Call for Protected Hardware Implementations of Finalists in the NIST Lightweight Cryptography Standardization Process

Cryptographic Engineering Research Group, George Mason University, U.S.A.

February 1, 2022

1 Introduction

We call for hardware implementations of finalists in the NIST Lightweight Cryptography Standardization Process resistant against side-channel attacks, such as power and electromagnetic analysis. The focus of this call is on the use of platform-independent algorithmic countermeasures. The submitted designs should demonstrate strong resistance against side-channel attacks when implemented on low-cost modern FPGAs, such as Artix-7 and Spartan-7 from Xilinx, Cyclone 10 LP from Intel, and ECP5 from Lattice Semiconductor. A potential for porting the designs to ASIC (Application-Specific Integrated Circuit) technology and demonstrating their resistance in this environment is highly desirable. All submitted implementations will be investigated by one or more Side-Channel Security Evaluation Labs. The primary goal of these labs will be to validate the security claims of the implementers. However, the tasks of these labs may extend beyond this major goal and include the key recovery attacks and various ways of assessing the secret information leakage. The labs will be able to choose freely from all implementations placed in the public domain. Additionally, implementers will be able to submit their designs to particular labs. Finally, labs will also be able to ask implementers for their deliverables.

2 Preliminary Requirements

Protected hardware implementations should follow the LWC Hardware API v1.1 or later. In this extended API, we assume that inputs and outputs are split into shares as shown in Fig. 1. Input that is not shared (e.g., an instruction or a segment header) should be put into share 1, with the remaining shares being set to zeros. The updated interface is shown in Fig. 2. In unprotected implementations, the public data input PDI accepts data of size w. For protected implementations, we modified this input, so it accepts pn shares of size w in parallel. The same holds for the data output DO, which now provides pn shares of size w. The number of shares on the secret data input SDI is denoted as sn, as it can differ from the number of shares on PDI.

A majority of common side-channel countermeasures require the consumption of randomness during cipher operations. Any randomness an LWC implementation needs can be provided by the random data input RDI, which is of size rw. This port, just like all the others, follows a simple FIFO protocol. Each read will provide rw bits. The value of rw can be arbitrary up to 2048 bits. Note that independent of how many random bits are used, our testbench assumes that all rw bits are used with each read.

We also assume that a deterministic random bit generator (DRBG) used as a source of fresh randomness is located outside of the protected LWC core. This way, resource utilization and power consumption of this DRBG can be excluded from the results generated during benchmarking.

We propose the following constraints on a first-order protected implementation of an LWC candidate: 8000 LUTs, 0 Block RAMs, and 0 DSP units of Artix-7 FPGAs. The number of LUTs corresponds to the smallest device of the Artix-7 family of FPGAs. This number is also consistent with the Round 2 limit on

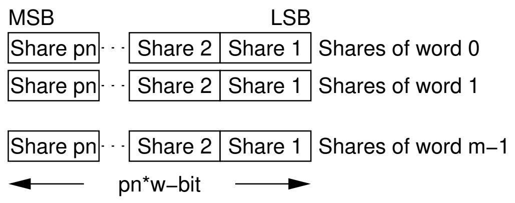

Figure 1: Pre-Shared Data

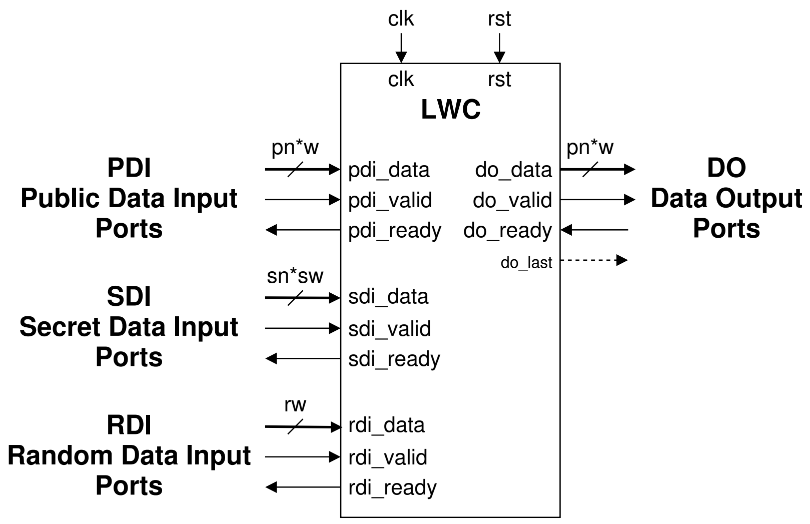

Figure 2: LWC API extended with Random Data Input (RDI)

the number of LUTs, set to 2000 LUTs, and the fact that first-order protected hardware implementations typically take 3-4x more hardware resources than the corresponding unprotected implementations.

Table 1: Proposed constraints on resource utilization

Type of Implementation	**#LUTs**	**#BRAMs**	**#DSP units**
Unprotected	≤ 2000	0	0
1st Order Protected	≤ 8000	0	0

3 Suggested Deliverables

To simplify benchmarking, security analysis, and further optimizations of protected hardware implementations, we propose a uniform way of publishing them on the web and submitting them to the benchmarking and security evaluation labs.

All protected implementations of a given candidate developed by a particular group should be stored in the same folder. This folder can either

1. become a basis of an online repository (e.g., a GitHub repository), in which case the submission consists of the repository URL including branch name, tag, or commit hash, or
2. be submitted as a single archive file (e.g., .zip) to the selected benchmarking and security evaluation labs.

Please name this folder using the following convention: `<LWC_Candidate_Name>_<Group_Name>`. Within this folder, please create the following structure of files and second-level folders:

```
LICENSE.txt [optional]
<variant_name_1>.toml
<variant_name_2>.toml
...
├── src_rtl
├── src_tb
├── KAT
└── docs
```

3.1 License (optional)

File: `LICENSE.txt`

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

3.2 Variant description files

File: `<variant_name>.toml`

In each of these files include the information fully characterizing a particular variant, prepared using the TOML (Tom's Obvious Minimal Language) file format[1].

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

- different algorithms of the same family
- different sizes of keys, nonces, tags, etc.

[1]https://toml.io/en/

- different parameters of the interface, such as w and sw
- different hardware architectures (e.g., basic iterative, unrolled, folded, pipelined, etc.),
- different protection methods against side-channel attacks,
- different orders of protection against side-channel attacks.

A TOML variant description file should include at least the required key/value pairs described in Appendix A. This file should capture attributes of an LWC hardware implementation to enable automated benchmarking and evaluation. An example of such a file for the GMU protected implementation of TinyJAMBU is given in Appendix B.

3.3 Source code

Folder: `src_rtl`

Place in this folder all synthesizable source files. 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.

3.4 Testbench

Folder: `src_tb`

Place in this folder only testbenches developed or modified by members of your group and any non-synthesizable source files used by your testbenches. The universal testbench provided as a part of the Development Package should be included only if it was modified.

3.5 Known-Answer Tests

Folder: `KAT`

Create subfolders named after the variant names corresponding to the unique identifiers of 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:

1. format accepted by the universal testbench, `LWC_SCA_TB.vhd` (including the `pdi_shared.txt`, `sdi_shared.txt`, and `do.txt` files). These files can be generated using the program `genShared` using as inputs default outputs from the `cryptotvgen` program, namely `pdi.txt`, `sdi.txt`, and `do.txt`.
2. a simplified format, listing each input share and expected output (e.g., key[1]..key[n], npub[1]..npub[n], ad[1]..ad[n], pt[1]..pt[n], ct, tag) using a sequence of hexadecimal digits located in the same line, e.g.

```
key[1]  = 55565758595A5B5C5D5E5F6061626364
...
key[n]  = 38AC7B4294D5A6B97C42E37A92EBCA52
npub[1] = B0B1B2B3B4B5B6B7B8B9BABBBCBDBEBF
...
npub[n] = D284C9E253AB629CD4385A8BDE25682E
ad[1]   =
...
ad[n]   =
pt[1]   = FF
```

```
...
pt[n]   = AC
ct   = 76
tag  = FDB8FCCD8A5C78DC9445457B341F13B2
```

In the latter case, all test vectors should be placed in the same file `test_vectors.txt`, separated by at least one empty line. This file can be generated by using the programs `cryptotvgen` and `genShared` with the option `--human_readable`.

Please include in all aforementioned files - `pdi_shared.txt`, `sdi_shared.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_shared_failed.txt`, `sdi_shared_failed.txt`, `do_failed.txt` , and `test_vectors_failed`.txt.

3.6 Documentation of a protected implementation

Folder: `docs`

Please provide one or more PDF files describing:

1. Protection Method
   (a) name of the applied countermeasure, such as Threshold Implementation (TI), Domain Oriented Masking (DOM), Unified Masking Approach (UMA), General Low-Latency Masking (GLM), etc.
   (b) corresponding primary reference describing this countermeasure (when applied to an arbitrary cryptographic algorithm)
2. Results of the Preliminary Security Evaluation
   (a) Attack/leakage assessment type
   (b) Number of traces used
   (c) Experimental setup
      i. Measurement platform and device-under-evaluation (e.g., ChipWhisperer, CW305 FPGA board)
      ii. Description of measurements, e.g., shunt resistor value, current probe specification, electromagnetic probe specification and placement
      iii. Usage of bandwidth limiters, filters, amplifiers, etc. and their specification
      iv. Frequency of operation
      v. Oscilloscope and its major characteristics (e.g., bandwidth)
      vi. Sampling frequency and resolution
      vii. Are sampling clock and design-under-evaluation clock synchronized?
   (d) Attack/leakage assessment characteristics
      i. Data inputs and performed operations
      ii. Source of random and pseudorandom inputs (e.g., TRNG type, hardware implementation of Trivium, etc.)
      iii. Trigger location relative to the execution start time of the algorithm
      iv. Time required to collect data for a given attack/leakage assessment
      v. Total time of the attack/assessment
      vi. Total size of all traces (if stored)
      vii. Availability of raw measurement results
   (e) Attack-specific characteristics
      i. Power model

        ii. Attack point

    (f) Documentation of results

        i. Graphs illustrating the obtained results, e.g., Test Vector Leakage Assessment (TVLA) graphs, minimum traces to disclosure (MTD) graphs, guessing entropy (GE), etc.
        ii. Attack scripts.

4 Proposed Timeline

- Call for protected hardware implementations
  - First draft – December 13, 2021
  - Discussion on the `lwc-forum`
  - Final version – January 17, 2022
- Deadline for protected hardware implementations
  - Submission to the selected benchmarking and security evaluation labs – March 15, 2022
  - Announcement on the `lwc-forum` (optional) – March 15, 2022
- Security Evaluation Lab Reports
  - Preliminary version of the report – April 30, 2022
  - Final version of the report – June 30, 2022
- Benchmarking of Protected Implementations in terms of Throughput, Resource Utilization, Power, and Energy per bit
  - Preliminary version of the report – April 30, 2022
  - Final version of the report (considering relative security of evaluated implementations) – July 30, 2022.

5 Contact Information

Jens-Peter Kaps and Kris Gaj
Cryptographic Engineering Research Group
George Mason University
jkaps@gmu.edu , kgaj@gmu.edu
https://cryptography.gmu.edu

A Variant Description File

A TOML variant description file should contain a subset of key/value pairs listed below. Underlined keys should always be specified. Other keys are only required when a scheme or implementation uses a value different from the specified *Default* value.

A Schema describing the structure of the variant description file, as well as a Python script to assist validation, are available at `https://github.com/GMUCERG/lwc_description_schema`.

- **`name`** *(string)*: A unique identifier for the design. It can consist of English letters, digits, dashes, and underscores and must start with a letter.
- **`description`** *(string)*: A short description of the design.
- **`author`** *(string or array of string)*: The name(s) of the developer(s) of this implementation.
- **`url`** *(string)*: Uniform Resource Locator pointing to a webpage or source repository associated with this design or its author(s).
- **`license`** *(string or array of string)*: License or licenses covering this design's use and distribution. Use SPDX (ISO/IEC 5962:2021) short identifiers when applicable. *Examples:* `"SHL-2.1"`
- **`version`** *(string)*: Design version. *Examples:* `"0.0.1"`
- **`rtl`**: Details of the synthesizable RTL code.
  - **`sources`** *(array of string)*: Non-empty list of HDL source file paths in correct compilation order. All paths must be relative to the location of the configuration file. Path separator is `/` (slash) on all platforms. Paths are case-sensitive and should not contain whitespaces. HDL language is inferred from the file extension.
  - **`includes`** *(array of string)*: Non-empty list of HDL include file paths, such as Verilog headers. Include files are not directly compiled but need to be present for elaboration of design. Order is arbitrary. All paths must be relative to the location of the configuration file. Path separator is `/` (slash) on all platforms. Paths are case-sensitive and should not contain whitespaces. *Default:* `[]`
  - **`top`** *(string)*: Name of top-level RTL entity/module. *Default:* `LWC`
  - **`clock`**: Top level RTL clock signal. Only a single clock is supported by LWC API.
    - **`pin`** *(string)*: Name of the top-level clock input. *Default:* `clk`
  - **`reset`**: Top level reset signal. Only a single reset is supported by LWC API.
    - **`pin`** *(string)*: Name of top-level reset input. *Default:* `reset`
    - **`active_high`** *(boolean)*: Polarity of the reset signal. Active-high (positive) if true, otherwise active-low. *Default:* `true`
    - **`asynchronous`** *(boolean)*: Whether reset is asynchronous with respect to `rtl.clock`. *Default:* `false`
  - **`parameters`**: Top-level design parameters or generics specified as a key-value map. The default value of each parameter is overridden by synthesis tool, simulator, testbench, or wrapper. For the best tool compatibility, we only support integer and string values. *Default:* {} *Examples:* {`"G_NUM_SHARES": 2, "G_PARALLEL": 8`}
- **`tb`**: Details of test-bench used for verification of top-level design. [Optional].
  - **`sources`** *(array of string)*: Source files used only for verification. Should *not* contain any of the files included in 'rtl.sources'.
  - **`includes`** *(array of string)*: HDL include file paths. *Default:* `[]`
  - **`top`** *(string)*: Name of top-level test entity or module.
  - **`parameters`**: Testbench parameter or generics specified as a key-value map. The default value of each parameter is overridden by the simulator. For the best tool compatibility, we only support integer and string values. *Default:* {} *Examples:* {`"G_TEST_MODE": 0`}

- **`language`**: Information about Hardware Description/Design Language(s) used.
  - **`vhdl`**: Common VHDL features supported by all VHDL source files. VHDL files must have a `.vhd` or `.vhdl` extension.
    - **`version`** *(string)*: VHDL language standard. *Supported values:* `1993`, `2000`, `2002`, `2008` *Default:* `1993`
    - **`synopsys`** *(boolean)*: Use of non-standard Synopsys packages which were placed in the IEEE namespace, e.g. 'std_logic_arith'. Dependence on such packages is *strongly* discouraged. *Default:* `false`
  - **`verilog`**: Common Verilog (pre-SystemVerilog) language features supported by all Verilog source files. Verilog files must have a `.v` extension.
    - **`version`** *(string)*: Verilog language standard. *Supported values:* `1995`, `2001` *Default:* `2001`
  - **`systemverilog`**: SystemVerilog (IEEE 1800-2005 and onwards) language features supported by all SystemVerilog source files. SystemVerilog files must have a `.sv` extension.
    - **`version`** *(string)*: SystemVerilog language standard. *Supported values:* `2005`, `2009` *Default:* `2009`
- **`lwc`**: LWC-specific meta-data.
  - **`aead`**: Details about the AEAD scheme and its implementation.
    - **`algorithm`** *(string)*: Name of the implemented AEAD algorithm based on SUPERCOP convention. *Examples:* `"giftcofb128v1"`, `"romulusn1v12"`, `"gimli24v1"`
    - **`input_sequence`**: Order in which different input segment types should be fed to PDI.
      - **`encrypt`** *(array of string)*: Sequence of inputs during encryption. *Default:* `['npub', 'ad', 'pt', 'tag']`
      - **`decrypt`** *(array of string)*: Sequence of inputs during decryption. *Default:* `['npub', 'ad', 'ct', 'tag']`
  - **`hash`**
    - **`algorithm`** *(string)*: Name of the hashing algorithm based on SUPERCOP convention. Empty string if hashing is not supported. *Default:* " *Examples:* `""`, `"gimli24v1"`
    - **`digest_bits`** *(integer)*: Size of hash digest (output) in bits. *Default:* `128`
  - **`ports`**: Description of LWC ports.
    - **`pdi`**: Public Data Input port.
      - **`bit_width`** *(integer)*: Width of each word of PDI data in bits (`w`). The width of 'pdi_data' signal would be `pdi.bit_width × pdi.num_shares` (`w × n`) bits. *Minimum:* `8` *Maximum:* `32` *Default:* `32`
      - **`num_shares`** *(integer)*: Number of PDI shares (`n`).
    - **`sdi`**: Secret Data Input port.
      - **`bit_width`** *(integer)*: Width of each word of SDI data in bits (`sw`). The width of `sdi_data` signal would be `sdi.bit_width × sdi.num_shares` (`sw × sn`) bits. *Minimum:* `8` *Maximum:* `32` *Default:* `32`
      - **`num_shares`** *(integer)*: Number of SDI shares (`sn`).
    - **`rdi`**: Random Data Input port.
      - **`bit_width`** *(integer)*: Width of the `rdi` port in bits (`rw`), 0 if the port is not used. *Minimum:* `0` *Maximum:* `2048`
  - **`sca_protection`**: Implemented countermeasures against side-channel attacks.
    - **`target`** *(array of string)*: Type of side-channel analysis attack(s) against which this design is assumed to be secure. *Examples:* `["spa", "dpa", "cpa", "timing"]`, `["dpa", "sifa", "dfia"]`

- **`masking_schemes`** *(array of string)*: Masking scheme(s) applied in this implementation. Could be name/abbreviation of established schemes (e.g., ”DOM”, ”TI”) or reference to a publication. *Default:* `[]` *Examples:* `["TI"]`, `["DOM", "https://eprint.iacr.org/2022/000.pdf"]`
- **`order`** *(integer)*: Claimed order of protectcion. 0 means unprotected. *Default:* `0`

B Sample Variant Description File

```
name = "TinyJAMBU-DOM1-v1"
description = "TinyJAMBU with 1st order masking"
author = ["Sammy Lin", "Abubakr Abdulgadir"]
url = "https://github.com/GMUCERG/TinyJAMBU-SCA"
license = "GPL-3.0"
version = "0.1.0"

[rtl]
sources = [
    "src_rtl/design_pkg.vhd",
    "src_rtl/LWC_pr/NIST_LWAPI_pkg.vhd",
    "src_rtl/tinyjambu_dp_ops.vhd",
    "src_rtl/reg_n.vhd",
    "src_rtl/dom_mul_reg.vhd",
    "src_rtl/dom_mul.vhd",
    "src_rtl/dom_mul_dep.vhd",
    "src_rtl/dom_nlfsr_reg_feed.vhd",
    "src_rtl/dom_nlfsr.vhd",
    "src_rtl/tinyjambu_datapath.vhd",
    "src_rtl/tinyjambu_control.vhd",
    "src_rtl/Register_s.vhd",
    "src_rtl/CryptoCore.vhd",
    "src_rtl/LWC_pr/pdi_sipo.vhd",
    "src_rtl/LWC_pr/sdi_sipo.vhd",
    "src_rtl/LWC_pr/StepDownCountLd.vhd",
    "src_rtl/LWC_pr/key_piso.vhd",
    "src_rtl/LWC_pr/data_piso.vhd",
    "src_rtl/LWC_pr/PreProcessor.vhd",
    "src_rtl/LWC_pr/data_sipo.vhd",
    "src_rtl/LWC_pr/PostProcessor.vhd",
    "src_rtl/LWC_pr/do_piso.vhd",
    "src_rtl/LWC_pr/fwft_fifo.vhd",
    "src_rtl/LWC_pr/LWC.vhd",
]
# includes = []
top = "LWC"
# clock.port = "clk"
# reset.port = "reset"
# reset.active_high = true
# reset.asynchronous = false
# parameters = { G_GENERIC1 = 123 }

[language]
vhdl.version = "2008"
# vhdl.synopsys = false
# verilog.version = "2001"
# systemverilog.version = "2009"

# [tb]
# sources = ["LWC_TB_package.vhd", "LWC_SCA_TB.vhd"]
# includes = []
# top = "LWC_TB"

[lwc.aead]
algorithm = "tinyjambu128"
# key_bits = 128
# npub_bits = 128
# tag_bits = 128
# input_sequence.encrypt = [ "npub", "ad", "pt", "tag" ]
# input_sequence.decrypt = [ "npub", "ad", "ct", "tag" ]

# [lwc.hash]
# algorithm = "" ###### (hashing is not supported)
# digest_bits = 128
```

```
[lwc.ports]
# pdi.bit_width = 32
pdi.num_shares = 2
rdi.bit_width = 192
# sdi.bit_width = 32
sdi.num_shares = 2

[lwc.sca_protection]
target = ["timing", "sda", "dpa"]
order = 1
```