Math Refresher for

Cryptography W202

Master of Information and Cybersecurity (MICS)
University of California, Berkeley

Kevin R. Crook

(last update 8/14/2018)

Cybersecurity math falls into a category of mathematics called "discrete mathematics". To be successful at cybersecurity math, students need to first be proficient at college level algebra.

For some students, it may have been a while since you have taken a course in college level algebra. We don't need all of college algebra, so this document is going to review only the concepts from college algebra that you will need to understand cybersecurity math.

The reference textbook that we will be using for this refresher is an open source textbook:

https://openstax.org/details/books/college-algebra

OpenStax is a non-profit funded by the Bill and Melinda Gates Foundation that provides free peer-reviewed textbooks for college course. They are now heavily used in the undergraduate math courses. The previous link above is to their "OpenStax College Algebra" textbook. (I will refer to it as **OSCA** going forward.)

In this refresher, I'm only pulling out the subset of material that will directly be needed to save students the time of reviewing an entire textbook. For each topic, I will paste relevant tables, summaries, etc. from the textbook.

If this is enough to refresh your understanding, you are good to go and can move on.

If you need more work on a specific topic, I provide the relevant section of the textbook which you will want to read through and work the example problems.

If you still need more work on a specific topic, you may want to consider a private tutor. This would be on your own at your own expense. We cannot recommend nor arrange tutors.

There are also some additional topics not covered in OSCA that will be covered here.

Real Numbers and Properties

Purpose: In cybersecurity, we will be building closed mathematical systems. It's essential that we understand the hierarchy of real numbers and the subsets that make up real numbers. It's also essential that we understand the properties of real numbers.

Source: OSCA, 1.1 Real Numbers: Algebra Essentials

SETS OF NUMBERS

The set of **natural numbers** includes the numbers used for counting: $\{1, 2, 3, ...\}$.

The set of **whole numbers** is the set of natural numbers plus zero: $\{0, 1, 2, 3, ...\}$.

The set of **integers** adds the negative natural numbers to the set of whole numbers: $\{..., -3, -2, -1, 0, 1, 2, 3, ...\}$.

The set of **rational numbers** includes fractions written as $\left\{\frac{m}{n} \mid m \text{ and } n \text{ are integers and } n \neq 0\right\}$.

The set of **irrational numbers** is the set of numbers that are not rational, are nonrepeating, and are nonterminating: $\{h \mid h \text{ is not a rational number}\}$.

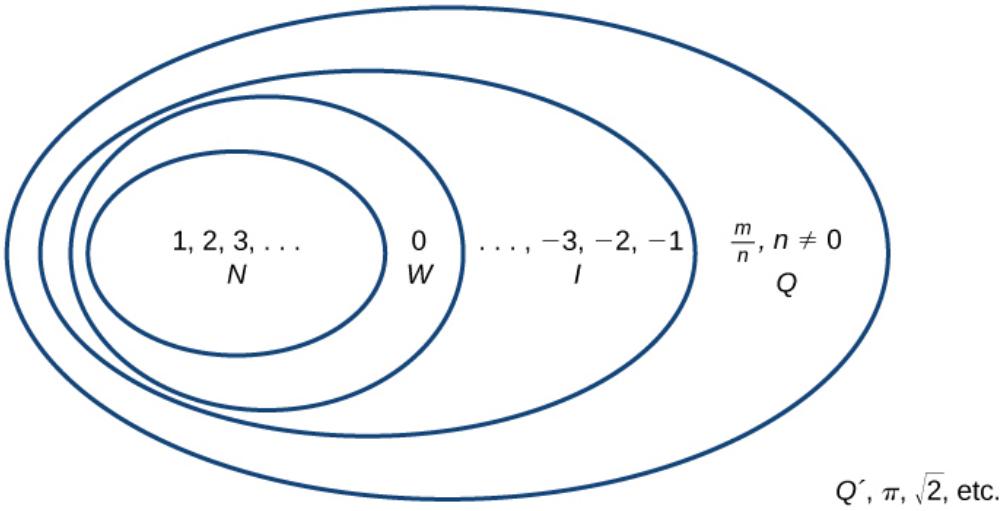


Figure 2. Sets of numbers

N: the set of natural numbers

W: the set of whole numbers

I: the set of integers

Q: the set of rational numbers

Q′: the set of irrational numbers

PROPERTIES OF REAL NUMBERS

The following properties hold for real numbers *a*, *b*, and *c*.

	Addition	Multiplication
Commutative Property	$a + b = b + a$	$a \cdot b = b \cdot a$
Associative Property	$a + (b + c) = (a + b) + c$	$a\,(bc) = (ab)\,c$
Distributive Property	$a \cdot (b + c) = a \cdot b + a \cdot c$	
Identity Property	There exists a unique real number called the additive identity, 0, such that, for any real number *a* $a + 0 = a$	There exists a unique real number called the multiplicative identity, 1, such that, for any real number *a* $a \cdot 1 = a$
Inverse Property	Every real number a has an additive inverse, or opposite, denoted –*a*, such that $a + (-a) = 0$	Every nonzero real number *a* has a multiplicative inverse, or reciprocal, denoted $\frac{1}{a}$, such that $a \cdot \left(\frac{1}{a}\right) = 1$

Exponents

Purpose: Exponents are heavily used in cybersecurity, both in actual algorithms and in mathematical proofs. Be sure you know them well!

Source: OSCA, 1.2 Exponents and Scientific Notation

Rules of Exponents For nonzero real numbers a and b and integers m and n	
Product rule	$a^m \cdot a^n = a^{m+n}$
Quotient rule	$\frac{a^m}{a^n} = a^{m-n}$
Power rule	$(a^m)^n = a^{m \cdot n}$
Zero exponent rule	$a^0 = 1$
Negative rule	$a^{-n} = \frac{1}{a^n}$
Power of a product rule	$(a \cdot b)^n = a^n \cdot b^n$
Power of a quotient rule	$\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$

Roots

Purpose: Roots are heavily used in cybersecurity, both in actual algorithms and in mathematical proofs. Be sure you know them well!

Source: OSCA, 1.3 Radicals and Rational Exponents

PRINCIPAL SQUARE ROOT

The **principal square root** of a is the nonnegative number that, when multiplied by itself, equals a. It is written as a **radical expression,** with a symbol called a **radical** over the term called the **radicand**: $\sqrt{a}$.

THE PRODUCT RULE FOR SIMPLIFYING SQUARE ROOTS

If a and b are nonnegative, the square root of the product ab is equal to the product of the square roots of a and b.

$$\sqrt{ab} = \sqrt{a} \cdot \sqrt{b}$$

THE QUOTIENT RULE FOR SIMPLIFYING SQUARE ROOTS

The square root of the quotient $\frac{a}{b}$ is equal to the quotient of the square roots of a and b, where $b \neq 0$.

$$\sqrt{\frac{a}{b}} = \frac{\sqrt{a}}{\sqrt{b}}$$

PRINCIPAL *N*TH ROOT

If a is a real number with at least one nth root, then the **principal *n*th root** of a, written as $\sqrt[n]{a}$, is the number with the same sign as a that, when raised to the nth power, equals a. The **index** of the radical is n.

RATIONAL EXPONENTS

Rational exponents are another way to express principal nth roots. The general form for converting between a radical expression with a radical symbol and one with a rational exponent is

$$a^{\frac{m}{n}} = (\sqrt[n]{a})^m = \sqrt[n]{a^m}$$

Logarithms

Purpose: Logarithms are heavily used in cybersecurity, both in actual algorithms and in mathematical proofs. Pay close attention to the logarithmic properties. Be sure you know them well!

In math, log means the logarithm in base 10.

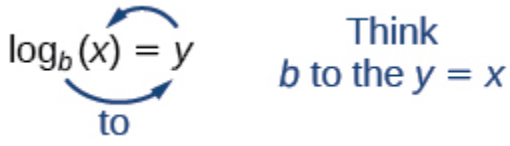
In theoretical computer science, log means the logarithm in base 2.

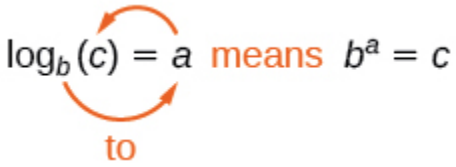
In programming languages, log often means the natural logarithm, that is, the logarithm in base e.

Note that e is Euler's Number approximately equal to 2.71828 and is derived as follows:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots$$

Source: OSCA, 6.3 Logarithmic Functions





DEFINITION OF THE LOGARITHMIC FUNCTION

A **logarithm** base b of a positive number x satisfies the following definition.

For $x > 0, b > 0, b \neq 1$,

$$y = \log_b(x) \text{ is equivalent to } b^y = x$$

where,

- we read $\log_b(x)$ as, "the logarithm with base b of x" or the "log base b of x."
- the logarithm y is the exponent to which b must be raised to get x.

Also, since the logarithmic and exponential functions switch the x and y values, the domain and range of the exponential function are interchanged for the logarithmic function. Therefore,

- the domain of the logarithm function with base b is $(0, \infty)$.
- the range of the logarithm function with base b is $(-\infty, \infty)$.

DEFINITION OF THE COMMON LOGARITHM

A **common logarithm** is a logarithm with base 10. We write $\log_{10}(x)$ simply as $\log(x)$. The common logarithm of a positive number x satisfies the following definition.

For $x > 0$,

$$y = \log(x) \text{ is equivalent to } 10^y = x$$

We read $\log(x)$ as, "the logarithm with base 10 of x" or "log base 10 of x."

The logarithm y is the exponent to which 10 must be raised to get x.

DEFINITION OF THE NATURAL LOGARITHM

A **natural logarithm** is a logarithm with base e. We write $\log_e(x)$ simply as $\ln(x)$. The natural logarithm of a positive number x satisfies the following definition.

For $x > 0$,

$$y = \ln(x) \text{ is equivalent to } e^y = x$$

We read $\ln(x)$ as, "the logarithm with base e of x" or "the natural logarithm of x."

The logarithm y is the exponent to which e must be raised to get x.

Since the functions $y = e^x$ and $y = \ln(x)$ are inverse functions, $\ln(e^x) = x$ for all x and $e^{\ln(x)} = x$ for $x > 0$.

Source: OSCA, 6.5 Logarithmic Properties

The Product Rule for Logarithms	$\log_b(MN) = \log_b(M) + \log_b(N)$
The Quotient Rule for Logarithms	$\log_b\left(\frac{M}{N}\right) = \log_b M - \log_b N$
The Power Rule for Logarithms	$\log_b(M^n) = n\log_b M$
The Change-of-Base Formula	$\log_b M = \frac{\log_n M}{\log_n b} \qquad n > 0, n \neq 1, b \neq 1$

Inverse Functions

Purpose: In cryptography, a lot of encryption and decryption algorithms are inverses of each other. You need to understand the basic concepts of what makes a function invertible and that not all relations are functions and not all functions are invertible.

Source: OSCA, 3.1 Functions and Function Notation

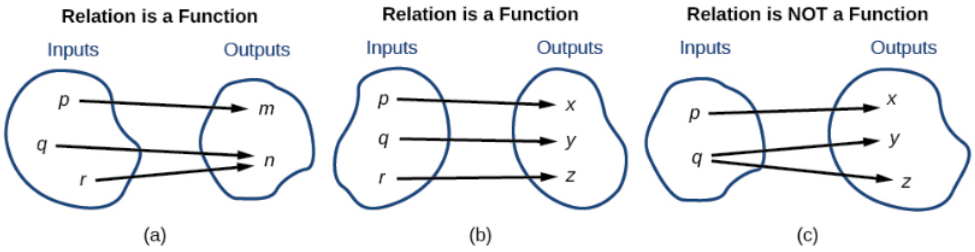


Figure 1. (a) This relationship is a function because each input is associated with a single output. Note that input q and r both give output n. (b) This relationship is also a function. In this case, each input is associated with a single output. (c) This relationship is not a function because input q is associated with two different outputs.

ONE-TO-ONE FUNCTION

A **one-to-one function** is a function in which each output value corresponds to exactly one input value. There are no repeated *x*- or *y*-values.

Source: OSCA, 3.7 Inverse Functions

INVERSE FUNCTION

For any **one-to-one function** $f(x) = y$, a function $f^{-1}(x)$ is an **inverse function** of f if $f^{-1}(y) = x$. This can also be written as $f^{-1}(f(x)) = x$ for all x in the domain of f. It also follows that $f(f^{-1}(x)) = x$ for all x in the domain of f^{-1} if f^{-1} is the inverse of f.

The notation f^{-1} is read "f inverse." Like any other function, we can use any variable name as the input for f^{-1}, so we will often write $f^{-1}(x)$, which we read as "f inverse of x." Keep in mind that

$$f^{-1}(x) \neq \frac{1}{f(x)}$$

and not all functions have inverses.

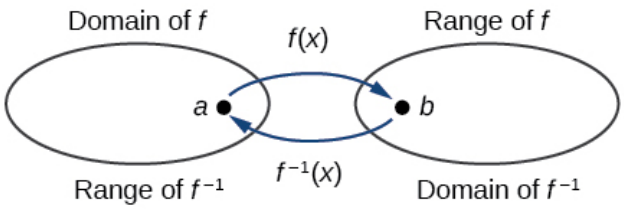


Figure 3. Domain and range of a function and its inverse

Solving Systems of Linear Equations

Purpose: In cryptography, we will need to be able to solve systems of linear equations. One reason is to prepare us for solving systems of linear congruences in modulo prime using the Chinese Remainder Theorem. Another area is to prepare us for quorum based secret sharing using Shamir's Secret Sharing over a finite field in modulo prime.

Source: OSCA, 7.2 Systems of Linear Equations: Three Variables

Elimination - We can solve a system of three equations in three variables using elimination. You can use this method for solving problems in class, however, most students will probably want to use an augmented matrix and a solver detailed in the next section. An example is provided here just in case:

Given a linear system of three equations, solve for three unknowns.

1. Pick any pair of equations and solve for one variable.
2. Pick another pair of equations and solve for the same variable.
3. You have created a system of two equations in two unknowns. Solve the resulting two-by-two system.
4. Back-substitute known variables into any one of the original equations and solve for the missing variable.

$$\begin{aligned} x-2y+3z &= 9 \quad (1)\\ -x+3y-z &= -6 \quad (2)\\ 2x-5y+5z &= 17 \quad (3) \end{aligned}$$

There will always be several choices as to where to begin, but the most obvious first step here is to eliminate x by adding equations (1) and (2).

$$\begin{aligned} x-2y+3z &= 9 \quad (1)\\ -x+3y-z &= -6 \quad (2)\\ \hline y+2z &= 3 \quad (3) \end{aligned}$$

The second step is multiplying equation (1) by -2 and adding the result to equation (3). These two steps will eliminate the variable x.

$$\begin{aligned} -2x+4y-6z &= -18 \quad (1)\text{ multiplied by } -2\\ 2x-5y+5z &= 17 \quad (3)\\ \hline -y-z &= -1 \quad (5) \end{aligned}$$

In equations (4) and (5), we have created a new two-by-two system. We can solve for z by adding the two equations.

$$\begin{aligned} y+2z &= 3 \quad (4)\\ -y-z &= -1 \quad (5)\\ \hline z &= 2 \quad (6) \end{aligned}$$

Choosing one equation from each new system, we obtain the upper triangular form:

$$\begin{aligned} x-2y+3z &= 9 \quad (1)\\ y+2z &= 3 \quad (4)\\ z &= 2 \quad (6) \end{aligned}$$

Next, we back-substitute $z = 2$ into equation (4) and solve for y.

$$\begin{aligned} y+2(2) &= 3\\ y+4 &= 3\\ y &= -1 \end{aligned}$$

Finally, we can back-substitute $z = 2$ and $y = -1$ into equation (1). This will yield the solution for x.

$$\begin{aligned} x-2(-1)+3(2) &= 9\\ x+2+6 &= 9\\ x &= 1 \end{aligned}$$

Source: OSCA, 7.6 Solving Systems with Gaussian Elimination

For a system of equations, with n equations and n unknowns, we can write an augmented matrix and use a computer program called a solver to solve for the n unknowns. One of the most popular is Gaussian Elimination. While you may want to review the logic for Gaussian Elimination in the textbook, the main thing is to learn how to write a system of equations as an augmented matrix and put it into a solver to get the solution.

$$\begin{aligned} x + 2y - z &= 3 \\ 2x - y + 2z &= 6 \\ x - 3y + 3z &= 4 \end{aligned}$$

$$\left[\begin{array}{rrr|r} 1 & 2 & -1 & 3 \\ 2 & -1 & 2 & 6 \\ 1 & -3 & 3 & 4 \end{array}\right]$$

Source: OSCA, 7.6 Solving Systems with Inverses

In addition to Gaussian Elimination, another way is to use Inverses. While you may want to review the logic for Inverses in the textbook, as mentioned above the main thing is to learn how to use an augmented matrix and a solver.

Quadratic Equations and Functions

Purpose: In cryptography, we will use closed mathematical systems involving squares and square roots. We need to first have an understanding of quadratic equations in the real number system, and pay especially close attention to the relationship between factoring and finding roots.

Source: OSCA, 2.5 Quadratic Equations

THE ZERO-PRODUCT PROPERTY AND QUADRATIC EQUATIONS

The **zero-product property** states

$$\text{If } a \cdot b = 0, \text{ then } a = 0 \text{ or } b = 0,$$

where *a* and *b* are real numbers or algebraic expressions.

A **quadratic equation** is an equation containing a second-degree polynomial; for example

$$ax^2 + bx + c = 0$$

where *a*, *b*, and *c* are real numbers, and if $a \neq 0$, it is in standard form.

Given a quadratic equation with the leading coefficient of 1, factor it.

1. Find two numbers whose product equals *c* and whose sum equals *b*.
2. Use those numbers to write two factors of the form $(x + k)$ or $(x - k)$, where *k* is one of the numbers found in step 1. Use the numbers exactly as they are. In other words, if the two numbers are 1 and -2, the factors are $(x + 1)(x - 2)$.
3. Solve using the zero-product property by setting each factor equal to zero and solving for the variable.

Solving a Quadratic Equation by Factoring when the Leading Coefficient is not 1

When the leading coefficient is not 1, we factor a quadratic equation using the method called grouping, which requires four terms. With the equation in standard form, let's review the grouping procedures:

1. With the quadratic in standard form, $ax^2 + bx + c = 0$, multiply $a \cdot c$.
2. Find two numbers whose product equals ac and whose sum equals b.
3. Rewrite the equation replacing the bx term with two terms using the numbers found in step 1 as coefficients of *x*.
4. Factor the first two terms and then factor the last two terms. The expressions in parentheses must be exactly the same to use grouping.
5. Factor out the expression in parentheses.
6. Set the expressions equal to zero and solve for the variable.

THE SQUARE ROOT PROPERTY

With the x^2 term isolated, the square root property states that:

$$\text{if } x^2 = k, \text{ then } x = \pm\sqrt{k}$$

where *k* is a nonzero real number.

Given a quadratic equation with an x^2 term but no x term, use the square root property to solve it.

1. Isolate the x^2 term on one side of the equal sign.
2. Take the square root of both sides of the equation, putting a $\pm$ sign before the expression on the side opposite the squared term.
3. Simplify the numbers on the side with the $\pm$ sign.

Completing the Square

Not all quadratic equations can be factored or can be solved in their original form using the square root property. In these cases, we may use a method for solving a **quadratic equation** known as **completing the square**. Using this method, we add or subtract terms to both sides of the equation until we have a perfect square trinomial on one side of the equal sign. We then apply the square root property. To complete the square, the leading coefficient, *a*, must equal 1. If it does not, then divide the entire equation by *a*. Then, we can use the following procedures to solve a quadratic equation by completing the square.

THE QUADRATIC FORMULA

Written in standard form, $ax^2 + bx + c = 0$,any quadratic equation can be solved using the **quadratic formula**:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

where *a*, *b*, and *c* are real numbers and $a \neq 0$.

THE DISCRIMINANT

For $ax^2 + bx + c = 0$, where a, b, and c are real numbers, the **discriminant** is the expression under the radical in the quadratic formula: $b^2 - 4ac$. It tells us whether the solutions are real numbers or complex numbers and how many solutions of each type to expect.

Value of Discriminant	Results
$b^2 - 4ac = 0$	One rational solution (double solution)
$b^2 - 4ac > 0$, perfect square	Two rational solutions
$b^2 - 4ac > 0$, not a perfect square	Two irrational solutions
$b^2 - 4ac < 0$	Two complex solutions

Source: OSCA, 5.1 Quadratic Functions

FORMS OF QUADRATIC FUNCTIONS

A quadratic function is a polynomial function of degree two. The graph of a **quadratic function** is a parabola.

The **general form of a quadratic function** is $f(x) = ax^2 + bx + c$ where $a, b,$ and c are real numbers and $a \neq 0$.

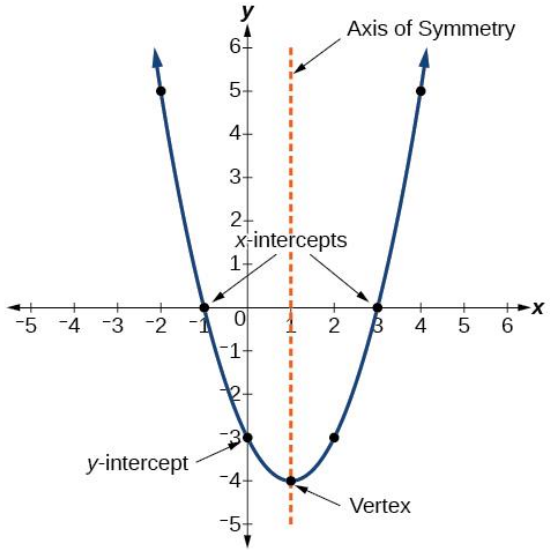
The **standard form of a quadratic function** is $f(x) = a(x - h)^2 + k$ where $a \neq 0$.

The vertex (h, k) is located at

$$h = -\frac{b}{2a},\ k = f(h) = f\left(\frac{-b}{2a}\right)$$

Given a quadratic function $f(x)$, find the y- and x-intercepts.

1. Evaluate $f(0)$ to find the y-intercept.
2. Solve the quadratic equation $f(x) = 0$ to find the x-intercepts.



- The graph of a quadratic is a parabola.
- The x-intercepts are the roots (but not necessarily square roots).
- If there are no x-intercepts (the parabola lies above the x axis), there are no roots. Not every quadratic has roots.
- If the axis of symmetry lies on the y axis, it is a square and has square roots (-r,0) and (+r,0)

Polynomial Division, Synthetic Division, Remainder Theorem, Fundamental Theorem of Algebra

Purpose: In cryptography, we will use polynomial division in various proofs. It will also help us to understand how finding zeros (roots) and remainders (residues) relate to factoring.

Source: OSCA, 5.4 Dividing Polynomials

THE DIVISION ALGORITHM

The **Division Algorithm** states that, given a polynomial dividend $f(x)$ and a non-zero polynomial divisor $d(x)$ where the degree of $d(x)$ is less than or equal to the degree of $f(x)$, there exist unique polynomials $q(x)$ and $r(x)$ such that

$$f(x) = d(x)q(x) + r(x)$$

$q(x)$ is the quotient and $r(x)$ is the remainder. The remainder is either equal to zero or has degree strictly less than $d(x)$.

If $r(x) = 0$, then $d(x)$ divides evenly into $f(x)$. This means that, in this case, both $d(x)$ and $q(x)$ are factors of $f(x)$.

$x + 2 \overline{)\, 2x^3 - 3x^2 + 4x + 5}$	Set up the division problem.
$\begin{array}{r} 2x^2 \\ x + 2 \overline{)\, 2x^3 - 3x^2 + 4x + 5} \end{array}$	$2x^3$ divided by x is $2x^2$.
$\begin{array}{r} 2x^2 \\ x + 2 \overline{)\, 2x^3 - 3x^2 + 4x + 5} \\ -(2x^3 + 4x^2) \\ \hline -7x^2 + 4x \end{array}$	Multiply $x + 2$ by $2x^2$. Subtract. Bring down the next term.
$\begin{array}{r} 2x^2 - 7x \\ x + 2 \overline{)\, 2x^3 - 3x^2 + 4x + 5} \\ -(2x^3 + 4x^2) \\ \hline -7x^2 + 4x \\ -(-7x^2 - 14x) \\ \hline 18x + 5 \end{array}$	$-7x^2$ divided by x is $-7x$. Multiply $x + 2$ by $-7x$. Subtract. Bring down the next term.
$\begin{array}{r} 2x^2 - 7x + 18 \\ x + 2 \overline{)\, 2x^3 - 3x^2 + 4x + 5} \\ -(2x^3 + 4x^2) \\ \hline -7x^2 + 4x \\ -(-7x^2 - 14x) \\ \hline 18x + 5 \\ -(18x + 36) \\ \hline -31 \end{array}$	$18x$ divided by x is 18. Multiply $x + 2$ by 18. Subtract.

We can identify the **dividend**, the **divisor**, the **quotient**, and the **remainder**.

$$2x^3 - 3x^2 + 4x + 5 = (x + 2)(2x^2 - 7x + 18) + (-31)$$

Dividend — Divisor — Quotient — Remainder

Writing the result in this manner illustrates the Division Algorithm.

SYNTHETIC DIVISION

Synthetic division is a shortcut that can be used when the divisor is a binomial in the form $x - k$ where k is a real number. In **synthetic division**, only the coefficients are used in the division process.

$$
\begin{array}{r|rrrr}
2 & 2 & -3 & 4 & 5 \\
 & -2 & -4 & & \\
\hline
 & -7 & 14 & & \\
 & & 18 & -36 & \\
\hline
 & & & -31 &
\end{array}
\qquad
\begin{array}{r|rrrr}
-2 & 2 & -3 & 4 & 5 \\
 & & -4 & 14 & -36 \\
\hline
 & 2 & -7 & 18 & -31
\end{array}
$$

Source: OSCA, 5.5 Zeros of Polynomial Functions

THE REMAINDER THEOREM

If a polynomial $f(x)$ is divided by $x - k$, then the remainder is the value $f(k)$.

Given a polynomial function f, evaluate $f(x)$ at $x = k$ using the Remainder Theorem.

1. Use synthetic division to divide the polynomial by $x - k$.
2. The remainder is the value $f(k)$.

THE FACTOR THEOREM

According to the **Factor Theorem**, k is a zero of $f(x)$ if and only if $(x - k)$ is a factor of $f(x)$.

THE RATIONAL ZERO THEOREM

The **Rational Zero Theorem** states that, if the polynomial $f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ has integer coefficients, then every rational zero of $f(x)$ has the form $\frac{p}{q}$ where p is a factor of the constant term a_0 and q is a factor of the leading coefficient a_n.

When the leading coefficient is 1, the possible rational zeros are the factors of the constant term.

THE FUNDAMENTAL THEOREM OF ALGEBRA

The **Fundamental Theorem of Algebra** states that, if $f(x)$ is a polynomial of degree $n > 0$, then $f(x)$ has at least one complex zero.

We can use this theorem to argue that, if $f(x)$ is a polynomial of degree $n > 0$, and a is a non-zero real number, then $f(x)$ has exactly n linear factors

$$f(x) = a(x - c_1)(x - c_2)\ldots(x - c_n)$$

where $c_1, c_2, \ldots, c_n$ are complex numbers. Therefore, $f(x)$ has n roots if we allow for multiplicities.

Sequences, Factorials, Arithmetic Sequences, Geometric Sequences, Series

Purpose: We will have mathematical proofs based on various types of sequences. Also, in cryptography, we will study collisions based on probability. These topics are needed to understand collision probability.

Source: OSCA, 9.1 Sequences and Their Notations

SEQUENCE

A **sequence** is a function whose domain is the set of positive integers. A **finite sequence** is a sequence whose domain consists of only the first n positive integers. The numbers in a sequence are called **terms**. The variable a with a number subscript is used to represent the terms in a sequence and to indicate the position of the term in the sequence.

$$a_1, a_2, a_3, \ldots, a_n, \ldots$$

We call a_1 the first term of the sequence, a_2 the second term of the sequence, a_3 the third term of the sequence, and so on. The term a_n is called the $n\text{th}$ **term of the sequence**, or the general term of the sequence. An **explicit formula** defines the $n\text{th}$ term of a sequence using the position of the term. A sequence that continues indefinitely is an **infinite sequence**.

Does a sequence always have to begin with a_1?

No. In certain problems, it may be useful to define the initial term as a_0 instead of a_1. In these problems, the domain of the function includes 0.

RECURSIVE FORMULA

A **recursive formula** is a formula that defines each term of a sequence using preceding term(s). Recursive formulas must always state the initial term, or terms, of the sequence.

Must the first two terms always be given in a recursive formula?

No. The Fibonacci sequence defines each term using the two preceding terms, but many recursive formulas define each term using only one preceding term. These sequences need only the first term to be defined.

FACTORIAL

***n* factorial** is a mathematical operation that can be defined using a recursive formula. The factorial of n, denoted $n!$, is defined for a positive integer n as:

$$0! = 1$$
$$1! = 1$$
$$n! = n\,(n-1)\,(n-2)\cdots(2)\,(1)\,, \text{for } n \geq 2$$

The special case $0!$ is defined as $0! = 1$.

Source: OSCA, 9.2 Arithmetic Sequences

ARITHMETIC SEQUENCE

An **arithmetic sequence** is a sequence that has the property that the difference between any two consecutive terms is a constant. This constant is called the **common difference**. If a_1 is the first term of an arithmetic sequence and d is the common difference, the sequence will be:

$$\{a_n\} = \{a_1, a_1 + d, a_1 + 2d, a_1 + 3d, ...\}$$

RECURSIVE FORMULA FOR AN ARITHMETIC SEQUENCE

The recursive formula for an arithmetic sequence with common difference d is:

$$a_n = a_{n-1} + d \qquad n \geq 2$$

EXPLICIT FORMULA FOR AN ARITHMETIC SEQUENCE

An explicit formula for the $n\text{th}$ term of an arithmetic sequence is given by

$$a_n = a_1 + d(n-1)$$

Source: OSCA, 9.3 Geometric Sequences

DEFINITION OF A GEOMETRIC SEQUENCE

A **geometric sequence** is one in which any term divided by the previous term is a constant. This constant is called the **common ratio** of the sequence. The common ratio can be found by dividing any term in the sequence by the previous term. If a_1 is the initial term of a geometric sequence and r is the common ratio, the sequence will be

$$\{a_1,\ a_1r,\ a_1r^2,\ a_1r^3,\ \ldots\}.$$

RECURSIVE FORMULA FOR A GEOMETRIC SEQUENCE

The recursive formula for a geometric sequence with common ratio r and first term a_1 is

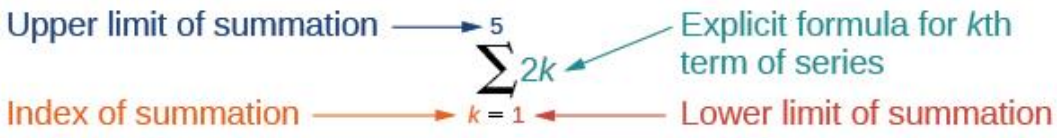
$$a_n = ra_{n-1}, n \geq 2$$

EXPLICIT FORMULA FOR A GEOMETRIC SEQUENCE

The *n*th term of a geometric sequence is given by the **explicit formula**:

$$a_n = a_1r^{n-1}$$

Source: OSCA, 9.4 Series and Their Notations



SUMMATION NOTATION

The sum of the first n terms of a **series** can be expressed in **summation notation** as follows:

$$\sum_{k=1}^{n} a_k$$

This notation tells us to find the sum of a_k from $k = 1$ to $k = n$.

k is called the **index of summation**, 1 is the **lower limit of summation**, and n is the **upper limit of summation**.

Does the lower limit of summation have to be 1?

No. The lower limit of summation can be any number, but 1 is frequently used. We will look at examples with lower limits of summation other than 1.

Combinatorics: Combinations, Permutations, Pascal's Triangle

Purpose: In cryptography, we will study collisions based on probability. These topics are needed to understand collision probability.

Source: OSCA, 9.1 Sequences and Their Notations

FACTORIAL

***n* factorial** is a mathematical operation that can be defined using a recursive formula. The factorial of n, denoted $n!$, is defined for a positive integer n as:

$$0! = 1$$
$$1! = 1$$
$$n! = n\,(n-1)\,(n-2)\cdots(2)\,(1)\,, \text{for } n \geq 2$$

The special case $0!$ is defined as $0! = 1$.

Source: OSCA, 9.5 Counting Principles

THE ADDITION PRINCIPLE

According to the **Addition Principle**, if one event can occur in m ways and a second event with no common outcomes can occur in n ways, then the first *or* second event can occur in $m + n$ ways.

THE MULTIPLICATION PRINCIPLE

According to the **Multiplication Principle**, if one event can occur in m ways and a second event can occur in n ways after the first event has occurred, then the two events can occur in $m \times n$ ways. This is also known as the **Fundamental Counting Principle**.

FORMULA FOR PERMUTATIONS OF *N* DISTINCT OBJECTS

Given n distinct objects, the number of ways to select r objects from the set in order is

$$P(n,r) = \frac{n!}{(n-r)!}$$

FORMULA FOR COMBINATIONS OF *N* DISTINCT OBJECTS

Given n distinct objects, the number of ways to select r objects from the set is

$$C(n,r) = \frac{n!}{r!(n-r)!}$$

FORMULA FOR THE NUMBER OF SUBSETS OF A SET

A set containing *n* distinct objects has 2^n subsets.

FORMULA FOR FINDING THE NUMBER OF PERMUTATIONS OF *N* NON-DISTINCT OBJECTS

If there are n elements in a set and r_1 are alike, r_2 are alike, r_3 are alike, and so on through r_k, the number of permutations can be found by

$$\frac{n!}{r_1!r_2!\ldots r_k!}$$

Source: OSCA, 9.6 Binomial Theorem

Pascal's Triangle

	Exponent	Pattern	# of Terms
$(x+y)^1 = x+y$	1	1 + 1	2
$(x+y)^2 = x^2+2xy+y^2$	2	2 + 1	3
$(x+y)^3 = x^3+3x^2y+3xy^2+y^3$	3	3 + 1	4
$(x+y)^4 = x^4+4x^3y+6x^2y^2+4xy^3+y^4$	4	4 + 1	5
	n	*n* + 1	*n* + 1

Exponent sum:	xy^{4+0}	xy^{3+1}	xy^{2+2}	xy^{1+3}	xy^{0+4}
Exponents on *x*:	4	3	2	1	0
Exponents on *y*:	0	1	2	3	4

Probability

Purpose: In cryptography, we will study collisions based on probability.

Source: OSCA, 9.7 Probability

COMPUTING THE PROBABILITY OF AN EVENT WITH EQUALLY LIKELY OUTCOMES

The probability of an event E in an experiment with sample space S with equally likely outcomes is given by

$$P(E) = \frac{\text{number of elements in } E}{\text{number of elements in } S} = \frac{n(E)}{n(S)}$$

E is a subset of S, so it is always true that $0 \leq P(E) \leq 1$.

PROBABILITY OF THE UNION OF TWO EVENTS

The probability of the union of two events E and F (written $E \cup F$) equals the sum of the probability of E and the probability of F minus the probability of E and F occurring together (which is called the **intersection** of E and F and is written as $E \cap F$).

$$P(E \cup F) = P(E) + P(F) - P(E \cap F)$$

PROBABILITY OF THE UNION OF MUTUALLY EXCLUSIVE EVENTS

The probability of the union of two *mutually exclusive* events E and F is given by

$$P(E \cup F) = P(E) + P(F)$$

THE COMPLEMENT RULE

The probability that the **complement of an event** will occur is given by

$$P(E') = 1 - P(E)$$

The following topics are not found in the OSCA textbook, but are mathematical topics covered in Computer Science classes that you will need to know before your first class.

Binary and Hexadecimal Numbers

Purpose: In cybersecurity, binary numbers and hexadecimal numbers are heavily used. Be sure you know them well!

Decimal Numbers – base 10

We are used to numbers in base 10 aka decimal numbers, which uses are familiar digits of 0, 1, 2, 3 ,4, 5, 6, 7, 8, 9. Other systems we need to be familiar with are binary numbers and hexadecimal numbers.

Binary Numbers – base 2

Numbers in base 2 are called binary numbers. They consist of **bits**, short for **binary digits**, which are limited to the digits 0 and 1.

Byte

A byte is our typical grouping of 8 bits into 1 byte.

Nibble (sometimes "Nybble")

A nibble is a less common grouping of 4 bits into 1 nibble. 1 nibble can be expresses as 1 digit in hexadecimal numbers, so it's most common usage is for hexadecimal numbers.

Powers of 2

Powers of 2 are heavily used in binary numbers and in cybersecurity:

2^{12}	2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
4096	2048	1024	512	256	128	64	32	16	8	4	2	1

Representation of a Binary Number

We will use the powers of 2 that we just covered to help us understand the representation of a binary number.

Least Significant Bit (LSB)	the right most bit with value 2^0 (which is always 1)
Most Significant Bit (MSB)	the left most bit with value 2^{n-1} where n is the number of bits

Example for an 8 bit byte, here are the powers of 2 representation:

MSB							**LSB**
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Converting a Binary to Decimal

To convert a binary to a decimal:

- use our powers of 2
- start our counter at 0
- loop through bits from MSB to LSB
    - If a bit is 1, add the power of 2 to our counter
    - If a bit is 0, do nothing

Example for an 8 bit byte, convert 11010101 to decimal. The following chart shows the powers of 2 and the bit pattern:

MSB							**LSB**
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1
1	1	0	1	0	1	0	1

Adding the powers of 2 where the bit is set to 1:

128 + 64 + 16 + 4 + 1 = 213

Converting a Decimal to Binary

Division by 2 Method – using integer division, divide the decimal number by 2. The "div" will be the next integer and the "mod" will be the binary digit, starting with LSB. Stop when the div is 0 in which the mod will be the MSB.

Example: convert 213 from decimal to binary

x	x div 2 = next x	x mod 2	
213	106	1	**LSB**
106	53	0	
53	26	1	
26	13	0	
13	6	1	
6	3	0	
3	1	1	
1	0	1	**MSB**

213 = 11010101

Bitwise Operations

AND	equals 1 if both bits are 1, else 0
OR	equals 1 if ether bit is 1 or both bits are 1, else 0
XOR (Exclusive OR)	equals 1 if 1 bit is 1 and the other bit is 0, 0 otherwise

x	0	0	1	1
y	0	1	0	1
x AND y	0	0	0	1
x OR y	0	1	1	1
x XOR y	0	1	1	0

X-bits for Keys and Block Sizes

In cryptography, we typically speak in terms of bits for keys and block sizes. Here are some common ones (compare this to our powers of 2):

bits	4096	2048	1024	512	256	128	64	32
bytes	512	256	128	64	32	16	8	4

Hexadecimal Numbers – base 16

Numbers in base 16 are called hexadecimal numbers. They consist of the familiar digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, plus the letters A = 10, B = 11, C = 12, D = 13, E = 14, F = 15. As mentioned before, it's common to represent a nibble (4 bits) using a hexadecimal digit. A byte can be expressed as two hexadecimal digits, often called a "nibble pair".

Decimal	Hexadecimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Primes

Purpose: In cybersecurity, prime numbers and relatively prime numbers are heavily used. Be sure you know them well!

Divisor – an integer evenly divides another integer with no remainder
(Note: the term divisor can also be used for any number being divided by, however in cryptography, we use the term divisor to mean an even divisor as stated above).

Divides – we say that a divisor divides the other integer
Example: 5 divides 10 (written as 5 | 10) because 10 / 5 = 2 R 0

Prime Number

- Greater than 1
  - 1 is NOT prime
- Only divisors are 1 and itself
- 2 is prime
  - 2 is the only even prime number
- Checking if a number is prime is computationally easy.

Fundamental Theorem of Arithmetic (FTA)

- (some call it the Unique Factorization Theorem
  or the Unique Prime Factorization Theorem)
- Every integer greater than 1 is either prime or is the product of a unique combination of prime numbers

Greatest Common Divisor (gcd)

- (some call it Greatest Common Factor) – the greatest or largest integer that evenly divides two integers.
- Finding gcd is computationally easy (Euclidean gcd algorithm)

Relatively Prime Numbers (aka co-primes)

- gcd = 1
- Relatively prime numbers do not have to be individually prime.
  - Example: 4 and 15 are relatively prime. 4 is NOT prime. 15 is NOT prime.
- Checking if two numbers are relatively prime is computationally easy because we can use the Euclidean gcd algorithm mentioned above to see if gcd equal 1

Product of two prime numbers

- is not prime
- factors are the two prime numbers and 1
- computationally easy to multiply two large prime numbers
- computationally hard to factor the product of two large prime numbers
    - computationally intractable to factor large prime numbers begins around 512 bits for the product but changes as computing power grows. Most encryption schemes use 1024 or 2048 or 4096 bits for the product.
- A common misconception is that all of cryptography is based on factoring a large prime number. Factoring a prime number is obviously trivial: the prime number and 1. Some (not all) asymmetrical cryptography is directly based on factoring the product of two large prime numbers. Other asymmetrical cryptography is based on the difficulty of finding quadratic residues in modulo of a product of primes, discrete logarithms in modulo prime, elliptic curve discrete logarithms in modulo prime, etc.

Modular Arithmetic

Purpose: In cybersecurity, modular arithmetic is heavily used. Be sure you know it well!

Integer division

- if x / n = a R b where x is positive
  - equivalent to
    - x (div n) = a
    - x (mod n) = b
  - n is the modulus, often said "in modulo n"
  - n must always be positive
  - anything (mod n) will only yield positive integers in the set { 0, …, n – 1 }

- - x (mod n)
  - Note that x is negative
  - Remember n must always be positive
  - Start with -x
  - Add n to -x
  - If the result is positive, that is the answer
  - If the result is still negative, keep adding n until you get a positive answer, which will always be in the set { 0, …, n – 1}

Examples:

```
-9 (mod 3) = 0          -3 (mod 3) = 0          3 (mod 3) = 0
-8 (mod 3) = 1          -2 (mod 3) = 1          4 (mod 3) = 1
-7 (mod 3) = 2          -1 (mod 3) = 2          5 (mod 3) = 2
-6 (mod 3) = 0           0 (mod 3) = 0          6 (mod 3) = 0
-5 (mod 3) = 1           1 (mod 3) = 1          7 (mod 3) = 1
-4 (mod 3) = 2           2 (mod 3) = 2          8 (mod 3) = 2
```

Note the "modulo 3" periodic pattern of residues: 0, 1, 2, 0, 1, 2, …

Many Calculators Give the Wrong Answer for Negative Numbers in modulo!

Try -8 (mod 3) on a Microsoft Windows calculator or Mac calculator. It gives the wrong answer.

Congruence in Modulo

The 3 line equal sign means congruence. Even though the (mod n) is written at the end of the statement on the right side, it applies to both sides of the congruence.

$$a \equiv b \ (mod\ n)$$

is equivalent to the following statements:

"a is congruent to b in modulo n"
"a and b have the same remainder when divided by n"

a (mod n) = b (mod n)

Examples:

$$-8 \equiv 7 \ (mod\ 5)$$
-8 (mod 5) = 2
7(mod 5) = 2

$$-231{,}528{,}752 \equiv 1{,}228{,}413 \ (mod\ 5)$$
-231,528,752 (mod 5) = 3
1,228,413 (mod 5) = 3

We will be using the following style of modulo reductions quite a bit. Please understand the logic of the following:

in the special case of:
if c < n, then for any positive integer k,
kn + c (mod n) =
kn (mod n) + c (mod n) =
0 + c (mod n)
= c

Note: in modular arithmetic, kn + b (mod n) is the same as writing (kn + b) (mod n) the (mod n) is assumes to apply to the entire expression and does not need to be grouped

Algorithms, Computer Programs, Solvers

Purpose: In cybersecurity, we need to understand the difference between an algorithm, a computer program, and a solver.

Algorithm – unambiguous specification of how to solve a class of problems. Not specific to any computer programming language.

Computer Program – specific code written in a computer programming language (or languages) and compiled on a computer to implement an algorithm

Solver – a computer program to solve a specific mathematical problem.

Examples: A solver to find the roots to a quadratic equation if they exist
A solver to find the inverse of a matrix
A solver to find the quadratic residues in modulo prime if they exist
A solver to find the solution to a system of linear equations

Analysis of Algorithms

Purpose: In cybersecurity, we need to understand how to analyze algorithms to help us understand computational intractability. We will need to be able to do formal proofs that demonstrate computational complexity of an algorithm to understand how strong the algorithm is.

Analysis of Algorithms – determining the computational complexity of an algorithms

Computational Complexity – classifying algorithms according to their inherent difficulty expressed in the Big O Notation (possibly additionally in the Big Theta Notation, and/or the Big Omega Notation).

Big O Notation – "order" - growth rate – as the size of data grows how does it affect the run time of the algorithm – an upper bound – worst case

$O(1)$	Constant	Takes the same amount of time regardless of the number of records
$O(n)$	Linear	If 100 records takes 1 minute then 200 records takes 2 minutes
$O(n^2)$	Quadratic (exponential)	If 100 records takes 1 minute then 200 records takes 4 minutes
$O(n^x)$	Polynomial (exponential)	If 100 records takes 1 minute then 200 records takes 2^x minutes
$O(\log n)$ means base 2: $O(\log_2 n)$	Logarithmic (inverse of powers of 2)	If 100 records takes 1 minute, then 200 records takes 1 minute and 9 seconds (worse than linear, but much better than exponential)

Similar Notations:

Big Theta Notation – similar to Big O Notation, but sets both a lower and upper bound, shows both the best and worst case.

Big Omega Notation – similar to Big O Notation, but the lower bound, shows the best case.

Examples of Analysis of Algorithms

$O(1)$

Given an array x of size n (zero based with n > 5):

```
y = (x[1] * 5) + ((x[4] – 12) * 6)
```

$O(n)$

```
for i in 1 to n do
     calculations
```

$O(n^2)$

```
for i in 1 to n do
     for j in 1 to n do
          calculations
```

$O(n^3)$

```
for i in 1 to n do
     for j in 1 to n do
          for k in 1 to n do
               calculations
```

$O(\log n)$

Given an array x of size n sorted in order.
We can perform a binary search.
Each iteration cuts our search space in half: n/2, n/4, n/8, n/16, n/32, etc.
This is the inverse of powers of 2, so it a log base 2.

(Remember in theoretical computer science, since so much is based on powers of 2 and its logarithmic inverse, we assume log means log base 2.)