CS 577: Project Report

Project Number :	P13
Group Number:	22
Name of the top modules:	Crypto_sign_keypair
Link for GitHub Repo:	https://github.com/aayush010

Group Members	Roll Numbers
Aayush Jaiswal	194101001
Animesh Ranjan	194101005
Ashish Kumar	194101009
Naman Garg	194101035

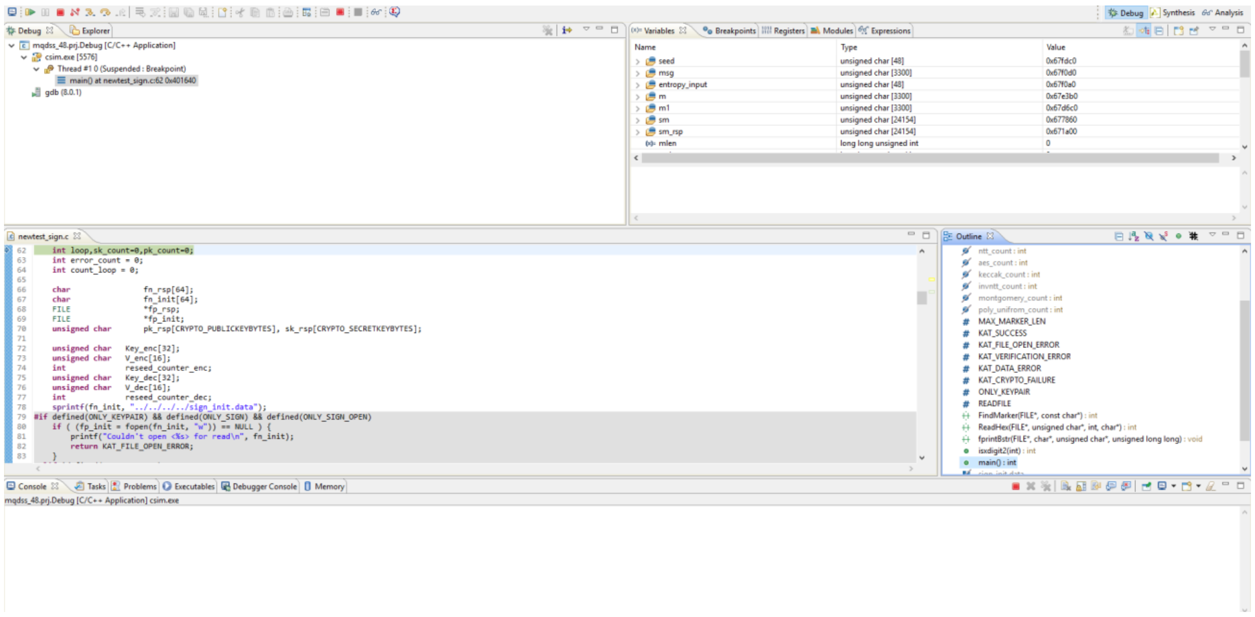
Date: 09.04.2020

INTRODUCTION

PHASE-1

1. Running the algorithm

1.1 Simulation screenshot



```
mqdss_48.prj.Debug [C/C++ Application]
  csim.exe [5576]
    Thread #1 0 (Suspended : Breakpoint)
      main() at newtest_sign.c:62 0x401640
  gdb (8.0.1)
```

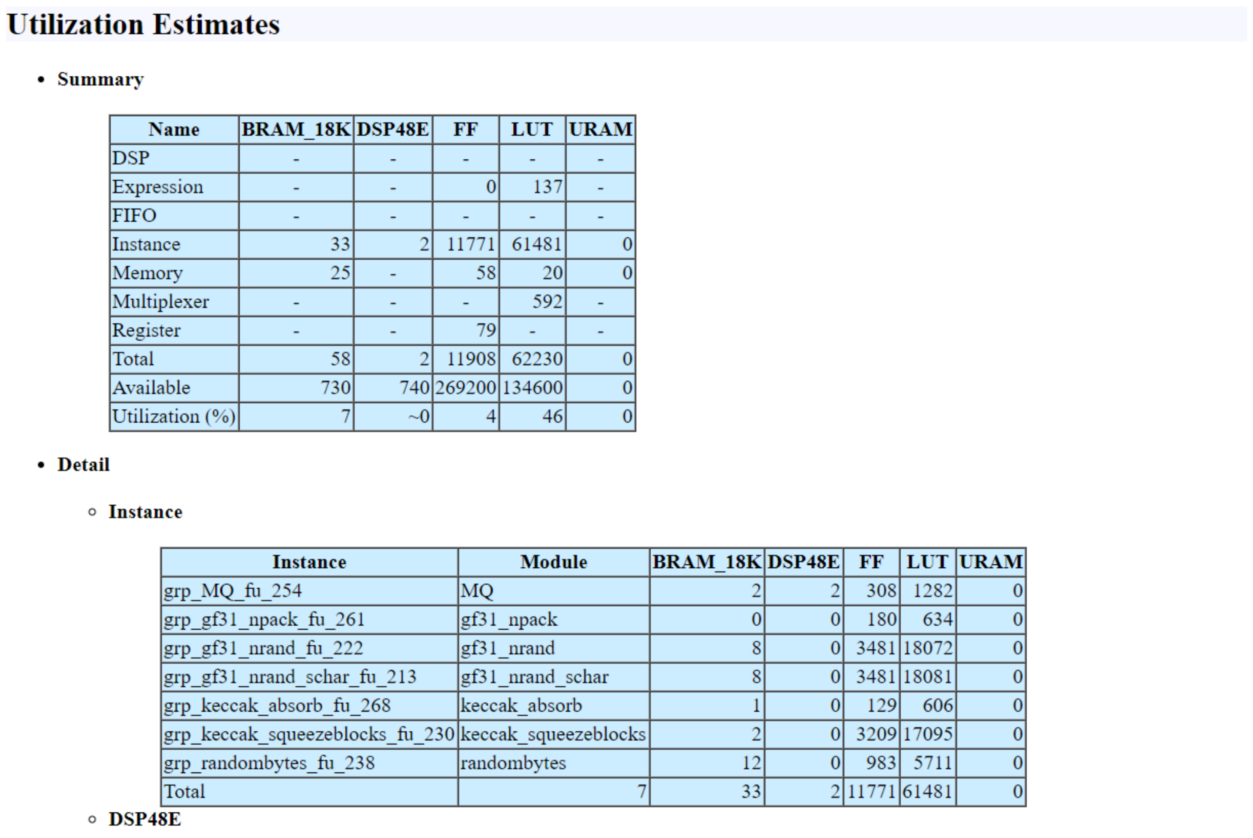
Name	Type	Value
seed	unsigned char [48]	0x67fdc0
msg	unsigned char [3300]	0x67f0d0
entropy_input	unsigned char [48]	0x67f0a0
m	unsigned char [3300]	0x67e3b0
m1	unsigned char [3300]	0x67d6c0
sm	unsigned char [24154]	0x677860
sm_rsp	unsigned char [24154]	0x671a00
mlen	long long unsigned int	0

```c
    int loop,sk_count=0,pk_count=0;
    int error_count = 0;
    int count_loop = 0;

    char                fn_rsp[64];
    char                fn_init[64];
    FILE                *fp_rsp;
    FILE                *fp_init;
    unsigned char       pk_rsp[CRYPTO_PUBLICKEYBYTES], sk_rsp[CRYPTO_SECRETKEYBYTES];

    unsigned char   Key_enc[32];
    unsigned char   V_enc[16];
    int             reseed_counter_enc;
    unsigned char   Key_dec[32];
    unsigned char   V_dec[16];
    int             reseed_counter_dec;
    sprintf(fn_init, "../../../../sign_init.data");
#if defined(ONLY_KEYPAIR) && defined(ONLY_SIGN) && defined(ONLY_SIGN_OPEN)
    if ( (fp_init = fopen(fn_init, "w")) == NULL ) {
        printf("Couldn't open <%s> for read\n", fn_init);
        return KAT_FILE_OPEN_ERROR;
    }
```

```
ntt_count : int
aes_count : int
keccak_count : int
invntt_count : int
montgomery_count : int
poly_unifrom_count : int
MAX_MARKER_LEN
KAT_SUCCESS
KAT_FILE_OPEN_ERROR
KAT_VERIFICATION_ERROR
KAT_DATA_ERROR
KAT_CRYPTO_FAILURE
ONLY_KEYPAIR
READFILE
FindMarker(FILE*, const char*) : int
ReadHex(FILE*, unsigned char*, int, char*) : int
fprintBstr(FILE*, char*, unsigned char*, unsigned long long) : void
isxdigit2(int) : int
main() : int
```

```
mqdss_48.prj.Debug [C/C++ Application] csim.exe
```

1.2 Synthesis screenshot



Utilization Estimates

- **Summary**

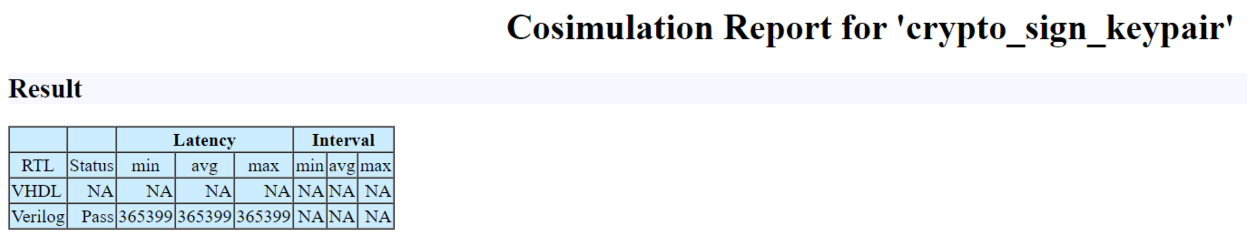
Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	137	-
FIFO	-	-	-	-	-
Instance	33	2	11771	61481	0
Memory	25	-	58	20	0
Multiplexer	-	-	-	592	-
Register	-	-	79	-	-
Total	58	2	11908	62230	0
Available	730	740	269200	134600	0
Utilization (%)	7	~0	4	46	0

- **Detail**
  - **Instance**

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
grp_MQ_fu_254	MQ	2	2	308	1282	0
grp_gf31_npack_fu_261	gf31_npack	0	0	180	634	0
grp_gf31_nrand_fu_222	gf31_nrand	8	0	3481	18072	0
grp_gf31_nrand_schar_fu_213	gf31_nrand_schar	8	0	3481	18081	0
grp_keccak_absorb_fu_268	keccak_absorb	1	0	129	606	0
grp_keccak_squeezeblocks_fu_230	keccak_squeezeblocks	2	0	3209	17095	0
grp_randombytes_fu_238	randombytes	12	0	983	5711	0
Total	7	33	2	11771	61481	0

  - **DSP48E**

1.3 C/RTL co-simulation screenshot



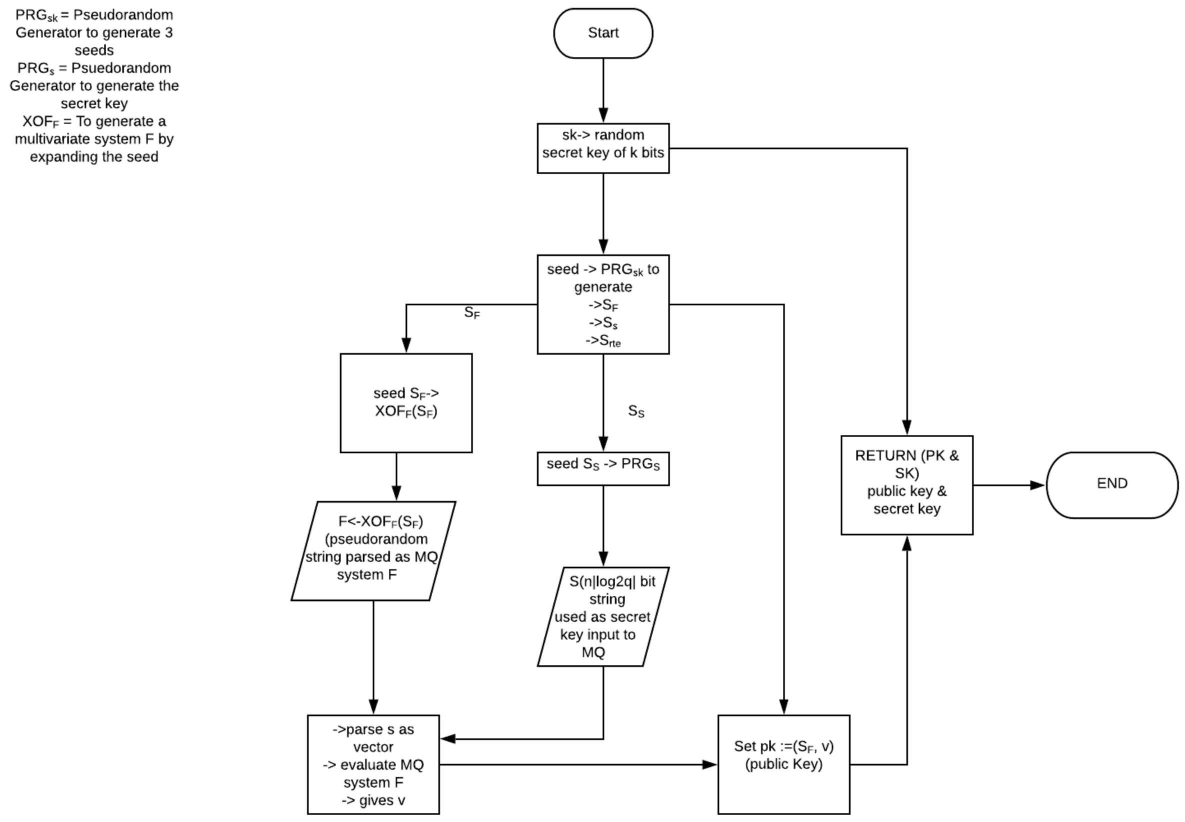
Cosimulation Report for 'crypto_sign_keypair'

Result

		Latency			Interval		
RTL	Status	min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	365399	365399	365399	NA	NA	NA

2. Flowchart (Give the flowchart of the function used. Describe basic understanding of the algorithm)

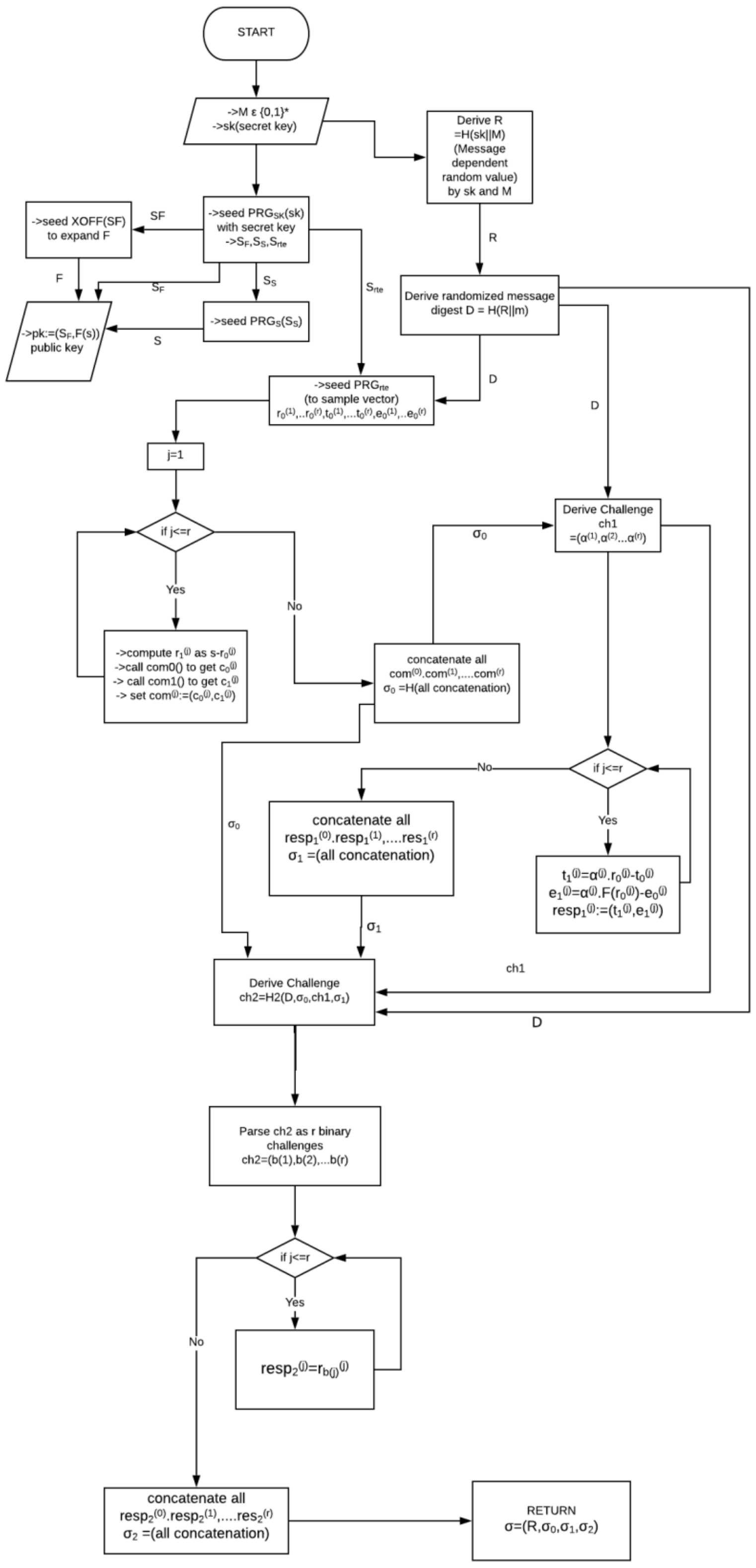
Chart 1 : Key Generation



PRG_{sk} = Pseudorandom Generator to generate 3 seeds
PRG_s = Psuedorandom Generator to generate the secret key
XOF_F = To generate a multivariate system F by expanding the seed
Start
sk-> random secret key of k bits
seed -> PRG_{sk} to generate ->S_F ->S_s ->S_{rte}
S_F
S_S
seed S_F-> $XOF_F(S_F)$
seed S_S -> PRG_S
F<-$XOF_F(S_F)$ (pseudorandom string parsed as MQ system F
S(n\|log2q\| bit string used as secret key input to MQ
->parse s as vector -> evaluate MQ system F -> gives v
Set pk :=(S_F, v) (public Key)
RETURN (PK & SK) public key & secret key
END

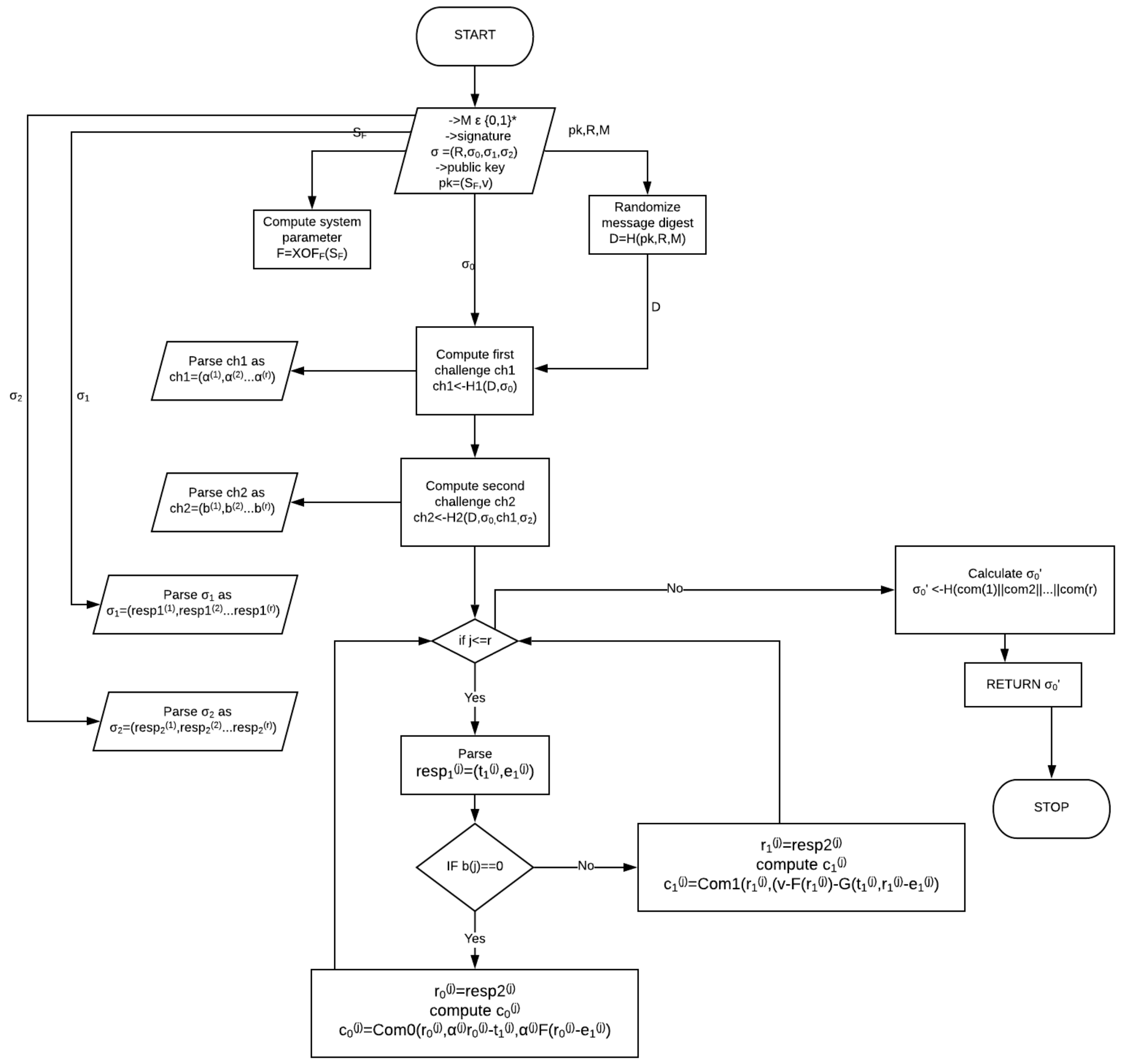
MQDSS KEY GENERATION

Chart 2 : MQDSS Sign Generation



START
->M ε {0,1}*
->sk(secret key)
Derive R =H(sk||M) (Message dependent random value) by sk and M
R
->seed PRG$_{SK}$(sk) with secret key ->S_F,S_S,S_{rte}
SF
->seed XOFF(SF) to expand F
F
S_F
S_S
S_{rte}
->seed PRG$_S$(S_S)
S
->pk:=(S_F,F(s)) public key
Derive randomized message digest D = H(R||m)
D
D
->seed PRG$_{rte}$ (to sample vector) $r_0^{(1)}$,..$r_0^{(r)}$,$t_0^{(1)}$,...$t_0^{(r)}$,$e_0^{(1)}$,..$e_0^{(r)}$
j=1
if j<=r
Yes
No
->compute $r_1^{(j)}$ as s-$r_0^{(j)}$ ->call com0() to get $c_0^{(j)}$ -> call com1() to get $c_1^{(j)}$ -> set $com^{(j)}$:=($c_0^{(j)}$,$c_1^{(j)}$)
concatenate all $com^{(0)}$.$com^{(1)}$,....$com^{(r)}$ $σ_0$ =H(all concatenation)
$σ_0$
Derive Challenge ch1 =($α^{(1)}$,$α^{(2)}$...$α^{(r)}$)
if j<=r
No
Yes
$t_1^{(j)}$=$α^{(j)}$.$r_0^{(j)}$-$t_0^{(j)}$ $e_1^{(j)}$=$α^{(j)}$.F($r_0^{(j)}$)-$e_0^{(j)}$ $resp_1^{(j)}$:=($t_1^{(j)}$,$e_1^{(j)}$)
concatenate all $resp_1^{(0)}$.$resp_1^{(1)}$,....$res_1^{(r)}$ $σ_1$ =(all concatenation)
$σ_0$
$σ_1$
ch1
D
Derive Challenge ch2=H2(D,$σ_0$,ch1,$σ_1$)
Parse ch2 as r binary challenges ch2=(b(1),b(2),...b(r)
if j<=r
Yes
No
$resp_2^{(j)}$=$r_{b(j)}^{(j)}$
concatenate all $resp_2^{(0)}$.$resp_2^{(1)}$,....$res_2^{(r)}$ $σ_2$ =(all concatenation)
RETURN σ=(R,$σ_0$,$σ_1$,$σ_2$)

Chart 3 : MQDSS Key Verification


START
->M ε {0,1}*
->signature
σ =(R,σ0,σ1,σ2)
->public key
pk=(SF,v)
SF
pk,R,M
Compute system parameter F=XOFF(SF)
Randomize message digest D=H(pk,R,M)
σ0
D
σ2
σ1
Compute first challenge ch1 ch1<-H1(D,σ0)
Parse ch1 as ch1=(α(1),α(2)...α(r))
Compute second challenge ch2 ch2<-H2(D,σ0,ch1,σ2)
Parse ch2 as ch2=(b(1),b(2)...b(r))
Parse σ1 as σ1=(resp1(1),resp1(2)...resp1(r))
Parse σ2 as σ2=(resp2(1),resp2(2)...resp2(r))
if j<=r
No
Yes
Calculate σ0' σ0' <-H(com(1)||com2||...||com(r))
RETURN σ0'
STOP
Parse resp1(j)=(t1(j),e1(j))
IF b(j)==0
No
Yes
r1(j)=resp2(j) compute c1(j) c1(j)=Com1(r1(j),(v-F(r1(j))-G(t1(j),r1(j)-e1(j))
r0(j)=resp2(j) compute c0(j) c0(j)=Com0(r0(j),α(j)r0(j)-t1(j),α(j)F(r0(j)-e1(j))

3. Result

FPGA Part	Name of Top Module	FF	LUT	BRAM	DSP	Latency	II
Artix7	Crypto_sign_keypair	11908	62230	58	2	365399	365399

3.1 Explain the result

Here as given from above simulation and cosimulation results the area utilization is pretty decent i.e. 46% we will try to decrease it future if possible considering the latency which is high i.e. 365399, as decreasing the area may cause increase in latency so we will mainly focus on maintaining the latency and decreasing is further more. Moreover we observer the algorithm is slow.

3.2 Problems and its solution

The main problem here is that the latency is very high as compare to area, so we mainly try to decrease the latency keeping the area intact and changing it as low as possible by optimizing the loops so that latency decrease. Followed by increasing the performance of the algorithm.

PHASE-2

The target FPGA board is **Artix-7 board**

Benchmark	Type (Area /Latency)	Resource Utilization				Latency		Major Optimizations
		LUT	FF	DSP	RAM	No of Clock cycle/latency	Clock period	
Baseline	Latency	46%	4%	~0	7%	365399	10ns	
Optimization1	Latency	46%	4%	~0	7%	303960	10ns	HLS Dataflow, Pipeline
Optimization 2	Latency	46%	4%	~0	7%	246138	10ns	Unroll, HLS Dataflow, Pipeline
Optimization 3	Latency & Area	47%	5%	~0	9%	240028	10ns	Pipeline

Optimization 1 : Latency

For reducing the Latency here and putting a upper bound on the LUT utilization so as in optimizing the latency the LUT is intact. Here we observed that many loops are too big but are computing in sequential manner that can be changed to perform the computing in parallel so that the time by loops can be reduce by PIPELINE in the loops and functions, and dataflow this idea was used as there were many nested loops with constant variables. And pipelining allows the loop to compute in concurrent manner.

As result the latency decreased from **365399** to **303960** but the LUT was intact.

Optimization 2 : Latency

For further reducing the latency we used Dataflow , Pipeline and Unroll in different functions and loops, where the simple pipelining technique is “fine grain” parallelizing optimization at operation level, as multiplier, adder, DATAFLOW pipelining exploits the “coarse grain” parallelism at the level of functions and loops that increases further more concurrency in loops. Here we observed that some of the loops can change the latency effectively by dataflow.

As result the latency reduced from **303960** to **246138.**

Optimization 3 : Latency & Area

The above all optimization was done keeping the LUT intact so that there is no effect on the area by any optimization done to latency and further optimization wasn’t feasible by impacting the LUT as there was still a slim chance of reducing the latency by only minimal effecting the LUT area. So here we used pipelining on such loops which decreases the latency only by a little amount but has negligible change in the LUT.

As result we got latency **240028** which is decently less than the original one i.e. **365399**

And here we doesn’t have any upper bound on the LUT area so it goes from **46%** to **47%** which is comparatively less.

Here our final Result is shown in the last Row.