2. Provide the algorithms and source codes of the desktop and mobile applications.

Algorithms :-

Caesar :-

Choose a shift key k.

For each letter in the plaintext:

- **If it's a letter, shift it forward by k positions in the alphabet.**
- **Wrap around if necessary (e.g., 'Z' shifted by 1 becomes 'A').**

Rail Fence :-

- **Choose a rail count R.**
- **Write the message in a zigzag pattern across RRR rows.**
- **Read row-wise to get the ciphertext**

Vigenere :-

- **Choose a keyword and repeat it to match the plaintext length.**
- **Convert letters into numbers (A = 0, ..., Z = 25).**
- **Shift each letter based on the corresponding keyword letter**

Code: -

Decrypt.js

```
const __filename = fileURLToPath(import.meta.url);

const __dirname = join(__filename, "..");

// Load dictionary manually

async function loadDictionary() {

  try {

    const affix = await readFile(join(__dirname, "node_modules", "dictionary-en", "index.aff"),
"utf8");

    const wordList = await readFile(join(__dirname, "node_modules", "dictionary-en", "index.dic"),
"utf8");

    return new nspell(affix, wordList);

  } catch (error) {

    console.error("Error loading dictionary:", error);

    return null;

  }}
```

```
// Function to decrypt Caesar cipher without a key
async function caesarDecryptSmart(ciphertext) {
  const spell = await loadDictionary();
  if (!spell) {
    console.log("Dictionary could not be loaded.");
    return;
  }
  let cleanedText = ciphertext.replace(/\s+/g, '');
  let bestMatch = { text: '', score: 0, key: 0, spaced: '' };

  for (let shift = 1; shift < 26; shift++) {
    let decryptedText = "";
    for (let i = 0; i < cleanedText.length; i++) {
      let char = cleanedText[i];
      if (char.match(/[a-zA-Z]/)) {
        let base = char === char.toUpperCase() ? 65 : 97;
        let decryptedChar = String.fromCharCode(((char.charCodeAt(0) - base - shift + 26) % 26) +
base);
        decryptedText += decryptedChar;
      } else {
        decryptedText += char;
      } }
    let { score, words } = isReadable(decryptedText);
    if (score > bestMatch.score) {
      bestMatch = { text: decryptedText, score, key: shift, spaced: words.join(' ') };
    }  }
  return bestMatch;
}
// Decrypt Rail Fence Cipher
```

```
function decryptRailFence(cipher, key) {
  let rail = new Array(key).fill(0).map(() => new Array(cipher.length).fill('\n'));
  let dir_down;
  let row = 0, col = 0;
  for (let i = 0; i < cipher.length; i++) {
    if (row === 0) dir_down = true;
    if (row === key - 1) dir_down = false;
    rail[row][col++] = '*';
    dir_down ? row++ : row--;
  }
  let index = 0;
  for (let i = 0; i < key; i++) {
    for (let j = 0; j < cipher.length; j++) {
      if (rail[i][j] === '*' && index < cipher.length) {
        rail[i][j] = cipher[index++];
      }      }   }
  let result = '';
  row = 0;
  col = 0;
  for (let i = 0; i < cipher.length; i++) {
    if (row === 0) dir_down = true;
    if (row === key - 1) dir_down = false;

    if (rail[row][col] !== '*') {
      result += rail[row][col++];
    }
    dir_down ? row++ : row--;
  }
  return result;
```

```
}
// Restore proper word spacing using a backtracking method
function segmentText(text, spell) {
  let result = [];
  function backtrack(start, path) {
    if (start === text.length) {
      let sentence = path.join(" ");
      let validCount = sentence.split(" ").filter(word => spell.correct(word)).length;
      if (validCount > result.length) result = path.slice();
      return;
    }
    for (let end = start + 1; end <= text.length; end++) {
      let word = text.slice(start, end);
      if (spell.correct(word)) {
        path.push(word);
        backtrack(end, path);
        path.pop();
      }      }   }
  backtrack(0, []);
  return result.length ? result.join(" ") : text;
}
//  Find best decryption with proper spacing
async function bestRailFenceDecrypt(cipherText) {
  const spell = await loadDictionary();
  if (!spell) {
    console.log("Dictionary could not be loaded.");
    return;
  }
  function splitWords(text) {
        let result = [];
        let i = 0;
        while (i < text.length) {
            let found = false;
            for (let j = Math.min(text.length, i + 20); j > i; j--) {
                let word = text.slice(i, j).toLowerCase();
                if (spell.correct(word)) {
                    result.push(text.slice(i, j));
                    i = j;
                    found = true;
                    break;
                }
            }
            if (!found) {
                result.push(text[i]);
                i++;
            }      }
        return result;
    }
    function isReadable(text) {
        let words = splitWords(text);
        let totalWords = words.length;
        if (totalWords === 0) return { isReadable: false, score: 0 };
        let validLongWords = words.filter(word => word.length > 2 &&
spell.correct(word.toLowerCase())).length;
        let score = validLongWords / totalWords;
        return { isReadable: score >= 0.3, score: score, words: words };
    }
    let bestMatch = { text: '', score: 0, key: 0, spaced: '' };
    for (let key = 2; key <= Math.min(cipherText.length, 10); key++) {
        let decryptedText = decryptRailFence(cipherText, key);
        let { score, words } = isReadable(decryptedText);

        if (score > bestMatch.score) {
            bestMatch = { text: decryptedText, score, key, spaced: words.join(' ') };
        }   }
    return bestMatch && bestMatch.score > 0 ? bestMatch : { text: "No readable text found." };
}
function decryptVigenere(text, key) {
    let result = '';
    key = key.toUpperCase();
    for (let i = 0, j = 0; i < text.length; i++) {
        let c = text[i];
        if (c.match(/[A-Z]/)) {
            let shift = key.charCodeAt(j % key.length) - 65;
            let decryptedChar = String.fromCharCode(((c.charCodeAt(0) - 65 - shift + 26) % 26) + 65);
            result += decryptedChar;
            j++;
        } else {
            result += c;
        }   }
    return result;
}
function splitWords(text, spell) {
    let result = [];
    let i = 0;
    while (i < text.length) {
        let found = false;
        for (let j = Math.min(text.length, i + 20); j > i; j--) {
            let word = text.slice(i, j).toLowerCase();
            if (spell.correct(word)) {
                result.push(text.slice(i, j));
                i = j;
                found = true;
                break;
            }       }
        if (!found) {
            result.push(text[i]);
            i++;
        }   }
    return result;
}
async function bestVigenereDecrypt(ciphertext) {
    if (!ciphertext) return "Error: No ciphertext provided.";
    const spell = await loadDictionary();
    const wordlist = fs.readFileSync('dictionary.txt', 'utf-8').split(/\r?\n/).filter(w => w.length > 0);
    // const ciphertext = 'LXFOPVEFRNHR'; // Change this with your cipher
    const cleanedText = ciphertext.replace(/\s+/g, '').toUpperCase();
    let candidates = [];
    for (let keyLen = 2; keyLen <= 15; keyLen++) {
        let keys = wordlist.filter(word => word.length === keyLen);
        for (let key of keys) {
            let decrypted = decryptVigenere(cleanedText, key);
            let { isReadable: ok, score, words } = isReadable(decrypted, spell);
            candidates.push({ key, length: keyLen, decrypted, score, words });
        }   }
    // Sort logic with a slight preference for shorter keys when close in score
```

```
  candidates.sort((a, b) => {
    const diff = b.score - a.score;
    if (Math.abs(diff) < 0.03) {
      return a.length - b.length; // prefer shorter key if score is close
    }
    return b.score - a.score;
  });
  const best = candidates[0];
  return best && best.score > 0.65
    ? best
    : { text: "No readable text found." }; }
```

Detect.js

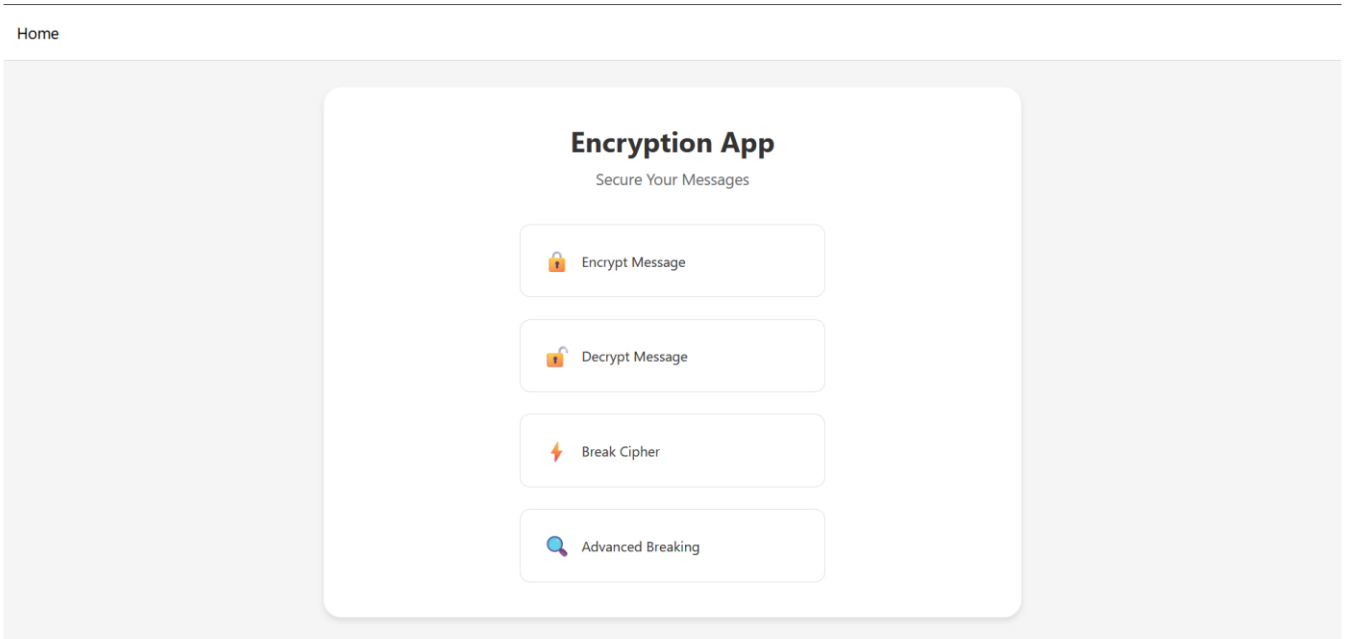
```
const __filename = fileURLToPath(import.meta.url);
const __dirname = join(__filename, "..");
async function loadDictionary() {
  try {
    const affix = await readFile(join(__dirname, "node_modules", "dictionary-en", "index.aff"), "utf8");
    const wordList = await readFile(join(__dirname, "node_modules", "dictionary-en", "index.dic"), "utf8");
    return new nspell(affix, wordList);
  } catch (error) {
    console.error("Error loading dictionary:", error);
    return null;
  }}
async function detectAndDecrypt(ciphertext) {
  const spell = await loadDictionary();
```

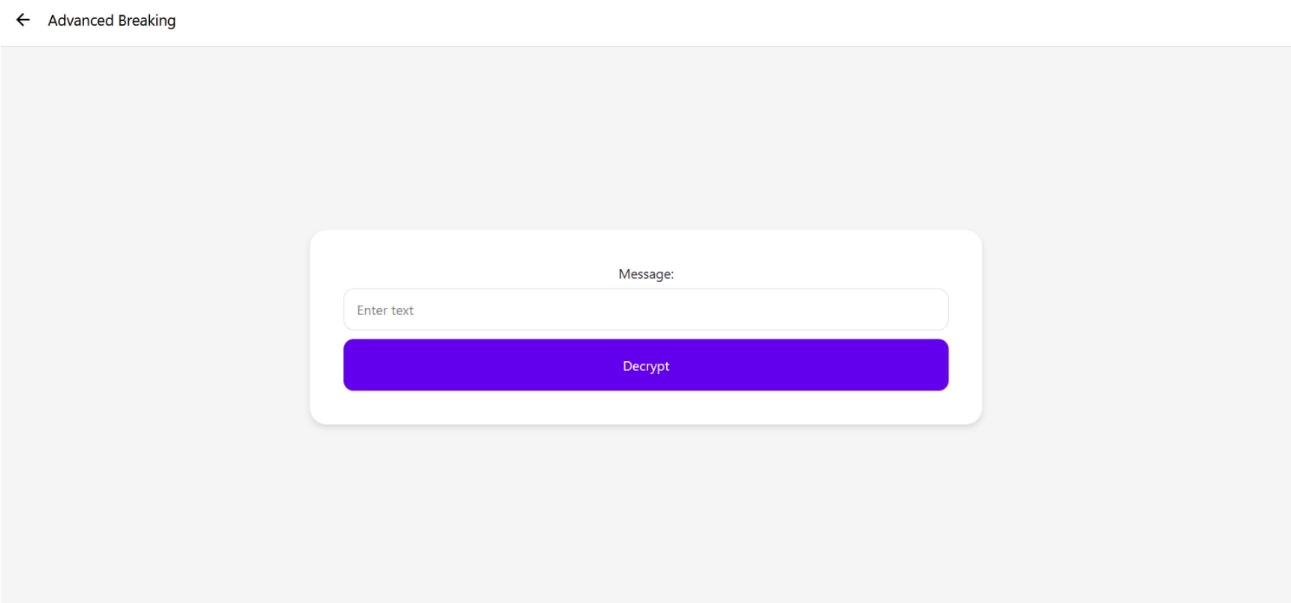
```
    const wordlist = fs.readFileSync('dictionary.txt', 'utf-8').split(/\r?\n/).filter(w => w.length > 0);

    function splitWords(text) {

        let result = [];

        let i = 0;

        while (i < text.length) {

            let found = false;

            for (let j = Math.min(text.length, i + 20); j > i; j--) {

                let word = text.slice(i, j).toLowerCase();

                if (spell.correct(word)) {

                    result.push(text.slice(i, j));

                    i = j;

                    found = true;

                    break;

                }          }

            if (!found) {

                result.push(text[i]);

                i++;

            }       }       return result;    }

    function isReadable(text) {

        let words = splitWords(text);

        let totalWords = words.length;

        if (totalWords === 0) return { isReadable: false, score: 0 };

        let validLongWords = words.filter(word => word.length > 2 &&
spell.correct(word.toLowerCase())).length;

        let score = validLongWords / totalWords;

        return { isReadable: score >= 0.3, score: score, words: words };

    }

    function caesarDecrypt(ciphertext, shift) {

        return ciphertext.split('').map(char => {
```

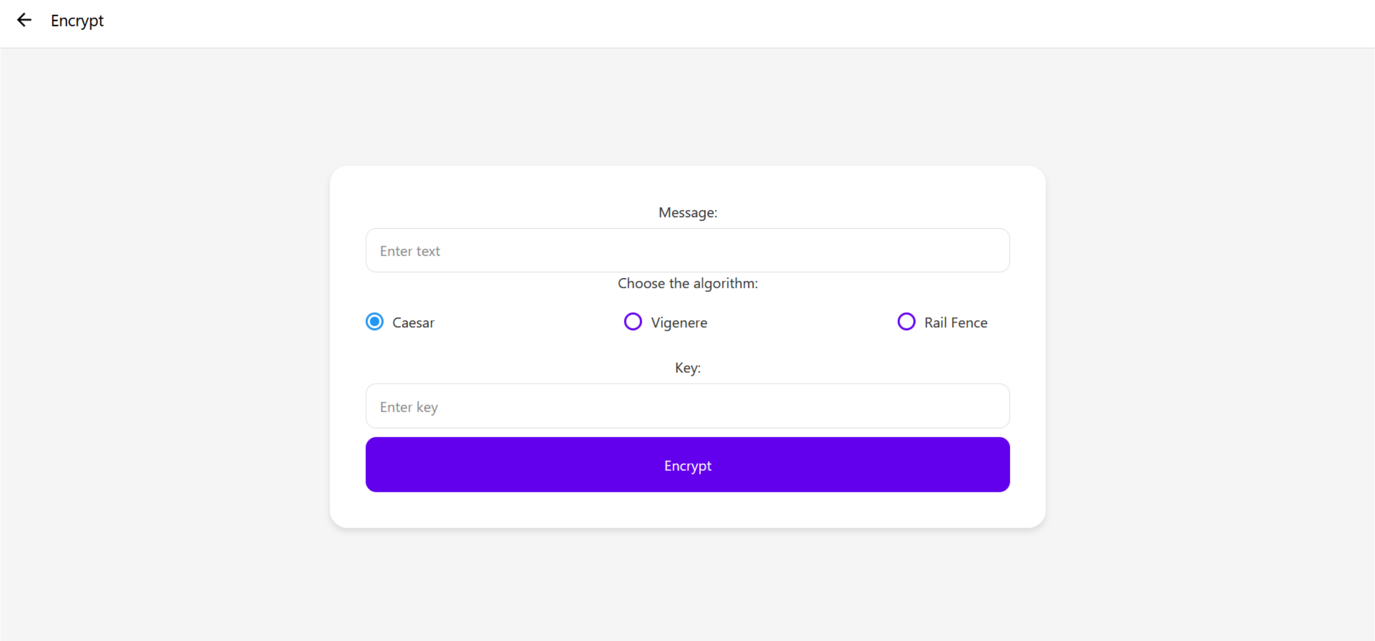
```
        if (char.match(/[a-zA-Z]/)) {
            let code = char.charCodeAt(0);
            let base = code >= 65 && code <= 90 ? 65 : 97;
            return String.fromCharCode(((code - base - shift + 26) % 26) + base);
        }
        return char;
    }).join('');
}
function decryptRailFence(ciphertext, key, reverse = false) {
    if (key <= 1) return ciphertext;
    let rail = Array.from({ length: key }, () => Array(ciphertext.length).fill(null));
    let dir_down = null;
    let row = 0, col = 0;

    for (let i = 0; i < ciphertext.length; i++) {
        if (row === 0) dir_down = true;
        if (row === key - 1) dir_down = false;
        rail[row][col++] = '*';
        row += dir_down ? 1 : -1;
    }
    let index = 0;
    for (let i = 0; i < key; i++) {
        for (let j = 0; j < ciphertext.length; j++) {
            if (rail[i][j] === '*' && index < ciphertext.length) {
                rail[i][j] = ciphertext[index++];
            }        }      }
    let result = '';
    row = 0;
    col = 0;
```

```
  dir_down = null;
  for (let i = 0; i < ciphertext.length; i++) {
    if (row === 0) dir_down = !reverse;
    if (row === key - 1) dir_down = reverse;
    if (rail[row] && rail[row][col] !== null) {
      result += rail[row][col++];
    } else {
      col++;
    }          row += dir_down ? 1 : -1;
  }      return result;    }
function decryptVigenere(text, key) {
  let result = '';
  key = key.toUpperCase();
  for (let i = 0, j = 0; i < text.length; i++) {
    let c = text[i];
    if (c.match(/[A-Z]/)) {
      let shift = key.charCodeAt(j % key.length) - 65;
      let decryptedChar = String.fromCharCode(((c.charCodeAt(0) - 65 - shift + 26) % 26) + 65);
      result += decryptedChar;
      j++;
    } else {
      result += c;          }       }
  return result;    }
let cleanedText = ciphertext.replace(/\s+/g, '').toUpperCase();
let bestMatch = { text: '', score: 0, key: 0, reverse: false, type: '', spaced: '' };
for (let shift = 1; shift < 26; shift++) {
  let decrypted = caesarDecrypt(cleanedText, shift);
  let { score, words } = isReadable(decrypted);
  if (score > bestMatch.score) {
    bestMatch = { text: decrypted, score, key: shift, reverse: false, type: 'Caesar', spaced: words.join(' ') };
  }   }
let maxKey = Math.min(Math.floor(cleanedText.length / 2), 15);
for (let key = 2; key <= maxKey; key++) {
  let decrypted1 = decryptRailFence(cleanedText, key, false);
  let { score: score1, words: words1 } = isReadable(decrypted1);
  if (score1 > bestMatch.score) {
    bestMatch = { text: decrypted1, score: score1, key, reverse: false, type: 'Rail Fence', spaced: words1.join(' ') };
  }
  let decrypted2 = decryptRailFence(cleanedText, key, true);
  let { score: score2, words: words2 } = isReadable(decrypted2);
  if (score2 > bestMatch.score) {
    bestMatch = { text: decrypted2, score: score2, key, reverse: true, type: 'Rail Fence', spaced: words2.join(' ') };
  }   }
for (let keyLen = 2; keyLen <= 15; keyLen++) {
  let keys = wordlist.filter(word => word.length === keyLen);
  for (let key of keys) {
    let decrypted = decryptVigenere(cleanedText, key);
    let { score, words } = isReadable(decrypted);
    if (score > bestMatch.score) {
      bestMatch = { text: decrypted, score, key, type: 'Vigenère', spaced: words.join(' ') };
    }      }   }   return bestMatch; }
```

Web Photos: -

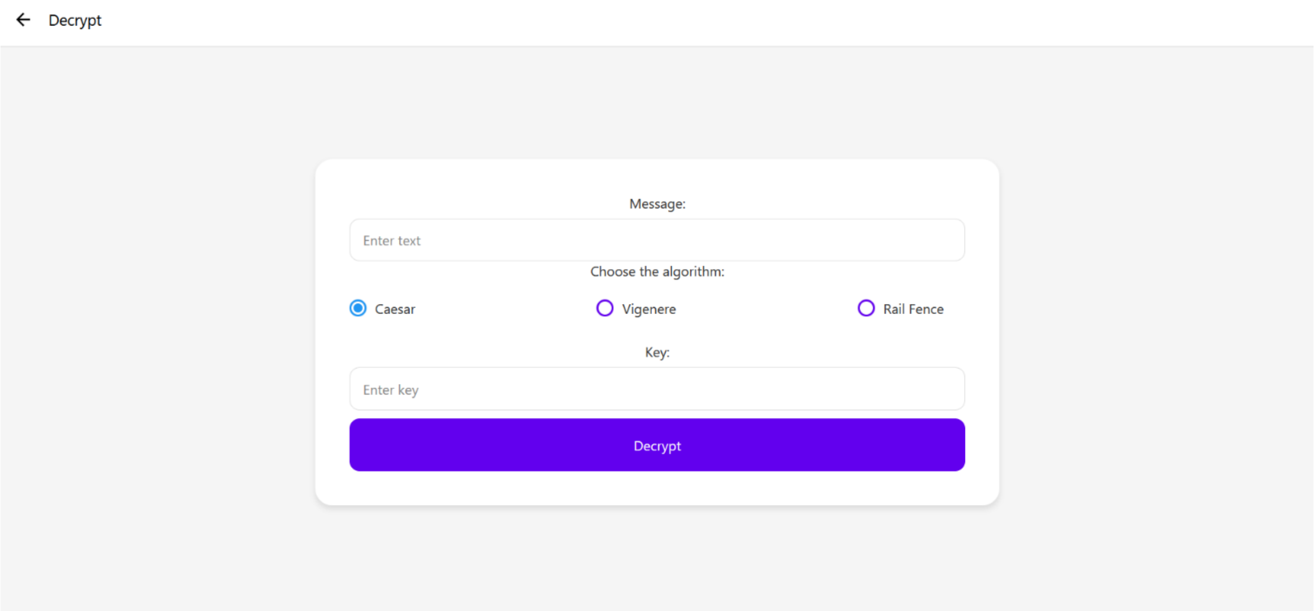

Home
Encryption App
Secure Your Messages
🔒 Encrypt Message
🔓 Decrypt Message
⚡ Break Cipher
🔍 Advanced Breaking


← Advanced Breaking
Message:
Enter text
Decrypt



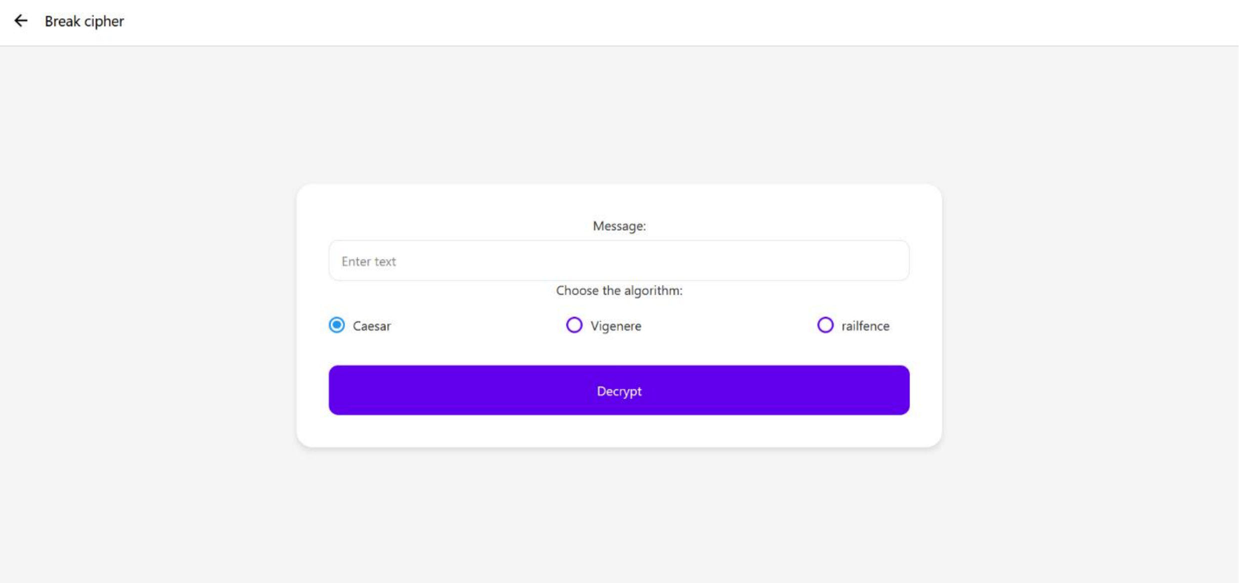
Choose the algorithm:

● Caesar ○ Vigenere ○ Rail Fence



Choose the algorithm:

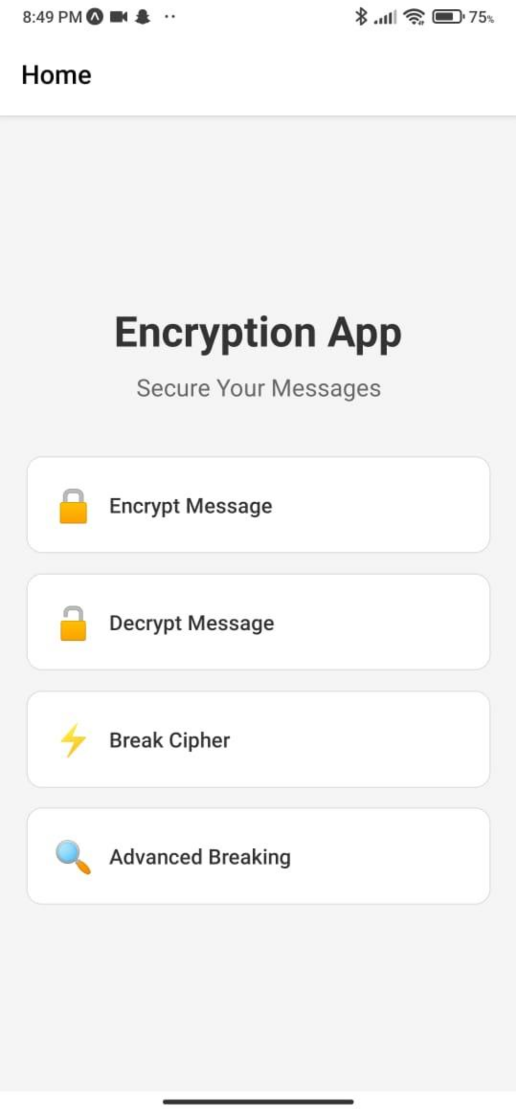
● Caesar ○ Vigenere ○ Rail Fence



Choose the algorithm:

● Caesar ○ Vigenere ○ railfence

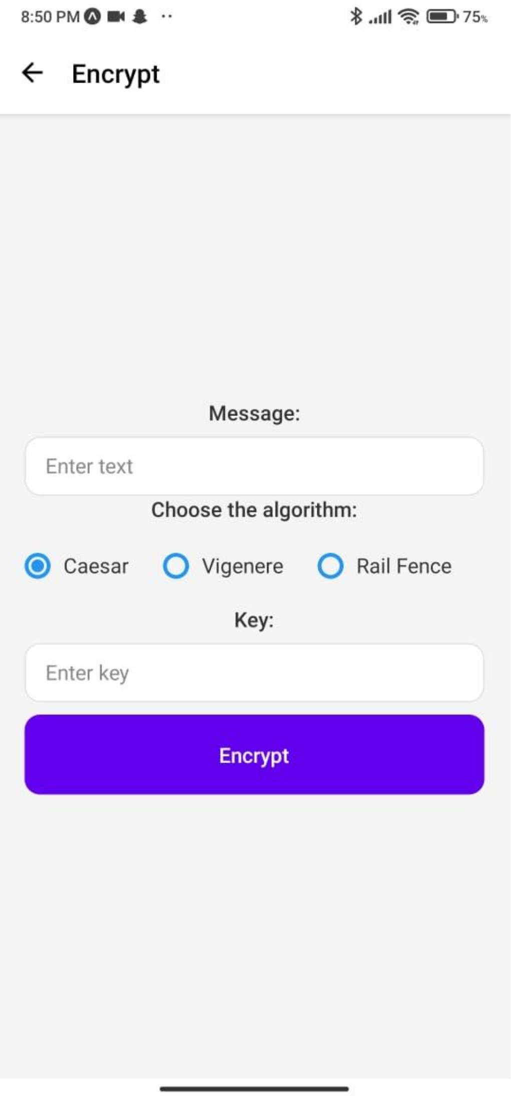
Mobile Photos: -



Home

Encryption App

Secure Your Messages



Message:

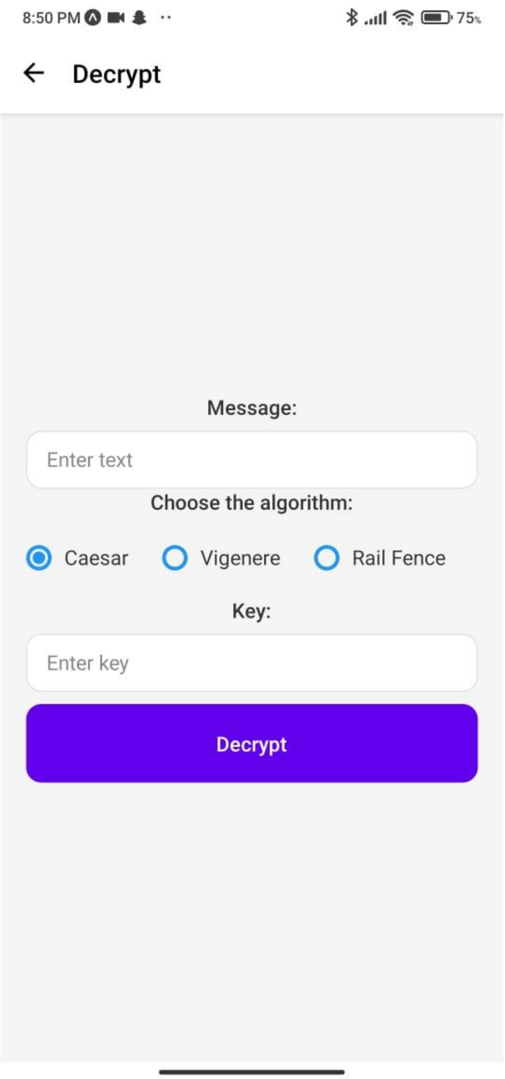
Enter text

Choose the algorithm:

● Caesar ○ Vigenere ○ Rail Fence

Key:

Enter key



Message:

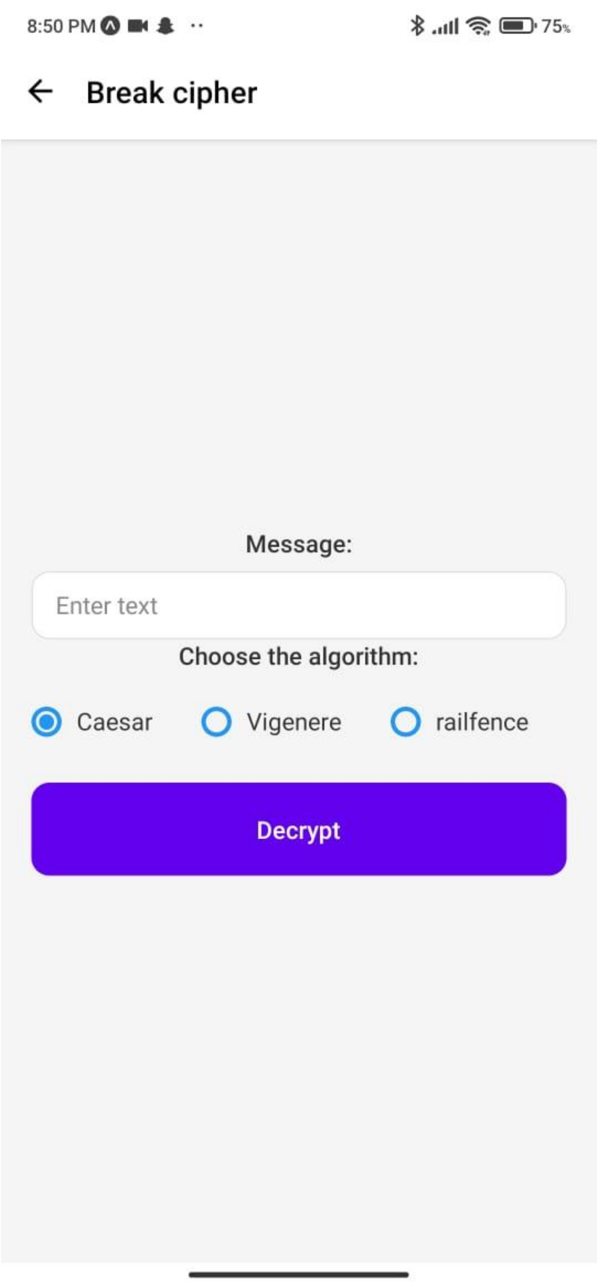
Enter text

Choose the algorithm:

● Caesar ○ Vigenere ○ Rail Fence

Key:

Enter key



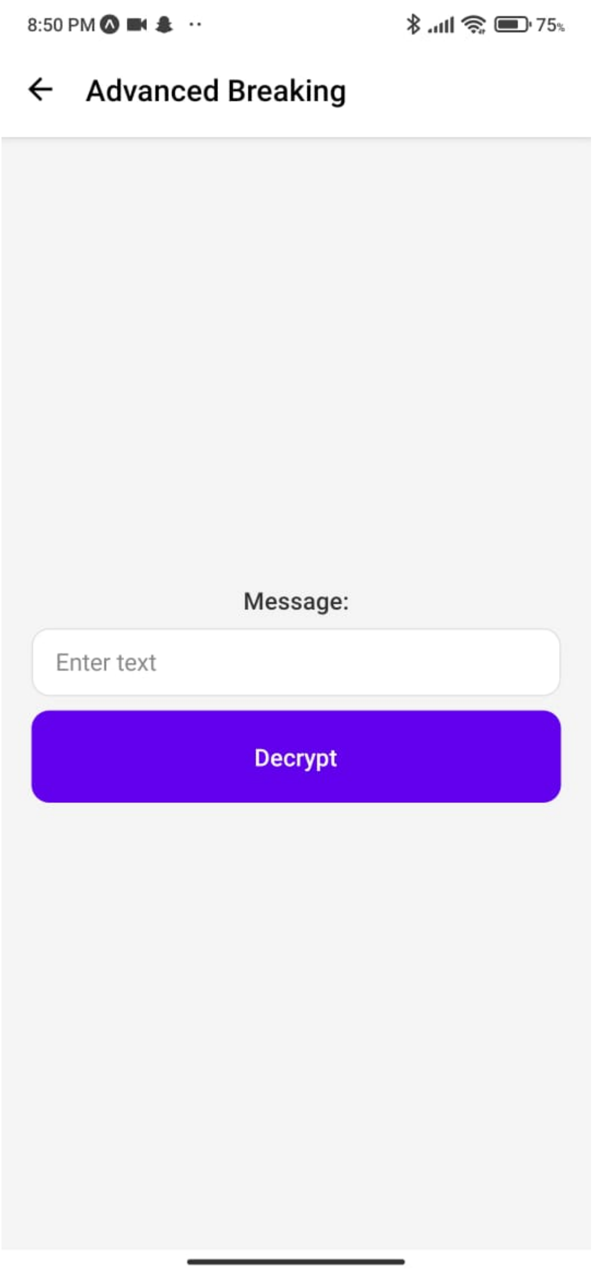
Break cipher

Message:

Enter text

Choose the algorithm:

● Caesar ○ Vigenere ○ railfence



Advanced Breaking

Message:

Enter text