Guidelines for Self-Assessment of Cryptographic Algorithms Performance on FPGA

Institute of Commercial Cryptography Standards

June 2026

Contents

1 Notes on Algorithm Implementation

1.1 Purpose

These Guidelines are intended to standardize the self-assessment process for FPGA implementations of the next-generation public-key cryptographic algorithms performance, including key encapsulation mechanisms (KEMs), digital signature schemes, and key exchange protocols, as well as cryptographic hash algorithms. They define the self-assessment metrics and requirements for the correctness, comparability, and efficiency of algorithm implementations.

Based on the characteristics of the FPGA Implementation, these Guidelines provide algorithm submitters with a technical reference and procedural guidance for preparing the Reference Implementation Version and Optimization Version for the FPGA Implementation. They do not, by itself, impose any additional implementation requirements. Algorithm submitters may, in accordance with the algorithm submission requirements, submit corresponding reference implementation versions and optimized implementation versions (including performance-optimized and resource-optimized versions).

The self-assessment results serve as checkable technical evidence in the algorithm submission materials and are used solely to describe the implementation environment, testing process, and test results.

1.2 Scope

These Guidelines apply to the self-assessment of two categories of cryptographic algorithms performance on FPGA: public-key cryptographic algorithms and cryptographic hash algorithms, including the reference implementation and optimized implementation versions. The optimized implementation version can be further classified into the resource optimization and performance optimization versions. The versions are described as follows:

(1) Reference Implementation Version: This version focuses on the correctness of algorithmic logic, implementation simplicity, and ease of development. It does not require additional designs of performance or resource optimization, nor does it introduce specific optimization techniques, thereby facilitating understanding of the implementation principles of the algorithm.

(2) Performance Optimization Version: This version is primarily intended to maximize the performance of the algorithm implementation. Optimization techniques such as parallel processing, pipelining, and loop unrolling may be employed to improve operating frequency and data throughput, thereby enabling deployment in high-performance, high-throughput hardware application scenarios.

(3) Resource Optimization Version: This version is primarily intended to minimize resource consumption in the algorithm implementation., while also taking the throughput-to-area ratio into consideration. Optimization techniques such as reducing parallel processing paths, decreasing data-path width, and simplifying computational units may be employed to reduce hardware resource consumption, thereby enabling deployment in hardware application scenarios with limited resources.

1.3 Assessment Guidelines

The guidelines for self-assessment of cryptographic algorithms performance on FPGA are as follows:

(1) Unified Requirements: The self-assessment of public-key cryptographic algorithms and cryptographic hash algorithms should be conducted based on the implementation self-assessment metrics provided in Chapter 3.

(2) Categorized Submission: Submission materials should be submitted separately by algorithm category and implementation version, and the differences among different versions should be reflected. For public-key cryptographic algorithms, the three functions (namely key encapsulation mechanisms (KEMs), digital signature schemes, and key exchange protocols) should be separately assessed.

(3) Full Metric Coverage: The assessment results for every implementation version should cover metrics including functionality, performance, and resource consumption.

(4) Authenticity and Reproducibility: The execution results in the implementation self-assessment test report should be authentic and reproducible.

1.4 Material Preparation

(1) Algorithm submitters should organize the materials separately by algorithm category and implementation version, and submit them as a complete package for each version. Each version should include at least the following materials:

No.	Material Name	Supplementary Notes
1	Algorithm Hardware Implementation Source Code	Complete synthesizable RTL source code (Verilog/System Verilog).
2	Compilation/Synthesis Configuration Files	XDC Constraint Files, EDA tool Tcl scripts, etc.
3	Self-Assessment Code and Data	Vivado project including Tcl scripts (synthesis, implementation, timing analysis, power analysis), run log summaries, top-level RTL and necessary wrappers, performance assessment materials (test cases, timing signal definitions, performance scripts/simulation logs for cycles, ops/s, throughput, latency). Provided separately per version.
4	Algorithm Implementation Description Document	FPGA algorithm principles, design description, architecture, version-specific design, optimization methods, module boundaries, etc.
5	Self-Assessment Report	Complete results of functional correctness, synthesis, performance, area, and power tests.

(2) Material naming rule: The material files should be named in the format "Algorithm Category-Algorithm Name-FPGA-Implementation Version", such as "Key Encapsulation-SM2-FPGA-Resource Optimization Version".

2 Algorithm Implementation Guidelines

2.1 Hardware and Software Environment

To ensure the comparability of self-assessment results, it is recommended to adopt the following baseline environment configurations for FPGA implementations.

The hardware environment configurations are recommended as follows:

No.	Component	Configuration Description
1	FPGA Model	AMD Zynq-7000 series XC7Z100 SoC FPGA.
2	Constraint Files	XDC (or SDC-compatible) constraints covering clock/reset, I/O timing/load, timing exceptions, and design rules.
3	Operating Conditions	Typical industrial conditions. For multi-corner analysis, specify the selected corner in the report.
4	Hardware Resources	LUT、FF、BRAM、DSP、URAM、IO、etc.

The software environment configurations are recommended as follows:

No.	Component (EDA Tools)	Configuration Description
1	Synthesis and Implementation	Xilinx Vivado 2023.2
2	Synthesis Strategy	Vivado Synthesis Defaults
3	Implementation Strategy	Vivado Implementation Defaults (Perf_Explore or specified strategy)
4	Static Timing Analysis	Vivado Timing (Report Timing Summary / Report Timing)
5	Power Analysis	Vivado Power Estimation / Report Power (based on placed-and-routed netlist and activity files)

2.2 Algorithm Implementation

The implementation guidelines for the FPGA implementation are as follows:

(1) **Interface Specifications**: The top-level module interface should be clearly defined, including the module name and port list (control signals/data signals/clock signals). A RAM-like interface is recommended for data interfaces.

(2) **Synthesizability**: RTL source code should satisfy EDA tool synthesizability requirements and use standard Verilog/System Verilog syntax. Non-synthesizable statements (such as delay statements and dynamic arrays) should be avoided to ensure that logic synthesis can be completed directly and that metrics such as post-synthesis area and frequency can be accurately extracted.

(3) **Logic Specification**: The implementation logic should be fully consistent with the algorithm standard, without logic deviations. Common hardware design issues such as combinational logic loops, potential timing violations, and redundant logic should be avoided to ensure functional correctness.

3 Self-Assessment Metrics

3.1 Metric Definitions

The self-assessment content is divided into three major categories: basic verification items, record information items, and quantitative comparison metrics. Basic verification items include RTL source-code compliance, correctness of KAT verification, reproducibility of synthesis scripts, and completeness of submitted materials. Record information items cover random number generation methods, resource occupancy, logic units, built-in IP cores, optimization methods, and test environments. Quantitative comparison metrics include clock cycles, computational throughput, computational latency, area, power consumption, throughput-to-area ratio, etc.Basic verification items and record information items are mainly used to determine the authenticity and validity of assessment results and ensure that data can be re-verified and cross-checked. Quantitative comparison metrics support subsequent analysis of performance, resource consumption and other relevant aspects.

The implementation self-assessment metric definitions are as follows:

Level-1 Metric	Level-2 Metric	Unit	Core Description
Performance	Clock Cycles	cycles	Number of clock cycles required to complete one core algorithm operation (key generation, signature generation, signature verification, encapsulation, decapsulation, shared secret derivation, or hash computation).

Level-1 Metric	Level-2 Metric	Unit	Core Description
	Operating Frequency	MHz	Highest frequency F_max satisfying STA convergence, obtained by frequency sweeping under the same constraint template (recommended step: 1-2 MHz; final WNS close to 0 but non-negative).
	Computational | Throughput	ops/s	Public-key cryptographic algorithms: number of algorithm operations completed per unit time, including key generation, signature generation, signature verification, encapsulation, decapsulation, shared secret derivation, etc.
		Mb/s	Cryptographic hash algorithms: number of data bits processed per unit time, including hash-value generation.
	Throughput-to-Area Ratio	ops/s/LUT or ops/s/DSP	The ratio of throughput to key resource consumption (LUT or DSP), reflecting the efficiency of logic resource utilization.
	Computational Latency	ns	The actual time taken to complete a core algorithmic operation at the maximum frequency.
Resource	LUT (Look-Up Table)	count	The number of LUTs required to implement combinatorial logic (such as addition/subtraction, selection, logic operations, etc.).
	FF (Flip-Flop / Register)	count	The number of FFs required to implement sequential logic (storing data, signal synchronization, etc.).
	BRAM (Block RAM)	count	The number of large-capacity on-chip memory (replacing external RAM) used for cache, FIFOs, look-up tables, etc.).
	DSP (Digital Signal Processing unit)	count	The number of DSPs used for accelerating cryptographic algorithm implementations (e.g., multipliers, adders, and modular multipliers).
	Other	-	The number of resources such as I/Os and PLLs.
Power	Static Power Consumption	mW	Static/leakage power consumed under typical PVT conditions at zero workload.
	Dynamic Power Consumption	mW	Dynamic power consumed under typical PVT conditions at full workload.
	Total Power Consumption	mW	Sum of static power and dynamic power consumption.

3.2 Algorithm Implementation Versions

These guidelines provide algorithm submitters with implementation guidance for three versions on the FPGA architecture: the Reference Implementation Version, the Performance Optimization Version, and the Resource Optimization Version. In addition, the submission of other versions targeting specific application scenarios and having practical value is encouraged.

Algorithm submitters can follow the algorithm implementation guidelines specified in this guide to complete the FPGA implementation of cryptographic algorithms. Please refer to the appendix for the source code directory structure.

3.2.1 Reference Implementation Version

The reference implementation self-assessment focuses on logical correctness, implementation simplicity, and portability across technologies, without pursuing extreme performance and area optimization.

(1) Implementation Plan: No optimizations such as parallelism or pipelining; use the native logic architecture of the algorithm; no resource trimming; logical modules correspond one-to-one with the theoretical algorithm architecture.

(2) Controller: The controller handles all control logic of the algorithm, including start/stop/reset operations, key/IV/mode configuration, data handshake and flow control, state machine scheduling, status and error reporting, register configuration, and interaction with the upper-layer system, and specifies the controller used in the algorithm.

(3) Synthesis Constraints: Designers should provide F_max in the summary table according to the implementation version and explain the method used to search for F_max (such as stepping/bisection, etc.) to ensure reproducibility.

(4) Assessment Focus: Verify the functional correctness and consistency of the algorithm logic.

(5) Source Code Implementation: RTL source code should be concise and readable, with complete comments, clear module partitioning, and no redundant logic, to facilitate metric reproducibility by the evaluator.

3.2.2 Performance Optimization Version

The performance optimization self-assessment focuse on throughput maximization and operating frequency improvement as the core goals and is suitable for high-performance and high-throughput scenarios.

(1) Implementation Plan: High-performance optimization methods such as parallel computing, pipelining, loop unrolling, and multiple computation unit reuse can be adopted; hardware resources can be reasonably increased to improve algorithm processing speed, supporting high-frequency, high-throughput operation, with priority given to maximizing performance.

(2) Synthesis Constraints: Under the premise of maintaining consistent constraint templates, the highest frequency that satisfies STA convergence is obtained through frequency scanning; ensure timing convergence at high frequencies, giving priority to performance metrics.

(3) Assessment Focus: Focus on collecting computation throughput and operating frequency data, completing performance statistics.

(4) Source Code Implementation: RTL source code should be concise and readable, with complete comments, clear module partitioning, and no redundant logic, to facilitate metric reproducibility by the evaluator.

(5) Optimization Description: The report should clearly specify the concrete performance-optimization methods and the increase in hardware resources used, and should explain the differences between

performance statistics and those of the reference implementation to demonstrate the practical effect of performance optimization.

3.2.3 Resource Optimization Version

The resource optimization self-assessment focuses on resource usage optimization as the core goal and is suitable for resource optimization scenarios.

(1) Implementation Plan: Hardware resources may be pruned by removing parallel paths, reducing bit widths, simplifying arithmetic units, adopting iterative implementations, and other methods. High-performance optimization methods are not used, and throughput-to-area optimization is achieved on the basis of area optimization.

(2) Synthesis Constraints: Under the premise of ensuring timing convergence, priority should be given to configuring constraints related to throughput-to-area ratio optimization.

(3) Assessment Focus: Focus on the total area data after synthesis.

(4) Source Code Implementation: RTL source code should be concise and readable, with complete comments, clear module partitioning, and no redundant logic, to facilitate metric reproducibility by the evaluator.

(5) Optimization Description: Clearly specify the specific methods of area optimization and the scope of resource pruning in the report; also explain the differences in area statistics results compared to the reference implementation to reflect the actual effects of resource optimization.

3.3 Self-Assessment Testing

The FPGA implementation self-assessment includes the following issues:

(1) Functional Testing: It is recommended that the test process for PFGA implementation self-assessment be executed step by step according to a fixed process, mainly including the following tests.

(2) Logic Synthesis and Report Generation: Use synthesizable RTL and XDC constraint files to complete synthesis and place-and-route, generate a set of reports, summarizing key data such as frequency and resource usage.

(3) Power Assessment and Report Generation: Power assessment uses power analysis tools (based on post-place-and-route netlist and activity files), calculating static power, dynamic power, and total power. The statistical window, workload type, frequency point (F_max), and device operating conditions (voltage/temperature) need to be specified.

(4) Performance Metric Calculation: Calculate computational throughput based on the operating frequency obtained from logic synthesis and the number of clock cycles obtained from functional simulation.

The self-assessment test data should cover all algorithm instances, all declared security strength levels, and all specified programming interfaces.

Note: For scenarios requiring repeated execution and statistical analysis, at least 100 measurements are recommended.

3.4 Self-Assessment Test Data

All test vectors for functional testing should be recorded in the report. Performance and resource consumption tests should use reproducible inputs, and the source or generation method of the input data should be specified in the test report.

For tests requiring random numbers, it is recommended to use a fixed test set or to record the random seed and generation rules so that the results are reproducible. For random number generation rules, the method specified in the programming interface files provided in the algorithm submission requirements is recommended.

3.4.1 Test Data for Public-Key Cryptographic Algorithms

For the key encapsulation function, functional test data for key generation, encapsulation, and decapsulation should be prepared, and the sizes of the public key, private key, ciphertext, and encapsulated key should be recorded.

For the digital signature function, functional test data for key generation, signature generation, and signature verification should be prepared, and the sizes of the public key, private key, and signature should be recorded.

For the key exchange function, functional test data for initialization, message generation, and shared secret key derivation should be prepared, and the message length, number of interaction rounds, and sizes of the long-term public key, if any, long-term private key, if any, and shared secret key should be recorded.

For public-key cryptographic algorithm test data, performance test results should be provided separately for the three functions of key encapsulation, digital signature, and key exchange under different security strength levels.

Function	Metrics	Remarks
Key Encapsulation	Key generation performance; encapsulation performance; decapsulation performance	Only the performance metrics of this function are tested; the effect of differences in encapsulated key length is not considered.
Digital Signature	Key generation performance; signature generation performance; signature verification performance	A 64-byte message should be used as the message input to be signed for testing, so as to avoid the impact of differences in hash computation overhead on the comparability of test results.
Key Exchange	Key exchange performance	Network transmission time is not counted.

3.4.2 Test Data for Cryptographic Hash Algorithms

For cryptographic hash algorithms, functional test data for hash computation should be prepared. If the algorithm supports extendable output or variable-length output, functional testing should cover different output lengths.

For cryptographic hash algorithms, eight levels of data, S1 to S8, should be selected as inputs for performance testing. The data range from the byte level to the kilobyte level. The specific levels are as follows:

Level	Recommended Data Length
S1	32 Bytes
S2	128 Bytes
S3	512 Bytes
S4	1024 Bytes
S5	4096 Bytes
S6	8192 Bytes

Level	Recommended Data Length
S7	16384 Bytes
S8	65536 Bytes

3.5 Self-Assessment Test Report

After completing the self-assessment testing, the algorithm designers should prepare a self-assessment test report containing the following information:

(1) Basic Algorithm Information: algorithm category, algorithm name, subdivided function, implementation version, parameter set/security level.

(2) Assessment Environment Information: FPGA model and speed grade, EDA tool version, XDC constraint template version.

(3) Functional Testing: test results of each test case.

(4) Performance Testing: basic performance data, such as operating frequency, number of clock cycles (cycles), computation latency (ns), computation throughput (ops/s or Mb/s), etc.

(5) Area Testing: key data in the summary table such as frequency and resource usage must allow locating hardware resource usage in the Vivado report, etc.

(6) Power Testing: basic power data, such as static power, dynamic power, total power. For hierarchical power, it is recommended to provide the proportion of key modules, method of marking activity, and coverage explanation (if applicable).

(7) Raw Evidence Index: a list of the test commands, input data, original logs, result files, and calculation scripts corresponding to each summary metric.

The test results should be clear, explicit, unambiguous, and checkable.

4 Miscellaneous

Algorithm submitters are responsible for resolving the legal sources and usage authorization of EDA tools on their own and are responsible for the compliance of the submitted materials. The requesting party does not provide any resources related to EDA tools.

Appendix: Source Code Directory Structure

Algorithm submitters should complete the FPGA implementation of the cryptographic algorithm, and prepare complete RTL source code (Verilog/System Verilog), including operation-core modules, necessary input/output interface logic (wrapper), and control logic (start/reset/done). The source code should satisfy synthesizable requirements and contain no non-synthesizable statements.

Algorithm source files should be placed according to a unified directory structure so that EDA tool scripts can automatically identify and compile them. Source code for different implementation versions should be independent and stored separately. The directory structure is as follows:

(1) PKC public-key cryptographic algorithms:

├── Public_Key_Cipher # Public-Key Cryptographic Algorithm FPGA Implementation Directory (Designer Implementation Replacement)
│ ├── KeyGen/ # Key Generation
│ │ ├── Basic # Basic Edition
│ │ │ ├── README.md # The shortest steps to reproduce from scratch (script entry, dependencies, expected output list)
│ │ │ ├── Summary/ # Summary Spreadsheet File Package
│ │ │ │ ├── Summary.xlsx # Summary table file （Excel/Word）
│ │ │ │ ├── KeyGen_Declaration.pdf # Declaration of the signature algorithm
│ │ │ ├── RTL/ # Resource-optimized version implements RTL source code
│ │ │ │ ├── top/ # Top-level RTL source file
│ │ │ │ ├── wrapper/ # Necessary input and output encapsulation logic
│ │ │ ├── Constraints/ # XDC Constraint File
│ │ │ │ ├── top.xdc # XDC Constraint File
│ │ │ ├── Scripts/ # TCL script file
│ │ │ │ ├── run_synth.tcl # Synthesis script
│ │ │ │ ├── run_impl.tcl # Implementation Script
│ │ │ │ ├── run_timing.tcl # Timing Analysis/Convergence Script
│ │ │ │ ├── run_power.tcl # Power Analysis Script
│ │ │ ├── Reports/ # Report File Set
│ │ │ │ ├── synth/ # Synthesis report
│ │ │ │ ├── impl/ # Implementation report
│ │ │ │ ├── timing/ # Timing report
│ │ │ │ ├── power/ # power report
│ │ │ ├── Activity/ # Dynamic activity data file required for power analysis
│ │ │ │ ├── saif_or_vcd/ # Activity statistics format / waveform dump format (if the file is too large, provide a downloadable link and ensure its validity)
│ │ ├── LowSize # Resource optimization version; directory structure is the same as the basic version
│ │ ├── HighPerformance #Performance optimization version; directory structure is the same as the basic version
│ ├── SignVerify/ # Digital Signature: Directory Structure Same as Key Generation

│ ├── KEM/ # Key encapsulation: directory structure is the same as key generation
│ ├── KEA/ # Key Exchange: Directory Structure Same as Key Generation
├── Summary_Comparison /#Cross-version summary comparison table; a cross-version metrics comparison table storing all algorithms/functions

（2）CryptHash cryptographic hash algorithm：

├── Hash # Hashing Algorithm FPGA Implementation Directory (Designer Implementation Replacement)
│ ├── Basic # Basic Edition
│ │ ├── README.md # The shortest steps to reproduce from scratch (script entry, dependencies, expected output list)
│ │ ├── Summary/ # Summary Spreadsheet File Package
│ │ │ ├── Summary.xlsx # Summary table file （Excel/Word）
│ │ │ ├── Hash_Declaration.pdf # Declaration of the hash algorithm
│ │ ├── RTL/ # Resource-optimized version implements RTL source code
│ │ │ ├── top/ # Top-level RTL source file
│ │ │ ├── wrapper/ # Necessary input and output encapsulation logic
│ │ ├── Constraints/ # XDC Constraint File
│ │ │ ├── top.xdc # XDC Constraint File
│ │ ├── Scripts/ # TCL script file
│ │ │ ├── run_synth.tcl # Synthesis script
│ │ │ ├── run_impl.tcl # Implementation Script
│ │ │ ├── run_timing.tcl # Timing Analysis/Convergence Script
│ │ │ ├── run_power.tcl # Power Analysis Script
│ │ ├── Reports/ # Report File Set
│ │ │ ├── synth/ # Synthesis report
│ │ │ ├── impl/ # Implementation report
│ │ │ ├── timing/ # Timing report
│ │ │ ├── power/ # power report
│ │ ├── Activity/ # Dynamic activity data file required for power analysis
│ │ │ ├── saif_or_vcd/ # Activity statistics format / waveform dump format (if the file is too large, provide a downloadable link and ensure its validity)
│ ├── LowSize # Resource optimization version; directory structure is the same as the basic version
│ ├── HighPerformance # Performance optimization version; directory structure is the same as the basic version
├── Summary_Comparison / # Cross-version summary comparison table; a cross-version metrics comparison table storing all algorithms/functions