## Overview of SHA-256 Hashing Process for Input "red block blue"

- The input string
- The explanation covers the

## Step 1: ASCII to Binary Conversion

- Each character in the input is converted to its
- For the 12 characters in "red block blue", this results in a
- These bits are concatenated into one long binary string.
- The bits are then

## Step 2: Pre-processing and Padding

- SHA-256 requires input length to be a
- Current input length:
- Round up to nearest multiple of 512 →
- Padding procedure:
- Final padded input length:

| Step | Bit Length | Description |
|---|---|---|
| Original input | 96 | ASCII binary of "red block blue" |
| Append '1' bit | +1 | Single '1' bit appended |
| Append zeros | +440 | Padding zeros to reach 448 bits |
| Append length (64-bit) | +64 | Binary representation of 96 |
| Total | 512 | Padded input length |

### 🔹 Step 3: Splitting into Blocks and Words

- The padded input is split into
- Since input is exactly 512 bits, there is
- Each 512-bit block is split into
- These words are labeled

### ⚙️ Step 4: Initialization of Hash Values (h0 to h7)

- Initialize
- These initial values are derived from the
- Purpose: To ensure
- Example:
- These constants are fixed and standardized.

### 🔹 Step 5: Constants k0 to k63

- 64 constants
- Derived from the
- These constants are fixed and standardized.
- Examples:

### 🔹 Step 6: Message Schedule Preparation (w0 to w63)

- Prepare a
- First 16 words:
- Remaining words (w16 to w63) are computed iteratively using the formula:

[ w_t = \sigma_1(w_{t-2}) + w_{t-7} + \sigma_0(w_{t-15}) + w_{t-16} \quad (mod 2^{32}) ]

- Where:

### 🔹 Explanation of (\sigma_0) and (\sigma_1) Functions

- For a 32-bit word (x):

| Function | Operations |
|---|---|
| (\sigma_0(x)) | ( (x ROTR 7) \oplus (x ROTR 18) \oplus (x SHR 3) ) |
| (\sigma_1(x)) | ( (x ROTR 17) \oplus (x ROTR 19) \oplus (x SHR 10) ) |

- ROTR n

- SHR n
- (\oplus) denotes

### 🔹 Step 7: Compression Loop (t = 0 to 63)

- Initialize working variables (a, b, c, d, e, f, g, h) with initial hash values (h_0) to (h_7).
- For each iteration (t) from 0 to 63:

| Step | Calculation |
|---|---|
| (T_1) | (h + \Sigma_1(e) + Ch(e, f, g) + k_t + w_t) |
| (T_2) | (\Sigma_0(a) + Maj(a, b, c)) |
| Update | (h = g), (g = f), (f = e), (e = d + T_1), (d = c), (c = b), (b = a), (a = T_1 + T_2) |

- Where:

| Function | Description |
|---|---|
| (\Sigma_0(x)) | ( (x ROTR 2) \oplus (x ROTR 13) \oplus (x ROTR 22) ) |
| (\Sigma_1(x)) | ( (x ROTR 6) \oplus (x ROTR 11) \oplus (x ROTR 25) ) |
| Ch(x,y,z) | Choose function: bitwise selection based on bits of (x) to choose bits from (y) or (z) |
| Maj(x,y,z) | Majority function: bitwise majority of bits among (x, y, z) |

### 🔹 Details of Choose (Ch) and Majority (Maj) Functions

- Choose (Ch)
- Majority (Maj)

### 🔹 Example Calculation for (w_{16})

- (w_{16} = \sigma_1(w_{14}) + w_9 + \sigma_0(w_1) + w_0 \mod 2^{32})
- Given many of these (w_i) are zero, the calculation simplifies.
- Bitwise operations and modulo addition ensure (w_{16}) remains 32 bits.

### 🔹 Step 8: Final Hash Value Update

- After completing all 64 rounds, update the hash values:

[ h_i = h_i + working variable_i \quad for i=0 ⋯ 7 ]

- This produces the final

### Step 9: Output

- Concatenate the eight 32-bit words to form the
- Output is typically represented in
- For input "red block blue", the final SHA-256 hash is a

### Summary Table of Key Components

| Component | Description |
|---|---|
| Input | "red block blue" (12 characters, 96 bits ASCII) |
| Padding | Append '1' bit, zeros, and 64-bit length encoding to reach 512 bits |
| Blocks | 1 block of 512 bits |
| Words | 16 initial 32-bit words (m0 to m15), extended to 64 words (w0 to w63) |
| Initial Hash Values (h0-h7) | Derived from fractional parts of square roots of first 8 primes |
| Constants (k0-k63) | Derived from fractional parts of cube roots of first 64 primes |
| Functions | (\sigma_0, \sigma_1, \Sigma_0, \Sigma_1, Ch, Maj) |
| Compression Rounds | 64 iterations updating working variables and hash values |
| Final Output | 256-bit hash (eight 32-bit words concatenated), shown in hexadecimal |

### ⚠ Important Notes

- All additions are
- Bitwise operations (rotations, shifts, XOR) are fundamental to the transformations.
- The use of prime number roots ensures
- The process is deterministic: same input → same output.
- The example input is simple, so only one 512-bit block is processed.

This detailed breakdown covers the entire SHA-256 hashing process as applied to the input "red block blue," including all intermediate steps, constants, and functions involved.

Here is a cleanly formatted version of your Markdown text, using best practices for section breaks, headers, and tables:

### Overview of SHA-256 Hashing Process for Input "red block blue"

- The input string
- The explanation covers the

### Step 1: ASCII to Binary Conversion

- Each character in the input is converted to its
- For the 12 characters in "red block blue", this results in a
- These bits are concatenated into one long binary string.
- The bits are then

### Step 2: Pre-processing and Padding

- SHA-256 requires input length to be a
- Current input length:
- Round up to nearest multiple of 512 →
- Padding procedure:
- Final padded input length:

| Step | Bit Length | Description |
|---|---|---|
| Original input | 96 | ASCII binary of "red block blue" |
| Append '1' bit | +1 | Single '1' bit appended |
| Append zeros | +440 | Padding zeros to reach 448 bits |
| Append length (64-bit) | +64 | Binary representation of 96 |
| Total | 512 | Padded input length |

### Step 3: Splitting into Blocks and Words

- The padded input is split into
- Since input is exactly 512 bits, there is
- Each 512-bit block is split into
- These words are labeled

### Step 4: Initialization of Hash Values (h0 to h7)

- Initialize
- These initial values are derived from the
- Purpose: To ensure
- Example:
- These constants are fixed and standardized.

### Step 5: Constants k0 to k63

- 64 constants
- Derived from the
- These constants are fixed and standardized.
- Examples:

### Step 6: Message Schedule Preparation (w0 to w63)

- Prepare a
- First 16 words:
- Remaining words (w16 to w63) are computed iteratively using:

```
w_t = σ₁(w_{t-2}) + w_{t-7} + σ₀(w_{t-15}) + w_{t-16}   (mod 2^32)
```

- Where:

### Explanation of ($\sigma_0$) and ($\sigma_1$) Functions

- For a 32-bit word (x):

| Function | Operations |
|---|---|
| $\sigma_0(x)$ | (x ROTR 7) ⊕ (x ROTR 18) ⊕ (x SHR 3) |
| $\sigma_1(x)$ | (x ROTR 17) ⊕ (x ROTR 19) ⊕ (x SHR 10) |

- ROTR n
- SHR n
- (⊕) denotes

### Step 7: Compression Loop (t = 0 to 63)

- Initialize working variables (a, b, c, d, e, f, g, h) with initial hash values (h_0) to (h_7).
- For each iteration (t) from 0 to 63:

| Step | Calculation |
|---|---|
| ($T_1$) | h + $\Sigma_1$(e) + Ch(e, f, g) + k_t + w_t |
| ($T_2$) | $\Sigma_0$(a) + Maj(a, b, c) |
| Update | h = g, g = f, f = e, e = d + $T_1$, d = c, c = b, b = a, a = $T_1$ + $T_2$ |

- Where:

| Function | Description |
|---|---|
| $\Sigma_0(x)$ | (x ROTR 2) ⊕ (x ROTR 13) ⊕ (x ROTR 22) |
| $\Sigma_1(x)$ | (x ROTR 6) ⊕ (x ROTR 11) ⊕ (x ROTR 25) |
| Ch(x, y, z) | Choose function: bitwise selection based on bits of x to choose from y, z |
| Maj(x, y, z) | Majority function: bitwise majority of bits among x, y, z |

### Details of Choose (Ch) and Majority (Maj) Functions

- Choose (Ch)
- Majority (Maj)

### Example Calculation for (w_{16})

- w_{16} = $\sigma_1$(w_{14}) + w_9 + $\sigma_0$(w_1) + w_0 (mod 2^32)
- Given many of these w_i are zero, the calculation simplifies.
- Bitwise operations and modulo addition ensure w_{16} remains 32 bits.

### Step 8: Final Hash Value Update

- After completing all 64 rounds, update the hash values:

```
h_i = h_i + working_variable_i  for i = 0..7
```

- This produces the final

### Step 9: Output

- Concatenate the eight 32-bit words to form the
- Output is typically represented in
- For input "red block blue", the final SHA-256 hash is a

### Summary Table of Key Components

| Component | Description |
|---|---|
| Input | "red block blue" (12 characters, 96 bits ASCII) |
| Padding | Append '1' bit, zeros, and 64-bit length encoding to 512 bits |
| Blocks | 1 block of 512 bits |
| Words | 16 initial 32-bit words (m0 to m15), extended to 64 (w0 to w63) |

| Component | Description |
| --- | --- |
| Initial Hash Values (h0-h7) | Derived from square roots of first 8 primes |
| Constants (k0-k63) | Derived from cube roots of first 64 primes |
| Functions | $\sigma_0$, $\sigma_1$, $\Sigma_0$, $\Sigma_1$, Ch, Maj |
| Compression Rounds | 64 iterations updating hash values |
| Final Output | 256-bit hash (eight 32-bit words), shown in hexadecimal |

### Important Notes

- All additions are
- Bitwise operations (rotations, shifts, XOR) are fundamental to the transformations.
- The use of prime number roots ensures
- The process is deterministic: same input → same output.
- The example input is simple, so only one 512-bit block is processed.

This detailed breakdown covers the entire SHA-256 hashing process as applied to the input "red block blue," including all intermediate steps, constants, and functions involved.

Here’s your Markdown file reformatted into a clean, consistent, and professional style with proper headers, spacing, formulas, and tables for readability:

# SHA-256 Hashing Process for Input "red block blue"

### Overview

- Input string: `"red block blue"`
- Explanation covers all major stages of SHA-256, from input to final hash.

### Step 1: ASCII to Binary Conversion

- Each character in the input is converted to its ASCII binary representation.
- For the 12 characters in `"red block blue"`, this produces a 96-bit binary string.
- These binary values are concatenated into one long bit sequence.
- The bits are then used for padding.

## 🔹 Step 2: Pre-processing and Padding

- SHA-256 requires input length to be **a multiple of 512 bits**.
- Current input length: 96 bits.
- Round up to 512 bits using SHA-256 padding rules.

**Padding procedure:**

1. Append `1` bit.
2. Pad with zeros until length = 448 bits.
3. Append 64-bit binary encoding of original length (96).

| Step | Bit Length | Description |
|---|---|---|
| Original input | 96 | Binary of `"red block blue"` |
| Append 1 bit | +1 | Adds a single '1' |
| Append zeros | +440 | Fill with zeros to reach 448 bits |
| Append length (64-bit) | +64 | `000...1100000` (96 in 64-bit binary) |
| **Total** | 512 | Padded input |

## 🔹 Step 3: Splitting into Blocks and Words

- Padded input is **512 bits** → only 1 block.
- Each 512-bit block is split into 16 words of 32 bits each.
- These are labeled `m0` to `m15`.

## ⚙️ Step 4: Initialization of Hash Values

- Initialize eight working variables: `h0` to `h7`.
- These are derived from the **square roots of the first 8 prime numbers**.
- They ensure randomness and standardization.

## 🔹 Step 5: Constants (k0 to k63)

- 64 constants, derived from **cube roots of the first 64 primes**.
- They are fixed, predefined values.

## 🔹 Step 6: Message Schedule Preparation

- Extend the 16 original words into 64 words (`w0`–`w63`).
- For t ≥ 16:

$$w_t = \sigma_1(w_{t-2}) + w_{t-7} + \sigma_0(w_{t-15}) + w_{t-16} \pmod{2^{32}}$$

## 🔹 σ Functions

For a 32-bit word $x$:

| Function | Formula |
|---|---|
| $\sigma_0(x)$ | $(x \text{ ROTR } 7) \oplus (x \text{ ROTR } 18) \oplus (x \text{ SHR } 3)$ |
| $\sigma_1(x)$ | $(x \text{ ROTR } 17) \oplus (x \text{ ROTR } 19) \oplus (x \text{ SHR } 10)$ |

## 🔹 Step 7: Compression Loop (64 rounds)

- Initialize working variables $a, b, c, d, e, f, g, h$ with hash values.
- For each round $t$ (0–63):

| Step | Formula |
|---|---|
| $T_1$ | $h + \Sigma_1(e) + Ch(e,f,g) + k_t + w_t$ |
| $T_2$ | $\Sigma_0(a) + Maj(a,b,c)$ |
| Update | $h = g,\ g = f,\ f = e,\ e = d + T_1,\ d = c,\ c = b,\ b = a,\ a = T_1 + T_2$ |

**Functions:**

| Function | Description |
|---|---|
| $\Sigma_0(x)$ | $(x \text{ ROTR } 2) \oplus (x \text{ ROTR } 13) \oplus (x \text{ ROTR } 22)$ |
| $\Sigma_1(x)$ | $(x \text{ ROTR } 6) \oplus (x \text{ ROTR } 11) \oplus (x \text{ ROTR } 25)$ |
| $Ch(x,y,z)$ | If bit of x=1 → choose y, else z |
| $Maj(x,y,z)$ | Majority vote of bits among x,y,z |

## 🔹 Example Expansion for w16

$$w_{16} = \sigma_1(w_{14}) + w_9 + \sigma_0(w_1) + w_0 \pmod{2^{32}}$$

- Many initial values are zero, simplifying calculations.

### Step 8: Final Hash Value Update

After 64 rounds:

$$h_i = h_i + workingVar_i \quad (i = 0..7)$$

Eight updated 32-bit words are combined.

### Step 9: Output

- Concatenate all eight 32-bit words → **256-bit hash**.
- Represent in **hexadecimal** format.
- For `"red block blue"`, the SHA-256 produces a fixed unique hash.

### Summary of Key Components

| Component | Description |
|---|---|
| Input | `"red block blue"` (96 bits ASCII) |
| Padding | `+1 bit + zeros + 64-bit length` → 512 bits |
| Blocks | 1 (512-bit block) |
| Words | 16 → extended to 64 |
| Initial Hash Values | Fractional square roots of first 8 primes |
| Constants | Fractional cube roots of first 64 primes |
| Functions | $\sigma_0, \sigma_1, \Sigma_0, \Sigma_1, Ch, Maj$ |
| Compression | 64 rounds |
| Output | 256-bit hexadecimal hash |

### ⚠ Notes

- All additions are mod $2^{32}$.
- Bitwise ops (shift, rotate, XOR) are crucial.
- Prime-root constants ensure cryptographic randomness.
- Deterministic: same input → same output.
- Example here: only one block processed since input is small.

Would you like me to also include the *actual computed SHA-256 hash* of `"red block blue"` in the final section so the document is complete with output?