گزارش تمرین ۴ - پیاده سازی سیستم رمزنگاری RSA

نگین مشایخی ۹۸۲۴۳۰۵۴- فائزه سرلکی فر ۹۸۲۴۳۰۳۳

بخش اول-توضیحات اولیه)

در این تمرین یک سیستم رمزنگاری را با روش های مبتنی بر همطراحی سخت افزار و نرم افزار پیاده سازی کردیم.

طبق خواسته ی مسئله بخش های محاسباتی به صورت سخت افزاری پیاده شدند و مابقی به صورت نرم افزاری.

کلیت پروژه شامل ۲ کد rsa.c (که کد های بخش نرم افزاری را شامل میشود) و rsa.fdl (که کد های بخش سخت افزاری را شامل میشود) میباشد.

ارتباط سخت افزار و نرم افزار به صورت Memory-mapped میباشد که در ادمه توضیح داده خواهد شد به چه صورت این رویکرد پیاده سازی شده است.

بخش های اصلی:

- سخت افزار: gcd, power
- نرم افزار: مابقی برنامه

بخش دوم-نرم افزار)

همانطور که توضیح داده شد شیوه اتصال سخت افزار و نرم افزار به صورت Memory-mapped میباشد.برای پیاده سازی این روش باید متغیر های مشترکی بین سخت افزار و نرم افزار وجود داشته باشند که در خانه هایی مشخص از حافظه قرار بگیرند و هر دو بخش دسترسی به این خانه های حافظه داشته باشند.

همانطور که در شکل زیر دیده میشود متغیر های مشترک به خانه هایی مشخص از حافظه تخصیص داده شده اند که در بخش سخت افزار(بخش سوم) همین خانه های حافظه استفاده شده اند.

C rsa.c

```c
#include <stdio.h>

// memory map
volatile unsigned long long int *gcd_inp1 = (unsigned long long int *) 0x80000000;
volatile unsigned long long int *gcd_inp2 = (unsigned long long int *) 0x80000008;
volatile unsigned long long int *gcd_out = (unsigned long long int *) 0x80000010;
volatile unsigned  int *gcd_load = (unsigned int *) 0x80000018;
volatile unsigned  int *gcd_done = (unsigned int *) 0x8000001c;
volatile unsigned long long int *power_x = (unsigned long long int *) 0x80000020;
volatile unsigned long long int *power_y = (unsigned long long int *) 0x80000028;
volatile unsigned long long int *power_m = (unsigned long long int *) 0x80000030;
volatile unsigned long long int *power_outp = (unsigned long long int *) 0x80000038;
volatile unsigned  int *power_load = (unsigned int *) 0x80000040;
volatile unsigned  int *power_done = (unsigned int *) 0x80000044;

// prototypes
unsigned long long int gcd(unsigned long long int, unsigned long long int);
unsigned long long power(unsigned long long, unsigned long long, unsigned long long);
unsigned int calculate_e(unsigned int z);
unsigned int calculate_d(unsigned int z, unsigned int e);
unsigned long long encryption(unsigned long long msg, unsigned int e, unsigned int n);
unsigned long long decryption(unsigned long long c, unsigned int d, unsigned int n);
```

شکل ۱

در این سیستم رمزنگاری ۳ بخش اصلی وجود دارد.

```c
int main(){

    // setup private and public keys
    unsigned int p = 23;
    unsigned int q = 2;
    unsigned int n = p*q;
    unsigned int z = (p-1)*(q-1);
    unsigned int e = calculate_e(z);
    unsigned int d = calculate_d(z, e);

    // Message to be encrypted
    unsigned long long msg = 26;
    printf("Message data = %lld\n", msg);

    unsigned long long cypher = encryption(msg,e,n);
    decryption(cypher,d,n);

    return 0;
}
```

شکل ۲

- مقداردهی به کلیدهای عمومی و خصوصی = خط ۲۷ تا ۳۲ (شکل ۲)

```
65  unsigned int calculate_e(unsigned int z){
66      unsigned int e = 2;
67      while (e < z) {
68          // e must be co-prime to z and smaller than z
69          if (gcd(e, z)==1)
70              break;
71          else
72              e++;
73      }
74      return e;
75  }
76  unsigned int calculate_d(unsigned int z, unsigned int e){
77      unsigned int d = 2;
78      while (d < z) {
79          if ((d*e)%z == 1)
80              break;
81          else
82              d++;
83      }
84      return d;
85  }
```

شکل ۳

کد فراخوانی شده توسط تابع calculate_e (شکل۳) در ادامه (شکل۴) آمده است.

این کد صرفا خانه های حافظه ی مربوط به تابع gcd (که در ابتدا مشخص شد) را مقداردهی کرده و سپس سیگنال لود این تابع را در سخت افزار فراخوانی میکند و منتظر میماند تا سیگنال done توسط سخت افزار مقدار یک بگیرد.(خط ۹۴ و خط ۱۰۶ (شکل۵))

```
44  unsigned long long int gcd(unsigned long long int a, unsigned long long int b){
45      *gcd_inp1 = a;
46      *gcd_inp2 = b;
47      *gcd_load = 1;
48      while (*gcd_done !=1 );
49      long long int result = *gcd_out;
50      *gcd_load = 0;
51      return result;
52  }
53
54  unsigned long long power(unsigned long long x, unsigned long long y, unsigned long long m){
63  }
64
```

شکل۴

```
74 > dp gcd_dp(…
80   ) {
81       reg a, b: ns(64);
82       always { outp = b; }
83       sfg ready { done = 0; }
84 >     sfg init {…
88       }
89 >     sfg calculate {…
93       }
94       sfg idle { done = 1; }
95   }
96
97   fsm gcd_fsm(gcd_dp) {
98       initial s0;
99       state s1, s2;
100      @s0
101          if (load) then (init) -> s1;
102          else (ready) -> s0;
103      @s1
104          if (a%b==0) then (idle) -> s2;
105          else (calculate ) -> s1;
106      @s2 if (load) then (idle) -> s2;
107          else (ready) -> s0;
108  }
```

شکل ۵

- رمزگذاری = خط ۳۸ (شکل ۲)

```
86
87  unsigned long long encryption(unsigned long long msg, unsigned int e, unsigned int n){
88      unsigned long long cypher  = power(msg, e, n);
89      printf("Encrypted data = %lld\n", cypher);
90      return cypher;
91  }
```

شکل۶

این کد تابع power را فراخوانی میکند که مانند تابع gcd (شکل۴) صرفا خانه های حافظه ی مربوط به عملیات power در سخت افزار را مقداردهی کرده و منتظر میماند تا سخت افزار محاسبه را انجام دهد.پارامترهای این تابه کلید عمومی و پیام اصلی هستند.

سپس کد رمز شده را در کنسول چاپ میکند.

- رمزگشایی = خط ۳۹ (شکل ۲)

```
92
93  unsigned long long decryption(unsigned long long c, unsigned int d, unsigned int n){
94      unsigned long long msg = power(c, d, n);
95      printf("Original Message Sent = %lld\n", msg);
96      return msg;
97  }
```

شکل۷

این کد کاملا مشابه قبل است با این تفاوت که ورودی هایش پیام رمز شده و کلید خصوصی میباشد و در نهایت پیام رمزگشایی شده که معادل پیام اول است را چاپ میکند.

بخش سوم-سخت افزار)

در این بخش ابتدا Interface آمده است و حافظه های استفاده شده در بخش نرم افزار اینجا نیز برای استفاده ی سخت افزار تعریف شده اند.(شکل۸)

```
ipblock myarm{
    iptype "armsystem";
    ipparm "exec=rsa";
    ipparm "period=1";
}

ipblock arm_gcd_in_1(out data : ns(64)){
    iptype "armsystemsource";
    ipparm "core=myarm";
    ipparm "address=0x80000000";
}

ipblock arm_gcd_in_2(out data : ns(64)){…
}

ipblock arm_gcd_load(out data : ns(1)){…
}

ipblock arm_gcd_out(in data : ns(64)){…
}
```

شکل ۸

در ادامه ۲ تابع سخت افزاری gcd و power تعریف شده اند که هرکدام شامل یک dp و یک fsm هستند.

در شکل۹ پیاده سازی gcd با زبان جزل آمده است.

```
> dp gcd_dp(…
) {
    reg a, b: ns(64);
    always { outp = b; }
    sfg ready { done = 0; }
    sfg init {
        a = inp1;
        b = inp2;
        done = 0;
    }
    sfg calculate {
        a = b;
        b = a%b;
        done = 0;
    }
    sfg idle { done = 1; }
}

fsm gcd_fsm(gcd_dp) {
    initial s0;
    state s1, s2;
    @s0
        if (load) then (init) -> s1;
        else (ready) -> s0;
    @s1
        if (a%b==0) then (idle) -> s2;
        else (calculate ) -> s1;
    @s2 if (load) then (idle) -> s2;
        else (ready) -> s0;
}
```

شکل۹

این تابع زمانی که سیگنال لود فعال باشد محاسبات را شروع کرده و با فعال کردن سیگنال done به نرم افزار پیام اتمام محاسبات میدهد و نرم افزار زمان دریافت این سیگنال مقدار خروجی را میخواند.

برای تابع power هم ۲ بخش مشابه شکل۹ وجود دارد که توان را محاسبه کرده و از خروجی باقی مانده به پیمانه ی مشخص شده میگیرد و در خروجی مینویسد.(شکل ۱۰)

```
111 > dp power_dp (…
118   ){
119       reg temp, counter: ns(64);
120       always { outp = temp; }
121       sfg ready { done = 0; }
122       sfg init {
123           temp = 1;
124           counter = y;
125           done = 0;
126       }
127       sfg calculate {
128           temp = (temp * x) % m;
129           counter = counter - 1;
130           done = 0;
131       }
132       sfg idle { done = 1; }
133   }
134
135   fsm power_fsm(power_dp) {
136       initial s0;
137       state s1, s2;
138
139       @s0
140           if (load) then (init) -> s1;
141           else (ready) -> s0;
142       @s1
143           if (counter==0) then (idle) -> s2;
144           else (calculate) -> s1;
145       @s2 if (load) then (idle) -> s2;
146           else (ready) -> s0;
147   }
```

شکل۱۰

بخش چهارم-شبیه سازی و اجرا)

در انتها کد نوشته شده به زبان c را کامپایل کرده و در اختیار سخت افزار قرار میدهیم.(خط اول شکل۱۱)

شکل۱۱

سپس سیستم را شبیه سازی میکنیم(خط دوم شکل۱۱)

خروجی سیستم قابل مشاهده است و از آنجا که پیام اصلی با پیام رمزگشایی شده بعد از رمز شدن مطابقت دارد عملکرد سیستم درست میباشد.