

# CS 577: Project Report

<i>Project Number :</i>	P18
Group Number:	18
<i>Name of the top modules:</i>	crypto_sign_open
<i>Link for GitHub Repo:</i>	<a href="https://github.com/LohithMudragada/vlsi">https://github.com/LohithMudragada/vlsi</a>

Group Members	Roll Numbers
Sanil Upadhyay	194101043
M. lohith	194101034
Rajat shukla	194101039
Singh Priyanshu Pancham	194101048

Date: 14.05.2020

## INTRODUCTION

Quantum computers use qubits instead of normal bits and because of that, we can implement those algorithms which are not possible to implement in classical computers. And because of this, it can dramatically affect the security of various encryption algorithms. To provide security against the attack done by quantum computers as well as classical computers, **the Picnic** algorithm is designed.

The building blocks to implement picnic algorithm are as follow:-

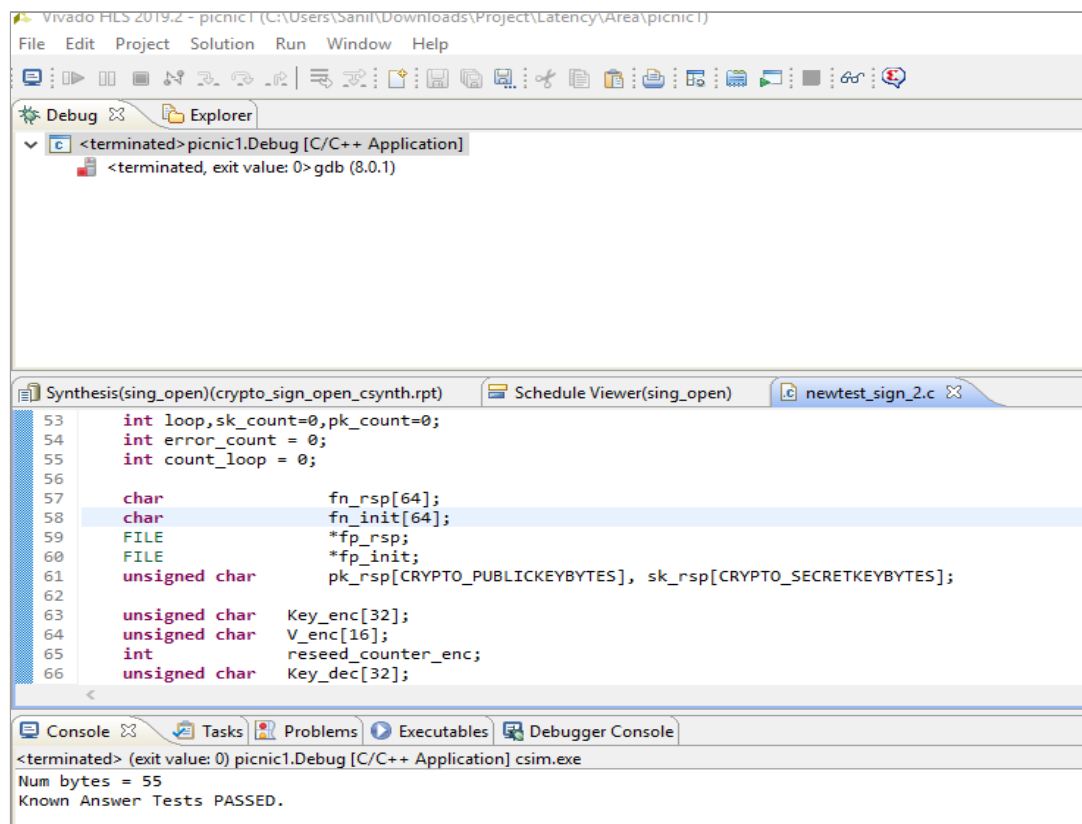
1. Zero-Knowledge proof system with post-quantum security.
2. Symmetric key primitive like hash function and block cypher
3. Multi-Party Computation (MPC)
4. Hash-based Commitment scheme

The main advantage of Picnic algorithm is that it does not use number-theoretic, or structured hardness assumptions. Some researchers believe that a fundamental public-key crypto scheme may be broken by a quantum computer by 2030

## PHASE-1

### 1. Running the algorithm

#### 1.1 Simulation screenshot



The screenshot displays the Vivado HLS 2019.2 interface. The top menu bar includes File, Edit, Project, Solution, Run, Window, and Help. The toolbar contains various icons for file operations and execution. The 'Debug' tab is active, showing a list of debug sessions. The first session, '<terminated>picnic1.Debug [C/C++ Application]', is selected, and its details are shown below: '<terminated, exit value: 0>gdb (8.0.1)'. The 'Explorer' tab is also visible, showing the project structure. The 'Synthesis(sing\_open)(crypto\_sign\_open\_csynth.rpt)' tab is active, displaying the C++ source code for the Picnic algorithm. The code includes headers for 'FILE' and 'fopen', and defines several arrays and variables for the signing process. The 'Console' tab at the bottom shows the output of the simulation, indicating that the program terminated successfully with an exit value of 0 and that all known answer tests passed.

```
53 int loop,sk_count=0,pk_count=0;
54 int error_count = 0;
55 int count_loop = 0;
56
57 char fn_rsp[64];
58 char fn_init[64];
59 FILE *fp_rsp;
60 FILE *fp_init;
61 unsigned char pk_rsp[CRYPTO_PUBLICKEYBYTES], sk_rsp[CRYPTO_SECRETKEYBYTES];
62
63 unsigned char Key_enc[32];
64 unsigned char V_enc[16];
65 int reseed_counter_enc;
66 unsigned char Key_dec[32];
```

<terminated> (exit value: 0) picnic1.Debug [C/C++ Application] csim.exe  
Num bytes = 55  
Known Answer Tests PASSED.

## 1.2 Synthesis screenshot

### Utilization Estimates

#### • Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	491	-
FIFO	-	-	-	-	-
Instance	451	2	51731	143553	0
Memory	0	-	32	8	0
Multiplexer	-	-	-	281	-
Register	-	-	345	-	-
Total	451	2	52108	144333	0
Available	730	740	269200	134600	0
Utilization (%)	61	~0	19	107	0

#### • Detail

##### ◦ Instance

Instance	Module	BRAM_18K	DSP48E	FF	LUT	URAM
grp_picnic_verify_fu_288	picnic_verify	451	2	51731	143553	0
Total	1	451	2	51731	143553	0

##### ◦ DSP48E

## 1.3 C/RTL co-simulation screenshot

### Result

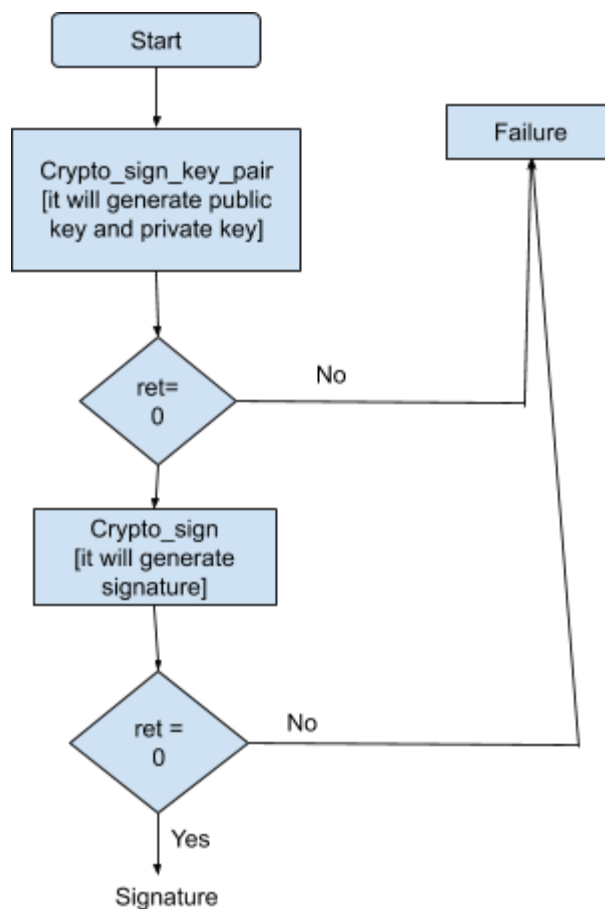
RTL	Status	Latency			Interval		
		min	avg	max	min	avg	max
VHDL	NA	NA	NA	NA	NA	NA	NA
Verilog	Pass	92273409	92273409	92273409	NA	NA	NA

## 2. Flowchart

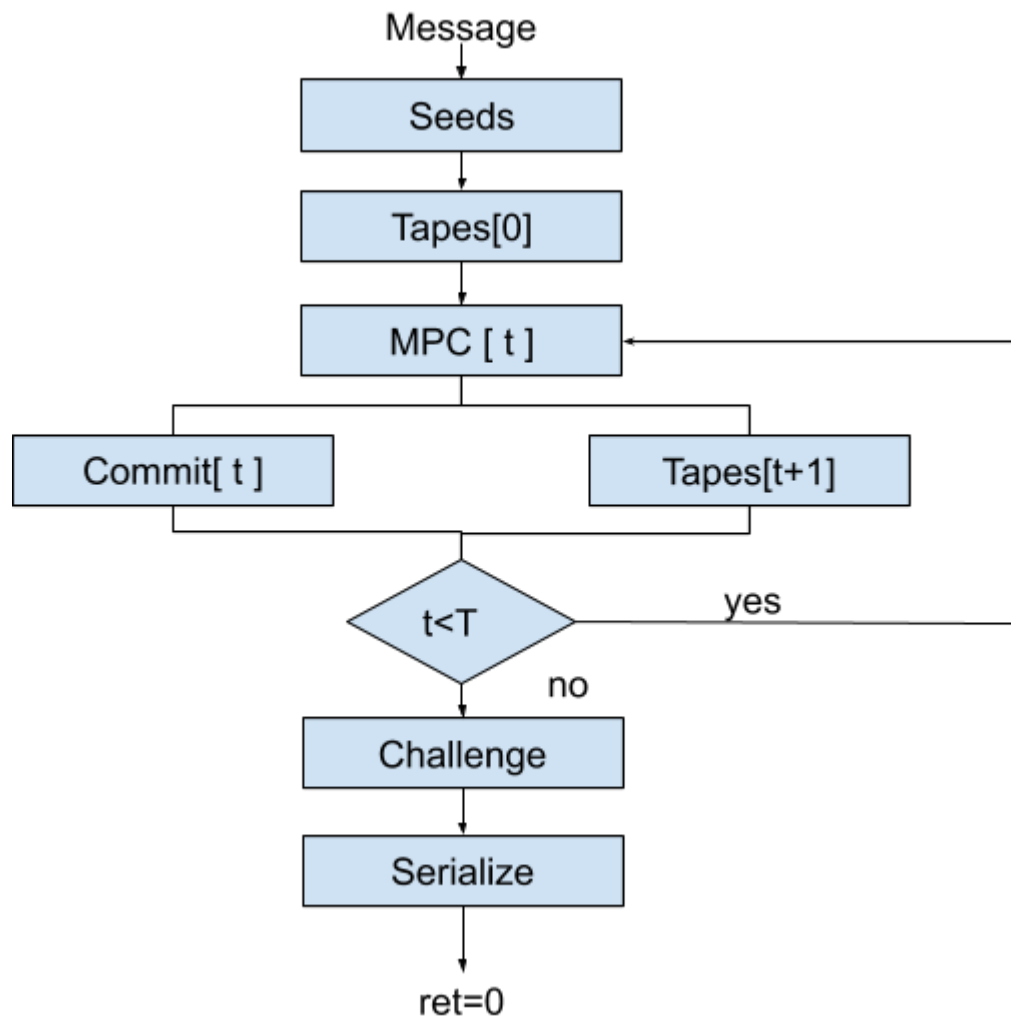
Picnic algorithms use the Digital Signature Scheme

- 1) **Key generation algorithms (run by signer)**
  - a) Secret key(SK)
  - b) Public key (PK)
- 2) **Signing algorithm (run by signer)**
  - a) Inputs: Secret key and message
  - b) Output: signature (s).
- 3) **Verification algorithm (run by verifier)**
  - a) Inputs: Public key, message(m), signature
  - b) Output: Signature s on m “valid” or “invalid

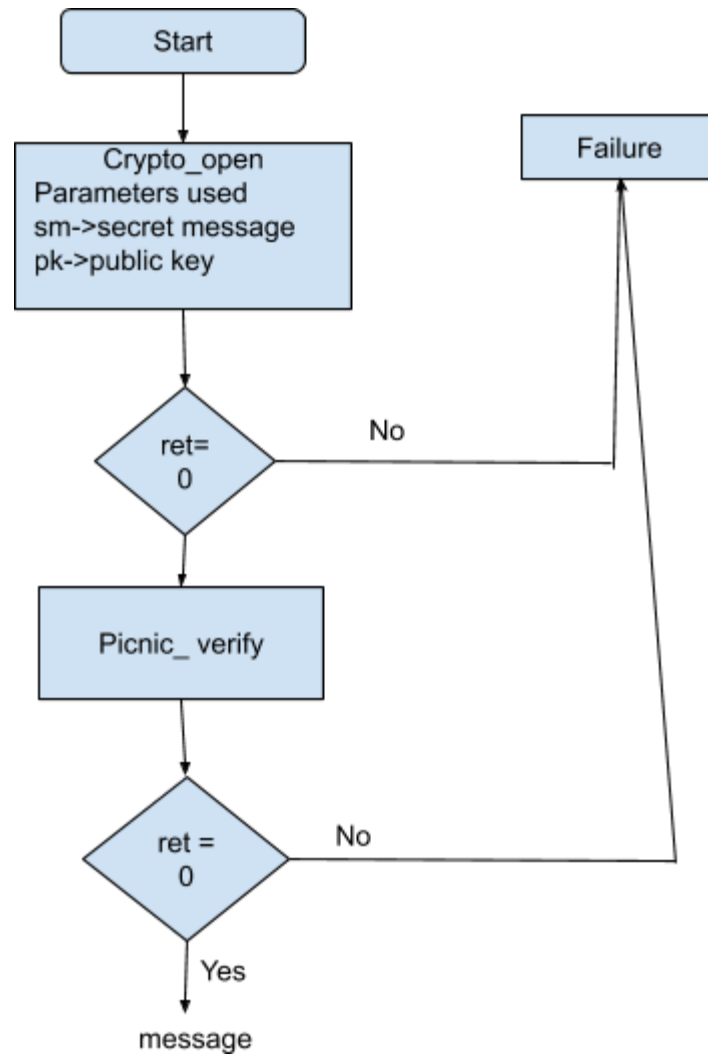
### The flow chart at the sender end



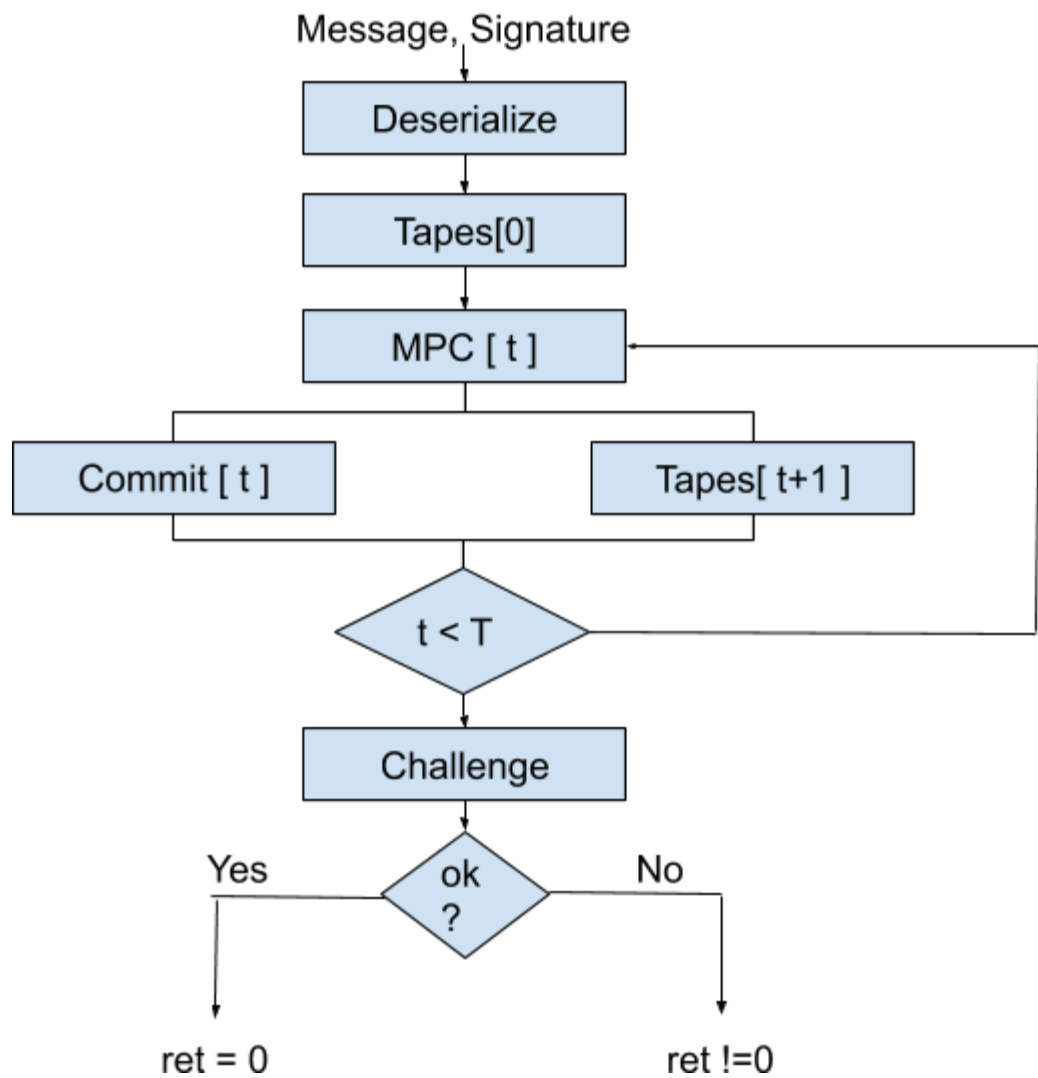
### Flow chart of crypto\_sign



### Flow Chart at the Receiver End



### Flow Chart of Picnic verify function



## Result

FPGA Part	Name of Top Module	FF	LUT	BRAM	DS P	Latency	II
Artix7	crypto_sign_open	51731	143553	451	2	92273409	92273409

### 3.1 Explain the result

As shown in the screenshots and the table above that area utilization is 107%, meaning our components won't fit on the Artrix board so our main concern is to first minimize the area. Then we will try to minimize the latency. First, we will work on the area optimization without any loop or latency optimization. But then we will try to optimize latency, as the main disadvantage of this algorithm is that it is too slow.

### 3.2 Problems and its solution

- One of the biggest problems we faced during the project was C/RTL co-simulation. As it takes approximately 10hrs to complete we have to wait for that long period of time after doing the changes.
- After we did the simulation we got the synthesis report and in that report, we found out that area is above 100%.
- In order to improve this, we did optimization on the area and then on latency
- The reason for this was latency optimization has a negative impact on area utilization
- To further improve this we try to balance both area and latency.



## 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	B	107%	19%	~0%	61%	92273409	10 ns	
Optimization1	Area	69%	12%	~0	43%	92273409	10 ns	Function Inlining
Optimization 2	Latency	69%	19%	~0%	61%	87869767	10 ns	Functions Pipelining and loop Pipelining
Optimization 3	Latency + Area	69%	12%	~0%	43%	87869767	10 ns	optimization1 + optimization2

### Optimization 1 (Area)

- By observing the synthesis report, we found the instances that are taking maximum LUT's we inlined those instances and reduced the area of them.
- The `INLINE` directive was used to implement the technique. The LUT utilization significantly dropped after applying this technique from 107% to 69%.

### Optimization 2 (Latency)

- In the original code, we have observed that there were many functions which have multiple nested loops operations with constant variables. In order to overcome this overhead and to optimize it, we used `PIPELINING` in functions and loops.
- If we pipeline a function the operations can be implemented in a concurrent manner. Because of concurrency, there was a significant improvement in latency.
- We identified similar functions and pipeline them. It significantly reduces the number of clock cycles.
- Initiation interval for pipelining was kept at default i.e. 1. As a result of this, latency or number of clock cycles reduced from 92273409 to 87869767.
- With the use of the pipeline, we optimized the latency of the program keeping the upper bound on LUT utilization of 100%.

### Optimization 3 (Area + Latency)

- when we optimized latency by using pipelining in the function, LUT was increased to 100%.
- In order to balance LUT and area we applied a combination of area optimization and latency optimization.
- Because of this we significantly decreased the area from 107% to 69% as well as we obtained better latency value 87869767.