LWC HDL Code:
Suggested List of Deliverables
July 5, 2020

I. Introduction

In order to simplify benchmarking and any further optimizations of the developed hardware description language implementations, we propose a uniform way of publishing them on the web and/or submitting to benchmarking labs.

All implementations that share the same source code and differ only in values of generics and/or constants should be stored in the same folder. This folder can either

a) become a basis of an online repository (e.g., a GitLab or GitHub repository), or
b) be submitted as a single .zip file to the selected benchmarking labs or NIST.

Please name this folder using the following convention:

<LWC_Candidate_Name>_R<Round_number >

where <Round_number>, denotes a round of the NIST Lightweight Cryptography standardization process. For example, a valid name could be Xoodyak_R2.

In case the implementation remains the same throughout multiple rounds of the standardization process, multiple round numbers can be listed in the ascending order, e.g.,

Xoodyak_R1R2.

Within this folder, please create the following structure of files and second-level folders:

```
README.txt
LICENSE.txt [optional]
|-docs
|-src_rtl
|-src_tb
|-KAT
|-src_sw [optional]
|-scripts [optional]
|-bd      [optional]
|-results [optional]
```

The recommended content of these files and folders is described below.

II. List of Deliverables

README.txt

Please include in this file at least:

Name of the Hardware Design Group, e.g., <LWC_Candidate_Name>-Team,
CERG GMU, etc.

List of Primary Hardware Designer(s), including their first and last names, web pages, and e-mail addresses.

List of Academic advisors/Program managers, including their first and last names, web pages, and e-mail addresses.

Implemented LWC candidate, specified using the submission name, as shown on the NIST webpage devoted to the list of candidates qualified to a given round of the LWC standardization process.

For example, the README file may include the following text:

Hardware Design Group: CERG GMU
Primary Hardware Designers: Rich Haeussler,
https://cryptography.gmu.edu/team/rhaeussl.php, rhaessl@gmu.edu
Academic advisors:
Kris Gaj, https://ece.gmu.edu/~kgaj/, kgaj@gmu.edu
Jens-Peter Kaps, https://ece.gmu.edu/~jkaps/, jkaps@gmu.edu
LWC candidate: Xoodyak

The remaining portion of this file can be devoted to the explanation of the folder structure, changelog, optimization target, and any other information that may be useful to NIST, the benchmarking labs, and other hardware designers interested in optimizing the design or porting it to a different technology.

1. Assumptions

File: `docs/assumptions.pdf` or `docs/assumptions.txt`
(depending on the file format used)

This file should contain at least the following information:

A. Hardware description language used

e.g., VHDL, Verilog, System Verilog, Mixed.

A version of a language can be specified as well. However, for compatibility with a wide range of CAD tools, the use of VHDL-93 and Verilog-2001 is strongly encouraged.

B. *Use of the hardware description language source files provided as a part of the Development Package*

Please include the following table:

File name	Used	Release number	Functional
			modifications
	(Y/N)		(Y/N)
NIST_LWAPI_pkg.vhd			
StepDownCountLd.vhd			
data_piso.vhd			
data_sipo.vhd			
key_piso.vhd			
PreProcessor.vhd			
PostProcessor.vhd			
fwft_fifo.vhd			
LWC.vhd			

[*] The Release number refers to a version of the Development Package for the LWC Hardware API (e.g., v1.0.3, etc.).

[**] Functional modifications refer to any changes other than the changes related to the list and default values of generics.

C. *Supported types and order of segment types*

Please list an order of segment types supported by your implementation, using the following abbreviations:

npub	: public message number
ad	: associated data
ad_npub	: associated data || npub
npub_ad	: npub || associated data
data	: data (plaintext/ciphertext)
data_tag	: data (plaintext/ciphertext) || tag
tag	: tag

Please note that according to the LWC Hardware API:

- ad, ad_npub, npub_ad, data, and data_tag can be divided into multiple segments of the same type (each limited to the maximum of 2^{16}-1 bytes)
- npub and tag are always composed of only one segment.

For clarity, please provide the required order of segment types for all four cases, e.g.,

a. input to encryption	npub, ad, data
b. output from encryption	data, tag
c. input to decryption	npub, ad, data, tag
d. output from decryption	data

Please note that all of the above orders can be expressed using the following single option of the cryptotvgen app:

```
--msg_format npub_ad data_tag
```

D. Deviations from the LWC Hardware API v1.0 specification

These deviations may include deviations regarding the following components of the API:

D.1 Minimum compliance criteria

Please list all deviations from the criteria described in Section 1 of the LWC API specification.

For example:

- the core supports only encryption,
- the core handles only associated data, messages, and ciphertexts composed of full blocks
- AD and/or message is assumed to be padded before entering the LWC core
- unused portions of the last block are not cleared before being sent to the output port `do`
- the LWC core does not support empty AD, message, or hash message
- the supported maximum sizes of AD/plaintext/ciphertext/hash message are smaller than the limits described in the API specification (e.g., smaller than the default maximum of 2^{16}-1 bytes)
- the core requires two or more clocks (with different frequencies)
- the widths of the PDI, DO, or SDI data ports do not belong to the set {8, 16, 32}.

D.2 Interface

Please list all deviations from the interface described in Section 2 of the LWC API specification.

For example:

- any differences in the names, widths, and/or meanings of ports
- different widths of the `pdi_data` and `do_data` buses
- no use of the TWO-PASS FIFO Data Input and Output Ports in the implementation of a two-pass algorithm.

D.3 Protocol

Please list all deviations from the protocol described in Section 3 of the LWC API specification.

For example:

- no support for multiple consecutive segments of the type: AD, Plaintext, and Ciphertext (or Ciphertext||Tag if appropriate)
- special use for Reserved fields of an Instruction/Status or a Segment Header

- extra words added beyond the minimum number of words necessary to input AD, message, ciphertext, etc. of a given length (e.g., to always enter data in full block chunks)
- extra zeros added in the input words other than the last words of a given type (e.g., using less than `w` bits of each word)
- the use of a Length segment as a required input to an "online" algorithm, in which all lengths can be calculated as the AD/plaintext/ciphertext arrives and is processed
- a different format of the Length segment in an "offline" algorithm, such as AES-CCM, understood as an algorithm that requires the availability of the lengths of the AD and plaintext (ciphertext) in advance, before the authenticated encryption (decryption) starts.

D.4 Timing characteristics

Please list all deviations from the timing characteristics described in Section 4 of the LWC API specification. For example, a different order of bytes within a word of data bus.

2. Variants

File: `docs/variants.pdf` or `docs/variants.txt`
(depending on the file format used)

We define variants of the design as different versions of the design that

A. share the same synthesizable source code
B. share the same testbench
C. differ only with values of generics or constants.

Different variants may correspond to

- different algorithms of the same family
- different sizes of keys, nonces, tags, etc.
- different parameters of the interface, such as `w` and `sw`
- different hardware architectures (e.g., basic iterative, unrolled, folded, pipelined, etc.)

Please describe in this file all variants submitted for hardware benchmarking in the order of your preference (primary recommendations first).

Please start from:
Notation:

This notation should be common for all variants.

Two recommended sets of variables include:

Option 1:

Na, Nm, Nc, Nh : the number of blocks of associated data, plaintext, ciphertext, and hash message, respectively.

This option is recommended for implementations that have the same formulas for the execution times and latencies independently, whether their inputs are composed of complete blocks only or also contain incomplete blocks.

Option 2:

Na, Nm, Nc, Nh : the number of complete blocks of associated data, plaintext, ciphertext, and hash message, respectively

Ina, Inm, Inc, Inh : binary variables equal to 1 if the last block of the respective data type is incomplete, and 0 otherwise

Bla, Blm, Blc, Blh : the number of bytes in the incomplete block of associated data, plaintext, ciphertext, and hash message, respectively.

This option is recommended for implementations that have different formulas for the execution times and latencies dependent on whether their inputs are composed of complete blocks only or also contain incomplete blocks.

For each variant, provide at least the following information:
v<variant_number>: [variant name]
<variant number> should be unique for each variant of a given design submitted for benchmarking by the same group, even if variants use different source code.
Providing a variant name is optional.

Please follow with the full characterization of each variant described using the following items, including the respective headers:

a. Design goal
   Please use the description similar to that provided in the document "Suggested FPGA Design Goals."
b. Supported maximum sizes of inputs
   Use 2^{16}-1 (default), 2^{32}-1, 2^{50}-1, or other value specific to your algorithm and its implementation.
c. Reference software implementation
   Provide the name of the corresponding reference software implementation and the information where this reference implementation comes from (e.g., a specific submission package, SUPERCOP distribution, GitHub repository, etc.)
d. Non-default values of generics and constants
   List all non-default values of generics and constants. The meaning of generics does not need to be explained, but their names should be exactly the same as in the source code (e.g., as in VHDL packages), including the same capitalization.
e. Block sizes
   Please provide values of the associated data block size, message/ciphertext block size, and hash block size.
f. Execution times
   Please provide the exact formulas for the execution time of
   - authenticated encryption
   - authenticated decryption
   - hashing (if supported).

All execution times should be expressed in clock cycles.
The formulas should use variables defined at the beginning of the file, e.g.,
Na, Nm, Nc, Nh, Ina, Inm, Inc, Inh, Bla, Blm, Blc, Blh.
For authenticated encryption and decryption, the use of a new key should be assumed.
The starting time should be the moment when the *instruction Activate Key (ACTKEY)* is read by the LWC Core, using the bus `pdi_data`.
For hashing, the starting time should be the moment when the *instruction Hash* is read by the LWC Core, using the bus `pdi_data`.
For all three operations mentioned above, the ending time should be the moment when the *status word* is released by the LWC Core, using the bus `do_data`.

g. Latencies
Please provide formulas for the latency of
- authenticated encryption
- authenticated decryption.

All latencies should be expressed in clock cycles.
The formulas should use variables defined at the beginning of the file, such as
Nm, Nc, Inm, Inc, Blm, Blc.
The starting time should be the moment when the *first word of the plaintext/ciphertext* is read by the LWC Core, using the bus `pdi_data`.
The ending time should be the moment when the *first word of the corresponding ciphertext/plaintext* is released by the LWC Core, using the bus `do_data`.
We assume that the measurements are performed for the case of empty Associated Data.

h. Difference between execution times for a new key and the same key.
Please describe a difference between execution times of authenticated encryption/decryption for a new key and the same key.
For the latter case, the starting time should be the moment when the *instruction describing an operation to be performed* (authenticated encryption or authenticated decryption) is read by the LWC Core, using the bus `pdi_data`. The ending time should be the moment when the *status word* is released by the LWC Core, using the bus `do_data`.

<u>All formulas should contain only integers. No fixed-point or floating-point real numbers should be used. All formulas should be confirmed using functional simulation.</u>

<u>Example:</u>

```
Notation:

Na, Nm, Nc, Nh : the number of complete blocks of associated data,
plaintext, ciphertext, and hash message, respectively
Ina, Inm, Inc, Inh : binary variables equal to 1 if the last block of
the respective data type is incomplete, and 0 otherwise
Bla, Blm, Blc, Blh : the number of bytes in the incomplete block of
associated data, plaintext, ciphertext, and hash message, respectively.

v1: Xoodyak-128
```

```
a. Design goal

Support for authenticated encryption, decryption, and hashing.
Folded architecture, providing trade-off between throughput and area.
No BRAMs, no DSP units.
All RAMs using asynchronous read.

b. Supported maximum sizes of inputs

2^50-1

c. Reference software implementation

crypto_aead/xoodyakv1/ref
in https://bench.cr.yp.to/supercop/supercop-20200702.tar.xz

d. Non-default values of generics and constants

None

e. Block sizes

AD block size = 352 bits
Plaintext/Ciphertext block size = 192 bits
Hash block size = 128 bits

f. Execution times

Execution time of authenticated encryption:
269+259+(266*Na)+Ina(255+Bla/4)+(261*Nm)+Inm(255+Blm/4)+4

Execution time of authenticated decryption:
269+259+(266*Na)+Ina(255+Bla/4)+(261*Nc)+Inc(255+Blc/4)+4

Execution time of hashing:
12+(259*Nh)+Inh(255+Blh/4)+4

g. Latencies

Latency of authenticated encryption:
515

Latency of authenticated decryption:
515

h. Difference between execution times for a new key and the same key

259

v2: Xoodyak-384
. . .
```

Please do your best to limit the number of variants recommended for hardware benchmarking (e.g., by including only primary variants of the LWC algorithms declared in the algorithm specification, and/or by performing initial design space exploration using FPGA tools).

3. Synthesizable source code

Folder: `src_rtl`

Please place in this folder all synthesizable source files. Please place files being a part of the Development Package for the LWC Hardware API (such as `LWC.vhd`, `PreProcessor.vhd`, `PostProcessor.vhd`, etc.) in the subfolder LWC.

Please make sure to set the default values of generics in the top-level file (such as `LWC.vhd`) and the default values of constants in the corresponding package (such as `NIST_LWAPI_pkg.vhd`) to values specific to the primary variant of your algorithm.

Please also place in the same folder the file `source_list.txt`, containing the list of all design files in the bottom-up order, i.e., packages and low-level units first, and the top-level unit last.

4. Testbench

Folder: `src_tb`

Please place in this folder only your testbenches and any non-synthesizable source files used by your testbenches.

In case you use the universal testbench provided as a part of the Development Package, these files should include only `LWC_TB.vhd`.
Please also place in the same folder the file `source_list.txt`, containing the list of all testbench files in the bottom-up order, i.e., packages and low-level units first, and the top-level unit last.

5. Known-answer tests

Folder: `KAT`

Create subfolders, named `v1`, `v2`, `v3`, etc., corresponding to unique identifiers of variants, defined using recommendations described in Section 2 Variants.

In each respective subfolder, place test vector files you have used to verify your implementation of a particular variant.

It is recommended that all test vectors are described using two formats:

A. format accepted by the universal testbench `LWC_TB.vhd` (including the `pdi.txt`, `sdi.txt`, and `do.txt files`), generated *by default* by the cryptotvgen program, and
B. a simplified format, listing each input and expected output component (e.g., key, npub, ad, pt, ct, tag) using a sequence of hexadecimal digits located in the same line, e.g.
```
key     = 55565758595A5B5C5D5E5F6061626364
npub    = B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF
ad      =
pt      = FF
ct      = 76
tag     = FDB8FCCD8A5C78DC9445457B341F13B2
```

All test vectors should be placed in the same file `test_vectors.txt`, separated by at least one empty line. This file can be automatically generated by the cryptotvgen app by using the option `--human_readable`.

Please include in the aforementioned files `pdi.txt`, `sdi.txt`, `do.txt`, and `test_vectors.txt` only test vectors that successfully passed verification.
Place all test vectors that did not pass verification in separate files:
`pdi_failed.txt`, `sdi_failed.txt`, `do_failed.txt`, and
`test_vectors_failed.txt`.

6. Reference software implementation (optional)

Folder: `src_sw`

A reference software implementation used to generate test vectors.

7. Simulation scripts (optional)

Folder: `scripts`

Place in this folder all simulation scripts, such as `modelsim.tcl`, `vivado.tcl`, etc.

8. Block diagrams (optional)

Folder: `bd`

If possible, please include a simplified block diagram of the datapath for the primary variant of your algorithm. For consistency, and future use in publications, please consider using *Rules for Reduced Complexity Block Diagrams*, developed by William Diehl, and made available at
https://cryptography.gmu.edu/athena/LWC/Reduced_Complexity_Block_Diagrams.pdf

9. License (optional)

File: `LICENSE.txt`

Include in this file any licensing and copyright information that applies to your code.

10. Preliminary results (optional)

Folder: `results`

The GMU LWC Team is planning to perform hardware benchmarking of all Round 2 LWC candidates for FPGA technology only, using approach described in the Implementer's Guide to the LWC Hardware API, Section 7, Generation and Publication of Results.

In order to allow the comparison of designs in terms of Resource Utilization, the GMU implementation runs will enforce the use of no DSP units and no embedded block memories.

Each team is encouraged to produce and include in their submission the preliminary results of their own benchmarking runs, conducted using a similar approach (possibly without Minerva and ATHENa optimization runs).

These results will be used for a sanity check. In case better results are obtained as a result of GMU benchmarking, only these results will be reported. In case worse results are obtained as a result of GMU benchmarking, the authors of the implementations may be contacted with the requests for providing more optimal options of tools.

The FPGA results should be reported for the specific FPGA devices, from three major vendors, Xilinx, Intel, and Lattice Semiconductor, listed below.

Vendor	**Family**	**Device Code**
Xilinx	Artix-7	xc7a12tcsg325-3 (xc7a12t-3csg325)
Xilinx	Spartan-7	xc7s15cpga196-2 (xc7s15-2cpga196)
Intel	Cyclone 10 LP	10CL016-YU256C6
Lattice Semiconductor	ECP5	LFE5U-25F-6BG381C

The smallest device of Xilinx Artix-7 family was selected first to demonstrate ciphers' suitability for constrained environments. The devices of other families were selected in such a way to approximately match the resources of the smallest Artix-7 FPGA. All devices are believed to be sufficient to hold both unprotected and protected implementations of LWC candidates. The maximum speed grade has been chosen in order to make the results optimal and representative for the entire FPGA family.

For each FPGA device please report *at least* the maximum clock frequency and the resource utilization, including the numbers of

- LUTs, flip-flops (FFs), Slices, BRAMs (should be 0), and DSP slices (should be 0) for Xilinx FPGAs,
- LEs, flip-flops (registers), embedded memory in Kb (should be 0), and 18x18 multipliers (should be 0) for Intel FPGAs, and
- LUTs, flip-flops (FFs), sysMEM Blocks (should be 0), and 18x18 multipliers (should be 0) for Lattice Semiconductor FPGAs.

All results should be placed in a single file in the Excel, PDF, or ASCII format.

For your reference, we list below major resources available in each of these devices:

Xilinx:

Family	Artix-7	Spartan-7
Device	xc7a12tcsg325-3	xc7s15cpga196-2
LUTs	8,000	8,000
Flip-flops	16,000	16,000
Slices	2,000	2,000
18Kb BRAMs	40	20
DSP Slices	40	20
User I/Os	150	100

Intel:

Family	Cyclone 10 LP
Device	10CL016-YU256C6
LEs	15,408
Flip-flops	15,408
M9K Memory	56 blocks 504 Kbits
18 x 18 Multipliers	56
User I/Os	162

Lattice Semiconductor:

Family	ECP5
Device	LFE5U-25F-6BG381C
LUTs	24,000
Memory	56 sysMEM Blocks (18 Kbits each) 1008 Kbits
18 x 18 Multipliers	28
User I/Os	197

Other teams are encouraged to perform independent benchmarking for

- The same set of FPGA devices
- A different, independently selected set of FPGA devices.

Submission

Any other materials related to the submitted implementation, e.g., related papers or technical reports, should be placed in the `docs` folder.

In the case of the submission as a file, the top-level folder should be compressed to a single file

<LWC_Candidate_Name>_<Implementation_Team_Name>.zip

e.g., Xoodyak_CERG-GMU.zip.

Either the file itself or its location should be then submitted to the benchmarking lab.