### **Project Scope: Cryptography Algorithms Visualization Tool**

#### **Objective**

To create an interactive, step-by-step visualization platform for cryptographic algorithms to help users, including students, researchers, and professionals, understand the internal operations of these algorithms. The tool will provide theoretical explanations, process breakdowns, and hands-on experimentation opportunities.

### **Technology Stack**

* **Frontend Framework:** ReactJS (primary framework for building UI).
* **Visualization Libraries:**
  + D3.js for custom visualizations and graph-based animations.
  + Konva.js for canvas-based dynamic and interactive elements.
  + Chart.js for graphical representations of processes (optional).
* **Styling:**
  + CSS with TailwindCSS or Material-UI for responsive and modern UI design.
* **Mathematical Computation Libraries:**
  + Math.js for complex arithmetic and modular computations.
  + BigInt.js for handling large integers used in cryptographic operations.
* **Data Handling:** JSON for storing algorithm configurations and steps.
* **Deployment:** Deployed via static hosting platforms like Netlify or Vercel.

### **Expected Input and Output**

#### **Input**

* **User-provided inputs** such as:
  + Plaintext: Text or data to be encrypted or hashed.
  + Keys: Encryption/decryption keys.
  + Algorithm Parameters: Key size, block size, rounds, etc.
  + Custom Values: For prime numbers, initial seeds, or other algorithm-specific parameters.
* **Algorithm Selection:** Dropdown or tile-based interface to select the desired cryptographic algorithm.

#### **Output**

* **Step-by-Step Visualizations:**
  + Intermediate results at each step (e.g., round keys, XOR outputs, shifted rows).
  + Final output such as encrypted ciphertext or hash digest.
* **Graphical Representations:**
  + Dynamic graphs, flow diagrams, and tables displaying data transformations.
* **Annotations:**
  + Real-time explanations or tooltips for mathematical operations in progress.
* **Exportable Files:**
  + PDF or PNG exports of visualization steps.
  + JSON files of algorithm configurations for reuse.

### **Detailed Step-by-Step Visualizations for Algorithms**

#### **AES (Advanced Encryption Standard)**

1. **Key Expansion:**
   * Visualize all 16 steps of key expansion.
   * Show bit-level transformations as the key moves through the S-box.
   * Display the generated round keys in a table format for each round.
2. **AddRoundKey:**
   * Highlight the XOR operation of the plaintext with the round key.
   * Provide a visual explanation of how each bit is XORed.
3. **SubBytes:**
   * Display a 4x4 grid representation of the state matrix.
   * Show how each byte is substituted using the S-box, with animations showing the lookup process.
4. **ShiftRows:**
   * Animate the shifting of rows in the state matrix, row by row.
   * Show before and after states for clarity.
5. **MixColumns:**
   * Represent column operations using matrix multiplication.
   * Provide real-time calculations for each step with visual aids for intermediate results.
6. **Repeat for All Rounds:**
   * Clearly distinguish between the first 9/11/13 rounds and the final round (which skips MixColumns).
7. **Final Ciphertext:**
   * Display the final state matrix as the encrypted ciphertext.

#### **DES (Data Encryption Standard)**

1. **Initial Permutation (IP):**
   * Show the bitwise reordering of plaintext using the IP table.
   * Include before and after visualization of bit positions.
2. **Feistel Rounds:**
   * Visualize all 16 rounds of the Feistel structure, step by step:
     + **Expansion (E):** Animate the expansion from 32 bits to 48 bits.
     + **Key Mixing:** XOR operation with the round key.
     + **S-Box Substitution:** Highlight the lookup process in S-boxes.
     + **Permutation (P):** Display the reordering of bits post S-box substitution.
3. **Final Permutation (FP):**
   * Show the reordering of bits to produce the ciphertext.

#### **RSA (Rivest–Shamir–Adleman)**

1. **Key Generation:**
   * Display the selection of prime numbers pp and qq.
   * Visualize the calculation of n=p×qn = p \times q and ϕ(n)\phi(n).
   * Show step-by-step selection of ee and computation of dd.
2. **Encryption:**
   * Animate the modular exponentiation process for C=Memod  nC = M^e \mod n.
   * Highlight intermediate calculations for large powers and modulo reductions.
3. **Decryption:**
   * Display the decryption process M=Cdmod  nM = C^d \mod n with step-by-step operations.

#### **SHA-256 (Secure Hash Algorithm)**

1. **Pre-Processing:**
   * Visualize the padding of the input message.
   * Show the splitting of the message into 512-bit blocks.
2. **Message Schedule:**
   * Animate the creation of the message schedule for each block.
   * Display intermediate values for all 64 entries.
3. **Hash Computation:**
   * Represent the initialization of hash values.
   * Show each round’s operations, including bitwise additions, rotations, and XORs.
   * Provide graphical representations of the compression function.

### **Features and Functionalities**

#### **1. Educational Content**

* Overview of each algorithm, its purpose, and use cases.
* Step-by-step breakdowns of mathematical operations.

#### **2. Interactive Visualizations**

* Parameter Adjustments:
  + Real-time input fields for users to modify parameters like key size or prime numbers.
* Intermediate Results:
  + Display matrices, tables, and graphs showing intermediate values.

#### **3. Experimentation and Testing**

* Custom Inputs:
  + Allow users to input their own plaintext or keys.
* Side-by-Side Comparisons:
  + Compare outputs of different algorithms or configurations.

#### **4. Export and Sharing**

* Visualization Exports:
  + Save visualizations as images (PNG, GIF) or PDFs.

#### **5. Advanced Features**

* Themes:
  + Light and dark modes.
* Search and Navigation:
  + Include search functionality for quick access to algorithms.

### **Security Considerations**

* All computations should be performed client-side for data privacy.
* Avoid storing user inputs on servers.

### **Deliverables**

1. Fully Functional Tool:
   * Interactive and responsive web application.
2. Documentation:
   * User guide explaining how to use the tool and interpret the visualizations.
3. Test Cases:
   * Preconfigured test scenarios for algorithms.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

AES Example :

### **Advanced Encryption Standard (AES): Comprehensive Visualization Scope**

AES involves multiple stages of transformations applied to the plaintext to produce the ciphertext. Each stage will be visually represented, step by step, with intuitive animations and interactive elements. Below is the expanded scope.

#### **Visualization Components for AES**

1. **Key Expansion (16 Steps):**
   * **Purpose:** Derive all round keys from the original input key.
   * **Visualization:**
     + **Round Key Table:** A tabular representation displaying how each round key is generated.
     + **Step-by-Step Process:**
       - **RotWord Operation:**
         * Rotate the bytes of a word (e.g., [01, 02, 03, 04] → [02, 03, 04, 01]).
         * Visualize byte-by-byte shifting in a **horizontal animation**.
       - **SubWord Operation:**
         * Substitute each byte using the S-Box.
         * Highlight each byte lookup in the S-Box with **color-coded animations**.
       - **Rcon Application:**
         * Apply the round constant (Rcon) to the first word of the round key.
         * Display each XOR operation at the bit level.
       - **Key Calculation:**
         * Show how the new key is formed by XORing the previous round key with the modified word.
       - **Output:** Display all round keys in a **Key Expansion Table**.
2. **State Representation (4x4 Matrix):**
   * **Purpose:** Represent the plaintext (or intermediate state) as a grid of bytes.
   * **Visualization:**
     + Use a **dynamic 4x4 matrix box** to show transformations after each step.
     + Highlight the flow of data as it moves through rounds.
3. **Round Operations (10/12/14 Rounds):** AES has several stages per round:  
   * **AddRoundKey:**
     + XOR operation between the current state and the round key.
     + Visualize this by animating bit-by-bit XOR transformations.
     + Highlight bytes in the matrix as they change.
   * **SubBytes (S-Box):**
     + Replace each byte of the state with its corresponding value in the S-Box.
     + **Interactive S-Box Table:**
       - Users can hover over bytes in the state to see the corresponding substitution in the S-Box.
       - Highlight the substituted byte in both the matrix and the S-Box.
   * **ShiftRows:**
     + Shift the rows of the state matrix:
       - Row 0: No shift.
       - Row 1: Shift left by 1 byte.
       - Row 2: Shift left by 2 bytes.
       - Row 3: Shift left by 3 bytes.
     + Animate the row-by-row shifting with **arrows and transitions**.
   * **MixColumns:**
     + Transform each column of the state matrix using matrix multiplication in GF(2^8).
     + **Interactive MixColumns Tool:**
       - Show the multiplication and addition operations for each byte.
       - Visualize intermediate results using tables and matrix diagrams.
   * **Final Round (No MixColumns):**
     + Omit the MixColumns step in the final round and directly move to AddRoundKey.
4. **Initial and Final Transformations:**
   * **Initial AddRoundKey:** Show how the plaintext is XORed with the initial round key before entering the rounds.
   * **Final State to Ciphertext:**
     + Display the final state as the ciphertext in hexadecimal format.
     + Highlight how each byte of the matrix contributes to the final ciphertext.

#### **Detailed Box Components**

Each step of AES will use distinct visual elements:

1. **Key Expansion Table:**
   * Display the progression of key generation for all 16 rounds.
   * Allow users to click on a round to see detailed calculations (RotWord, SubWord, Rcon, XOR).
2. **State Matrix Box:**
   * A 4x4 grid dynamically updating to show the current state after each operation.
3. **S-Box Lookup Table:**
   * Interactive substitution table for byte transformations.
4. **MixColumns Matrix:**
   * Visual representation of the column transformations.
   * Display each multiplication and addition operation in detail.
5. **Round Constant (Rcon) Table:**
   * Show the constants applied during key expansion.

#### **Process Flow for AES Visualization**

1. **Input Stage:**
   * **User Inputs:**
     + Plaintext (16 bytes) in hexadecimal or ASCII format.
     + Key (16 bytes) in hexadecimal or ASCII format.
   * Provide real-time validation for input sizes.
   * Show a **Preview Panel** displaying the input state matrix.
2. **Initial AddRoundKey:**
   * Animate the XOR operation between plaintext and the initial key.
   * Highlight each bit that changes.
3. **Rounds 1 to 9 (or 11/13):**
   * For each round, animate all operations sequentially:
     + **SubBytes:** Highlight byte-by-byte substitutions.
     + **ShiftRows:** Animate the row-wise shifting.
     + **MixColumns:** Step-by-step visualization of column transformations.
     + **AddRoundKey:** Display the bitwise XOR operation.
4. **Final Round (Round 10/12/14):**
   * Exclude the MixColumns step.
   * Animate the final SubBytes, ShiftRows, and AddRoundKey operations.
5. **Output Stage:**
   * Show the final ciphertext in hexadecimal format.
   * Allow users to download the ciphertext or intermediate steps as a **JSON or PDF report**.

#### **Advanced Features**

1. **Interactive Mode:**
   * Allow users to hover over any byte or matrix element to see its history.
   * Enable toggling between binary, hexadecimal, and decimal views for clarity.
2. **Step-by-Step Replay:**
   * Provide controls to pause, rewind, or replay any stage of the visualization.
3. **Comparison Tool:**
   * Allow users to compare two plaintexts or keys to observe how minor changes (e.g., single-bit differences) affect the final ciphertext (avalanche effect).
4. **Customizable Rounds:**
   * Enable users to select the number of rounds for educational purposes.

#### **Deliverables for AES Visualization**

1. **Interactive UI Components:**
   * Fully interactive, step-by-step AES visualization tool.
2. **Export Options:**
   * PDF or PNG export of intermediate states and final results.
   * JSON export of round-wise transformations and key expansions.
3. **User Documentation:**
   * A comprehensive guide explaining AES and how to use the visualization tool.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

### **Blowfish: Comprehensive Visualization Scope**

Blowfish is a symmetric block cipher designed for fast encryption and decryption. It operates on 64-bit blocks with a variable key size (32 to 448 bits) and uses a Feistel structure with 16 rounds. This detailed scope outlines step-by-step visualization requirements for Blowfish.

#### **Visualization Components for Blowfish**

1. **Key Expansion and Subkey Generation:**
   * **Purpose:** Generate subkeys PP-Array (18 keys) and four SS-Boxes (256 entries each).
   * **Visualization:**
     + **P-Array Generation:**
       - Display the initial PP-Array populated with the hexadecimal digits of Pi.
       - Animate the XOR operation with the input key to create the subkeys.
     + **S-Box Initialization:**
       - Show the initialization of the SS-Boxes with Pi's digits.
       - Highlight how each entry is modified during subkey generation.
   * **Step-by-Step Process:**
     + **Input Key Transformation:**
       - Visualize how the key bits are applied to update PP-Array and SS-Boxes.
     + **Encryption of Zero Blocks:**
       - Animate how a 64-bit block of zeros is encrypted to refine the subkeys.
2. **Data Encryption (Feistel Structure):**
   * **Purpose:** Encrypt 64-bit blocks of plaintext using 16 rounds of Feistel operations.
   * **Visualization:**
     + **State Representation:**
       - Represent the left (L) and right (R) halves of the plaintext as two boxes.
       - Show the step-by-step transformation of LL and RR in each round.
     + **F-Function (Core Operation):**
       - Visualize the process inside the FF-Function:
         * **Input Splitting:** Divide the 32-bit input into four 8-bit sections.
         * **S-Box Lookups:** Show lookup operations in the SS-Boxes for each section.
         * **Mathematical Operations:** Highlight bitwise XORs and additions performed in the FF-Function.
     + **Round Key Addition:**
       - Show how each P[i]P[i] subkey is added during the XOR operation in every round.
     + **Final Swap:**
       - Display the swap of LL and RR after the 16th round.
   * **Intermediate Outputs:**
     + Animate the updated values of LL and RR after each round.
3. **Decryption:**
   * **Purpose:** Reverse the encryption process using the same subkeys in reverse order.
   * **Visualization:**
     + Use the same Feistel structure visualization as encryption but reverse the order of PP-Array keys.
     + Highlight the symmetric nature of the algorithm.
4. **Final Output:**
   * Display the final ciphertext (or decrypted plaintext) in hexadecimal format.

#### **Detailed Box Components**

1. **P-Array Table:**
   * A dynamic table displaying all 18 PP-Array subkeys.
   * Highlight updates during key expansion and each round of encryption.
2. **S-Box Tables (Four 256-Entry Boxes):**
   * Interactive tables to display the initial values and changes in the SS-Boxes during key expansion.
   * Allow users to hover over entries to see their transformations.
3. **State Boxes (Left and Right Halves):**
   * Two boxes to show the LL and RR values for each round.
   * Highlight updates with animations.
4. **F-Function Diagram:**
   * A flow diagram illustrating the splitting, S-Box lookups, and mathematical operations within the FF-Function.

#### **Process Flow for Blowfish Visualization**

1. **Input Stage:**
   * **User Inputs:**
     + Plaintext (64 bits) in hexadecimal or binary format.
     + Key (32 to 448 bits) in hexadecimal or ASCII format.
   * Provide real-time validation for key size and plaintext format.
   * Display a **Preview Panel** showing the initial state.
2. **Key Expansion:**
   * **P-Array Updates:**
     + Visualize the XOR operation between the key and initial PP-Array.
   * **S-Box Updates:**
     + Animate how each SS-Box entry is refined using encrypted zero blocks.
3. **Encryption Process:**
   * **Round-by-Round Visualization:**
     + Highlight the transformation of LL and RR after each round.
     + Show intermediate FF-Function results, including:
       - S-Box lookups.
       - XOR and addition operations.
   * **Final Swap:**
     + Display the swap of LL and RR values.
4. **Decryption Process:**
   * Use the same structure as encryption, but reverse the PP-Array key order.
   * Animate intermediate transformations in reverse.
5. **Output Stage:**
   * Display the final ciphertext (or decrypted plaintext) in hexadecimal format.
   * Allow users to download the result as a **JSON or PDF report**.

#### **Advanced Features**

1. **Interactive Mode:**
   * Allow users to hover over PP-Array and SS-Box entries to see their transformations.
   * Enable toggling between binary, hexadecimal, and decimal views for clarity.
2. **Step-by-Step Replay:**
   * Provide controls to pause, rewind, or replay any stage of the visualization.
3. **Comparison Tool:**
   * Compare ciphertext outputs for different plaintexts or keys to observe the avalanche effect.
4. **Customizable Rounds:**
   * Enable users to select fewer rounds for educational purposes.

#### **Deliverables for Blowfish Visualization**

1. **Interactive UI Components:**
   * Fully interactive, step-by-step Blowfish visualization tool.
2. **Export Options:**
   * PDF or PNG export of intermediate states and final results.
   * JSON export of P-Array, S-Boxes, and round-wise transformations.
3. **User Documentation:**
   * A comprehensive guide explaining Blowfish and how to use the visualization tool.

### **RSA: Comprehensive Visualization Scope**

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem used for secure data transmission. It involves three primary stages: key generation, encryption, and decryption. This scope provides step-by-step visualization requirements for all these stages.

#### **Visualization Components for RSA**

1. **Key Generation:**
   * **Purpose:** Generate a public and private key pair.
   * **Visualization:**
     + **Prime Number Selection:**
       - Animate the selection of two large prime numbers pp and qq.
       - Allow users to input their own primes or visualize the tool selecting primes.
     + **Modulus Calculation:**
       - Show the computation of n=p×qn = p \times q with intermediate steps.
       - Display nn as the modulus for both the public and private keys.
     + **Euler’s Totient Function:**
       - Visualize the calculation of ϕ(n)=(p−1)×(q−1)\phi(n) = (p-1) \times (q-1).
     + **Public Exponent Selection (ee):**
       - Allow users to select ee or show the tool picking a suitable ee.
       - Validate ee by showing that it is coprime with ϕ(n)\phi(n).
     + **Private Key Calculation (dd):**
       - Display the computation of d=e−1mod  ϕ(n)d = e^{-1} \mod \phi(n) (modular multiplicative inverse).
       - Highlight intermediate steps using the **Extended Euclidean Algorithm**.
   * **Output:** Display the public key (n,e)(n, e) and private key (n,d)(n, d).
2. **Encryption:**
   * **Purpose:** Encrypt plaintext using the public key.
   * **Visualization:**
     + **Plaintext Preparation:**
       - Convert plaintext characters to numerical values (ASCII or UTF-8).
       - Split the message into blocks smaller than nn.
     + **Encryption Formula:** C=Memod  nC = M^e \mod n
       - Animate the modular exponentiation process for each block.
       - Highlight intermediate calculations:
         * Powers of MM.
         * Modulo operation at each step.
   * **Output:** Display the ciphertext blocks in hexadecimal or binary format.
3. **Decryption:**
   * **Purpose:** Recover plaintext using the private key.
   * **Visualization:**
     + **Decryption Formula:** M=Cdmod  nM = C^d \mod n
       - Animate the modular exponentiation process for each block.
       - Highlight intermediate calculations:
         * Powers of CC.
         * Modulo operation at each step.
     + **Plaintext Reconstruction:**
       - Convert numerical values back to characters to reconstruct the original message.
   * **Output:** Display the recovered plaintext.

#### **Detailed Box Components**

1. **Key Generation Table:**
   * Display pp, qq, nn, ϕ(n)\phi(n), ee, and dd values in a structured format.
   * Highlight the progression of calculations step by step.
2. **Modular Exponentiation Table:**
   * Visualize the step-by-step calculation for Memod  nM^e \mod n or Cdmod  nC^d \mod n.
   * Include intermediate results for powers and modulo reductions.
3. **Ciphertext and Plaintext Boxes:**
   * Separate boxes for displaying the numerical and textual representations of ciphertext and plaintext.
4. **Interactive Diagrams:**
   * Represent the flow of data through encryption and decryption using arrows and labeled components.

#### **Process Flow for RSA Visualization**

1. **Key Generation:**
   * **Step 1: Prime Selection:**
     + Allow users to input or visualize pp and qq.
     + Animate the primality test using trial division or the Miller-Rabin algorithm.
   * **Step 2: Modulus (nn) and Totient (ϕ(n)\phi(n)) Calculation:**
     + Show intermediate multiplication and subtraction steps.
   * **Step 3: Public Key (ee):**
     + Highlight ee selection and the coprimality check with ϕ(n)\phi(n).
   * **Step 4: Private Key (dd):**
     + Use the Extended Euclidean Algorithm to compute dd with step-by-step explanation.
2. **Encryption:**
   * Convert plaintext into numeric blocks.
   * Animate the modular exponentiation process for C=Memod  nC = M^e \mod n.
   * Display the ciphertext in both numerical and hexadecimal formats.
3. **Decryption:**
   * Input ciphertext blocks.
   * Animate the modular exponentiation process for M=Cdmod  nM = C^d \mod n.
   * Convert numeric results back into text.
4. **Output Stage:**
   * Show the complete encryption-decryption cycle with:
     + Plaintext → Ciphertext → Recovered Plaintext.
   * Highlight the symmetry between encryption and decryption processes.

#### **Advanced Features**

1. **Interactive Mode:**
   * Allow users to hover over any value to see its calculation history.
   * Provide tooltips explaining modular arithmetic concepts.
2. **Visualization of Prime Testing:**
   * Step-by-step animation of primality tests (e.g., trial division or Miller-Rabin).
3. **Custom Inputs:**
   * Enable users to provide their own primes, ee, or plaintext.
4. **Side-by-Side Comparisons:**
   * Allow comparison of encryption and decryption processes with different keys.
5. **Replay Controls:**
   * Include options to pause, rewind, or replay any stage of the visualization.

#### **Deliverables for RSA Visualization**

1. **Interactive UI Components:**
   * Fully interactive visualization for RSA key generation, encryption, and decryption.
2. **Export Options:**
   * PDF or PNG export of key generation steps and encryption-decryption results.
   * JSON export of intermediate calculations and keys.
3. **User Documentation:**
   * A guide explaining RSA and how to use the visualization tool.

### **SHA (Secure Hash Algorithm): Comprehensive Visualization Scope**

SHA (e.g., **SHA-256**) is a cryptographic hash function that produces a fixed-size hash (digest) from any input data. The visualization will focus on every step, from preprocessing to hash generation.

#### **Visualization Components for SHA**

1. **Preprocessing:**
   * **Purpose:** Prepare the input for the hashing process.
   * **Visualization:**
     + **Padding the Message:**
       - Display the original message in binary format.
       - Animate the addition of a single 1 bit, followed by 0 bits, to make the length congruent to 448mod  512448 \mod 512448mod512.
       - Show the addition of the message length as a 64-bit binary value.
     + **Chunk Splitting:**
       - Split the padded message into 512-bit chunks.
       - Highlight each chunk as it is processed.
2. **Message Schedule (W[0]–W[63]):**
   * **Purpose:** Generate a schedule of 64 words (32-bit each) for each chunk.
   * **Visualization:**
     + Represent the 512-bit chunk as a **table of 16 words** (32 bits each).
     + Animate the extension of these 16 words into 64 words using:
       - Logical operations like XOR, right shift (>>\text{>>}>>), and rotate right (ROTR\text{ROTR}ROTR).
       - Step-by-step calculation for each W[t]W[t]W[t], showing intermediate results.
3. **Hash Computation (Compression Function):**
   * **Purpose:** Process each 512-bit chunk to update hash values.
   * **Visualization:**
     + **Initialization:**
       - Display the eight initial hash values (H[0]H[0]H[0] to H[7]H[7]H[7]).
       - Allow users to hover over these values to see their origins (constants derived from the fractional parts of square roots).
     + **Round-by-Round Operations (64 Rounds):**
       - Animate the operations for each round:
         1. **Choice Function (Ch):**

Visualize the formula (E∧F)⊕(¬E∧G)(E \land F) \oplus (\lnot E \land G)(E∧F)⊕(¬E∧G).

* + - * 1. **Majority Function (Maj):**

Visualize the formula (A∧B)⊕(A∧C)⊕(B∧C)(A \land B) \oplus (A \land C) \oplus (B \land C)(A∧B)⊕(A∧C)⊕(B∧C).

* + - * 1. **Sigma Functions (Σ0\Sigma\_0Σ0​, Σ1\Sigma\_1Σ1​):**

Show rotate and shift operations for AAA, BBB, and EEE.

* + - * 1. **Update Operations:**

Update AAA through HHH values at each step.

* + - * Highlight the intermediate hash values after each round.

1. **Final Hash Value:**
   * **Purpose:** Produce the 256-bit hash.
   * **Visualization:**
     + Concatenate H[0]H[0]H[0] through H[7]H[7]H[7] to form the final hash.
     + Display the hash in hexadecimal format.

#### **Detailed Box Components for SHA**

1. **Message Schedule Table (W):**
   * Display 64 words for each chunk.
   * Highlight each word as it is used in a round.
2. **Round Operation Table:**
   * Show the values of AAA to HHH before and after each round.
3. **Hash Value Table:**
   * Show the updated hash values after processing each chunk.
4. **Interactive Diagrams:**
   * Represent the flow of data through logical operations and functions.

#### **Process Flow for SHA Visualization**

1. **Input Stage:**
   * **User Inputs:** Allow users to input a message.
   * Convert the message into binary format and display it.
2. **Preprocessing:**
   * Animate the padding and chunk splitting process.
3. **Message Schedule Generation:**
   * Display the calculation of W[0]W[0]W[0] to W[63]W[63]W[63] for each chunk.
4. **Hash Computation:**
   * Animate each round's operations, highlighting changes to AAA through HHH.
5. **Output Stage:**
   * Display the final 256-bit hash in hexadecimal format.

### **MD5 (Message Digest 5): Comprehensive Visualization Scope**

MD5 is a cryptographic hash function that produces a 128-bit hash value. Although considered insecure for cryptographic purposes, it remains useful for checksums.

#### **Visualization Components for MD5**

1. **Preprocessing:**
   * **Purpose:** Prepare the input for hashing.
   * **Visualization:**
     + **Padding the Message:**
       - Display the message in binary format.
       - Animate the addition of a single 1 bit, followed by 0 bits, to make the length congruent to 448mod  512448 \mod 512448mod512.
       - Show the addition of the message length as a 64-bit binary value.
     + **Chunk Splitting:**
       - Split the padded message into 512-bit chunks.
       - Highlight each chunk as it is processed.
2. **Initialization of Buffers:**
   * **Purpose:** Set up the four buffers AAA, BBB, CCC, and DDD.
   * **Visualization:**
     + Display the initial values of AAA, BBB, CCC, and DDD derived from the fractional parts of square roots.
3. **Hash Computation (Compression Function):**
   * **Purpose:** Process each 512-bit chunk to update buffer values.
   * **Visualization:**
     + **Four Rounds (64 Operations):**
       - Animate the operations in each round:
         1. **Round 1 (F Function):**

Visualize F(X,Y,Z)=(X∧Y)⊕(¬X∧Z)F(X, Y, Z) = (X \land Y) \oplus (\lnot X \land Z)F(X,Y,Z)=(X∧Y)⊕(¬X∧Z).

* + - * 1. **Round 2 (G Function):**

Visualize G(X,Y,Z)=(X∧Z)⊕(Y∧¬Z)G(X, Y, Z) = (X \land Z) \oplus (Y \land \lnot Z)G(X,Y,Z)=(X∧Z)⊕(Y∧¬Z).

* + - * 1. **Round 3 (H Function):**

Visualize H(X,Y,Z)=X⊕Y⊕ZH(X, Y, Z) = X \oplus Y \oplus ZH(X,Y,Z)=X⊕Y⊕Z.

* + - * 1. **Round 4 (I Function):**

Visualize I(X,Y,Z)=Y⊕(X∨¬Z)I(X, Y, Z) = Y \oplus (X \lor \lnot Z)I(X,Y,Z)=Y⊕(X∨¬Z).

* + - * Highlight the shifts, additions, and bitwise operations for each step.
      * Use a **Round Operation Table** to display intermediate buffer values.
    - **Constants and Shifts:**
      * Show the constants (K[i]K[i]K[i]) and left rotation amounts used in each step.
  + **Buffer Updates:**
    - Update AAA, BBB, CCC, and DDD after each operation.
    - Highlight changes to buffer values.

1. **Final Hash Value:**
   * **Purpose:** Concatenate buffer values to form the final 128-bit hash.
   * **Visualization:**
     + Display AAA, BBB, CCC, and DDD as 32-bit hexadecimal values.
     + Concatenate to form the 128-bit hash.

#### **Detailed Box Components for MD5**

1. **Buffer Table:**
   * Show AAA, BBB, CCC, and DDD values during each step.
2. **Round Operation Table:**
   * Display the inputs and outputs of each operation (e.g., F, G, H, I).
3. **Message Table:**
   * Highlight each chunk and its transformations.

#### **Process Flow for MD5 Visualization**

1. **Input Stage:**
   * **User Inputs:** Allow users to input a message.
   * Convert the message into binary format and display it.
2. **Preprocessing:**
   * Animate the padding and chunk splitting process.
3. **Hash Computation:**
   * Animate all 64 operations in four rounds.
   * Highlight the changes to AAA, BBB, CCC, and DDD after each operation.
4. **Output Stage:**
   * Display the final 128-bit hash in hexadecimal format.

### **Advanced Features for Both Algorithms**

1. **Interactive Mode:**
   * Hover over intermediate values to view detailed calculations.
   * Allow toggling between binary, hexadecimal, and decimal formats.
2. **Step-by-Step Replay:**
   * Provide controls to pause, rewind, or replay any stage.
3. **Comparison Tool:**
   * Allow comparison of hash outputs for different inputs to demonstrate sensitivity.
4. **Custom Inputs:**
   * Enable users to input custom messages for hashing.

### **Deliverables for SHA and MD5 Visualization**

1. **Interactive UI Components:**
   * Fully interactive, step-by-step visualization tools for SHA and MD5.
2. **Export Options:**
   * PDF or PNG export of intermediate steps and final results.
   * JSON export of hash computation details.
3. **User Documentation:**
   * Comprehensive guides explaining SHA and MD5 and how to use the tools.