General Framework for Evaluating LWC Finalists in Terms of Resistance to Side-Channel Attacks

Jens-Peter Kaps, Kris Gaj,
Abubakr Abdulgadir and Kamyar Mohajerani

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

Abstract. Side-channel resistance is one of the primary criteria identified by NIST for use in evaluating candidates in the Lightweight Cryptography (LWC) Standardization process. In Rounds 1 and 2 of this process, when the number of candidates was still substantial (56 and 32, respectively), evaluating this feature was close to impossible. With ten finalists remaining, side-channel resistance and its effect on the performance and cost of practical implementations become of utmost importance. In this paper, we propose a general framework for evaluating side-channel resistance of LWC candidates using resources, experience, and general practices of the cryptographic engineering community developed over the last two decades. The primary features of our approach are a) self-identification and self-characterization of side-channel security evaluation labs, b) distributed development of protected software and hardware implementations, matching certain high-level requirements and deliverable formats, and c) dynamic and transparent matching of evaluators with implementers in order to achieve the most meaningful and fair evaluation reports.

Keywords: Lightweight Cryptography · authenticated ciphers · hash functions · hardware · software · benchmarking

1 Introduction

NIST has specified resistance to Side-Channel Analysis (SCA) as one of the primary criteria for evaluating candidates in the Lightweight Cryptography (LWC) Standardization Process [1]. To assist NIST in evaluating finalists in this process, we have developed the following three calls:

1. Call for Side-Channel Security Validation Labs
2. Call for Protected Hardware Implementations, targeting low-cost modern FPGAs
3. Call for Protected Software Implementations, targeting low-cost modern embedded processors.

The general idea is that no single group is likely to have resources and expertise to develop and evaluate SCA-protected implementations of all 10 finalists. Additionally, self-evaluation by developers may be insufficient and/or error-prone. Therefore, it is the collective responsibility of the cryptographic engineering community to contribute to the evaluation process and make it as transparent and fair as possible. Contributions by multiple groups will make:

- each group's workload more manageable;

- coverage of implementation platforms more complete;
- results more credible.

These contributions are strongly encouraged and justified by at least the following factors:

- The new LWC standard is likely to be used for decades. The development and evaluation of protected implementations are scheduled to last about six months. Choosing the right algorithm can save the community countless man-hours necessary to secure implementations of a hard-to-protect standard or start a new standardization process from scratch.
- It is a joint project that all experts in the field can focus on in the limited amount of time devoted to analysis. Most implementations will be, by nature, open-source. Most evaluations will be very transparent and reproducible. This process is likely to reveal and highlight some implementation and evaluation methods that rarely get fully disclosed and published. The community can benefit tremendously by saving thousands of man-hours spent on rediscovering previously-known tricks.
- Automated insertion of countermeasures is highly desirable (especially considering the very short period reserved for developing protected implementations). Any insights gained through these developments may lead to tremendous progress in the field of Computer-Aided Design (CAD) tools for SCA.
- The developed protected implementations can become benchmarks for new attacks and leakage assessment methods that can be discovered and published in years to come.
- Research on NIST standards is highly visible. Participants are rewarded with recognition by the cryptographic community that may translate to new collaboration, funding, and publication opportunities.
- The project is very suitable for a Master's Thesis or a part of a Ph.D. thesis. All results should be easily publishable. Student participants may be rewarded with attractive job offers.
- The project is a source of excellent topics for individualized projects in various cryptographic engineering, digital system design, and programming courses.
- Some members of each participating group may decide to commercialize their contributions, and the project's high visibility may help attract investors and collaborators.

2 Side-Channel Security Evaluation Labs

2.1 General Idea

We called for groups capable and willing to serve as side-channel security evaluation labs to identify their capabilities and contribute to the evaluation process. Our draft call was sent for comments to `lwc-forum` in December 2021. The final version of this call was published on January 18, 2022. The deadline for submitting lab specifications was initially set to February 28, 2022, and then extended to March 15, 2022, for groups that expressed initial interest.

The assumption was that submitters should have access to the equipment used for side-channel leakage assessment and/or attacks, experience, and human resources necessary to perform security analysis. Suggested devices used for evaluating hardware implementations were low-cost modern FPGAs, such as Artix-7 and Spartan-7 from Xilinx, Cyclone 10 LP from Intel, and ECP5 from Lattice Semiconductor. Suggested embedded processors used for evaluating software implementations were ARM Cortex-M4F, RISC-V (e.g., RV32IMAC), Microchip 8-bit AVR, and TI MSP430. A particular lab could specialize in evaluating only hardware implementations, only software implementations, or both.

2.2 Suggested Deliverables

Suggested deliverables included:

1. Equipment and Software Used
   - (a) General type of the evaluation platform, e.g., Rambus DPA Workstation, Riscure Inspector, NewAE ChipWhisperer, SAKURA, SASEBO, FOBOS
   - (b) The exact names and versions of all FPGA or embedded processor boards used to host the protected implementations (victim boards)
   - (c) The exact names and versions of all FPGA and embedded processor boards used to support measurements
   - (d) Oscilloscope and its major characteristics (e.g., bandwidth)
   - (e) Current and electromagnetic probes
   - (f) Usage of bandwidth limiters, filters, amplifiers, etc. and their specification
   - (g) Information on whether sampling clock and design-under-evaluation clock are synchronized
   - (h) Names and versions of programs used for evaluating side-channel resistance.
2. Supported Leakage Assessment Methods
   - (a) Type of the method, e.g., TVLA (Test Vector Leakage Assessment) a.k.a. Welch's t-test, Pearson's χ^2-test, deep learning leakage assessment (DL-LA), etc.
   - (b) Approximate number of traces used in evaluations of authenticated ciphers
   - (c) Typical clock frequency of the device-under-evaluation
   - (d) Sampling frequency and resolution
   - (e) Graphical representation of results, e.g., TVLA graphs, χ^2 graphs, etc.
3. Supported Attacks
   - (a) Types of Power Analyses, e.g., Simple Power Analysis (SPA), Differential Power Analysis (DPA), Correlation Power Analysis (CPA), Template Attacks (TA), Mutual Information Analysis (MIA), etc.
   - (b) Types of Electromagnetic Analyses
   - (c) Types of Fault Analyses, e.g., Differential Fault Analysis (DFA), Fault Sensitivity Attack (FSA), Differential Fault Intensity Analysis (DFIA), and Fault Behavior Analysis (FBA)
   - (d) Graphical representation of results, e.g., the minimum traces to disclosure (MTD) graphs.
4. Ability to generate and publish raw measurements to be analyzed by other groups
5. Support for side-channel analysis as service, with the feedback provided to designers of protected implementations during the development process

Table 1: Side-Channel Security Evaluation Labs Targeting Hardware Implementations

No.	Team	Evaluation Platform	Target FPGA Family	Target Boards	Leakage Assessment Methods	Attacks
1	IAIK, TU Graz, Austria	NewAE ChipWhisperer	Artix-7	NewAE CW305	t-test	
2	Telecom Paris, France		Spartan-6, Virtex-5, Virtex-II Pro, Stratix II	SASEBO-W, SASEBO-GII, SASEBO-G/ SASEBO-R, SASEBO-B	NICV, t-test, χ^2-test, DL-LA	SPA, DPA, CPA, MIA, TA, LRA; SEMA, DEMA, CEMA; SFA, DFA, FSA, DFIA, FBA
3	CCSL, Shanghai Jiao Tong University, China	Riscure Inspector, NewAE ChipWhisperer, SAKURA	Kintex-7, Spartan-6	SAKURA-G, SAKURA-X	t-test, χ^2-test, DL-LA	CPA, TA, MIA, DL-based methods
4	HSCP Lab, Tsinghua University, Beijing, China	SAKURA	Kintex-7, Spartan-6	SAKURA-G, SAKURA-X	NICV, t-test, χ^2-test	SPA, DPA, CPA, MIA, TA, LRA,
5	CESCA Lab, Radboud University, the Netherlands	Riscure Inspector, NewAE ChipWhisperer, Jupyter notebook scripts	Artix-7, Spartan-6	NewAE CW305, SAKURA-G	t-test, χ^2-test, DL-LA	SPA, DPA, CPA, TA; DEMA; DFA, FI attacks
6	Secure-IC, France	Secure-IC Analyzr, SAKURA	Spartan-6	SAKURA-G	Tests specified in ISO/IEC 17825:2016	
7	CERG, George Mason University, USA	FOBOS3	Artix-7	NewAE CW305	t-test	
8	Ruhr-Universitat Bochum, Germany	SILVER and other simulation-based probing security leakage-detection tools			simulation-based probing security evaluation	

Table 2: Side-Channel Security Evaluation Labs Targeting Software Implementations

No.	**Team**	**Evaluation Platform**	**Target Processors**	**Leakage Assessment Methods**	**Attacks**
1	IAIK, TU Graz, Austria	NewAE ChipWhisperer	ARM Cortex-M4F	t-test	
2	Telecom Paris, France	NewAE ChipWhisperer	ARM Cortex-M0, ARM Cortex-M4F, ATxmega128D4	NICV, t-test, χ^2-test, DL-LA	SPA, DPA, CPA, MIA, TA, LRA; SEMA, DEMA, CEMA; SFA, DFA, FSA, DFIA, FBA
3	LSSS, OTH Regensburg, Germany		ATmega328P, ARM Cortex-M3, RISC-V GD32VF103CBT6, ARM Cortex-M7, Tensilica Xtensa LX6	t-test	
4	CCSL, Shanghai Jiao Tong University, China	Riscure Inspector, NewAE ChipWhisperer	ARM Cortex-M4F, ATxmega128D4, ATmega128A	t-test, χ^2-test, DL-LA	CPA, TA, MIA, DL-based methods
5	HSCP Lab, Tsinghua University, Beijing, China		ARM Cortex-M4F, ARM Cortex-M3	NICV, t-test, χ^2-test	SPA, DPA, CPA, MIA, TA, LRA
6	CESCA Lab, Radboud University, the Netherlands	Riscure Inspector, NewAE ChipWhisperer, Jupyter notebook scripts	ARM Cortex-M4F, ATxmega128D4	t-test, χ^2-test, DL-LA	SPA, DPA, CPA TA; DEMA; DFA, FI attacks

6. Short description of the personnel and its qualifications

7. Intended period of the lab operation

8. Contact information.

2.3 Lab Specifications

The summary of lab specifications submitted in response to our call by March 20, 2022, is given in Tables 1 and 2. We have received a total of 9 submissions, specifying

- 5 labs that support both software and hardware implementations,

- 3 labs that support only hardware implementations, and

- 1 lab that supports only software implementations.

The detailed specifications are posted on our ATHENa Lightweight Cryptography web page at `https://cryptography.gmu.edu/athena/index.php?id=LWC`.

In Table 1, we summarize major capabilities of the labs targeting hardware implementations in terms of the Evaluation Platform, Target FPGA Family, Target Board, Leakage Assessment Methods, and Key Recovery Attacks.

The most popular Evaluation Platforms are NewAE ChipWhisper and SAKURA, declared by 4 out of 8 specification submitters. Next comes Riscure Inspector, used by two labs, followed by Secure-IC Analyzr, FOBOS3, and SILVER, used by one. SILVER and related probing security leakage-detection tools are the only tools based entirely on simulation.

Most of the labs support Xilinx 7 Series FPGA families, such as Artix-7 and Kintex-7. Two labs support only Spartan-6 and earlier FPGA families based on four-input Look-Up Tables (LUTs). Among the Target Boards, the most popular are SAKURA boards and NewAE CW305.

The most widely supported Leakage Assessment Method is Welch's t-test a.k.a. TVLA (Test Vector Leakage Assessment) [2]–[7]. Four labs support a newer and supplementary Pearson's χ^2-test introduced in [8]. The team representing Secure-IC uses tests specified in ISO/IEC 17825:2016. These tests are described and critically analyzed in [9]. Two labs declare support for NICV: Normalized Inter-Class Variance for Detection of Side-Channel Leakage [10], [11]. Three labs list among their methods DL-LA: Deep Learning Leakage Assessment, defined in [12]. The team from Ruhr-Universitat Bochum relies on the simulation-based SILVER (introduced in [13] and available at [14]) and other, unpublished probing security leakage-detection tools.

Four labs support attacks. Out of them, two limit attacks to power-based methods, such as Simple Power Analysis (SPA), Differential Power Analysis (DPA), Correlation Power Analysis (CPA), Template Attacks (TA), Mutual Information Analysis (MIA), and Deep Learning (DL)-based methods [15]. The remaining two support also several types of Electromagnetic Analysis (e.g., Simple Electromagnetic Analysis - SEMA, Differential Electromagnetic Analysis - DEMA, Correlation Electromagnetic Analysis - CEMA) and Fault Analysis (e.g., Differential Fault Analysis - DFA, Fault Sensitivity Attack - FSA, Differential Fault Intensity Analysis - DFIA, and Fault Behavior Analysis - FBA).

In Table 2, we summarize major capabilities of the labs targeting software implementations in terms of the Evaluation Platform, Target Processors, Leakage Assessment Methods, and Key Recovery Attacks.

Four out of six labs rely on the NewAE ChipWhisperer platform. The most supported target processor is ARM Cortex-M4F, listed by 5 out of 6 labs. Three labs support ATxmega128D4, and two ARM Cortex-M0. The Laboratory for Safe and Secure Systems

(LSSS) at OTH Regensburg, Germany, has listed five different target processors but is tentatively planning to support only two.

In terms of the Leakage Assessment Methods, all labs support t-test, and 4 out of 6 also χ^2-test. The third most popular test is Deep Learning Leakage Assessment (DL-LA), supported by half of the current labs.

The support for attacks is exactly the same as in the case of hardware implementations, as all labs supporting attacks target both software and hardware implementations.

3 Protected Hardware Implementations

3.1 Introduction

We submitted a draft version of the Call for Protected Hardware Implementations to `lwc-forum` on December 13, 2021. After analyzing all received comments and incorporating the best-received suggestions, we have posted a final version of this call on the GMU Lightweight Cryptography website on January 18, 2022. According to the call, 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 has been highly desirable. All submitted implementations will be investigated by one or more Side-Channel Security Evaluation Labs.

3.2 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 to accept 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 actually used, our testbench assumes that all rw bits are used with each read.

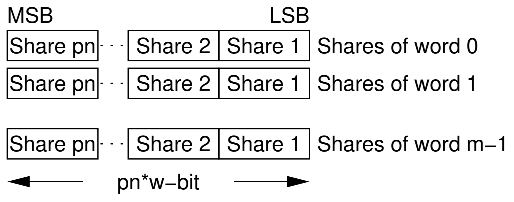

Figure 1: Pre-Shared Data

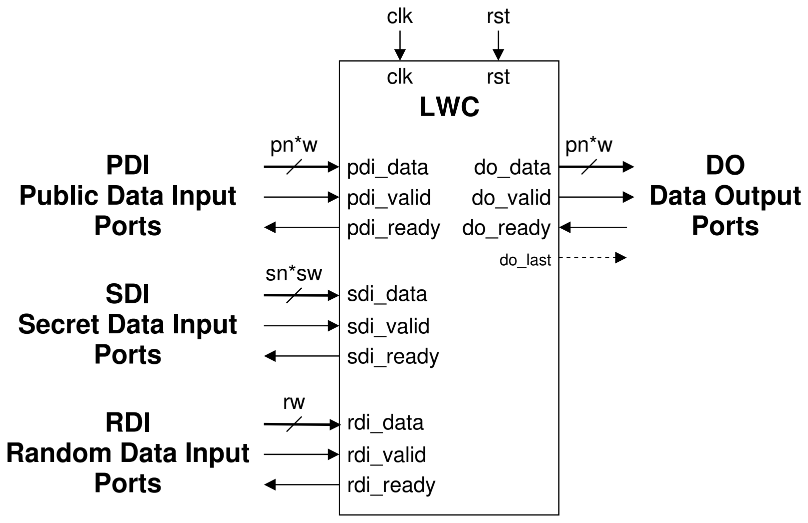

Figure 2: LWC API extended with Random Data Input (RDI)

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. The important advantages of this approach include:

- ability to share DBRG with other units (e.g., for the generation of nonces, protection of other units, e.g., those implementing public-key cryptography, etc.)
- ease of replacing the type of DBRG (e.g., due to compliance with other standards, validation requirements, evolving understanding of how cryptographically strong the DBRG used for refreshing randomness must be, etc.)
- we are interested in evaluating/benchmarking LWC candidates and not DBRGs. The final NIST LWC selection itself could become the basis of future lightweight DBRGs.
- Concurrent operation of the DBRG circuit could introduce additional noise in the measurements and make leakage detection more difficult. There is no guarantee that this type of noise by itself could hinder an actual attack scenario, but it is likely to make the leakage evaluation more difficult (more traces, more computations, or more expensive measurement equipment).

Our testbench will count how many random bits were consumed by the protected implementation during its operation, and we will use this information, together with the width of the rdi_data bus, to differentiate between various protected designs. Specifically, the total number of consumed fresh random bits will be one of the primary items on the list of reported evaluation metrics. In principle, we do not mind allowing designers to improve this metric by placing the source of some or all random bits inside of their implementations. At the same time, designs using significantly different assumptions may be quite challenging to rank.

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

Table 3: Proposed constraints on resource utilization

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

3.3 Suggested Deliverables

To simplify benchmarking, security analysis, and further optimizations of protected hardware implementations, we proposed 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 are expected to 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.

This folder should be named using the following convention:
`<LWC_Candidate_Name>_<Group_Name>`.
Within this folder, the submitters should create the structure of files and second-level folders shown in Fig. 3.

`LICENSE.txt` should include any licensing and copyright information that applies to the code. `<variant_name>.toml` is expected to include information fully characterizing a particular variant, encoded using the TOML (Tom's Obvious Minimal Language) file format[1]. This file should capture attributes of an LWC hardware implementation to enable automated benchmarking and evaluation. 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.
- 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.

`src_rtl` is a folder that should contain all synthesizable source files, and `src_tb` a folder containing testbenches developed or modified by a given submitter and any non-synthesizable source files used by these testbenches.

The `KAT` folder should include subfolders named after the variant names corresponding to the unique identifiers of variants. Each respective subfolder is expected to contain test vector files used to verify the implementation of a particular variant.

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

```
<Candidate_Name>_<Group_Name>
├── LICENSE.md       # or LICENSE.txt
├── README.md        # or README.txt
├── <variant_name_1>.toml     # TOML variant description file(s)
├── <variant_name_2>.toml
├── ...
├── src_rtl
│   ├── LWC          # common RTL sources
│   ├── <variant_name_1>     # sources specific to variant_1
│   │   ├── design_pkg.vhd
│   │   ├── LWC_config.vhd
│   │   ├── CryptoCore.vhd
│   │   └── ...
│   ├── <variant_name_2>     # sources specific to variant_2
│   └── ...
├── src_tb
│   ├── LWC_TB.vhd
│   └── ...
├── KAT          # Test vectors
│   ├── <variant_name_1>
│   │   ├── pdi_shared_2.txt
│   │   ├── sdi_shared_2.txt
│   │   ├── do_shared_2.txt
│   │   └── ...
│   ├── <variant_name_2>
│   │   └── ...
│   └── ...
├── cref     # C reference software implementation
│   └── ...
└── docs     # Additional documentation
    └── ...
```

Figure 3: Recommended format of deliverables for protected hardware implementation

The `docs` folder should include one or more PDF files describing:

1. Protection Method
2. Results of the Preliminary Security Evaluation, including at least
   (a) Attack/leakage assessment type
   (b) Number of traces used
   (c) Experimental setup
   (d) Attack/leakage assessment characteristics
   (e) Attack-specific characteristics
   (f) Documentation of results.

4 Protected Software Implementations

4.1 Introduction

We have called for software implementations of finalists resistant against side-channel attacks such as power and electromagnetic analysis, using the same timeline as in the case of hardware implementations. The focus of our call is on the use of platform-independent algorithmic countermeasures. The submitted code should demonstrate strong resistance against side-channel attacks when executed on low-cost modern embedded processors, such as ARM Cortex M4F, RISC-V (e.g., RV32IMAC), Microchip 8-bit AVR, and TI MSP430. This code can contain assembly language instructions specific to a given Instruction Set Architecture (ISA).

4.2 Requirements

Protected software implementations should use the standard NIST API defined in Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process, published in August 2018[2]. Protected implementations should not use `nsec`, beyond specifying it as an argument of
`crypto_aead_encrypt()` and `crypto_aead_decrypt()` set to NULL.

Considering the short amount of time devoted to analyzing protected implementations by the Side-Channel Security Evaluation Labs, it is important that all submissions can be evaluated using the Test Vector Leakage Assessment (TVLA) method, a.k.a., Welch's t-test. To make it possible, we have suggested that at least one variant of the protected implementation is designed to accommodate this test.

To simplify power correlation evaluation via Welch's t-test without spurious correlation from sharing and un-sharing operations, we proposed the clear division of the protected implementation into three functions:
`generate_shares_encrypt()`, `crypto_aead_encrypt_shared()`, and
`combine_shares_encrypt()`, for encryption, and
`generate_shares_decrypt()`, `crypto_aead_decrypt_shared()`, and
`combine_shares_decrypt()`, for decryption.

Only `crypto_aead_encrypt_shared()` for encryption and
`crypto_aead_decrypt_shared()` for decryption should be used for leakage assessment.

The `generate_shares_encrypt()` function should allow the division of all inputs to encryption into Boolean shares. The `generate_shares_decrypt()` function should allow the division of all inputs to decryption into Boolean shares. This function should first pad

[2]https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf

the last word of the ciphertext (not including the tag) with zeros. The tag should then follow, starting on a boundary of a 32-bit word. Only after this initial preprocessing the ciphertext and tag should be masked. The number of shares may be different for each input, as defined in `api.h`.

4.3 Suggested Deliverables

To simplify benchmarking, security analysis, and further optimizations of protected software implementations of LWC algorithms, we have proposed 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. The contents of this folder should follow the guidance of the NIST LWC specification[3] section 3.5.1 (AEAD) and 3.5.2 (Hash). Per this specification, in addition to the protected implementation, submissions should include a reference implementation (`ref`) and Known-answer-test file (`KAT`) to allow existing tools to verify implementation correctness.

Source code for protected implementations should be provided as C99 standard C suitable for compilation, linkage, and assembly using standard tooling (e.g., GCC) for the target architecture(s). Architecture specific optimizations (e.g., assembly language) may additionally be provided to demonstrate performance enhancement. If the use of assembly language is intended to enhance resistance against side-channel attacks, then this should be stated explicitly in the supporting documentation.

Submissions for side-channel evaluation should not depend on any external headers or libraries, including cryptographic libraries (e.g., OpenSSL), outside of the C99 standard, with the exception of the `randombytes.h` header from SUPERCOP, which may be used for masking or sharing.

To ensure deterministic results, the implementation of the function randombytes() should be provided by the test harness (not the submission) and initialized prior to execution of the cryptographic algorithm. Implementations should use the call to this function directly, rather than including a DRBG (based on AES, ChaCha, SHAKE, etc.) within the implementation.

The structure of a compliant submission is shown in Fig. 4.

The `<Candidate_and_Variant_Name>` is a single name incorporating the name of the candidate and its specific variant. Different variants correspond to

- different algorithms of the same family
- different parameter sets, such as sizes of keys, nonces, tags, etc.

The following optional metadata files may be included alongside source code within a 3rd-level folder:

1. `goal_powersca_1st` - When present, this file[4] indicates the implementation has been protected against 1st order Power Analysis Side-Channel attacks.
2. `goal_powersca_2nd` - When present, this file[4] indicates the implementation has been protected against 2nd order Power Analysis Side-Channel attacks.
3. `goal_emsca` - When present, this file[4] indicates the implementation has been protected against Electromagentic Side-Channel attacks.
4. `architectures` - File with one target microcontroller/microprocessor architecture. Contents described in the **Architectures** section below.

[3] https://csrc.nist.gov/CSRC/media/Projects/Lightweight-Cryptography/documents/final-lwc-submission-requirements-august2018.pdf

[4] This file may be empty, or contain human readable text briefly describing the mitigation

```
<Candidate_Name>_<Group_Name>
├── Documents
│   ├── changelog.pdf          # optional
│   ├── coversheet.pdf
│   └── documentation.pdf
└── Implementations
    └── crypto_aead
        └── <Candidate_and_Variant_Name>
            ├── designers
            ├── LWC_AEAD_KAT_<CRYPTO_KEYBYTES*8>_<CRYPTO_NPUBBYTES*8>.txt
            ├── protected_<implementation_name>
            │   ├── api.h
            │   ├── aead.c
            │   ├── other.h
            │   ├── other.c
            │   ├── other.S              # assembly is acceptable
            │   ├── goal_powersca_1st     # 1st order protection against power analysis
            │   ├── goal_powersca_2nd     # 2nd order protection against power analysis
            │   ├── goal_emsca        # protection against electromagnetic analysis
            │   ├── architectures      # one target arch per line
            │   └── implementers      # one implementer per line
            ├── ...                    # additional implementations
            └── ref
                ├── api.h
                ├── encrypt.c
                ├── other.c
                └── implementers
```

Figure 4: Recommended format of deliverables for protected software implementation

5. `implementers` - File with one name per line indicating the authors of the protected implementation source code

For uniformity, a standard set of values for use within the `architectures` file are defined in Table 4.

Table 4: A standard set of values used within the `architectures` file

Value	ISA	Example Targets	Compiler
`arm`	ARM	STM32F	`arm-none-eabi-gcc -mcpu=cortex-m4 -mthumb`
`avr`	AVR	ATMega2560	avr-gcc
`riscv32`	RISC-V	FE310	`riscv64-unknown-elf-gcc -march=rv32imac`
`msp430`	MSP-430	MSP430FR5969	`msp430-elf-gcc`

Each submission should include a KAT produced by `genkat_aead.c` available from NIST [5] in the second-level folder (common to all implementations of a particular algorithm). Implementations which support hashing should also include a KAT produced by `genkat_hash.c`

Our understanding is that there is only one reference software implementation per candidate - the unprotected one, included in the given candidate's submission package.

Even assuming identical randomness values ("consumed" by the protected implementation during its operation), each protected implementation can generate different values of output shares for the ciphertext and tag during authenticated encryption. There is no golden standard for how the protected implementation should work other than the XOR of all ciphertext shares should give the ciphertext, and the XOR of all tag shares should give the tag, as calculated by the unprotected reference implementation.

5 Matching implementations with the labs

Our team will be in constant communication with the evaluation labs and the implementation submitters aiming at the best match between both groups. The general idea of a match will be based on the concept of bidding, combined with the detailed analysis of pros and cons of each assignment.

The evaluations by the majority of labs are likely to start from leakage assessments. In the case of selected labs, an attempt at various key recovery attacks is likely to be made as well.

Independently, each lab may freely choose from all implementations placed in the public domain. Additionally, protected implementations may be submitted to the labs directly by their developers in a format consistent with the respective calls. Finally, labs may also ask implementers for their deliverables. The developers may require a lab to keep the distribution of the source code limited to the lab personnel but may not prevent the publication of the obtained results.

Eventually, after all implementations are divided into classes with a comparable security level, the GMU team will benchmark and rank these implementations in terms of Throughput, Area/Memory Usage, Power, Energy per bit, Maximum amount of fresh randomness per clock cycle, and the number of random bits per each byte of AD and plaintext. Similar contributions by other benchmarking labs are very welcome.

[5] https://csrc.nist.gov/Projects/Lightweight-Cryptography

Our team will publish the record of evaluations in progress and reports from the completed evaluations on our website (subject to the explicit permissions by the respective implementers and evaluators).

6 Status and Proposed Timeline

At the time of the paper submission, the authors were aware of the following submissions:

- protected software implementations submitted by two groups, covering Ascon, GIFT-COFB, and Romulus (i.e., 3 out of 10 finalists)
- protected hardware implementations submitted by three groups, covering Ascon, Elephant, GIFT-COFB, ISAP, TinyJAMBU, and Xoodyak (i.e., 6 out of 10 finalists).

Other protected implementations are still under development. All protected implementation submitted before the end of the evaluation period (tentatively scheduled for June 30, 2022) will be recommended for evaluation by the Side-Channel Security Evaluation Labs.

The preliminary timeline for the remaining steps of the evaluation process is as follows:

- 30 Apr 2022: Preliminary reports of the Side-Channel Security Evaluation Labs and the Benchmarking Labs (covering only implementations submitted by April 15).
- Jun 30 2022: Final reports of the Side-Channel Security Evaluation Labs.
- Jul 31 2022: Final reports of the Benchmarking Labs with performance/cost rankings applied only to implementations with a comparable side-channel resistance.

However, if the implementers or the evaluation labs will need more time, we are open to possible extensions, subject to approval by NIST.

References

[1] NIST, "Submission Requirements and Evaluation Criteria for the Lightweight Cryptography Standardization Process," Tech. Rep., Aug. 2018.

[2] G. Goodwill, B. Jun, J. Jaffe, and P. Rohatgi, "A testing methodology for side–channel resistance validation," in *NIST Non-invasive Attack Testing Workshop*, Nara, Japan, 2011.

[3] G. Becker, J. Cooper, E. DeMulder, *et al.*, "Test Vector Leakage Assessment (TVLA) methodology in practice," in *International Cryptographic Module Conference*, 2013, p. 13.

[4] T. Schneider and A. Moradi, "Leakage Assessment Methodology: A Clear Roadmap for Side-Channel Evaluations," in *Cryptographic Hardware and Embedded Systems – CHES 2015*, T. Güneysu and H. Handschuh, Eds., ser. Lecture Notes in Computer Science, Berlin, Heidelberg: Springer, 2015, pp. 495–513.

[5] A. A. Ding, C. Chen, and T. Eisenbarth, "Simpler, Faster, and More Robust T-Test Based Leakage Detection," in *Constructive Side-Channel Analysis and Secure Design*, F.-X. Standaert and E. Oswald, Eds., ser. Lecture Notes in Computer Science, Cham: Springer International Publishing, 2016, pp. 163–183.

[6] A. Moradi, *How to Evaluate Side-Channel Leakages*, Workshop Talk, Bochum, Germany, Jun. 2017.

[7] F.-X. Standaert and F.-X. Standaert, “How (Not) to Use Welch’s T-Test in Side-Channel Security Evaluations,” in *Smart Card Research and Advanced Applications, CARDIS 2018*, J.-B. Fischer, Ed., vol. 11389, Cham: Springer International Publishing, 2019, pp. 65–79.

[8] A. Moradi, B. Richter, T. Schneider, and F.-X. Standaert, “Leakage Detection with the $X2$-Test,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 209–237, 2018.

[9] C. Whitnall, E. Oswald, C. Whitnall, and E. Oswald, “A Critical Analysis of ISO 17825 (‘Testing Methods for the Mitigation of Non-invasive Attack Classes Against Cryptographic Modules’),” in *Advances in Cryptology – ASIACRYPT 2019*, vol. 11923, Cham: Springer International Publishing, 2019, pp. 256–284.

[10] S. Bhasin, J.-L. Danger, S. Guilley, and Z. Najm, “NICV: Normalized inter-class variance for detection of side-channel leakage,” in *2014 International Symposium on Electromagnetic Compatibility*, Tokyo, Japan: IEEE, May 2014, p. 4.

[11] D. B. Roy, S. Bhasin, S. Guilley, A. Heuser, S. Patranabis, and D. Mukhopadhyay, “CC Meets FIPS: A Hybrid Test Methodology for First Order Side Channel Analysis,” *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 347–361, Mar. 2019.

[12] T. Moos, F. Wegener, and A. Moradi, “DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations,” *TCHES*, vol. 2021, no. 3, pp. 552–598, Jul. 2021.

[13] D. Knichel, P. Sasdrich, and A. Moradi, “SILVER – Statistical Independence and Leakage Verification,” in *Advances in Cryptology – ASIACRYPT 2020*, ser. LNCS, vol. 12491, Cham: Springer International Publishing, 2020, pp. 787–816.

[14] *SILVER - Statistical Independence and Leakage Verification*, https://github.com/Chair-for-Security-Engineering/SILVER, Jun. 2021.

[15] M. Randolph and W. Diehl, “Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman,” *Cryptography*, vol. 4, no. 2, p. 15, May 2020.