הוא בערך 3 שעות (תלוי בגודל המפתח וכמה דברים נוספים).

את המאגר הזה ומאגרים נוספים ניתן למצוא [כאן](#).

כעת ניקח את המאגר וננתח אותו באמצעות סקריפט נוסף.

בכל שלב כפי שהסברנו, הסקריפט יניח שהביט הבא הוא 1 וינסה לחלק את ההודעות המוצפנות לפי זה (קבוצה אחת להודעות שהיו צריכות הפחתה וקבוצה שניה להודעות שלא היו צריכות הפחתה בזמן ההצפנה).

כמובן שלצורך כך הסקריפט המפענח יצפין לפי אלגוריתם RSA שממומש עם Repeated Squaring ועם Montgomery Product.

לכן הקוד המצפין שלנו יראה בדיוק כמו שראינו למעלה בלוח ה-Arduino אך עם תוספת של דגל נוסף בשם red שמסמל האם הייתה הפחתה או לא. ככה שביחד עם תוצאת ההצפנה נדע לאיזה קבוצה לסווג את ההודעה:

```
def rsa_sim(m, d, n, k, r, j):
    """
    This function simulate a encrypt/decrypt message m by rsa protocol
    :param m: The message we want to encrypt/decrypt
    :param d: The key
    :param n: The modulo
    :param k: The k for montgomery product
    :param r: The r for montgomery product
    :param j: The bit we want to test if we have reduce or not
    :return: The encrypt/decrypt of message m and True/False if we have reduce or not
    """
    m_ = (m * r) % n
    x_ = (1 * r) % n
    new_d = d[j]
    new_d += '1'
    red = False
    for i in range(0, len(new_d)):
        x_, _ = MontgomeryProduct(x_, x_, n, k, r)
        if new_d[i] == '1':
            x_, red = MontgomeryProduct(m_, x_, n, k, r)
    x, _ = MontgomeryProduct(x_, 1, n, k, r)
    return x, red
```

בריבוע האדום אנו "קובעים" מהו המפתח שעכשיו נעבוד עליו. שזה בעצם שרשור של המפתח שמצאנו עד עכשיו בתוספת ההנחה שהביט הבא הוא 1 כפי שהסברנו.

הפונקציה הבאה מקבלת כל פעם הודעה מתוך המאגר, שולחת לפונקציה `rsa_sim` ומקבלת בחזרה האם התקיימה הפחתה או לא. לפי זה היא מסווגת את ההודעות לשתי קבוצות, `red` אם הייתה הפחתה ו- `nored` אם לא הייתה הפחתה.

כפי שרואים בקוד:

```
def split_messages(d, n, k, r, bit, dataList):  
    """  
    This function split the messages to two groups. reduce in bit(bit) or not  
    :param d: The key  
    :param n: The modulo  
    :param k: The k for montgomery product  
    :param r: The r for montgomery product  
    :param bit: The bit we want to test if we have reduce or not  
    :param dataList: The messages  
    :return: two groups. reduce in bit(bit) or not  
    """  
    red = []  
    nored = []  
    for m in dataList:  
        c, bucket = rsa_sim(m[0], d, n, k, r, bit)  
        if bucket:  
            red.append(m)  
        else:  
            nored.append(m)  
    return red, nored
```

הפונקציה הבאה היא הפונקציה החשובה שלנו ולכן נעבור עליה שורה אחר שורה. היא נקראת `main` ותפקידה לעבור על המאגר ולמצוא את המפתח בפועל.

```

139 def RSATimingAttack(n, data, ratio):
140     """
141     This function analysis the time that took to encrypt the messages and find the private key
142     :param n: The modulo
143     :param data: the messages with the time
144     :param ratio: the delta
145     """
146     (r, k) = nPrime(n)
147     private_key = '1'
148     bit = 1
149     finished = False
150     while not finished and bit < 32:
151         (red, nored) = split_messages(private_key, n, k, r, bit, data)
152         red_avg = calcAvg([m[2] for m in red])
153         nored_avg = calcAvg([m[2] for m in nored])
154         print("Difference of averages:", abs(red_avg - nored_avg))
155
156         if abs(red_avg - nored_avg) > ratio:
157             private_key += '1'
158             print("The next bit is probably 1.")
159         else:
160             private_key += '0'
161             print("The next bit is probably 0.")
162         print("The private key until now is: ", private_key)
163         print()
164
165         if testKey(data, private_key, n, k, r):
166             print("We did it! The private key is: \t", private_key)
167             finished = True
168
169         bit += 1
170
171     if not finished:
172         print("can't find the private key...")

```

בשורה 139 ניתן לראות שהפונקציה מקבלת שלושה פרמטרים:

n – עולם המודולו.

data – מאגר ההודעות שאותן.

ratio – טווח הסטייה שלפיו נקבע האם צדקנו בהנחה שהביט הוא 1 או שהביט צריך להיות 0. בסוף נצלול אל השימוש בו וכיצד קובעים אותו.

לאחר מכן ב-146 נכין את המשתני עזר ל-Mongomery Product באמצעות nPrime כפי שכבר ראינו לפני.

בשורה הבאה נאתחל את המשתנה שישמור את המפתח הפרטי לאורך הדרך ושורה אחרי נאתחל מונה שסופר כמה ביטים כבר עברנו.

לאחר מכן בשורה 170 - 150 נכנס ללולאה שעובדת עד שנמצא את המפתח הפרטי (או עד שנגיע ל-32 ביט – כדי לדעת שמשו לא עובד).

בתוך הלולאה בשורה 151 קודם כל נסווג את ההודעות לשתי קבוצות כפי שהסברנו, קבוצה red שלה הייתה הפחתה ו קבוצה nored שלה לא הייתה הפחתה.

כעת בשורות 152 – 153 נחשב את ממוצע הזמנים עבור כל קבוצה ולאחר מכן בשורה 156 נבדוק אם יחס הזמנים בין הקבוצות לא זניח (גדול מration) ולכן אכן הביט הוא 1 (שורה 157) או שהוא זניח (קטן מration) ולכן הביט הוא 0 (שורה 160).

כיצד לקבוע את ratio נסביר בהמשך.

לבסוף בשורה 165 אנו נפעיל את הפונקציה testKey שתבדוק האם צדקנו במפתח עד עכשיו ואם כן – נעצור את הלולאה.

הפונקציה testKey פשוט לוקחת מספר ערכים מהמאגר, מצפינה אותם עם המפתח שמצאנו ומחזירה האם התוצאה נכונה או לא לפי מה ששמור במאגר.

כעת נריץ את הסקריפט ונראה כיצד הוא מוצא את המפתח הפרטי.

בדוגמת ההרצה שלנו הration יהיה 150000.

את הקוד נריץ על המאגר שבנינו לפני בפרק של [בניית מאגר ערכים למתקפה](#).

| | |
|----|-------------------------------|
| 1 | N,E |
| 2 | 3839256683,548447537 |
| 3 | message,signature,duration |
| 4 | 1825426693,481823779,2280893 |
| 5 | 2110686704,3072422398,2497166 |
| 6 | 1334428208,1599528745,2093178 |
| 7 | 1265195384,2942524450,2186568 |
| 8 | 2409436678,1375084233,2093802 |
| 9 | 401252935,2464631800,2186973 |
| 10 | 1695206886,2565208005,2690551 |
| 11 | 2933055133,329494375,2593669 |

נזכיר שהמאגר מכיל נתוני הצפנה של 9000 הודעות.

כעת נריץ את הסקריפט (זמן הריצה הוא בערך חצי דקה) ונקבל המון שורות כאלה:

The first bit is 1

Difference of averages: 187986.2268589628

The next bit is probably 1.

The private key until now is: 11

Difference of averages: 173582.78072837694

The next bit is probably 1.

The private key until now is: 111

Difference of averages: 137093.0452079922

The next bit is probably 0.

The private key until now is: 1110

עד שלבסוף נקבל:

Difference of averages: 206346.6489106221

The next bit is probably 1.

The private key until now is: 11100100110101001000010001010001

We did it! The private key is: 11100100110101001000010001010001

נסביר מה קיבלנו:

ניתן לראות שבכל שלב הסקריפט ידפיס לנו שלוש שורות.

בשורה הראשונה ההפרש בין ממוצע הזמנים עבור שתי הקבוצות ולאחר מכן בשורה הבאה האם הביט הוא 0 או 1.

מזכיר שזה נקבע לפי הערך ratio (שבמקרה שלנו הוא 150000) ותלוי האם יחס ממוצע הזמנים בין הקבוצות לא זניח (גדול מration) ולכן אכן הביט הוא 1 או שהוא זניח (קטן מration) ולכן הביט הוא 0.

בשורה הבאה הסקריפט ידפיס את המפתח שהוא מצא עד כה.

לבסוף, בשלב מסוים, הפונקציה testKey זיהתה שמצאנו את המפתח הפרטי ולכן היא עצרה את הלולאה והודפסה לנו ההודעה על סיום הפעולה והמפתח:

We did it! The private key is: 11100100110101001000010001010001

קביעת הערך ratio

קביעת הערך ratio שישמש את הסקריפט תוך כדי ניתוח הזמנים על מנת לקבוע האם הפרש הזמנים זניח או לא זאת פעולה מסובכת .

הערך ratio נקבע לפי חישובים מתמטיים שמתחשבים בזמני הריצה של המערכת, זמן ביצוע הצפנה, זמן ביצוע הפחתה, המהירות של המערכת (אצלנו גם השימוש בלוח Arduinon שמשפיע מאוד על המהירות) וכו'.

באופן כללי ניתן לומר שזה תלוי בכמה זמן בערך לוקח לנו להצפין ערך ולקבל את התוצאה.

כמובן שבמקום זה אפשר להריץ את הקוד מספר פעמים, בכל פעם עם ערך ratio אחר עד שנמצא את הערך הנכון (כשנמצא את המפתח הפרטי). אומנם זה ייקח קצת זמן, אבל בסוף מדובר על 5 דקות של ריצה במקום חצי דקה וזה לא משמעותי.

דרך נוספת היא להתבונן בנתונים ולנסות להסיק מהם מהו הערך הנכון.

כך נעשה את זה:

קודם כל, נציב בratio 0 ונריץ שוב את המערכת, כמובן שהיא תיכשל במציאת המפתח:

```
can't find the private key...
```

אבל, נסתכל על הזמנים שהודפסו לנו וננסה להסיק מהם מהו הערך ratio שלנו.

ננסה לזהות שני קבוצות של זמנים, קבוצה אחת עם ערכים גבוהים יותר וקבוצה שניה עם ערכים נמוכים יותר, את הערך ratio נקבע איפשהו ביניהם.

The first bit is 1

Difference of averages: 187986.2268589628
The next bit is probably 1.
The private key until now is: 11

Difference of averages: 173582.78072837694
The next bit is probably 1.
The private key until now is: 111

Difference of averages: 137093.0452079922
The next bit is probably 1.
The private key until now is: 1111

Difference of averages: 147840.38634238113
The next bit is probably 1.
The private key until now is: 11111

Difference of averages: 133634.03459492
The next bit is probably 1.
The private key until now is: 111111

Difference of averages: 141838.72920653317
The next bit is probably 1.
The private key until now is: 1111111

Difference of averages: 127732.18294484867
The next bit is probably 1.
The private key until now is: 11111111

Difference of averages: 130305.78003233159
The next bit is probably 1.
The private key until now is: 111111111

Difference of averages: 137619.8641706151
The next bit is probably 1.
The private key until now is: 1111111111

Difference of averages: 139925.38081441494
The next bit is probably 1.
The private key until now is: 11111111111

Difference of averages: 144675.34778241627
The next bit is probably 1.
The private key until now is: 111111111111

Difference of averages: 145315.79071280127
The next bit is probably 1.
The private key until now is: 1111111111111

Difference of averages: 137270.6971478518
The next bit is probably 1.
The private key until now is: 11111111111111

Difference of averages: 136953.3723948421
The next bit is probably 1.
The private key until now is: 111111111111111

Difference of averages: 140107.39883236773
The next bit is probably 1.
The private key until now is: 1111111111111111

Difference of averages: 131728.17272960348
The next bit is probably 1.
The private key until now is: 11111111111111111

Difference of averages: 140500.97501439275
The next bit is probably 1.
The private key until now is: 111111111111111111

Difference of averages: 135371.55694211507
The next bit is probably 1.
The private key until now is: 1111111111111111111

Difference of averages: 142729.97881431598
The next bit is probably 1.
The private key until now is: 1111111111111111111

Difference of averages: 132948.27755187592
The next bit is probably 1.
The private key until now is: 11111111111111111111

Difference of averages: 134367.15433216142
The next bit is probably 1.
The private key until now is: 111111111111111111111

Difference of averages: 145426.18276158022
The next bit is probably 1.
The private key until now is: 1111111111111111111111

Difference of averages: 131342.06003474537
The next bit is probably 1.
The private key until now is: 11111111111111111111111

Difference of averages: 136155.43027170328
The next bit is probably 1.
The private key until now is: 111111111111111111111111

Difference of averages: 130312.2138843555
The next bit is probably 1.
The private key until now is: 1111111111111111111111111

Difference of averages: 132867.6554789669
The next bit is probably 1.
The private key until now is: 11111111111111111111111111

Difference of averages: 126829.95983718289
The next bit is probably 1.
The private key until now is: 111111111111111111111111111

Difference of averages: 138589.06965380115
The next bit is probably 1.
The private key until now is: 1111111111111111111111111111

Difference of averages: 147822.24046116136
The next bit is probably 1.
The private key until now is: 11111111111111111111111111111

Difference of averages: 135589.44690011628
The next bit is probably 1.
The private key until now is: 111111111111111111111111111111

Difference of averages: 122560.57186330017
The next bit is probably 1.
The private key until now is: 11111111111111111111111111111111

can't find the private key...

באדום ניתן לראות שעם 0 ratio הסקריפט נכשל.

אם נביט בזמנים שהודפסו, נראה שיש לנו קבוצה נמוכה יותר של זמנים שהם באזור 130000 וקבוצה של זמנים יותר גבוהים באזור 15000.

לכן נקבע את ratio להיות איפשהו באמצע, למשל 140000.

נריץ את הסקריפט שוב ונראה שגם הוא נכשל:

```
def main():  
    difference = 140000  
    RSATimingAttack() > if not finished  
    findTheKey x  
    The next bit is probably 0.  
    The private key until now is: 11100100110110100000101100000010  
    can't find the private key...
```

באדום ניתן לראות את הערך ratio ואת ההודעה על כך שהמפתח הפרטי לא נמצא.

לכן נחזור לזמנים וננסה לנתח אותם שוב בצורה חדה יותר.

אם נבחן את הזמנים שהודפסו שוב, נראה שלא דייקנו מספיק בחלוקה שלנו.

ניתן לראות שאומנם יש לנו קבוצה נמוכה יותר של זמנים שהם באזור 130000, אך הקבוצה של הזמנים היותר גבוהים היא באזור 160000 ולא באזור 15000.

לכן נקבע את ratio להיות שוב איפשהו באמצע, הפעם 150000.

כמובן שניתן לחלק את הזמנים לשתי קבוצות ולמצוא את האמצע ביניהם עם אלגוריתם שזה תפקידו (למשל knee) אבל ניתן להסתדר בלי זה.

נריץ את הסקריפט שוב ונקבל את הפתח הפרטי. כלומר צדקנו בניחוש הערך ratio:

```
def main():  
    difference = 150000  
    RSATimingAttack()  
    findTheKey x  
    The private key until now is: 11100100110101001000010001010001  
    We did it! The private key is: 11100100110101001000010001010001
```

באדום ניתן לראות את הערך ratio ואת המפתח הפרטי וההודעה על כך שהוא נמצא.

הצלחנו לנחש את הערך ratio.

סיכום

בפרויקט זה עסקנו במתקפת ערוץ צד.

כפי שהראנו, ניתן לבצע מתקפת ערוץ צד על RSA וע"י כך למצוא את המפתח הפרטי.

בנוסף הראנו כיצד ניתן לתקוף מערכות כמו קוד לבניין ולמצוא את הסיסמה בקלות - תוך מספר מועט של ניסיונות.

במהלך הפרויקט התנסנו בשימוש בלוח Arduino ולוח Tiva. עליהם ממשנו מערכות שונות כמו הקוד לבניין או מערכת הצפנת RSA, אותם תקפנו.

הפרויקט היה חוויתי ומהנה והעמיד אותנו בהתמודדות מול התגרים חדשים ושונים שלא התנסנו בהם בעבר.

האתגרים היו בעיקר ההתעסקות עם לוחות Arduino. כאשר היה עלינו ללמוד כיצד לחבר את הרכיבים השונים כמו כפתורים, לדים נגדים ועוד בצורה נכונה.

אתגר נוסף שאיתו התמודדנו הוא מימוש מתקפה על RSA. מדובר במתקפה מורכבת, ברמה גבוהה הרבה ממה שהתנסינו עד היום. היה עלינו ללמוד כיצד מערכות RSA ממומשות וכיצד לנצל את נקודת התורפה של צורת המימוש על מנת לבצע מתקפת ערוץ צד ולמצוא את המפתח הפרטי.

בנוסף, תוך כדי העבודה נתקלנו באתגרים נוספים "קטנים יותר" שהיה עלינו לפתור כמו State Change Detection, מגבלת גודל המשתנים ב-Arduino ועוד.

זמן הבניה של הפרויקט כולל המחקר וקתיבת הדוח הוא בערך 150 שעות. כאשר חלק מהזמן עבדנו במקביל על חלקים שונים מהפרויקט וחלק מהזמן עבדנו ביחד.

אין ספק שהפרויקט לקח אותנו צעד קדימה והעשיר אותנו בידע רב ובכלים רבים להתמודדות עם אתגרים.

מקורות

1. <https://www.tinkercad.com/dashboard>
2. <https://create.arduino.cc/projecthub/rowan07/make-a-simple-led-circuit-ce8308>
3. <https://il.farnell.com/resistor-colour-code-calculator>
4. <https://www.arduino.cc/en/Tutorial/BuiltInExamples/Button>
5. <https://www.the-diy-life.com/multiple-push-buttons-on-one-arduino-input/>
6. <https://security.stackexchange.com/questions/96489/can-i-prevent-timing-attacks-with-random-delays>
7. https://en.wikipedia.org/wiki/Modular_arithmetic
8. <https://www.cs.sjsu.edu/faculty/stamp/students/article.html>
9. http://www.mat.ucm.es/congresos/mweek/XII_Modelling_Week/Informes/Report5.pdf
10. https://en.wikipedia.org/wiki/Exponentiation_by_squaring
11. https://en.wikipedia.org/wiki/Montgomery_modular_multiplication
12. https://en.wikipedia.org/wiki/Modular_exponentiation
13. <https://pyserial.readthedocs.io/en/latest/pyserial.html>
14. <https://forum.arduino.cc/t/arbitrary-precision-big-number-library-port-for-arduino/84007>
15. <https://create.arduino.cc/projecthub/ansh2919/serial-communication-between-python-and-arduino-e7cce0>
16. <https://slideplayer.com/slide/4976535/>
17. https://images.slideplayer.com/16/4976535/slides/slide_6.jpg
18. https://en.wikipedia.org/wiki/%D7%94%D7%AA%D7%A7%D7%A4%D7%AA_%D7%A2%D7%A8%D7%95%D7%A5_%D7%A6%D7%93%D7%93%D7%99
19. https://en.wikipedia.org/wiki/Side-channel_attack
20. <https://www.digitalwhisper.co.il/files/Zines/0x4F/DW79-2-SideChannel.pdf>
21. <https://github.com/avifeder/SideChannelAttack>