

The Fundamentals of Data

THIS BOOK IS DESIGNED to address the fundamental concepts found in the emerging and rapidly evolving field of cyber forensics.

Before one can profess to be knowledgeable and fully cognizant of the breadth encompassing the professional discipline of cyber forensics, a foundation, rooted in the basics of information technology, data storage, handling, and processing, as well as how data is moved and manipulated, is essential.

For the cyber forensic investigator, data is evidence. Understanding how evidence emerges from data is pivotal; however, more important is being able to confidently articulate how evidential data was identified, collected, and processed.

As a cyber forensic investigator, simply pressing buttons or checking off options in a forensic software suite, without the knowledge of what is happening behind the scenes, creates a potential liability. Understanding the “life cycle” of data is pivotal, from its humble beginnings as electronic *bits*, evolving into bytes,

characters, then words, finally emerging as a language, as information, and perhaps eventually as evidence.

This book will provide a platform for both broadening as well as enhancing your skills in the basic elements of information technology as the technology supports and is embedded within the science of cyber forensic investigations.

As you read this book, you will encounter words that have been *italicized*. These words represent key concepts and are more fully defined by a working definition, which is included within a glossary at the end of the book. Should you desire an explanation of any *italicized* word, please refer to this glossary.

As with most tasks, one must crawl prior to walking and certainly before dashing off in a full run. Therefore, our first chapter begins naturally, at the beginning, with a discussion of the prime building blocks of data and how as a society we carbon-based humans have learned to communicate with a silicon-based technology—computers.

BASE 2 NUMBERING SYSTEM: BINARY AND CHARACTER ENCODING

Modern humans use character sets (or alphabets) to represent written sounds and words. In many alphabets, including Latin-based alphabets, each symbol or letter has its own phonetic sound.

The letter (or combination of letters, such as “ph”) is paired to its corresponding sound, forming a character code. It is through the combination of these symbols or letters that humans generate words, then phrases, and ultimately complex communication.

Symbolic characters, such as alphanumeric symbols found in Latin-based languages, work reasonably well for the complex computing power of the human brain. Computers, however, have yet to evolve to a level capable of exactly duplicating the complex processing—consistently, seamlessly, and reliably—of the human brain. Currently, computers can best communicate with other computers, in a manner based upon the principles of fundamental mathematics. Computer-to-human communication, while having evolved to a certain degree of voice replication, is still based, again, upon the principles of fundamental mathematics.

The current methodology for digital data transfer is called binary, and it is the basis for all computing technology. In order to understand how computers handle, move, store, access, save, delete, or otherwise manipulate data, it is essential to first understand the concepts of the *binary system*.

Binary is a name given to a Base 2 numbering system or encoding scheme. As the name Base 2 implies, there are two and only two possible states. In fact, a Base 2 encoding scheme is the only option of communication when only two possible states exist. Such an encoding scheme works well with electronic communication.

Consider electricity, where only two states are present. Electricity is either on or off; there exists no other possible option or state. A circuit is either open or closed. So, if we were to attach a light bulb to an electrical circuit we could visually see when the circuit is open or closed, as the light would either be off or on, respectively (remember, a closed circuit implies closed loop, and is therefore on).

COMMUNICATION IN A TWO-STATE UNIVERSE

Communication in a two-state environment is now possible; the light is either on, equal to “yes,” or it is off, equal to “no.” The potential to answer rudimentary, close-ended questions simply by indicating a response as either “yes” (1) or “no” (0) is entirely feasible.

This is important, being that presently, computers essentially can pass or store information as either electrical or magnetic states. Remember our light bulb can be “on” or “off” only.

Without going into great detail on the basics of electricity or magnetism, perhaps it is necessary to delve ever so gently into the very basic concepts of magnetism and electricity, and their relationship to data construction, storage, mobility, and processing.

ELECTRICITY AND MAGNETISM

Magnetism is the force whereby objects are attracted to or repelled by one another. Usually these objects are metals such as iron. (See Figure 1.1.)

Magnetism can store electricity, as in a battery, for example. Magnetism can also generate electricity (e.g., a generator). Magnetism, as with electricity, has only two states or opposing poles, positive and negative. Magnetic states can also be contained or preserved; for example, the direction of an iron oxide shaving can be manipulated by a magnet. This is called a magnetic domain, which is a series of atoms that point their poles in the same direction. A bar magnet is made up of a group of domains.

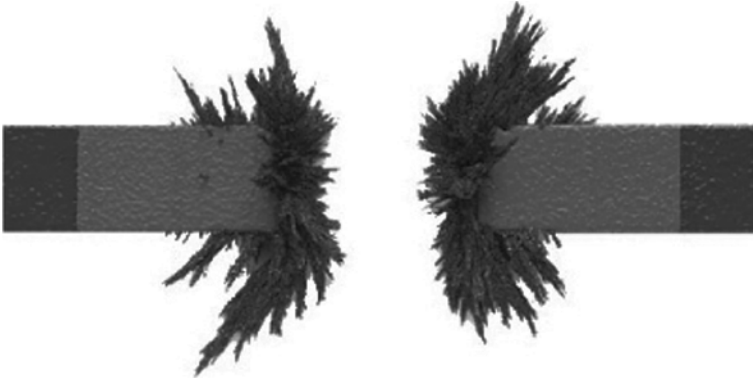


FIGURE 1.1 Magnetic Force

The most common source of magnetic fields is the electric current loop. Electricity is a type of activity arising from the existence of charge. The basic unit of charge is that on the proton or electron. The proton's charge is called positive while the electron's is negative.

Electricity tends to move or flow in its active state. This being the case, electricity is good at representing data in motion and magnetism is good at representing data at rest. Both, however, have two separate and opposing states, and as discussed, having two separate states allows for Base 2 digital communication.

With computers, the movement of digital data is easily represented by the two states of electricity or magnetism, and is conveniently presented by 1 and 0, respectively. Therefore, as the technology used to communicate and to represent data currently exists, this representation is accomplished through a two-state or binary numbering system.

BUILDING BLOCKS: THE ORIGINS OF DATA

A single zero (0) or a single one (1) is equal to what is called a bit. This representation of the two possible states of digital data is the smallest unit of data recognized or processed by a computer.

Therefore, in a one-bit, Base 2 encoding scheme (or as it is usually called, binary), we have the basic building blocks of a communication system: an ability to communicate through and between silicon-based technologies.

To communicate, for example, that the status of a light is on, we can assign the value of one (1). To communicate that the light is turned off, we can just as easily set the value to zero (0). For more complex situations, to indicate on

or off (yes or no), we can assign similar values: yes/on equals one (1), or no/off equals zero (0).

As communications grow in breadth and complexity, we are constrained by a single-bit, Base 2 encoding scheme. Essentially, we have two and only two possible outcomes of communication when constrained to a single bit (e.g., yes or no, on or off, 1 or 0).

Communication is possible, then, when only two states or conditions are required. Once we desire to expand the possibilities of communication options to a broader lexicon beyond a two-option state, one bit falls short, severely limiting communication possibilities.

GROWING THE BUILDING BLOCKS OF DATA

As you connect consecutive 0s and 1s (or bits), however, the ability to represent an increasingly larger set of characters, words, communication, and messaging possibilities increases geometrically. Just by adding another bit we double the potential outcomes or states (from two to four).

There are two possible states with one bit: one (1) or zero (0). Add another bit and now the number of possible states doubles: 00, 01, 10, and 11. Armed with such a system we can now represent more complex ideas or those conditions requiring more than a simplistic, on/off, yes/no, two-state description. For example, the four seasons could now be depicted with two bits, for example, 00 = winter, 01 = spring, 10 = summer, and 11 = fall.

To better understand the geometric growth of possible outcomes attained by combining bits, let's look at a few examples. The following discussion might send shivers down the spines of many readers, harking back to younger days and thoughts that math challenges were all behind us; however, an understanding of this basic math principle is critical in understanding the finer working details of data storage and ultimately data extraction using forensic software.

What is 2 to the 0 power?

A short explanation, which requires us to use the law of exponents, may be helpful to fire up those math synapses. One of the laws of exponents is:

$$\frac{n^x}{n^y} = n^{(x - y)}$$

for all n , x , and y . So, for example,

$$\frac{2^4}{2^2} = 2^{(4 - 2)}$$

$$\frac{2^4}{2^3} = 2^{(4-3)}$$

Now suppose we have the fraction:

$$\frac{2^4}{2^4} = 1$$

This fraction equals 1, because the numerator and the denominator are the same. If we apply the law of exponents, we get:

$$1 = \frac{2^4}{2^4} = 2^{(4-4)} = 2^0$$

$$\text{So, } 2^0 = 1$$

We can plug in any number in the place of 2, and that number raised to the zero power will still be 1. In fact, the whole proof works if we just plug in x for 2:

$$x^0 = x^{(4-4)} = \frac{x^4}{x^4} = 1$$

Wow, math flashbacks—we proved that 2^0 equals 1, so what about the following:

What is 2 to the first power? Second power? Third power?

Well, naturally, then we would answer $2 \times 1 = 2$, $2 \times 2 = 4$, and $2 \times 2 \times 2 = 8$!

Why is this important? It provides us with a better way to understand the geometric growth of possible states or outcomes attained by combining bits.

The following is a small example of the power of 2 and the exponential growth of increasing the bit combination possibilities:

$$\begin{aligned} 2^0 &= 1 \\ 2^1 &= 2 \\ 2^2 &= 4 \\ 2^3 &= 8 \\ 2^4 &= 16 \\ 2^5 &= 32 \\ 2^6 &= 64 \\ 2^7 &= 128 \\ 2^8 &= 256 \\ 2^9 &= 512 \\ 2^{10} &= 1,024 \end{aligned}$$

From our previous question, “What is 2 to the third power?” we find the answer in our encoding scheme. The number 2 represents our encoding scheme, Base 2 or binary; the power (third) represents how many bits will be strung together.

The answer 8 is how many outcomes or combinations are possible when we can string together three bits: 000, 111, 001, 010, 100, 110, 101, 011. That’s it! There are 8 possible outcomes, thus 2 to the third = 8.

MOVING BEYOND BASE 2

Eight possible outcomes or combinations is still fairly limiting for complex human communications, as necessary in today’s global business economy. As we continue to add 0s and 1s, the potential for very complex digital signaling or communication is increased, as stated exponentially. In fact, if we string together enough bits, we will be able to represent complete alphabets, alphabets of more than one language, and alphabets to even represent graphical concepts and expressions.

In order to represent the English alphabet (A–Z) and the numbers 1 through 10, we would need 26 unique representations for letters and 10 for numbers (0–9). Thus, we need 36 unique identifiers. How many bits would be needed to represent 36 unique identifiers or outcomes?

Well, from our earlier math lesson, 6 bits would easily cover our needs, resulting in 2 to the sixth or 2^6 , represented as a result of $2 \times 2 = 4 \times 2 = 8 \times 2 = 16 \times 2 = 32 \times 2 = 64$.

This six-bit combination not only produces the necessary 36 unique identifiers required, but also gives us some unique identifiers to spare, 28 to be specific, which we can use to represent special symbols such as (!, @, #, \$, %, ^, &,*) and so forth. In fact, 64 unique characters, while significant in the amount of combinations possible, do not suffice. In representing most basic characters of the English language, we use 7 bits or 2^7 , resulting in 128 unique characters to be identified or mapped.

AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE

The history of the American Standard Code for Information Interchange (ASCII) and its development is a long story and will only be briefly touched

upon in this chapter. The characters identified by 2^7 , or the 128-bit unique characters to be identified or mapped, are known as American Standard Code for Information Interchange/extended ASCII or just ASCII.

English-language personal computers used in America employ a seven-bit character code called American Standard Code for Information Interchange (ASCII), which allows for a character set of 128 items of upper- and lower-case Latin letters, Arabic numerals, signs, and control characters (i.e., $2^7 = 128$ code points). ASCII also serves as the foundation of the Universal Character Set (UCS), containing 0–9, A–Z, a–z, and special characters).

When an eighth bit is used as a “parity bit,” with its value used for checking whether or not data have been transmitted properly, then ASCII becomes an eight-bit, or one-byte (eight bits = one byte), character code. A true eight-bit character code allows for up to 256 items to be encoded ($2^8 = 256$ code points).¹

ASCII is a character-encoding scheme based on the ordering of the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that use text. Most modern character-encoding schemes, which support many more characters than did the original, are based on ASCII.

Tables 1.1 and 1.2 highlight the ASCII coding scheme and associated binary equivalents. Table 1.1 presents the numbers 0–9 and Table 1.2 depicts a list of special characters.

Alphabetic characters from the English language have similar representations in the ASCII coding scheme, as represented in Table 1.3.

TABLE 1.1 The Numbers Represented by 0–9

Character	ASCII	Binary
0	chr(48)	110000
1	chr(49)	110001
2	chr(50)	110010
3	chr(51)	110011
4	chr(52)	110100
5	chr(53)	110101
6	chr(54)	110110
7	chr(55)	110111
8	chr(56)	111000
9	chr(57)	111001

TABLE 1.2 Special Character Representation

Character	ASCII	Binary
!	chr(33)	100001
"	chr(34)	100010
#	chr(35)	100011
\$	chr(36)	100100
%	chr(37)	100101
&	chr(38)	100110
'	chr(39)	100111
(chr(40)	101000
)	chr(41)	101001
*	chr(42)	101010
+	chr(43)	101011

TABLE 1.3 English-Language Representations in the ASCII Coding Scheme

Character	ASCII	Binary
A	chr(65)	1000001
B	chr(66)	1000010
C	chr(67)	1000011
D	chr(68)	1000100
E	chr(69)	1000101
F	chr(70)	1000110
G	chr(71)	1000111
H	chr(72)	1001000
I	chr(73)	1001001
J	chr(74)	1001010
K	chr(75)	1001011
L	chr(76)	1001100
M	chr(77)	1001101

CHARACTER CODES: THE BASIS FOR PROCESSING TEXTUAL DATA

Many people are unaware of the fact that to a computer, textual data is also numerical data.

In modern computer systems, the individual characters of the scripts that humans use to record and transmit their languages are encoded in the form of binary numerical codes, just as are the Arabic numerals used in calculation programs. (See Tables 1.1, 1.2, and 1.3.) This is because the circuitry of the microprocessor that lies at the heart of a modern computer system can only do two things—calculate binary arithmetic operations and perform Boolean (i.e., true or false) logic operations.²

A *character code* pairs a character set, such as an alphabet, with something else, in this case with a decimal and/or binary system. An example most would be familiar with is the Braille Encoding System. While some of us may not know what the Braille encoded dots translate to, we have seen them, as many elevators display the floor number along with its Braille counterpart. This combination of information would be considered a character code.

The maximum characters possible in a character code depend upon the numbering system (Base 2 for Binary) and the number of bits. As demonstrated previously, the more bits in the character code, the bigger the character set.

In regard to character codes, it should also be noted that computers operate most efficiently when they process data in bytes. This is because their internal circuitry is usually designed with data pathways that are 8, 16, 32, or 64 bits wide. For that reason, a 10-bit or a 15-bit character code is clumsy and inefficient to handle inside a personal computer.

On the other hand, if too many bytes are used for encoding characters, computers will tend to process data inefficiently. For example, a three-byte character code could encode almost 17 million characters ($2^{24} = 16,777,216$ code points), which would cover all known historical and currently used character sets throughout the world. But the majority of the world's languages only need a one-byte (eight bits) code for character encoding, since they are alphabetical scripts.²

EXTENDED ASCII AND UNICODE

As people gradually required computers to understand additional characters, the ASCII set became restrictive. Extended ASCII is an eight-bit encoding scheme that includes the standard seven-bit ASCII characters as well as others.

Unicode is an industry standard developed by the Unicode Consortium. The Unicode Standard is a character coding system designed to support the worldwide interchange, processing, and display of the written texts of the diverse languages and technical disciplines of the modern world. In addition, it supports classical and historical texts of many written languages.

Unicode (or Universal Character Set) is another binary mapping scheme intended to be more universal, and includes a wider array of characters, which helps to accommodate a truly global character set. UCS incorporates the initial ASCII character mapping scheme, allowing for backward compatibility.

Unicode could be roughly described as “wide-body ASCII” that has been stretched from 8 bits to 16 bits. Unicode also allows for 8-, 16-, or 32-bit binary formats. A 16-bit coding scheme will allow for 65,536 potential outcomes. It is these 65,536 potential bit outcomes that allow Unicode to encompass most of the characters of all the world’s living languages.

Figure 1.2 shows the first 20 values from the Unicode Arabic character set, with the Arabic letter THAL highlighted.







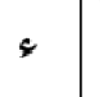

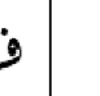




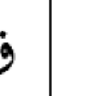


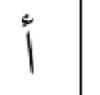

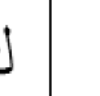
	060	061	062	063	064
0	 0600	 0610		 0630	 0640
1	 0601	 0611	 0621	 0631	 0641
2	 0602	 0612	 0622	 0632	 0642
3	 0603	 0613	 0623	 0633	 0643

FIGURE 1.2 Arabic Range 0600–06FF—Unicode
 Arabic Range 0600–06FF, Unicode Standard 5.2, www.unicode.org/charts/PDF/U0600.pdf,
 Copyright © 1991–2009 Unicode, Inc. All rights reserved.



FIGURE 1.3 The Arabic Letter THAL: Unicode Value 0630

Thus, if the user desired to generate the Arabic letter THAL, the individual would use the Unicode value 0630. This specific numeric value (0630) has a unique and special meaning when used in electronic, computational communications, and effectively represents the Arabic character THAL (see Figure 1.3) to a computing device, which currently can only interpret and calculate numeric values.

SUMMARY

This first chapter began with a brief introduction and discussion on how computational communication systems have evolved and how we attempt to codify our ability to communicate in a world with only two possible states, a binary existence.

We moved on to a further discussion, not only of the role that a binary numbering systems plays in our ability to represent the most basic patterns of human communication, but also how this binary system has allowed us to expand into producing complex alphabetic patterns, character sets, and ultimately a method that enables us to represent entire languages.

From primary states of existence to representation of complex language patterns, carbon-based communication, like that of silicon-based communication, began with primary building blocks. For computers, that is the bit, represented by either a one (1) or a zero (0).

The combination and pairing of these 1s and 0s allows computational machines to communicate and in the end, to perform complex data manipulation. Representing complex textual words or graphics to a mechanical device is now possible simply by arranging and rearranging the pairings and groupings of 1s and 0s.

Establishing a method of pairing alphabetic characters with the characters' binary equivalents produced character codes (which have evolved into more complex character sets), allowing us to expand our ability to represent a greater range of characters and also control how computers store, manipulate, and transmit data.

Binary representation of numbers and characters is required when working in a world restricted to only two states of description or existence (e.g., electrical or magnetic). Fortunately for us, our human world is more robust, more colorful, and exists in many states, well beyond that of a binary life. It would also be more difficult and time-consuming if we were required to perform all of our figuring, communicating, and similar functioning with numbers or letters represented by groups and pairings of 1s and 0s (e.g., the statement, “Hi, my name is Tom” would be represented by a string of ones and zeroes 144 characters long, 01001000 01101001 00101100 00100000 01101101 01111001 00100000 01101110 01100001 01101101 01100101 00100000 01101001 01110011 00100000 01010100 01101111 01101101).

Luckily, only silicon-based computational devices have to process data in this binary fashion. Humans, on the other hand, have a more convenient method. Humans work more effectively and more efficiently representing numbers in a Base 10 or decimal equivalent to the computer’s binary representation, making life a tad bit easier.

In Chapter 2 we discuss how to convert a binary number into its decimal equivalent, and why this knowledge is also essential for gaining a greater depth of understanding of how data is stored, moved, manipulated, and processed and how this treatment of data is critical to a better understanding of cyber forensics.

NOTES

1. Searle, S. “A Brief History of Character Codes in North America, Europe, and East Asia,” TRON Web, Sakamura Laboratory, University Museum, University of Tokyo, August 6, 2004, retrieved October 2009, <http://tronweb.super-nova.co.jp/characcodehist.html>.
2. Ibid.

<http://www.pbookshop.com>