Encryption Algorithms Made Natural

Misha Koshelev, Vladik Kreinovich, and Luc Longpré

Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
email {mkosh,vladik,longpre}@cs.utep.edu

***Abstract*—Modern cryptographic algorithms, such as DES, IDEA, etc., are very complex and therefore difficult to learn. Textbooks explain in detail *how* these algorithms work, but they usually do not explain *why* these algorithms were designed as they were. In this paper, we explain why, and thus, hopefully, make cryptographic algorithms easier to learn.**

Cryptography is important. Our economy is becoming more and more dependent on networked computers; the Internet, intranets, and the World Wide Web are becoming more and more important. Therefore, communication *security* is playing an ever-increasing role. One of the main methods of achieving communication security is *encryption*; therefore it is important to teach encryption and decryption algorithms.

Cryptography is complex. Modern cryptographic algorithms, such as DES, IDEA, etc., are very *complex* (see, e.g., [1, 3, 4, 2]) and therefore difficult to learn. Textbooks explain in detail *how* these algorithms work, but they usually do not explain *why* these algorithms were designed as they were. In this paper, we explain why, and thus, hopefully, make cryptographic algorithms *easier to learn.*

Requirements for encryption. The main idea of encryption is that instead of sending the original message m, we *encode* this message using some key k, and send the encoded message $c = E(m,k)$ instead. The receiving computer uses the same key k to reconstruct (*decode*) the original message $c, k \to m = D(c,k)$.

Communication must be *fast.* Therefore, we cannot wait for the entire message, we must start encoding as soon as we get a reasonable "chunk" of it. So, the message will be sent chunk by chunk and the receiving computer will decode it chunk by chunk.

There are two natural requirements for an encryption algorithm E:

- First, we must be able to reconstruct the message m from its code $c = E(m,k)$. This requirement actually consists of two requirements:
  - it must be *theoretically possible* to reconstruct, i.e., that *different* messages $m_1 \neq m_2$ must be transformed into *different* codes: $c_1 = E(m_1,k) \neq c_2 = E(m_2,k)$;
  - it must be *algorithmically possible* to reconstruct m if we know c and k.

  An operation $E(m,k)$ that satisfies these two properties is usually called *reversible.*
- Second, both encryption and decryption must be *fast.*

How can we satisfy them?

Component operations of an encryption algorithm must also be fast and reversible. The fastest possible operations are the ones that are *hardware supported.* Therefore, to make encryption fast, we must implement encryption as a *sequence* of directly hardware-supported computer operations.

For the *sequence* to be *reversible*, all its *component* operations must also be *reversible.* Thus, an encryption algorithm must consist of *reversible hardware-supported* computer operations.

What hardware-supported operations are reversible? In modern computers, two types of operations are usually hardware-implemented:

- *arithmetic operations,* such as addition, subtraction, and multiplication; and
- *bitwise operations,* such as bitwise AND, bitwise OR, etc.

(Division is usually microprogrammed as a sequence of multiplications, additions, and subtractions.) Let's check which of these operations are reversible.

Arithmetic operations. In a computer, a negative integer $-a$ is usually implemented as the 2's complement of a. In the simplified example of 8-bit words, $28_{10} = 0001\,1100_2$, and therefore, -28 is represented by adding 1 to the bitwise complement of 28's binary code, i.e., by $1110\,0011 + 1 = 1110\,0100$. The *main advantage* of 2's complement is that, in this format, the same algorithm implements arithmetic operations with positive and negative integers. A *side effect* of 2's complement is that it is a *reversible* operation: if we use the addition $+_2$ of 2's complement numbers for coding, i.e., $m, k \to c = m +_2 k$, we can easily reconstruct the original message m as $c +_2 (-k)$.

Multiplication, as implemented in computers, is *not* reversible because of overflow. We can, however, make it reversible if we consider multiplication modulo a prime number p, i.e., if we take $c = m \cdot k \pmod{p}$. Then, we can also easily reconstruct the original message m as $m = c \cdot k^{-1} \pmod{p}$, where k^{-1} is the inverse modulo p, i.e., the number for which $k^{-1} \cdot k \equiv 1 \pmod{p}$.

Bitwise operations. Which bitwise operations are reversible? A bitwise operations is a bit-by-bit application of some bit operation. Therefore, a bitwise operation is reversible if and only if the corresponding bit operation is reversible.

Let us describe all reversible bit operations $m, k \to c = E(m, k)$. Each bit operation can be represented by its *truth table*:

m	k	c
0	0	$*$
0	1	$*$
1	0	$*$
1	1	$*$

where $*$ means 0 or 1.

In principle, the first code $E(0, 0)$ can be equal either to 0 or to 1.

- Let us start with reversible operations for which $E(0, 0) = 0$. This means that the key $k = 0$ transforms the message $m = 0$ into the code $c = 0$.

  Since we want a *reversible* operation, this same key $k = 0$ must transform a different message $m = 1 \neq 0$ into a different code $E(1, 0) \neq E(0, 0) = 0$. Thus, we must have $E(1, 0) = 1$. Hence, our truth table takes the following form:

m	k	c
0	0	0
0	1	$*$
1	0	1
1	1	$*$

  To complete the description of this operation, we must choose $E(0, 1)$ and $E(1, 1)$. Let's start with $E(0, 1)$. We can take $E(0, 1) = 0$ or $E(0, 1) = 1$.

  – Let us first try $E(0, 1) = 0$. Then, since we need a reversible operation, the same key $k = 1$ must transform the message $m = 1 \neq 0$ into $E(1, 1) \neq E(0, 1) = 0$ and therefore $E(1, 1) = 1$. This leads to the following truth table:

m	k	c
0	0	0
0	1	0
1	0	1
1	1	1

    Here, for every key k, the code c simply coincides with the original message m so there is no coding at all. Thus, if we want coding, we must take $E(0, 1) = 1$.

  – When we take $E(0, 1) = 1$, reversibility implies that the same key $k = 1$ must transform the message $m = 1 \neq 0$ into $E(1, 1) \neq E(0, 1) = 1$, i.e., into $E(1, 1) = 0$.

  Thus, we arrive at the following truth table:

m	k	c
0	0	0
0	1	1
1	0	1
1	1	0

  If we, as usual, interpret 1 as true and 0 as false, we see that this table corresponds to *exclusive OR* $c = E(m, k) = m \oplus k$.

- Similarly, if we take $E(0, 0) = 1$, we arrive at the following truth table:

m	k	c
0	0	1
0	1	0
1	0	0
1	1	1

  This operation is the *equivalence* $c = E(m, k) = m \equiv k$. (Equivalence is equal to the negation of exclusive OR: $m \equiv k = \neg(m \oplus k)$.)

Conclusion. A successful encryption operation must consist of *additions*, *multiplications* modulo a prime number, and *exclusive ORs* (or their negations). This is exactly what the cryptographic algorithms DES and IDEA are made of.

Acknowledgments. This work was supported in part by NSF under grant No. EEC-9322370, by NASA under cooperative agreement NCCW-0089 and Grant NCC 5-97, and by the Future Aerospace Science and Technology Program (FAST) for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518.

References

[1] *Data Encryption Standard,* FIPS PUB 46, National Bureau of Standards, U.S. Department of Commerce, Washington, D.C., January 1977.

[2] C. Kaufman, R. Perlman, and M. Speciner, *Network Security*, Prentice Hall, Englewood Cliffs, New Jersey, 1995.

[3] A. Konheim, *Cryptography: A Primer*, John Wiley & Sons, 1981.

[4] J. Seberry and J. Pieprzyk, *Cryptography: An Introduction to Computer Security*, Prentice Hall, Englewood Cliffs, New Jersey, 1989.