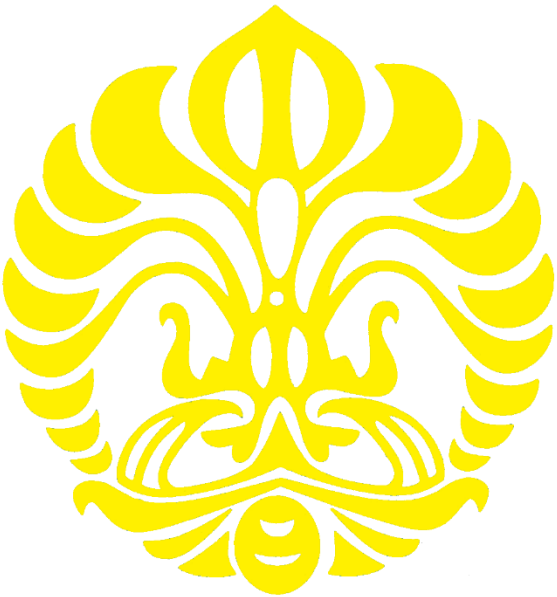
AVR HASH KEY GENERATOR USING LIGHT INTENSITY

Disusun guna memenuhi Tugas Akhir mata kuliah

Sistem Siber-Fisik

Dosen Pengampu:

F. Astha Ekadiyanto., S.T., M.Sc.



Oleh:

Evandita Wiratama Putra / 2206059572

Teknik Komputer

Departemen Teknik Elektro

Fakultas Teknik Universitas Indonesia

DAFTAR ISI

BAB I
PENDAHULUAN

1.1 Abstrak

Sebagai bentuk pelaksanaan tugas akhir semester mata kuliah Sistem Siber Fisik, saya mengusulkan sebuah solusi rangkaian AVR yang dapat diimplementasikan pada bidang **Keamanan Jaringan**, yakni *Hash Key Generator using Light Intensity*.

Cara kerja dari rangkaian ini adalah dengan memanfaatkan microcontroller Arduino Uno untuk membaca besaran intensitas cahaya melalui perangkat sensor cahaya LDR (*Light Dependent Resistor*), dan menggunakan hasil konversi ADC (*Analog to Digital Converter*) dari nilai tersebut untuk menghasilkan sebuah *Hash Key*.

Selebihnya, AVR juga memiliki output yang terdiri dari beberapa LED untuk menghasilkan gangguan pada pembacaan LDR agar *Hash Key* yang dihasilkan lebih sulit untuk diprediksi.

1.2 Latar Belakang

Hashing adalah sebuah konsep pada bidang keamanan jaringan yang seringkali digunakan karena sangat bermanfaat dalam menjaga data yang bersifat sensitif. Dengan dilakukannya *hashing*, maka data yang tersimpan akan terenkripsi secara satu arah, sehingga mampu menambahkan lapisan keamanan.

Akan tetapi, hal ini belum menjamin data tersebut 100% aman. Hal ini dikarenakan algoritma dari fungsi matematis yang digunakan untuk *hashing* mayoritas masih diprogram pada sistem semua, sehingga apabila perentas data berhasil

mempelajari cara kerja sistem tersebut, maka seluruh data yang telah di-hashing dapat dipredeksi secara perlahan.

Hal ini juga belum melibatkan faktor kekuatan algoritma *hashing* itu sendiri, di mana semakin lemah algoritma yang digunakan, maka semakin mudah pula data untuk diprediksi. Semakin kuat algoritma *hashing* juga belum tentu lebih baik karena akan memberikan beban tambahan pada CPU juga.

1.3 Pendekatan Solusi

Untuk memperkuat algoritma *Hashing* yang sudah ada, maka salah satu pendekatan yang dapat dilakukan adalah dengan melibatkan sebuah variabel dengan nilai yang sangat sulit untuk diprediksi, untuk proses hashing tersebut. Di sinilah tempat di mana nilai besaran fisik mulai berperan.

Nilai acak yang dihasilkan oleh program, tentu berbeda dengan nilai acak yang didapat dari hasil pengambilan data pada dunia nyata. Hal ini dikarenakan pada dunia nyata, banyak sekali faktor yang memengaruhi besaran tersebut, sehingga sangat sulit untuk memprediksi nilainya. Di sisi lain, nilai yang dihasilkan oleh program, cenderung lebih mudah untuk diprediksi karena hanya seseorang programmer yang menulis seluruh aturan penentuan besaran nilai acak, sehingga faktor yang terlibat umumnya masih terbatas. Selain itu, penggunaan nilai fisik juga mengurangi beban CPU dalam proses hashing karena tidak perlu menjalankan program untuk menhasilkan nilai acak.

Selebihnya, pendekatan solusi ini merupakan bentuk inspirasi dari penggunaan *lava lamp* sebagai alat bantu enkripsi data yang diterapkan oleh perusahaan jaringan besar bernama *cloudflare*. Meskipun begitu, metode yang dilakukan oleh perusahaan tersebut sangat kompleks karena sudah melibatkan pembacaan indera untuk

mengambil “nilai acak” dari *lava lamp* tersebut, sehingga solusi *AVR Hash Key Generator Using Light Intensity* dilahirkan sebagai bentuk alternatif yang lebih sederhana dengan biaya yang lebih terjangkau, serta masih menerapkan konsep yang sama.

BAB II
IMPLEMENTASI DAN RANCANGAN

Untuk merealisasikan penjabaran dari pendekatan solusi yang telah disinggung sebelumnya, maka rancangan serta bentuk implementasi desain rangkaian perlu dipertimbangkan terlebih dahulu.

2.1 Cara Kerja Rangkaian

Tujuan utama yang ingin diselesaikan dari permasalahan ini adalah melakukan *hashing* pada data yang diberikan dengan memanfaatkan sumber daya CPU sekaligus sumber daya fisik. Pada solusi yang ditawarkan, mikrokontroller *Arduino Uno* akan bertindak sebagai sistem yang mengeksekusi algoritma *hashing*, sedangkan besaran intensitas cahaya akan berperan sebagai penyedia data yang acak untuk mempersulit prediksi data pada hasil *hashing*.

Rancangan tersebut sekilas terlihat siap untuk digunakan, namun perlu diperhatikan bahwa pada kenyataannya, intensitas cahaya bukanlah nilai yang dapat bervariasi dalam jangka waktu yang singkat. Dengan kata lain, kemungkinan besar nilai intensitas cahaya yang terekam akan konsisten secara linier seiring berjalannya waktu. Kemungkinan terdapat nilai yang bervariasi hanya terjadi pada waktu-waktu tertentu saja, seperti malam hari, hujan, dsb.

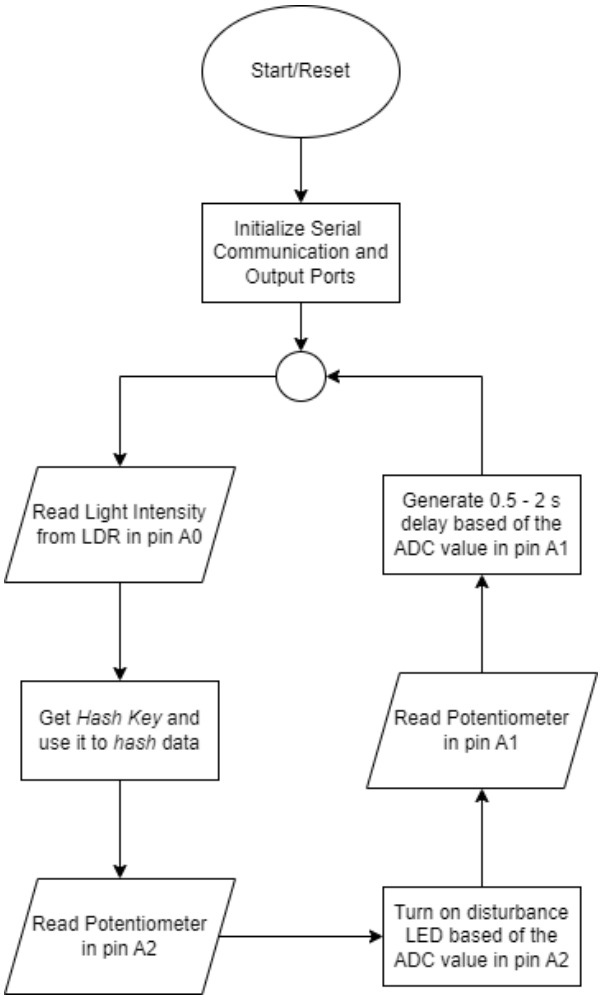
Untuk mengatasi permasalahan tersebut, maka dapat dilibatkan faktor tambahan yang mampu mempengaruhi nilai intensitas cahaya, seperti melibatkan lampu led disekitar LDR yang dapat dinyala/matikan melalui mikrokontroller *Arduino Uno*. Dengan adanya lampu led ini, maka apabila dirasa pada suatu saat intensitas cahaya yang diukur kurang bervariatif (mudah diprediksi), maka sebuah gangguan (*disturbance*) pada nilai intensitas cahaya dapat dihasilkan melalui lampu led yang

telah dipasang disekitar sensor cahaya LDR, serta besar/kecil-nya gangguan dapat diatur melalui input *potentiometer* pada *Arduino Uno*.

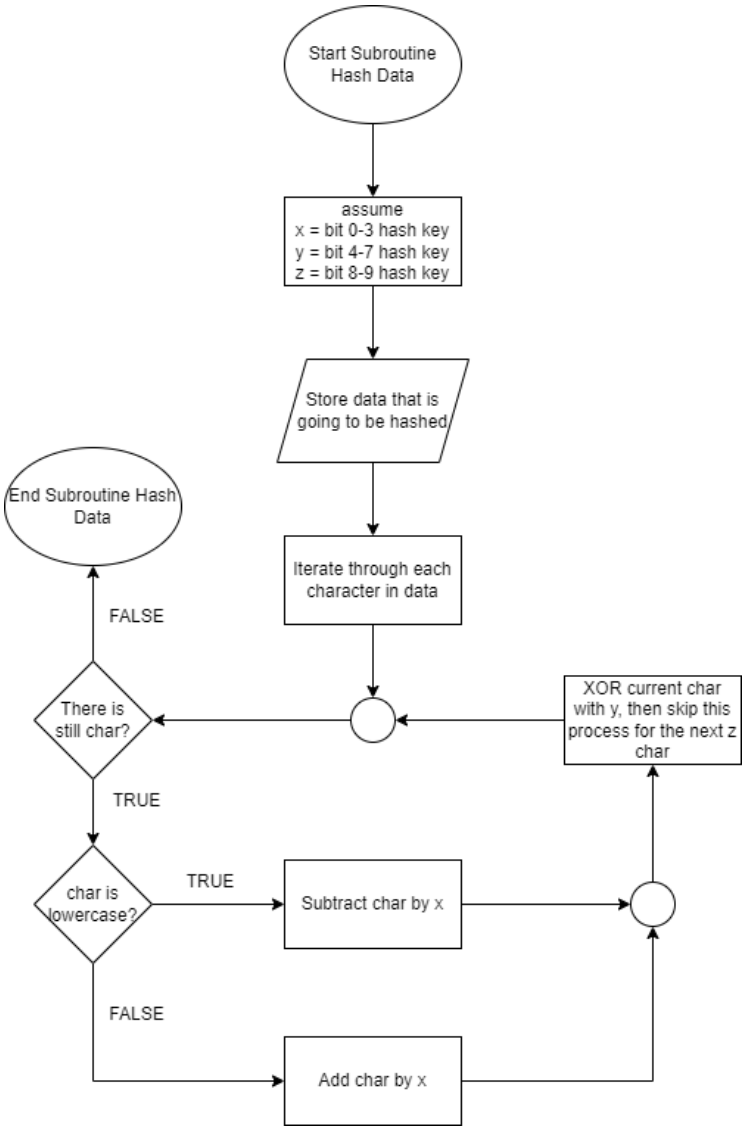
Selebihnya, untuk memudahkan penyimpanan data *hash key* pada *database* secara fleksibel, maka periode penghasilan *hash key* dapat pula diatur. Untuk model rancangan saat ini, periode pembuatan *hash key* akan berada pada rentang 0.5 – 2 detik. Nilai ini diperoleh melalui input *potentiometer* juga yang terdapat di *Arduino Uno*.

2.2 Alur Kerja Rangkaian

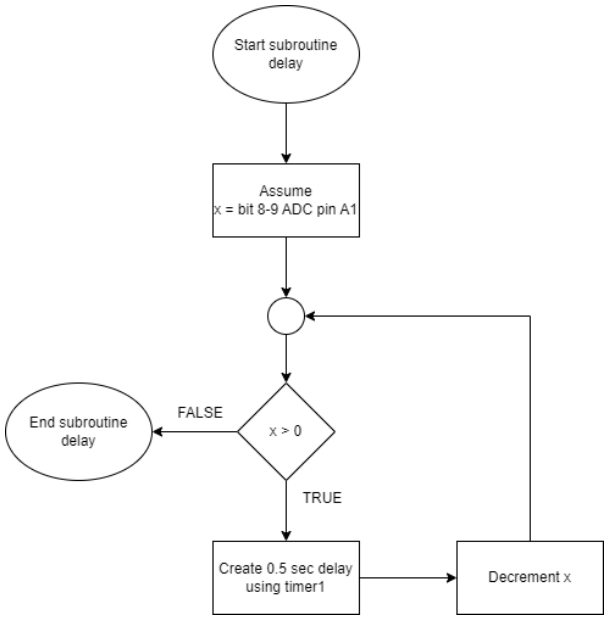
Secara garis besar, rancangan rangkaian akan memiliki 3 macam input nilai (LDR, Potentiometer untuk LED, Potentiometer untuk delay) dan 2 macam output nilai (Hash Key + Hashed Data, disturbance LED). Dikarenakan seluruh input nilai bersifat kontinu dan bukan diskrit, maka akan menggunakan port analog (A0, A1, A2) pada *Arduino Uno* yang nantinya akan diubah menjadi nilai diskrit melalui proses konversi ADC. Lalu untuk output, hasil *Hash Key* beserta *Hashed Data* akan ditampilkan pada serial monitor melalui protokol USART, sedangkan untuk *disturbance led* dapat menggunakan pin digital pada PORTD *Arduino Uno*. Berikut merupakan bentuk representasi *flowchart* untuk cara AVR menerima data, mengolah data, dan menampilkan data:



Rancangan rangkaian akan menampilkan contoh hasil *hashed data* menggunakan nilai dari *hash key* yang bervariatif, sehingga dibutuhkan sebuah algoritma *hash* yang tertanam pada mikrokontroller *Arduino Uno*. Agar tidak membebani CPU mikrokontroller, maka algoritma *hash* yang digunakan masih tergolong sederhana (meskipun hal ini akan menimbulkan celah keamanan). Alur kerja dari algoritma *hashing* yang diimplementasikan pada *Arduino Uno* antara lain sebagai berikut:



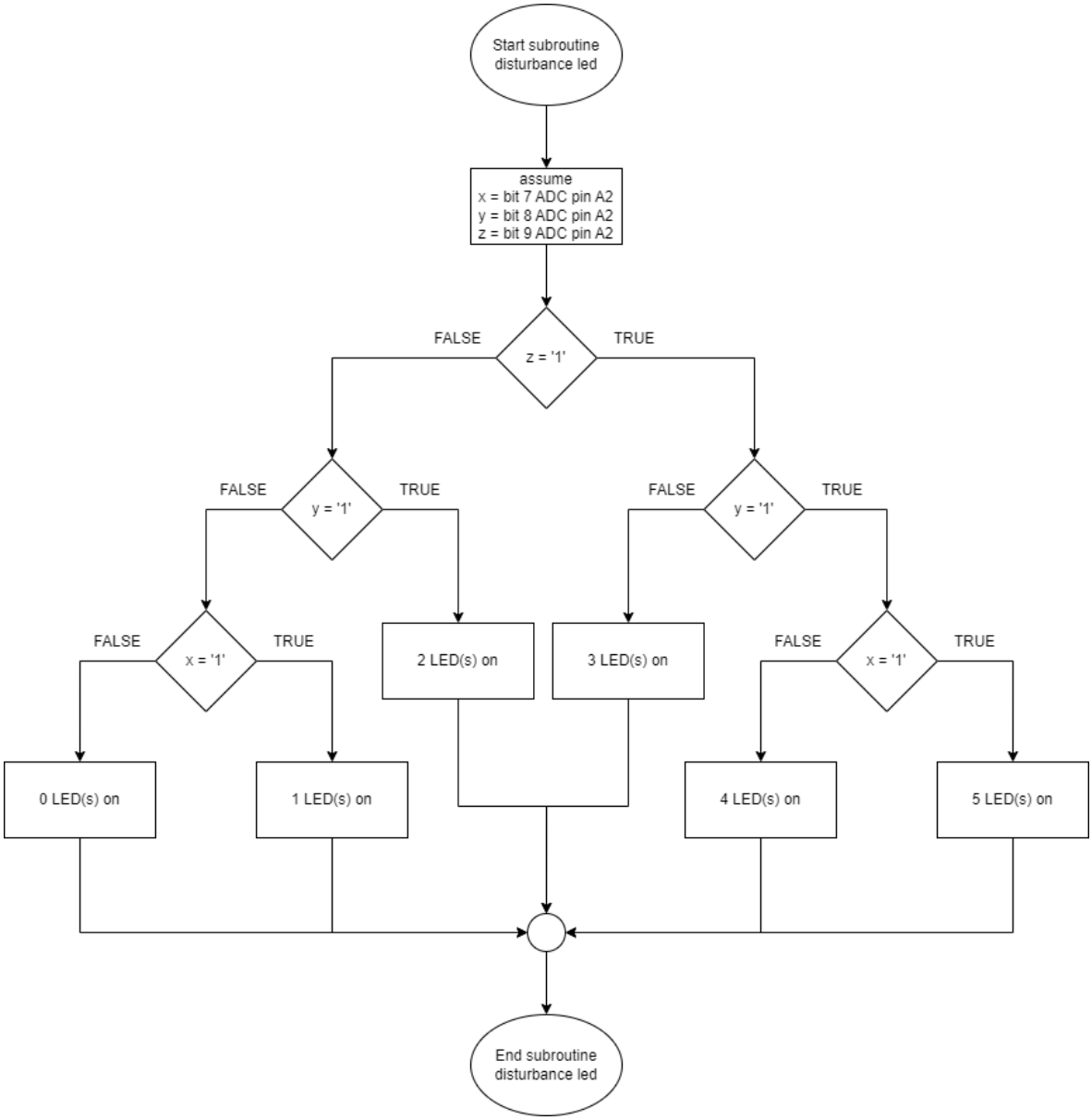
Selain perlunya algoritma *hashing*, sempat disinggung juga bahwa terdapat mekanisme pada rangkaian yang dapat menghasilkan delay pada rentang 0.5 detik sampai 2 detik melalui input *potentiometer* yang tersampung pada pin A1 di *Arduino Uno*. Sinyal analog yang diterima oleh pin A1 akan dikonversi menjadi nilai diskrit melalui proses ADC dengan interval nilai 0 – 1024. Nilai tersebut kemudian diolah sedemikian rupa sehingga mampu menghasilkan delay yang diinginkan memiliki alur kerja sebagai berikut:



Selebihnya, terdapat juga input analog untuk *disturbance led* pada pin A2. Hasil konversi ADC dari input analog tersebut nantinya digunakan untuk menyalakan 0 sampai 5 LED yang terletak didekat LDR untuk menciptakan gangguan pada intensitas cahaya yang diterima. Berikut merupakan tabel pemetakan nilai interval ADC ke jumlah LED yang menyala:

ADC Value	**ON LEDs**
00 0000 0000 – 00 0111 1111	0
00 1000 0000 – 00 1111 1111	1
01 0000 0000 – 01 1111 1111	2
10 0000 0000 – 10 1111 1111	3
11 0000 0000 – 11 0111 111	4
11 1000 0000 – 11 1111 1111	5

Dapat diihat bahwa dengan nilai ADC yang berada pada interval 0 – 1024, dapat dibagi menjadi 6 kemungkinan kondisi dari *disturbance led.* Tujuan dari adanya beberapa LED sekaligus yang dapat diatur jumlah menyalanya adalah untuk menyesuaikan besar/kecilnya gangguan intensitas cahaya yang ingin diberikan. Berikut merupakan alur kerja rangkaian agar dapat mengimplementasikan tabel pemetakan di atas:

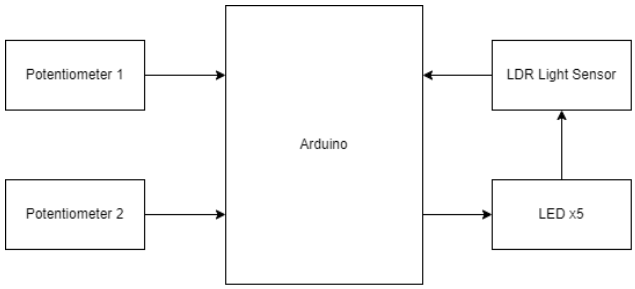


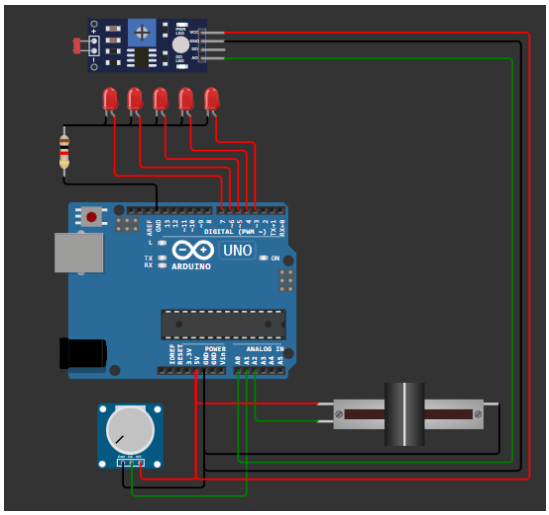
2.3 Desain Rangkaian

Berdasarkan penjelasan rancangan rangkaian yang telah dibahas sebelumnya, dapat disimpulkan bahwa perkiraan komponen yang akan dibutuhkan untuk mengimplementasikan rangkaian asli antara lain sebagai berikut:

No	Komponen	Penjelasan
1	Arduino Uno	Mikrokontroller Rangkaian
2	Jumper	Kabel untuk menghubungkan antar komponen
3	Resistor	Mengatur arus pada LED
4	LED (5)	Untuk memberikan gangguan pada input LDR
5	LDR	Untuk mendapatkan besaran intensitas cahaya
6	Potentiometer 1	Untuk mengatur besar/kecil-nya delay
7	Potentiometer 2	Untuk mengatur besar/kecil-nya gangguan

Apabila seluruh komponen tersebut dirakit, maka kurang lebih akan didapatkan hasil sebagai berikut:





Pada gambar di atas dapat dilihat bahwa sensor cahaya LDR terletak tepat di sebelah *disturbance led* untuk memaksimalkan efek gangguan yang dapat diberikan. Selain itu, terdapat juga 2 jenis *potentiometer* yang digunakan. *Regulator Potentiometer* yang terletak di sebelah kiri berguna untuk mengatur delay, sedangkan *Slide Potentiometer* yang terletak di sebelah kanan berguna untuk mengatur gangguan. Selebihnya, rangkaian tersebut dirancang pada wokwi.com dan dapat disimulasikan melalui tautan berikut:

Link: https://wokwi.com/projects/400464062500201473

2.4 Program Assembly

2.4.1 Main Program

Main Program berisikan kode Assembly yang digunakan untuk melakukan *setup* serta *loop* seperti yang terjadi pada umumnya untuk program C pada AVR. Alur kerja dari main program sudah disinggung sebelumnya menggunakan *flowchart*. Berikut merupakan kode Asssembly untuk *Main Program*:

```
;====================================================================
main:
;--------------------------------------------------------------------
; This is the assembly code to turn ADC value of LDR into hash key
; The hash key will later be used to hash the given data (Data is stored in the last line of this program)
; Potentiometer in pin A01 can be used to determine the period of hash key generated every 0.5 - 2 sec
; There is also a second potentiometer that is connected to pin A02, which is used for disturbance led
; Disturbance leds are useful to manipulate light intensity with human interference
; This makes the hashed key more unpredictable
;
; Author: Evandita Wiratama Putra
; Student Id: 2206059572
;
;--------------------------------------------------------------------
    RCALL init_serial                   ; Initialize Serial Communication
    RCALL init_disturbance_led          ; Set all port B to output for led
main_loop:
    RCALL init_ADC0                     ; Initialize pin A0 for analog input
    RCALL read_ADC                      ; Get analog value of LDR in pin A0

    RCALL print_key                     ; Print the hash key of ADC
    RCALL hash_data                     ; Print the hashed data using the key from LDR

    RCALL init_ADC2                     ; Initialize pin A1 for analog input
    RCALL read_ADC                      ; Get analog value of Potentiometer in pin A2
    RCALL display_disturbance_led       ; Set disturbance led based on the ADC value

    RCALL init_ADC1                     ; Initialize pin A1 for analog input
    RCALL read_ADC                      ; Get analog value of Potentiometer in pin A1

    RCALL delay                         ; 0.5 - 2 second delay depending of Potentiometer ADC value
    rjmp main_loop
;====================================================================
```

2.4.2 Init Serial

Blok program *Init Serial* digunakan untuk melakukan inisialisasi protokol USART untuk komunikasi serial antara *Arduino Uno* dengan *Serial Monitor* pada program Assembly. Spesifikasi yang diharapkan untuk inisialisasi komunikasi serial antara lain sebagai berikut:

1. Baud Rate 9600
2. Mode operasi Asinkron
3. Tidak ada Parity Bit

Untuk mendapatkan besaran Baud Rate sebesar 9600, maka perlu mengisi Register UBBR0 dengan nilai yang diperoleh dari perhitungan berikut:

$$UBBR = \frac{CPU_{frequency}}{16 \times Baud_{rate}} - 1$$

$$UBBR = \frac{16 \times 10^6}{16 \times 9600} - 1 = 103$$

Maka, konfigurasi yang perlu dilakukan dalam kode Assembly untuk memenuhi seluruh persyaratan pada tahap inisialisasi di atas antara lain sebagai berikut:

```
;=====================================================================
init_serial:
;---------------------------------------------------------------------
; Uses R24 temporarily
; Initialisz UART Serial Communication Protocol for printing key and hashed data
;---------------------------------------------------------------------
    CLR   R24
    STS   UCSR0A, R24                ;clear UCSR0A register
    STS   UBRR0H, R24                ;clear UBRR0H register
    LDI   R24, 103                   ;& store in UBRR0L 103 value
    STS   UBRR0L, R24                ;to set baud rate 9600
    LDI   R24, 1<<RXEN0 | 1<<TXEN0   ;enable RXB & TXB
    STS   UCSR0B, R24
    LDI   R24, 1<<UCSZ00 | 1<<UCSZ01  ;asynch, no parity, 1 stop, 8 bits
    STS   UCSR0C, R24
    RET
;=====================================================================
```

2.4.3 Init Disturbance LED

Pada tahap inisialisasi *disturbance led*, maka hal yang perlu dilakukan hanyalah mengkonfigurasi agar seluruh Port D menjadi pin Output. Berikut merupakan kode Assembly yang digunakan untuk merealisasikan hal tersebut

```
;=====================================================================
init_disturbance_led:
;---------------------------------------------------------------------
; Uses R16 temporarily
; Set all pin D to output pins
;---------------------------------------------------------------------
    LDI   R16, 0xff
    OUT   DDRD, R16
    RET
;=====================================================================
```

2.4.4 Init ADC0/ADC1/ADC2

Pada blok program ADC0/ADC1/ADC2 sebenarnya memiliki konfigurasi yang sama semuanya, hanya saja letak perbedaannya terdapat pin Analog yang ingin dipakai untuk input analog pada saat itu. Blok program ADC0 menginisialisasi pin analog A0, ADC1 menginisialisasi pin analog A1, dan ADC2 menginisialisasi pin analog A2. Adapun spesifikasi dari proses konversi ADC untuk masing-masing pin analog antara lain sebagai berikut:

1. Menggunakan referensi tegangan internal 2.56V
2. Data bersifat *right-justified*
3. Menggunakan prescaler CLK/128

Untuk memenuhi seluruh persyaratan tersebut, maka kode Assembly yang perlu diimplementasikan antara lain sebagai berikut:

```
;=====================================================================
init_ADC0:
;---------------------------------------------------------------------
; Uses R20 temporarily
; Input in pin PC0
;---------------------------------------------------------------------
    SBI   DDRC, 0     ;set pin PC0 as i/p for ADC0
    ;-----------------------------------------------------------------
    LDI   R20, 0x40   ;internal 2.56V, right-justified data, ADC0
    STS   ADMUX, R20
    LDI   R20, 0x87   ;enable ADC, ADC prescaler CLK/128
    STS   ADCSRA, R20
    RET
;=====================================================================
```

2.4.5 Read ADC

Setelah menginisialisasi pin analog, maka hal berikut yang perlu diimplementasikan adalah proses konversi nilai analog tersebut menjadi nilai diskrit melalui proses konversi ADC dengan spesifikasi yang telah didefinisikan sebelumnya. Hasil dari konversi ADC akan berkisar pada nilai digital 0 – 1023, sehingga memerlukan register sebesar 10-bit untuk menampung hasil tersebut. Register R18 akan digunakan untuk menyimpan hasil bit 0-7 hasil konversi ADC, sedangkan register R19 akan digunakan untuk menyimpan hasil bit 8-9 hasil konversi ADC. Berikut merupakan kode Assembly untuk merealisasikan hal tersebut:

```
;=====================================================================
read_ADC:
;---------------------------------------------------------------------
; Uses R17, R20, R21 temporarily
; R18 stores bit 0-7 of ADC conversion
; R19 stores bit 8-9 of ADC conversion
;---------------------------------------------------------------------
    LDI   R20, 0xC7   ;set ADSC in ADCSRA to start conversion
    STS   ADCSRA, R20
    ;---------------------------------------------------------------
wait_ADC:
    LDS   R21, ADCSRA ;check ADIF flag in ADCSRA
    SBRS  R21, 4      ;skip jump when conversion is done (flag set)
    RJMP  wait_ADC    ;loop until ADIF flag is set
    ;---------------------------------------------------------------
    LDI   R17, 0xD7   ;set ADIF flag again
    STS   ADCSRA, R17 ;so that controller clears ADIF
    ;---------------------------------------------------------------
    LDS   R18, ADCL   ;get low-byte result from ADCL
    LDS   R19, ADCH   ;get high-byte result from ADCH
    RET
;=====================================================================
```

2.4.6 Delay Timer1

Pada blok program ini, dilakukan inisialisasi terhadap komponen timer1 yang terdapat pada *Arduino Uno*. Komponen timer1 ingin diinisialisasi dengan sedemikian rupa agar mampu menghasilkan spesfikasi sebagai berikut:

1. Menghasilkan delay 0.5 detik
2. Bekerja dalam mode normal
3. menggunakan prescaler CLK/1024

Untuk menghasilkan delay sebesar 0.5 detik, maka yang dapat dilakukan adalah mengatur nilai yang terdapat pada register TCNT0. Jumlah delay yang dihasilkan dapat dirumuskan sebagai berikut:

$$TCNT = 2^{16} - \frac{t}{64 \times 10^{-6}}$$

$$TCNT = 2^{16} - \frac{0.5}{64 \times 10^{-6}} = 57724$$

Program Assembly yang digunakan untuk merealisasikan seluruh persyaratan di atas dapat dilihat sebagai berikut:

```
;=====================================================================
delay_timer1:
;---------------------------------------------------------------------
; Uses R20 temporarily
; Uses built-in timer component of Arduino for precise delay
; Create 0.5 sec delay
;---------------------------------------------------------------------
.EQU value, 57724          ;value to give 0.5 sec delay
    LDI   R20, hi8(value)
    STS   TCNT1H, R20
    LDI   R20, lo8(value)
    STS   TCNT1L, R20      ;initialize counter TCNT1 = value
    ;-----------------------------------------------------
    LDI   R20, 0b00000000
    STS   TCCR1A, R20
    LDI   R20, 0b00000101
    STS   TCCR1B, R20      ;normal mode, prescaler = 1024
    ;-----------------------------------------------------
timer_loop:
    IN    R20, TIFR1       ;get TIFR1 byte & check
    SBRS  R20, TOV1        ;if TOV1=1, skip next instruction
    RJMP  timer_loop       ;else, loop back & check TOV1 flag
    ;-----------------------------------------------------
    LDI   R20, 1<<TOV1
    OUT   TIFR1, R20       ;clear TOV1 flag
    ;-----------------------------------------------------
    LDI   R20, 0b00000000
    STS   TCCR1B, R20      ;stop timer0
    RET
;=====================================================================
```

2.4.7 Delay

Blok program delay memanfaatkan *delay timer1* untuk menciptakan delay yang bervariasi dengan rentang interval dari 0.5 detik sampai dengan 2 detik. Alur kerja dari mekanisme program telah dijelaskan sebelumnya menggunakan *flowchart*. Untuk program Assembly yang digunakan pada delay, antara lain sebagai berikut:

```
;====================================================================
delay:
;--------------------------------------------------------------------
; Gives 0.5 - 2 seconds delay
;--------------------------------------------------------------------
delay_loop:
    DEC   R19
    RCALL delay_timer1
    BRGE  delay_loop
    RET
;====================================================================
```

2.4.8 Hash data

Blok program *hash data* digunakan untuk melakukan *hashing* pada data menggunakan *hash key* yang didapat, sekaligus menampilkan hasil *hash* tersebut pada layar *serial monitor*. Alur kerja dari program ini telah dijelaskan sebelumnya menggunakan *flowchart*. Program Assembly yang digunakan dapat dilihat sebagai berikut:

```
;====================================================================
hash_data:
;--------------------------------------------------------------------
; Uses R16, R17, R18, R19, R30, R31 temporarily
; Announce the game is about to start
;--------------------------------------------------------------------
    LDI   R30, lo8(data)
    LDI   R31, hi8(data)      ; Z points to string message
    MOV   R19, R26            ; Load xor hash counter
hash_agn:
    LPM   R18, Z+             ;load char of string onto R18
    CPI   R18, 0              ;check if R18=0 (end of string)
    BREQ  hash_exit           ;if yes, exit
    ;--------------------------------------------------------------------
hash_loop:
    LDS   R17, UCSR0A
    SBRS  R17, UDRE0          ;test data buffer if data can be sent
    RJMP  hash_loop
    CPI   R18, 96
    BRGE  hash_a_z
    ADD   R18, R22
    RJMP  hash_xor
```

```
hash_a_z:
    SUB   R18, R22
hash_xor:
    DEC   R19
    BRGE  print_hash          ; if (hash counter > 0) don't do hash xor
    MOV   R19, R26            ; Load hash xor counter again
    EOR   R18, R25            ; Do hash xor
print_hash:
    STS   UDR0, R18           ;send char in R18 to serial monitor
    ;---------------------------------------------------------------------
    RJMP  hash_agn            ;loop back & get next character
    ;---------------------------------------------------------------------
hash_exit:
    ;---------------------------------------------------------------
    LDI   R16, 0x0A
    RCALL LCD_buffer
    STS   UDR0, R16           ;Enter in Serial
    ;---------------------------------------------------------------------
    LDI   R16, 0x0D
    RCALL LCD_buffer
    STS   UDR0, R16           ;Finish Statement
    ;---------------------------------------------------------------------
    RET
;---------------------------------------------------------------------------
data:                         ; This is the data that is going to be hashed
    .ascii "This Contains Credential Information"
    .byte 0
;===========================================================================
```

2.4.9 Display Disturbance LED & Print Key

Terdapat sisa 2 blok program pada kode Assembly yang tidak terlampir pada dokumen ini karena terlalu panjang. Namun, kode assembly dapat diakses melalui projek yang telah diunggah pada wokwi.com melalui tautan berikut:

Link: https://wokwi.com/projects/400464062500201473

Untuk blok program *disturbance led* sendiri digunakan untuk menyalakan sejumlah 0 sampai dengan 5 led sekaligus secara bersamaan berdasarkan nilai ADC

yang diterima pada pin input analog A2. Alur kerja untuk blok program tersebut telah disediakan dalam bentuk *flowchart* sebelumnya. Sedangkan untuk blok program *print key* digunakan untuk menampilkan *hash key* yang didapatkan dari hasil konversi ADC pada pin input analog A1. Selebihnya, hasil tersebut akan ditampilkan pada layar *serial monitor*.

2.5 Hasil ADC Sampling

Sampling data akan dilakukan untuk hasil konversi ADC yang digunakan pada nilai *hash key* beserta *hashed data* yang dihasilkan. Untuk nilai asli data sebelum di-*hash* sendiri adalah sebagai berikut:

```
data:                          ; This is the data that is going to be hashed
    .ascii "This Contains Credential Information"
    .byte 0
```

Berikut merupakan tabel yang berisikan 10 contoh hasil data dengan nilai *hash key* bervariatif yang diperoleh dari suatu percobaan:

Key (ADC)	Hashed Data
176	Zecj&Nion\com!Ik_Y_ond[a&Hhgikg\ndio
261	Ugnr!Bnmu`hkr!Bqdedmuh`m!Jkenwl`uhnk
31C	`\]f,OcchU]cg,OgYXYch]Ua,Ub[cfaTh]cc
3A4	Xdee$Gk`p]e`o$Gda`a`pe]b$MjhkniWpek`
0D7	Vloa*Gej`Woja*GfSPSj`oWh*]jRefkW`oej
00D	a[\f-PbagT\af-PeXWXag\T_-VaYbe`Tg\ba
322	Vfgs"Emnr_gnq"Ercbcnrg_h"Klfmpk]rgmn
1D6	Zoc`&DienVcem+Ia_S_enn[k&BhmiagVnnie
259	]_ej)IfenX``j)Ii\^\en`Xf)R`]fldXn`f`
2E5	Ycjn%Fjia\dgn%Fm`Q`iad\i%Ngajch\adjg

2.6 Analisis Waktu eksekusi

Dikarenakan program Assembly menggunakan mekanisme delay dengan memangaatkan komponen *timer1*, maka hasil analisa waktu akan diaprokmasikan menuju delay yang dihasilkan tersebut. Hal ini dikarenakan waktu eksekusi yang digunakan untuk menjalankan setiap instruksi AVR relatif **jauh lebih cepat** daripada waktu yang terjadi pada mekanisme delay itu sendiri. Maka untuk mendapatkan nilai yang cukup representatif, hanya waktu delay yang menjadi pertimbangan waktu eksekusi.

Seperti yang diketahui sebelumnya, komponen blok *delay timer1* pasti menghasilkan delay waktu sebesar 0.5 detik. Hal ini telah terbukti oleh perhitungan matematis yang terlampir di atas. Namun, pada program delay yang ditimbulkan disebabkan oleh komponen blok *delay*, dimana pada blok itu sendiri akan dilkukan pengulangan pemanggilan *delay timer1*, sehingga mampu menghasilkan delay sebesar 0.5-2 detik. Dengan mengamati alur kerja blok program *delay*, maka didapatkan hasil waktu eksekusi untuk setiap penghasilan *hash key* beserta *hashed data*-nya sebagai berikut:

Potentiometer 1 (ADC)	**Delay**
00 0000 0000 – 00 1111 1111	0.5 s
01 0000 0000 – 01 1111 1111	1 s
10 0000 0000 – 10 1111 1111	1.5 s
11 0000 0000 – 11 1111 1111	2 s

BAB III
PENUTUP

3.1 Kesimpulan

Dari seluruh penjelasan mengenai rancangan rangkaian untuk tugas akhir Sistem Siber-Fisik di atas, dapat disimpulkan bahwa terdapat beberapa poin penting terkait *AVR Hash Key Generator Using Light Intensity*, antara lain sebagai berikut:

1. Rancangan AVR yang diusulkan bertujuan untuk meningkatkan keamanan dalam proses *hashing* dengan memanfaatkan nilai acak dari besaran fisik intensitas cahaya. Hal ini berguna untuk mempersulit prediksi hasil hash key.
2. Implementasi rancangan melibatkan mikrokontroler *Arduino Uno* yang berfungsi untuk mengeksekusi algoritma hashing, serta sensor LDR untuk membaca nilai intensitas cahaya. Selain itu, terdapat juga LED di sekitar LDR yang dapat diatur melalui *potentiometer* untuk memberikan gangguan pada nilai intensitas cahaya agar lebih bervariasi.
3. Mekanisme kerja rancangan secara umum adalah pembacaan nilai analog intensitas cahaya oleh LDR, yang kemudian dikonversi menjadi nilai digital melalui ADC. Hasil konversi tersebut digunakan sebagai *hash key* dalam algoritma *hashing* sederhana yang ditanamkan pada *Arduino Uno*. Hasil *hash* key beserta data yang telah di-*hash* kemudian ditampilkan melalui komunikasi serial.
4. Periode penghasilan hash key dapat diatur pada rentang 0.5 - 2 detik melalui *potentiometer*. Hal ini dimaksudkan untuk memudahkan penyimpanan *hash* key pada database secara fleksibel.
5. Berdasarkan percobaan yang telah dilakukan, didapatkan bahwa hash key yang dihasilkan cukup bervariasi dengan adanya mekanisme gangguan LED pada sensor LDR. Waktu eksekusi untuk setiap penghasilan hash key juga sesuai dengan pengaturan delay melalui *potentiometer*.

Dengan demikian, dapat disimpulkan bahwa rancangan *AVR Hash Key Generator Using Light Intensity* mampu menghasilkan *hash key* yang cukup acak dan sulit diprediksi dengan memanfaatkan nilai intensitas cahaya. Pendekatan ini dapat menjadi alternatif sederhana dan terjangkau untuk meningkatkan keamanan dalam proses *hashing* data, terutama dalam bidang keamanan jaringan.

3.2 Link Video Presentasi

https://youtu.be/EYq948GAAsM?si=qqCs-83GXfIi-gpC

Pertanyaan yang dijawab:

1) Apabila anda menggunakan ADC, jelaskan cara anda mengukur input dari sensor hingga menjadi informasi yang dapat diolah!

5) Apabila sistem anda menggunakan delay, jelaskan tujuan anda melakukan delay dan bagaimana cara anda dapat mengatur agar delay yang diinginkan tercapai!

REFERENSI

1. “What is hashing in cybersecurity? - crowdstrike,” crowdstrike.com, https://www.crowdstrike.com/cybersecurity-101/data-protection/data-hashing/ (accessed Jun. 13, 2024).
2. How do lava lamps help with internet encryption? | cloudflare, https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/ (accessed Jun. 13, 2024).
3. “Assembly via Arduino (part 4) - programming ADC,” YouTube, https://youtu.be/7PVTnT59cqE (accessed Jun. 13, 2024).
4. “Assembly via Arduino (part 8) - ADC value on Serial Monitor,” YouTube, https://youtu.be/pmiTszZhPoI (accessed Jun. 13, 2024).
5. “Assembly via Arduino (part 13) - programming timer 1,” YouTube, https://youtu.be/uBHvHNJ1A7I (accessed Jun. 13, 2024).