Introduction to Information Security

Christian J. Rudder

November 2024

Contents

This page is left intentionally blank.

Big thanks to **Professor Sharon Goldberg**
for teaching CS357 (Introduction to Information Security)
at Boston University.

The following five sections are pre-requisites from Concise Work Section Modules [81].
Available at: https://github.com/Concise-Works/sect-modules.
These are not needed, but are recommended for a better understanding of the material.

— 1 —

Prerquesites

1.1 Asymptotic Notation

Asymptotic analysis is a method for describing the limiting behavior of functions as inputs grow infinitely.

Definition 1.1: Asymptotic

Let $f(n)$ and $g(n)$ be two functions. As n grows, if $f(n)$ grows closer to $g(n)$ never reaching, we say that "$f(n)$ is **asymptotic** to $g(n)$."

We call the point where $f(n)$ starts behaving similarly to $g(n)$ the **threshold** n_0. After this point n_0, $f(n)$ follows the same general path as $g(n)$.

Definition 1.2: Big-O: (Upper Bound)

Let f and g be functions. $f(n)$ our function of interest, and $g(n)$ our function of comparison.

Then we say $f(n) = O(g(n))$, "$f(n)$ **is big-O of** $g(n)$," if $f(n)$ grows no faster than $g(n)$, up to a constant factor. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq f(n) \leq c \cdot g(n)$$

Represented as the ratio $\dfrac{f(n)}{g(n)} \leq c$ for all $n \geq n_0$. Analytically we write,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty$$

Meaning, as we chase infinity, our numerator grows slower than the denominator, bounded, never reaching infinity.

Examples:

(i.) $3n^2 + 2n + 1 = O(n^2)$

(ii.) $n^{100} = O(2^n)$

(iii.) $\log n = O(\sqrt{n})$

Proof 1.1: $\log n = O(\sqrt{n})$

We setup our ratio:

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}}$$

Since $\log n$ and $\sqrt{n}$ grow infinitely without bound, they are of indeterminate form $\frac{\infty}{\infty}$. We apply L'Hopital's Rule, which states that taking derivatives of the numerator and denominator will yield an evaluateable limit:

$$\lim_{n\to\infty} \frac{\log n}{\sqrt{n}} = \lim_{n\to\infty} \frac{\frac{d}{dn}\log n}{\frac{d}{dn}\sqrt{n}}$$

Yielding derivatives, $\log n = \frac{1}{n}$ and $\sqrt{n} = \frac{1}{2\sqrt{n}}$. We substitute these back into our limit:

$$\lim_{n\to\infty} \frac{\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = \lim_{n\to\infty} \frac{2\sqrt{n}}{n} = \lim_{n\to\infty} \frac{2}{\sqrt{n}} = 0$$

Our limit approaches 0, as we have a constant factor in the numerator, and a growing denominator. Thus, $\log n = O(\sqrt{n})$, as $0 < \infty$. ■

Definition 1.3: Big-Ω: (Lower Bound)

The symbol Ω reads "Omega." Let f and g be functions. Then $f(n) = \Omega(g(n))$ if $f(n)$ grows no slower than $g(n)$, up to a constant factor. I.e., lower bounded by $g(n)$. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq c \cdot g(n) \leq f(n)$$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)}$$

Meaning, as we chase infinity, our numerator grows faster than the denominator, approaching 0 asymptotically.

Examples: $n! = \Omega(2^n)$; $\frac{n}{100} = \Omega(n)$; $n^{3/2} = \Omega(\sqrt{n})$; $\sqrt{n} = \Omega(\log n)$

Definition 1.4: Big Θ: (Tight Bound)

The symbol Θ reads "Theta." Let f and g be functions. Then $f(n) = \Theta(g(n))$ if $f(n)$ grows at the same rate as $g(n)$, up to a constant factor. I.e., $f(n)$ is both upper and lower bounded by $g(n)$. Let n_0 be our asymptotic threshold, and $c_1 > 0, c_2 > 0$ be some constants. Then, for all $n \geq n_0$,

$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

$$0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty$$

Meaning, as we chase infinity, our numerator grows at the same rate as the denominator.

Examples: $n^2 = \Theta(n^2)$; $2n^3 + 2n = \Theta(n^3)$; $\log n + \sqrt{n} = \Theta(\sqrt{n})$.

Definition 1.5: Little o: (Strict Upper Bound)

The symbol o reads "little-o." Let f and g be functions. Then $f(n) = o(g(n))$ if $f(n)$ grows strictly slower than $g(n)$, meaning $f(n)$ becomes insignificant compared to $g(n)$ as n grows large. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq f(n) < c \cdot g(n)$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

Meaning, as we chase infinity, the ratio of $f(n)$ to $g(n)$ shrinks to zero.

Examples: $n = o(n^2)$; $\log n = o(n)$; $n^{0.5} = o(n)$.

Definition 1.6: Little ω: (Strict Lower Bound)

The symbol ω reads "little-omega." Let f and g be functions. Then $f(n) = \omega(g(n))$ if $f(n)$ grows strictly faster than $g(n)$, meaning $g(n)$ becomes insignificant compared to $f(n)$ as n grows large. Let n_0 be our asymptotic threshold. Then, for all $n \geq n_0$,

$$0 \leq c \cdot g(n) < f(n)$$

$$\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$$

Meaning, as we chase infinity, the ratio of $g(n)$ to $f(n)$ shrinks to zero.

Examples: $n^2 = \omega(n)$; $n = \omega(\log n)$.

Definition 1.7: Asymptotic Equality ($\sim$)

The symbol $\sim$ reads "asymptotic equality." Let f and g be functions. Then $f(n) \sim g(n)$ if, as $n \to \infty$, the ratio of $f(n)$ to $g(n)$ approaches 1. I.e., the two functions grow at the same rate asymptotically. Formally,

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$$

Meaning, as n grows large, the two functions become approximately equal.

Examples: $n + 100 \sim n$, $\log(n^2) \sim 2\log n$.

Tip: To review:

- **Big-O:** $f(n) < g(n)$ (Upper Bound); $f(n)$ grows no faster than $g(n)$.
- **Big-Ω:** $f(n) > g(n)$ (Lower Bound); $f(n)$ grows no slower than $g(n)$.
- **Big-Θ:** $f(n) = g(n)$ (Tight Bound); $f(n)$ grows at the same rate as $g(n)$.
- **Little-o:** $f(n) < g(n)$ (Strict Upper Bound); $f(n)$ grows strictly slower than $g(n)$.
- **Little-ω:** $f(n) > g(n)$ (Strict Lower Bound); $f(n)$ grows strictly faster than $g(n)$.
- **Asymptotic Equality:** $f(n) \sim g(n)$; $f(n)$ grows at the same rate as $g(n)$.

Theorem 1.1: Types of Asymptotic Behavior

The following are common relationships between different types of functions and their asymptotic growth rates:

- **Polynomials.** Let $f(n) = a_0 + a_1 n + \cdots + a_d n^d$ with $a_d > 0$. Then, $f(n)$ is $\Theta(n^d)$.
  E.e., $3n^2 + 2n + 1$ is $\Theta(n^2)$.
- **Logarithms.** $\Theta(\log_a n)$ is $\Theta(\log_b n)$ for any constants $a, b > 0$. That is, logarithmic functions in different bases have the same growth rate.
  E.g., $\log_2 n$ is $\Theta(\log_3 n)$.
- **Logarithms and Polynomials.** For every $d > 0$, $\log n$ is $O(n^d)$. This indicates that logarithms grow slower than any polynomial.
  E.g., $\log n$ is $O(n^2)$.
- **Exponentials and Polynomials.** For every $r > 1$ and every $d > 0$, n^d is $O(r^n)$. This means that exponentials grow faster than any polynomial.
  E.e., n^2 is $O(2^n)$.

1.2 Evaluating Algorithms

When analyzing algorithms, we are interested in two primary factors: time and space complexity.

Definition 2.1: Time Complexity

The **time complexity** of an algorithm is the amount of time it takes to run as a function of the input size. We use asymptotic notation to describe the time complexity of an algorithm.

Definition 2.2: Space Complexity

The **space complexity** of an algorithm is the amount of memory it uses to store inputs and subsequent variables during the algorithm's execution. We use asymptotic notation to describe the space complexity of an algorithm.

Below is an example of a function and its time and space complexity analysis.

Function 2.1: Arithmetic Series - `Fun1`(A)

Computes a result based on a length-n array of integers:

Input: A length-n array of integers.
Output: An integer p computed from the array elements.

```
1 Function Fun1(A):
2 |  p ← 0;
3 |  for i ← 1 to n − 1 do
4 |  |  for j ← i + 1 to n do
5 |  |  |  p ← p + A[i] · A[j];
```

Time Complexity: For $f(n) :=$ `Fun1`(A), $f(n) = \frac{n^2}{2} = O(n^2)$. This is because the function has a nested loop structure, where the inner for-loop runs $n - i$ times, and the outer for-loop runs $n - 1$ times. Thus, the total number of iterations is $\sum_{i=1}^{n-1} n - i = \frac{n^2}{2}$.

Space Complexity: We yield $O(n)$ for storing an array of length n. The variable p is $O(1)$ (constant), as it is a single integer. Hence, $f(n) = n + 1 = O(n)$.

Additional Example: Let $f(n, m) = n^2m + m^3 + nm^3$. Then, $f(n, m) = O(n^2m + m^3)$. This is because both n and m must be accounted for. Our largest n term is n^2m, and our largest m term is m^3 both dominate the expression. Thus, $f(n, m) = O(n^2m + m^3)$.

1.3 Computers & Number Base Systems

Definition 3.1: Turing Machine

A **Turing Machine** is a theoretical computational model used to describe the capabilities of a general-purpose **computer**. It consists of an infinite tape (memory) and a read/write head processing symbols on the tape, one at a time, according to a set of predefined rules. The machine moves left or right, reading or writing symbols, and changing states based on what it reads.

The machine **halts** once it reaches a final state or continues indefinitely. Serving as a flexible, **higher-order function** (a function which receives functions).

Definition 3.2: Von Neumann Architecture

Modern computers operate on a model known as the **Von Neumann architecture**, which consists of three primary components:

1. **Memory**: Stores data and instructions as sequences of bits.
2. **Arithmetic and Logic Unit (ALU)**: Executes operations such as addition, subtraction, multiplication, and division on numbers stored in memory.
3. **Control Unit**: Directs the execution of instructions and manages the flow of data between memory and the ALU.

Where numbers are stored in memory cells, each cell holding an integer value represented in a fixed base, typically $B = 2$, meaning **binary**. Where each digit is less than the base B. We represent integers in memory as:

$$a = \sum_{i=0}^{k-1} a_i B^i$$

where a_i represents the individual digits, and B is the base. For large integers, computations may require manipulating several memory cells to store the full number.

Example: Consider the integer $a = 13$, and let us represent it in base $B = 2$ (binary). We can express this number as a sum of powers of 2, corresponding to the binary representation of 13:

$$a = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 4 + 0 + 1 = 13$$

In binary, this is represented as the sequence of digits: $a = (1101)_2$. Here, the coefficients $a_3 = 1$, $a_2 = 1$, $a_1 = 0$, and $a_0 = 1$ correspond to the binary digits of 13.
Similarly, if we want to represent $a = 45$ in base $B = 10$ **(decimal)**, we write:

$$a = 4 \cdot 10^1 + 5 \cdot 10^0 = 40 + 5 = 45$$

In this case, the coefficients $a_1 = 4$ and $a_0 = 5$ correspond to the decimal digits of 45.

Definition 3.3: Hexadecimal

Hexadecimal base $B = 16$, using digits 0-9 and the letters A-F, where $A = 10$, $B = 11$, $C = 12$, $D = 13$, $E = 14$, and $F = 15$. Hexadecimal is commonly used in computing due to its compact representation of binary data. For example, a **byte** (8 bits) can be represented as two hexadecimal digits, simplifying the display of binary data.

Theorem 3.1: Base $2 \leftrightarrow 16$ Conversion

Let bases $B := 2$ (binary) and $H := 16$ (hexadecimal). At a high-level:

Binary to Hexadecimal:

1. Group B digits in sets of 4, right to left. **Pad** leftmost group with 0's if necessary for a full group.
2. Compute each group, replacing the result with their H digit.
3. Finally, combine each H group.

Hexadecimal to Binary:

1. Convert each H digit into a 4 bit B group.
2. Finally, combine all B groups.

Additionally we may also trim any leading 0's.

Example:

- Binary to Hexadecimal:

$$101101111010_2 \quad \Rightarrow \quad \text{Group as ([1011] [0111] [1010)} \quad \Rightarrow \quad B7A_{16}$$

- Hexadecimal to Binary:

$$3F5_{16} \quad \Rightarrow \quad [0011]\ [1111]\ [0101]_2 \Rightarrow \quad 1111110101_2$$

The following definition is for completeness: applications of such a base are currently uncommon.

Definition 3.4: Unary

Unary, base $B = 1$. A system where each number is represented by a sequence of B symbols. Where the number n is represented by n symbols. Often used in theoretical computer science to prove the existence of computable functions.

Example: The number 5 in unary is represented by 5 symbols: $5 = \texttt{IIIII}$ or $2 = \texttt{II}$. There is no concept of 0, the absence of symbols represents 0.

Definition 3.5: Most & Least Significant Bit

In a binary number, the **most significant bit (MSB)** is the leftmost bit. The **least significant bit (LSB)** is the rightmost bit.

Example: Consider this byte (8 bits), $[1111\ 1110]_2$, the MSB = 1 and the LSB = 0.

Theorem 3.2: Adding Binary

We may use the add and carry method alike decimal addition: **Binary Addition Rules:**

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$ (add 1 to the next digit (left))

We call the last step, **carry**, as we carry our overflow to the next digit.

Example: Adding $0010\ 0011\ 0100_2$ and 0100_2:

$$\begin{array}{r} \overset{1}{} \\ 1\,1000\,0100 \\ +\,0\,0001\,0100 \\ \hline 1\,1001\,1000 \end{array}$$

Where $[1\ 1000\ 0100]_2 + [0\ 0001\ 0100]_2 = [1\ 1001\ 1000]_2$.

Theorem 3.3: Signed Binary Numbers - Two's Complement

In a **two's complement system**, an n-bit signed (positive or negative) binary number can represent values in the range $[-2^{n-1}, 2^{n-1} - 1]$. Then by most significant bit (MSB):

- If MSB is 0, the number is positive;
- If MSB is 1, the number is negative.

Conversion to Two's Complement :

1. Take an unsigned binary number and invert all bits, turning 0's to 1's and 1's to 0's.
2. Finally add 1 to the least significant bit.

Example: Converting -5 into a 4-bit two's complement:

$$\begin{aligned} 5 \to 0101 \quad & \text{(binary for 5)} \\ 1010 \quad & \text{(inverted)} \\ 1011 \quad & \text{(add 1)} \end{aligned}$$

Thus, -5 is represented as 1011 in 4-bits under two's complement.

1.4 Computing Large Numbers

In this section we discuss algorithms for computing large numbers, but first we define algorithmically, addition, subtraction, multiplication, division, and modula.

Definition 4.1: Wordsize

Our machine has a fixed **wordsize**, which is how much each memory cell can hold. Systems like 32-bit or 64-bit can hold 2^{32} (≈ 4.3 billion) or 2^{64} (≈ 18.4 quintillion) bits respectively.

We say the ALU can performs arithmetic operations at $O(1)$ time, within wordsize. Operations beyond this size we deem **large numbers**.

The game we play in the following algorithms is to compute large integers without exceeding wordsize. Moving forward, we assume our machine is a typical 64-bit system.

Function 4.1: Length of digits - $\|a\|$

We will use the notation $\|a\|$ to denote the number of digits in the integer a. For example, $\|123\| = 3$ and $\|0\| = 1$.

Definition 4.2: Computer Integer Division

Our ALU only returns the quotient after division. We denote the quotient as $\lfloor a/b \rfloor : a, b \in \mathbb{Z}$.

Our first hurdle is long division as , which will set up long addition and subtraction for success.
Scenario - Grade School Long Division: Goes as follows, take $\frac{a}{b}$. Find how times b fits into a evenly, q times. Then $a - bq$ is our remainder r.
Examples: let $a = \{12, 5, 17, 40, 89\}$, $b = \{4, 2, 3, 9, 10\}$ respectively, and base $B = 10$,

$$\begin{array}{lrlrlrlrl} (1.) & \begin{array}{r} 3 \\ 4\overline{)12} \\ 12 \\ \hline 0 \end{array} & (2.) & \begin{array}{r} 2 \\ 2\overline{)5} \\ 4 \\ \hline 1 \end{array} & (3.) & \begin{array}{r} 5 \\ 3\overline{)17} \\ 15 \\ \hline 2 \end{array} & (4.) & \begin{array}{r} 4 \\ 9\overline{)40} \\ 36 \\ \hline 4 \end{array} & (5.) & \begin{array}{r} 8 \\ 10\overline{)89} \\ 80 \\ \hline 9 \end{array} \end{array}$$

Take (3.), $a = 17$, $b = 3$: 3 fits into 17 five times, which is 15. 17 take away 15 is 2, our remainder. We create an algorithm to compute this process.

Key Observation: Consider the following powers of 2 of form $x = 2^n + s$, where $x, n, s \in \mathbb{Z}$:

$$\begin{aligned} 3 = 2 + 1 &= 0000\ 0011_2 \quad (1) \\ 6 = 4 + 2 &= 0000\ 0110_2 \quad (2) \\ 12 = 8 + 4 &= 0000\ 1100_2 \quad (3) \\ 24 = 16 + 8 &= 0001\ 1000_2 \quad (4) \\ 48 = 32 + 16 &= 0011\ 0000_2 \quad (5) \\ 96 = 64 + 32 &= 0110\ 0000_2 \quad (6) \\ 192 = 128 + 64 &= 1100\ 0000_2 \quad (7) \end{aligned}$$

Notice that as we increase the power of 2, the number of bits shift left towards a higher-order bit. Now, instead of calculating powers of 2, we shift bits left or right, to yield instantaneous results.

Theorem 4.1: Binary Bit Shifting (Powers of 2)

Let x be a binary unsigned integer. Where "$\ll$" and "$\gg$" are left and right bit shifts:
Left Shift by k bits: $x \ll k := x \cdot 2^k$
Right Shift by k bits: $x \gg k = \lfloor x/2^k \rfloor$
Remainder: bits pushed out after right shift(s).

Example: Observe, $16 = 10000$ (4 zeros), $8 = 1000$ (3 zeros), we shift by 4 and 3 respectively:

- Instead of $3 \cdot 16$ in base 10, we can $3 \ll 4 = 48$, as $3 \cdot 2^4 = 48$.
- Conversely, Instead of $48/16$ in base 10, $48 \gg 4 = 3$, as $\lfloor 48/2^4 \rfloor = 3$.
- Catching the remainder: say we have $37/8$ base 10, then,

$$37 = 100101_2 \quad \text{and} \quad 8 = 1000_2 \text{ then } 37 \gg 3 = 4 \text{ remainder } 5,$$

as $[100101] \gg 3 = [000100]101$, where 101_2 is our remainder 5_{10}.

Function 4.2: Division with Remainder in Binary (Outline) - *QuoRem()*

For binary integers, let dividend $a = (a_{k-1} \cdots a_0)_2$ and divisor $b = (b_{\ell-1} \cdots b_0)_2$ be unsigned, with $k \geq 1$, $\ell \geq 1$, ensuring $0 \leq b \leq a$, and $b_{\ell-1} \neq 0$, ensuring $b > 0$.

We compute q and r such that, $a = bq + r$ and $0 \leq r < b$. Assume $k \geq \ell$; otherwise, $a < b$. We set $q \leftarrow 0$ and $r \leftarrow a$. Then quotient $q = (q_{m-1} \cdots q_0)_2$ where $m := k - \ell + 1$.

Input: a, b (binary integers)
Output: q, r (quotient and remainder in binary)

```
1 Function QuoRem(a, b):
2     r ← a;
3     q ← {0_{m-1} ··· 0};
4     for i ← ||a|| − ||b|| − 1 down to 0 do
5         q_i ← ⌊ r / (b ≪ i) ⌋;
6         r ← r − (q_i · (b ≪ i));
```

Time Complexity: $O(\|a\|(\|a\| - \|b\|))$. In short, line 5 we preform division on $\|a\|$ bits of decreasing size. Though **not totally necessary**, For more detail visit https://shoup.net/ntb/ntb-v2.pdf on page 60. General n cases can be found in Theorem (4.2).

Example Let $a = 47_{10} = 101111_2$ and $b = 5_{10} = 101_2$, we run *QuoRem(a, b)*. We summarize the above example as, "How many times does 101_2 fit into 101111_2?"

1. Does $5 \ll 3$ fit into 101000_2? It fits! $q = 1000_2$, $r = 101111_2 - 101000_2$.
2. Does $5 \ll 2$ fit into 0111_2? no fits! $q = 1000_2$, $r = 0111_2$.
3. Does $5 \ll 1$ fit into 0111_2? no fits! $q = 1000_2$, $r = 0111_2$.
4. Does $5 \ll 0$ fit into 0111_2? It fits! $q = 1001_2$, $r = 0111_2 - 0101$.
5. Return $q = 1001_2 = 9_{10}$, $r = 0010 = 2_{10}$

Long addition: We craft an algorithm for grade school long addition, which goes as follows:

$$\begin{array}{r} \overset{1}{2}5\,\overset{1}{3}08 \\ +\,39\,406 \\ \hline 64\,714 \end{array}$$

Where adding, $25,308 + 39,406 = 64,714$. We create an algorithm to compute this in the following function.

Here the function *QuoRem* (4.2) to return (quotient, remainder) preforms in $O(1)$ as bits are small enough for word size.

Function 4.3: Addition of Binary Integers - *Add()*

Let $a = (a_{k-1} \cdots a_0)_2$ and $b = (b_{\ell-1} \cdots b_0)_2$ be unsigned binary integers, where $k \geq \ell \geq 1$. We compute $c := a + b$ where the result $c = (c_k c_{k-1} \cdots c_0)_2$ is of length $k + 1$, assuming $k \geq \ell$. If $k < \ell$, swap a and b. This algorithm computes the binary representation of $a + b$.

Input: a, b (binary integers)
Output: $c = (c_k \cdots c_0)_2$ (sum of $a + b$)

```
1  Function Add(a, b):
2      carry ← 0;
3      for i ← 0 to ℓ − 1 do
4          tmp ← a_i + b_i + carry;
5          (carry, c_i) ← QuoRem(tmp, 2);
6      for i ← ℓ to k − 1 do
7          tmp ← a_i + carry;
8          (carry, c_i) ← QuoRem(tmp, 2);
9      c_k ← carry;
10     return c = (c_k ··· c_0)_2;
```

Note: $0 \leq carry \leq 1$ and $0 \leq tmp \leq 3$.
Time Complexity: $O(\max(\|a\|, \|b\|))$, as we iterate at most the length of the largest input.
Space Complexity: $O(\|a\| + \|b\|)$, though $c = k + 1$, constants are negligible as $k, \ell \to \infty$.

For subtracting, $5,308 - 3,406 = 1,904$, where we borrow 10 from the 5 to make 13:

$$\begin{array}{r} {\scriptstyle 4\ 10} \\ \not{5}\ 3\,08 \\ -\ 3\ \ 406 \\ \hline 1\ \ 904 \end{array}$$

Function 4.4: Subtraction of Binary Integers - *Subtract()*

Let $a = (a_{k-1} \cdots a_0)_2$ and $b = (b_{\ell-1} \cdots b_0)_2$ be unsigned binary integers, where $k \geq \ell \geq 1$ and $a \geq b$. We compute $c := a - b$ where the result $c = (c_{k-1} \cdots c_0)_2$ is of length k, assuming $a \geq b$. If $a < b$, swap a and b and set a negative flag to indicate the result is negative. This algorithm computes the binary representation of $a - b$.

Input: a, b (binary integers)
Output: $c = (c_{k-1} \cdots c_0)_2$ (difference of $a - b$)

```
1  Function Subtract(a, b):
2      borrow ← 0;
3      for i ← 0 to ℓ − 1 do
4          tmp ← a_i − b_i − borrow;
5          if tmp < 0 then
6              borrow ← 1;
7              c_i ← tmp + 2;
8          else
9              borrow ← 0;
10             c_i ← tmp;
11     for i ← ℓ to k − 1 do
12         tmp ← a_i − borrow;
13         if tmp < 0 then
14             borrow ← 1;
15             c_i ← tmp + 2;
16         else
17             borrow ← 0;
18             c_i ← tmp;
19     return c = (c_{k−1} ··· c_0)_2;
```

Note: $0 \leq borrow \leq 1$. Subtraction may produce a borrow when $a_i < b_i$.
Time Complexity: $O(\max(\|a\|, \|b\|))$, iterating at most the length of the largest input.
Space Complexity: $O(\|a\| + \|b\|)$, as the length of c is at most k, with constants negligible as $k, \ell \to \infty$.

For multiplication, $24 \cdot 16 = 384$:

$$\begin{array}{r} {\scriptstyle 2} \\ 24 \\ \times\ 16 \\ \hline 144 \\ +\,240 \\ \hline 384 \end{array}$$

Where $6 \cdot 4 = 24$, we write the 4 and carry the 2. Then $6 \cdot 2 = 12$ plus the carried 2 is 14. Then we multiply the next digit, 1, we add a 0 below our 144, and repeat the process. Every new 10s place we add a 0. Then we add our two products to get 384.

We create an algorithm to compute this process in the following function:

Function 4.5: Multiplication of Base-B Integers - *Mul()*

Let $a = (a_{k-1} \cdots a_0)_B$ and $b = (b_{\ell-1} \cdots b_0)_B$ be unsigned integers, where $k \geq 1$ and $\ell \geq 1$. The product $c := a \cdot b$ is of the form $(c_{k+\ell-1} \cdots c_0)_B$, and may be computed in time $O(k\ell)$ as follows:

Input: a, b (base-B integers)
Output: $c = (c_{k+\ell-1} \cdots c_0)_B$ (product of $a \cdot b$)

```
1  Function Mul(a, b):
2      for i ← 0 to k + ℓ − 1 do
3          c_i ← 0;
4      for i ← 0 to k − 1 do
5          carry ← 0;
6          for j ← 0 to ℓ − 1 do
7              tmp ← a_i · b_j + c_{i+j} + carry;
8              (carry, c_{i+j}) ← QuoRem(tmp, B);
9          c_{i+ℓ} ← carry;
10     return c = (c_{k+ℓ−1} ··· c_0)_B;
```

Note: At every step, the value of *carry* lies between 0 and $B - 1$, and the value of *tmp* lies between 0 and $B^2 - 1$.
Time Complexity: $O(\|a\| \cdot \|b\|)$, since the algorithm involves $k \cdot \ell$ multiplications.
Space Complexity: $O(\|a\| + \|b\|)$, since we store the digits of a, b, and c.

Function 4.6: Decimal to Binary Conversion - *DecToBin()*

This function converts a decimal number n into its binary equivalent by repeatedly dividing the decimal number by 2 and recording the remainders.

Input: n (a decimal number)
Output: b (binary representation of n)

```
1 Function DecToBin(n):
2     b ← empty string;
3     while n > 0 do
4         r ← n mod 2;
5         n ← ⌊n/2⌋;
6         b ← r + b;
7     return b;
```

Time Complexity: $O(\log n)$, as the number of iterations is proportional to the number of bits in n.
Space Complexity: $O(n)$, storing our input n.

Example: Converting 89 to binary given the above function:

$$\begin{aligned}
89_{10} \div 2 &= 44 \quad \text{rem } 1, \longleftarrow \textbf{ LSB} \\
44_{10} \div 2 &= 22 \quad \text{rem } 0, \\
22_{10} \div 2 &= 11 \quad \text{rem } 0, \\
11_{10} \div 2 &= 5 \quad \text{rem } 1, \\
5_{10} \div 2 &= 2 \quad \text{rem } 1, \\
2_{10} \div 2 &= 1 \quad \text{rem } 0, \\
1_{10} \div 2 &= 0 \quad \text{rem } 1. \longleftarrow \textbf{ MSB}
\end{aligned}$$

Thus, $89_{10} = 1011001_2$.

Theorem 4.2: Time Complexity of Basic Arithmetic Operations

We generalize the time complexity to a and b as n-bit integers.

(i) **Addition & Subtraction:** $a \pm b$ in time $O(n)$.

(ii) **Multiplication:** $a \cdot b$ in time $O(n^2)$.

(iii) **Quotient Remainder** quotient $q := \left\lfloor \frac{a}{b} \right\rfloor : b \neq 0, a > b$; and remainder $r := a \mod b$ has time $O(n^2)$.

1.5 Computational Efficiency

Theorem 5.1: Binary Length of a Number - $\|a\|$

The binary length of an integer a_{10} in binary representation, is given by:

$$\|a\| := \begin{cases} \lfloor \log_2 |a| \rfloor + 1 & \text{if } a \neq 0, \\ 1 & \text{if } a = 0, \end{cases}$$

as $\lfloor \log_2 |a| \rfloor + 1$ correlates to the highest power of 2 required to represent a.

Example: Think about base 10 first. Let there be a 9 digit number $d = 684,301,739$. To reach 9 digits takes 10^8; The exponent plus 1 yields $\|d\|$. Hence, $\lfloor \log_{10} d \rfloor + 1$ is $\|d\|$.

Now, let there be a 7 digit binary number $b = 1001000$, which expanded is:

$$(1 \cdot 2^6) + (0 \cdot 2^5) + (0 \cdot 2^4) + (1 \cdot 2^3) + (0 \cdot 2^2) + (0 \cdot 2^1) + (0 \cdot 2^0) = 72,$$

Taking 6 powers of 2 to reach 72, we add 1 to get $\|b\| = 7$. Hence, $\|b\| = \lfloor \log_2 b \rfloor + 1$.
Additionally, if $a = 0_2$ then $\|a\| = 1$. as $a^0 = 1$.

Theorem 5.2: Splitting Higher and Lower Bits

Let a be a binary number with n bits. We can split a into two numbers A_1 and A_0 with $n/2$ bits each, representing the first and second halves respectively. Where:

$$A_1 := \frac{a}{2^{\lceil n/2 \rceil}} \quad \text{and} \quad A_0 := a \bmod 2^{\lceil n/2 \rceil}$$

Example: Let's start with base 10. To achieve $A_1 = 7455$ and $A_0 = 62,010$, for $a = 745,562,010$. we take the length $\|a\| := \lfloor \log_{10}(745,562,010) \rfloor + 1 = 9$, as $10^8 \leq 745,562,010 < 10^9$. Then:

$$A_1 = \frac{745,562,010}{10^{\lceil 9/2 \rceil}} = 7455, \quad \text{and} \quad A_0 = 745,562,010 \bmod 10^{\lceil 9/2 \rceil} = 62,010$$

as $10^5 \leq 62,010 < 10^6$. Likewise to finding the remainder in base 2, we can use the same bit shifting technique for base 10 (4.1). We see,

$$[745,562,010]_{10} \text{ right shift by } 5, [000,007,455]_{10} \; 62,010.$$

Hence, $62,010$ is pushed out, and our remainder. Then, we can apply the same technique to base 2. Let $a = 1111\ 1111\ 1001\ 1001_2$. We have $\|a\| := 16$, then:

$$A_1 = \frac{1111\ 1111\ 1001\ 1001_2}{2^{\lceil 16/2 \rceil}} = 1111\ 1111_2, \text{ and } A_0 = 1111\ 1111\ 1001\ 1001_2 \bmod 2^{\lceil 16/2 \rceil} = 1001\ 1001_2$$

Scenario - *Divide and Conquer Multiplication*: We are to compute,

$$A_1 2^{\lceil n/2 \rceil} + A_0 =: a \quad \times \quad b := B_1 2^{\lceil n/2 \rceil} + B_0.$$

Then we have,

$$\begin{aligned} a \cdot b &= (A_1 2^{\lceil n/2 \rceil} + A_0)(B_1 2^{\lceil n/2 \rceil} + B_0) \\ &= (A_1 2^{\lceil n/2 \rceil})(B_1 2^{\lceil n/2 \rceil}) + (A_1 2^{\lceil n/2 \rceil})B_0 + (B_1 2^{\lceil n/2 \rceil})A_0 + A_0 B_0 \\ &= (A_1 B_1)2^n + (A_1 B_0 + B_1 A_0)2^{\lceil n/2 \rceil} + A_0 B_0. \end{aligned}$$

We need to compute 4 products, (A_1B_1), $(A_1B_0), (B_1A_0)$, and (A_0B_0). We now attempt to solve them independently:

Function 5.1: Multiplication of n-bit Integers - *Multiply()*

Let a and b be n-bit integers of base 2. This algorithm recursively computes the product of a and b using a straightforward divide-and-conquer approach, without using Karatsuba's optimization.

Input: n, a, b (where a and b are n-bit integers)
Output: The product $a \times b$

```
1  Function Multiply(n, a, b):
2      if n < 2 then
3          return the result of grade-school multiplication for a × b;
4      else
5          A_1 ← a ÷ 2^{n/2};  A_0 ← a mod 2^{n/2};
6          B_1 ← b ÷ 2^{n/2};  B_0 ← b mod 2^{n/2};
7          p_1 ← Multiply(n/2, A_1, B_1);
8          p_2 ← Multiply(n/2, A_1, B_0);
9          p_3 ← Multiply(n/2, A_0, B_1);
10         p_4 ← Multiply(n/2, A_0, B_0);
11         return p_1 · 2^n + (p_2 + p_3) · 2^{n/2} + p_4;
```

Time Complexity: $O(n^2)$, as in our master method $T(n) = 4T(n/2) + O(n)$, Theorem (**??**).
Space Complexity: $O(n)$, storing $n + n$ bits for a and b, while we track $O(\log_2 n)$ depth in the recursion stack.

We appear to make no improvement, however there's a small trick to reduce the number of multiplications. We continue on the next page.

Observe our full term, $c := (A_1B_1)2^n + (A_1B_0 + B_1A_0)2^{\lceil n/2 \rceil} + A_0B_0$. Say we computed another term,

$$z := (A_1 + A_0)(B_1 + B_0) = (A_1B_1) + (A_1B_0) + (B_1A_0) + (A_0B_0).$$

Notice how z also contains (A_1B_1) and (A_0B_0), which are also in c. Say $m = (A_1B_0) + (B_1A_0)$. Let $x := (A_1B_1)$ and $y := (A_0B_0)$ then $z - x - y = m$. This reduces the number of multiplications to 3, as we only compute (A_1B_1), (A_0B_0) once, and then z.

We employ the above strategy, which is **Karatsuba's multiplication algorithm**:

Function 5.2: Karatsuba's Multiplication Algorithm - *KMul()*

Let a and b be n-bit integers of base 2. This algorithm recursively computes the product of a and b using a divide-and-conquer approach.

Input: n, a, b (where a and b are n-bit integers)
Output: The product $a \times b$

```
1  Function Multiply(n, a, b):
2      if n < 2 then
3          return the result of grade-school multiplication for a × b;
4      else
5          A_1 ← a ÷ 2^{n/2};  A_0 ← a mod 2^{n/2};
6          B_1 ← b ÷ 2^{n/2};  B_0 ← b mod 2^{n/2};
7          x ← Multiply(n/2, A_1, B_1);
8          y ← Multiply(n/2, A_0, B_0);
9          z ← Multiply(n/2, A_1 + A_0, B_1 + B_0);
10         return x · 2^n + (z − x − y) · 2^{n/2} + y;
```

Time Complexity: $O(n^{\log_2 3}) \approx O(n^{1.585})$, as in our master method $T(n) = 3T(n/2) + O(n)$, Theorem (**??**).
Space Complexity: $O(n)$.

— 2 —

Networking Fundamentals

2.1 The Internet

Terminology and concepts of the internet, which will be used throughout this text.

Definition 1.1: Protocol

A **protocol** is a set of rules which govern the exchange of data between devices. Protocols define the format, timing, sequencing, and error control of data transmission [66].

Definition 1.2: Internet

The **Internet** is a global network of distributed system communicating over an **Internet Protocol** (IP) [25]. Documents served over the internet are referred to as **webpages** or **websites**.

Definition 1.3: HTTP & HTML

HTTP (HyperText Transfer Protocol), the protocol which transfer data over the internet, distributing **HTML** (HyperText Markup Language) documents. Such documents include **hyperlinks** to other websites, images, and other media [40].

Definition 1.4: RFC (Request for Comments)

RFC (Request for Comments) is a publication from the **Internet Engineering Task Force** (IETF) and the **Internet Society** (ISOC). This body governs the specifications for the internet and its protocols [77].

Definition 1.5: DNS and IP Addresses

An **Internet Protocol address** (IP address) is a unique identifier for a device on a network. The **Domain Name System** (DNS) maps domain names to IP addresses [1].

Definition 1.6: Web Browser

A **web browser** is a software application for accessing the **World Wide Web** (WWW) [79].

Definition 1.7: URL (Uniform Resource Locator)

A **URL** (Uniform Resource Locator) references each webpage, specifying protocol, domain, and path [82]. E.g., `http://www.example.com/path/to/resource`.

- **Protocol**: `http`
- **Domain**: `www.example.com`
- **Path**: `/path/to/resource`

Definition 1.8: Client-Server Model

Most of the internet operates on a **client-server model**, where an agent device–the **client**–requests data from another agent–the **server**–which serves an appropriate response. Clients are not servers and vice versa, as they receive and interpret data differently [22].

Definition 1.9: HTTP Methods

When a client makes a request to a server, they must specify their intent, categorized by **HTTP methods** [38]:

- **GET**: Retrieve data from the server.
- **POST**: Send data to the server.
- **PUT**: Update data on the server.
- **DELETE**: Remove data from the server.

Definition 1.10: HTTP Headers

HTTP headers are key-value pairs sent between the client and server to provide **metadata** about the request or response. **Metadata** is data about the transmitted data, telling the receiver how the incoming data should be interpreted [38].

Tim Berners-Lee and his team at CERN developed the first web server and browser in 1989 [83].

HTTP Version	Description
HTTP/0.9 (1991)	Only supports GET method (retrieving HTML alone).
HTTP/1.0 (1996)	RFC#1945, adding support for metadata in HTTP headers, status codes, and POST and HEAD methods [9].
HTTP/1.1 (1997)	Defined in RFC#2068 and later updated by RFC#2616, introduced persistent connections, chunked transfer encoding, and additional cache control mechanisms [37][38].
HTTP/2 (2015)	RFC#7540, improving performance by enabling request and response multiplexing, header compression, and prioritization [8].
HTTP/3 (2022)	Builds upon HTTP/2's features and uses the QUIC transport protocol to reduce latency and improve security. [11]

Table 2.1: Evolution of HTTP Versions

Note: In short, **Persistent Connections** allow multiple requests and responses to be sent over a single connection, reducing latency and improving performance [38]. **Chunked Transfer Encoding** allows the server to send data in chunks, enabling the client to start processing data before the entire response is received [38]. **Multiplexing**, is the ability to send multiple requests and responses over a single connection, reducing latency and improving performance [35]. **QUIC** will be discussed alter on with other transfer protocols in a later section.

2.2 Data Transmission

This section details how internet traffic is transmitted between devices.

Definition 2.1: ISO Model

The **ISO model** (International Organization for Standardization) is a conceptual framework for transmitted data between devices. It is divided into seven layers of function[15]. Published in 1984 by the International Organization for Standardization (ISO) [49].

Definition 2.2: TCP/IP Model

The **TCP/IP model** (Transmission Control Protocol/Internet Protocol) is a concise representation of the ISO model used in practical settings [84].

Definition 2.3: ISO Layers

1. **Physical**: Converts data into physical signals (e.g., electrical, optical, or radio waves) for transmission across the network medium (e.g., cables, fiber optics, or wireless channels).
2. **Data Link**: local delivery of directly connected devices within the same network.
3. **Network**: Handles addressing, routing, in external networks from source to destination.
4. **Transport**: Ensures end-to-end delivery, via a message delivery protocol.
5. **Session**: Initiates and terminates network connections, ensuring efficient resource usage.
6. **Presentation**: To translate, compress, and encrypt data (e.g., Operating Systems).
7. **Application**: User facing services such as, HTTP , FTP, DNS, SMTP, etc.

[52][69]

Note: Many of the above layers are closely related, if not identical. In practice, layers 5-6 are integrated into layer 7, and layers 1-2 are often combined into a single layer in the TCP/IP model.

Definition 2.4: TCP/IP Layers

1. **Network Interface**: Physical and data link layers from ISO.
2. **Internet**: Attaches IP addresses to data packets for routing across the internet.
3. **Transport**: Defines the delivery protocol, segmenting data into packets.
4. **Application**: The Session, Presentation, and Application layers from ISO.

[69]

Despite the numbering of the layers, the user interacts with the application layer, which communicates down the chain of layers to the physical layer, where the data is transmitted over the network medium. The receiving device then interprets the data, moving back up the chain to the application layer.

To illustrate the contrast between the ISO and TCP/IP models, consider the diagram:

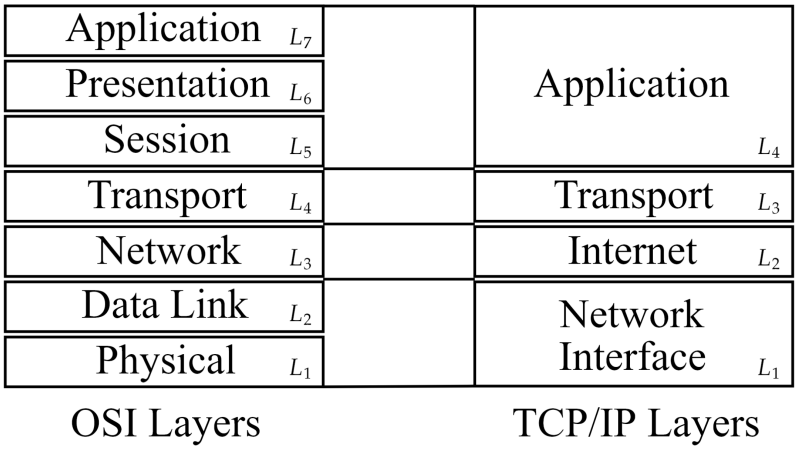


Figure 2.1: ISO vs TCP/IP Model

To illustrate two devices communicating over the internet, consider the diagram:

2.3 Routing Networks

When IP addresses began

Definition 3.1: Routing

Routing is the process of selecting the best path across networks. Data is segmented into packets, each with a destination address. **Routers** are devices which forward this data through the network.

Routers have a **routing table** which maps to other reachable networks. When a packet arrives, the router checks against its routing table to find the best path. [26]

Definition 3.2: Hop-by-Hop & End-to-End Routing

- **Hop-by-Hop Routing**: When a packet of data is forwarded from one router to the next, a forward decision is called a **hop**.
- **End-to-End Routing**: The process of sending data from source to destination without intermediate hops.

It is often rare to see end-to-end routing in modern networks, as data is often forwarded through multiple routers. A target destination may be unreachable from a given router. [44]

Definition 3.3: Router Advertising

When routers inform each other of their existence and the networks they can reach [57].

Definition 3.4: Routing Protocols

- **IP** (Internet Protocol): The primary protocol for routing data across the internet.
- **BGP** (Border Gateway Protocol): The protocol for routing data between **Autonomous Systems** (AS). an AS is a collection of IP networks and routers under the control of a single entity (e.g., an **ISP** (Internet Service Provider)). These may only connect with each other if they have a mutual agreement. ASes identify themselves to external networks using a unique **Autonomous System Number** (ASN). These are unique 16 bit numbers between 1-65534 or 32 bit numbers between 131072-4294967294 (e.g., `AS12345`) [24].
- **OSPF** (Open Shortest Path First): A link-state routing protocol used within an AS. Link-state protocols are a set of algorithms which determine the best path, based on the topology of a network graph [51]. It is also an **IGP** (Interior Gateway Protocol), meaning it operates within a single AS. It does so by sending out **LSAs** (Link State Advertisements) to other routers in the AS. Then routers in the system build a **LSADB** (Link State Advertisement Database) of the network topology. Then a shortest path algorithm is run to determine the best path to each network [13].
- **RIP** (Routing Information Protocol): RIP employs hop count as a routing metric, with a maximum allowable hop count of 15 (network size limitation). It operates as an **IGP** within a single AS, periodically broadcasting the entire routing table to neighboring routers every 30 seconds, which can lead to slower convergence and higher bandwidth usage compared to other protocols. RIP is largely deprecated [48].

Definition 3.5: IP Addressing

IP addresses are unique identifiers for devices on a network. There are two versions of IP addresses, **IPv4** and **IPv6**. IPv4 A 32-bit address (2^{32} addresses), employed since 1983, quickly exhausted all available addresses by the 2010s [53]. IPv6 is a 128-bit address (2^{128} addresses), introduced in 1998 , in an attempt to address this shortage [32][45]. For example,

- **IPv4**: a decimal octet "$x.x.x.x$": $x \in [0, 255]$ (e.g., `192.168.1.1`).
- **IPv6**: a hexadecimal segment "$y:y:y:y:y:y:y:y$": $y \in [0, FFFF]$ (e.g., `2001:0db8:85a3:0000:0000:8a2e:0370:7334`).

Definition 3.6: Subnetting

Instead of a large monolith network of routers, networks can be divided into smaller networks called **subnets**. I.e., Instead of passing data to every device on a network, routers forward data to a representative device on each subnet. [23]

Definition 3.7: Subnet Masking

A **subnet mask** defines which part of an IP address identifies the **network** and which part identifies the **host**.

Definition 3.8: Classful Network

In the beginning, the first octet of an IPv4 address determined the network class—only allowing for 256 networks. The RFC#791 published in 1981 introduced **Classful Networks** [65]. It uses the first three bits of the first octet's binary representation as a subnet mask to determine a class ranging from A-E—D and E were rarely if ever used.

Class	Binary Prefix	Range (Decimal)	Purpose	Details
A	`0xx`	1.0.0.0 to 126.0.0.0	Unicast (large networks)	For large organizations; 8 bits for the network, 24 for hosts.
B	`10x`	128.0.0.0 to 191.255.0.0	Unicast (medium networks)	For medium-sized networks; 16 bits for the network, 16 for hosts.
C	`110`	192.0.0.0 to 223.255.255.0	Unicast (small networks)	For small networks; 24 bits for the network, 8 for hosts.
D	`1110`	224.0.0.0 to 239.255.255.255	Multicast	Reserved for multicast addressing; not for general use.
E	`1111`	240.0.0.0 to 255.255.255.255	Experimental and future use	Reserved for research and development; not assigned for standard use.

Table 2.2: Overview of IPv4 Address Classes

Definition 3.9: Fixed Length Subnet Masking (FLSM)

Fixed Length Subnet Masking (FLSM) is a technique which divides a network into equal-sized subnets. This may lead to inefficient use of IP addresses. [7]

Definition 3.10: Variable Length Subnet Masking (VLSM)

Variable Length Subnet Masking (VLSM) is a technique which allows for the creation of subnets with different sizes. As some ASes may require more IP addresses than others, VLSM allows for more efficient use of IP addresses. [7]

Definition 3.11: Classless Inter-Domain Routing (CIDR)

CIDR, introduced in 1993 through RFC#1518 and RFC#1519 to address IPv4 exhaustion. **CIDR replaced classful subnetting** with VLSM [41].
CIDR denoted as `IP Address/Prefix Length` (e.g., `192.168.1.0/24`), where:

- `IP Address`: Represents the starting address of the network.
- `Prefix Length`: The number of bits used for the **network portion** of the address. E.g.,

`255.0.0.0/8`; `255.255.0.0/16`; `255.255.255.0/24`; `255.255.255.192/26`;

Definition 3.12: Route Aggregation

CIDR introduced **Route Aggregation** also known as **Supernetting**, or **Route Summarization**, is the process of combining multiple routes into a single route advertisement. **Example**: Consider an organization assigned the following contiguous IP address blocks:

`192.168.1.0/24`; `192.168.2.0/24`; `192.168.3.0/24`; `192.168.4.0/24`;

Each block holding 256 IP addresses with a subnet mask of 255.255.255.0, requiring four routing table entries. However, these networks share a common prefix: the first 22 bits (`192.168.0.0/22`), which aggregates to: `192.168.0.0/22` [41].

Theorem 3.1: Routers - Path of Least Resistance

If the target address of a router is `192.168.1.5` and the router has routes, `192.168.0.0/16` vs. `192.168.1.0/24`, the prefix length 24 will be chosen.

Definition 3.13: IP Address Components

Network Address:

- Identifies the specific network segment to which a device is connected.
- Determined by setting all bits in the host portion to 0.
- Example: For the IP address `192.168.1.10` with a subnet mask of `255.255.255.0` (/24), the network address is `192.168.1.0`.

Host Address:

- Uniquely identifies a device within a network segment.
- The bits in the IP address designated for hosts, specified by the subnet mask.
- Example: In the IP address `192.168.1.10` with a /24 subnet mask, the host portion is the last octet (10).

Broadcast Address:

- Used to communicate with all devices on a specific network segment simultaneously.
- Determined by setting all bits in the host portion to 1.
- Example: For the network `192.168.1.0/24`, the broadcast address is `192.168.1.255`.

[76] [68] [54]

Consider the IP address `192.168.2.12/26` and its binary `11000000.10101000.00000010.00001100`:

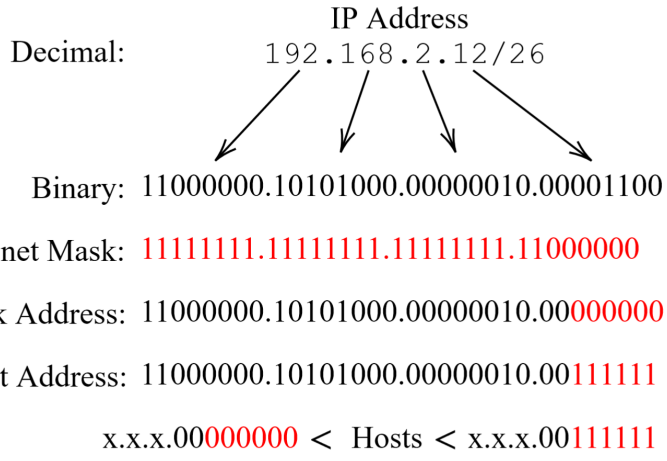


Figure 2.2: Binary Subnetting: Red indicating parts of the IP address identified for each component.

Definition 3.14: Common Types of Area Networks (ANs)

PAN (Personal Area Network): A network for personal devices, such as smartphones, smartwatches, or earbuds, with a short range (typically a few meters) using technologies like Bluetooth or infrared.

LAN (Local Area Network): Connects devices within a small area, such as a home, office, or school, enabling high-speed communication and resource sharing.

WLAN (Wireless Local Area Network): A wireless version of LAN that uses Wi-Fi to connect devices within a localized area like a home or office.

CAN (Campus Area Network): A network that connects multiple LANs across a limited geographical area, such as a university campus or corporate facility, for resource sharing and communication.

MAN (Metropolitan Area Network): Covers a larger area than a LAN, typically a city or metropolitan region, connecting multiple LANs or CANs via high-speed infrastructure like fiber optics.

WAN (Wide Area Network): Connects LANs, MANs, or other networks over large geographical areas, such as countries or continents. The internet is the largest WAN example.

SAN (Storage Area Network): A high-speed network that provides access to storage devices for data centers, ensuring fast and reliable storage management.

EPN (Enterprise Private Network): A private network created by organizations to connect their various locations securely, often including VPNs for remote access.

VPN (Virtual Private Network): Creates a secure, encrypted connection over public networks, like the internet, to allow users to access private networks remotely.

HAN (Home Area Network): A network within a home environment, connecting personal devices like computers, printers, and smart home gadgets.

GAN (Global Area Network): A large-scale network that connects multiple WANs and supports worldwide communication, with the internet as its most prominent example.

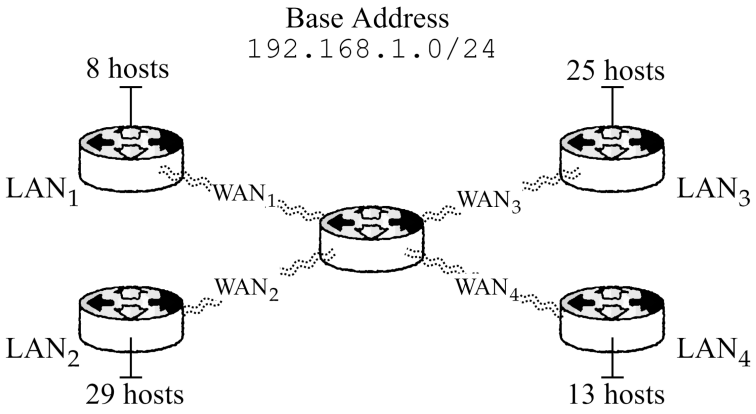
[62]

Example 3.1: Subnetting a Network via VLSM

Consider the FLSM below, which all have 62 hosts per network:

Network Address	Hosts	Broadcast Address
192.168.100.0	.1 – .62	.63
192.168.100.64	.65 – .126	.127
192.168.100.128	.129 – .190	.191
192.168.100.192	.193 – .254	.255

Below illustrates this network, where a router of base address `192.168.1.0/24`, connects four LANs:



Subnetting: Process each LAN from largest to smallest, Select the nearest block size, identify the network, host, and broadcast addresses. Since there is a subnet mask of /24, blocks $[128, 64, 32, 16, 8, 4, 2, 1]$ are available. This is the case as $2^8 = 256$. If a LAN has 129 hosts, that LAN will occupy all 256 addresses.

1. **LAN**$_2$: 29 hosts ⇒ Block size 32. Network: `x.0`, Broadcast: `x.31`, Hosts: `x.1-x.30`.
2. **LAN**$_3$: 25 hosts ⇒ Block size 32. Network: `x.32`, Broadcast: `x.63`, Hosts: `x.33-x.62`.
3. **LAN**$_4$: 13 hosts ⇒ Block size 16. Network: `x.64`, Broadcast: `x.79`, Hosts: `x.65-x.78`.
4. **LAN**$_1$: 8 hosts ⇒ Block size 16. Network: `x.80`, Broadcast: `x.95`, Hosts: `x.81-x.94`.

8 hosts need occupy 16, as a block size of 8 only has 6 usable addresses. The computed subnet now only occupies addresses `x.0-x.95`, leaving room for additional LANs [14].

■

Definition 3.15: Ports

A host may have serve multiple resources, HTTP, FTP, SSH, etc. **Ports** are used to distinguish between these services. Port numbers are managed by the **Internet Assigned Numbers Authority (IANA)**, outlined in RFC#6335. The RFC divides ports into three categories:

- **Well-Known Ports (0–1023)**: Reserved for standardized services (HTTP: port 80).
- **Registered Ports (1024–49151)**: User applications or services upon request.
- **Dynamic/Private Ports (49152–65535)**: Temporary communications. [30]

Which can be found here: IANA Service Names and Port Numbers.

Definition 3.16: Transport Layer Protocols (TCP, UDP, QUIC)

Out of numerous transport layer protocols, there are three primary protocols used to dictate the flow of data between devices:

- **TCP**: A connection-oriented protocol that ensures reliable communication by establishing a connection before transmitting data. TCP guarantees delivery, maintains packet order, and retransmits lost packets. It is ideal for applications like web browsing (HTTP/HTTPS) and file transfers (FTP). [67]
- **UDP**: A connectionless protocol that prioritizes speed by sending data without establishing a connection. It does not guarantee delivery, order, or error correction, making it suitable for time-sensitive applications like video streaming, online gaming, and DNS lookups. [63]
- **QUIC**: A modern, connection-oriented protocol built on UDP that combines speed with reliability. QUIC provides features like multiplexed streams, faster connection establishment, and built-in encryption, addressing TCP's latency issues while retaining UDP's efficiency. It is optimized for HTTP/3 and increasing in use. [47]

Definition 3.17: MAC Address

A **MAC address** (Media Access Control address) is a globally unique identifier assigned to a device's network interface card (NIC) at the **network interface layer** of the TCP/IP model. MAC addresses consist of 48 bits, typically represented as six pairs of hexadecimal digits (e.g., `00:1A:2B:3C:4D:5E`).

The first 24 bits represent the **Organizationally Unique Identifier (OUI)**, assigned to manufacturers by the **Institute of Electrical and Electronics Engineers (IEEE)**. The remaining 24 bits are left to the manufacturer to ensure device uniqueness [46].

Definition 3.18: Ethernet

Ethernet is a protocol used at the **network interface layer** of the TCP/IP model for communication within local area networks (LANs). If a network is communicating at these layers—such as in homes, offices, or data centers—Ethernet provides the rules for framing, addressing, and transmitting data between devices using **MAC addresses**. Ethernet is the dominant standard for wired LAN communication. [4]

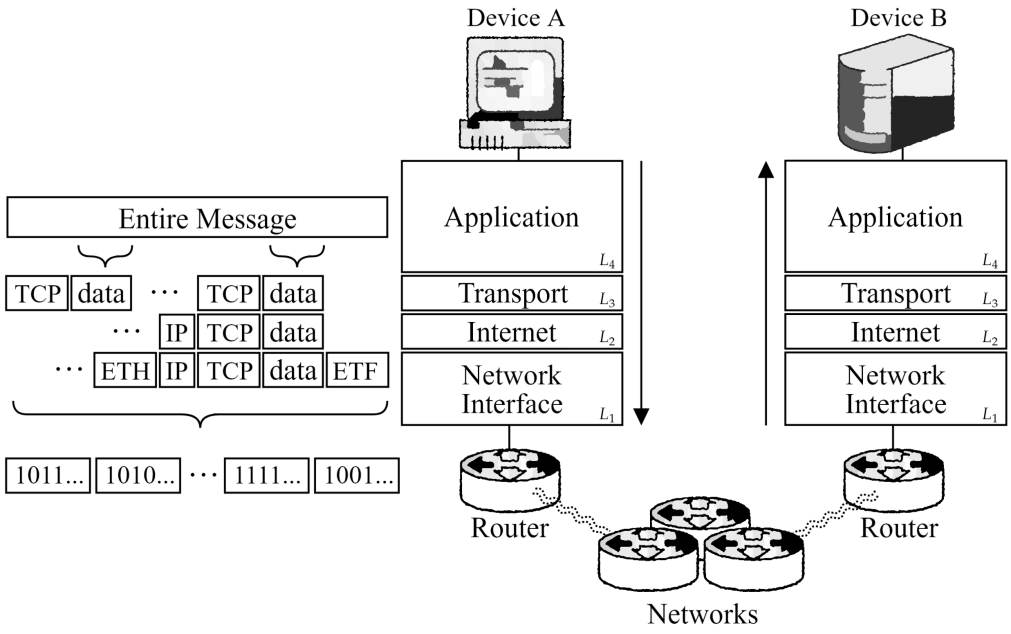


Figure 2.3: Data transmission between devices from application to physical layer and back.

1. **Transport Layer (L3)**: Data is segmented and encapsulated by a protocol like TCP, UDP, or QUIC. Headers with source and destination port numbers are added to enable communication with the correct application.
2. **Network Layer (L2)**: Data is encapsulated into an IP packet, including the source and destination IP addresses. The router uses the destination IP to determine if the packet should be sent locally or to a remote network.
3. **Data Link Layer (L1)**: For local delivery, the packet is encapsulated into an Ethernet frame with source and destination MAC addresses. If the packet is leaving the local network, it is sent to the router. Data is wrapped with a **Ethernet Header and Trailer**.
4. **Routing Across Networks**: If the destination is on another network, routing protocols like **BGP** or **OSPF** determine the optimal path. The packet is forwarded across multiple routers, each stripping the current Ethernet frame and applying a new one for the next hop.

Definition 3.19: TCP Header

A **TCP header**, as defined in RFC#793, is a component of a TCP segment that contains essential information for reliable data delivery between devices. Minimum size of 20 bytes and a max 60 bytes with options. It's called a header as it sits *ahead* of the **payload** (data) in the segment. Eg., [TCP Header (20–60 bytes), Payload (0-65,495 bytes)]. [67]

Field	Size (bits)	Description
Source Port	16	Identifies the sending application on the source device.
Destination Port	16	Identifies the receiving application on the destination device.
Sequence Number	32	Specifies the position of the first byte of data in the current segment relative to the start of the data stream.
Acknowledgment Number	32	Used to confirm receipt of data by specifying the next expected byte.
Data Offset	4	Indicates the size of the TCP header in 32-bit words.
Reserved	6	Reserved for future use; must be set to 0.
Flags (Control Bits)	6	Includes control flags (e.g., SYN, ACK, FIN) for managing connections and flow.
Window Size	16	Specifies the size of the sender's receive window for flow control.
Checksum	16	Ensures the integrity of the TCP header and payload.
Urgent Pointer	16	Points to urgent data within the segment, if the URG flag is set.
Options	Variable	Optional fields for additional settings (e.g., Maximum Segment Size).
Padding	Variable	Ensures the header is a multiple of 32 bits.

Table 2.3: Fields of the TCP Header

Definition 3.20: Maximum Transmission Unit (MTU)

The **Maximum Transmission Unit (MTU)** is the maximum size of a packet, in bytes, that a network can receive before requiring further fragmentation. Theoretically, a payload of 65,535 bytes can be sent, but in practice, are much lower (e.g., Ethernet 1,500 bytes). [43][56]

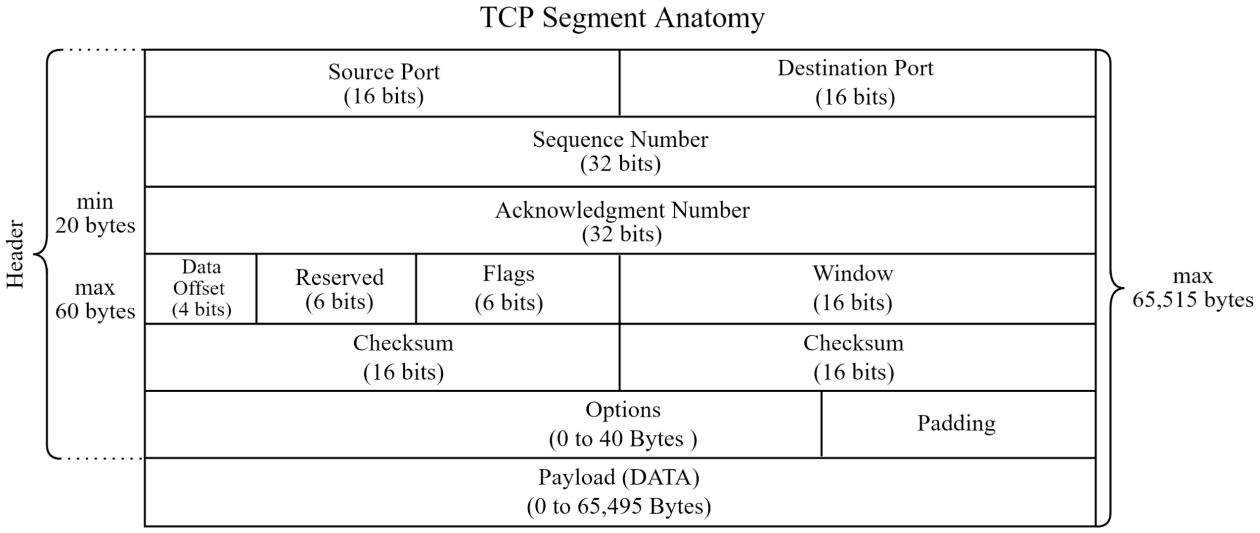


Figure 2.4: TCP Header

Definition 3.21: Flags (Control Bits) in TCP Header

The **Flags (Control Bits)** in the TCP header are a 6-bit field used to control and manage the state of a TCP connection. Each bit represents a specific flag, and one or more flags can be set simultaneously to define the segment's purpose.

List of Control Bits:

- **URG (Urgent Pointer)**: Indicates that the Urgent Pointer field is significant.
- **ACK (Acknowledgment)**: Confirms receipt of data; the Acknowledgment Number field is valid.
- **PSH (Push Function)**: Requests the receiver to pass the data to the application immediately.
- **RST (Reset)**: Resets the connection due to errors or unexpected conditions.
- **SYN (Synchronize)**: Initiates a connection by synchronizing sequence numbers.
- **FIN (Finish)**: Signals the sender's intention to terminate the connection.

[67]

Definition 3.22: TCP Options

- **Maximum Segment Size (MSS)**: Specifies the largest segment size the sender is willing to accept (4 bytes).
- **Window Scale**: Extends the window size field to support larger flow control (3 bytes).
- **Selective Acknowledgment (SACK)**: Allows acknowledgment of specific data blocks, improving performance in packet loss scenarios (variable size).
- **Timestamps**: Provides timing information for round-trip time measurement and protection against wrapped sequence numbers (10 bytes).
- **NOP (No Operation)**: Used for padding to ensure proper alignment (1 byte).
- **End of Option List (EOL)**: Marks the end of the options field (1 byte). [67]

Definition 3.23: SYN-ACK Handshake

The **SYN-ACK Handshake** or is a three-step process used to establish a TCP connection between two devices. After the handshake, data is exchanged bidirectionally via ACKs receipts. Connection termination follows a similar process **FIN-ACK**.

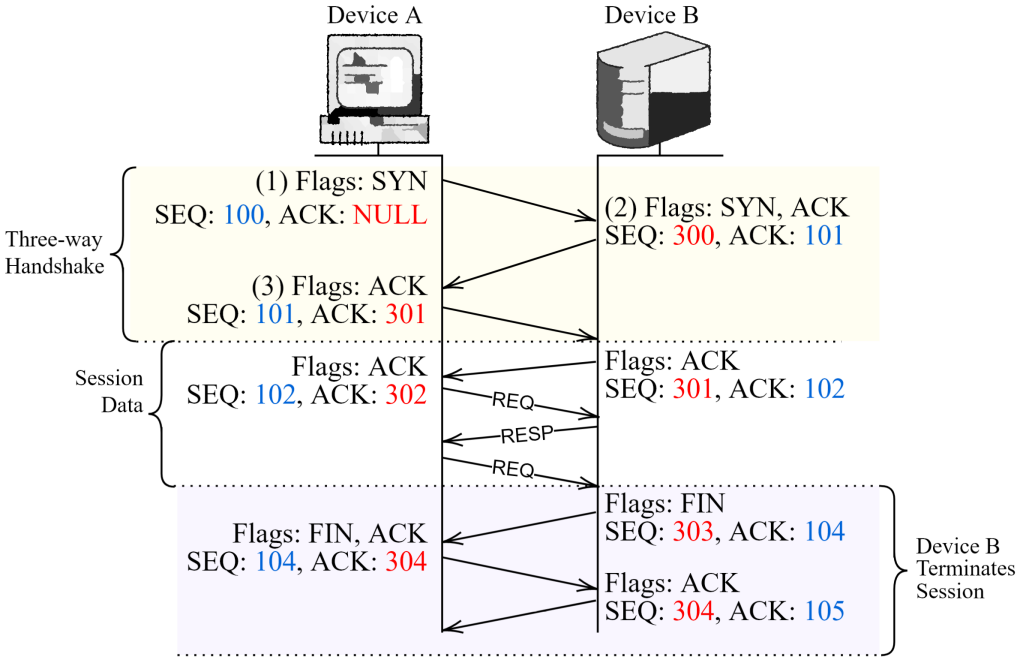


Figure 2.5: SYN-ACK Handshake, showing Sequence (SEQ) and Acknowledgment (ACK) Numbers.

Definition 3.24: TCP Windows

The **TCP Window** is a flow control mechanism that determines how much data can be sent before receiving an acknowledgment from the receiver. The size of the window is specified by the **Window Size** field in the TCP header and can dynamically adjust during the connection.

E.g., Consider a window allowance of 10 packets. If the sender is still waiting on an ACK for packet 1 while it has sent out packets 2-10, the window size reduces momentarily waiting for the receiver to acknowledge the first packet before sending more. [67]

Definition 3.25: UDP Header

The **UDP (User Datagram Protocol) Header** is a fixed-size, lightweight header. It consists of only 8 bytes, containing only essential information. [63]

Field	Size (bits)	Description
Source Port	16	Identifies the sending application. This field is optional and may be set to zero if not used.
Destination Port	16	Identifies the receiving application.
Length	16	Specifies the total length of the UDP packet, including the header and payload, in bytes.
Checksum	16	Provides basic error-checking for the header and payload. If not used, it can be set to zero.

Table 2.4: Fields of the UDP Header

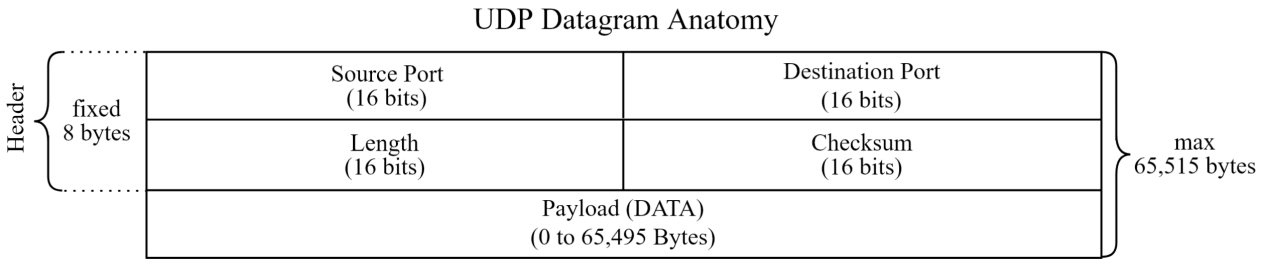


Figure 2.6: UDP Header

Definition 3.26: IPv4 Packet Header

An **IPv4 packet header** as outline in RFC#791, contains essential information for routing and delivering data across networks. The header consists of several fields, each serving a specific purpose. Eg., [IP Header (20–60 bytes), Transport Header, Payload]. in a stream of bits. [66]

Field	Size (bits)	Description
Version	4	Specifies the IP version; for IPv4, this value is 4.
Internet Header Length (IHL)	4	Indicates the header length in 32-bit words; minimum value is 5 (20 bytes).
Type of Service (ToS)	8	Defines packet priority and request for specific handling, such as low delay or high throughput.
Total Length	16	Total size of the packet (header and data) in bytes; minimum is 20 bytes, maximum is 65,535 bytes.
Identification	16	Unique identifier for fragmenting and reassembling packets.
Flags	3	Control flags for fragmentation: Reserved (1 bit), Don't Fragment (1 bit), More Fragments (1 bit).
Fragment Offset	13	Specifies the position of this fragment in the original packet; measured in 8-byte blocks.
Time to Live (TTL)	8	Limits the packet's lifespan; decremented by each router, and discarded when reaching zero to prevent infinite loops.
Protocol	8	Indicates the encapsulated protocol (e.g., 6 for TCP, 17 for UDP).
Header Checksum	16	Error-checking of the header; recalculated at each hop.
Source Address	32	IP address of the sender.
Destination Address	32	IP address of the receiver.
Options	0-320	Optional settings for control, routing, or security; length varies and is not commonly used.
Padding	Variable	Added to ensure the header length is a multiple of 32 bits.

Figure 2.7: IPv4 Packet Header Fields

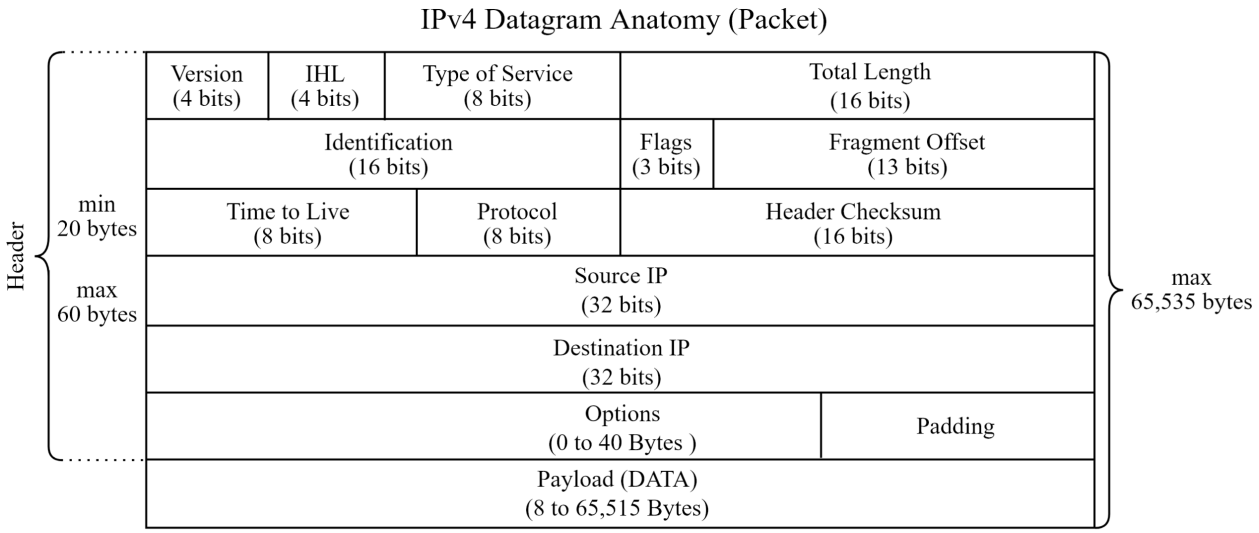


Figure 2.8: IPv4 Packet Header.

Payload is possibly 8-65,515 bytes, because of UDP's 8-byte header. Normally the size is variable, and much more in the ball park of 1,500 bytes.

Definition 3.27: IPv4 Flags

Reserved (Bit 0): Reserved for future use, must be set to `0`.
Don't Fragment (DF, Bit 1): `0`: Fragmenting allowed. `1`: Fragmented prohibited; if required, the packet is discarded, and an ICMP error (next definition) is sent to the sender.
More Fragments (MF, Bit 2): `0`: Last fragment. `1`: More fragments are expected. [66]

Definition 3.28: Internet Control Message Protocol (ICMP)

Internet Control Message Protocol (ICMP) is a supporting protocol used by network devices to communicate error messages and operational information. To name a few:

- **Destination Unreachable**: Indicates that a packet could not reach its destination.
- **Time Exceeded**: Signals that the packet's **Time to Live (TTL)** has expired.
- **Fragmentation Needed**: Informs the sender the packet is too large without fragmenting.
- **Echo Request and Reply (Ping)**: Tests connectivity between devices. [64]

For full specifications visit: ICMP Parameters. Note viewing their reference RFCs may be more informative.

Definition 3.29: IPv4 Options

The **IPv4 Options** field is an optional. Every options occupies 1 byte regardless do data (var = variable, '-' = none).

Common IPv4 Options (class, number, data length (bytes)):

- **End of Option List (0, 0, -)**: Marks the end of the options field.
- **No Operation (0, 1, -)**: Used for padding.
- **Security (0, 2, 11)**: Includes security and compartmentalization details.
- **Loose Source Routing (0, 3, var)**: Specifies a loose route for pathing.
- **Internet Timestamp (2, 4, var)**: Records timestamps at each hop.
- **Strict Source Routing (0, 9, var)**: Specifies an exact route for pathing.
- **Record Route (0, 7, var)**: Records the route taken by the datagram.
- **Stream ID (0, 8, 4)**: Identifies the stream of communication.

Classes: 0: Control. **1**: Reserved for future use. **2**: Debugging and Measurement. **3**: Reserved for future use. [66]

QUIC Headers will be introduced later, as it incorporates uncovered content. The following solution was implemented to address the exhaustion of IPv4 addresses:

Definition 3.30: Private and Public IP Addresses

IP addresses are classified as **private** or **public** based on their scope and accessibility:

- **Private IP Addresses**: Reserved for use within private networks (e.g., home or corporate networks). These addresses cannot communicate directly over the internet and include ranges like:
  - $10.0.0.0 - 10.255.255.255$
  - $172.16.0.0 - 172.31.255.255$
  - $192.168.0.0 - 192.168.255.255$
- **Public IP Addresses**: Globally unique addresses assigned by the Internet Assigned Numbers Authority (IANA) for communication over the internet.

Devices on a private network can communicate internally using private IPs, while a public IP is needed to access external networks like the internet. Communication between these address types is facilitated by **Network Address Translation (NAT)**. [70]

Definition 3.31: Network Address Translation (NAT)

Network Address Translation (NAT) is a networking technique that translates private IP addresses within a local network to a public IP address for communication over external networks, such as the internet. NAT modifies the source IP address in outbound packets and reverses the process for inbound packets, ensuring data is correctly routed to devices within the private network.

NAT enables multiple devices to share a single public IP address, facilitating efficient communication while keeping private IP addresses hidden from external networks. [70][75]

Definition 3.32: Dynamic & Static NATs and PATs

When a device crosses a router into an external network, the router replaces parts of the packet with a public translation. There are **NATs** and **Port Address Translations (PATs)**:

- **NAT**: Only translates the IP address, leaving the port unchanged.
- **PAT**: Translates both the IP address and port number (rarely, only the port).

There are two types of configurations for NAT and PAT:

- **Static:** The administrator explicitly defines the translations of IP addresses and ports.
- **Dynamic:** The administrator defines a pool of public IP addresses and ports, allowing the router to assign them dynamically.

PATs are referred to in RFC#2663 as **Network Address Port Translation (NAPT)**. This overall process is also known as **Overloading** or **IP Masquerading**. [75]

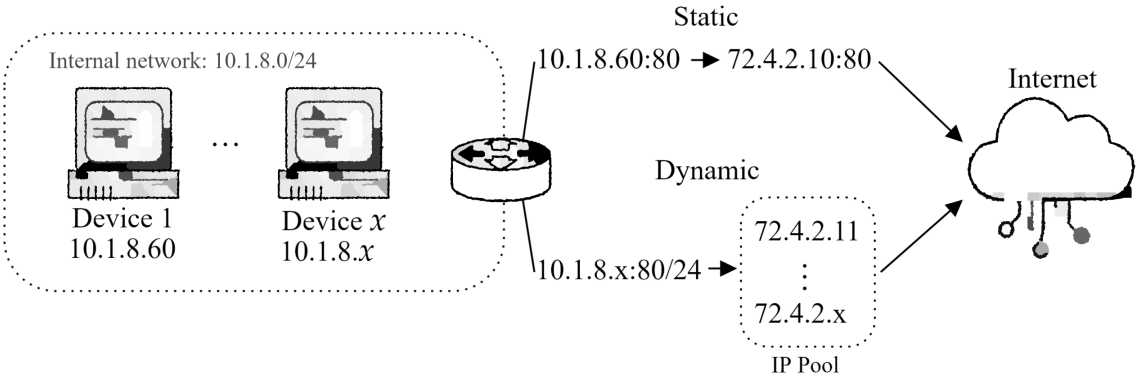


Figure 2.9: Static and Dynamic NATs from an internal network to the internet.

Incoming traffic will reverse the process, translating the public IP back into its private mapping. In many cases, an administrator would be an ISP, who distributes public IPs to clients.

Definition 3.33: Regional Internet Registries (RIRs)

Regional Internet Registries (RIRs) are non-profit organizations responsible for the allocation, distribution, and management of Internet number resources, including **IPv4 and IPv6 addresses** and **Autonomous System Numbers (ASNs)**, within specific geographic regions.

Key Functions of RIRs:

- **Database Management**: Maintain public records documenting the allocation and assignment of Internet number resources.
- **Policy Development**: Facilitate community-driven processes for creating policies on resource management.

The Five RIRs and Their Regions:

- **ARIN**: North America, Canada, parts of the Caribbean.
- **RIPE NCC**: Europe, the Middle East, and Central Asia.
- **APNIC**: Asia and the Pacific region.
- **LACNIC**: Latin America and parts of the Caribbean.
- **AFRINIC**: Africa and the Indian Ocean region.

[59]

Definition 3.34: Carrier-Grade NAT (CGNAT/NAT444)

Carrier-Grade NAT (CGNAT), also known as **NAT444**, is a network translation mechanism used by Internet Service Providers (ISPs) to address IPv4 address exhaustion. It refers to three sets of IPv4 addresses:

- **Customer Private**: IPv4 addresses within the customer's local network.
- **ISP Private**: IPv4 addresses used internally within the ISP's network.
- **Public Internet**: IPv4 addresses used for external communication.

Network Address Translation occurs in two stages:

- **Customer Private → ISP Private**: Translation by the customer's router—**Customer Premises Equipment (CPE)**.
- **ISP Private → Public Internet**: Translation by the ISP's CGNAT device.

[75][73]

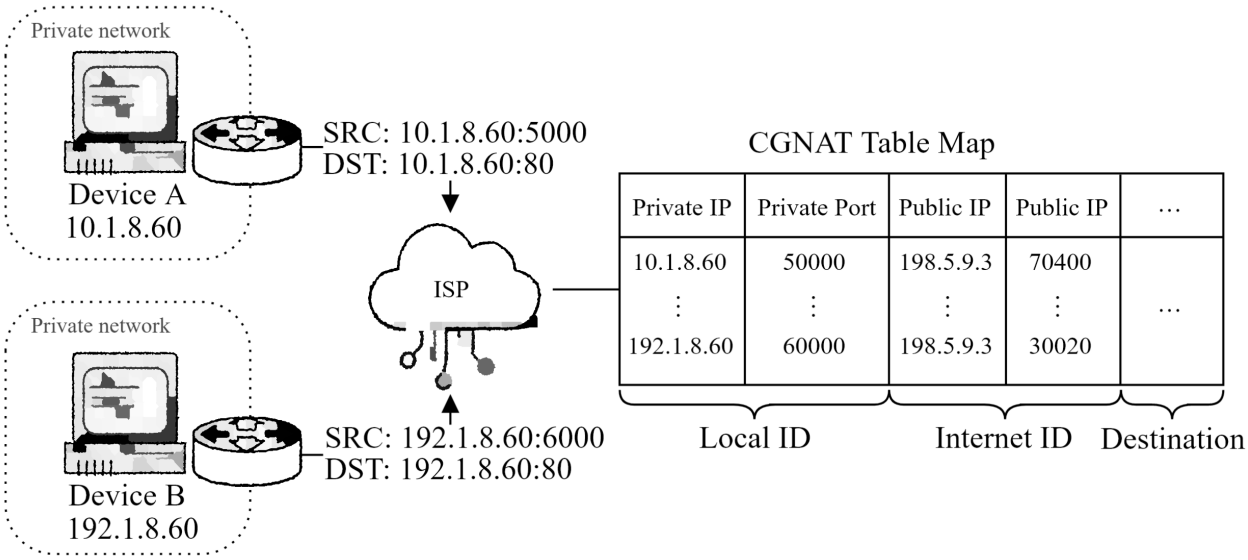


Figure 2.10: Carrier-Grade NAT (CGNAT) Translation Process

Definition 3.35: DNS Servers

Domain Name System (DNS) Servers are specialized servers responsible for translating human-readable domain names (e.g., `example.com`) into machine-readable IP addresses (e.g., `192.0.2.1`), a process known as a **DNS Lookup**. Retrieving a website involves:

- Querying DNS servers to resolve a domain name into its corresponding IP address.
- Traversing a hierarchical network of DNS servers: starting from **root servers**, through **Top-Level Domain (TLD) servers** (e.g., `.com`), to the **authoritative DNS server** for the domain.

Control and Operation:

- **Root DNS Servers** are managed by independent organizations under the coordination of the **Internet Corporation for Assigned Names and Numbers (ICANN)**.
- **TLD DNS Servers** manage domains under specific extensions like `.com`, `.org`, and country-specific domains like `.uk`.
- **Authoritative DNS Servers** are controlled by entities that manage specific domain names, including private companies, governments, or educational institutions.

Caching: Devices (e.g., computers, routers) and services (e.g., ISPs) may store previously resolved domain names in a **DNS cache**, reducing the need for repeated lookups and improving connection speed. [55]

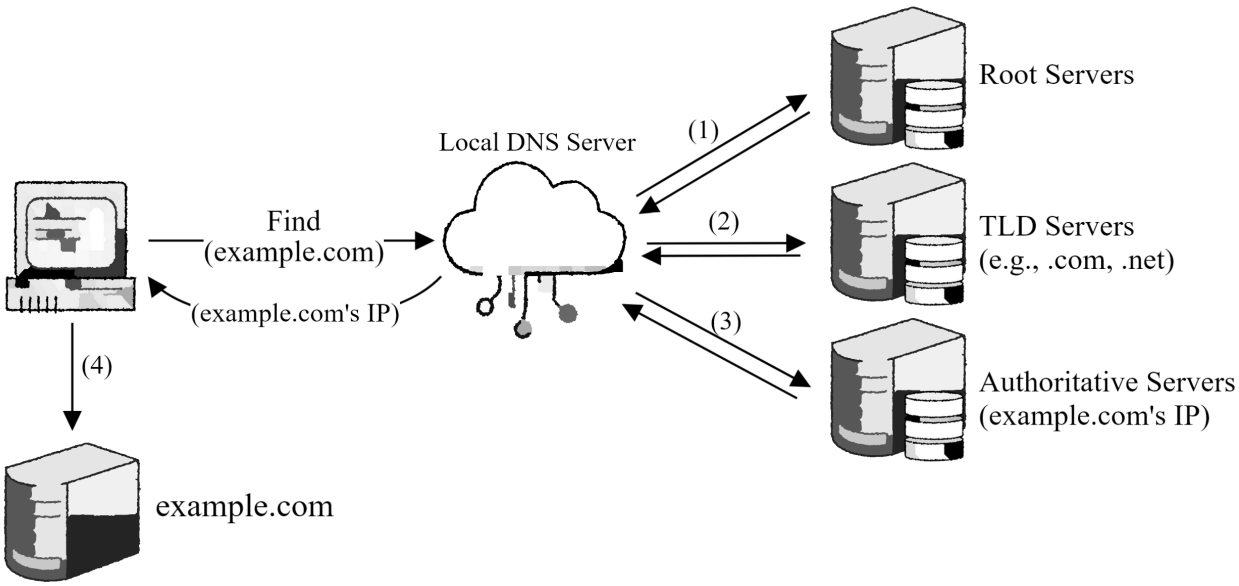


Figure 2.11: DNS Lookup Process: (1). Query root server for (`.com`), (2). Query TLD for (`example.com`), (3). Query Authority for (`example.com's IP`).

Definition 3.36: DNS Hierarchy: Root, TLD, and Authoritative Servers

- **Root Servers**: Only provide references to the appropriate **TLD servers** based on the queried domain extension (e.g., `.com`, `.org`).
  - Operated by organizations like VeriSign, USC, and ICANN.
  - Example: The `A` Root Server is managed by VeriSign.
  - View the full list at: https://root-servers.org/.
- **TLD Servers**: Manages **Top-Level Domains (TLDs)** such as `.com`, `.org`, and `.uk`. They direct queries to the **authoritative servers** for the requested domain.
  - Example: VeriSign manages TLD servers for `.com` and `.net`.
  - Often operated by private companies or regional internet registries.
- **Authoritative Servers**: These servers store DNS records for specific domain names, responding with the IP address of the queried domain.
  - Example: A hosting provider like AWS or Google Cloud might manage the authoritative server for `example.com`.
  - Businesses, ISPs, or hosting services typically operate these servers.

[6][17]

Definition 3.37: HTTP Messages

When communicating with a web server, a client establishes an **HTTP session** to exchange data. The client sends an **HTTP request** to the server, which processes the request and responds with an **HTTP response**. The HTTP header and Response consists of:

1. **Request Line**: Contains the request method (e.g., GET, POST), URL, and HTTP version.
2. **Request Headers**: Include additional information such as DNS and client software.
3. **Empty Line**: Separates the header from the body.
4. **Request Body**: Contains data sent to the server (e.g., form data). Optional for some requests (e.g., GET).

Messages are sent over **TCP** or **UDP** stored in the **Payload** of the datagram. [39]

Definition 3.38: Uniform Resource Identifier (URI) & URLs

A **Uniform Resource Identifier (URI)**: a string which identifies resources on the web. A **Uniform Resource Locator (URL)** a subset of URI that specifies the resource and its location. A **Uniform Resource Name (URN)** is a URI under a different schema, a unique global identifier of a resource even after it is no longer available (e.g., a book isbn, urn:isbn:0451450523). [10]

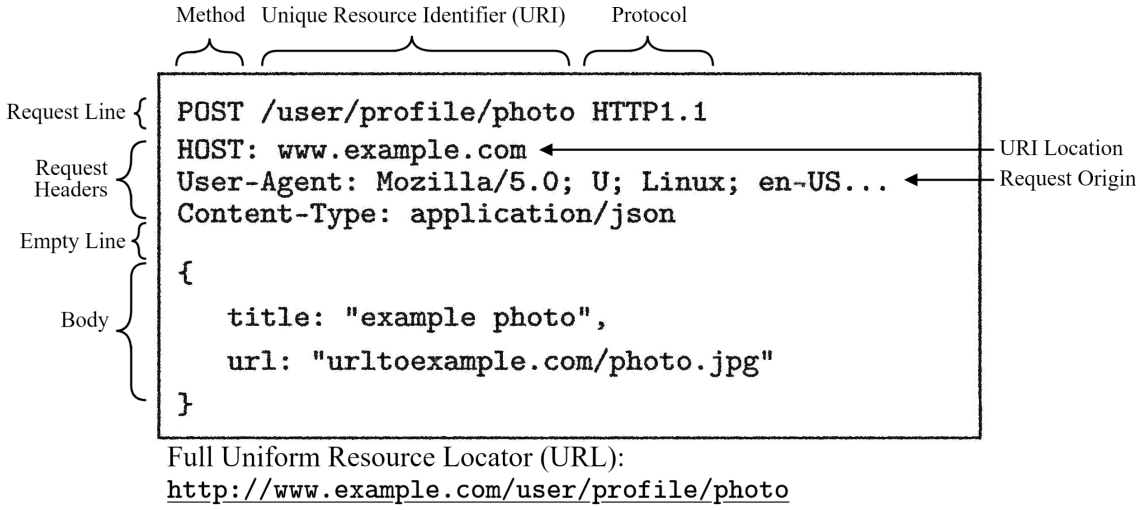


Figure 2.12: HTTP Request and Response Structure

Definition 3.39: Ethernet Frame

Ethernet Frames at the **Data Link Layer** (ISO model), encapsulate IP packets for transmission over a local network. The specifications for Ethernet, including frame structure and operational parameters, are defined by the **Institute of Electrical and Electronics Engineers (IEEE)** in the **IEEE 802.3** standard. [2]

Field	Size (bytes)	Description
Preamble	7	A sequence of alternating 1s and 0s (10101010) used for synchronization between the sending and receiving devices.
Start Frame Delimiter (SFD)	1	Indicates the start of the frame; typically has the bit pattern 10101011.
Destination MAC Address	6	The hardware address of the intended recipient.
Source MAC Address	6	The hardware address of the sender.
EtherType/Length	2	Specifies either the protocol type encapsulated in the payload (if ≥ 1536) or the length of the payload in bytes (if ≤ 1500).
VLAN Tag (optional)	4	Present if IEEE 802.1Q tagging is used; includes Priority Code Point (PCP), Drop Eligible Indicator (DEI), and VLAN Identifier (VID).
Payload	46–1500	The encapsulated data from higher network layers. If the payload is less than 46 bytes, padding is added to meet the minimum frame size requirement.
Frame Check Sequence (FCS)	4	A cyclic redundancy check (CRC) value used to detect errors in the transmitted frame.

Table 2.5: Structure of an Ethernet Frame

Theorem 3.2: Establishing a Website Connection

1. **Application Layer (DNS Request)**: The browser initiates a DNS query to resolve the website's domain name (e.g., `example.com`) into an IP address. The DNS query traverses:

   - **Root DNS Server→TLD Server→Authoritative DNS Server**. (e.g., `.com`→`example.com`→`example.com's IP`).

   If the IP address is cached locally or by the ISP, the query resolves faster.

2. **Transport Layer (Connection Establishment)**: After the IP address is resolved, the browser establishes a connection to the server. If using **TCP**, the connection involves a **three-way handshake**:

   - The browser sends a **SYN** packet to the server.
   - The server responds with a **SYN-ACK** packet.
   - The browser completes the handshake by sending an **ACK** packet.

   For protocols like **UDP**, no handshake occurs, and packets are sent directly without connection establishment.

3. **Network Layer (Routing)**: Packets, regardless of protocol (e.g., TCP, UDP, or ICMP), are encapsulated within IP datagrams and routed to the destination. Routers along the path forward the packets based on the destination IP address in the IP header. The routing process is agnostic to the transport protocol and works universally across all IP-based communication.

4. **Data Link Layer (Frame Transmission)**: If connection is within a local network (e.g., LAN), IP packets incur Ethernet frame encapsulation (or equivalent, depending on the network).

5. **Physical Layer (Transmission)**: Frames are transmitted as electrical signals, light pulses, or radio waves, depending on the physical medium used. **Inter-domain routing protocols**, such as **OSPF** or **BGP**, facilitate the transfer of packets through **AS**es.

6. **Application Layer (HTTP Request)**: Once the connection is established, the browser sends an **HTTP request** to the server via datagram **Payload**:

   - The browser requests the document.
   - The server responds with the HTML content, which is sent back over the established connection.

7. **Response and Rendering**: The server's response is segmented into TCP packets (if applicable) and reassembled by the browser. The browser processes the HTML and renders the webpage for the user.

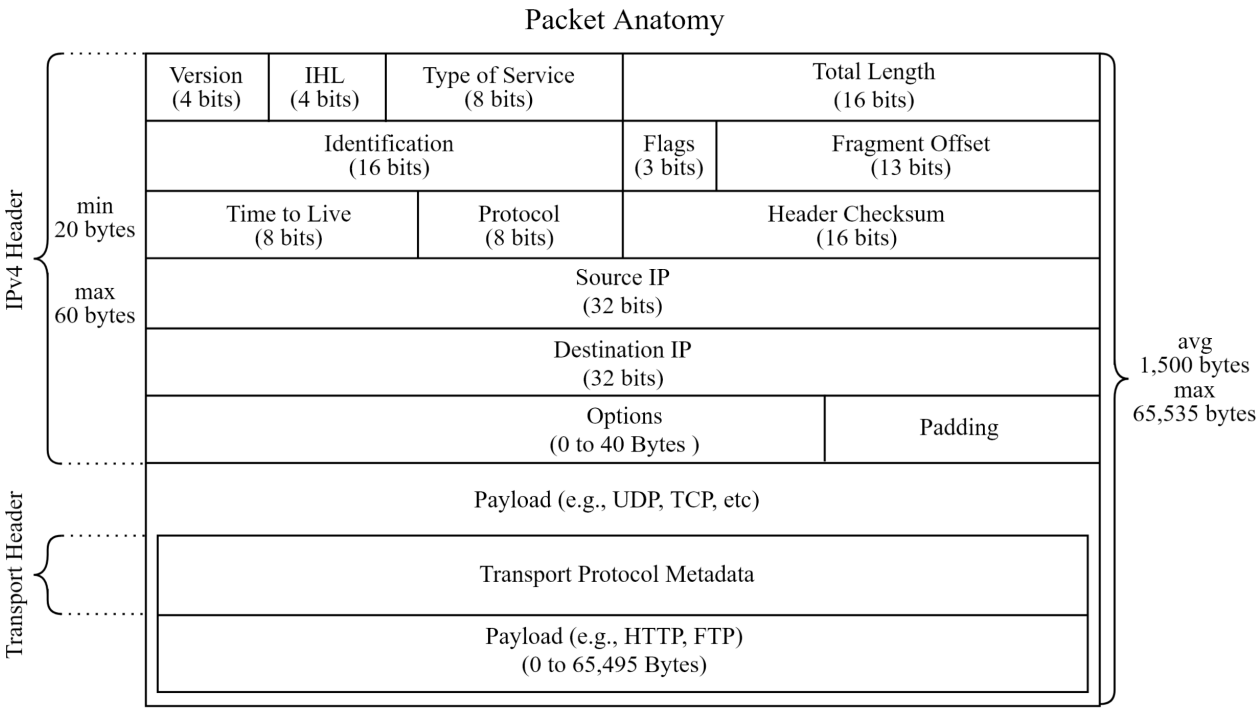


Figure 2.13: Anatomy of a packet, from IPv4[TCP/UDP[Data]]

— 3 —

Software Security & Defenses

3.1 Document Certificates & Binding Encryptions

Before, all communication sent "**over the wire**" (from device to device), was sent "**in the clear**" (unencrypted). This means that anyone could view data sent between devices in plain text. This is a problem when setting up infrastructure such as banking, e-commerce, or any other service that requires sensitive information to be sent over the internet.

Definition 1.1: Integrity & Authenticity

Integrity is the assurance that data has not been altered in transit.
Authenticity is the assurance that the data is coming from the correct source.

Definition 1.2: Transport Layer Security (TLS)

TLS is a protocol providing end-to-end encryption of data. It authenticates the server via **TLS certificates** to ensure the client is connecting to the correct host. It also ensures integrity of the data.

The Engineering Task Force (IETF) published the first version of TLS in 1999. As of today the most recent version is TLS 1.3. (2018). [20]

Definition 1.3: Secure Sockets Layer (SSL) [Deprecated

SSL is the predecessor to TLS. It was developed by Netscape in the 1990s. SSL 3.0 was released in 1996. SSL 3.0 was found to be insecure and was replaced by TLS 1.0 in 1999. [20]

Definition 1.4: Certificate Authority (CA)

A CA is a third-party entity that issues digital certificates. Often called **SSL certificates** or TLS certificates. The protocol supports both SSL and TLS. Despite SSL's deprecation the name stuck due branding issues. Browsers and Operating systems have a list of trusted CAs called the **root store**. A full list of Microsoft's trusted CAs can be found here:
https://ccadb.my.salesforce-sites.com/microsoft/... [50]

Definition 1.5: Encryption

Encryption is the process of converting plaintext into ciphertext (indiscernible text).
Decryption is the process of converting ciphertext back into plaintext.

Definition 1.6: Symmetric & Asymmetric Encryption

Key: is a seed/piece of information used to encrypt or decrypt data.
Symmetric Encryption: uses the same key for both encryption and decryption.
Asymmetric Encryption: uses a public key for encryption and a private key for decryption. [5]

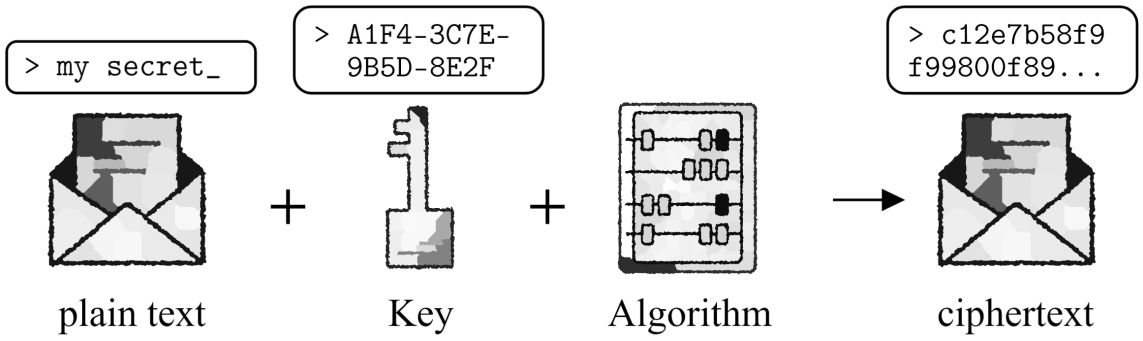


Figure 3.1: High-level depiction of encryption.

Encryption takes a key, data, and an algorithm to produce ciphertext. Decryption takes the same key, ciphertext, and algorithm to produce the original data.

Definition 1.7: Hashing

Hashing is the process of converting data into a fixed-length string of characters. Hashing is a one-way function, meaning it cannot be reversed (theoretically). In practice, it is computationally infeasible to reverse a hash without brute force (trying all possible inputs) or exploiting weaknesses in the hashing algorithm. The results of a hashing algorithm is often called the **digest**. [27]

Definition 1.8: Hypertext Transfer Protocol Secure (HTTPS)

A version of HTTP that uses TLS to encrypt data. [21]

Definition 1.9: SSL/TLS Certificate Specifications

- **Common Name (CN)**: The domain name the certificate is issued for.
- **Subject Alternative Name (SAN)**: Additional domain names or subdomains covered by the certificate.
- **Key Length**: A minimum of 2048 bits, ensuring strong encryption.
- **Hashing Algorithm**: Typically SHA-256 for secure data integrity.
- **Valid From/To**: The validity period, usually up to 397 days.
- **Issuer**: The trusted Certificate Authority (CA) that issued the certificate.
- **Extended Key Usage**: Specifies purposes like server authentication or client authentication.

[50]

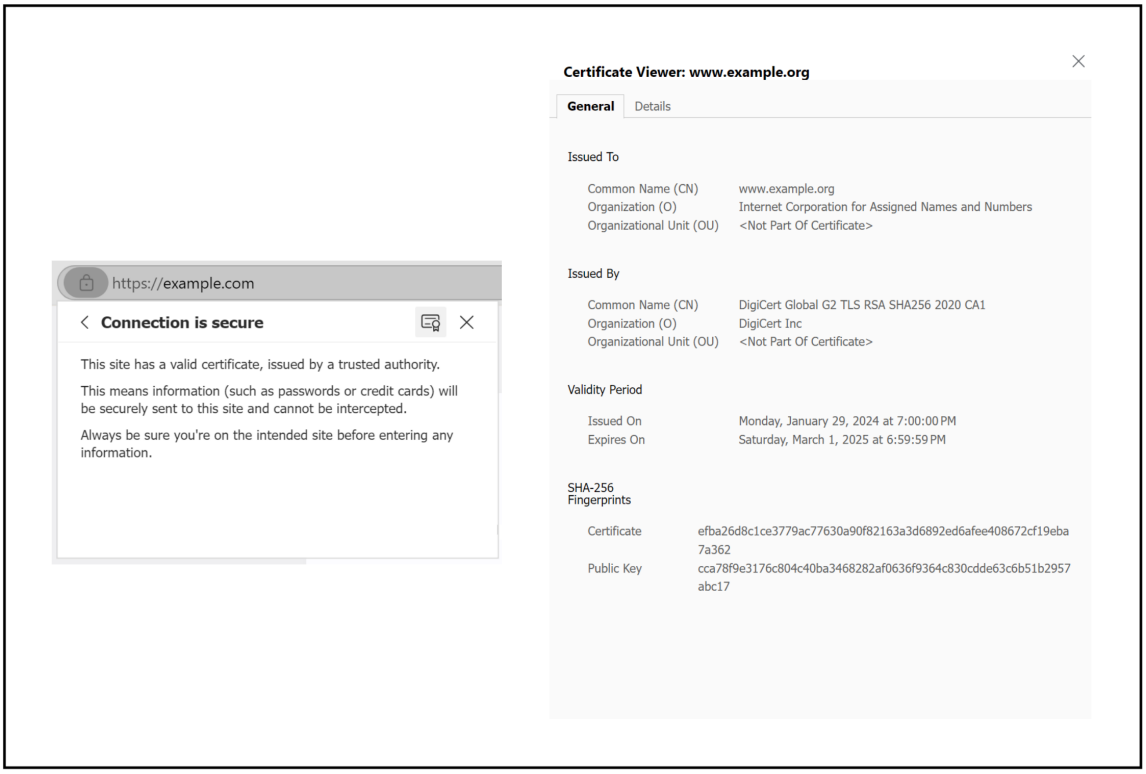


Figure 3.2: SSL certificate obtained through the Edge browser on example.com

Definition 1.10: Public Key Infrastructure (PKI)

PKI is a system for managing digital certificates. It includes the creation, distribution, and revocation of certificates. PKI is used to secure data transmission over the internet such as secure email, VPNs, and other services. [61]

Definition 1.11: Digtial & Authority Certificates

Digital Certificate: An electronic document binding and proving ownership of a public key. **Authority Certificate**: or **Root Certificate** issued by a CA, signs other certificates. it is **self-signed** and is the top of the certificate chain.

This establishes layers of trust and distance between the root certificate and the end-user certificates. If the root certificate is compromised, all certificates through the chain are compromised. [85]

Definition 1.12: Digital Signatures

To verify a source, a **digital signature** is used.

- **Key Generation**: A private and public key pair is generated beforehand.
- **Hashing**: A cryptographic hash of the data is created.
- **Encryption**: The private key encrypts the hash, creating the digital signature through an algorithm.
- **Verification**: The recipient decrypts the signature with the public key and compares the result to their own hash of the data. [31]

Definition 1.13: Certificate Signing Request (CSR)

To obtain a digital certificate, a client generates a CSR, which involves the following steps:

1. **Generate Key Pair**: Create a private key (kept secret) and a public key (shared).
2. **Produce CSR Data**: Include identifying information (e.g., Organization (O), Common Name (CN)), the public key, and other details in a standardized format such as X.509.
3. **Sign the CSR**: Hash the CSR data and sign it using the private key to prove ownership.
4. **Submit and Issue Certificate**: Submit the CSR to a CA, which validates the signature, signs the certificate with its own private key, and issues the certificate. [60]

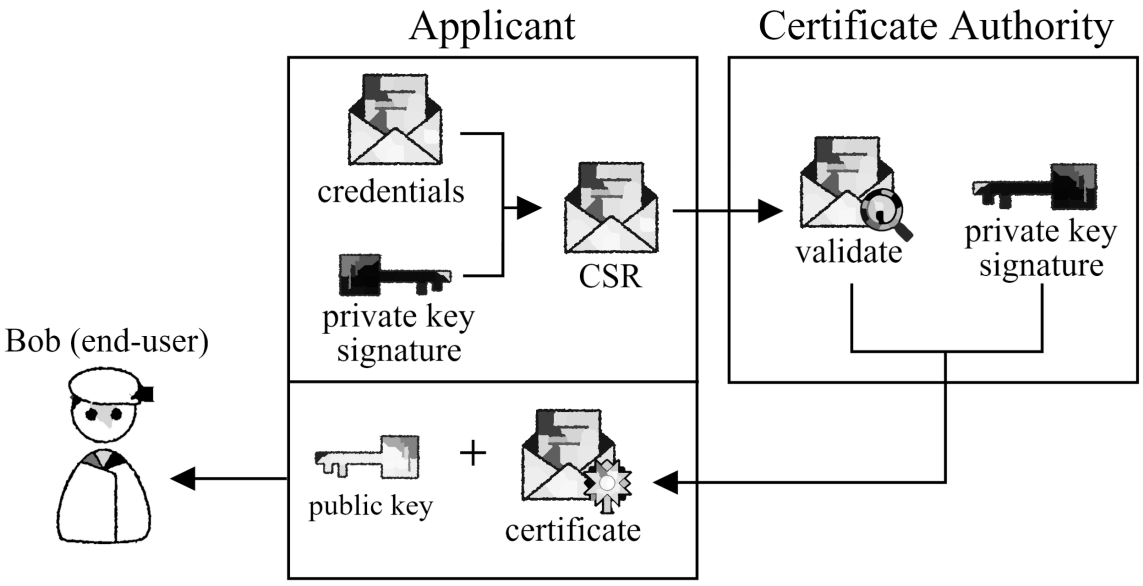


Figure 3.3: Example of a Certificate Signing Request (CSR)

Definition 1.14: Certificate Revocation List (CRL)

A CRL is a list of certificates that have been revoked by the CA before their expiration date. This is used to prevent the use of compromised certificates. [29]

Definition 1.15: Chain of Trust

A **Chain of Trust** is a hierarchical sequence of certificates used in PKI to establish trust between entities. It consists of:

- **Root Certificates**: Self-signed certificates at the top of the trust chain, trusted directly by operating systems and browsers.
- **Intermediate Certificates**: Issued by the root CA to delegate trust, adding a layer of security by isolating the root CA from direct interactions.
- **End-Entity Certificates**: Issued to users, servers, or devices to authenticate their identity and enable secure communications.

Each certificate is digitally signed by the private key of the certificate authority above it in the chain, with the root certificate serving as the ultimate trust anchor. Certified issuers are preferred over self-signed certificates because they undergo rigorous external validation, creating a verifiable path of trust. [29]

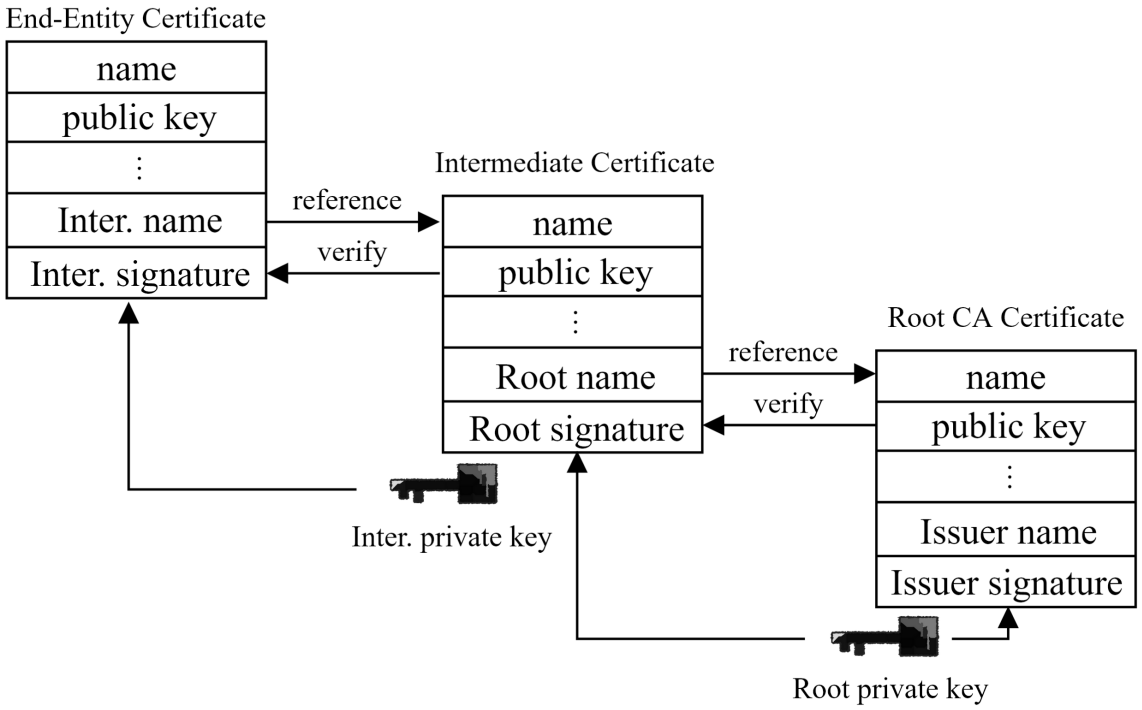


Figure 3.4: A Chain of Trust from the Root Certificate to the End-Entity Certificate.

Definition 1.16: Root Certificate Authority (CA) Signing Ceremonies

A **Root Certificate Authority (CA) Signing Ceremony** is a highly secure process in which a Root CA's private key is used to sign subordinate certificates, establishing trust within a Public Key Infrastructure (PKI). Key characteristics include:

- **Rigorous Security**: Conducted in offline, access-controlled environments with multiple layers of physical and procedural security. Entry requires the presence of multiple trusted individuals simultaneously.
- **Defined Roles**: Roles such as Crypto Officers, Witnesses, and Administrators are assigned to ensure transparency and accountability.
- **Global Trust Anchors**: Managed by organizations like ICANN for DNSSEC, these ceremonies protect the integrity of critical internet infrastructure.
- **Independent Operations**: Root CAs are typically independent from government oversight, though some, like the U.S. Department of Defense, manage government-affiliated Root CAs.

Learn More: https://cloudflare.com/learning/dns/dnssec/root-signing-ceremony/ [19]

Definition 1.17: DNS over HTTPS (DoH) and DNS over TLS (DoT)

DNS over HTTPS (DoH) and **DNS over TLS (DoT)** are protocols designed to encrypt DNS queries, improving privacy and security:

- **DoH**: Encrypts DNS queries over the HTTPS protocol (port 443), making them indistinguishable from regular HTTPS traffic.
- **DoT**: Encrypts DNS queries using the TLS protocol (port 853), ensuring DNS requests are secure and tamper-proof. Though because of its use of port 853, traffic is more easily identifiable as DNS, making DoH the preferred method.

DNS security is known as **DNSSEC**. [18]

3.2 Encryption Algorithms & Security Definitions

In the previous section the notion of encryption was introduced (asymmetric and symmetric). Here adversaries (i.e., attackers, or hackers) will be defined along with the specifications an algorithm must meet.

Definition 2.1: Adversaries

Adversaries are entities that attempt to break the security of a system. There are two types of adversaries:

- **Eavesdroppers:** Can only intercept and read messages.
- **Man-in-the-Middle (MitM):** Can intercept, read, and modify messages.

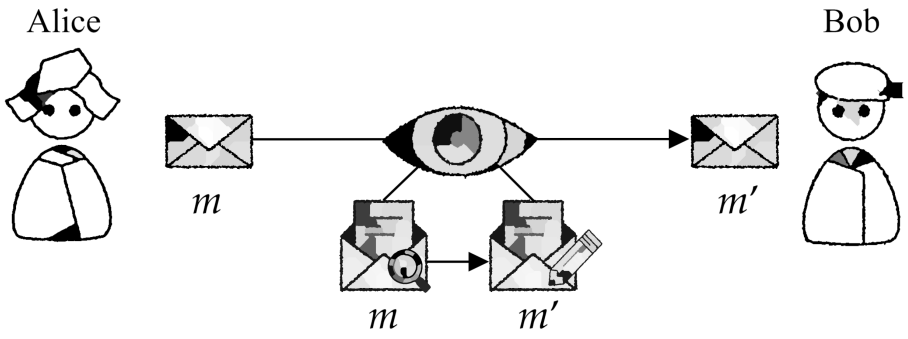


Figure 3.5: A MitM Attack, reading and altering the contents of m and sending m'.

Instead plain variables A and B, often Alice and Bob are used. There are other common for entities, learn more here: https://en.wikipedia.org/wiki/Alice_and_Bob.

Definition 2.2: Security Definitions

Security definitions formalize the properties a system must satisfy to resist adversarial attacks. These include:

- **Confidentiality:** Ensures that adversaries cannot learn the contents of the message.
- **Integrity:** Guarantees that adversaries cannot alter the message without detection.
- **Authenticity:** Verifies that the message originates from the claimed sender and has not been tampered with.

Theorem 2.1: Kerckhoffs's Principle

"Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi."

Literal translation: [The method] must not be required to be secret, and it must be able to fall into the enemy's hands without causing inconvenience [72].

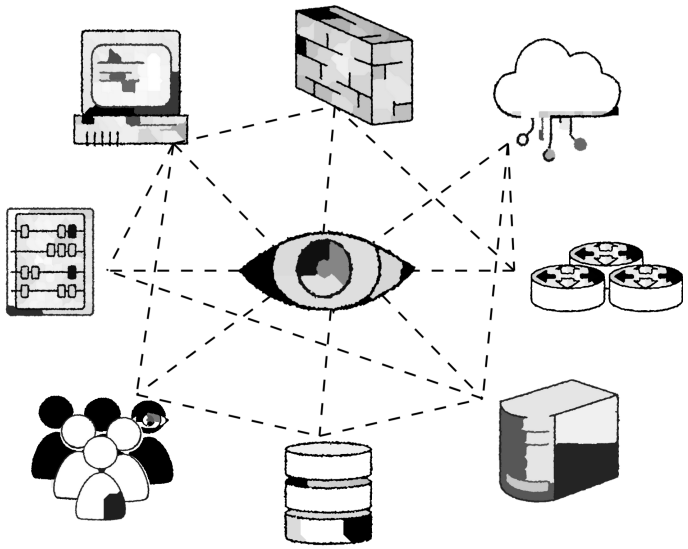


Figure 3.6: Kerckhoffs's Principle, an adversary's unbounded sight.

I.e., **The adversary knows all**—the algorithm, the architecture, an insider, any exploit—all communication is naked, visible on the wire. Though despite all, is secure. This is the essence of Kerckhoffs's Principle.

Definition 2.3: Non-cryptographic Security

A system which does not follow Kerkhoff's Principle: security through obscurity (hiding the algorithm) is not a cryptographic. As if any rosetta stone is found, the system is compromised.

Note: The rosetta stone was a slab of stone found in 1799, which helped decipher Egyptian hieroglyphics. I.e., a key between languages. Learn more: https://wikipedia.org/rosetta_stone.

The rest of the section will cover methods used over centuries to attempt to secure messages.

Theorem 2.2: Caesar Cipher

The Caesar Cipher is a non-cryptographic scheme, named after Julius Caesar, dating back 45BC to protect military communications. Each letter in the plaintext is shifted x places down the alphabet. E.g., $x = 3$, 'A'='D', 'B'='E', and so on. [78]

Note: Here is a fun online tool to try the Caesar Cipher: https://cryptii.com/caesar-cipher.

Theorem 2.3: Vigenère Cipher

The Vigenère Cipher is a non-cryptographic scheme, created in mid-1500's by italian cytologist Giovan Battista Bellaso, later popularized and misattributed to Blaise de Vigenère. It addressed the Caesar Cipher's weakness by using attributing odd and even digit places to different Caesar Ciphers.

This was aimed to stop frequency analysis attacks, as for instance, 'E' is the most common letter in English. If a letter is repeated at high frequency, it is likely 'E'. [78]

0	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
1	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e
2	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	a	b	c	d	e	f	g	h	i	j

Figure 3.7: Vigenère Cipher Table: row 0 is the key, 1 even, and 2 odd shift.

For example "Coffee" would be "hykpjo" deterring frequency analysis of the letter 'E'.

Function 2.1: Key & String Length ($\lambda, \|a\|$)

The rest of the text may denote 'λ' (lambda) as the bit length of the key. E.g., $\lambda = 8$ bits, which in in binary may hold $2^8 = 256$ values ($[0000\ 0000]_2$-$[1111\ 1111]_2$). The λ is variable is often called the **security parameter**.

For ease of notation, $\|a\|$ denotes the length of a character or binary string a. E.g., $\|a\| = 5$ for the string $a =$ "hello" (the use of which will always be explicit). [72]

Definition 2.4: One-Time Pad (OTP)

OTP, also known as the **Vernam Cipher** is a cryptographic scheme, invented by Gilbert Vernam in 1919. Earlier depictions though date back to 1882 by Frank Miller on telegraphy.

Security Definition: Confidentiality is guaranteed if the key is used only once.

Function 2.2: One Time Pad - `Enc(m, k)` & `Dec(k, c)`

Let function $k \leftarrow \{0,1\}^\lambda$ generate keys k:

- $\{0,1\}$ denotes the set of possible inputs.
- k is a random bit string of length λ bits consisting of 0's and 1's. E.g., $\lambda = 8$ then $k = [1010\ 1101]_2$ is a possible output.
- $\{0,1\}^\lambda$ should be a uniform distribution, i.e., each bit is equally likely to be 0 or 1.

Let m be and c be an encrypted message, both length λ bits. Then:

- $c \leftarrow \text{Enc}(m, k) := m \oplus k$. (Encryption)
- $m \leftarrow \text{Dec}(k, c) := c \oplus k$. (Decryption)

Where $\oplus$ denotes the XOR operation ($1 \oplus 1 = 0; 0 \oplus 1 = 1; 1 \oplus 0 = 1; 0 \oplus 0 = 0$).

$$c \leftarrow \text{Enc}(m,k) := \begin{cases} 0011\ 0100\ 1101\ 1000\ 1111 & (m) \\ \oplus\ 1110\ 1010\ 0110\ 1000\ 1101 & (k) \\ \hline 1101\ 1110\ 1011\ 0000\ 0010 & (c) \end{cases} \quad \text{(Encryption)}$$

$$m \leftarrow \text{Dec}(c,k) := \begin{cases} 1101\ 1110\ 1011\ 0000\ 0010 & (c) \\ \oplus\ 1110\ 1010\ 0110\ 1000\ 1101 & (k) \\ \hline 0011\ 0100\ 1101\ 1000\ 1111 & (m) \end{cases} \quad \text{(Decryption)}$$

The larger the key, the more secure the scheme becomes, as smaller keys have a higher probability of being brute-forced by generating all possible keys.

Theorem 2.4: Computational Security

Computational Security is a security definition that guarantees that the adversary cannot break the scheme in a reasonable amount of time. In todays standards, exponential time algorithms are considered infeasible, (e.g., $O(2^\lambda)$ time complexity).

Efficient algorithm known:	**No known efficient algorithm:**
Computing GCDs	Factoring integers
Arithmetic mod N	Computing $\phi(N)$ given N
Inverses mod N	Discrete logarithm
Exponentiation mod N	Square roots mod composite N

Figure 3.8: Comparison of problems with known efficient algorithms and those without [72].

Definition 2.5: Known & Chosen Plaintext Attacks

Known Plaintext Attack (KPA):
The adversary has access to one or more known unencrypted and encrpyted message pairs.

Chosen Plaintext Attack (CPA):
The adversary encrypts plaintext of their choosing to analyze the corresponding ciphertexts.

The Caesar Cipher and Vigenère Cipher are both vulnerable to plaintext attacks. The One-Time Pad (OTP), becomes vulnerable if the key is small or reused.

Definition 2.6: Block Ciphers

A cryptographic scheme that separates and encrypts fixed-length blocks of plaintext into ciphertext. Let β (beta) be the block size, and λ the key length. Then for a set of B blocks and message M: $\sum_{b \in B} \beta = \|M\|$ (the sum of all blocks equals the length of the message).
We define $\text{Enc}_\lambda(M, \beta)$, $\text{Dec}_\lambda(C, \beta)$, C as the set of ciphertext blocks, s.t.:

$$\text{Enc}_\lambda(M, \beta) \to C : b \in B \mapsto c \in C$$
$$\text{Dec}_\lambda(C, \beta) \to M : c \in C \mapsto b \in B$$

Where $\lambda := \{(b, c), \dots\}$, a dictionary of unique message block to ciphertext pairs (bijection).

The below figure demonstrates use of a block cipher with a block size of 2, using the Electronic Codebook Mode (ECB).

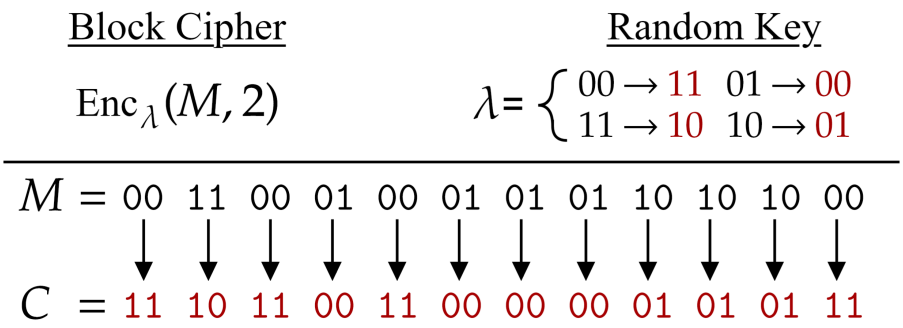


Figure 3.9: Electronic Codebook Mode (ECB) with a block size of 2 [36].

Though in its simplicity falls to the same weakness as the Caesar Cipher, as identical plaintext blocks will encrypt to the same ciphertext block.

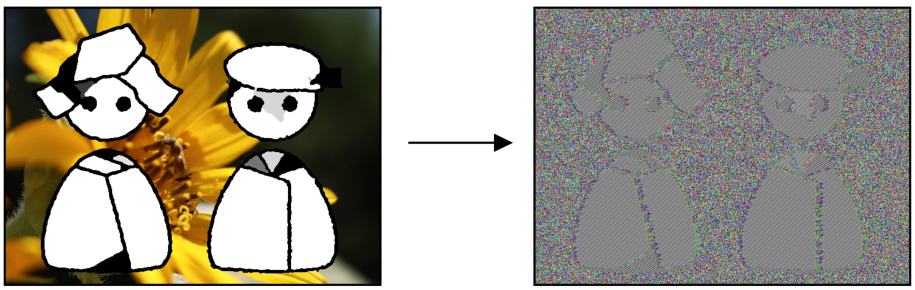


Figure 3.10: An unencrypted image (left) and the same image encrypted with ECB (right).

Here the image on the right is still partially recognizable, as when ECB encountered white space, it encrypted it to the same block. Shockingly the popular video conferencing software Zoom used ECB during the 2020 COVID-19 pandemic, of which now has been patched.

Definition 2.7: Cipher Block Chaining (CBC)

CBC encrypts blocks of plaintext into ciphertext. CBC uses a **Initialization Vector (IV)** to start the chain of XORing. The IV is XORed with the first plaintext block. Each XOR is indexed into the key value pair dictionary λ. The result is the cypher text block. Then the ciphertext block XORs with the next plaintext block and so on. [34]

Security Definition: Confidentiality. Integrity and Authenticity are not guaranteed.

The below figure demonstrates use of a block cipher with the Cipher Block Chaining Mode (CBC).

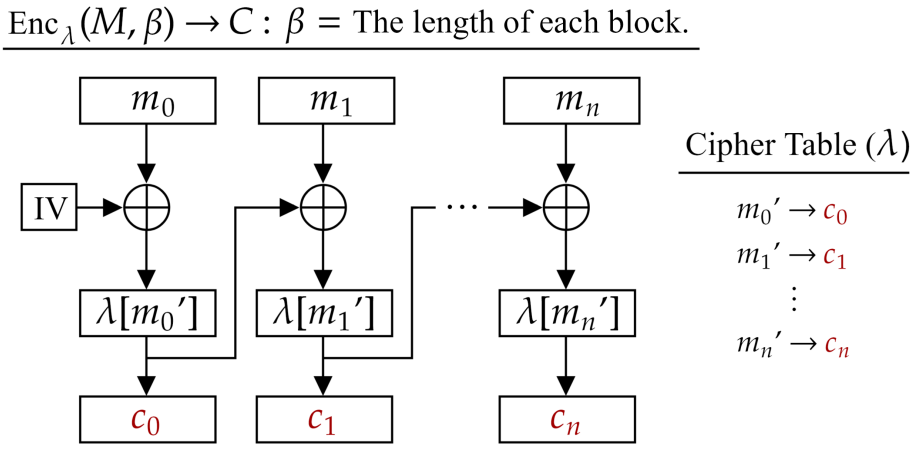


Figure 3.11: A block cipher utilizing the Cipher Block Chaining Mode (CBC) method. [36].

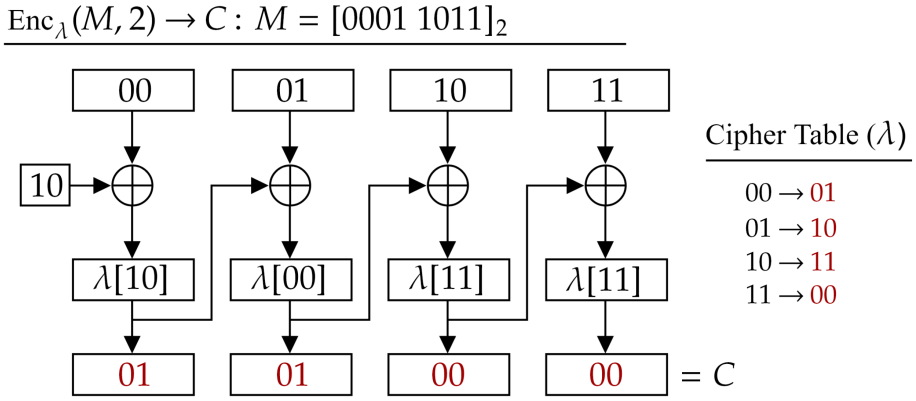


Figure 3.12: CBC encryption example with input $[0001\ 1011]_2$ and outputs $[0101\ 0000]_2$.

Plenty of block cyphers elaborate on these concepts. The different flavors are called a **mode of operation**.

Definition 2.8: Message Authentication Code (MAC)

A MAC combines a message with a secret key before hashing. Let M be the message, λ the key, and the function $T \leftarrow \text{Enc}_\lambda(M)$. Where T is a resulting tag, sometimes called a **digest** or **hash**. Then (M, T) is sent over the wire. The receiver also has λ and runs $\text{Enc}_\lambda(M)$ to verify T.

Security Definition: Integrity and Authenticity. Not confidentiality.

The following figure demonstrates that a MAC protects integrity, as if the message were altered, the receiver would not get the same tag with their key. Though an adversary may still intercept and read the message.

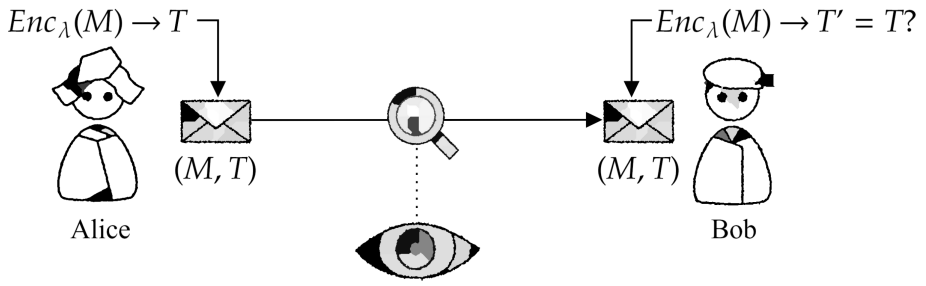


Figure 3.13: A MAC protecting integrity.

Theorem 2.5: Replay Attack

A replay attack is when an adversary intercepts a message and resends it to the receiver. The receiver may not know the message was sent twice.

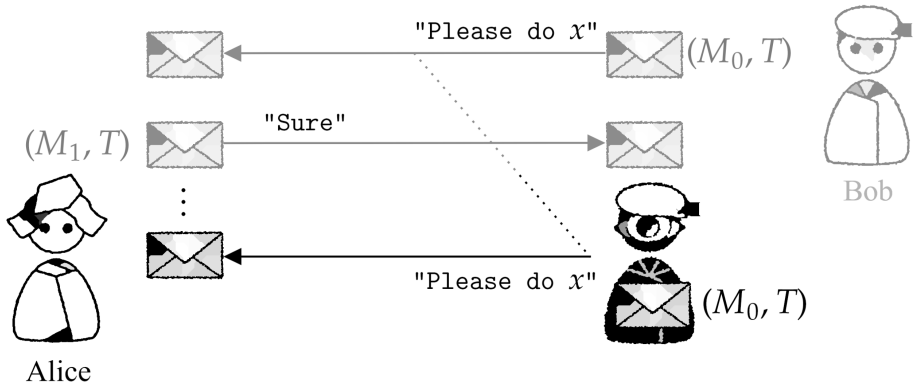


Figure 3.14: A replay attack, where the adversary intercepts and resends a message.

Theorem 2.6: Replay Attack Prevention

To prevent replay attacks, a timestamp or nonce (number used once) is added to the message. The receiver checks the timestamp or nonce to ensure the message is fresh.

Definition 2.9: Hashed-based Message Authentication Code (HMAC)

HMACs are a type of MAC which are standardized and deemed secure. They take a predefined hash function (e.g., SHA-256) and apply it to a MAC. I.e., an HMAC is a specific recipe for a MAC. [74]

3.3 Block Ciphers & Modes of Operation

Definition 3.1: Galios Counter Mode (GCM)

GCM uses a MAC called GMAC, which is a variant of the HMAC. It encrypts data with a desired Encryption Algorithm and then GMACs (MACs) the data into cipher text. The final hash is the tag. This process goes as follows:

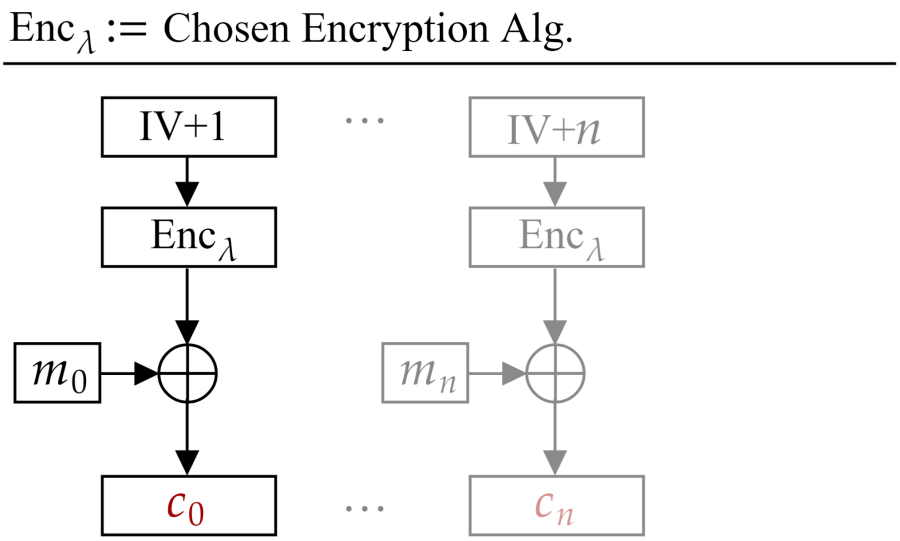
- **Encryption**:
  1. Each block of plaintext is XORed with the encryption key.
  2. To ensure randomness, a counter is added to each encryption before XORing.
  3. To ensure randomness of the counter, an IV is added to it.
- **MAC**:
  1. After encryption, the data is XORed with a previous hash then GMACed.
  2. The GMAC is then used to XOR the next block's cipher before hashing again.
  3. To start the chain of XORing, a 128-bits of zeros is encrypted and GMACed.
  4. After the final block, the length of the message is XORed with the prior hash, then GMACed.
  5. Finally, an encryption of 32-bits of zeros is XORed with the final hash, producing the tag.

GCM also supports **Authentication of Additional Data (AAD)**, which is data that is not encrypted but is still hashed. So in addition to authenticating and encrypting a message, GCM can also authenticate additional unencrypted data. If GCM is only used for authentication, it is called **GMAC**. [80]

Security Definition: Confidentiality, Integrity, and Authenticity.

GCM Diagram and Elaborations on the next page.

The following figures are first broken up and then combined for clarity.



1em

Figure 3.15: GCM IV and Counter XORing with Plaintext to create Ciphertext.

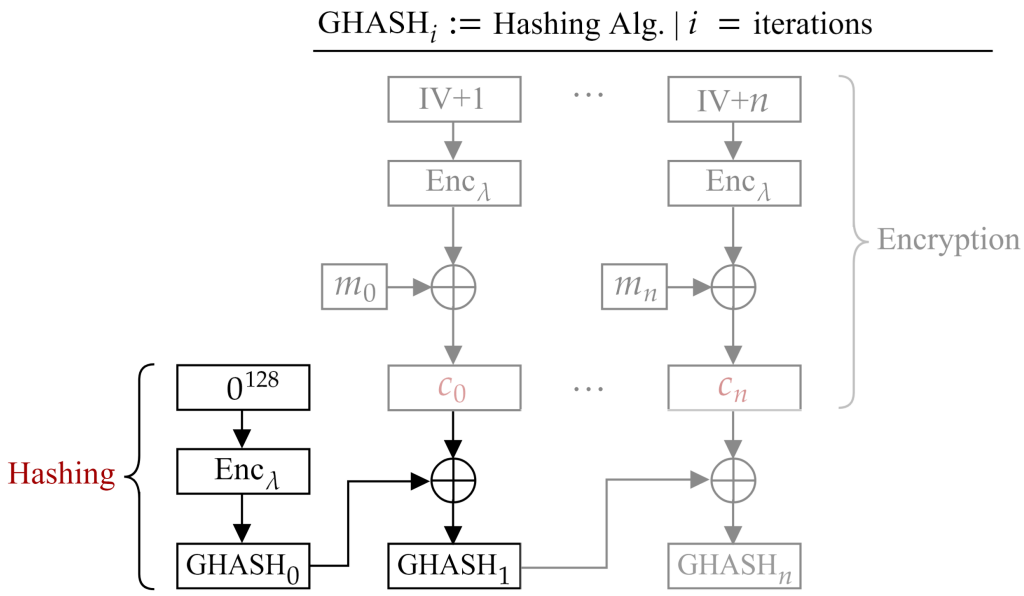


Figure 3.16: 128-bits of zeros is encrypted and GMACed, starting the chain of XOR hashing.

Continued on the next page.

To finish the chain of XOR hashing:

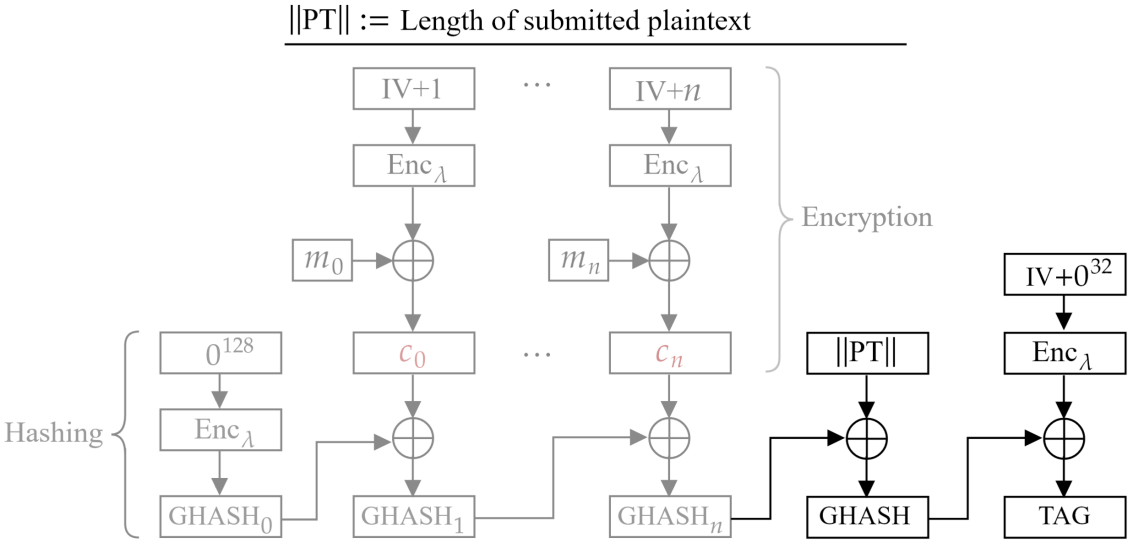


Figure 3.17: The length of the message is XORed with the prior hash, then GMACed with 32-bits of encrypted zeros concatenated with the IV.

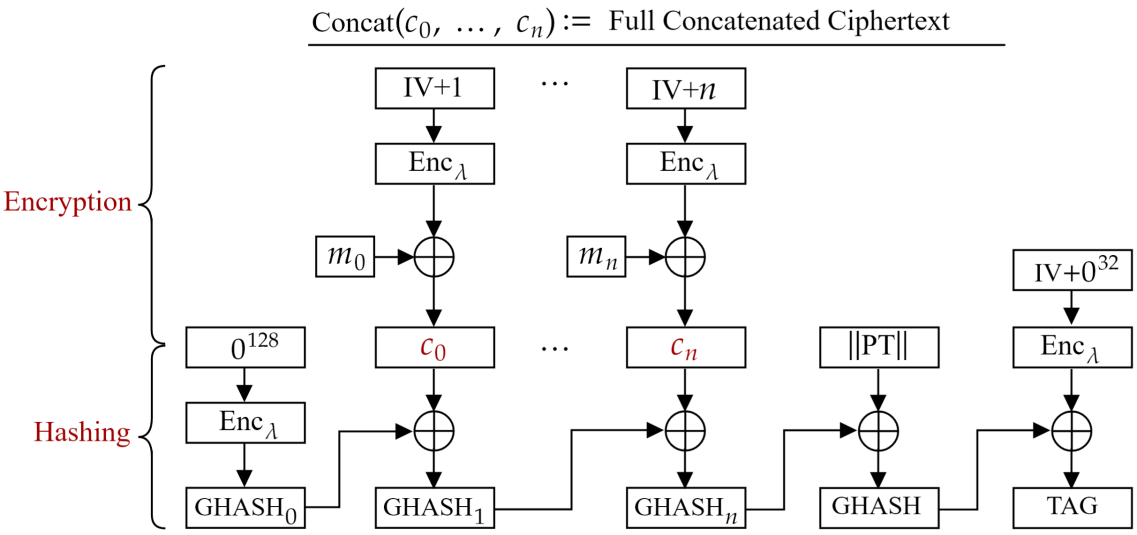


Figure 3.18: A semi complete GCM diagram (Encryption and Authentication), with AAD left out.

Continued on the next page.

A full GCM diagram with AAD:

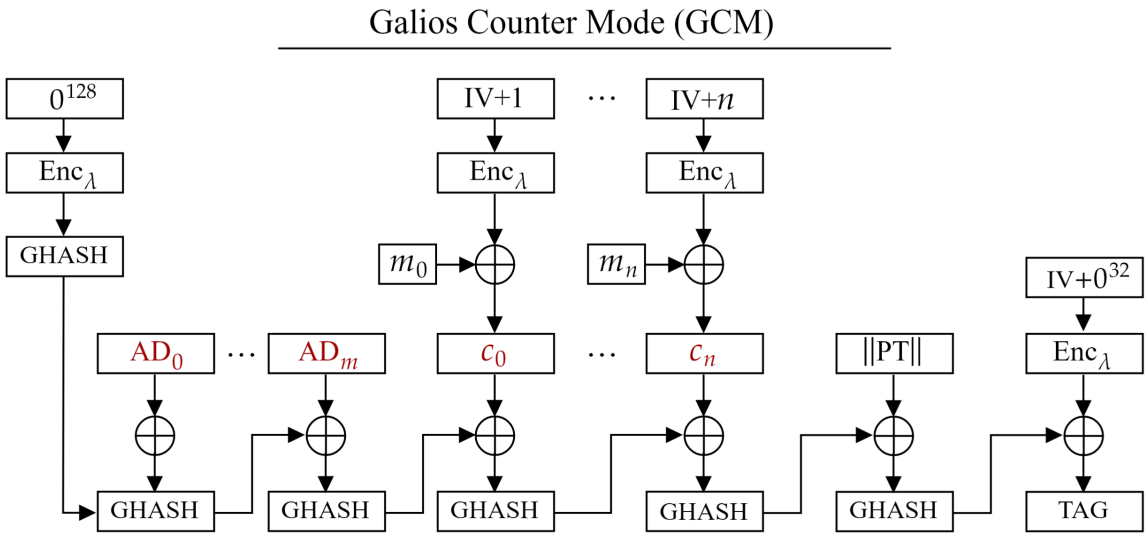


Figure 3.19: A full GCM diagram: Encryption, Authentication, and unencrypted Additional Authenticated Data (AAD).

Now to discuss a possible shared key (symmetric key) algorithm usable Enc$_\lambda$ of GCM.

Definition 3.2: Advanced Encryption Standard (AES)

During a worldwide competition in 2001, the National Institute of Standards and Technology (NIST) selected the Rijndael algorithm as the Advanced Encryption Standard (AES). AES is a symmetric key block cipher that encrypts plaintext (PT) in 128-bit blocks. AES works in rounds permuting the PT with partial keys generated from the initial key.

- **Key Expansion**: An initial key is expanded into a key schedule. These keys will be assigned to each round. The rounds needed depend on the initial key size:
  1. 128-bit key: 10 rounds, 11 keys.
  2. 192-bit key: 12 rounds, 13 keys.
  3. 256-bit key: 14 rounds, 15 keys.
- **Input Transformation**: The PT is transformed into a 4×4 matrix, called the **State**. The state undergoes four main operations per round:
  1. **SubBytes**: Each byte is substituted with a value from the S-Box.
  2. **ShiftRows**: Each row is shifted left by an offset.
  3. **MixColumns**: Each column is mixed with a fixed matrix.
  4. **AddRoundKey**: Each byte in the State is XORed with a sub-key. [16] [58] [3]

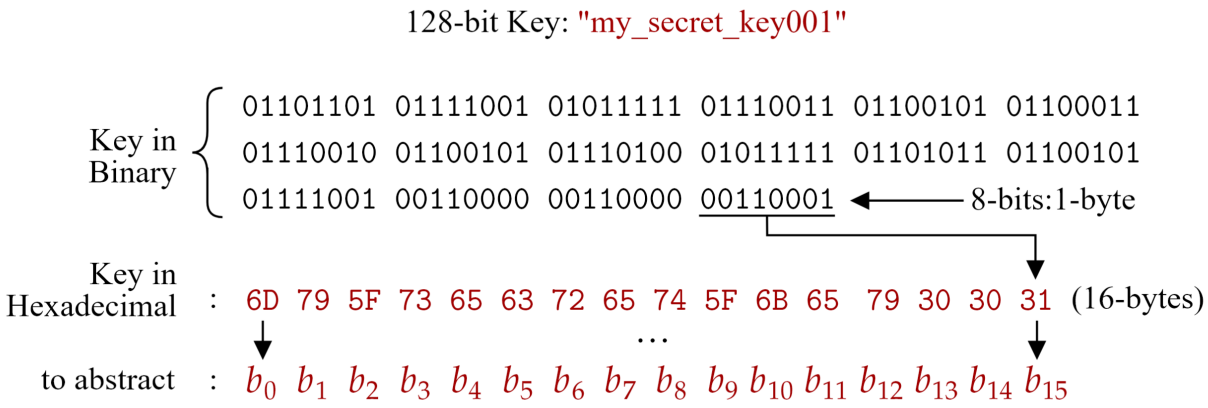
Definition 3.3: Key Expansion

Key expansion, an AES process which takes a single key and expands it into multiple keys. The initial key is broken up into 16-byte 4×4 matrices. Each column a **Word** (32-bits). The process follows four main steps:

RotWord→SubWord→Rcon→XOR.
(Rotate Word, Substitute Word, Round Constant, XOR)

This generates a **Sub-Key** for one round. Each round generates for the next round.

AES Key Expansion: Given the an initial 128-bit key, λ_0 "my_secret_key001":



Key represented an a 4×4 matrix

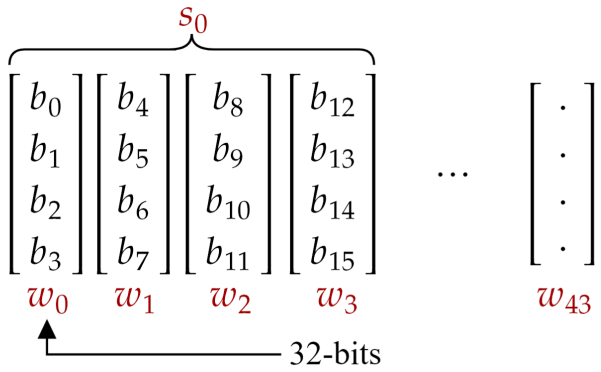
$$\begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{13} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

The Key is broken up into Words

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \begin{bmatrix} b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \begin{bmatrix} b_8 \\ b_9 \\ b_{10} \\ b_{11} \end{bmatrix} \begin{bmatrix} b_{12} \\ b_{13} \\ b_{14} \\ b_{15} \end{bmatrix}$$
$$w_0 \quad w_1 \quad w_2 \quad w_3$$

These Words w_0, w_1, w_2, w_3 are the initial key. This will then generate the rest of the Words needed. This first group is the first sub-key s_0 for the first round.

This is a Subkey (s_0), used for a round. The number of rounds depend on the initial Key.



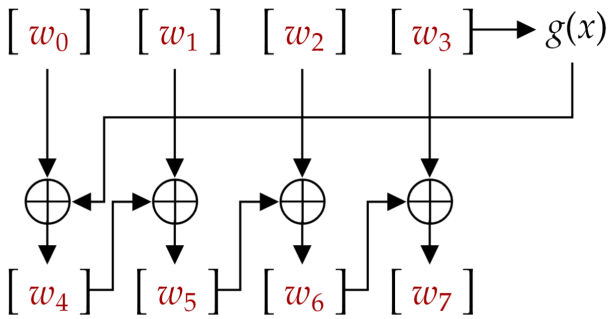
Round Dependence:

- 128-bit key: 10 rounds, 11 keys.
- 192-bit key: 12 rounds, 13 keys.
- 256-bit key: 14 rounds, 15 keys.

Hence, $4 \times 11 = 44$ Words

Rounds depend on $\|\lambda_0\|$: 128-bits $\rightarrow$ 10 rounds, 192-bits $\rightarrow$ 12 rounds, 256-bits $\rightarrow$ 14 rounds.

There need be a total of 44 Words. Expansion is used to find the rest:



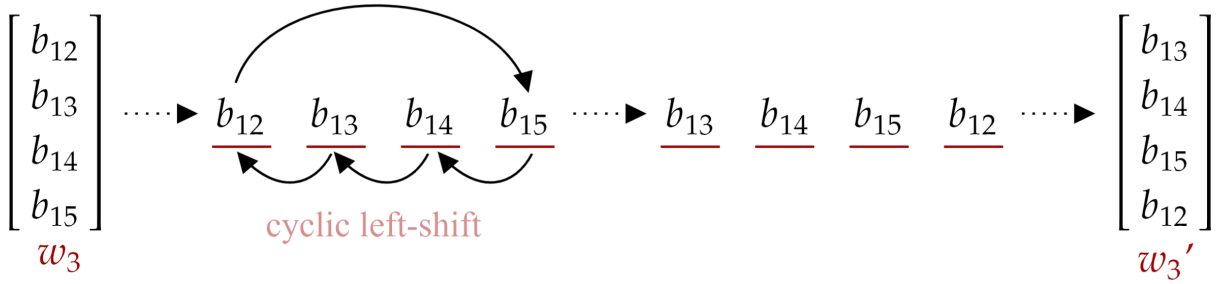
Here s_0 generates s_1

And so forth s_i generates s_{i+1}
Until all Words are satisfied

And $g(x)$ some permuting function

The $g(x) := RotWord \circ SubWord \circ Rcon \circ XOR$, generating the next sub-key s_1.

Here, $g(x)$ performs a cyclic left-shift on w_3; Often called $RotWord(x)$ (rotate Word).



Definition 3.4: Nibble

A **nibble** is a four-bit aggregation, referring to "half a **byte**." A nibble splits 8-bits into two 4-bit values, the **upper nibble** and the **lower nibble**. E.g., given the hex value $0A$, the upper and lower nibble is 0 and A respectively.

Now to quickly introduce some concepts needed for the AES borrowed from group theory.

Definition 3.5: Field

A **field** is a set of elements equipped with two operations, addition and multiplication, such that the following properties hold:

- **Closure**: Adding or multiplying any two elements in the set remains in the set.
- **Associativity**: Addition and multiplication are associative.
- **Commutativity**: Addition and multiplication are commutative.
- **Identities**: There exist identity elements for addition (0) and multiplication (1).
- **Inverses**: Every element has an additive inverse (negative) and, except for 0, a multiplicative inverse (reciprocal).
- **Distributivity**: Multiplication distributes over addition.

E.g., The set of real numbers $\mathbb{R}$, with standard addition and multiplication, forms a field.

Theorem 3.1: Abel Ruffini Theorem

There is no general for solving polynomials of degree five or higher. Moreover, no formula is possible using these finite amount of: $+, -, \times, \div, \sqrt[n]{x}$.

This will play a key part in the AES algorithm, stopping attackers from reversing the encryption process.

Definition 3.6: Galois Field $GF(2^8)$

The Galois Field $GF(2^8)$ is a finite field consisting of $2^8 = 256$ elements, where each element is an 8-bit binary value (a byte). Addition and multiplication in $GF(2^8)$ are defined modulo an irreducible polynomial of degree 8 over $GF(2)$ (the binary field).
In AES, the irreducible polynomial used is:

$$p(x) = x^8 + x^4 + x^3 + x + 1.$$

Tip: To learn more consider this video on Galois Theory: "Why is there no quintic formula?"

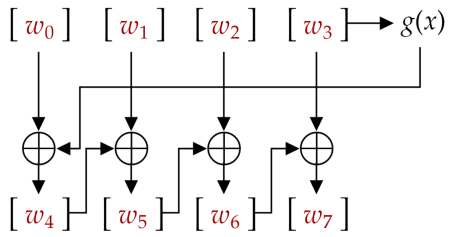
Definition 3.7: AES S-Box

The AES S-Box is a substitution box used in the AES algorithm. It is a 16×16 matrix of 8-bit values. The first column and row represent the upper and lower nibble of the input byte. E.g., given $C7$, column $c0$ and row 07 intersect $c6$.

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08	ba
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	89	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

For reference, previous figures are shown again.

There need be a total of 44 Words. Expansion is used to find the rest:

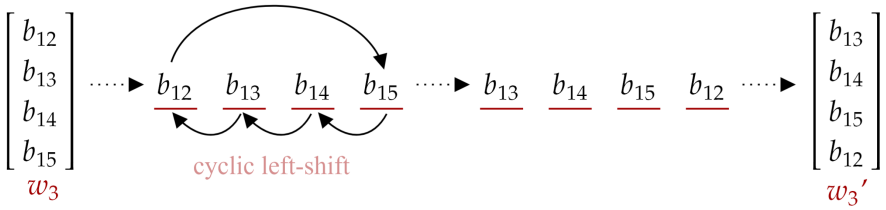


Here s_0 generates s_1

And so forth s_i generates s_{i+1}
Until all Words are satisfied

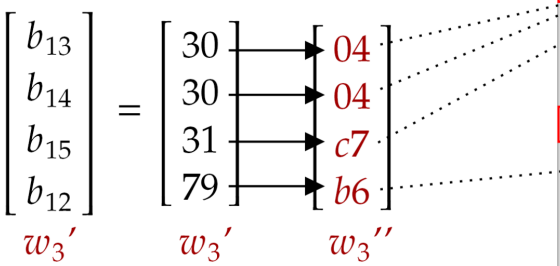
And $g(x)$ some permuting function

Here, $g(x)$ performs a cyclic left-shift on w_3; Often called $RotWord(x)$ (rotate Word).



Next $SubWord(x)$ takes the Hex digit places of b_i to index the **AES S-Box**

Key Hex : 6D 79 5F 73 65 63 72 65 74 5F 6B 65 79 30 30 31



	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
10	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
20	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
30	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
40	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
50	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
60	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
70	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
80	cd	0e	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
90	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a0	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b0	e7	c8	37	6d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08	ba
c0	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d0	70	3e	b5	66	48	03	f6	0e	61	35	57	89	86	c1	1d	9e
e0	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f0	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Definition 3.8: Round Constants

Round constants take the output of $SubWord(x)$ and XOR the first byte with a constant value. The constant value, denoted $Rcon[j]$, is derived from powers of 2 in the Galois Field $GF(2^8)$, where the irreducible polynomial is $x^8 + x^4 + x^3 + x + 1$.

Each round constant ensures the uniqueness of round keys during the AES key expansion process, introducing nonlinearity and breaking symmetry between rounds.

Round (i)	*Rcon* Value	Power of 2 in GF(2^8)
1	0x01 00 00 00	$2^0 = 1$
2	0x02 00 00 00	$2^1 = 2$
3	0x04 00 00 00	$2^2 = 4$
4	0x08 00 00 00	$2^3 = 8$
5	0x10 00 00 00	$2^4 = 16$
6	0x20 00 00 00	$2^5 = 32$
7	0x40 00 00 00	$2^6 = 64$
8	0x80 00 00 00	$2^7 = 128$
9	0x1B 00 00 00	2^8 mod poly
10	0x36 00 00 00	2^9 mod poly

Table 3.1: Round Constants, where poly:= $x^8 + x^4 + x^3 + x + 1$.

Finally w_3'' is XORed with it's respective round constant $Rcon[j]$ from $GF(2^8)$.

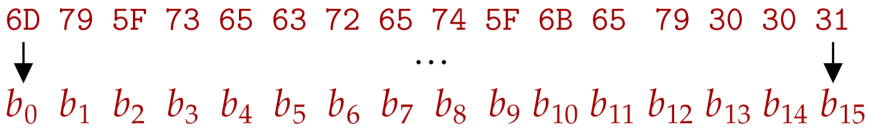
$$
\begin{aligned}
w_3'' &= 0000\ 0100\ 0000\ 0100\ 1100\ 0111\ 1011\ 0110 \\
Rcon[1] &= 0000\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \\
\hline
w_3'' \oplus Rcon[1] &= 0000\ 0101\ 0000\ 0100\ 1100\ 0111\ 1011\ 0110
\end{aligned}
\quad \rightarrow \texttt{05 04 C7 B6}
$$

$$
= \begin{bmatrix} 05 \\ 04 \\ C7 \\ B6 \end{bmatrix}
$$

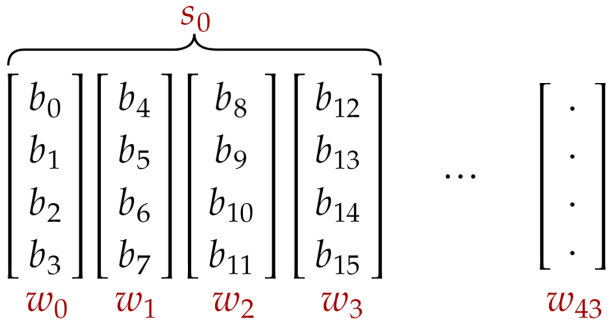
$g(w_3)$

All together:

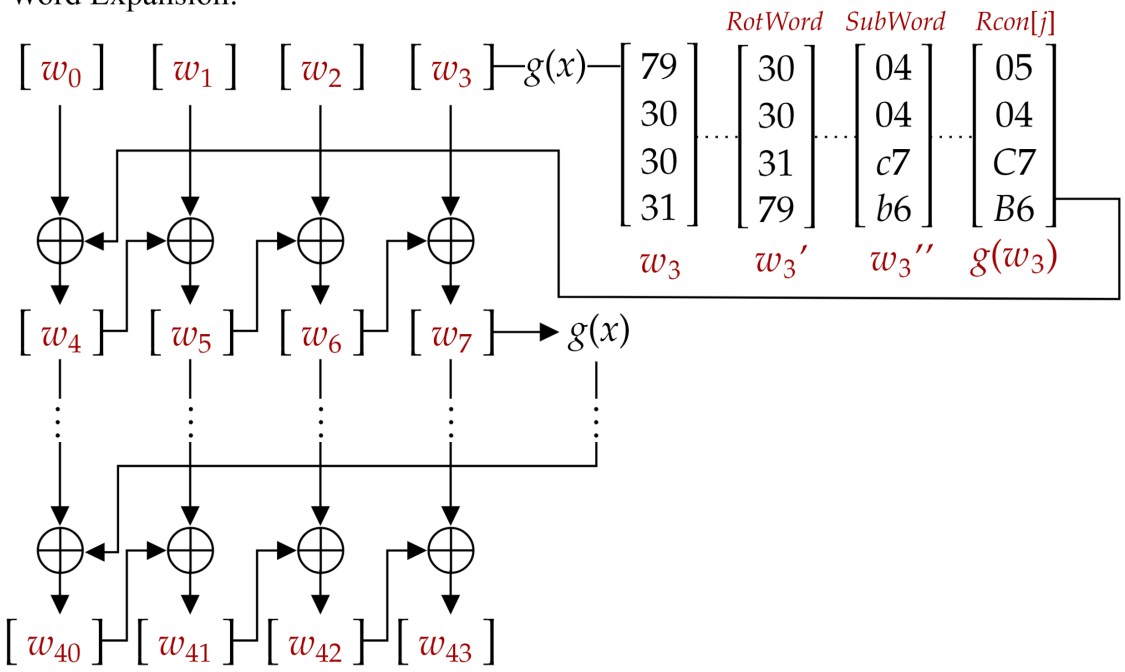
AES Key Expansion of a 128-bit Key: "my_secret_key001". First, Hex conversion:



Word Aggregation into Subkeys:



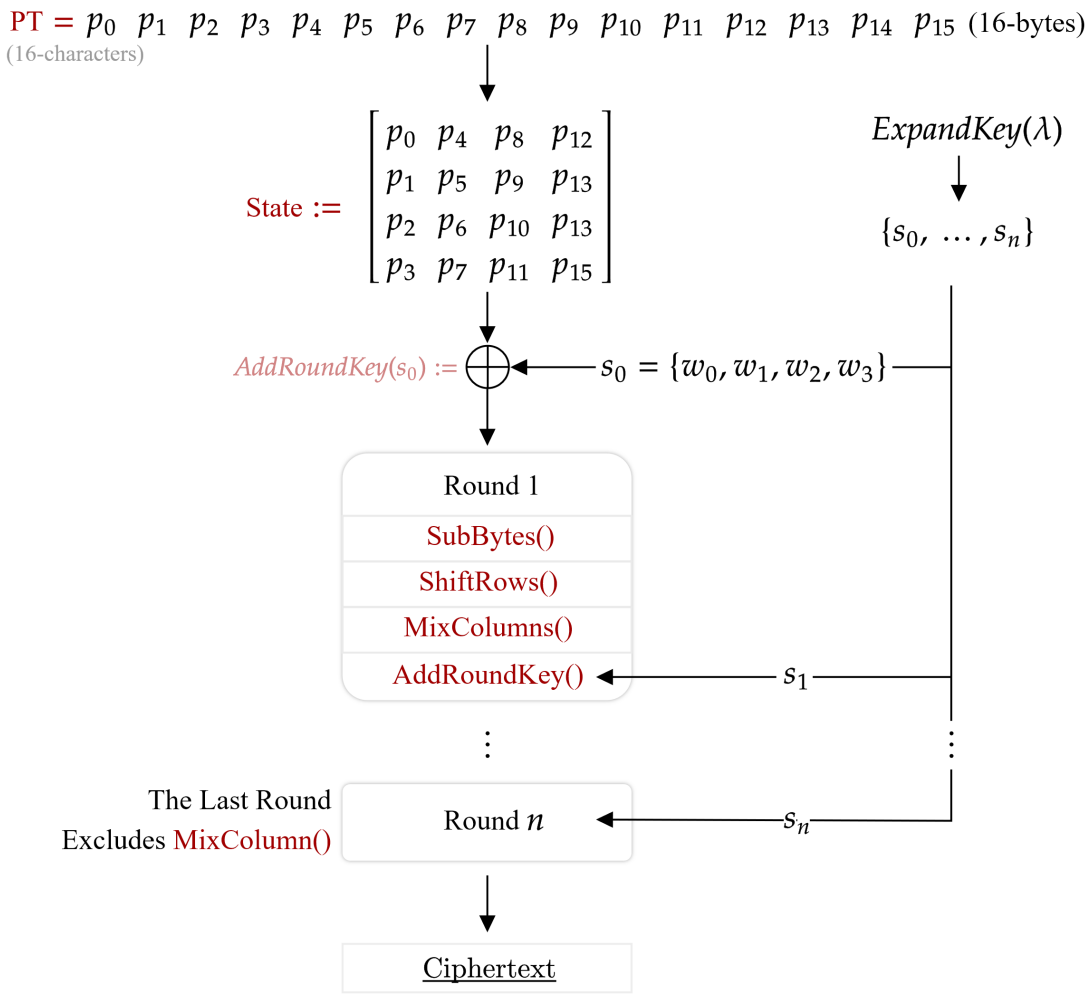
Word Expansion:



This is AES Key Expansion.

Now to discuss the AES encryption process.

The Four main steps in AES Encryption of a 128-bit Plaintext (PT) with a 128-bit key (λ).



First the PT is transformed into a 4×4 matrix, called the **State**. The initial transformation is the **AddRoundKey** operation, which XORs each column of the state with sub-key $s_0 = \{w_0, w_1, w_2, w_3\}$. I.e., $[p_0, p_1, p_2, p_3] \oplus [w_0]$ and so on. Recall $w_0 = [b_0, b_1, b_2, b_3]$ for some byte b_i of the initial key.

To elaborate on the AddRoundKey operation:

The AddRoundKey() XORs each column c_i with a sub-key Word w_i

PT = "plain_text_mssg0" Hex = `70 6C 61 69 6E 5F 74 65 78 74 5F 6D 73 73 67 30`

$$\text{State} := \begin{bmatrix} c_0 & c_1 & c_2 & c_3 \\ 70 & 6E & 78 & 73 \\ 6C & 5F & 74 & 73 \\ 61 & 74 & 5F & 67 \\ 69 & 65 & 6D & 30 \end{bmatrix} \oplus \underbrace{\begin{bmatrix} w_0 & w_1 & w_2 & w_3 \\ b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{13} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}}_{s_0} = \begin{bmatrix} p_0' & p_4' & p_8' & p_{12}' \\ p_1' & p_5' & p_9' & p_{13}' \\ p_2' & p_6' & p_{10}' & p_{13}' \\ p_3' & p_7' & p_{11}' & p_{15}' \end{bmatrix}$$

$$\begin{matrix} 6E \oplus b_4 \\ \vdots \\ 65 \oplus b_7 \end{matrix}$$

AddRoundKey takes a sub-key s_i and XORs each Word w_i with the corresponding column c_i of the State.

The SubBytes() performs an **AES S-Box** substitution with every byte of the State.

$$\begin{bmatrix} p_0' & p_4' & p_8' & p_{12}' \\ p_1' & p_5' & p_9' & p_{13}' \\ p_2' & p_6' & p_{10}' & p_{13}' \\ p_3' & p_7' & p_{11}' & p_{15}' \end{bmatrix} \rightarrow \boxed{\text{AES S-Box}} \rightarrow \begin{bmatrix} p_0'' & p_4'' & p_8'' & p_{12}'' \\ p_1'' & p_5'' & p_9'' & p_{13}'' \\ p_2'' & p_6'' & p_{10}'' & p_{13}'' \\ p_3'' & p_7'' & p_{11}'' & p_{15}'' \end{bmatrix}$$

Just like in Key Expansion's (3.3) use of the S-Box on the last word of each sub-key. The PT undergoes a full substitution with the S-Box.

The ShiftRows() a fixed number of left-shifts depending on the row r_i of the State.

$S(r_i, x) :=$ shifts row r_i to the left by x places (for ease of notation)

$$\begin{bmatrix} p_0'' & p_4'' & p_8'' & p_{12}'' \\ p_1'' & p_5'' & p_9'' & p_{13}'' \\ p_2'' & p_6'' & p_{10}'' & p_{13}'' \\ p_3'' & p_7'' & p_{11}'' & p_{15}'' \end{bmatrix} \begin{matrix} \\ S(r_0,0) \\ S(r_1,1) \\ S(r_2,2) \\ S(r_3,3) \end{matrix} \rightarrow \begin{bmatrix} p_0'' & p_4'' & p_8'' & p_{12}'' \\ p_5'' & p_9'' & p_{13}'' & p_1'' \\ p_{10}'' & p_{13}'' & p_2'' & p_6'' \\ p_{15}'' & p_3'' & p_7'' & p_{11}'' \end{bmatrix}$$

Below, the notation {01} refers to a hex value.

In MixColumns(), Columns c_i are considered four-term polynomials of $GF(2^8)$ and multiplied by polynomial $a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$

$$\vdots$$

$$= \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \quad \text{(polynomial matrix representation)}$$

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} p_0'' \\ p_5'' \\ p_{10}'' \\ p_{15}'' \end{bmatrix} = \begin{bmatrix} p_0''' \\ p_5''' \\ p_{10}''' \\ p_{15}''' \end{bmatrix}$$

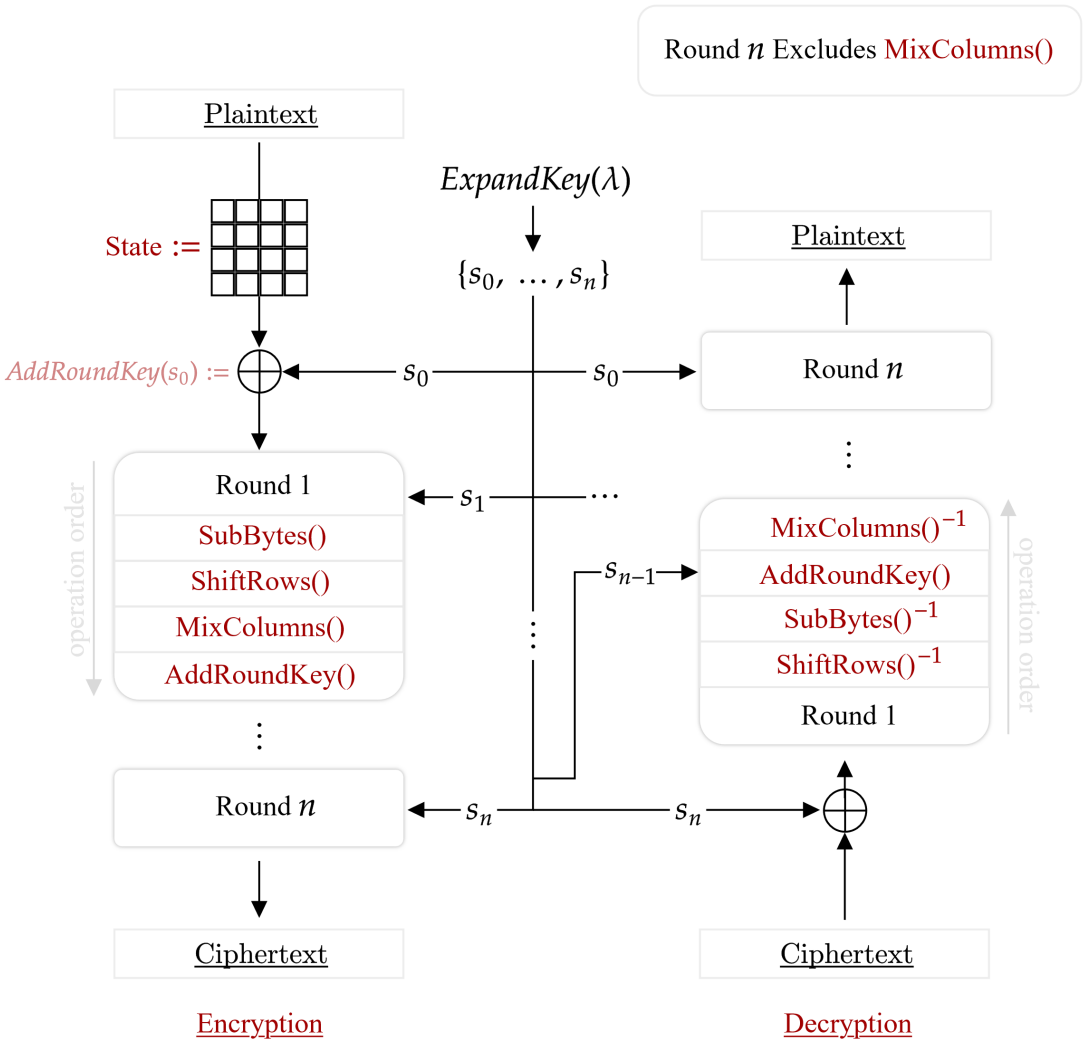
$$\vdots$$

$$p_0''' = (\{02\} \cdot p_0'') \oplus (\{03\} \cdot p_5'') \oplus p_{10}'' \oplus p_{15}''$$

$G(2^8)$, $a + b := a \oplus b$ (XOR)

$$\begin{bmatrix} p_0'' & p_4'' & p_8'' & p_{12}'' \\ p_5'' & p_9'' & p_{13}'' & p_1'' \\ p_{10}'' & p_{13}'' & p_2'' & p_6'' \\ p_{15}'' & p_3'' & p_7'' & p_{11}'' \end{bmatrix} \text{MixColumns()} \rightarrow \begin{bmatrix} p_0''' & p_4''' & p_8''' & p_{12}''' \\ p_5''' & p_9''' & p_{13}''' & p_1''' \\ p_{10}''' & p_{13}''' & p_2''' & p_6''' \\ p_{15}''' & p_3''' & p_7''' & p_{11}''' \end{bmatrix}$$

This is the AES Encryption & Decryption.



For Decryption, the last scheduled key for Encryption is used as the initial transformation. Note the inverse operations of Decryption follow a different order than Encryption. This covers the basics of AES.

3.4 Establishing Shared Secret & Communication

This section will discuss how to establish a shared key between two parties, and begin communication over TLS. The following content expects some knowledge of modular arithmetic. Though it is not necessary as the concepts will be explained in detail enough to follow along.

Definition 4.1: Discrete Logarithm Problem

The **Discrete Logarithm Problem (DLP)** is the challenge of determining the exponent x given modular congruence:

$$g^x \equiv h \pmod{p},$$

where $g, h \in \mathbb{Z}_p^*$, and p a prime number. Computing $g^x \bmod p$ (exponentiation) is efficient, reversing this process to find x is computationally infeasible for large primes. [42]

If Alice and Bob wanted to establish a shared connection, they could use the following method. Alice first places a lock on her message, and sends it to Bob. Bob then places his lock on the message, and sends it back to Alice. Alice then removes her lock, and sends it back to Bob. Bob then removes his lock, and reads the message.

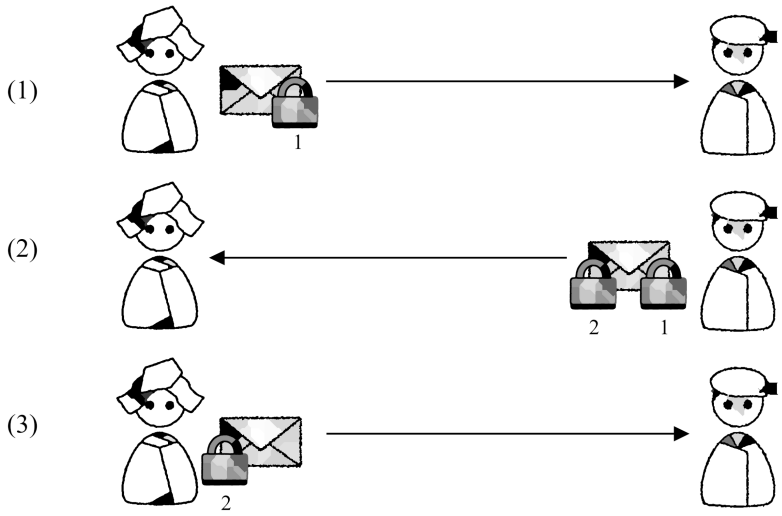


Figure 3.20: Alice and Bob send a secure message via alternating locks.

This is the basic idea behind the following method.

Definition 4.2: Diffie-Hellman Key Exchange (DHE)

def:dh The **Diffie-Hellman Key Exchange** is a method for two parties A and B to establish a shared secret over an insecure channel. The method is as follows:

1. A and B agree on a prime number p and a generator g.
2. Both A and B select exponents a and b (secret key), and computes $\alpha = g^a \bmod p$ and $\beta = g^b \bmod p$ respectively.
3. A and B exchange α (alpha) and β (beta).
4. A computes $\lambda = \beta^a \bmod p = (g^b)^a \bmod p$.
5. B computes $\lambda = \alpha^b \bmod p = (g^a)^b \bmod p$.

Both A and B establish a shared secret λ without exchanging their secret keys. [42]

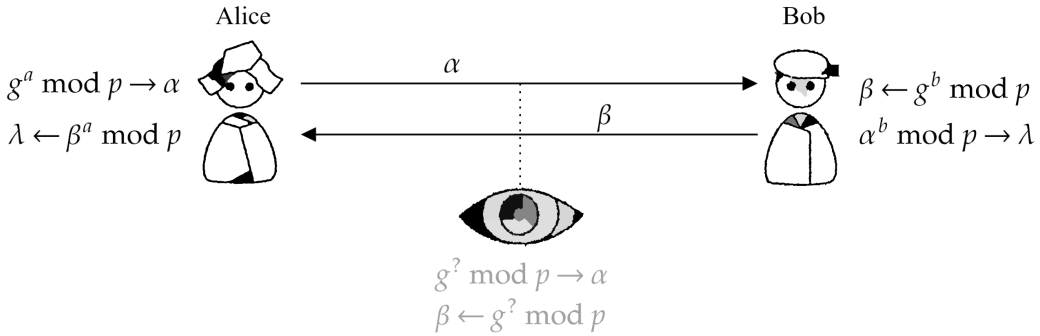


Figure 3.21: Diffie-Hellman Key Exchange protects against eavesdropping.

However,

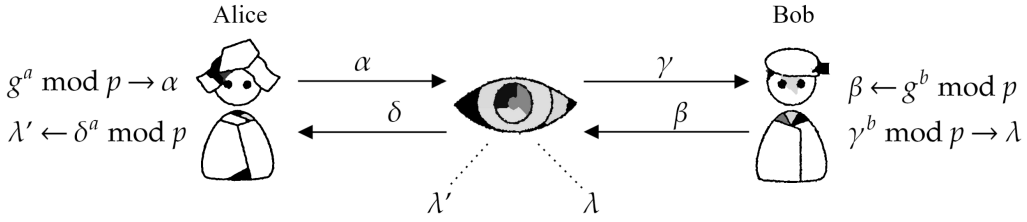


Figure 3.22: Diffie-Hellman Key Exchange Attack via MITM.

If Alice and Bob were purely trying to connect on their own, there would be no way to verify that they are who they say they are.

Though this can be fixed by a previous topic discussed, **Public Key Infrastructure (PKI)**. If Alice and Bob had a certificate from a trusted third party, they could verify each other's identity.

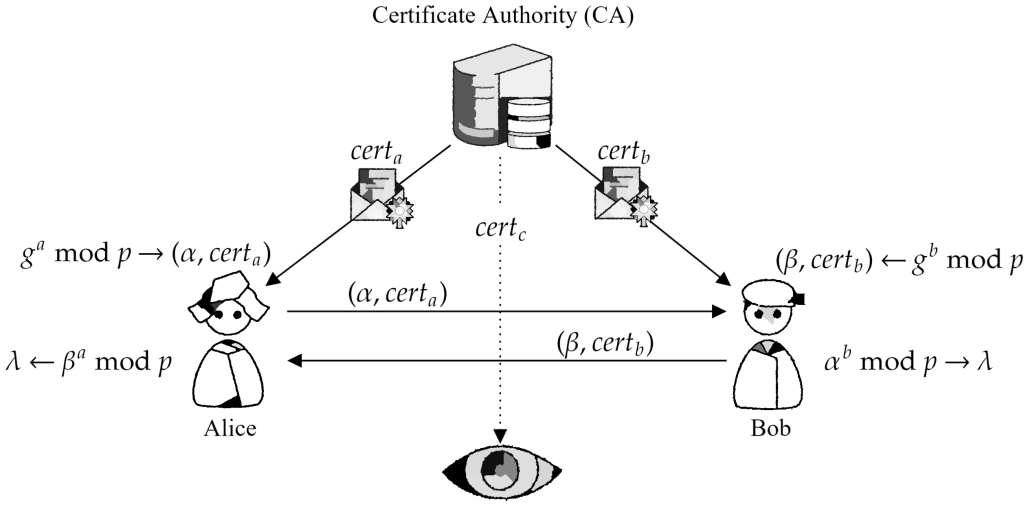


Figure 3.23: Diffie-Hellman Key Exchange with PKI.

Here Alice and Bob previously made a CSR (Certificate Signing Request) to a CA (Certificate Authority) to get a certificate. Here the CA—a trusted entity that predominately come pre-baked into OSs and Web Browsers—signs Alice and Bob's certificate. This way, Alice and Bob may verify each other's identity via signing their data and sharing a public key.

An adversary cannot forge a signature without the certificates' private keys, and cannot forge a certificate as only CA's can issue them via their own private key. Recall certificates come with identifying information of who the certificate belongs to and the issuer.

Theorem 4.1: Authenticated Diffie-Hellman Key Exchange

The **Authenticated Diffie-Hellman Key Exchange** is a method for two parties A and B to establish a shared secret over an insecure channel, with the help of a trusted third party. Before performing the Diffie-Hellman Key Exchange (DHE), A and B obtain certificates from a Certificate Authority (CA) via a Certificate Signing Request (CSR). Once the certificates are obtained, both parties can perform DHE via signing and verifying data sent. [42]

Theorem 4.2: Post DHE Data Exchange

After DHE is complete between parties A and B, they may now communicate via symmetric encryption. Moreover, they can use MACs to ensure data integrity, or in particular, AES in GCM providing encryption.

3.5 Securing Certificates

Definition 5.1: Euler's Totient Function

The **Euler's Totient Function** $\phi(n)$ is the number of positive integers less than n that are coprime to n. For a prime number p, $\phi(p) = p - 1$. For two distinct prime numbers p and q, $\phi(p \cdot q) = (p-1)(q-1)$. For a prime power p^k, $\phi(p^k) = p^{k-1}(p-1)$.

Theorem 5.1: Modular Multiplicative Inverse

The **Modular Multiplicative Inverse** of an integer (a modulo m) is an integer x, s.t., $a \cdot x \equiv 1 (\text{mod } m)$. Inverse exists if and only if $\gcd(a, m) = 1$ (coprime).

Denoted: $a^{-1} \iff a \cdot a^{-1} \equiv 1 (\text{mod } m)$.

Definition 5.2: Rivest-Shamir-Adleman (RSA)

The **RSA** algorithm is a public-key cryptosystem. First described in 1977 by Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman at MIT. The RSA algorithm involves three steps: key generation, encryption, and decryption. It goes as follows:

1. **Key Generation:**
   a) Select two large prime numbers p and q.
   b) Compute $n = p \cdot q$.
   c) Compute $\phi(n) = (p-1)(q-1)$.
   d) Select an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$.
   e) Compute d as $d \equiv e^{-1} \bmod \phi(n)$.
   f) The public key is (n, e) and the private key is (n, d).
2. **Encryption:**
   a) To encrypt a message m, compute $c \equiv m^e (\bmod\ n)$.
3. **Decryption:**
   a) To decrypt a ciphertext c, compute $m \equiv c^d (\bmod\ n)$.

For emphasis:

$$(m^e)^d \equiv m^{ed} \equiv m^1 (\bmod\ n)$$

, The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. [71]

Example 5.1: RSA Encryption and Decryption

The following is an example of the RSA algorithm, including key generation, encryption, and decryption. Smaller primes are chosen for brevity:

1. **Key Generation:**
   a) Choose two distinct prime numbers: $p = 61$ and $q = 53$.
   b) Compute $n = p \cdot q$:
   $$n = 61 \cdot 53 = 3233.$$
   c) Compute the totient function $\phi(n)$ as:
   $$\phi(n) = (p-1)(q-1) = 60 \cdot 52 = 780.$$
   d) Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$:
   $$e = 17.$$
   e) Compute d, the modular multiplicative inverse of e mod $\phi(n)$:
   $$d \equiv e^{-1} \text{mod } \phi(n) \implies d = 413.$$
   f) The public key is $(n, e) = (3233, 17)$, and the private key is $(n, d) = (3233, 413)$.
2. **Encryption:**
   a) To encrypt a plaintext message m, compute:
   $$c \equiv m^e (\text{mod } n).$$
   For $m = 65$, we calculate:
   $$c = 65^{17} \text{mod } 3233 = 2790.$$
3. **Decryption:**
   a) To decrypt a ciphertext c, compute:
   $$m \equiv c^d (\text{mod } n).$$
   For $c = 2790$, we calculate:
   $$m = 2790^{413} \text{mod} 3233 = 65.$$

This example demonstrates the RSA encryption and decryption process, where a message $m = 65$ is encrypted and successfully decrypted back to its original value. ■

Theorem 5.2: RSA Security Definition

RSA provides authenticity of a data source; However,

RSA does not provide confidentiality or integrity.

In this case certificates are meant to be public so confidentiality is not a concern; However, integrity is. For integrity, **HMACs** are used. The hashing algorithm used on certificates is often **SHA-256**.

Definition 5.3: Secure Hash Algorithm 256 (SHA-256)

The **SHA-256** algorithm is a cryptographic hash function belonging to the SHA-2 (Secure Hash Algorithm 2) family, standardized by NIST in 2001. It produces a fixed-length 256-bit hash value from an input message of any size. The algorithm is as follows:

1. **Initialization:**
   a) Define the **message input** as a bit string of arbitrary length.
   b) Pad the message so that its length (in bits) is congruent to 448 mod 512.
   c) Append the 64-bit representation of the original message length to create a message of size divisible by 512 bits.
2. **Message Schedule:**
   a) Divide the padded message into blocks of 512 bits each.
   b) Each block is processed in 64 rounds using a schedule of 32-bit words $W_0, \ldots, W_{63}$, which are derived from the 512-bit block.
3. **Hash Computation:**
   a) Initialize eight 32-bit **hash values** $H_0, H_1, \ldots, H_7$ to predefined constants.
   b) For each message block:
      i. Perform 64 rounds of operations involving:
         - Bitwise logical functions (`AND`, `OR`, `XOR`),
         - Modular addition,
         - Circular shifts and rotations.
4. **Output:**
   a) After processing all blocks, concatenate the final hash values:

$$\text{SHA-256}(M) = H_0 \| H_1 \| H_2 \| H_3 \| H_4 \| H_5 \| H_6 \| H_7.$$

   b) The output is a 256-bit (32-byte) fixed-length hash digest.

Theorem 5.3: Digital Signatures with RSA and SHA-256

Digital signatures are used to verify the authenticity and integrity of a message. Using RSA and SHA-256, the process is as follows:

1. **Hashing:**
   a) The sender computes the SHA-256 hash of the message or certificate:
   $$H = \text{SHA-256}(\text{Message}).$$
2. **Signing:**
   a) The sender encrypts the hash H using their RSA private key to generate the digital signature:
   $$S = H^d \mod n.$$
3. **Transmission:**
   a) The sender sends the message and the digital signature $(\text{Message}, S)$.
4. **Verification:**
   a) The recipient computes the SHA-256 hash of the received message:
   $$H' = \text{SHA-256}(\text{Message}).$$
   b) The recipient decrypts the digital signature S using the sender's RSA public key:
   $$H'' = S^e \mod n.$$
   c) The recipient verifies the signature by checking if:
   $$H' = H''.$$
   If the hashes match, the message is authentic and unaltered.

In Short:

- **Sender:** Specifies an HMAC and signs the digest with their private key.
- **Receiver:** Computes the HMAC and decrypts the signature to see if the hashes match.

Definition 5.4: Brief History of Hashing Algorithms

Hashing algorithms are essential cryptographic tools that map data of arbitrary size to fixed-size hash values. Over time, vulnerabilities in older algorithms have driven the development of more secure alternatives.

MD5 (Message Digest Algorithm 5), introduced in 1991, quickly became popular for checksums and data integrity checks. However, it is considered insecure due to susceptibility to *collision attacks*, where two different inputs produce the same hash.

Similarly, **SHA-1** (Secure Hash Algorithm 1), released by NIST in 1995, generated 160-bit hash values. By the mid-2000s, researchers demonstrated its vulnerability to collisions.

The **SHA-2** family, introduced in 2001, offered stronger 256-bit (SHA-256) and 512-bit (SHA-512) outputs. Despite its robustness, skepticism arose because SHA-2 was designed by the U.S. National Security Agency (NSA), prompting concerns about potential backdoors.

To address these concerns, NIST held a public competition, resulting in the standardization of **SHA-3** in 2015. [28]

3.6 Establishing a Secure Connection HTTPS

Definition 6.1: TLS Handshake Protocol

The **TLS Handshake Protocol** is a key exchange protocol that allows two parties to establish a secure connection. First a **TCP handshake** to establish a connection. Then a "**Client Hello**" message is sent, presenting the following parameters:

- Supported TLS versions, cipher suites.

The server responds with a "**Server Hello**" message, containing:

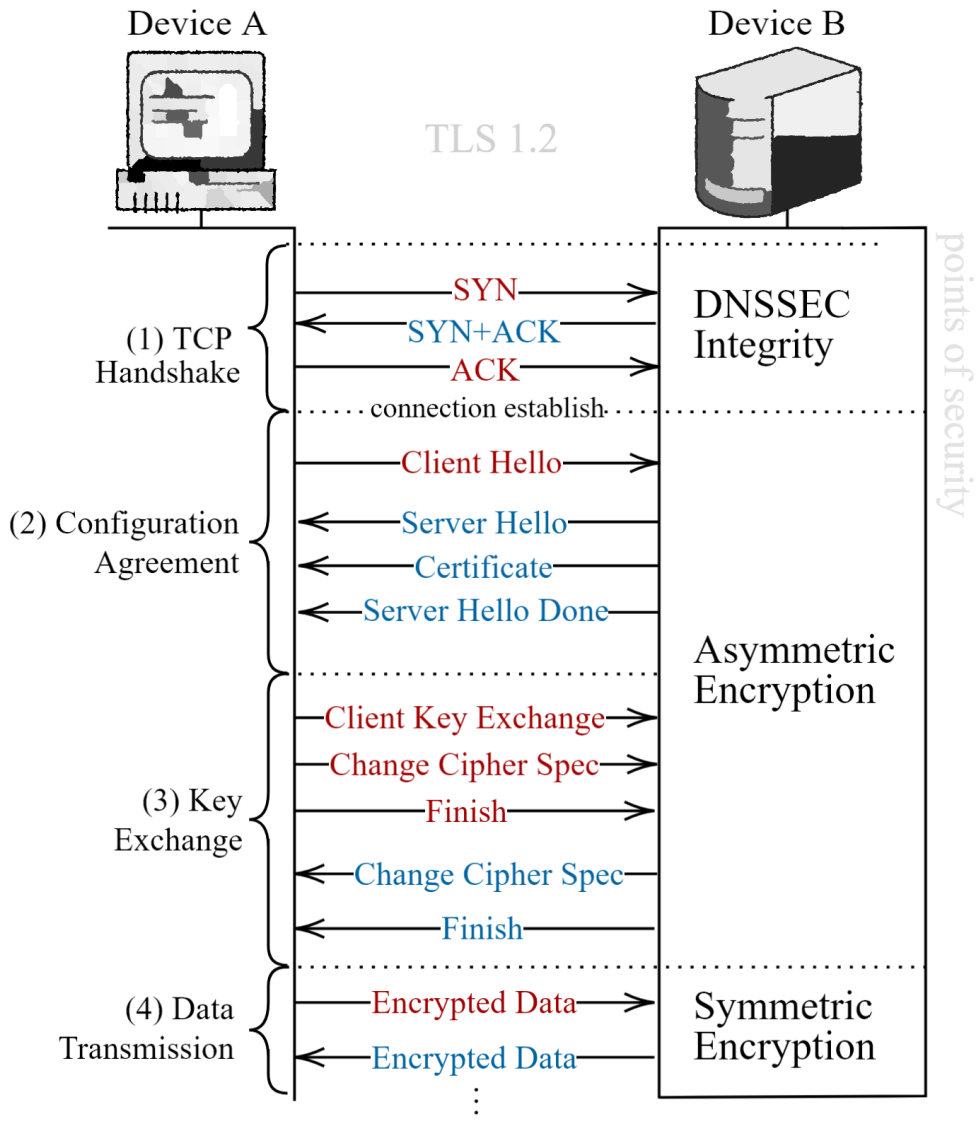
- Choosing TLS versions, and a cipher suite.

The server also sends its certificate, which the client verifies using the PKI. Then a **Server Hello Done** message is sent, and waits for the client to respond. The key exchange begins:

- RSA Method:The client generates a **pre-master secret** and encrypts it with the server's public key. The server decrypts the pre-master secret using its private key. The client may also use DHE instead.
- Both parties signal a **Change Cipher Spec** message, indicating the switch to encrypted communication.

The client sends a **Finished** message, which is encrypted with the negotiated algorithms and keys. The server decrypts the message and sends its own **Finished** message. If the server's message is successfully decrypted, the handshake is complete, and the secure connection is established. [12] [33]

Here is a visual representation of the TLS 1.2 Handshake Protocol:



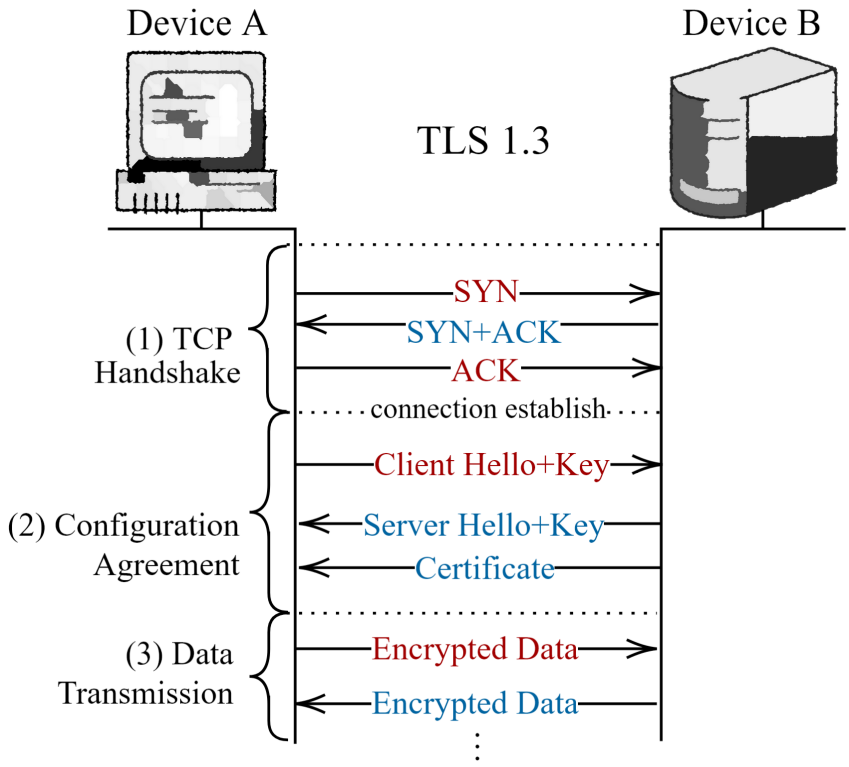
Definition 6.2: Improvements in TLS 1.3 and QUIC

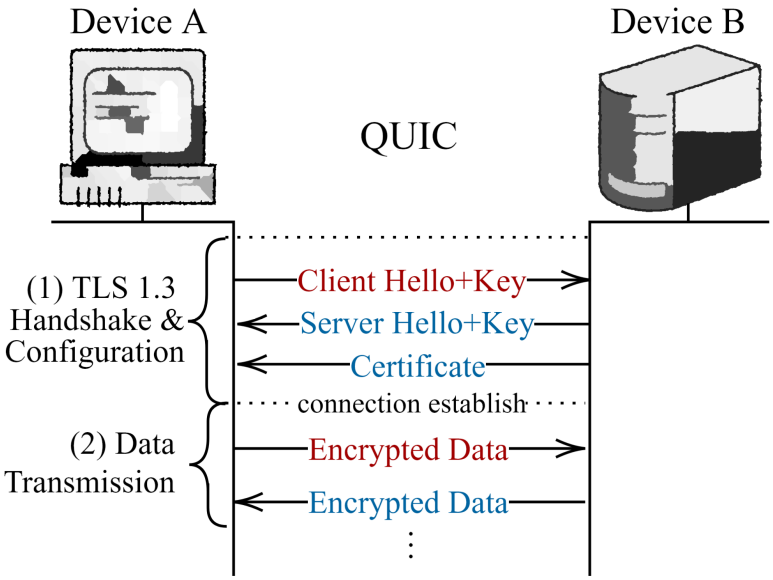
1. **Improvements in TLS 1.3:**
   - **Reduced Handshake Latency:**
     - The **Client Key Exchange** and the **Change Cipher Spec** steps are merged into the initial **ClientHello** and **ServerHello** messages.
     - The client sends the **key share** (for Diffie-Hellman Ephemeral) in the first message, enabling a **1-RTT (One Round-Trip Time)** handshake.
   - **Zero Round-Trip Time (0-RTT):** For resumed connections, clients can send encrypted application data immediately in the first message, reducing the handshake to **0-RTT**.
   - **Forward Secrecy:** TLS 1.3 enforces the use of ephemeral Diffie-Hellman (DHE or ECDHE) key exchange, providing **Perfect Forward Secrecy**.
   - **Simplified Cipher Suites:** Only modern cipher suites (e.g., AES-GCM, ChaCha20-Poly1305) are supported, enhancing security and reducing complexity.
2. **QUIC: A Modern Transport Protocol** QUIC (Quick UDP Internet Connections) replaces TCP and integrates **TLS 1.3** into its protocol stack.
   - QUIC combines the transport-layer connection setup and the **TLS 1.3 handshake** into a single step:
     - The **ClientHello** (including key share) is sent in the first QUIC packet.
     - The **ServerHello** (including key share and certificate) is sent in the server's first response.
   - **Zero Round-Trip Time (0-RTT):** QUIC supports **0-RTT resumption**.
   - **Elimination of TCP Handshake:** QUIC operates over UDP, removing the need for the TCP three-way handshake (SYN → SYN-ACK → ACK).

Important things to note thus far:

- **MACs**: Integrity of data. Requires either a shared secret between two parties or specified hashing algorithm (E.g., SHA-256). There is no reversing a hash function, they are one-way functions.
- **Certificates & Digital Signatures**: Authenticity of data. Requires a public key infrastructure (PKI) to verify the identity of the sender (E.g., RSA).
- **Encryption**: Confidentiality of data. Between parties requires a shared secret key (E.g., AES).

The following is a visual representation of the TLS 1.3 and QUIC Handshake Protocols:





— 4 —

Reverse Engineering & Attack Vectors

4.1 HTTP

Despite the security provided by TLS, the websites themselves can still be vulnerable to attacks. Especially if the wrong website is accessed, or the website is not properly secured.

Definition 1.1: HTML - HyperText Markup Language Elaboration

HTML (HyperText Markup Language) is the standard markup language used for creating and structuring content on the web. HTML provides a system of elements (tags) that describe the structure and content of a webpage.

- HTML documents are made up of **elements**, which consist of:
  1. Opening tags (e.g., `<p>` for a paragraph),
  2. Content (text, images, links, etc.), and
  3. Closing tags (e.g., `</p>`).
- The structure of an HTML document includes:
  - **Head Section:** Metadata, title, and links to external resources.
  - **Body Section:** Visible content such as headings, paragraphs, images, and links.

HTML is the backbone of webpages and works alongside CSS for styling and JavaScript for interactivity.

The task know is to understand the structure of an HTML document. Just like with any software, someone had to program it. Even the best of programers can make mistakes, these mistakes open up backdoors and vulnerabilities in the software. Such vulnerabilities can be exploited by attackers to gain access to the system.

Theorem 1.1: Schneier's Law

> *"Any person can invent a security system so clever that they can't imagine a way of breaking it."*

Human ingenuity has its limits. If one vulnerability exists, there are bound to be more.

Example 1.1: HTML Structure

The following is an example of a basic HTML document:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>My First HTML Page</title>
</head>
<body>
    <h1>Hello, World!</h1>
    <p>This is my first paragraph.</p>
    <a href="https://www.example.com">Visit Example</a>
</body>
</html>
```

Explanation:

- `<!DOCTYPE html>` declares the document type.
- The `<html>` element is the root of the document.
- The `<head>` section contains metadata and the page title.
- The `<body>` section includes visible content like headings, paragraphs, and links.

■

Theorem 1.2: Content Loading in HTTP and HTTPS

When a user requests a webpage over **HTTP** or **HTTPS**, the browser does not load the page as a single entity. Instead, the following process occurs:

1. The initial **skeleton of the page** (HTML) is fetched from the primary server.
2. The browser identifies and requests any **linked resources**, such as:
    - Images, videos, and scripts (e.g., `<img>`, `<script>`),
    - Stylesheets (e.g., `<link rel="stylesheet">`),
    - Fonts or third-party resources (e.g., analytics, ads, or widgets).
3. These subsequent resources may originate from **multiple servers or domains**, even if the user initially visited a single site.

Theorem 1.3: Multiple Requests Implications

Despite visiting one site, the user's browser may establish connections to **several servers**, often unknowingly. This introduces potential security and privacy concerns, such as:

- Content loaded over insecure connections (HTTP) may compromise the integrity of the page.
- Third-party services can track user behavior across different sites.

Example 1.2: Multiple Requests in Practice

Suppose a user visits `https://example.com`. The server provides the main HTML document. The browser then identifies and fetches linked resources, such as:

```
GET https://example.com/index.html       (main page)
GET https://cdn.example.com/style.css    (stylesheet)
GET http://images.example.com/logo.png  (image)
GET https://ads.thirdparty.com/ad.js     (third-party script)
```

Here, the user's browser interacts with `example.com`, `cdn.example.com`, `images.example.com`, and `thirdparty.com`.
As a result, the user's session involves multiple servers, illustrating the distributed nature of webpage loading. Here the **images** being accessed are over **HTTP**, which can compromise the integrity of the page. ■

Theorem 1.4: HTTPS Enforcement in Modern Browsers

Modern websites and browsers enforce the use of **HTTPS** connections to ensure secure communication between the client and server. This includes:

- **Client-Side Enforcement:** Most browsers block mixed content (HTTP resources on HTTPS pages) and warn users against accessing HTTP-only sites.
- **Server-Side Enforcement:** Websites implement HTTP Strict Transport Security (HSTS) to automatically upgrade all connections to HTTPS and prevent insecure fallback.

By requiring HTTPS, both client and server reduce risks of eavesdropping, tampering, and impersonation, ensuring confidentiality, integrity, and authenticity of the data exchanged.

Definition 1.2: Tracking Pixels

A **tracking pixel** (also known as a web beacon, pixel tag, or 1x1 pixel) is a small, transparent image or code snippet (typically `1x1` pixels in size) embedded in emails, webpages, or advertisements. It is used to track user behavior and collect data about interactions.

- **How It Works:**
  1. The tracking pixel is hosted on a server.
  2. When a user loads the webpage or opens an email, the browser requests the pixel from the server.
  3. This request contains metadata such as:
     - IP address of the user,
     - Device and browser information,
     - Time of access, and
     - Referrer URL.
- **Purpose:** Tracking pixels are primarily used for:
  - **Analytics:** Measuring email open rates, ad impressions, and user engagement.
  - **Behavioral Tracking:** Monitoring user activity on websites or across advertising campaigns.
  - **Personalization:** Collecting data to tailor ads or content to specific users.
- **Privacy Implications:** Tracking pixels can operate without explicit user consent, raising concerns about:
  - User privacy and data collection without awareness,
  - Cross-site tracking by third-party advertisers.

Example 1.3: Tracking Pixel in HTML

A tracking pixel is typically implemented as an invisible image hosted on a remote server:

```
<img src="https://tracking.example.com/pixel" width="1" height="1"
alt="" style="display:none;">
```

When this HTML is loaded, the browser requests the pixel from the server. The server records metadata such as the user's IP address, browser type, and referrer. ■

Everything is code, if one wishes to search something on the internet. Some server has to take the users response, interpret it, and send back the results. This interaction with the internal code opens the door for attack vectors.

Theorem 1.5: Injection Attacks

An **Injection Attack** occurs when an adversary provides malicious input to a system, exploiting the way user data is processed and interpreted by internal code. This attack vector leverages the fact that systems often interact with external inputs, such as databases, command interpreters, or web applications.

- **Mechanism:** The attacker injects crafted data that is processed as executable code or commands, causing unintended behavior such as:
  - Executing unauthorized commands,
  - Accessing sensitive data,
  - Manipulating system operations.
- **Mitigation:** Strong input validation, parameterized queries, and escaping special characters are key defenses against injection attacks.

Theorem 1.6: Cross-Site Scripting (XSS)

theo:xss **Cross-Site Scripting (XSS)** is a type of injection attack where an adversary injects malicious scripts into trusted websites, causing the browser to execute them on behalf of the victim.

- **Mechanism:** XSS exploits vulnerabilities in web applications that fail to properly sanitize user inputs. The attacker injects malicious code into webpages or inputs, which is then executed in the context of the victim's browser.
- **Impact:** XSS attacks can lead to user session hijacking, defacement of webpages, or unauthorized data access.
- **Types of XSS:**
  - **Stored XSS:** Malicious scripts are permanently stored on the server and executed whenever the resource is accessed.
  - **Reflected XSS:** The script is reflected off a web server, typically via a URL parameter or input field.
  - **DOM-Based XSS:** The script is executed directly in the browser, manipulating the Document Object Model (DOM) without server-side interaction.

Example 1.4: SQL Injection on a Login Page

Scenario: A login page allows users to enter a username and password. The server processes these inputs using a vulnerable SQL query. An attacker probes the system to identify weaknesses and then injects malicious SQL to bypass authentication.

Initial Probe: The attacker enters `' OR 1=1--` in the username field to test if the input is sanitized.

```
Input:
    Username: ' OR 1=1--
    Password: (leave blank)

Server Response:
    Error: Incorrect SQL syntax near '--'.
```

The server's error response indicates that the input is directly included in the SQL query without proper sanitization.

Malicious Injection: The attacker crafts a payload to bypass authentication:

```
Input:
    Username: admin'--
    Password: (leave blank)

Server-Side Query (Vulnerable):
    SELECT * FROM users WHERE username = 'admin'--' AND password = '';

Server Response:
    Authentication Successful.
```

Explanation:

- The injection (`'--`) comments out the password check.
- The query becomes valid, allowing the attacker to log in as `admin` without providing a password.

■

Theorem 1.7: Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) is an attack where an adversary tricks a user into performing unintended actions on a trusted website where they are authenticated.

- **Mechanism:** CSRF exploits the trust that a web application has in a user's browser session. An attacker crafts a malicious request that appears legitimate and relies on the user's authenticated state to execute it. Examples include:
  - Submitting a form to transfer funds or update account settings.
  - Sending unauthorized actions like deleting resources or creating accounts.
- **Impact:** CSRF attacks can result in:
  - Unauthorized transactions or changes to user data.
  - Exploitation of privileged user accounts to modify application settings.
  - Compromised security policies or leakage of sensitive data.
- **Example:** A victim is tricked into clicking a malicious link:

  ```
  <img src="https://example.com/transfer?amount=1000&to=attacker">
  ```

  If the user is logged into `example.com`, the request is executed using their session.
- **Mitigation:** Preventing CSRF involves:
  - Implementing anti-CSRF tokens to validate legitimate requests.
  - Verifying the `Referer` or `Origin` headers for requests.
  - Requiring user authentication for sensitive actions (e.g., re-entering passwords).

Theorem 1.8: Same-Origin Policy (SOP)

The **Same-Origin Policy (SOP)** is a browser security mechanism that restricts scripts and resources from one origin from interacting with another origin. The **origin** is defined by the protocol, domain, and port of a URL.

SOP validates the Origin or **Referer** (the URL of the previous webpage from which a link was followed) If the origins differ, the browser enforces restrictions.

Mechanisms like CORS (Cross-Origin Resource Sharing) and the PostMessage API allow servers and applications to configure and whitelist specific cross-origin interactions securely.

Definition 1.3: Cookies

A **cookie** is a small piece of data stored by a browser and sent with each HTTP request to the originating server. Cookies are commonly used for session management, personalization, and tracking. They are defined by key-value pairs and include options that control their behavior and scope.

- **Key Cookie Options:**
  - **Domain:** Specifies the domain where the cookie is valid.
  - **Path:** Limits the scope of the cookie to a specific URL path.
  - **Secure:** Ensures the cookie is sent only over HTTPS connections.
  - **HttpOnly:** Prevents access to the cookie via JavaScript, reducing the risk of XSS attacks.
  - **SameSite:** Controls cross-site usage:
    - **Strict:** Prevents the cookie from being sent with cross-site requests.
    - **Lax:** Allows the cookie to be sent with some cross-site requests (e.g., navigations).
    - **None:** Permits the cookie to be sent with all cross-site requests (must also use **Secure**).
  - **Expiration:** Determines the lifetime of the cookie:
    - **Session Cookie:** Deleted when the browser is closed.
    - **Persistent Cookie:** Remains until the specified expiration date.

Banks and other secure sites often use cookies to identify and remember users. This is how they can keep users logged in even after closing the browser.

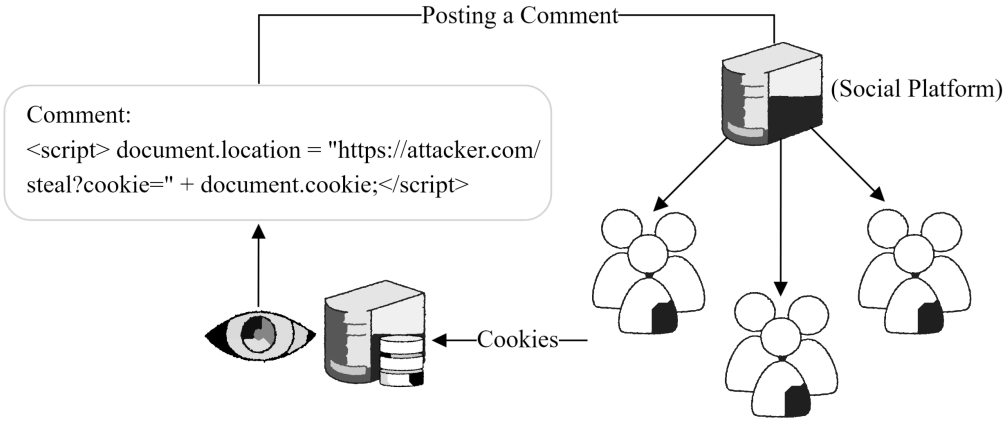


Figure 4.1: A **Stored XXS** attack via posting a comment containing malicious code.

The following attack uses both **Stored XSS** and **CSRF** to steal a user's session cookie:

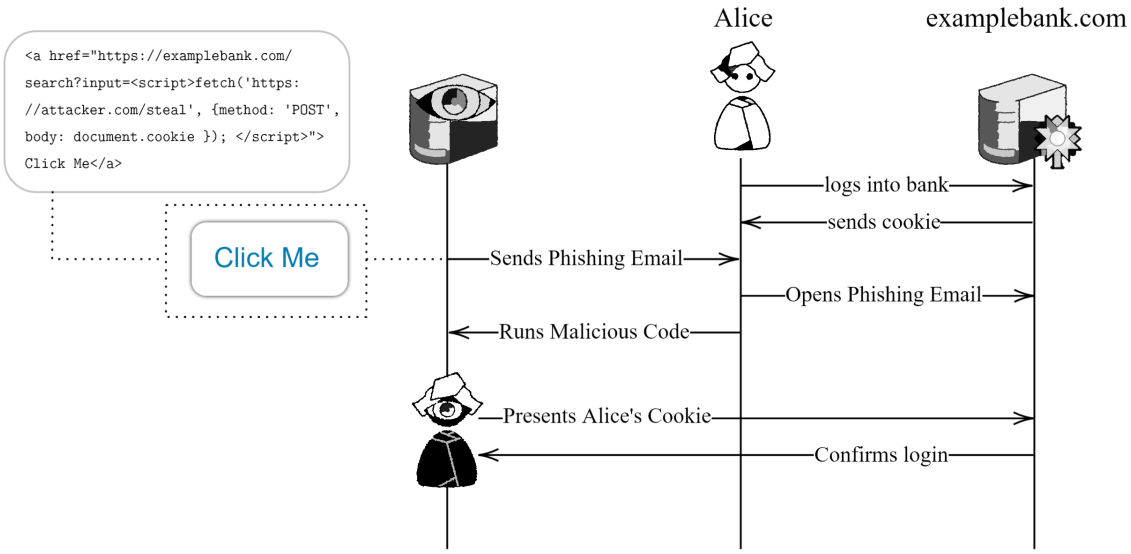


Figure 4.2: A **CSRF** attack via a malicious link.

This attack can be easily thwarted by making cookies **HttpOnly**. Remember, **HttpSecure** ensures that the cookie is only sent over HTTPS connections, nothing else.

Theorem 1.9: Password Attacks

Password attacks exploit weak, reused, or poorly managed passwords to gain unauthorized access. Common types include:

- **Credential Stuffing:** Using leaked username-password pairs from data breaches to access other accounts where users reused credentials.
- **Dictionary Attacks:** Automated guessing of passwords using common words, phrases, or precomputed password lists.
- **Fake Login Pages (Phishing):** Tricking users into entering passwords on fraudulent pages that mimic legitimate websites.
- **Social Engineering:** Manipulating users into revealing passwords through deception, trust exploitation, or psychological tactics.

These attacks emphasize the importance of unique, strong passwords, user awareness, and multi-factor authentication (MFA) to mitigate risks.

Bibliography

[1] DoD standard Internet Protocol. RFC 760, January 1980.

[2] Ieee standard for ethernet. *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)*, pages 1–5600, 2018.

[3] Aes (advanced encryption standard) structure, 2018–2023. Copyright © 2018–2023 BrainKart.com; Developed by Therithal info, Chennai. Accessed: 2024-06-15.

[4] Ieee standard for ethernet. *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pages 1–7025, 2022.

[5] Daniel Adetunji. Symmetric and asymmetric key encryption – explained in plain english, April 2023. Accessed: 2024-12-14.

[6] Internet Assigned Numbers Authority. Root servers. https://www.iana.org/domains/root/servers. Accessed November 30, 2024.

[7] Rahul Awati. Variable length subnet mask (vlsm). TechTarget, October 2021. Last updated October 2021, accessed November 2024.

[8] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). Request for Comments, May 2015. Category: Standards Track.

[9] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. Request for Comments, May 1996. Category: Informational.

[10] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax. https://www.rfc-editor.org/rfc/rfc3986#section-1.1.3, January 2005. Network Working Group, RFC 3986, Accessed: November 30, 2024.

[11] M. Bishop. HTTP/3. Request for Comments, June 2022. Category: Standards Track.

[12] ByteByteGo. Ssl, tls, https explained. YouTube Video, 2024. Accessed: 2024-12-17.

[13] CertBros. Ospf explained | step by step. YouTube, n.d. Accessed November 2024.

[14] East Charmer. Easy vlsm subnetting | step by step vlsm. https://www.youtube.com/watch?v=IgthYZ9N1vs&ab_channel=EastCharmer, n.d. Accessed November 2024.

[15] Chrystal R. China and Michael Goodwin. What is the osi model? IBM Think, June 2024. Published: 11 June 2024.

[16] Satish CJ. Aes iii - advanced encryption standard - introduction, key expansion in aes cyber security cse4003, 2024. Accessed: 2024-06-15.

[17] Cloudflare. Dns server types. https://www.cloudflare.com/learning/dns/dns-server-types/. Accessed November 30, 2024.

[18] Cloudflare. Dns over tls vs. dns over https | secure dns. https://www.cloudflare.com/learning/dns/dns-over-tls/, n.d. Accessed: 2024-12-14.

[19] Cloudflare. The dnssec root signing ceremony. https://www.cloudflare.com/learning/dns/dnssec/root-signing-ceremony/, n.d. Accessed: 2024-12-14.

[20] Cloudflare. What is tls (transport layer security)? https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/, n.d. Accessed: 2024-12-14.

[21] Cloudflare. Why is http not secure? | http vs. https. https://www.cloudflare.com/learning/ssl/why-is-http-not-secure/, n.d. Accessed: 2024-12-14.

[22] Cloudflare Learning Center. What do client side and server side mean?, n.d. Accessed: 2024-11-27.

[23] Cloudflare Learning Center. What is a subnet? | how subnetting works. Cloudflare, n.d. Accessed November 2024.

[24] Cloudflare Learning Center. What is an autonomous system? | what are asns? Cloudflare, n.d. Accessed November 2024.

[25] Cloudflare Learning Center. What is internet protocol (ip)?, n.d. Accessed: 2024-11-27.

[26] Cloudflare Learning Center. What is routing? | ip routing. Cloudflare, n.d. Accessed November 2024.

[27] Codecademy. What is hashing, and how does it work? https://www.codecademy.com/resources/blog/what-is-hashing/, April 2023. Accessed: 2024-12-14.

[28] CodeSigningStore Team. Hash algorithm comparison: Md5, sha-1, sha-2 & sha-3. https://codesigningstore.com/hash-algorithm-comparison, 2024. Accessed: 2024-12-17.

[29] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet x.509 public key infrastructure certificate and certificate revocation list (crl) profile. Request for Comments (RFC) 5280, May 2008. Obsoletes RFC 3280, RFC 4325, RFC 4630.

[30] M. Cotton, L. Eggert, J. Touch, M. Westerlund, and S. Cheshire. Internet assigned numbers authority (iana) procedures for the management of the service name and transport protocol port number registry. Request for Comments 6335, Internet Engineering Task Force, August 2011. BCP 165.

[31] Cybersecurity and Infrastructure Security Agency (CISA). Understanding digital signatures. https://www.cisa.gov/news-events/news/understanding-digital-signatures, February 2021. Accessed: 2024-12-14.

[32] S. Deering and R. Hinden. Internet protocol, version 6 (ipv6) specification. Technical Report RFC 8200, Internet Engineering Task Force (IETF), July 2017. Internet Standard 86, obsoletes RFC 2460.

[33] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. Request for Comments 5246, RFC Editor, August 2008. Standards Track.

[34] Morris Dworkin. Recommendation for block cipher modes of operation: Methods and techniques. Special Publication 800-38A, National Institute of Standards and Technology (NIST), 2001.

[35] Editorial Team. Multiplexing. *Network Encyclopedia*, October 2023. Last edited October 5, 2023.

[36] Aleksander Essex. Encrypting with block ciphers, 2024. YouTube video.

[37] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments, January 1997. Category: Standards Track.

[38] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Request for Comments, June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, and 7235. Errata exist.

[39] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. https://www.rfc-editor.org/rfc/rfc2616, June 1999. Network Working Group, RFC 2616, Accessed: November 30, 2024.

[40] Roy T. Fielding, Mark Nottingham, and Julian Reschke. HTTP Semantics. RFC 9110, June 2022.

[41] V. Fuller, T. Li, J. Yu, and K. Varadhan. Classless inter-domain routing (cidr): An address assignment and aggregation strategy. Technical Report RFC 1519, Internet Engineering Task Force (IETF), September 1993. Accessed November 2024.

[42] Sharon Goldberg. Cs357: Introduction to information security. Lecture notes, Boston University, Fall Semester, 2024. Boston University, CS Department.

[43] Charles L. Hedrick. A standard for the transmission of ip datagrams over ethernet networks. Request for Comments 894, Internet Engineering Task Force, April 1984.

[44] Simon Heimlicher, Pavan Nuggehalli, and Martin May. End-to-end vs. hop-by-hop transport. In *Proceedings of the ACM SIGMETRICS*. Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland; Centre for Electronics Design and Technology, Indian Institute of Science, Bangalore, India, 2007. Accessed November 2024.

[45] IBM Documentation. Ipv4 and ipv6 address formats. IBM Documentation, January 2022. Last updated 2022-01-18, accessed November 2024.

[46] IEEE Computer Society. *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture*. IEEE, 2001. IEEE Std 802-2001 (Revision of IEEE Std 802-1990).

[47] J. Iyengar and M. Thomson. Quic: A udp-based multiplexed and secure transport. Request for Comments 9000, Internet Engineering Task Force, May 2021.

[48] Javatpoint. Rip protocol. Javatpoint, n.d. Accessed November 2024.

[49] Vijay Kanade. What is the osi model? definition, layers, and importance. Spiceworks, November 2022. Accessed November 2024.

[50] Kinsta. Tls vs ssl: What's the difference? which one should you use? https://kinsta.com/knowledgebase/tls-vs-ssl/, 2019. Published December 19, 2019. Updated August 14, 2023. Accessed: 2024-12-14.

[51] Jim Kurose. Computer networks and the internet: Routing algorithms and link state routing. YouTube, n.d. Video presentation for Computer Networking: A Top-Down Approach (8th edition), J.F. Kurose and K.W. Ross, Pearson, 2020. Taught at University of Massachusetts Amherst.

[52] Samson Leonard. Lecture 1: The osi model. SlidePlayer, 2014. Modified over 9 years ago. References: TCP/IP Protocol Suite, 4th Edition (Chapter 2).

[53] Kwun-Hung Li and Kin-Yeung Wong. Empirical analysis of ipv4 and ipv6 networks through dual-stack sites. *Information*, 12(6), 2021.

[54] B. Manning and P. Perkins. Variable length subnet table for ipv4. Technical Report RFC 1878, Internet Engineering Task Force (IETF), December 1995. Informational.

[55] P. Mockapetris. Domain names - concepts and facilities. Request for Comments 1034, 1987. Obsoletes RFCs 882, 883, 973.

[56] Jeffrey C. Mogul and Steven E. Deering. Path mtu discovery. Request for Comments 1191, Internet Engineering Task Force, November 1990.

[57] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor discovery for ip version 6 (ipv6). Technical Report RFC 4861, Internet Engineering Task Force (IETF), September 2007. Standards Track, Obsoletes RFC 2461.

[58] National Institute of Standards and Technology (NIST), Morris J. Dworkin, Elaine Barker, James R. Nechvatal, James Foti, Lawrence E. Bassham, E. Roback, and James F. Dray Jr. Advanced encryption standard (aes). Federal Information Processing Standards Publication 197, National Institute of Standards and Technology, November 2001. Accessed: 2024-06-15. Local download available at https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=901427.

[59] Number Resource Organization (NRO). Regional internet registries (rirs) overview. Website, 2024. Accessed November 28, 2024.

[60] M. Nystrom and B. Kaliski. Pkcs #10: Certification request syntax specification version 1.7. Request for Comments (RFC) 2986, November 2000. Obsoletes RFC 2314.

[61] Okta. What is public key infrastructure (pki) and how does it work? https://www.okta.com/identity-101/public-key-infrastructure/, August 2024. Accessed: 2024-12-14.

[62] Steve Petryschuk. Types of networks: Lan, wan, man, and more. https://www.auvik.com/franklyit/blog/types-of-networks/, n.d. Accessed November 2024.

[63] J. Postel. User datagram protocol. Request for Comments 768, Internet Engineering Task Force, August 1980.

[64] J. Postel. Internet control message protocol. Request for Comments 792, Internet Engineering Task Force, September 1981.

[65] J. Postel. Internet protocol - darpa internet program protocol specification. Technical Report RFC 791, Internet Engineering Task Force (IETF), September 1981. Internet Standard 5, obsoletes RFC 760, updated by RFCs 1349, 2474, and 6864.

[66] J. Postel. Internet protocol: Darpa internet program protocol specification. Technical Report RFC 791, Defense Advanced Research Projects Agency (DARPA), Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA, September 1981. Obsoleted by RFC 1349, RFC 2474, RFC 6864.

[67] J. Postel. Transmission control protocol. Request for Comments 793, Internet Engineering Task Force, September 1981.

[68] J. Postel. Broadcasting internet datagrams in the presence of subnets. Technical Report RFC 922, Internet Engineering Task Force (IETF), October 1984. Standards Track.

[69] Ammar Rayes and Samer Salam. *The Internet in IoT*, pages 35–62. Springer International Publishing, Cham, 2022.

[70] Y. Rekhter, B. Moskowitz, D. Karrenberg, G.J. de Groot, and E. Lear. Address allocation for private internets. Request for Comments 1918, Internet Engineering Task Force, February 1996. BCP 5.

[71] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[72] Mike Rosulek. The joy of cryptography. https://joyofcryptography.com.

[73] RtBrick. Rbfs carrier-grade network address translation overview. Website, 2024. Accessed November 28, 2024.

[74] Seth. What is the difference between mac and hmac? Crypto Stack Exchange, 2013. Answered on Mar 1, 2013, edited Feb 9, 2016. Available at: https://crypto.stackexchange.com/questions/6523/what-is-the-difference-between-mac-and-hmac [Accessed: YYYY-MM-DD].

[75] P. Srisuresh and M. Holdrege. Ip network address translator (nat) terminology and considerations. Request for Comments 2663, Internet Engineering Task Force, August 1999.

[76] C. Sunshine and J. Postel. Broadcasting internet datagrams. Technical Report RFC 919, Internet Engineering Task Force (IETF), October 1984. Standards Track.

[77] Martin Thomson and Francesca Palombini. The rfc series and rfc editor. RFC 9280, June 2022.

[78] Forrest Timour. An abridged history of cryptography, 2019. Available online.

[79] University of Oklahoma. History of the world wide web, n.d. Accessed: 2024-11-27.

[80] Vidder, Inc. Galois/counter mode (gcm) and gmac, 2016. Accessed: 2024-06-15.

[81] Concise Works. Sect modules. https://github.com/Concise-Works/sect-modules, n.d. Accessed November 2024.

[82] World Wide Web Consortium (W3C). Html working group draft: Href and url standards, n.d. Accessed: 2024-11-27.

[83] World Wide Web Consortium (W3C). Hypertext transfer protocol (http) as implemented in w3, n.d. Accessed: 2024-11-27.

[84] Kinza Yasar, Mary E. Shacklett, and Amy Novotny. What is tcp/ip? TechTarget, September 2024. Last updated September 2024.

[85] Tal Yitzhak. Understanding digital certificates: Self-signed vs. ca-signed certificates. https://medium.com/@talyitzhak/understanding-digital-certificates-and-self-signed-certificates-b1cdca759bbc, July 2024. Accessed: 2024-12-14.