

Hardware API for Post-Quantum Public Key Cryptosystems

Ahmed Ferozपुरi, Farnoud Farahmand, Malik Umar Sharif, Jens-Peter Kaps,
and Kris Gaj

Cryptographic Engineering Research Group
George Mason University
Fairfax, Virginia 22030
email: {aferozpu, ffarahma, msharif2, jkaps, kgaj}@gmu.edu

Abstract. In this paper, we specify the proposed hardware Application Programming Interface (API) for Post-Quantum Public Key Cryptosystems. This new hardware API intends to meet the diverse requirements of Post-Quantum Cryptosystems, and includes; minimum compliance criteria, interface, communication protocol, and the timing characteristics supported by the core. All of them have been defined with the goals of guaranteeing (a) compatibility among implementations of the same algorithm by different designers, and (b) fair benchmarking of Post-Quantum Public Key Cryptosystems in hardware. In the rest of this document, we refer to the core compatible with our API as a Post-Quantum Cryptography (PQC) core.

1 Minimum Compliance Criteria

The recommended minimum compliance criteria are listed below:

1.1 Encryption, Decryption, Signature Generation and Signature Verification

A PQC core can implement either (1) encryption and decryption, or (2) signature generation and signature verification. For each of the above two cases, only one operation, e.g., either encryption or decryption, should be performed at any given time (half-duplex).

This feature demonstrates an algorithm's ability to use shared resources for: (1) encryption and decryption, or (2) signature generation and signature verification.

Alternatives (not recommended):

- a) Separate cores for encryption and decryption or for signature generation and signature verification (simplex).
- b) Encryption and decryption or signature generation and signature verification within one core, with both operations capable of running in parallel (full-duplex).

1.2 Variants

A PQC core can either implement just one variant of a given algorithm (e.g., a variant supporting just one key size) or multiple variants of the same algorithm.

In case multiple variants are implemented within the same core, the choice among these variants can be made either

- a) at the time of synthesis, using constants or generics, or
- b) at the run-time, using system parameters provided as inputs.

1.3 Key Generation

Key generation should be fully implemented outside of the hardware core, e.g., in software.

Alternatives (not recommended):

- a) Key generation should be fully implemented within the hardware core.
- b) Key generation may be done either in hardware or in software.

1.4 Incomplete blocks

The core should properly handle incomplete blocks of message.

An alternative (not recommended):

- a) handling only messages composed of full blocks.

1.5 Padding

Padding in hardware, assuming that an unused portion of the last input data word is filled with zeros.

The padding type must be specified using either:

- a) Generics, or
- b) Inputs (system parameters)

Padding cost, in terms of area, is algorithm dependent, and not negligible.

Alternatives (not recommended):

- a) Padding in hardware, assuming that an unused portion of the last block is filled with zeros.
- b) Padding in software, followed, if needed, by modifications of the last blocks in hardware.

1.6 Empty message

Empty messages are allowed only as an input to signature generation and signature verification.

Alternatives (not recommended):

- a) Allowing an empty message for encryption.
- b) Not allowing empty messages for signature generation and signature verification.

1.7 Supported maximum size of a message

A PQC core should support at least one full block and no more than $2^{16}-1$ bytes of a message. A limit on the number of blocks and bytes of a message that can be processed by a given core should be clearly provided in the documentation of the core, taking into account that the size of one block of message in bits depends on a particular algorithm and an algorithm variant.

1.8 Fractions of bytes

The size of all inputs is assumed to be expressed in bytes. As a result, the core should support only inputs composed of full bytes. No fractions of bytes should be allowed.

An alternative (not recommended):

- a) the size of inputs expressed in bits.

Allowing inputs of arbitrary size in bits would substantially increase the area required for handling of incomplete blocks.

1.9 Maximum number of independent inputs processed in parallel

A PQC core should process only one input at a time. The core may still take advantage of parallel processing of blocks belonging to the same input.

An alternative (not recommended):

- a) an implementation that supports processing of multiple independent inputs in parallel.

1.10 External memory

External memory may be used only to store intermediate values, and not for look-up tables or calculations.

1.11 One clock domain

A PQC core should have only one clock input. This clock should be operating at the maximum clock frequency determined by the critical path located entirely inside of the hardware module.

An alternative (not recommended):

- a) multiple clock domains, e.g., for input module, output module, and the main cipher core.

1.12 Permitted widths of data ports (in bits)

Public Data Input (PDI) and Data Output (DO) ports:

Lightweight implementations: $w = 8, 16, 32$

High-speed implementations: $32 \leq w \leq 128$.

Secret Data Input (SDI) ports:

Lightweight implementations: $w = 8, 16, 32$

High-speed implementations: $32 \leq sw \leq 64$.

See Section 2 and Fig. 1 for the exact meaning of PDI, SDI, DO, w and sw .

Implementations of a particular Post-Quantum Public Key Cipher, with the same w and sw , following all other minimum compliance criteria, should be mutually compatible. Implementations with different values of w or sw should be compatible under the assumption that the decryption input is reformatted in software or hardware (from one input word width to another) using a universal function/circuit, common for all ciphers.

2 Interface

The general idea of a PQC core interface is shown in Fig. 1. This interface is composed of five major data buses for:

- Public Data Inputs (PDI)
- Secret Data Inputs (SDI)
- Random Data Inputs (RDI)
- Data Outputs (DO), and
- External Memory Inputs/Outputs (MEM), respectively,

The first four of these five buses are accompanied by the corresponding handshaking control signals, named `valid` and `ready`. The `valid` signal indicates that

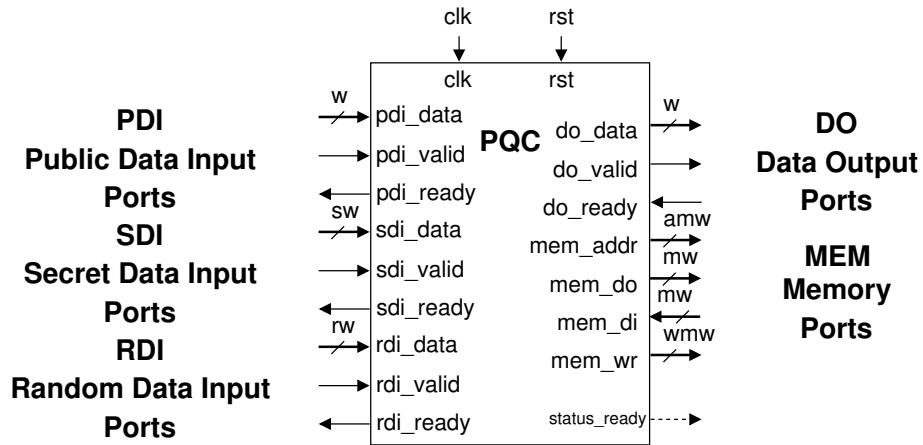


Fig. 1: PQC Interface

the data is ready at the source, and the ready signal indicates that the destination is ready to receive them.

The External Memory Inputs/Outputs have a different set of accompanying ports. The memory control signals support multiple memory configurations. The mem_addr signal is used to specify a memory address. The mem_do and mem_di signals are used to send and receive data from memory, respectively. Multiple memory blocks can be written to by using multiple mem_wr signals.

The status_ready signal is high when a status word is available at the (DO) port upon completion of a the corresponding instruction, and can be used for synchronization purposes.

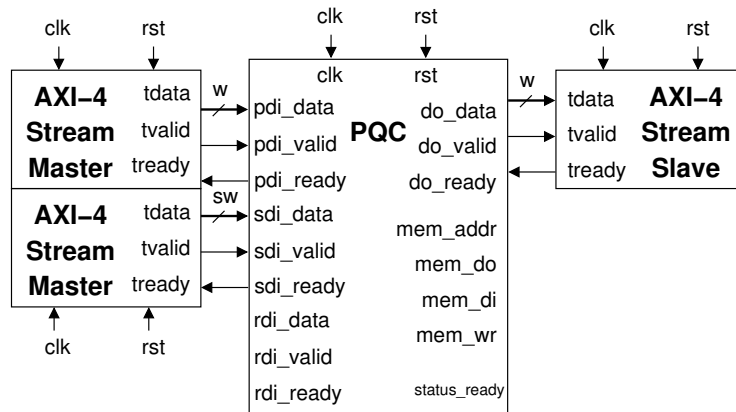


Fig. 2: Typical external circuits: AXI4-Stream IPs

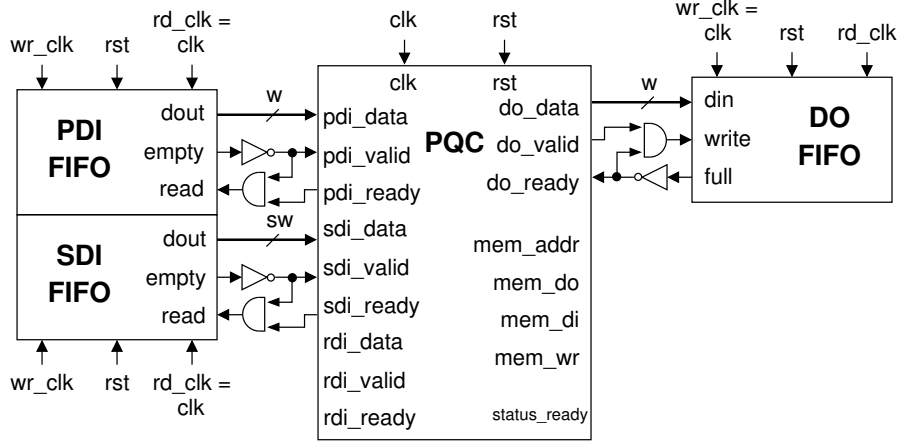


Fig. 3: Typical external circuits: FIFOs

The physical separation of Public Data Inputs (such as the public key, message, ciphertext, etc.) from Secret Data Inputs (such as the private key) is dictated by the resistance against any potential attacks aimed at accepting public data, manipulated by an adversary, as a new private key.

The handshaking signals are a subset of major signals used in the AXI4-Stream interface [1]. As a result, a PQC core can communicate directly with the AXI4-Stream Master through the Public Data Input and Secret Data Input, and with the AXI4-Stream Slave through the Data Output, as shown in Fig. 2. At the same time, PQC is also capable of communicating with much simpler external circuits, such as FIFOs, as shown in Fig. 3.

An additional advantage of using FIFOs at all data ports is their potential role as suitable boundaries between the two clock domains, used for communication and computations, respectively. This role is facilitated by the use of separate read and write clocks, shown in Fig. 3 as `rd_clk` and `wr_clk`, accordingly. For a better compatibility with the AXI communication interface, all FIFOs mentioned in our description are assumed to operate in the First-Word Fall-Through mode (as opposed to the standard mode).

The reset input can be either synchronous or asynchronous, and either active-high or active-low, depending on the conventions used in a given technology (e.g., FPGA vs. ASIC), as well as the personal preference of the designers.

3 Communication Protocol

All parts of a typical input and a typical output of a PQC core are shown in Fig. 4, for encryption (Encrypt), decryption (Decrypt), signature generation (Sign), and signature verification (Verify), respectively. System parameters can be omitted for all cases, assuming that either only one set of these parameters is

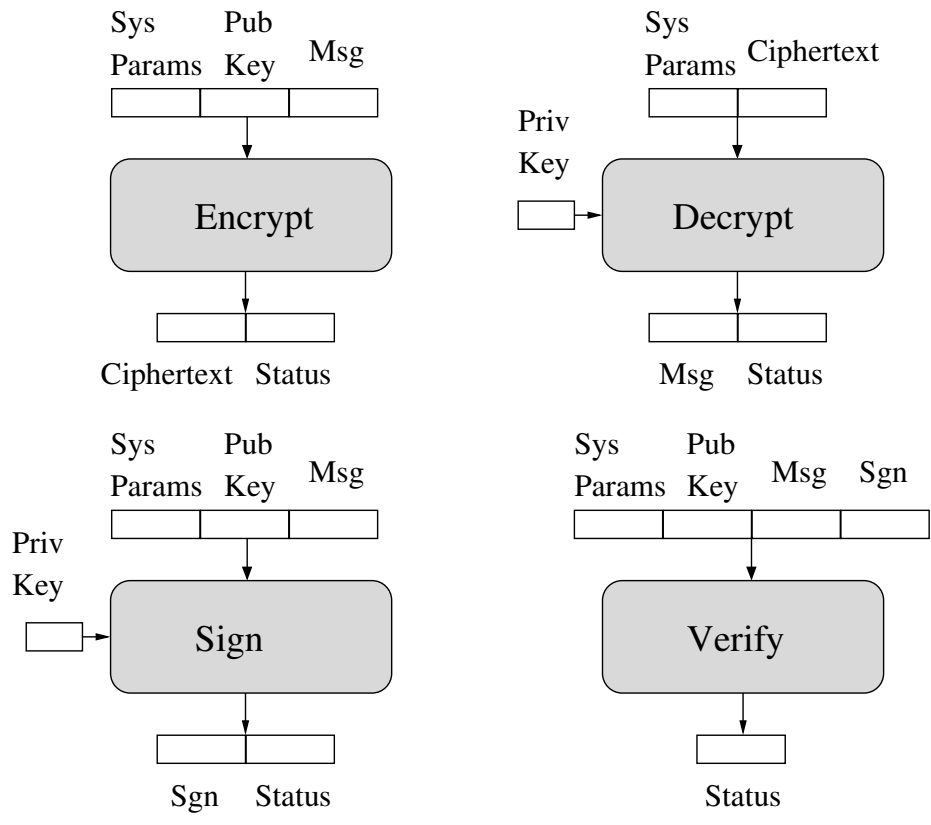


Fig. 4: Input and Output of a PQC core. Notation: Sys Params - System Parameters, Pub Key - Public Key, Msg - Message, Priv Key - Private Key, Sgn - Signature

supported, or the recently read values are to be used. Public key may be omitted as an input to signature generation if not required by a given algorithm. This key may be also omitted as an input to signature verification, assuming that it does not change compared to the last executed operation. Finally, an empty message can be used for signature generation and verification, but this message cannot be simply omitted; it must be provided in the form of a message segment with the Segment Length equal to zero.

Any instruction issued to a PQC core is followed by segments. Each segment starts with a separate header, describing its type and size.

The proposed format of the Secret Data Input is shown in Fig. 5. The entire input starts with an instruction, which in case of SDI is limited to Load Private Key (LDPRIVKEY). This instruction is then followed by the Private Key segment, containing a string of bits carrying a private key.

instr = LDPRIVKEY
seg_0_header
seg_0 = Priv Key

Fig. 5: Format of Secret Data Input for loading the private key

instr = ENC
seg_0_header
seg_0 = Sys Params
seg_1_header
seg_1 = Pub Key
seg_2_header
seg_2 = Msg

Fig. 6: Format of Public Data Input.

The proposed format of the Public Data Input is shown in Fig. 6. The allowed instruction types are: Activate Private Key (ACTPRIVKEY), Encryption (ENC), Decryption (DEC), Signature Generation (SGN), and Signature Verification (VER). The Activate Key instruction (ACTPRIVKEY) typically directly precedes Encryption (ENC), Decryption (DEC), Signature Generation (SGN) or Signature Verification (VER) instruction. The rest of the PDI input is divided into segments.

Figures 9 and 10 illustrate the allowed segment types for post-quantum encryption, decryption, signature verification and signature generation. Any segment type can be omitted, if it is not required by a given cipher. *However, empty messages must be provided using a separate segment, with the Segment Length field of the respective header set to zero.*

An optional Random Data Input can be used to feed the core with random inputs, required by some Post-Quantum Public Key Cryptosystems. The entire input is read when a data block is present, as indicated by the control signals of RDI, and the read operation is requested by the PQC core. There are no segment types necessary to meet the minimum compliance criteria, and every data block of width rw denotes a string of bits carrying a random value used by the PQC. In this fashion, both a fixed number of random inputs and an indefinite stream of random inputs can be accommodated.

The proposed Memory Port (MEM) offers a large degree of flexibility in order to support many different memory configurations. A single or multiple memory modules can be configured as required by splitting and connecting the `mem_addr`,

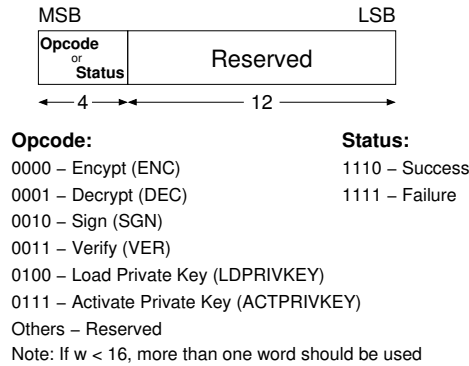


Fig. 7: Instruction/Status Format

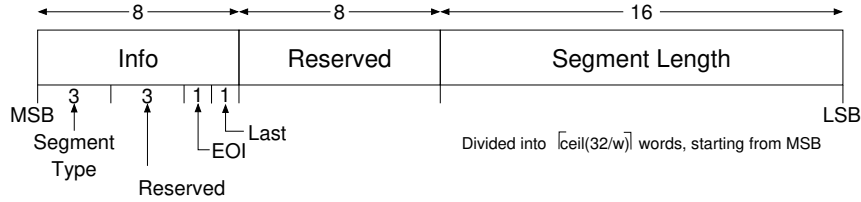


Fig. 8: Segment Header Format

mem_do, mem_di, and mem_wr signals, accordingly. Additionally, the memory can have either synchronous or asynchronous output.

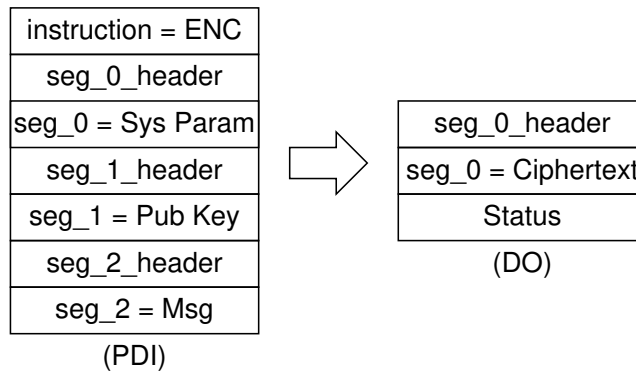
The instruction/status format of the PQC HW API is shown in Fig. 7. For instruction, the Opcode field determines which operation should be executed next. Alternatively, for status, the Opcode field is replaced by the Status field, which can be set to only two values, PASS or FAIL.

Table 1: Segment Type Encoding

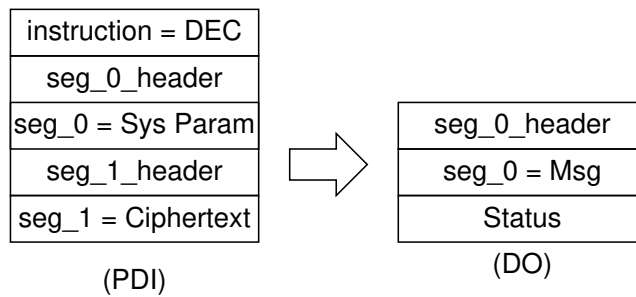
Encoding	Type
000	<i>Reserved</i>
001	Message
010	Signature
011	Ciphertext
100	System Parameters
101	Public Key
110	Private Key
111	<i>Reserved</i>

The segment header format is shown in Fig. 8. The segment header consists of:

- 3-bit *Segment Type*, which indicates the type of data that the current segment contains. The type encoding is defined in Table 1.
- 1-bit *EOI* (End-Of-Input) indicates that the current segment is the last segment of input other than an empty segment.
- 1-bit *Last* indicates that the current segment is the last segment, i.e., no more segments are associated with the given instruction.
- 11 reserved bits for future extensions.
- 16-bit *Segment Length* to specify the size of data in the given segment in *bytes*.

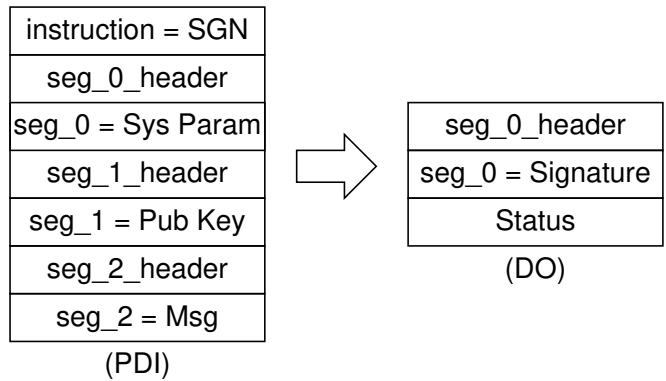


Encryption

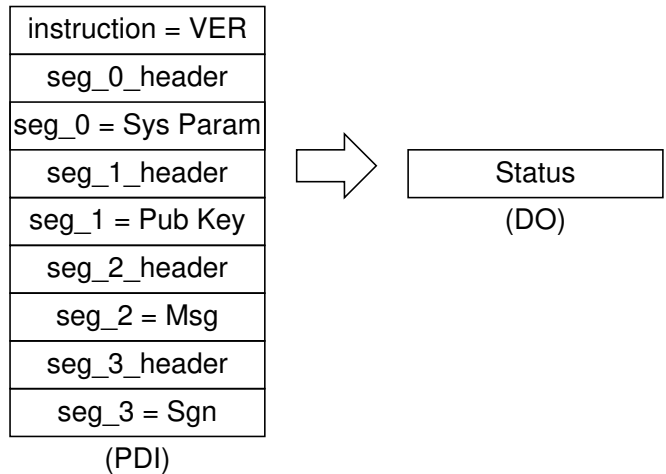


Decryption

Fig. 9: Format of Public Data Input (PDI) and Data Output (DO) for encryption and decryption.



Signature Generation



Signature Verification

Fig. 10: Format of Public Data Input (PDI) and Data Output (DO) for signature generation and signature verification.

Figures 9 and 10 present typical formats of the input and output for encryption, decryption, signature generation, and signature verification, respectively. At the input (PDI ports), a message starts with the key activation instruction (ACTPRIVKEY) for signature generation and decryption, followed by the operational instruction (ENC, DEC, SGN, or VER). Header and data segments for different data types subsequently follow.

The system parameter segment, Sys Params, should be sent first to setup the PQC core. The format of this segment is specific to a given algorithm and is

beyond the scope of this specification. In the future, with the emergence of the first implementations of post-quantum algorithms compliant with the proposed API, the concrete proposals in this area are likely to be published as independent documents.

4 Timing Characteristics

Figures 11 and 12 specify the timing characteristics of the ports PDI and DO, respectively. Input ports are shown in **blue** and the output ports in **red**. The contents of data buses are read and acknowledged when `*_valid` and its corresponding `*_ready` are both asserted. Data is assumed to be present at the output of the source module when `*_valid` is asserted.

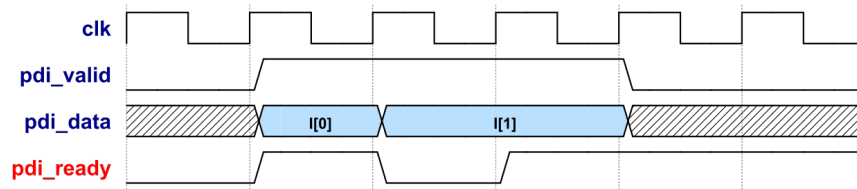


Fig. 11: Example timing diagram for PDI

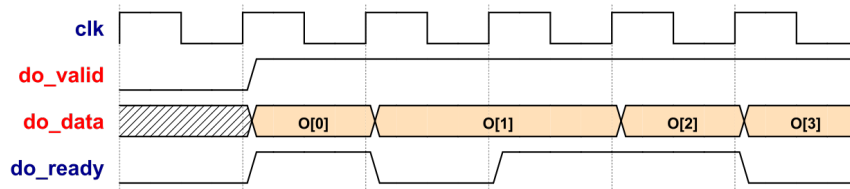


Fig. 12: Example timing diagram for DO

Additionally, the assumed order of bytes in memory and in 32-bit words is shown in Fig. 13.

5 Conclusions

We have defined the full specification of the hardware API for Post-Quantum Public Key Cryptosystems, suitable for hardware benchmarking and evaluation of candidates for new Post-Quantum Cryptography standards.

Our proposal meets one of the fundamental properties of every properly defined API:

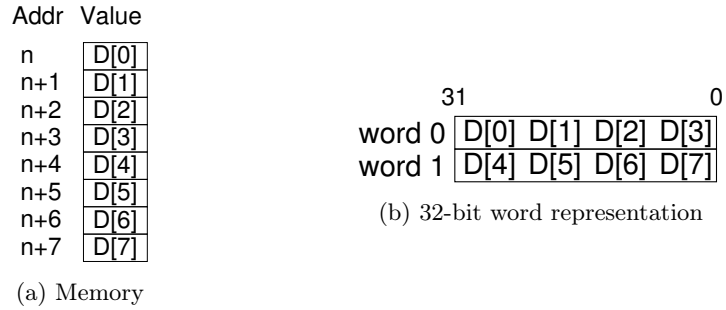


Fig. 13: Data representation

If a given algorithm is implemented independently by two different groups using the same API, one should be able to

- encrypt or sign a message using the first implementation, and
- decrypt or verify the result using the second implementation.

To be exact, this feature should hold under the following two assumptions:

1. Either both implementations use the same format of system parameters or this segment is converted from one representation to another.
2. Either both implementations use the same values of the data port widths w and sw , or simple reformatting (word width conversion) of the input to decryption (verification) is performed outside of the cipher core (in software or hardware).

A similar API, described in [2], has been widely adopted and successfully used to implement and benchmark almost all authenticated ciphers competing in Round 2 of the CAESAR contest [3].

References

1. ARM. AMBA Specifications. [Online]. Available: <http://www.arm.com/products/system-ip/amba-specifications.php>
2. E. Homsirikamol, W. Diehl, A. Ferozpuri, F. Farahmand, P. Yalla, J.-P. Kaps, and K. Gaj, “CAESAR Hardware API,” Cryptology ePrint Archive, Report 2016/626, 2016, <http://eprint.iacr.org/2016/626>.
3. Cryptographic Engineering Research Group (CERG) at GMU. (2016, Oct.) Benchmarking of Round 2 CAESAR Candidates. [Online]. Available: <https://cryptography.gmu.edu/athena/index.php?id=CAESAR>