A Bit Naming Convention for Cryptographic Algorithms

Markus G. Kuhn*

University of Cambridge, Computer Laboratory, New Museums Site,
Pembroke Street, Cambridge CB2 3QG, United Kingdom
mgk25@cl.cam.ac.uk

Abstract. This is a proposal for a simple notational convention, created in the hope that it will make descriptions of block ciphers, message digest functions, and similar cryptographic algorithms less ambiguous, easier to understand, and easier to implement in a portable way, especially in the context of the eternal Big Endian versus Little Endian wars.

1 Introduction

One obstacle in understanding and correctly implementing the block-cipher specifications proposed in NIST's Advanced Encryption Standard (AES) Contest [1] turned out to be the lack of an unambiguous convention for naming bits [2]. Notational conventions used for earlier standards like DES, which were targeted at hardware implementations, are not appropriate for the description of algorithms designed for fast software implementations.

The variables of cryptographic algorithms are commonly binary words consisting of 64, 128, 256, or more bits. As examples for such variables, we can consider a block-cipher that defines a bijection between a plaintext P and a ciphertext C using a key K. The key K is often expanded into a key schedule S or into a set of r round-keys $R[1], \ldots, R[r]$ and in every round, intermediate values $X[1], \ldots, X[r]$ are generated. All the parameters represented by a capital letter here will typically not fit into a single machine word of common processors, therefore they cannot simply be treated as abstract bit arrays or integer quantities, because their machine representation directly affects the design and efficient implementation of the algorithm.

In the following, we will refer only to a single example parameter X, and the notation described with X shall apply equally to all types of variables, no matter whether they represent plaintext, round keys, etc.

We can refer to a cryptographic variable X in several ways:

– **Array of bits.** Assigning a number to every bit in a variable might be the most intuitive view, if we think in terms of a circuit diagram for a hardware implementation, where processor word sizes and communication conventions do not matter. A bit-array notation was used in the original DES specification, since it was designed for hardware implementations.

* Supported by a European Commission Marie Curie training grant

– **Array of bytes.** An array of 8-bit bytes (or "octets" as Francophone and communications people prefer to say) is the most portable representation. Practically all commonly used storage media and communication channels process data as sequences of 8-bit words. Transferring a sequence of 8-bit bytes between platforms and environments is not a common problem, because the conventions to transport and manipulate ASCII text correctly are usually already in place, and ASCII text is just a sequence of 8-bit bytes.
– **Array of words.** This is the most convenient way of representing an algorithm that has been designed for the efficient software implementation on a processor with a given word size, for example 32-bit words in the case of the AES reference platform (Intel Pentium).

Unfortunately, there exists no well established convention to convert between the bit, byte, and word view of data. This is not a problem specific to cryptographic algorithms, but has been a cause of pain for the designers of file formats, network protocols, and multi-processor buses since the early days of interoperating computers [3–9]. It is unclear how to number bits within a byte, bits within a word, and bytes within a word. Equivalently, there exists no single convention for which bit or byte in a word is transmitted or stored first on a medium.

The numbering of bits and bytes within words is mostly a documentation nuisance. The conventions for the transmission of 8-bit byte sequences are usually well established for each type of interface. The real problem is the assignment of parts of words to bytes. Two incompatible families of processors support different memory conventions, which Cohen [3] nicknamed *Little Endian* if the least significant byte is transmitted first or stored on the lower address and *Big Endian* if the the most significant byte of a word comes first.

Examples for Little Endian processors are the Intel x86, Transputer, VAX, and MIPS, while examples for Big Endian processors include the Motorola 680x0, SPARC, Z8000, and RS/6000 processors, as well as the IBM mainframe architectures [8]. Newer processors, like the PowerPC family, can switch between both modes [7, 9], while some other architectures use bizarre and inconsistent mixtures of conventions. Formats specified in one convention force the implementors of the other camp to waste a few clock cycles on swapping bytes into the right order, even though many modern processors provide special instructions to do this extremely efficiently (e.g., `BSWAP` on the Intel486). Most internationally standardized file formats and network protocols follow the Big Endian camp (ASN.1/BER, TCP/IP, XDR, JPEG, JBIG, etc.), which has the advantage of corresponding to the usual positional notation for numbers.

2 Proposed Conventions

At any time, we view a parameter X as an array of m words, each of which is n bits long. We refer to the m words in the form $X_0, \ldots, X_{m-1}$ and we refer to the bits inside a word X_i in the form $X_i^{n-1}, \ldots, X_i^0$. The number of words m and the word size n are best introduced for each parameter X in the descriptive

text and kept unchanged throughout the description. If several different views of a parameter are really required in a description, then we can also refer to bit j in word i as $X_{i|m}^{j|n}$ for values $0 \le i < m$ and $0 \le j \le n$. Either $|m$ or $|n$ or both can be dropped from the notation if they are obvious from the context. Only one of $|m$ and $|n$ has to be present if multiple views of X are used in a text as long as the total number $k = mn$ of bits in X is clear from the context. We can also speak of $X_{|m}$ if we treat variable X as an array of m words, and we speak of $X^{|n}$ if we want to see X as an array of n-bit words.

If a word X_i is interpreted as an unsigned integer value modulo 2^n, so that arithmetic operations can be applied to it, then this shall always be in the sense of

$$X_i = \sum_{0 \le j < n} X_i^j \cdot 2^j.$$

In other words, bit 0 is always the least significant bit (LSB). Note that although this notation is mathematically intuitive and commonly used in computer architectures, it differs from both the TCP/IP convention [10] of numbering the most significant bit (MSB) with 0 and also from the ISO/ITU convention of numbering the LSB with 1.

Algorithms intended primarily for hardware implementation should use the $X^{|1}$ 1-bit array notation, that is the entire value is a sequence $X_0, \ldots, X_{k-1}$ of individual bits (word size $n = 1$). Algorithms designed for efficient software implementation on n-bit processors should be described in the $X^{|n}$ notation, that is as a sequence n-bit words. Transmission formats that might be processed on different types of hardware should be described in the $X^{|8}$ format, that is as a sequence of 8-bit bytes (word size $n = 8$). Test examples should be given in the same form in which the algorithm or format is specified, as sequences of n-bit words in hexadecimal notation.

In the textual or graphical presentation of the bits in a variable, the LSB in a word should appear rightmost to accommodate the common way of writing values in positional number systems and also to preserve the notion of arithmetic operations such as "rotate left". Arrays of words should be written starting with index 0 on the left to accommodate the writing direction of the Latin script and also to remove an unfair bias towards either byte ordering camp.

It should not be within the scope of the standard that specifies a cryptographic algorithm to define a single transformation between the portable $X^{|8}$ view and the implementation-oriented $X^{|n}$ view of its parameters. A standard should however name and define possible conventions, and the two commonly used ones will certainly be the following (assuming that the implementation word size n is a multiple of 8):

– **Big Endian:**

$$X_{\lfloor 8i/n \rfloor}^{n-8-(8i \bmod n)+j|n} = X_i^{j|8}$$

For example:

$$X_0^{31|32}, \ldots, X_0^{24|32} = X_0^{7|8}, \ldots, X_0^{0|8}$$

$$X_0^{23|32}, \ldots, X_0^{16|32} = X_1^{7|8}, \ldots, X_1^{0|8}$$
$$X_0^{15|32}, \ldots, X_0^{08|32} = X_2^{7|8}, \ldots, X_2^{0|8}$$
$$X_0^{07|32}, \ldots, X_0^{00|32} = X_3^{7|8}, \ldots, X_3^{0|8}$$
$$X_1^{31|32}, \ldots, X_1^{24|32} = X_4^{7|8}, \ldots, X_4^{0|8}$$

– **Little Endian:**

$$X_{\lfloor 8i/n \rfloor}^{(8i \bmod n)+j|n} = X_i^{j|8}$$

For example:

$$X_0^{31|32}, \ldots, X_0^{24|32} = X_3^{7|8}, \ldots, X_3^{0|8}$$
$$X_0^{23|32}, \ldots, X_0^{16|32} = X_2^{7|8}, \ldots, X_2^{0|8}$$
$$X_0^{15|32}, \ldots, X_0^{08|32} = X_1^{7|8}, \ldots, X_1^{0|8}$$
$$X_0^{07|32}, \ldots, X_0^{00|32} = X_0^{7|8}, \ldots, X_0^{0|8}$$
$$X_1^{31|32}, \ldots, X_1^{24|32} = X_7^{7|8}, \ldots, X_7^{0|8}$$

It shall be the responsibility of the specification of storage formats and communication protocols to select which transformation between the $X^{|n}$ word view and the $X^{|8}$ byte view has to be applied. In the tradition of international standardization, Big Endian should generally be preferred.

There also exist at least two conventions for converting arrays of bits into byte sequences:

– **Little Endian:**

$$X_{8i+7-j} = X_i^{j|8}$$

For example:

$$X_{000|128}, \ldots, X_{007|128} = X_0^{7|8}, \ldots, X_0^{0|8}$$
$$X_{008|128}, \ldots, X_{015|128} = X_1^{7|8}, \ldots, X_1^{0|8}$$
$$X_{016|128}, \ldots, X_{023|128} = X_2^{7|8}, \ldots, X_2^{0|8}$$

– **Big Endian:**

$$X_{k-8(i+1)+j} = X_i^{j|8}$$

For example:

$$X_{127|128}, \ldots, X_{120|128} = X_0^{7|8}, \ldots, X_0^{0|8}$$
$$X_{119|128}, \ldots, X_{112|128} = X_1^{7|8}, \ldots, X_1^{0|8}$$
$$X_{111|128}, \ldots, X_{104|128} = X_2^{7|8}, \ldots, X_2^{0|8}$$

For the transformation of bit arrays into byte sequences, neither convention has any immediate implementation benefit on any hardware architecture, therefore the specification of a cryptographic algorithm should clearly specify a single one. Again, Big Endian should be preferred.

References

1. *Request for Comments on Candidate Algorithms for the Advanced Encryption Standard (AES)*. US Federal Register, Vol. 63, No. 177, pp. 49091–49093, September 14, 1998.
2. Brian R. Gladman: *Implementation Experience with AES Candidate Algorithms*. Second AES Conference, Rome, Italy, March 22–23, 1999.
3. Danny Cohen: *On Holy Wars and a Plea for Peace*. Computer, Vol. 14, No. 10, IEEE Computer Society, October 1981, pp. 48–54.
4. Hubert Kirrmann: *Data Format and Bus Compatibility in Multiprocessors*. IEEE Micro, Vol. 3, No. 4, August 1983, pp. 32–47.
5. D.B. Gustavson: *More on Big-Endian vs Little-Endian Byte Ordering*. IEEE Micro, Vol. 5, No. 3, 1985, p. 4.
6. David V. James: *Multiplexed Buses: The Endian Wars Continue*, IEEE Micro, Vol. 10, No. 3, June 1990, pp. 9–21.
7. James R. Gillig: *Endian-Neutral Software*. Dr. Dobb's Journal, Vol. 19, No. 11, October 1994, pp. 62–70, No. 19, November 1994, pp. 44–51.
8. John Rogers: *Your Own Endian Engine*, Dr. Dobb's Journal, Vol. 22, No. 11, pp. 30–36, November 1995.
9. William Stallings: *Endian Issues*. Byte, Vol. 20, No. 9, pp. 263–264, September 1995.
10. J. Reynolds, J. Postel: *Assigned Numbers*. Request for Comments RFC 1700, Network Working Group, October 1994, `<ftp://ftp.isi.edu/in-notes/rfc1700.txt>`.