VHDL assignment

1 Introduction

In this tutorial you are going to :

- get familiar with lightweight symmetric-key cryptography.
- practise with FPGA tools for design, simulation and synthesis.
- implement a hardware architecture using VHDL.

We will work with Subterranean 2.0, a cryptographic suite that can be used for hashing, MAC computation, stream encryption and several types of session authenticated encryption schemes.
Subterranean's official page: `https://cs.ru.nl/~joan/subterranean.html`

2 Subterranean 2.0

At its core, Subterranean 2.0 has a duplex object with a 257-bit state and a lightweight single-round permutation. The round function has four steps:

$$R = \pi \circ \theta \circ \iota \circ \chi$$

Each step of the round function has a particular purpose: χ for non-linearity, ι for asymmetry, θ for mixing and π for dispersion.
We denote the state as s and its bits as s_i with position index i ranging from 0 to 256, where any expression in the index must be taken modulo 257. For all $0 \leq i < 257$:

$$\begin{aligned}
\chi &: s_i \leftarrow s_i + (s_{i+1} + 1)s_{i+2}, \\
\iota &: s_i \leftarrow s_i + \delta_i, \\
\theta &: s_i \leftarrow s_i + s_{i+3} + s_{i+8}, \\
\pi &: s_i \leftarrow s_{12i}.
\end{aligned}$$

Here the addition and multiplication of state bits are in $\mathbb{F}_2$, and δ_i is a Kronecker delta: $\delta_i = 1$ if $i = 0$ and 0 otherwise. We sometimes use subscript notation for indicating component functions of mappings, e.g. $\chi(s)_i$ denotes bit i of the state obtained by applying χ to s. Figure 1 illustrates the round function by the computational graph of a single bit of the state. At the core of the Subterranean duplex object is a simple (internal) duplex call that first applies the Subterranean round function R to the state and then injects a string σ of variable length of at most 32 bits. Before adding it into the state, it pads the string σ to 33 bits with simple padding (10^*) and hence the absorbing rate is 33 bits. In between duplex calls, one may extract 32-bit strings z from the state, so the squeezing rate is 32 bits.
The 33 bits of σ after padding are injected into the state at positions that form the first 33 powers of 12^4 in $\langle 124 \rangle$.

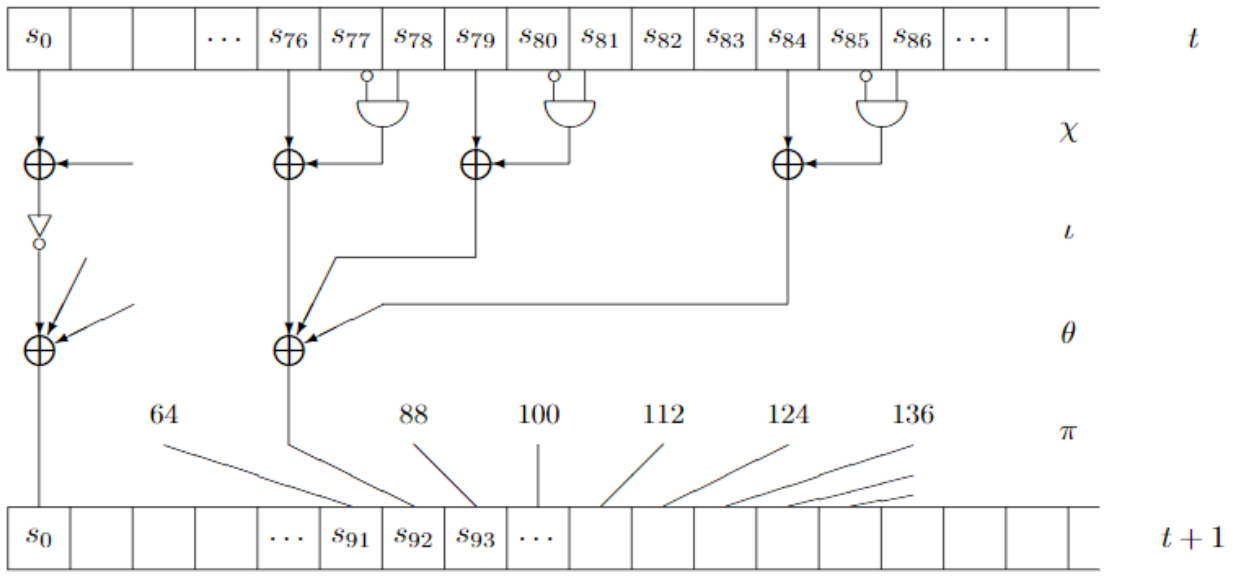


Figure 1: Subterranean round function, illustrated for bit s_{92}

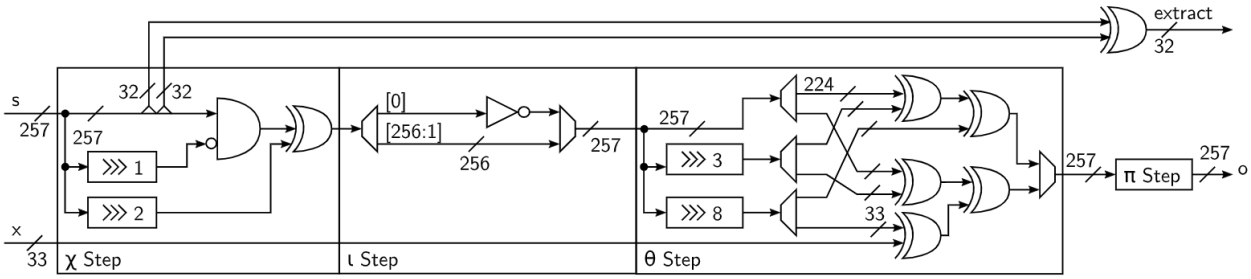


Figure 2: Subterranean duplex combinatorial circuit. Round function, input injection and output extraction

The central functions of the duplex object are the application of a permutation to the state and subsequent injection of an input string on the one hand and extraction of an output string on the other. On top of this, a number of wrapper functions are defined for absorbing arbitrary-length strings, possibly combined with encryption or decryption, for performing blank rounds and for squeezing arbitrary-length strings.
In this tutorial we are going to use the duplex object of Subterranean 2.0, as can be seen in Figure 3, to absorb with operation keyed. With this procedure, we can absorb strings of arbitrary length e.g the key, the nonce, and the message in authenticated encryption.
For more details about Subterranean, read "The ToSC paper containing specifications and design rationale" from its official page.

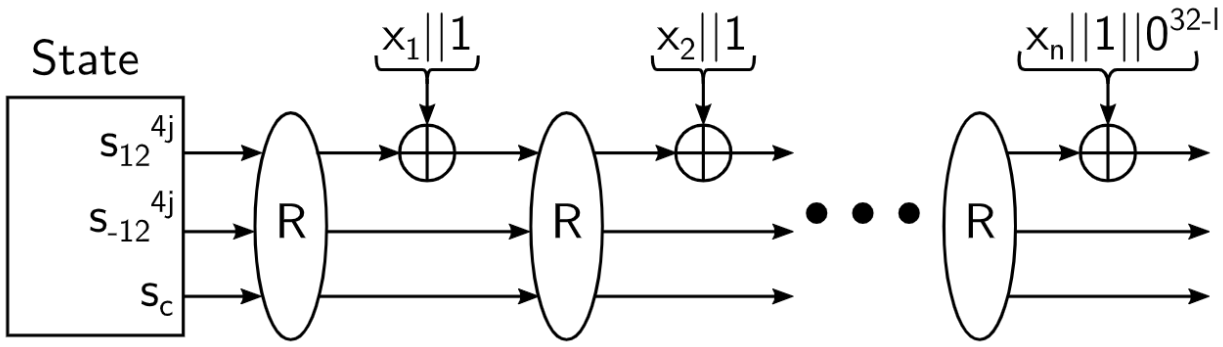
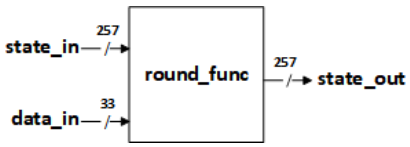


Figure 3: Subterranean absorb with operation keyed.

3 Exercise

From the official page of Subterranean, download the "Reference code". There are two implementations, one in C and one in Python. For this exercise, we refer to the Python code.
The whole cryptographic suite is implemented in the file `subterranean_bit.py`. The code has been tested with Spyder 5.1.5 for Python 3.9.
Try to understand the function `crypto_aead_encrypt(m, ad, nsec, npub, k, t_length_bytes)`. As the function's name states, it is used for authenticated encryption with associated data.
Inside `crypto_aead_encrypt()`, the basic function that is called is the `subterranean_SAE_direct_encrypt(key, nonce, associated_data, text, tag_length)`, that does encryption. With this function, the permutation will absorb the key, the nonce, the associated data and the plaintext and it will squeeze the ciphertext and the tag. Notice that `subterranean_SAE_direct_encrypt()` starts always with zero state.
Inside `subterranean_SAE_direct_encrypt()`, a basic function that you can see being called multiple times is the `subterranean_absorb_keyed(state, value_in)`.
This is the function that you need to implement in VHDL.
First, create a new python file with name `"test_sub.py"`.
Import `subterranean_bit.py`.
Create a random value of size 128 bits and print it.
Call `subterranean_init()`.
Call `subterranean_absorb_keyed()`. For its arguments, "state" and "value_in", use the result of `subterranean_init()` for "state", and the random number that you created, for "value_in".
Print the result of `subterranean_absorb_keyed()`.
At this point, you have created a golden reference for your design. You have an official reference code, and a set of valid input-outputs to test your VHDL code.

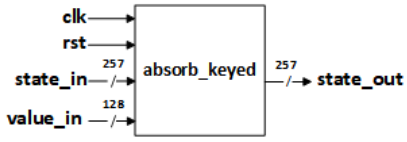
For your FPGA design, you are going to create two VHDL modules.
The first module is for the round function of subterranean. Name the module "round_func.vhd". The interface of the module will be the one in the figure below.



The module will have 2 inputs and 1 output. One input is the state, a 257-bit signal and the other is the

data that the permutation will absorb.
In this module, you are going to implement the steps of Subterranean's round function, as described above. In python, the function is `subterranean_duplex(state, sigma)`. The block diagram is the one in figure 2, except "extract". You will not do extraction with your module.
Create a testbench to verify the correct functionality of your design. Name it "tb_duplex.vhd". Use the python code to create and compare the input-output test vectors.
Run synthesis. If synthesis produces errors and/or warnings, modify your code accordingly. From the synthesis report, write down the maximum combinational path delay and the number of slice LUTs.

The second module is for the remaining functionality of `subterranean_absorb_keyed()`. Name the module "absorb_keyed.vhd". The interface of the module will be the one in the figure below.



Notice that in python "value_in" can have any size. In your VHDL implementation this size is fixed to 128 bits, same size as a key or a nonce.
Create a testbench to verify the correct functionality of your design. Name it "tb_absorb.vhd". Use the python code to create and compare the input-ouput test vectors.
Run synthesis. If synthesis produces errors and/or warnings, modify your code accordingly. From the synthesis report, write down the clock period, the number of slice LUTs and the number of Flip-Flops.
Create a project in Xilinx ISE, using the following parameters for your board:

- Family: Spartan6
- Device: XC6SLX75
- Package: CSG484
- Speed: -2
- Synthesis Tool: XST (VHDL/Verilog)
- Simulator: ISim (VHDL/Verilog)
- Preferred Language: VHDL

Create a report for this assignment. Describe your design and explain the design choices that you made. Create block diagram for the top level module. Do not create block diagram for the round function, this is already given in figure 2. Also, include the synthesis results and screenshots from the simulation diagrams. **How many clock cycles does your design need to produce a correct result?**

You need to deliver the .vhd codes of the two modules ("round_func.vhd" and "absorb_keyed.vhd") and the two testbenches ("tb_absorb.vhd" and "tb_duplex.vhd").
Also, you need to deliver the report and the python test that you created ("test_sub.py").