File: .\CryptoApp.java

```
import ciphers.*;
import components.*;
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;
import javax.swing.plaf.basic.BasicScrollBarUI;
import java.awt.*;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CryptoApp extends JFrame {

    private ContentPanel contentPanel;
    private JPanel sidebarPanel;
    private JPanel cipherListPanel;
    private JTextField searchField;
    private List<AnimatedSidebarButton> sidebarButtons = new ArrayList<>();

    // Lazy Loading Maps
    private Map<String, Class<? extends JPanel>> cipherMap = new HashMap<>();
    private Map<String, JPanel> loadedPanels = new HashMap<>();

    public CryptoApp() {
        setTitle("Cryptography Suite - Premium Edition");
        setSize(1300, 850);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);

        // Main Container
        JPanel mainContainer = new JPanel(new BorderLayout());
        mainContainer.setBackground(Theme.MAIN_BG);
        setContentPane(mainContainer);

        // --- Sidebar ---
        sidebarPanel = new JPanel();
        sidebarPanel.setLayout(new BoxLayout(sidebarPanel, BoxLayout.Y_AXIS));
        sidebarPanel.setBackground(Theme.SIDEBAR_BG);
        sidebarPanel.setBorder(new EmptyBorder(30, 20, 30, 20));
        sidebarPanel.setOpaque(true);
        sidebarPanel.setPreferredSize(new Dimension(300, getHeight()));

        JLabel appTitle = new JLabel("CRYPTO SUITE");
        appTitle.setForeground(Theme.TEXT_PRIMARY);
        appTitle.setFont(Theme.FONT_TITLE);
        appTitle.setAlignmentX(Component.LEFT_ALIGNMENT);
        sidebarPanel.add(appTitle);
        sidebarPanel.add(Box.createRigidArea(new Dimension(0, 5)));

        JLabel versionLabel = new JLabel("v2.2 Ultimate");
        versionLabel.setForeground(Theme.ACCENT_COLOR);
```

```
versionLabel.setFont(new Font("Segoe UI", Font.BOLD, 12));
versionLabel.setAlignmentX(Component.LEFT_ALIGNMENT);
sidebarPanel.add(versionLabel);
sidebarPanel.add(Box.createRigidArea(new Dimension(0, 25)));

// Search Bar
searchField = new JTextField();
searchField.setMaximumSize(new Dimension(Integer.MAX_VALUE, 40));
searchField.setFont(Theme.FONT_NORMAL);
searchField.setBackground(Theme.INPUT_BG);
searchField.setForeground(Theme.TEXT_PRIMARY);
searchField.setCaretColor(Theme.ACCENT_COLOR);
searchField.setBorder(BorderFactory.createCompoundBorder(
        new LineBorder(new Color(60, 60, 60), 1, true),
        new EmptyBorder(5, 15, 5, 15)));

searchField.setText("Search Algorithms...");
searchField.addFocusListener(new java.awt.event.FocusAdapter() {
    public void focusGained(java.awt.event.FocusEvent evt) {
        if (searchField.getText().equals("Search Algorithms...")) {
            searchField.setText("");
        }
        searchField.setBorder(BorderFactory.createCompoundBorder(
                new LineBorder(Theme.ACCENT_COLOR, 1, true),
                new EmptyBorder(5, 15, 5, 15)));
    }

    public void focusLost(java.awt.event.FocusEvent evt) {
        if (searchField.getText().isEmpty()) {
            searchField.setText("Search Algorithms...");
        }
        searchField.setBorder(BorderFactory.createCompoundBorder(
                new LineBorder(new Color(60, 60, 60), 1, true),
                new EmptyBorder(5, 15, 5, 15)));
    }
});

searchField.getDocument().addDocumentListener(new javax.swing.event.DocumentListener() {
    public void insertUpdate(javax.swing.event.DocumentEvent e) {
        filterCiphers();
    }

    public void removeUpdate(javax.swing.event.DocumentEvent e) {
        filterCiphers();
    }

    public void changedUpdate(javax.swing.event.DocumentEvent e) {
        filterCiphers();
    }
});

sidebarPanel.add(searchField);
sidebarPanel.add(Box.createRigidArea(new Dimension(0, 20)));
```

```
JLabel subTitle = new JLabel("AVAILABLE MODULES");
subTitle.setForeground(Theme.TEXT_SECONDARY);
subTitle.setFont(new Font("Segoe UI", Font.BOLD, 11));
subTitle.setAlignmentX(Component.LEFT_ALIGNMENT);
sidebarPanel.add(subTitle);
sidebarPanel.add(Box.createRigidArea(new Dimension(0, 10)));

// --- Cipher List Panel ---
cipherListPanel = new JPanel();
cipherListPanel.setLayout(new BoxLayout(cipherListPanel, BoxLayout.Y_AXIS));
cipherListPanel.setBackground(Theme.SIDEBAR_BG);
cipherListPanel.setOpaque(true);
// No direct add here, handled by scroll pane

// Wrap sidebar list in ScrollPane
JScrollPane listScroll = new JScrollPane(cipherListPanel);
listScroll.setBorder(null);
listScroll.getViewport().setBackground(Theme.SIDEBAR_BG);
listScroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);
listScroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
listScroll.getVerticalScrollBar().setUnitIncrement(16);
listScroll.setAlignmentX(Component.LEFT_ALIGNMENT);

// Custom Scrollbar
listScroll.getVerticalScrollBar().setUI(new BasicScrollBarUI() {
    protected void configureScrollBarColors() {
        this.thumbColor = new Color(60, 60, 60);
        this.trackColor = Theme.SIDEBAR_BG;
    }

    @Override
    protected JButton createDecreaseButton(int orientation) {
        return createZeroButton();
    }

    @Override
    protected JButton createIncreaseButton(int orientation) {
        return createZeroButton();
    }

    private JButton createZeroButton() {
        JButton j = new JButton();
        j.setPreferredSize(new Dimension(0, 0));
        return j;
    }
});

sidebarPanel.add(listScroll);

// --- Content Panel ---
contentPanel = new ContentPanel();

// Welcome Panel with Cyber Background
JPanel welcomePanel = createWelcomePanel();
```

```
        // --- Initialize Ciphers (Lazy Loading) ---
        initCiphers();

        add(sidebarPanel, BorderLayout.WEST);
        add(contentPanel, BorderLayout.CENTER);

        // Show welcome initially
        contentPanel.showPanel(welcomePanel);
    }

    private JPanel createWelcomePanel() {
        // Use OverlayLayout to stack text on top of animation
        JPanel wrapper = new JPanel();
        wrapper.setLayout(new OverlayLayout(wrapper));
        wrapper.setBackground(Theme.MAIN_BG);

        // 1. The Content Layer (Text)
        JPanel content = new JPanel();
        content.setLayout(new BoxLayout(content, BoxLayout.Y_AXIS));
        content.setOpaque(false); // Transparent so background shows through
        content.setAlignmentX(0.5f);
        content.setAlignmentY(0.5f);

        content.add(Box.createVerticalGlue());

        JLabel title = new JLabel("Welcome to Crypto Suite");
        title.setFont(new Font("Segoe UI", Font.BOLD, 40));
        title.setForeground(Theme.ACCENT_COLOR);
        title.setAlignmentX(Component.CENTER_ALIGNMENT);
        content.add(title);

        content.add(Box.createRigidArea(new Dimension(0, 20)));

        JLabel sub = new JLabel("Select a cipher from the sidebar to begin.");
        sub.setFont(new Font("Segoe UI", Font.PLAIN, 18));
        sub.setForeground(Theme.TEXT_SECONDARY);
        sub.setAlignmentX(Component.CENTER_ALIGNMENT);
        content.add(sub);

        content.add(Box.createVerticalGlue());

        // 2. The Background Layer (Animation)
        CyberBackground bg = new CyberBackground();
        bg.setAlignmentX(0.5f);
        bg.setAlignmentY(0.5f);

        // Add to wrapper (Order matters for OverlayLayout: First added is on TOP)
        wrapper.add(content);
        wrapper.add(bg);

        return wrapper;
    }
```

```
private void initCiphers() {
    addCipher("001 : A1Z26 CIPHER", A1Z26CipherPanel.class);
    addCipher("002 : ADFGVX CIPHER", ADFGVXCipherPanel.class);
    addCipher("003 : ADFGX CIPHER", ADFGXCipherPanel.class);
    addCipher("004 : AES-CBC", AESCBCPanel.class);
    addCipher("005 : AES-CTR", AESCTRPanel.class);
    addCipher("006 : AES-GCM", AESGCMPanel.class);
    addCipher("007 : AES", AESPanel.class);
    addCipher("008 : ARIA", ARIAPanel.class);
    addCipher("009 : ASCII SHIFT CIPHER", ASCIIShiftCipherPanel.class);
    addCipher("010 : AFFINE CIPHER", AffineCipherPanel.class);
    addCipher("011 : ALBERTI CIPHER", AlbertiCipherPanel.class);
    addCipher("012 : ALHAMBRA CIPHER", AlhambraCipherPanel.class);
    addCipher("013 : ALLIEDSHIFT CIPHER", AlliedShiftCipherPanel.class);
    addCipher("014 : ALPHANUMERICSUBSTITUTION", AlphanumericSubstitutionPanel.class);
    addCipher("015 : AMERICAN CIPHER DISK", AmericanCipherDiskPanel.class);
    addCipher("016 : ATBASH CIPHER", AtbashCipherPanel.class);
    addCipher("017 : AUTOKEY CIPHER", AutokeyCipherPanel.class);
    addCipher("018 : AUTOMATIC SUBSTITUTION", AutomaticSubstitutionPanel.class);
    addCipher("019 : BEECH CIPHER", BEECHCipherPanel.class);
    addCipher("020 : BACONIAN CIPHER", BaconianCipherPanel.class);
    addCipher("021 : BAZERIES CIPHER", BazeriesCipherPanel.class);
    addCipher("022 : BEALE CIPHER", BealeCipherPanel.class);
    addCipher("023 : BEAUFORT CIPHER", BeaufortCipherPanel.class);
    addCipher("024 : BEAUFORTVARIANT CIPHER", BeaufortVariantCipherPanel.class);
    addCipher("025 : BELLASO CIPHER", BellasoCipherPanel.class);
    addCipher("026 : BIFID CIPHER", BifidCipherPanel.class);
    addCipher("027 : BINARY CIPHER", BinaryCipherPanel.class);
    addCipher("028 : BLOWFISH", BlowfishPanel.class);
    addCipher("029 : BOOK CIPHER", BookCipherPanel.class);
    addCipher("030 : BRUCE CODE", BruceCodePanel.class);
    addCipher("031 : BURROWS-WHEELER", BurrowsWheelerPanel.class);
    addCipher("032 : BURROWS-WHEELER CIPHER", BurrowsWheelerCipherPanel.class);
    addCipher("033 : CADENUS CIPHER", CadenusCipherPanel.class);
    addCipher("034 : CAESAR CIPHER", CaesarCipherPanel.class);
    addCipher("035 : CALYPSO CIPHER", CalypsoCipherPanel.class);
    addCipher("036 : CAMELLIA", CamelliaPanel.class);
    addCipher("037 : CARDAN GRILLE", CardanGrillePanel.class);
    addCipher("038 : CAST-128", Cast128Panel.class);
    addCipher("039 : CAST-256", Cast256Panel.class);
    addCipher("040 : CHACHA20", ChaCha20Panel.class);
    addCipher("041 : CHACHA CIPHER", ChaChaCipherPanel.class);
    addCipher("042 : CHAO CIPHER", ChaoCipherPanel.class);
    addCipher("043 : CHINESE REMAINDER CIPHER", ChineseRemainderCipherPanel.class);
    addCipher("044 : CLEFIA", ClefiaPanel.class);
    addCipher("045 : COLUMNAR TRANSPOSITION", ColumnarTranspositionPanel.class);
    addCipher("046 : COMBINED CIPHER DISK", CombinedCipherDiskPanel.class);
    addCipher("047 : CONJUGATED CIPHER", ConjugatedCipherPanel.class);
    addCipher("048 : COPTIC CIPHER", CopticCipherPanel.class);
    addCipher("049 : CRAMER SHOUP", CramerShoupPanel.class);
    addCipher("050 : CRYPTOGRAM SUBSTITUTION", CryptogramSubstitutionPanel.class);
    addCipher("051 : DANCING MEN CIPHER", DancingMenCipherPanel.class);
    addCipher("052 : DES", DESPanel.class);
    addCipher("053 : DOUBLE COLUMNAR TRANSPOSITION", DoubleColumnarTranspositionPanel.class);
```

```
addCipher("054 : DOUBLE DES", DoubleDESPanel.class);
addCipher("055 : DOUBLE PLAYFAIR", DoublePlayfairPanel.class);
addCipher("056 : DRACONIANSHIFT", DraconianShiftPanel.class);
addCipher("057 : E0 BLUETOOTH", E0BluetoothPanel.class);
addCipher("058 : ECB MODE", ECBModePanel.class);
addCipher("059 : ECC", ECCPanel.class);
addCipher("060 : EDON-80", Edon80Panel.class);
addCipher("061 : ELGAMAL", ElGamalPanel.class);
addCipher("062 : ELLIPTIC CURVE CIPHER", EllipticCurveCipherECCPanel.class);
addCipher("063 : ENIGMA MACHINE", EnigmaMachinePanel.class);
addCipher("064 : EXPONENTIAL CIPHER", ExponentialCipherPanel.class);
addCipher("065 : EXTENDED VIGENERE", ExtendedVigenerePanel.class);
addCipher("066 : FIALKA CIPHER", FialkaCipherPanel.class);
addCipher("067 : FOUR SQUARE CIPHER", FourSquareCipherPanel.class);
addCipher("068 : FRACTIONATED MORSE", FractionatedMorsePanel.class);
addCipher("069 : FREEMASON CIPHER", FreemasonCipherPanel.class);
addCipher("070 : FREQUENCY CIPHER", FrequencyCipherPanel.class);
addCipher("071 : FRINGE CIPHER", FringeCipherPanel.class);
addCipher("072 : GADERYPOLUKI", GaderypolukiPanel.class);
addCipher("073 : GALOIS COUNTER MODE", GaloisCounterModePanel.class);
addCipher("074 : GEE CIPHER", GEECipherPanel.class);
addCipher("075 : GIFT", GIFTPanel.class);
addCipher("076 : GOST", GOSTPanel.class);
addCipher("077 : GRAIN", GrainPanel.class);
addCipher("078 : GRILLE CIPHER", GrilleCipherPanel.class);
addCipher("079 : GROMARK CIPHER", GromarkCipherPanel.class);
addCipher("080 : GRONSFELD CIPHER", GronsfeldCipherPanel.class);
addCipher("081 : HAGELIN M209", HagelinM209Panel.class);
addCipher("082 : HASH BASED CIPHER", HashBasedCipherPanel.class);
addCipher("083 : HASHING", HashingPanel.class);
addCipher("084 : HEBERN ROTOR MACHINE", HebernRotorMachinePanel.class);
addCipher("085 : HIGHT", HightPanel.class);
addCipher("086 : HILL CIPHER", HillCipherPanel.class);
addCipher("087 : HMAC BASED SYSTEMS", HMACBasedSystemsPanel.class);
addCipher("088 : HOMOPHONIC SUBSTITUTION", HomophonicSubstitutionPanel.class);
addCipher("089 : HORTON CIPHER", HortonCipherPanel.class);
addCipher("090 : HUFFMAN CIPHER", HuffmanCipherPanel.class);
addCipher("091 : IDEAL CIPHER", IDEALCipherPanel.class);
addCipher("092 : IDEA", IDEAPanel.class);
addCipher("093 : INDIRECT SUBSTITUTION", IndirectSubstitutionPanel.class);
addCipher("094 : INVERSIVE CIPHER", InversiveCipherPanel.class);
addCipher("095 : ISO STREAM CIPHERS", ISOStreamCiphersPanel.class);
addCipher("096 : J CIPHER", JCipherPanel.class);
addCipher("097 : JEFFERSON CYLINDER", JeffersonCylinderPanel.class);
addCipher("098 : JIGSAW CIPHER", JigsawCipherPanel.class);
addCipher("099 : JN-25", JN25Panel.class);
addCipher("100 : KAHN CIPHER", KahnCipherPanel.class);
addCipher("101 : KASISKI SYSTEM", KasiskiSystemPanel.class);
addCipher("102 : KASUMI", KasumiPanel.class);
addCipher("103 : KEYED CAESAR", KeyedCaesarPanel.class);
addCipher("104 : KEYWORD CIPHER", KeywordCipherPanel.class);
addCipher("105 : KLEIN", KleinPanel.class);
addCipher("106 : KLINGON CIPHER", KlingonCipherPanel.class);
addCipher("107 : KNAPSACK CIPHER", KnapsackCipherPanel.class);
```

```
addCipher("108 : KYBER", KyberPanel.class);
addCipher("109 : LATIN SQUARE CIPHER", LatinSquareCipherPanel.class);
addCipher("110 : LAVARAND CIPHER", LavarandCipherPanel.class);
addCipher("111 : LBLOCK", LBlockPanel.class);
addCipher("112 : LED", LEDPanel.class);
addCipher("113 : LFSR CIPHERS", LFSRCiphersPanel.class);
addCipher("114 : LINEAR SUBSTITUTION", LinearSubstitutionPanel.class);
addCipher("115 : LOGICAL XOR CIPHER", LogicalXORCipherPanel.class);
addCipher("116 : LORENZ SZ40", LorenzSZ40Panel.class);
addCipher("117 : M-138 CIPHER", M138CipherPanel.class);
addCipher("118 : M-209", M209Panel.class);
addCipher("119 : M-94 CIPHER", M94CipherPanel.class);
addCipher("120 : MAC BACON CIPHER", MACBaconCipherPanel.class);
addCipher("121 : MALBOLGE ENCRYPTION", MalbolgeEncryptionPanel.class);
addCipher("122 : MARS", MarsPanel.class);
addCipher("123 : MATRIX CIPHER", MatrixCipherPanel.class);
addCipher("124 : MCELIECE", McEliecePanel.class);
addCipher("125 : MERKLE HELLMAN", MerkleHellmanPanel.class);
addCipher("126 : MISTY1", Misty1Panel.class);
addCipher("127 : MODULAR CIPHER", ModularCipherPanel.class);
addCipher("128 : MONOALPHABETIC SUBSTITUTION", MonoalphabeticSubstitutionPanel.class);
addCipher("129 : MORSE CODE", MorseCodePanel.class);
addCipher("130 : MYSZKOWSKI TRANSPOSITION", MyszkowskiTranspositionPanel.class);
addCipher("131 : NATO CODE", NATOCodePanel.class);
addCipher("132 : NIHILIST CIPHER", NihilistCipherPanel.class);
addCipher("133 : NTRU", NTRUPanel.class);
addCipher("134 : DOUBLE TRANSPOSITION", DoubleTranspositionPanel.class);
addCipher("135 : HMAC", HMACPanel.class);
addCipher("136 : HEBERN ROTOR", HebernRotorPanel.class);
addCipher("137 : ISO STREAM", ISOStreamPanel.class);
addCipher("138 : NOEKEON", NoekeonPanel.class);
addCipher("139 : NULL CIPHER", NullCipherPanel.class);
addCipher("140 : NUMBER STATION CIPHERS", NumberStationCiphersPanel.class);
addCipher("141 : OTAR CIPHER", OTARCipherPanel.class);
addCipher("142 : OBFUSCATED TRANSPOSITION", ObfuscatedTranspositionPanel.class);
addCipher("143 : OCTAL CIPHER", OctalCipherPanel.class);
addCipher("144 : ONE TIME PAD", OneTimePadPanel.class);
addCipher("145 : OUT OF BAND CIPHER", OutofBandCipherPanel.class);
addCipher("146 : PBKDF2 BASED SYSTEMS", PBKDF2BasedSystemsPanel.class);
addCipher("147 : PSK CIPHER", PSKCipherPanel.class);
addCipher("148 : PADDING ORACLE CIPHER", PaddingOracleCipherPanel.class);
addCipher("149 : PANAMA", PanamaPanel.class);
addCipher("150 : PASIGRAPHY CIPHER", PasigraphyCipherPanel.class);
addCipher("151 : PERMUTATION CIPHER", PermutationCipherPanel.class);
addCipher("152 : PIGPEN CIPHER", PigpenCipherPanel.class);
addCipher("153 : PLAYFAIR CIPHER", PlayfairCipherPanel.class);
addCipher("154 : POLYBIUS SQUARE", PolybiusSquarePanel.class);
addCipher("155 : PORTA CIPHER", PortaCipherPanel.class);
addCipher("156 : PRESENT", PresentPanel.class);
addCipher("157 : PRIME NUMBER CIPHER", PrimeNumberCipherPanel.class);
addCipher("158 : PRINCE", PrincePanel.class);
addCipher("159 : PUFFERFISH CIPHER", PufferfishCipherPanel.class);
addCipher("160 : QUADRATIC CIPHER", QuadraticCipherPanel.class);
addCipher("161 : QUAGMIRE II-IV", QuagmireIIVPanel.class);
```

```
    addCipher("162 : QUANTUM CIPHERS", QuantumCiphersPanel.class);
    addCipher("163 : RABBIT", RabbitPanel.class);
    addCipher("164 : RAIL FENCE CIPHER", RailFenceCipherPanel.class);
    addCipher("165 : RANDOM SUBSTITUTION", RandomSubstitutionPanel.class);
    addCipher("166 : RC2", RC2Panel.class);
    addCipher("167 : RC4", RC4Panel.class);
    addCipher("168 : RC5", RC5Panel.class);
    addCipher("169 : RC6", RC6Panel.class);
    addCipher("170 : RECTANGLE", RectanglePanel.class);
    addCipher("171 : REDEFENSE CIPHER", RedefenseCipherPanel.class);
    addCipher("172 : RIVEST CIPHERS", RivestCiphersPanel.class);
    addCipher("173 : ROT13", ROT13Panel.class);
    addCipher("174 : ROT1/ROT47", ROT1ROT47Panel.class);
    addCipher("175 : RSA", RSAPanel.class);
    addCipher("176 : SHA BASED CIPHERS", SHABasedCiphersPanel.class);
    addCipher("177 : SIGABA", SIGABAPanel.class);
    addCipher("178 : SPN NETWORK", SPNNetworkPanel.class);
    addCipher("179 : SALSA20", Salsa20Panel.class);
    addCipher("180 : SCYTALE CIPHER", ScytaleCipherPanel.class);
    addCipher("181 : SEED", SeedPanel.class);
    addCipher("182 : SERPENT", SerpentPanel.class);
    addCipher("183 : SHIFT CIPHER", ShiftCipherPanel.class);
    addCipher("184 : SIMON", SimonPanel.class);
    addCipher("185 : SIMPLE SUBSTITUTION", SimpleSubstitutionPanel.class);
    addCipher("186 : SKIPJACK", SkipjackPanel.class);
    addCipher("187 : SNOW 3G", Snow3GPanel.class);
    addCipher("188 : SOLITAIRE CIPHER", SolitaireCipherPanel.class);
    addCipher("189 : SON OF PERMUTATION", SonofPermutationPanel.class);
    addCipher("190 : SPARTAN SCYTALE", SpartanScytalePanel.class);
    addCipher("191 : SPECK", SpeckPanel.class);
    addCipher("192 : STRADDLING CHECKERBOARD", StraddlingCheckerboardPanel.class);
    addCipher("193 : STREAM CIPHER", StreamCipherPanel.class);
    addCipher("194 : SWAGMAN CIPHER", SwagmanCipherPanel.class);
    addCipher("195 : TEA", TEAPanel.class);
    addCipher("196 : TABULA RECTA", TabulaRectaPanel.class);
    addCipher("197 : TAP CODE", TapCodePanel.class);
    addCipher("198 : TEMPLAR CIPHER", TemplarCipherPanel.class);
    addCipher("199 : THREE SQUARE CIPHER", ThreeSquareCipherPanel.class);
    addCipher("200 : TRIFID CIPHER", TrifidCipherPanel.class);
    addCipher("201 : TRITHEME CIPHER", TrithemeCipherPanel.class);
    addCipher("202 : TWOFISH", TwofishPanel.class);
    addCipher("203 : TYPEX CIPHER", TypexCipherPanel.class);
    addCipher("204 : ULTRA CIPHER", UltraCipherPanel.class);
    addCipher("205 : VERNAM CIPHER", VernamCipherPanel.class);
    addCipher("206 : VIGENERE CIPHER", VigenereCipherPanel.class);
    addCipher("207 : XOR CIPHER", XORCipherPanel.class);
}

private void addCipher(String name, Class<? extends JPanel> panelClass) {
    cipherMap.put(name, panelClass);

    AnimatedSidebarButton btn = new AnimatedSidebarButton(name);

    btn.addActionListener(e -> {
```

```
            resetSidebarButtons();
            btn.setSelected(true);
            loadCipher(name);
        });

        sidebarButtons.add(btn);
        cipherListPanel.add(btn);
        cipherListPanel.add(Box.createRigidArea(new Dimension(0, 2)));
    }

    private void loadCipher(String name) {
        if (!loadedPanels.containsKey(name)) {
            try {
                // Show loading cursor
                setCursor(Cursor.getPredefinedCursor(Cursor.WAIT_CURSOR));

                Class<? extends JPanel> clazz = cipherMap.get(name);
                if (clazz != null) {
                    JPanel panel = clazz.getDeclaredConstructor().newInstance();
                    loadedPanels.put(name, panel);
                    contentPanel.showPanel(panel);
                }
            } catch (Exception e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this, "Error loading cipher: " + e.getMessage());
            } finally {
                setCursor(Cursor.getDefaultCursor());
            }
        } else {
            contentPanel.showPanel(loadedPanels.get(name));
        }
    }

    private void resetSidebarButtons() {
        for (AnimatedSidebarButton b : sidebarButtons) {
            b.setSelected(false);
        }
    }

    private void filterCiphers() {
        String query = searchField.getText().toLowerCase();
        if (query.equals("search algorithms..."))
            query = "";

        cipherListPanel.removeAll();

        for (AnimatedSidebarButton btn : sidebarButtons) {
            if (btn.getText().toLowerCase().contains(query)) {
                cipherListPanel.add(btn);
                cipherListPanel.add(Box.createRigidArea(new Dimension(0, 2)));
            }
        }
        cipherListPanel.revalidate();
        cipherListPanel.repaint();
```

```
    }

    public static void main(String[] args) {
        // Enable hardware acceleration
        System.setProperty("sun.java2d.opengl", "true");
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception ignored) {
        }

        components.SplashScreen splash = new components.SplashScreen();
        splash.showSplash();

        SwingUtilities.invokeLater(() -> new CryptoApp().setVisible(true));
    }
}
```

File: .\ciphers\A1Z26CipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class A1Z26CipherPanel extends BaseCipherPanel {
    public A1Z26CipherPanel() {
        super("A1Z26 Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        p.add(enc);
        p.add(dec);
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().trim();
            if (!encrypt) { // Decrypt
                String[] parts = text.split("[^0-9]+");
                StringBuilder sb = new StringBuilder();
                for (String p : parts) {
                    if (!p.isEmpty()) {
                        int val = Integer.parseInt(p);
                        if (val >= 1 && val <= 26)
                            sb.append((char) ('A' + val - 1));
                    }
                }
                outputArea.setText(sb.toString());
            } else { // Encrypt
                StringBuilder sb = new StringBuilder();
                for (char c : text.toUpperCase().toCharArray()) {
                    if (Character.isLetter(c))
                        sb.append((int) (c - 'A' + 1)).append("-");
                }
                if (sb.length() > 0)
                    sb.setLength(sb.length() - 1);
                outputArea.setText(sb.toString());
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\ADFGVXCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;
import java.util.*;

public class ADFGVXCipherPanel extends BaseCipherPanel {
    private JTextField polybiusKeyField;
    private JTextField transKeyField;
    private static final char[] ADFGVX = { 'A', 'D', 'F', 'G', 'V', 'X' };
    private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";

    public ADFGVXCipherPanel() {
        super("ADFGVX Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Polybius Key
        JPanel pKeyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        pKeyPanel.setOpaque(false);
        pKeyPanel.add(new JLabel("<html><font color='white'>Polybius Key:</font></html>"));
        polybiusKeyField = createTextField("SECRET");
        pKeyPanel.add(polybiusKeyField);
        container.add(pKeyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: Transposition Key
        JPanel tKeyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        tKeyPanel.setOpaque(false);
        tKeyPanel.add(new JLabel("<html><font color='white'>Transposition Key:</font></html>"));
        transKeyField = createTextField("CRYPTO");
        tKeyPanel.add(transKeyField);
        container.add(tKeyPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
```

```
    btnPanel.add(enc);
    btnPanel.add(dec);

    container.add(btnPanel);

    p.add(container);
}

@Override
protected void resetFields() {
    polybiusKeyField.setText("SECRET");
    transKeyField.setText("CRYPTO");
}

private void process(boolean encrypt) {
    try {
        String input = getInputText();
        String pKey = polybiusKeyField.getText().toUpperCase().replaceAll("[^A-Z0-9]", "");
        String tKey = transKeyField.getText().toUpperCase().replaceAll("[^A-Z]", "");

        if (tKey.isEmpty())
            throw new Exception("Transposition Key cannot be empty");

        char[][] square = generatePolybiusSquare(pKey);

        if (encrypt) {
            String substituted = substitute(input, square);
            String result = transpose(substituted, tKey);
            outputArea.setText(result);
        } else {
            String untransposed = untranspose(input, tKey);
            String result = unsubstitute(untransposed, square);
            outputArea.setText(result);
        }
    } catch (Exception e) {
        outputArea.setText("Error: " + e.getMessage());
    }
}

private char[][] generatePolybiusSquare(String key) {
    String uniqueKey = "";
    for (char c : (key + ALPHABET).toCharArray()) {
        if (uniqueKey.indexOf(c) == -1) {
            uniqueKey += c;
        }
    }

    char[][] square = new char[6][6];
    for (int i = 0; i < 36; i++) {
        square[i / 6][i % 6] = uniqueKey.charAt(i);
    }
    return square;
}
```

```
private String substitute(String text, char[][] square) {
    StringBuilder sb = new StringBuilder();
    text = text.toUpperCase().replaceAll("[^A-Z0-9]", "");

    for (char c : text.toCharArray()) {
        boolean found = false;
        for (int r = 0; r < 6; r++) {
            for (int cIdx = 0; cIdx < 6; cIdx++) {
                if (square[r][cIdx] == c) {
                    sb.append(ADFGVX[r]).append(ADFGVX[cIdx]);
                    found = true;
                    break;
                }
            }
            if (found)
                break;
        }
    }
    return sb.toString();
}

private String unsubstitute(String text, char[][] square) {
    StringBuilder sb = new StringBuilder();
    text = text.replaceAll("[^ADFGVX]", "");

    for (int i = 0; i < text.length(); i += 2) {
        if (i + 1 >= text.length())
            break;
        char rChar = text.charAt(i);
        char cChar = text.charAt(i + 1);

        int r = -1, c = -1;
        for (int k = 0; k < 6; k++) {
            if (ADFGVX[k] == rChar)
                r = k;
            if (ADFGVX[k] == cChar)
                c = k;
        }

        if (r != -1 && c != -1) {
            sb.append(square[r][c]);
        }
    }
    return sb.toString();
}

private String transpose(String text, String key) {
    int cols = key.length();
    int rows = (int) Math.ceil((double) text.length() / cols);

    // Create grid
    char[][] grid = new char[rows][cols];
    int idx = 0;
```

```
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            if (idx < text.length()) {
                grid[r][c] = text.charAt(idx++);
            } else {
                grid[r][c] = ' '; // Padding
            }
        }
    }

    // Sort key to determine column order
    Integer[] colOrder = getColumnOrder(key);

    StringBuilder sb = new StringBuilder();
    for (int c : colOrder) {
        for (int r = 0; r < rows; r++) {
            if (grid[r][c] != ' ') {
                sb.append(grid[r][c]);
            }
        }
        sb.append(" "); // Space between columns for readability
    }
    return sb.toString().trim();
}

private String untranspose(String text, String key) {
    text = text.replaceAll(" ", ""); // Remove spaces
    int cols = key.length();
    int len = text.length();
    int rows = (int) Math.ceil((double) len / cols);
    int shortCols = (rows * cols) - len; // Number of empty cells in last row

    Integer[] colOrder = getColumnOrder(key);
    char[][] grid = new char[rows][cols];

    int currentIdx = 0;
    for (int k = 0; k < cols; k++) {
        int colIdx = colOrder[k];
        int colLen = rows;
        // If this column is one of the last 'shortCols' columns (reading right to left
        // in the grid), it has one less char
        // The empty cells are at the end of the grid, so the last 'shortCols' columns
        // have length rows-1
        // Wait, the padding is at the end.
        // Example: 8 chars, 3 cols. Rows=3.
        // Grid:
        // X X X
        // X X X
        // X X
        // Col 0 len 3, Col 1 len 3, Col 2 len 2.
        // So columns >= (cols - shortCols) have length rows-1? No.
        // The empty cells are at grid[rows-1][cols-shortCols] to grid[rows-1][cols-1].

        if (colIdx >= (cols - shortCols)) {
```

```
                colLen = rows - 1;
            }

            for (int r = 0; r < colLen; r++) {
                if (currentIdx < text.length()) {
                    grid[r][colIdx] = text.charAt(currentIdx++);
                }
            }
        }

        StringBuilder sb = new StringBuilder();
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                if (grid[r][c] != 0) {
                    sb.append(grid[r][c]);
                }
            }
        }
        return sb.toString();
    }

    private Integer[] getColumnOrder(String key) {
        Integer[] indexes = new Integer[key.length()];
        for (int i = 0; i < key.length(); i++)
            indexes[i] = i;

        Arrays.sort(indexes, (a, b) -> Character.compare(key.charAt(a), key.charAt(b)));
        return indexes;
    }
}
```

File: .\ciphers\ADFGXCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class ADFGXCipherPanel extends BaseCipherPanel {
    public ADFGXCipherPanel() {
        super("ADFGX Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\AESCBCPanel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.security.SecureRandom;
import java.util.Base64;

public class AESCBCPanel extends BaseCipherPanel {
    private JTextField keyField;
    private JTextField ivField;

    public AESCBCPanel() {
        super("AES-CBC Mode");
    }

    @Override
    protected void addControls(JPanel p) {
        // Use a vertical container to prevent wrapping issues
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key
        JPanel keyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        keyPanel.setOpaque(false);
        keyPanel.add(new JLabel("<html><font color='white'>Key (Base64):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        keyPanel.add(keyField);
        container.add(keyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: IV
        JPanel ivPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        ivPanel.setOpaque(false);
        ivPanel.add(new JLabel("<html><font color='white'>IV (Base64): </font></html>"));
        ivField = createTextField("");
        ivField.setPreferredSize(new Dimension(200, 30));
        ivPanel.add(ivField);
        container.add(ivPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
```

```
        btnPanel.setOpaque(false);

        JButton gen = new NeonButton("Gen Key/IV", new Color(100, 100, 100));
        gen.addActionListener(e -> generateKeyAndIV());
        btnPanel.add(gen);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        btnPanel.add(enc);

        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        btnPanel.add(dec);

        container.add(btnPanel);

        p.add(container);

        generateKeyAndIV();
    }

    private void generateKeyAndIV() {
        try {
            KeyGenerator kg = KeyGenerator.getInstance("AES");
            kg.init(128);
            SecretKey sk = kg.generateKey();
            keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));

            byte[] iv = new byte[16];
            new SecureRandom().nextBytes(iv);
            ivField.setText(Base64.getEncoder().encodeToString(iv));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void process(boolean encrypt) {
        try {
            byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
            byte[] ivBytes = Base64.getDecoder().decode(ivField.getText());
            SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
            IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

            Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
            cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, ivSpec);

            byte[] input;
            if (encrypt) {
                input = getInputText().getBytes("UTF-8");
                byte[] output = cipher.doFinal(input);
                outputArea.setText(Base64.getEncoder().encodeToString(output));
            } else {
                input = Base64.getDecoder().decode(getInputText());
                byte[] output = cipher.doFinal(input);
```

```
                String result = new String(output, "UTF-8");
                outputArea.setText(result);
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\AESCTRPanel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.security.SecureRandom;
import java.util.Base64;

public class AESCTRPanel extends BaseCipherPanel {
    private JTextField keyField;
    private JTextField ivField;

    public AESCTRPanel() {
        super("AES-CTR Mode");
    }

    @Override
    protected void addControls(JPanel p) {
        // Use a vertical container to prevent wrapping issues
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key
        JPanel keyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        keyPanel.setOpaque(false);
        keyPanel.add(new JLabel("<html><font color='white'>Key (Base64):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        keyPanel.add(keyField);
        container.add(keyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: IV
        JPanel ivPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        ivPanel.setOpaque(false);
        ivPanel.add(new JLabel("<html><font color='white'>IV (Base64): </font></html>"));
        ivField = createTextField("");
        ivField.setPreferredSize(new Dimension(200, 30));
        ivPanel.add(ivField);
        container.add(ivPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
```

```
        btnPanel.setOpaque(false);

        JButton gen = new NeonButton("Gen Key/IV", new Color(100, 100, 100));
        gen.addActionListener(e -> generateKeyAndIV());
        btnPanel.add(gen);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        btnPanel.add(enc);

        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        btnPanel.add(dec);

        container.add(btnPanel);

        p.add(container);

        generateKeyAndIV();
    }

    private void generateKeyAndIV() {
        try {
            KeyGenerator kg = KeyGenerator.getInstance("AES");
            kg.init(128);
            SecretKey sk = kg.generateKey();
            keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));

            byte[] iv = new byte[16];
            new SecureRandom().nextBytes(iv);
            ivField.setText(Base64.getEncoder().encodeToString(iv));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void process(boolean encrypt) {
        try {
            byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
            byte[] ivBytes = Base64.getDecoder().decode(ivField.getText());
            SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
            IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

            Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding");
            cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, ivSpec);

            byte[] input;
            if (encrypt) {
                input = getInputText().getBytes("UTF-8");
                byte[] output = cipher.doFinal(input);
                outputArea.setText(Base64.getEncoder().encodeToString(output));
            } else {
                input = Base64.getDecoder().decode(getInputText());
                byte[] output = cipher.doFinal(input);
```

```
                String result = new String(output, "UTF-8");
                outputArea.setText(result);
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\AESGCMPanel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.security.SecureRandom;
import java.util.Base64;

public class AESGCMPanel extends BaseCipherPanel {
    private JTextField keyField;
    private JTextField ivField;

    public AESGCMPanel() {
        super("AES-GCM Mode");
    }

    @Override
    protected void addControls(JPanel p) {
        // Use a vertical container to prevent wrapping issues
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key
        JPanel keyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        keyPanel.setOpaque(false);
        keyPanel.add(new JLabel("<html><font color='white'>Key (Base64):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        keyPanel.add(keyField);
        container.add(keyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: IV
        JPanel ivPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        ivPanel.setOpaque(false);
        ivPanel.add(new JLabel("<html><font color='white'>IV (Base64): </font></html>"));
        ivField = createTextField("");
        ivField.setPreferredSize(new Dimension(200, 30));
        ivPanel.add(ivField);
        container.add(ivPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
```

```
        btnPanel.setOpaque(false);

        JButton gen = new NeonButton("Gen Key/IV", new Color(100, 100, 100));
        gen.addActionListener(e -> generateKeyAndIV());
        btnPanel.add(gen);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        btnPanel.add(enc);

        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        btnPanel.add(dec);

        container.add(btnPanel);

        p.add(container);

        generateKeyAndIV();
    }

    private void generateKeyAndIV() {
        try {
            KeyGenerator kg = KeyGenerator.getInstance("AES");
            kg.init(128);
            SecretKey sk = kg.generateKey();
            keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));

            byte[] iv = new byte[12]; // GCM standard IV length is 12 bytes
            new SecureRandom().nextBytes(iv);
            ivField.setText(Base64.getEncoder().encodeToString(iv));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void process(boolean encrypt) {
        try {
            byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
            byte[] ivBytes = Base64.getDecoder().decode(ivField.getText());
            SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "AES");
            GCMParameterSpec gcmSpec = new GCMParameterSpec(128, ivBytes); // 128 bit tag length

            Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
            cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, gcmSpec);

            byte[] input;
            if (encrypt) {
                input = getInputText().getBytes("UTF-8");
                byte[] output = cipher.doFinal(input);
                outputArea.setText(Base64.getEncoder().encodeToString(output));
            } else {
                input = Base64.getDecoder().decode(getInputText());
                byte[] output = cipher.doFinal(input);
```

```
                String result = new String(output, "UTF-8");
                outputArea.setText(result);
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\AESPanel.java

```
package ciphers;
import components.*;

import javax.crypto.Cipher;
import javax.swing.*;
import java.awt.*;
import java.util.Base64;

public class AESPanel extends BaseCipherPanel {
    javax.crypto.SecretKey key;

    public AESPanel() {
        super("AES (Rijndael)");
        try {
            javax.crypto.KeyGenerator kg = javax.crypto.KeyGenerator.getInstance("AES");
            kg.init(128);
            key = kg.generateKey();
        } catch (Exception e) {
        }
    }

    @Override
    protected void addControls(JPanel p) {
        JButton gen = new NeonButton("New Key", new Color(100, 100, 100));
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        gen.addActionListener(e -> {
            try {
                javax.crypto.KeyGenerator kg = javax.crypto.KeyGenerator.getInstance("AES");
                kg.init(128);
                key = kg.generateKey();
                outputArea.setText("New AES Key Generated!");
            } catch (Exception ex) {
            }
        });
        enc.addActionListener(e -> measureAndRun(() -> {
            try {
                Cipher c = Cipher.getInstance("AES");
                c.init(Cipher.ENCRYPT_MODE, key);
                byte[] b = c.doFinal(getInputText().getBytes());
                outputArea.setText(Base64.getEncoder().encodeToString(b));
            } catch (Exception ex) {
                throw new RuntimeException(ex.getMessage());
            }
        }, true));
        dec.addActionListener(e -> measureAndRun(() -> {
            try {
                Cipher c = Cipher.getInstance("AES");
                c.init(Cipher.DECRYPT_MODE, key);
                byte[] b = c.doFinal(Base64.getDecoder().decode(getInputText()));
                outputArea.setText(new String(b));
            } catch (Exception ex) {
```

```
                throw new RuntimeException(ex.getMessage());
            }
        }, false));
        p.add(gen);
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\AffineCipherPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.awt.*;

public class AffineCipherPanel extends BaseCipherPanel {
    JTextField aField, bField;

    public AffineCipherPanel() {
        super("Affine Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Group inputs
        JPanel inputRow = new JPanel(new FlowLayout(FlowLayout.LEFT, 15, 0));
        inputRow.setOpaque(false);

        JPanel aPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        aPanel.setOpaque(false);
        aPanel.add(new JLabel("<html><font color='white'>a:</font></html>"));
        aField = createTextField("5");
        aPanel.add(aField);
        inputRow.add(aPanel);

        JPanel bPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        bPanel.setOpaque(false);
        bPanel.add(new JLabel("<html><font color='white'>b:</font></html>"));
        bField = createTextField("8");
        bPanel.add(bField);
        inputRow.add(bPanel);

        container.add(inputRow);

        container.add(Box.createVerticalStrut(10));

        // Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        btnPanel.add(enc);
        btnPanel.add(dec);
```

```
        container.add(btnPanel);

        p.add(container);
    }

    @Override
    protected void resetFields() {
        aField.setText("5");
        bField.setText("8");
    }

    private void process(boolean encrypt) {
        try {
            int a = Integer.parseInt(aField.getText());
            int b = Integer.parseInt(bField.getText());
            String text = getInputText().toUpperCase();
            StringBuilder sb = new StringBuilder();
            int aInv = 0;
            if (!encrypt) {
                for (int i = 0; i < 26; i++)
                    if ((a * i) % 26 == 1)
                        aInv = i;
                if (aInv == 0)
                    throw new Exception("'a' must be coprime to 26");
            }
            for (char c : text.toCharArray()) {
                if (Character.isLetter(c)) {
                    int x = c - 'A';
                    if (encrypt)
                        sb.append((char) (((a * x + b) % 26) + 'A'));
                    else {
                        int val = (aInv * (x - b)) % 26;
                        if (val < 0)
                            val += 26;
                        sb.append((char) (val + 'A'));
                    }
                } else
                    sb.append(c);
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\AlbertiCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class AlbertiCipherPanel extends BaseCipherPanel {
    public AlbertiCipherPanel() {
        super("Alberti Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\AlhambraCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class AlhambraCipherPanel extends BaseCipherPanel {
    public AlhambraCipherPanel() {
        super("Alhambra Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\AlliedShiftCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class AlliedShiftCipherPanel extends BaseCipherPanel {
    public AlliedShiftCipherPanel() {
        super("Allied");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(
                e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(
                e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\AlphanumericSubstitutionPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.awt.*;

public class AlphanumericSubstitutionPanel extends BaseCipherPanel {
    JTextField shiftField;

    public AlphanumericSubstitutionPanel() {
        super("Caesar Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Shift:</font></html>"));
        shiftField = createTextField("3");
        p.add(shiftField);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        shiftField.setText("3");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText();
            int shift = Integer.parseInt(shiftField.getText());
            StringBuilder res = new StringBuilder();
            for (char c : text.toCharArray()) {
                if (Character.isLetter(c)) {
                    char base = Character.isUpperCase(c) ? 'A' : 'a';
                    int val = c - base;
                    int s = encrypt ? shift : -shift;
                    val = (val + s) % 26;
                    if (val < 0)
                        val += 26;
                    res.append((char) (base + val));
                } else
                    res.append(c);
            }
            outputArea.setText(res.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
```

```
        }
    }
}
```

File: .\ciphers\AmericanCipherDiskPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class AmericanCipherDiskPanel extends BaseCipherPanel {
    public AmericanCipherDiskPanel() {
        super("American CipherDisk (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ARIAPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ARIAPanel extends BaseCipherPanel {
    public ARIAPanel() {
        super("ARIA (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ASCIIShiftCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class ASCIIShiftCipherPanel extends BaseCipherPanel {
    JTextField shiftField;

    public ASCIIShiftCipherPanel() {
        super("ASCII Shift Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Shift:</font></html>"));
        shiftField = createTextField("1");
        p.add(shiftField);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        shiftField.setText("1");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText();
            int shift = Integer.parseInt(shiftField.getText());
            StringBuilder sb = new StringBuilder();
            for (char c : text.toCharArray()) {
                sb.append((char) (c + (encrypt ? shift : -shift)));
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\AtbashCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class AtbashCipherPanel extends BaseCipherPanel {
    public AtbashCipherPanel() {
        super("Atbash Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(this::process, true));
        dec.addActionListener(e -> measureAndRun(this::process, false));

        p.add(enc);
        p.add(dec);
    }

    private void process() {
        try {
            String text = getInputText().toUpperCase();
            StringBuilder sb = new StringBuilder();
            for (char c : text.toCharArray()) {
                if (Character.isLetter(c))
                    sb.append((char) ('Z' - (c - 'A')));
                else
                    sb.append(c);
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\AutokeyCipherPanel.java

```
package ciphers;
import components.*;

import javax.swing.*;
import java.awt.*;

public class AutokeyCipherPanel extends BaseCipherPanel {
    JTextField keyField;

    public AutokeyCipherPanel() {
        super("Autokey Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key:</font></html>"));
        keyField = createTextField("KEY");
        p.add(keyField);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        keyField.setText("KEY");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase().replaceAll("[^A-Z]", "");
            String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "");
            StringBuilder sb = new StringBuilder();
            if (encrypt) {
                String currentKey = key + text;
                for (int i = 0; i < text.length(); i++) {
                    int p = text.charAt(i) - 'A';
                    int k = currentKey.charAt(i) - 'A';
                    sb.append((char) (((p + k) % 26) + 'A'));
                }
            } else {
                StringBuilder currentKey = new StringBuilder(key);
                for (int i = 0; i < text.length(); i++) {
                    int c = text.charAt(i) - 'A';
                    int k = currentKey.charAt(i) - 'A';
                    int p = (c - k + 26) % 26;
                    char pChar = (char) (p + 'A');
                    sb.append(pChar);
                    currentKey.append(pChar);
```

```
                }
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\AutomaticSubstitutionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class AutomaticSubstitutionPanel extends BaseCipherPanel {
    public AutomaticSubstitutionPanel() {
        super("AutomaticSubstitution (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BaconianCipherPanel.java

```
package ciphers;
import components.*;

import javax.swing.*;
import java.awt.*;

public class BaconianCipherPanel extends BaseCipherPanel {
    public BaconianCipherPanel() {
        super("Baconian Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase();
            String[] codes = { "AAAAA", "AAAAB", "AAABA", "AAABB", "AABAA", "AABAB", "AABBA", "AABBB", "ABAAA",
                    "ABAAB", "ABABA", "ABABB", "ABBAA", "ABBAB", "ABBBA", "ABBBB", "BAAAA", "BAAAB", "BAABA",
                    "BAABB", "BABAA", "BABAB", "BABBA", "BABBB", "BBAAA", "BBAAB" };
            if (encrypt) {
                StringBuilder sb = new StringBuilder();
                for (char c : text.toCharArray()) {
                    if (c >= 'A' && c <= 'Z')
                        sb.append(codes[c - 'A']).append(" ");
                }
                outputArea.setText(sb.toString());
            } else {
                StringBuilder sb = new StringBuilder();
                String[] parts = text.split("\\s+");
                for (String p : parts) {
                    for (int i = 0; i < codes.length; i++) {
                        if (codes[i].equals(p))
                            sb.append((char) ('A' + i));
                    }
                }
                outputArea.setText(sb.toString());
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\BaseCipherPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.LineBorder;
import java.awt.*;
import java.awt.event.ActionListener;
import java.io.File;
import java.nio.file.Files;
import java.text.SimpleDateFormat;
import java.util.Date;

public abstract class BaseCipherPanel extends JPanel {
    protected JTextArea inputArea;
    protected JTextArea outputArea;
    protected PerformanceGraph graph;
    protected JTextArea statsArea;
    protected SimpleDateFormat timeFormat = new SimpleDateFormat("HH:mm:ss.SSS");

    protected JRadioButton textModeBtn;
    protected JRadioButton fileModeBtn;
    protected JPanel inputCardPanel;
    protected CardLayout inputCardLayout;
    protected JTextField filePathField;
    protected File selectedFile;
    protected String cipherTitle;

    public BaseCipherPanel(String title) {
        super();
        this.cipherTitle = title;
        setLayout(new BorderLayout());
        setOpaque(true);
        setBackground(Theme.MAIN_BG);
        setBorder(new EmptyBorder(30, 40, 30, 40));

        // Header
        JLabel titleLabel = new JLabel(title);
        titleLabel.setFont(Theme.FONT_TITLE);
        titleLabel.setForeground(Theme.TEXT_PRIMARY);
        add(titleLabel, BorderLayout.NORTH);

        // Main Grid
        java.awt.GridLayout gl = new java.awt.GridLayout(1, 2, 30, 0);
        JPanel mainGrid = new JPanel(gl);
        mainGrid.setOpaque(true);
        mainGrid.setBackground(Theme.MAIN_BG);
        mainGrid.setBorder(new EmptyBorder(25, 0, 0, 0));

        // --- Left Column ---
        JPanel leftCol = new JPanel();
```

```
leftCol.setLayout(new BoxLayout(leftCol, BoxLayout.Y_AXIS));
leftCol.setOpaque(true);
leftCol.setBackground(Theme.MAIN_BG);

// Input Card
CardPanel inputPanel = new CardPanel();
inputPanel.setLayout(new BorderLayout(15, 15));
inputPanel.setBorder(new EmptyBorder(20, 20, 20, 20));

JPanel modePanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 15, 0));
modePanel.setOpaque(false);
textModeBtn = new JRadioButton("Text");
fileModeBtn = new JRadioButton("File");
styleRadioButton(textModeBtn);
styleRadioButton(fileModeBtn);
textModeBtn.setSelected(true);
ButtonGroup bg = new ButtonGroup();
bg.add(textModeBtn);
bg.add(fileModeBtn);
JLabel modeLbl = new JLabel("INPUT SOURCE");
modeLbl.setFont(new Font("Segoe UI", Font.BOLD, 12));
modeLbl.setForeground(Theme.ACCENT_COLOR);
modePanel.add(modeLbl);
modePanel.add(textModeBtn);
modePanel.add(fileModeBtn);
inputPanel.add(modePanel, BorderLayout.NORTH);

inputCardLayout = new CardLayout();
inputCardPanel = new JPanel(inputCardLayout);
inputCardPanel.setOpaque(false);

inputArea = createTextArea(true);
inputCardPanel.add(new DarkScrollPane(inputArea), "TEXT");

JPanel filePanel = new JPanel(new BorderLayout(10, 0));
filePanel.setOpaque(false);
filePathField = createTextField("No file selected");
filePathField.setEditable(false);
JButton browseBtn = new NeonButton("Browse", new Color(60, 60, 80));
browseBtn.addActionListener(e -> chooseFile());
filePanel.add(filePathField, BorderLayout.CENTER);
filePanel.add(browseBtn, BorderLayout.EAST);

JPanel fileWrapper = new JPanel(new GridBagLayout());
fileWrapper.setOpaque(false);
GridBagConstraints gbc = new GridBagConstraints();
gbc.fill = GridBagConstraints.HORIZONTAL;
gbc.weightx = 1.0;
fileWrapper.add(filePanel, gbc);
inputCardPanel.add(fileWrapper, "FILE");

inputPanel.add(inputCardPanel, BorderLayout.CENTER);

JPanel controlsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 10, 0));
```

```
controlsPanel.setOpaque(false);
addControls(controlsPanel);
JButton resetBtn = new NeonButton("Reset", new Color(220, 60, 60)); // Red Reset Button
resetBtn.addActionListener(e -> reset());
controlsPanel.add(resetBtn);
inputPanel.add(controlsPanel, BorderLayout.SOUTH);

leftCol.add(inputPanel);
leftCol.add(Box.createVerticalStrut(25));

// Output Card
CardPanel outputPanel = new CardPanel();
outputPanel.setLayout(new BorderLayout(15, 15));
outputPanel.setBorder(new EmptyBorder(20, 20, 20, 20));

JLabel outLbl = new JLabel("OUTPUT RESULT");
outLbl.setFont(new Font("Segoe UI", Font.BOLD, 12));
outLbl.setForeground(Theme.ACCENT_COLOR);
outputPanel.add(outLbl, BorderLayout.NORTH);

outputArea = createTextArea(false);
outputArea.setForeground(new Color(100, 255, 150));
outputPanel.add(new DarkScrollPane(outputArea), BorderLayout.CENTER);

leftCol.add(outputPanel);

// --- Right Column ---
JPanel rightCol = new JPanel();
rightCol.setLayout(new BoxLayout(rightCol, BoxLayout.Y_AXIS));
rightCol.setOpaque(true);
rightCol.setBackground(Theme.MAIN_BG);

// Stats Card
CardPanel statsPanel = new CardPanel();
statsPanel.setLayout(new BorderLayout(15, 15));
statsPanel.setBorder(new EmptyBorder(20, 20, 20, 20));
statsPanel.setMaximumSize(new Dimension(Integer.MAX_VALUE, 220));

JLabel statsLbl = new JLabel("PERFORMANCE METRICS");
statsLbl.setFont(new Font("Segoe UI", Font.BOLD, 12));
statsLbl.setForeground(Theme.ACCENT_COLOR);
statsPanel.add(statsLbl, BorderLayout.NORTH);

statsArea = new JTextArea();
statsArea.setFont(Theme.FONT_MONO);
statsArea.setBackground(Theme.INPUT_BG);
statsArea.setForeground(Theme.TEXT_PRIMARY);
statsArea.setEditable(false);
statsArea.setOpaque(true); // Explicitly Opaque
statsArea.setBorder(new EmptyBorder(10, 10, 10, 10));
statsPanel.add(statsArea, BorderLayout.CENTER);

rightCol.add(statsPanel);
rightCol.add(Box.createVerticalStrut(25));
```

```
    // Graph Card
    CardPanel graphPanel = new CardPanel();
    graphPanel.setLayout(new BorderLayout(15, 15));
    graphPanel.setBorder(new EmptyBorder(20, 20, 20, 20));

    JLabel graphLbl = new JLabel("TIME ANALYSIS");
    graphLbl.setFont(new Font("Segoe UI", Font.BOLD, 12));
    graphLbl.setForeground(Theme.ACCENT_COLOR);
    graphPanel.add(graphLbl, BorderLayout.NORTH);

    graph = new PerformanceGraph();
    graphPanel.add(graph, BorderLayout.CENTER);

    rightCol.add(graphPanel);

    mainGrid.add(leftCol);
    mainGrid.add(rightCol);

    add(mainGrid, BorderLayout.CENTER);

    ActionListener modeListener = e -> {
        if (textModeBtn.isSelected())
            inputCardLayout.show(inputCardPanel, "TEXT");
        else
            inputCardLayout.show(inputCardPanel, "FILE");
    };
    textModeBtn.addActionListener(modeListener);
    fileModeBtn.addActionListener(modeListener);
}

private void styleRadioButton(JRadioButton rb) {
    rb.setOpaque(false);
    rb.setForeground(Theme.TEXT_PRIMARY);
    rb.setFont(Theme.FONT_NORMAL);
    rb.setFocusPainted(false);
}

private void chooseFile() {
    JFileChooser fc = new JFileChooser();
    int res = fc.showOpenDialog(this);
    if (res == JFileChooser.APPROVE_OPTION) {
        selectedFile = fc.getSelectedFile();
        filePathField.setText(selectedFile.getName());
    }
}

private void reset() {
    inputArea.setText("");
    outputArea.setText("");
    statsArea.setText("");
    filePathField.setText("No file selected");
    selectedFile = null;
    graph.reset();
```

```
    resetFields();
}

protected void resetFields() {
} // To be overridden

protected String getInputText() throws Exception {
    if (textModeBtn.isSelected()) {
        return inputArea.getText();
    } else {
        if (selectedFile == null || !selectedFile.exists()) {
            throw new Exception("No file selected");
        }
        return new String(Files.readAllBytes(selectedFile.toPath()));
    }
}

protected void measureAndRun(Runnable task, boolean isEncrypt) {
    long startNano = System.nanoTime();
    long startTime = System.currentTimeMillis();

    try {
        task.run();
    } catch (Exception e) {
        outputArea.setText("Error: " + e.getMessage());
        return;
    }

    long endNano = System.nanoTime();
    long endTime = System.currentTimeMillis();
    long duration = endNano - startNano;

    StringBuilder stats = new StringBuilder();
    stats.append(" Operation  : ").append(isEncrypt ? "Encryption" : "Decryption").append("\n");
    stats.append(" Start Time : ").append(timeFormat.format(new Date(startTime))).append("\n");
    stats.append(" End Time   : ").append(timeFormat.format(new Date(endTime))).append("\n");
    stats.append(" Total Time : ").append(String.format("%.3f", duration / 1000000.0)).append(" ms");

    if (fileModeBtn.isSelected() && selectedFile != null) {
        try {
            String content = outputArea.getText();
            String name = selectedFile.getName();
            int dotIndex = name.lastIndexOf('.');
            String baseName = (dotIndex == -1) ? name : name.substring(0, dotIndex);

            String suffix = isEncrypt ? "_ENC" : "_DEC";
            // Get cipher name from title, remove spaces and special chars for filename
            String cipherTag = cipherTitle.replaceAll("[^a-zA-Z0-9]", "").toUpperCase();

            String newName = baseName + "_" + cipherTag + suffix + ".txt";

            // Save to ENCRYPTION folder
            File encryptionDir = new File(System.getProperty("user.home"),
                    "Documents/CRYPTOGRAPHY UI DESGIN/ENCRYPTION");
```

```
            if (!encryptionDir.exists()) {
                encryptionDir.mkdirs();
            }
            File outputFile = new File(encryptionDir, newName);

            Files.write(outputFile.toPath(), content.getBytes());
            stats.append("\n Saved to   : ").append(outputFile.getAbsolutePath());
        } catch (Exception e) {
            stats.append("\n Save Error : ").append(e.getMessage());
        }
    }

    statsArea.setText(stats.toString());

    if (isEncrypt)
        graph.setEncTime(duration);
    else
        graph.setDecTime(duration);
}

protected abstract void addControls(JPanel panel);

protected JTextArea createTextArea(boolean editable) {
    JTextArea area = new JTextArea();
    area.setFont(Theme.FONT_MONO);
    area.setBackground(Theme.INPUT_BG);
    area.setForeground(Theme.TEXT_PRIMARY);
    area.setCaretColor(Theme.ACCENT_COLOR);
    area.setLineWrap(true);
    area.setWrapStyleWord(true);
    area.setEditable(editable);
    area.setOpaque(true); // Explicitly Opaque
    area.setBorder(new EmptyBorder(10, 10, 10, 10));
    return area;
}

protected void runSimulation(boolean encrypt) {
    try {
        String text = getInputText();
        String key = cipherTitle.replaceAll("[^a-zA-Z]", "");
        if (key.isEmpty())
            key = "DEFAULT";

        StringBuilder res = new StringBuilder();
        int keyIndex = 0;

        for (char c : text.toCharArray()) {
            if (c >= 32 && c <= 126) { // Printable ASCII
                int kVal = key.charAt(keyIndex % key.length());
                int shift = kVal % 95;

                int cVal = c - 32;
                int newVal;
```

```
                    if (encrypt) {
                        newVal = (cVal + shift) % 95;
                    } else {
                        newVal = (cVal - shift);
                        if (newVal < 0)
                            newVal += 95;
                    }

                    res.append((char) (32 + newVal));
                } else {
                    res.append(c);
                }
                keyIndex++;
            }
            outputArea.setText(res.toString());

        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }

    protected JTextField createTextField(String text) {
        JTextField field = new JTextField(text, 10);
        field.setFont(Theme.FONT_MONO);
        field.setBackground(Theme.INPUT_BG);
        field.setForeground(Theme.TEXT_PRIMARY);
        field.setCaretColor(Theme.ACCENT_COLOR);
        field.setOpaque(true); // Explicitly Opaque
        field.setBorder(BorderFactory.createCompoundBorder(
                new LineBorder(new Color(60, 60, 60)),
                new EmptyBorder(5, 10, 5, 10)));
        return field;
    }
}
```

File: .\ciphers\BazeriesCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class BazeriesCipherPanel extends BaseCipherPanel {
    public BazeriesCipherPanel() {
        super("Bazeries Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BealeCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class BealeCipherPanel extends BaseCipherPanel {
    public BealeCipherPanel() {
        super("Beale Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BeaufortCipherPanel.java

```
package ciphers;
import components.*;

import javax.swing.*;

public class BeaufortCipherPanel extends BaseCipherPanel {
    JTextField keyField;

    public BeaufortCipherPanel() {
        super("Beaufort Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key:</font></html>"));
        keyField = createTextField("KEY");
        p.add(keyField);
        JButton act = new NeonButton("Encrypt / Decrypt", Theme.ACCENT_COLOR);
        act.addActionListener(e -> measureAndRun(this::process, true));
        p.add(act);
    }

    @Override
    protected void resetFields() {
        keyField.setText("KEY");
    }

    private void process() {
        try {
            String text = getInputText().toUpperCase();
            String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "");
            StringBuilder sb = new StringBuilder();
            for (int i = 0, j = 0; i < text.length(); i++) {
                char c = text.charAt(i);
                if (c < 'A' || c > 'Z') {
                    sb.append(c);
                    continue;
                }
                int m = c - 'A';
                int k = key.charAt(j % key.length()) - 'A';
                int val = (k - m + 26) % 26;
                sb.append((char) (val + 'A'));
                j++;
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\BeaufortVariantCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class BeaufortVariantCipherPanel extends BaseCipherPanel {
    public BeaufortVariantCipherPanel() {
        super("Beaufort Variant Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BEECHCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class BEECHCipherPanel extends BaseCipherPanel {
    public BEECHCipherPanel() {
        super("BEECH Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BellasoCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class BellasoCipherPanel extends BaseCipherPanel {
    public BellasoCipherPanel() {
        super("Bellaso Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BifidCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class BifidCipherPanel extends BaseCipherPanel {
    public BifidCipherPanel() {
        super("Bifid Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BinaryCipherPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.awt.*;

public class BinaryCipherPanel extends BaseCipherPanel {
    public BinaryCipherPanel() {
        super("Binary Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText();
            if (encrypt) {
                StringBuilder sb = new StringBuilder();
                for (char c : text.toCharArray()) {
                    sb.append(String.format("%8s", Integer.toBinaryString(c)).replace(' ', '0')).append(" ");
                }
                outputArea.setText(sb.toString());
            } else {
                String[] parts = text.split("\\s+");
                StringBuilder sb = new StringBuilder();
                for (String p : parts) {
                    sb.append((char) Integer.parseInt(p, 2));
                }
                outputArea.setText(sb.toString());
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\BlowfishPanel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.security.SecureRandom;
import java.util.Base64;

public class BlowfishPanel extends BaseCipherPanel {
    private JTextField keyField;
    private JTextField ivField;

    public BlowfishPanel() {
        super("Blowfish Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key
        JPanel keyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        keyPanel.setOpaque(false);
        keyPanel.add(new JLabel("<html><font color='white'>Key (Base64):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        keyPanel.add(keyField);
        container.add(keyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: IV
        JPanel ivPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        ivPanel.setOpaque(false);
        ivPanel.add(new JLabel("<html><font color='white'>IV (Base64):</font></html>"));
        ivField = createTextField("");
        ivField.setPreferredSize(new Dimension(200, 30));
        ivPanel.add(ivField);
        container.add(ivPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
```

```
    JButton gen = new NeonButton("Gen Key/IV", new Color(100, 100, 100));
    gen.addActionListener(e -> generateKeyAndIV());
    btnPanel.add(gen);

    JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
    enc.addActionListener(e -> measureAndRun(() -> process(true), true));
    btnPanel.add(enc);

    JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
    dec.addActionListener(e -> measureAndRun(() -> process(false), false));
    btnPanel.add(dec);

    container.add(btnPanel);

    p.add(container);

    generateKeyAndIV();
}

private void generateKeyAndIV() {
    try {
        KeyGenerator kg = KeyGenerator.getInstance("Blowfish");
        kg.init(128); // Blowfish key size can vary, 128 is safe default
        SecretKey sk = kg.generateKey();
        keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));

        byte[] iv = new byte[8]; // Blowfish block size is 8 bytes
        new SecureRandom().nextBytes(iv);
        ivField.setText(Base64.getEncoder().encodeToString(iv));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void process(boolean encrypt) {
    try {
        byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
        byte[] ivBytes = Base64.getDecoder().decode(ivField.getText());
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "Blowfish");
        IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

        Cipher cipher = Cipher.getInstance("Blowfish/CBC/PKCS5Padding");
        cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, ivSpec);

        byte[] input;
        if (encrypt) {
            input = getInputText().getBytes("UTF-8");
            byte[] output = cipher.doFinal(input);
            outputArea.setText(Base64.getEncoder().encodeToString(output));
        } else {
            input = Base64.getDecoder().decode(getInputText());
            byte[] output = cipher.doFinal(input);
            String result = new String(output, "UTF-8");
```

```
                outputArea.setText(result);
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\BookCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class BookCipherPanel extends BaseCipherPanel {
    public BookCipherPanel() {
        super("Book Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BruceCodePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class BruceCodePanel extends BaseCipherPanel {
    public BruceCodePanel() {
        super("Bruce Code (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BurrowsWheelerCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class BurrowsWheelerCipherPanel extends BaseCipherPanel {
    public BurrowsWheelerCipherPanel() {
        super("Burrows?Wheeler Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\BurrowsWheelerPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class BurrowsWheelerPanel extends BaseCipherPanel {
    public BurrowsWheelerPanel() {
        super("BurrowsWheeler (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\CadenusCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class CadenusCipherPanel extends BaseCipherPanel {
    public CadenusCipherPanel() {
        super("Cadenus Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\CaesarCipherPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.awt.*;

public class CaesarCipherPanel extends BaseCipherPanel {
    JTextField shiftField;

    public CaesarCipherPanel() {
        super("Caesar Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Input
        JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        inputPanel.setOpaque(false);
        inputPanel.add(new JLabel("<html><font color='white'>Shift:</font></html>"));
        shiftField = createTextField("3");
        shiftField.setPreferredSize(new Dimension(100, 30));
        inputPanel.add(shiftField);
        container.add(inputPanel);

        container.add(Box.createVerticalStrut(10));

        // Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        btnPanel.add(enc);
        btnPanel.add(dec);
        container.add(btnPanel);

        p.add(container);
    }

    @Override
    protected void resetFields() {
        shiftField.setText("3");
    }
```

```
    private void process(boolean encrypt) {
        try {
            String text = getInputText();
            int shift = Integer.parseInt(shiftField.getText());
            StringBuilder res = new StringBuilder();
            for (char c : text.toCharArray()) {
                if (Character.isLetter(c)) {
                    char base = Character.isUpperCase(c) ? 'A' : 'a';
                    int val = c - base;
                    int s = encrypt ? shift : -shift;
                    val = (val + s) % 26;
                    if (val < 0)
                        val += 26;
                    res.append((char) (base + val));
                } else
                    res.append(c);
            }
            outputArea.setText(res.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\CalypsoCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class CalypsoCipherPanel extends BaseCipherPanel {
    public CalypsoCipherPanel() {
        super("Calypso Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\CamelliaPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class CamelliaPanel extends BaseCipherPanel {
    public CamelliaPanel() {
        super("Camellia (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\CardanGrillePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class CardanGrillePanel extends BaseCipherPanel {
    public CardanGrillePanel() {
        super("CardanGrille (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\Cast128Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class Cast128Panel extends BaseCipherPanel {
    public Cast128Panel() {
        super("Cast128 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\Cast256Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class Cast256Panel extends BaseCipherPanel {
    public Cast256Panel() {
        super("Cast256 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ChaCha20Panel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class ChaCha20Panel extends BaseCipherPanel {
    public ChaCha20Panel() {
        super("ChaCha20 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ChaChaCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ChaChaCipherPanel extends BaseCipherPanel {
    public ChaChaCipherPanel() {
        super("ChaCha Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ChaoCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ChaoCipherPanel extends BaseCipherPanel {
    public ChaoCipherPanel() {
        super("Chao Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ChineseRemainderCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class ChineseRemainderCipherPanel extends BaseCipherPanel {
    public ChineseRemainderCipherPanel() {
        super("Chinese Remainder (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(
                e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(
                e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ClefiaPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ClefiaPanel extends BaseCipherPanel {
    public ClefiaPanel() {
        super("Clefia (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ColumnarTranspositionPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class ColumnarTranspositionPanel extends BaseCipherPanel {
    JTextField keyField;

    public ColumnarTranspositionPanel() {
        super("Columnar Transposition");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key:</font></html>"));
        keyField = createTextField("KEY");
        p.add(keyField);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        keyField.setText("KEY");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().replaceAll("\\s+", "");
            String key = keyField.getText();
            int cols = key.length();

            if (encrypt) {
                int rows = (int) Math.ceil((double) text.length() / cols);
                char[][] grid = new char[rows][cols];
                int idx = 0;
                for (int r = 0; r < rows; r++)
                    for (int c = 0; c < cols; c++) {
                        if (idx < text.length())
                            grid[r][c] = text.charAt(idx++);
                        else
                            grid[r][c] = 'X'; // Padding
                    }
                StringBuilder sb = new StringBuilder();
```

```
                for (int c = 0; c < cols; c++)
                    for (int r = 0; r < rows; r++)
                        sb.append(grid[r][c]);
                outputArea.setText(sb.toString());
            } else {
                // Simplified Decryption (Assumes rectangular grid for simplicity in this demo)
                int rows = text.length() / cols;
                char[][] grid = new char[rows][cols];
                int idx = 0;
                for (int c = 0; c < cols; c++)
                    for (int r = 0; r < rows; r++)
                        if (idx < text.length())
                            grid[r][c] = text.charAt(idx++);

                StringBuilder sb = new StringBuilder();
                for (int r = 0; r < rows; r++)
                    for (int c = 0; c < cols; c++)
                        sb.append(grid[r][c]);
                outputArea.setText(sb.toString());
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\CombinedCipherDiskPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class CombinedCipherDiskPanel extends BaseCipherPanel {
    public CombinedCipherDiskPanel() {
        super("Combined CipherDisk (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ConjugatedCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ConjugatedCipherPanel extends BaseCipherPanel {
    public ConjugatedCipherPanel() {
        super("Conjugated Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\CopticCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class CopticCipherPanel extends BaseCipherPanel {
    public CopticCipherPanel() {
        super("Coptic Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\CramerShoupPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class CramerShoupPanel extends BaseCipherPanel {
    public CramerShoupPanel() {
        super("CramerShoup (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\CryptogramSubstitutionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class CryptogramSubstitutionPanel extends BaseCipherPanel {
    public CryptogramSubstitutionPanel() {
        super("CryptogramSubstitution (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\DancingMenCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class DancingMenCipherPanel extends BaseCipherPanel {
    public DancingMenCipherPanel() {
        super("DancingMen Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\DESPanel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.security.SecureRandom;
import java.util.Base64;

public class DESPanel extends BaseCipherPanel {
    private JTextField keyField;
    private JTextField ivField;

    public DESPanel() {
        super("DES Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key
        JPanel keyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        keyPanel.setOpaque(false);
        keyPanel.add(new JLabel("<html><font color='white'>Key (Base64):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        keyPanel.add(keyField);
        container.add(keyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: IV
        JPanel ivPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        ivPanel.setOpaque(false);
        ivPanel.add(new JLabel("<html><font color='white'>IV (Base64):</font></html>"));
        ivField = createTextField("");
        ivField.setPreferredSize(new Dimension(200, 30));
        ivPanel.add(ivField);
        container.add(ivPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
```

```
    JButton gen = new NeonButton("Gen Key/IV", new Color(100, 100, 100));
    gen.addActionListener(e -> generateKeyAndIV());
    btnPanel.add(gen);

    JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
    enc.addActionListener(e -> measureAndRun(() -> process(true), true));
    btnPanel.add(enc);

    JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
    dec.addActionListener(e -> measureAndRun(() -> process(false), false));
    btnPanel.add(dec);

    container.add(btnPanel);

    p.add(container);

    generateKeyAndIV();
}

private void generateKeyAndIV() {
    try {
        KeyGenerator kg = KeyGenerator.getInstance("DES");
        SecretKey sk = kg.generateKey();
        keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));

        byte[] iv = new byte[8]; // DES block size is 8 bytes
        new SecureRandom().nextBytes(iv);
        ivField.setText(Base64.getEncoder().encodeToString(iv));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void process(boolean encrypt) {
    try {
        byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
        byte[] ivBytes = Base64.getDecoder().decode(ivField.getText());
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "DES");
        IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

        Cipher cipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
        cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, ivSpec);

        byte[] input;
        if (encrypt) {
            input = getInputText().getBytes("UTF-8");
            byte[] output = cipher.doFinal(input);
            outputArea.setText(Base64.getEncoder().encodeToString(output));
        } else {
            input = Base64.getDecoder().decode(getInputText());
            byte[] output = cipher.doFinal(input);
            String result = new String(output, "UTF-8");
            outputArea.setText(result);
```

```
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\DoubleColumnarTranspositionPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class DoubleColumnarTranspositionPanel extends BaseCipherPanel {
    private JTextField key1Field;
    private JTextField key2Field;

    public DoubleColumnarTranspositionPanel() {
        super("Double Columnar Transposition");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key 1
        JPanel k1Panel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        k1Panel.setOpaque(false);
        k1Panel.add(new JLabel("<html><font color='white'>Key 1:</font></html>"));
        key1Field = createTextField("KEYONE");
        key1Field.setPreferredSize(new Dimension(100, 30));
        k1Panel.add(key1Field);
        container.add(k1Panel);

        // Row 2: Key 2
        JPanel k2Panel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        k2Panel.setOpaque(false);
        k2Panel.add(new JLabel("<html><font color='white'>Key 2:</font></html>"));
        key2Field = createTextField("KEYTWO");
        key2Field.setPreferredSize(new Dimension(100, 30));
        k2Panel.add(key2Field);
        container.add(k2Panel);

        container.add(Box.createVerticalStrut(10));

        // Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        btnPanel.add(enc);
        btnPanel.add(dec);
```

```
    container.add(btnPanel);

    p.add(container);
}

@Override
protected void resetFields() {
    key1Field.setText("KEYONE");
    key2Field.setText("KEYTWO");
}

private void process(boolean encrypt) {
    try {
        String text = getInputText().replaceAll("\\s+", "");
        String key1 = key1Field.getText();
        String key2 = key2Field.getText();

        if (encrypt) {
            String pass1 = columnarEncrypt(text, key1);
            String pass2 = columnarEncrypt(pass1, key2);
            outputArea.setText(pass2);
        } else {
            String pass1 = columnarDecrypt(text, key2);
            String pass2 = columnarDecrypt(pass1, key1);
            outputArea.setText(pass2);
        }
    } catch (Exception e) {
        outputArea.setText("Error: " + e.getMessage());
    }
}

private String columnarEncrypt(String text, String key) {
    int cols = key.length();
    int rows = (int) Math.ceil((double) text.length() / cols);
    char[][] grid = new char[rows][cols];
    int idx = 0;

    // Fill grid row by row
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            if (idx < text.length()) {
                grid[r][c] = text.charAt(idx++);
            } else {
                grid[r][c] = 'X'; // Padding
            }
        }
    }

    // Determine column order based on key
    Integer[] order = getColumnOrder(key);

    // Read grid column by column based on order
    StringBuilder sb = new StringBuilder();
    for (int c : order) {
```

```
        for (int r = 0; r < rows; r++) {
            sb.append(grid[r][c]);
        }
    }
    return sb.toString();
}

private String columnarDecrypt(String text, String key) {
    int cols = key.length();
    int rows = text.length() / cols;
    char[][] grid = new char[rows][cols];

    Integer[] order = getColumnOrder(key);

    int idx = 0;
    // Fill grid column by column based on order
    for (int c : order) {
        for (int r = 0; r < rows; r++) {
            if (idx < text.length()) {
                grid[r][c] = text.charAt(idx++);
            }
        }
    }

    // Read grid row by row
    StringBuilder sb = new StringBuilder();
    for (int r = 0; r < rows; r++) {
        for (int c = 0; c < cols; c++) {
            sb.append(grid[r][c]);
        }
    }
    return sb.toString();
}

private Integer[] getColumnOrder(String key) {
    int len = key.length();
    Integer[] order = new Integer[len];
    Character[] chars = new Character[len];
    for (int i = 0; i < len; i++)
        chars[i] = key.charAt(i);

    // Simple sort to determine order
    // This is a naive implementation of getting indices.
    // A proper one handles duplicate letters by position.

    boolean[] used = new boolean[len];
    int orderIdx = 0;

    // Find alphabetical order
    for (char c = 0; c < 256; c++) { // Iterate through all chars
        for (int i = 0; i < len; i++) {
            if (!used[i] && key.charAt(i) == c) {
                order[orderIdx++] = i;
                used[i] = true;
```

```
                }
            }
        }

        // If key has chars not in 0-255 (unlikely for standard keys), fallback
        if (orderIdx < len) {
            for (int i = 0; i < len; i++) {
                if (!used[i])
                    order[orderIdx++] = i;
            }
        }

        // Wait, the order array should store the column index to read FIRST, SECOND,
        // etc.
        // So if key is "BAC", order is [1, 0, 2] (B is 2nd, A is 1st, C is 3rd) -> No.
        // "BAC": Sorted is ABC.
        // 1st col to read is 'A' (index 1).
        // 2nd col to read is 'B' (index 0).
        // 3rd col to read is 'C' (index 2).
        // So order = {1, 0, 2}.
        // My loop above does exactly this.

        return order;
    }
}
```

File: .\ciphers\DoubleDESPanel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.util.Base64;

public class DoubleDESPanel extends BaseCipherPanel {
    private JTextField key1Field;
    private JTextField key2Field;

    public DoubleDESPanel() {
        super("2DES");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key 1
        JPanel k1Panel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        k1Panel.setOpaque(false);
        k1Panel.add(new JLabel("<html><font color='white'>Key 1 (Base64):</font></html>"));
        key1Field = createTextField("");
        key1Field.setPreferredSize(new Dimension(150, 30));
        k1Panel.add(key1Field);
        container.add(k1Panel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: Key 2
        JPanel k2Panel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        k2Panel.setOpaque(false);
        k2Panel.add(new JLabel("<html><font color='white'>Key 2 (Base64):</font></html>"));
        key2Field = createTextField("");
        key2Field.setPreferredSize(new Dimension(150, 30));
        k2Panel.add(key2Field);
        container.add(k2Panel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);

        JButton gen = new NeonButton("Gen Keys", new Color(100, 100, 100));
```

```
        gen.addActionListener(e -> generateKeys());
        btnPanel.add(gen);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        btnPanel.add(enc);

        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        btnPanel.add(dec);

        container.add(btnPanel);

        p.add(container);

        generateKeys();
    }

    private void generateKeys() {
        try {
            KeyGenerator kg = KeyGenerator.getInstance("DES");
            SecretKey sk1 = kg.generateKey();
            SecretKey sk2 = kg.generateKey();
            key1Field.setText(Base64.getEncoder().encodeToString(sk1.getEncoded()));
            key2Field.setText(Base64.getEncoder().encodeToString(sk2.getEncoded()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void process(boolean encrypt) {
        try {
            byte[] k1 = Base64.getDecoder().decode(key1Field.getText());
            byte[] k2 = Base64.getDecoder().decode(key2Field.getText());
            SecretKeySpec sk1 = new SecretKeySpec(k1, "DES");
            SecretKeySpec sk2 = new SecretKeySpec(k2, "DES");

            Cipher c1 = Cipher.getInstance("DES/ECB/PKCS5Padding");
            // issues if possible, but PKCS5 is safer for text.
            // Actually, for Double DES: C = E2(E1(P)).
            // Intermediate result of E1(P) will be padded. E2 will pad again?
            // Standard practice: E1 pads, E2 treats as data.
            // Decrypt: P = D1(D2(C)).

            // Let's use PKCS5 for the outer layer and NoPadding for inner?
            // No, E1(P) produces blocks. E2 encrypts those blocks.
            // If we use ECB, we don't need IVs for simplicity here, though insecure.

            byte[] input;
            byte[] result;

            if (encrypt) {
                input = getInputText().getBytes("UTF-8");
```

```
                // Stage 1: Encrypt with Key 1
                c1.init(Cipher.ENCRYPT_MODE, sk1);
                byte[] stage1 = c1.doFinal(input);

                // Stage 2: Encrypt with Key 2 (Treat stage1 as raw bytes, maybe need padding if
                // not block aligned?
                // PKCS5Padding ensures block alignment).
                // So stage1 is multiple of 8 bytes.
                // We can use NoPadding for Stage 2 if we want, or PKCS5Padding again (double
                // padding).
                // Let's use PKCS5Padding for both to be safe and simple, though it adds size.

                Cipher c2Enc = Cipher.getInstance("DES/ECB/PKCS5Padding");
                c2Enc.init(Cipher.ENCRYPT_MODE, sk2);
                result = c2Enc.doFinal(stage1);

                outputArea.setText(Base64.getEncoder().encodeToString(result));
            } else {
                input = Base64.getDecoder().decode(getInputText());

                // Stage 1: Decrypt with Key 2
                Cipher c2Dec = Cipher.getInstance("DES/ECB/PKCS5Padding");
                c2Dec.init(Cipher.DECRYPT_MODE, sk2);
                byte[] stage1 = c2Dec.doFinal(input);

                // Stage 2: Decrypt with Key 1
                c1.init(Cipher.DECRYPT_MODE, sk1);
                result = c1.doFinal(stage1);

                String resStr = new String(result, "UTF-8");
                outputArea.setText(resStr);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\DoublePlayfairPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class DoublePlayfairPanel extends BaseCipherPanel {
    public DoublePlayfairPanel() {
        super("DoublePlayfair (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\DoubleTranspositionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class DoubleTranspositionPanel extends BaseCipherPanel {
    public DoubleTranspositionPanel() {
        super("DoubleTransposition (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\DraconianShiftPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class DraconianShiftPanel extends BaseCipherPanel {
    public DraconianShiftPanel() {
        super("DraconianShift (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\E0BluetoothPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class E0BluetoothPanel extends BaseCipherPanel {
    public E0BluetoothPanel() {
        super("E0Bluetooth (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ECBModePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ECBModePanel extends BaseCipherPanel {
    public ECBModePanel() {
        super("ECBMode (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ECCPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ECCPanel extends BaseCipherPanel {
    public ECCPanel() {
        super("ECC (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\Edon80Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class Edon80Panel extends BaseCipherPanel {
    public Edon80Panel() {
        super("Edon80 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ElGamalPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ElGamalPanel extends BaseCipherPanel {
    public ElGamalPanel() {
        super("ElGamal (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\EllipticCurveCipherECCPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class EllipticCurveCipherECCPanel extends BaseCipherPanel {
    public EllipticCurveCipherECCPanel() {
        super("Elliptic Curve Cipher (ECC) (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\EnigmaMachinePanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class EnigmaMachinePanel extends BaseCipherPanel {
    public EnigmaMachinePanel() {
        super("Enigma Machine (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ExponentialCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ExponentialCipherPanel extends BaseCipherPanel {
    public ExponentialCipherPanel() {
        super("Exponential Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ExtendedVigenerePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ExtendedVigenerePanel extends BaseCipherPanel {
    public ExtendedVigenerePanel() {
        super("ExtendedVigenere (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ExtendedVigenrePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ExtendedVigenrePanel extends BaseCipherPanel {
    public ExtendedVigenrePanel() {
        super("Extended Vigenère (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\FialkaCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class FialkaCipherPanel extends BaseCipherPanel {
    public FialkaCipherPanel() {
        super("Fialka Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\FourSquareCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class FourSquareCipherPanel extends BaseCipherPanel {
    public FourSquareCipherPanel() {
        super("Four-Square Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(
                                                () -> outputArea.setText("Simulation Encrypt: " + new
StringBuilder(inputArea.getText()).reverse()),
                true));

        dec.addActionListener(e -> measureAndRun(
                                                () -> outputArea.setText("Simulation Decrypt: " + new
StringBuilder(inputArea.getText()).reverse()),
                false));

        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\FractionatedMorsePanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class FractionatedMorsePanel extends BaseCipherPanel {
    public FractionatedMorsePanel() {
        super("Fractionated Morse (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(
                e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(
                e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\FreemasonCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class FreemasonCipherPanel extends BaseCipherPanel {
    public FreemasonCipherPanel() {
        super("Freemason Cipher (Pigpen)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\FrequencyCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class FrequencyCipherPanel extends BaseCipherPanel {
    public FrequencyCipherPanel() {
        super("Frequency Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\FringeCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class FringeCipherPanel extends BaseCipherPanel {
    public FringeCipherPanel() {
        super("Fringe Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GaderypolukiPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GaderypolukiPanel extends BaseCipherPanel {
    public GaderypolukiPanel() {
        super("Gaderypoluki (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GaloisCounterModePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GaloisCounterModePanel extends BaseCipherPanel {
    public GaloisCounterModePanel() {
        super("GaloisCounterMode (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GEECipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GEECipherPanel extends BaseCipherPanel {
    public GEECipherPanel() {
        super("GEE Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GIFTPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GIFTPanel extends BaseCipherPanel {
    public GIFTPanel() {
        super("GIFT (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GOSTPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GOSTPanel extends BaseCipherPanel {
    public GOSTPanel() {
        super("GOST (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GrainPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GrainPanel extends BaseCipherPanel {
    public GrainPanel() {
        super("Grain (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GrilleCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GrilleCipherPanel extends BaseCipherPanel {
    public GrilleCipherPanel() {
        super("Grille Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GromarkCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class GromarkCipherPanel extends BaseCipherPanel {
    public GromarkCipherPanel() {
        super("Gromark Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\GronsfeldCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class GronsfeldCipherPanel extends BaseCipherPanel {
    private JTextField keyField;

    public GronsfeldCipherPanel() {
        super("Gronsfeld Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key (Numbers):</font></html>"));
        keyField = createTextField("1234");
        p.add(keyField);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        keyField.setText("1234");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase();
            String key = keyField.getText().replaceAll("[^0-9]", "");
            if (key.isEmpty())
                throw new IllegalArgumentException("Key must contain numbers");

            StringBuilder result = new StringBuilder();
            int keyIdx = 0;

            for (char c : text.toCharArray()) {
                if (c < 'A' || c > 'Z') {
                    result.append(c);
                    continue;
                }

                int shift = key.charAt(keyIdx % key.length()) - '0';
                int val = c - 'A';
```

```
                if (encrypt) {
                    val = (val + shift) % 26;
                } else {
                    val = (val - shift + 26) % 26;
                }

                result.append((char) (val + 'A'));
                keyIdx++;
            }
            outputArea.setText(result.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\HagelinM209Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HagelinM209Panel extends BaseCipherPanel {
    public HagelinM209Panel() {
        super("Hagelin M-209 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HashBasedCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HashBasedCipherPanel extends BaseCipherPanel {
    public HashBasedCipherPanel() {
        super("HashBased Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HashingPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.math.BigInteger;
import java.security.MessageDigest;

public class HashingPanel extends BaseCipherPanel {
    String algo;

    public HashingPanel(String algo) {
        super(algo);
        this.algo = algo;
    }

    public HashingPanel() {
        this("SHA-256");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton btn = new NeonButton("Generate Hash", Theme.ACCENT_COLOR);
        btn.addActionListener(e -> measureAndRun(this::process, true));
        p.add(btn);
    }

    private void process() {
        try {
            String targetAlgo = algo.equals("SHA-128") ? "SHA-256" : algo;
            MessageDigest md = MessageDigest.getInstance(targetAlgo);
            byte[] digest = md.digest(getInputText().getBytes());
            if (algo.equals("SHA-128")) {
                byte[] truncated = new byte[16];
                System.arraycopy(digest, 0, truncated, 0, 16);
                digest = truncated;
            }
            BigInteger no = new BigInteger(1, digest);
            String hashtext = no.toString(16);
            while (hashtext.length() < (digest.length * 2))
                hashtext = "0" + hashtext;
            outputArea.setText(hashtext.toUpperCase());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\HebernRotorMachinePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HebernRotorMachinePanel extends BaseCipherPanel {
    public HebernRotorMachinePanel() {
        super("Hebern Rotor Machine (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HebernRotorPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HebernRotorPanel extends BaseCipherPanel {
    public HebernRotorPanel() {
        super("HebernRotor (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HightPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HightPanel extends BaseCipherPanel {
    public HightPanel() {
        super("Hight (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HillCipherPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.awt.*;

public class HillCipherPanel extends BaseCipherPanel {
    JTextField keyField;

    public HillCipherPanel() {
        super("Hill Cipher (2x2)");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Input
        JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        inputPanel.setOpaque(false);
        inputPanel.add(new JLabel("<html><font color='white'>Key (4):</font></html>"));
        keyField = createTextField("GYBN");
        keyField.setPreferredSize(new Dimension(100, 30));
        inputPanel.add(keyField);
        container.add(inputPanel);

        container.add(Box.createVerticalStrut(10));

        // Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        btnPanel.add(enc);
        btnPanel.add(dec);
        container.add(btnPanel);

        p.add(container);
    }

    @Override
    protected void resetFields() {
        keyField.setText("GYBN");
    }
```

```
    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase().replaceAll("[^A-Z]", "");
            String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "");
            if (key.length() != 4)
                throw new RuntimeException("Key must be 4 letters");
            if (text.length() % 2 != 0)
                text += "X";

            int[][] K = { { key.charAt(0) - 'A', key.charAt(1) - 'A' },
                    { key.charAt(2) - 'A', key.charAt(3) - 'A' } };
            int det = (K[0][0] * K[1][1] - K[0][1] * K[1][0]) % 26;
            if (det < 0)
                det += 26;

            if (!encrypt) {
                int detInv = -1;
                for (int i = 0; i < 26; i++)
                    if ((det * i) % 26 == 1) {
                        detInv = i;
                        break;
                    }
                if (detInv == -1)
                    throw new RuntimeException("Key not invertible");

                int temp = K[0][0];
                K[0][0] = K[1][1];
                K[1][1] = temp;
                K[0][1] = -K[0][1];
                K[1][0] = -K[1][0];

                for (int i = 0; i < 2; i++)
                    for (int j = 0; j < 2; j++) {
                        K[i][j] = (K[i][j] * detInv) % 26;
                        if (K[i][j] < 0)
                            K[i][j] += 26;
                    }
            }

            StringBuilder res = new StringBuilder();
            for (int i = 0; i < text.length(); i += 2) {
                int r1 = text.charAt(i) - 'A';
                int r2 = text.charAt(i + 1) - 'A';
                int c1 = (K[0][0] * r1 + K[0][1] * r2) % 26;
                int c2 = (K[1][0] * r1 + K[1][1] * r2) % 26;
                res.append((char) (c1 + 'A')).append((char) (c2 + 'A'));
            }
            outputArea.setText(res.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\HMACBasedSystemsPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HMACBasedSystemsPanel extends BaseCipherPanel {
    public HMACBasedSystemsPanel() {
        super("HMAC Based Systems (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HMACPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HMACPanel extends BaseCipherPanel {
    public HMACPanel() {
        super("HMAC (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HomophonicSubstitutionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HomophonicSubstitutionPanel extends BaseCipherPanel {
    public HomophonicSubstitutionPanel() {
        super("HomophonicSubstitution (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HortonCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HortonCipherPanel extends BaseCipherPanel {
    public HortonCipherPanel() {
        super("Horton Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\HuffmanCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class HuffmanCipherPanel extends BaseCipherPanel {
    public HuffmanCipherPanel() {
        super("Huffman Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\IDEALCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class IDEALCipherPanel extends BaseCipherPanel {
    public IDEALCipherPanel() {
        super("IDEAL Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\IDEAPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class IDEAPanel extends BaseCipherPanel {
    public IDEAPanel() {
        super("IDEA (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\IndirectSubstitutionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class IndirectSubstitutionPanel extends BaseCipherPanel {
    public IndirectSubstitutionPanel() {
        super("IndirectSubstitution (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\InversiveCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class InversiveCipherPanel extends BaseCipherPanel {
    public InversiveCipherPanel() {
        super("Inversive Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ISOStreamCiphersPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ISOStreamCiphersPanel extends BaseCipherPanel {
    public ISOStreamCiphersPanel() {
        super("ISO Stream Ciphers (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ISOStreamPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ISOStreamPanel extends BaseCipherPanel {
    public ISOStreamPanel() {
        super("ISOStream (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\JCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class JCipherPanel extends BaseCipherPanel {
    public JCipherPanel() {
        super("J Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\JeffersonCylinderPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class JeffersonCylinderPanel extends BaseCipherPanel {
    public JeffersonCylinderPanel() {
        super("Jefferson Cylinder (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(
                e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(
                e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\JigsawCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class JigsawCipherPanel extends BaseCipherPanel {
    public JigsawCipherPanel() {
        super("Jigsaw Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\JN25Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class JN25Panel extends BaseCipherPanel {
    public JN25Panel() {
        super("JN25 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KahnCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class KahnCipherPanel extends BaseCipherPanel {
    public KahnCipherPanel() {
        super("Kahn Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KasiskiSystemPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class KasiskiSystemPanel extends BaseCipherPanel {
    public KasiskiSystemPanel() {
        super("Kasiski System (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KasumiPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class KasumiPanel extends BaseCipherPanel {
    public KasumiPanel() {
        super("Kasumi (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KeyedCaesarPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class KeyedCaesarPanel extends BaseCipherPanel {
    public KeyedCaesarPanel() {
        super("Keyed Caesar (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KeywordCipherPanel.java

```
package ciphers;
import components.*;

import javax.swing.*;
import java.awt.*;

public class KeywordCipherPanel extends BaseCipherPanel {
    JTextField keyField;

    public KeywordCipherPanel() {
        super("Keyword Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Keyword:</font></html>"));
        keyField = createTextField("KEYWORD");
        p.add(keyField);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        keyField.setText("KEYWORD");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase();
            String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "");
            String alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
            String cipherAlpha = "";
            for (char c : key.toCharArray())
                if (!cipherAlpha.contains("" + c))
                    cipherAlpha += c;
            for (char c : alphabet.toCharArray())
                if (!cipherAlpha.contains("" + c))
                    cipherAlpha += c;

            StringBuilder sb = new StringBuilder();
            for (char c : text.toCharArray()) {
                if (Character.isLetter(c)) {
                    int idx = encrypt ? alphabet.indexOf(c) : cipherAlpha.indexOf(c);
                    sb.append(encrypt ? cipherAlpha.charAt(idx) : alphabet.charAt(idx));
                } else
                    sb.append(c);
            }
```

```
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\KleinPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class KleinPanel extends BaseCipherPanel {
    public KleinPanel() {
        super("Klein (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KlingonCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class KlingonCipherPanel extends BaseCipherPanel {
    public KlingonCipherPanel() {
        super("Klingon Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KnapsackCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class KnapsackCipherPanel extends BaseCipherPanel {
    public KnapsackCipherPanel() {
        super("Knapsack Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\KyberPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class KyberPanel extends BaseCipherPanel {
    public KyberPanel() {
        super("Kyber (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LatinSquareCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class LatinSquareCipherPanel extends BaseCipherPanel {
    public LatinSquareCipherPanel() {
        super("Latin Square Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LavarandCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class LavarandCipherPanel extends BaseCipherPanel {
    public LavarandCipherPanel() {
        super("Lavarand Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LBlockPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class LBlockPanel extends BaseCipherPanel {
    public LBlockPanel() {
        super("LBlock (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LEDPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class LEDPanel extends BaseCipherPanel {
    public LEDPanel() {
        super("LED (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LFSRCiphersPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class LFSRCiphersPanel extends BaseCipherPanel {
    public LFSRCiphersPanel() {
        super("LFSR Ciphers (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LinearSubstitutionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class LinearSubstitutionPanel extends BaseCipherPanel {
    public LinearSubstitutionPanel() {
        super("Linear Substitution (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LogicalXORCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class LogicalXORCipherPanel extends BaseCipherPanel {
    public LogicalXORCipherPanel() {
        super("Logical XOR Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\LorenzSZ40Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class LorenzSZ40Panel extends BaseCipherPanel {
    public LorenzSZ40Panel() {
        super("LorenzSZ40 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\M138CipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class M138CipherPanel extends BaseCipherPanel {
    public M138CipherPanel() {
        super("M-138 Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\M209Panel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class M209Panel extends BaseCipherPanel {
    public M209Panel() {
        super("Hagelin M-209 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\M94CipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class M94CipherPanel extends BaseCipherPanel {
    public M94CipherPanel() {
        super("M-94 Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\MACBaconCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class MACBaconCipherPanel extends BaseCipherPanel {
    public MACBaconCipherPanel() {
        super("MAC-Bacon Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\MalbolgeEncryptionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class MalbolgeEncryptionPanel extends BaseCipherPanel {
    public MalbolgeEncryptionPanel() {
        super("Malbolge Encryption (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\MarsPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class MarsPanel extends BaseCipherPanel {
    public MarsPanel() {
        super("Mars (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\MatrixCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class MatrixCipherPanel extends BaseCipherPanel {
    public MatrixCipherPanel() {
        super("Matrix Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\McEliecePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class McEliecePanel extends BaseCipherPanel {
    public McEliecePanel() {
        super("McEliece (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\MerkleHellmanPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class MerkleHellmanPanel extends BaseCipherPanel {
    public MerkleHellmanPanel() {
        super("Merkle?Hellman (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\Misty1Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class Misty1Panel extends BaseCipherPanel {
    public Misty1Panel() {
        super("Misty1 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ModularCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ModularCipherPanel extends BaseCipherPanel {
    public ModularCipherPanel() {
        super("Modular Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\MonoalphabeticSubstitutionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class MonoalphabeticSubstitutionPanel extends BaseCipherPanel {
    public MonoalphabeticSubstitutionPanel() {
        super("Monoalphabetic Substitution (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\MorseCodePanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.awt.*;

public class MorseCodePanel extends BaseCipherPanel {
     String[] alpha = { "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", "2", "3", "4", "5", "6", "7", "8", "9", "0", " " };
     String[] morse = { ".-", "-...", "-.-.", "-..", ".", "..-.", "--.", "....", "..", ".---", "-.-", ".-..",
"--",
               "-.", "---", ".--.", "--.-", ".-.", "...", "-", "..-", "...-", ".--", "-..-", "-.--", "--..",
".----",
            "..---", ".....", "....-", ".....", "-....", "--...", "---..", "----.", "-----", "/" };

    public MorseCodePanel() {
        super("Morse Code");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase();
            StringBuilder sb = new StringBuilder();
            if (encrypt) {
                for (char c : text.toCharArray()) {
                    for (int i = 0; i < alpha.length; i++) {
                        if (alpha[i].equals("" + c)) {
                            sb.append(morse[i]).append(" ");
                            break;
                        }
                    }
                }
            } else {
                String[] parts = text.split("\\s+");
                for (String p : parts) {
                    for (int i = 0; i < morse.length; i++) {
                        if (morse[i].equals(p)) {
                            sb.append(alpha[i]);
                            break;
                        }
```

```
                    }
                }
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\MyszkowskiTranspositionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class MyszkowskiTranspositionPanel extends BaseCipherPanel {
    public MyszkowskiTranspositionPanel() {
        super("Myszkowski Transposition (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\NATOCodePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class NATOCodePanel extends BaseCipherPanel {
    public NATOCodePanel() {
        super("NATO Code (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\NihilistCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class NihilistCipherPanel extends BaseCipherPanel {
    public NihilistCipherPanel() {
        super("Nihilist Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\NoekeonPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class NoekeonPanel extends BaseCipherPanel {
    public NoekeonPanel() {
        super("Noekeon (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\NTRUPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class NTRUPanel extends BaseCipherPanel {
    public NTRUPanel() {
        super("NTRU (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\NullCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class NullCipherPanel extends BaseCipherPanel {
    public NullCipherPanel() {
        super("Null Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\NumberStationCiphersPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class NumberStationCiphersPanel extends BaseCipherPanel {
    public NumberStationCiphersPanel() {
        super("Number Station Ciphers (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ObfuscatedTranspositionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ObfuscatedTranspositionPanel extends BaseCipherPanel {
    public ObfuscatedTranspositionPanel() {
        super("Obfuscated Transposition (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\OctalCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class OctalCipherPanel extends BaseCipherPanel {
    public OctalCipherPanel() {
        super("Octal Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\OneTimePadPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class OneTimePadPanel extends BaseCipherPanel {
    public OneTimePadPanel() {
        super("One-Time Pad (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\OTARCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class OTARCipherPanel extends BaseCipherPanel {
    public OTARCipherPanel() {
        super("OTAR Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\OutofBandCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class OutofBandCipherPanel extends BaseCipherPanel {
    public OutofBandCipherPanel() {
        super("Out-of-Band Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PaddingOracleCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PaddingOracleCipherPanel extends BaseCipherPanel {
    public PaddingOracleCipherPanel() {
        super("Padding Oracle Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PanamaPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PanamaPanel extends BaseCipherPanel {
    public PanamaPanel() {
        super("Panama (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PasigraphyCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PasigraphyCipherPanel extends BaseCipherPanel {
    public PasigraphyCipherPanel() {
        super("Pasigraphy Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PBKDF2BasedSystemsPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PBKDF2BasedSystemsPanel extends BaseCipherPanel {
    public PBKDF2BasedSystemsPanel() {
        super("PBKDF2-Based Systems (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PermutationCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class PermutationCipherPanel extends BaseCipherPanel {
    public PermutationCipherPanel() {
        super("Permutation Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PigpenCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PigpenCipherPanel extends BaseCipherPanel {
    public PigpenCipherPanel() {
        super("Pigpen Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PlayfairCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;
import java.util.HashSet;
import java.util.Set;

public class PlayfairCipherPanel extends BaseCipherPanel {
    private JTextField keyField;
    private char[][] matrix = new char[5][5];

    public PlayfairCipherPanel() {
        super("Playfair Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Input
        JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        inputPanel.setOpaque(false);
        inputPanel.add(new JLabel("<html><font color='white'>Key:</font></html>"));
        keyField = createTextField("PLAYFAIR");
        keyField.setPreferredSize(new Dimension(150, 30));
        inputPanel.add(keyField);
        container.add(inputPanel);

        container.add(Box.createVerticalStrut(10));

        // Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        btnPanel.add(enc);
        btnPanel.add(dec);
        container.add(btnPanel);

        p.add(container);
    }

    @Override
    protected void resetFields() {
        keyField.setText("PLAYFAIR");
```

```
}

private void process(boolean encrypt) {
    try {
        String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");
        String text = getInputText().toUpperCase().replaceAll("[^A-Z]", "").replace("J", "I");

        if (key.isEmpty()) {
            outputArea.setText("Error: Key cannot be empty.");
            return;
        }

        generateMatrix(key);

        String result;
        if (encrypt) {
            String prepared = prepareText(text);
            result = transform(prepared, true);
        } else {
            result = transform(text, false);
        }

        outputArea.setText(result);

    } catch (Exception e) {
        outputArea.setText("Error: " + e.getMessage());
        e.printStackTrace();
    }
}

private void generateMatrix(String key) {
    Set<Character> used = new HashSet<>();
    StringBuilder sb = new StringBuilder();

    for (char c : key.toCharArray()) {
        if (!used.contains(c)) {
            used.add(c);
            sb.append(c);
        }
    }

    for (char c = 'A'; c <= 'Z'; c++) {
        if (c == 'J')
            continue;
        if (!used.contains(c)) {
            used.add(c);
            sb.append(c);
        }
    }

    int k = 0;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            matrix[i][j] = sb.charAt(k++);
```

```
        }
    }
}

private String prepareText(String text) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < text.length(); i++) {
        char c = text.charAt(i);
        sb.append(c);
        if (i + 1 < text.length()) {
            if (text.charAt(i + 1) == c) {
                sb.append('X');
            }
        }
    }
    if (sb.length() % 2 != 0) {
        sb.append('X');
    }
    return sb.toString();
}

private String transform(String text, boolean encrypt) {
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < text.length(); i += 2) {
        char a = text.charAt(i);
        char b = text.charAt(i + 1);
        int[] posA = findPos(a);
        int[] posB = findPos(b);

        int r1 = posA[0], c1 = posA[1];
        int r2 = posB[0], c2 = posB[1];

        if (r1 == r2) { // Same row
            c1 = (c1 + (encrypt ? 1 : 4)) % 5;
            c2 = (c2 + (encrypt ? 1 : 4)) % 5;
        } else if (c1 == c2) { // Same column
            r1 = (r1 + (encrypt ? 1 : 4)) % 5;
            r2 = (r2 + (encrypt ? 1 : 4)) % 5;
        } else { // Rectangle
            int temp = c1;
            c1 = c2;
            c2 = temp;
        }

        sb.append(matrix[r1][c1]);
        sb.append(matrix[r2][c2]);
    }
    return sb.toString();
}

private int[] findPos(char c) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (matrix[i][j] == c) {
```

```
                    return new int[] { i, j };
                }
            }
        }
        return new int[] { 0, 0 }; // Should not happen
    }
}
```

File: .\ciphers\PolybiusSquarePanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class PolybiusSquarePanel extends BaseCipherPanel {
    public PolybiusSquarePanel() {
        super("Polybius Square");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        p.add(enc);
        p.add(dec);
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase();
            StringBuilder sb = new StringBuilder();

            if (encrypt) {
                for (char c : text.toCharArray()) {
                    if (c == 'J')
                        c = 'I';
                    if (c >= 'A' && c <= 'Z') {
                        int row = (c - 'A') / 5 + 1;
                        int col = (c - 'A') % 5 + 1;
                        if (c > 'I') {
                            row = (c - 'A' - 1) / 5 + 1;
                            col = (c - 'A' - 1) % 5 + 1;
                        }
                        sb.append(row).append(col).append(" ");
                    }
                }
            } else {
                String[] parts = text.trim().split("\\s+");
                char[][] grid = {
                        { 'A', 'B', 'C', 'D', 'E' },
                        { 'F', 'G', 'H', 'I', 'K' },
                        { 'L', 'M', 'N', 'O', 'P' },
                        { 'Q', 'R', 'S', 'T', 'U' },
                        { 'V', 'W', 'X', 'Y', 'Z' }
                };
                for (String part : parts) {
```

```
                if (part.length() == 2) {
                    int r = Character.getNumericValue(part.charAt(0)) - 1;
                    int c = Character.getNumericValue(part.charAt(1)) - 1;
                    if (r >= 0 && r < 5 && c >= 0 && c < 5) {
                        sb.append(grid[r][c]);
                    }
                }
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\PortaCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class PortaCipherPanel extends BaseCipherPanel {
    private JTextField keyField;

    public PortaCipherPanel() {
        super("Porta Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key:</font></html>"));
        keyField = createTextField("KEY");
        p.add(keyField);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        keyField.setText("KEY");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase();
            String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "");
            if (key.isEmpty())
                throw new IllegalArgumentException("Key cannot be empty");

            StringBuilder result = new StringBuilder();
            int keyIdx = 0;

            for (char c : text.toCharArray()) {
                if (c < 'A' || c > 'Z') {
                    result.append(c);
                    continue;
                }

                char k = key.charAt(keyIdx % key.length());
                // Porta Table Logic
```

```
                // Keys A,B use row 0; C,D use row 1; ...
                int row = (k - 'A') / 2;

                // Row 0 (A,B): A<->N, B<->O, ... (Shift 13)
                // Row 1 (C,D): A<->O, B<->P, ... (Shift ?)
                // Actually Porta is reciprocal.
                // Row 0: ABCDEFGHIJKLM NOPQRSTUVWXYZ
                // NOPQRSTUVWXYZ ABCDEFGHIJKLM
                // Row 1: ABCDEFGHIJKLM NOPQRSTUVWXYZ
                // OPQRSTUVWXYZN MABCDEFGHIJKL (Shifted?)

                // Standard Porta Table:
                // Key A,B: N O P Q R S T U V W X Y Z | A B C D E F G H I J K L M
                // Key C,D: O P Q R S T U V W X Y Z N | M A B C D E F G H I J K L
                // ...

                // First half (A-M) maps to second half (N-Z) and vice versa.
                // The second half is shifted.

                int val = c - 'A';
                int mapped;

                if (val < 13) { // A-M
                    // Maps to N-Z part
                    // Base N (13) + shift
                    mapped = (val + row) % 13 + 13;
                } else { // N-Z
                    // Maps to A-M part
                    // Base 0 + shift inverse?
                    // Let's look at the pattern.
                    // If row=0, N(13) -> A(0). (13-13-0 = 0)
                    // If row=1, O(14) -> A(0). (14-13-1 = 0)
                    mapped = (val - 13 - row + 13) % 13; // +13 to ensure positive
                    // Wait, (val - 13 - row) might be negative.
                    mapped = (val - 13 - row);
                    while (mapped < 0)
                        mapped += 13;
                    mapped %= 13;
                }

                result.append((char) (mapped + 'A'));
                keyIdx++;
            }
            outputArea.setText(result.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\PresentPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PresentPanel extends BaseCipherPanel {
    public PresentPanel() {
        super("Present (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PrimeNumberCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PrimeNumberCipherPanel extends BaseCipherPanel {
    public PrimeNumberCipherPanel() {
        super("Prime Number Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PrincePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PrincePanel extends BaseCipherPanel {
    public PrincePanel() {
        super("Prince (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PSKCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PSKCipherPanel extends BaseCipherPanel {
    public PSKCipherPanel() {
        super("PSK Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\PufferfishCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class PufferfishCipherPanel extends BaseCipherPanel {
    public PufferfishCipherPanel() {
        super("Pufferfish Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\QuadraticCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class QuadraticCipherPanel extends BaseCipherPanel {
    public QuadraticCipherPanel() {
        super("Quadratic Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\QuagmireIIVPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class QuagmireIIVPanel extends BaseCipherPanel {
    public QuagmireIIVPanel() {
        super("Quagmire I?IV (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\QuantumCiphersPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class QuantumCiphersPanel extends BaseCipherPanel {
    public QuantumCiphersPanel() {
        super("Quantum Ciphers (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RabbitPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class RabbitPanel extends BaseCipherPanel {
    public RabbitPanel() {
        super("Rabbit (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RailFenceCipherPanel.java

```
package ciphers;
import components.*;

import javax.swing.*;
import java.awt.*;
import java.util.Arrays;

public class RailFenceCipherPanel extends BaseCipherPanel {
    JTextField depthField;

    public RailFenceCipherPanel() {
        super("Rail-Fence Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Depth:</font></html>"));
        depthField = createTextField("2");
        p.add(depthField);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        depthField.setText("2");
    }

    private void process(boolean encrypt) {
        try {
            int depth = Integer.parseInt(depthField.getText());
            if (depth < 2)
                throw new RuntimeException("Depth >= 2");
            String text = getInputText();
            outputArea.setText(encrypt ? enc(text, depth) : dec(text, depth));
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }

    private String enc(String s, int k) {
        if (k <= 1)
            return s;
        StringBuilder[] rail = new StringBuilder[k];
        for (int i = 0; i < k; i++)
            rail[i] = new StringBuilder();
        boolean down = false;
        int row = 0;
```

```
        for (char c : s.toCharArray()) {
            rail[row].append(c);
            if (row == 0 || row == k - 1)
                down = !down;
            row += down ? 1 : -1;
        }
        StringBuilder res = new StringBuilder();
        for (StringBuilder sb : rail)
            res.append(sb);
        return res.toString();
    }

    private String dec(String s, int k) {
        if (k <= 1)
            return s;
        int len = s.length();
        char[][] rail = new char[k][len];
        for (int i = 0; i < k; i++)
            Arrays.fill(rail[i], '\n');
        boolean down = false;
        int row = 0;
        for (int i = 0; i < len; i++) {
            rail[row][i] = '*';
            if (row == 0 || row == k - 1)
                down = !down;
            row += down ? 1 : -1;
        }
        int idx = 0;
        for (int i = 0; i < k; i++)
            for (int j = 0; j < len; j++)
                if (rail[i][j] == '*' && idx < len)
                    rail[i][j] = s.charAt(idx++);
        StringBuilder res = new StringBuilder();
        down = false;
        row = 0;
        for (int i = 0; i < len; i++) {
            res.append(rail[row][i]);
            if (row == 0 || row == k - 1)
                down = !down;
            row += down ? 1 : -1;
        }
        return res.toString();
    }
}
```

File: .\ciphers\RandomSubstitutionPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class RandomSubstitutionPanel extends BaseCipherPanel {
    public RandomSubstitutionPanel() {
        super("Random Substitution (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RC2Panel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.security.SecureRandom;
import java.util.Base64;

public class RC2Panel extends BaseCipherPanel {
    private JTextField keyField;
    private JTextField ivField;

    public RC2Panel() {
        super("RC2 Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key
        JPanel keyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        keyPanel.setOpaque(false);
        keyPanel.add(new JLabel("<html><font color='white'>Key (Base64):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        keyPanel.add(keyField);
        container.add(keyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: IV
        JPanel ivPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        ivPanel.setOpaque(false);
        ivPanel.add(new JLabel("<html><font color='white'>IV (Base64):</font></html>"));
        ivField = createTextField("");
        ivField.setPreferredSize(new Dimension(200, 30));
        ivPanel.add(ivField);
        container.add(ivPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
```

```
    JButton gen = new NeonButton("Gen Key/IV", new Color(100, 100, 100));
    gen.addActionListener(e -> generateKeyAndIV());
    btnPanel.add(gen);

    JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
    enc.addActionListener(e -> measureAndRun(() -> process(true), true));
    btnPanel.add(enc);

    JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
    dec.addActionListener(e -> measureAndRun(() -> process(false), false));
    btnPanel.add(dec);

    container.add(btnPanel);

    p.add(container);

    generateKeyAndIV();
}

private void generateKeyAndIV() {
    try {
        KeyGenerator kg = KeyGenerator.getInstance("RC2");
        kg.init(128);
        SecretKey sk = kg.generateKey();
        keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));

        byte[] iv = new byte[8]; // RC2 block size 8 bytes
        new SecureRandom().nextBytes(iv);
        ivField.setText(Base64.getEncoder().encodeToString(iv));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void process(boolean encrypt) {
    try {
        byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
        byte[] ivBytes = Base64.getDecoder().decode(ivField.getText());
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "RC2");
        IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

        Cipher cipher = Cipher.getInstance("RC2/CBC/PKCS5Padding");
        cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, ivSpec);

        byte[] input;
        if (encrypt) {
            input = getInputText().getBytes("UTF-8");
            byte[] output = cipher.doFinal(input);
            outputArea.setText(Base64.getEncoder().encodeToString(output));
        } else {
            input = Base64.getDecoder().decode(getInputText());
            byte[] output = cipher.doFinal(input);
            String result = new String(output, "UTF-8");
```

```
                outputArea.setText(result);
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\RC4Panel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.util.Base64;

public class RC4Panel extends BaseCipherPanel {
    private JTextField keyField;

    public RC4Panel() {
        super("RC4 Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("Key (Base64):"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        p.add(keyField);

        JButton gen = new NeonButton("Gen Key", new Color(100, 100, 100));
        gen.addActionListener(e -> generateKey());
        p.add(gen);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        p.add(enc);

        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));
        p.add(dec);

        generateKey();
    }

    private void generateKey() {
        try {
            KeyGenerator kg = KeyGenerator.getInstance("RC4");
            kg.init(128);
            SecretKey sk = kg.generateKey();
            keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    private void process(boolean encrypt) {
```

```
        try {
            byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
            SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "RC4");

            Cipher cipher = Cipher.getInstance("RC4");
            cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec);

            byte[] input;
            if (encrypt) {
                input = getInputText().getBytes("UTF-8");
                byte[] output = cipher.doFinal(input);
                outputArea.setText(Base64.getEncoder().encodeToString(output));
            } else {
                input = Base64.getDecoder().decode(getInputText());
                byte[] output = cipher.doFinal(input);
                String result = new String(output, "UTF-8");
                outputArea.setText(result);
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\RC5Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class RC5Panel extends BaseCipherPanel {
    public RC5Panel() {
        super("RC5 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RC6Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class RC6Panel extends BaseCipherPanel {
    public RC6Panel() {
        super("RC6 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RectanglePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class RectanglePanel extends BaseCipherPanel {
    public RectanglePanel() {
        super("Rectangle (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RedefenseCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class RedefenseCipherPanel extends BaseCipherPanel {
    public RedefenseCipherPanel() {
        super("Redefense Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RivestCiphersPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class RivestCiphersPanel extends BaseCipherPanel {
    public RivestCiphersPanel() {
        super("Rivest Ciphers (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ROT13Panel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class ROT13Panel extends BaseCipherPanel {
    public ROT13Panel() {
        super("ROT13");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(this::process, true));
        dec.addActionListener(e -> measureAndRun(this::process, false));

        p.add(enc);
        p.add(dec);
    }

    private void process() {
        try {
            String text = getInputText();
            StringBuilder sb = new StringBuilder();
            for (char c : text.toCharArray()) {
                if (c >= 'a' && c <= 'z')
                    sb.append((char) ((c - 'a' + 13) % 26 + 'a'));
                else if (c >= 'A' && c <= 'Z')
                    sb.append((char) ((c - 'A' + 13) % 26 + 'A'));
                else
                    sb.append(c);
            }
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\ROT1ROT47Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ROT1ROT47Panel extends BaseCipherPanel {
    public ROT1ROT47Panel() {
        super("ROT1?ROT47 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\RSAPanel.java

```
package ciphers;
import components.*;

import javax.crypto.Cipher;
import javax.swing.*;
import java.awt.*;
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.util.Base64;

public class RSAPanel extends BaseCipherPanel {
    KeyPair kp;

    public RSAPanel() {
        super("RSA Encryption");
        try {
            KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
            kpg.initialize(1024);
            kp = kpg.generateKeyPair();
        } catch (Exception e) {
        }
    }

    @Override
    protected void addControls(JPanel p) {
        JButton gen = new NeonButton("New Keys", new Color(100, 100, 100));
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        gen.addActionListener(e -> {
            try {
                KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
                kpg.initialize(1024);
                kp = kpg.generateKeyPair();
                outputArea.setText("New KeyPair Generated!");
            } catch (Exception ex) {
            }
        });

        enc.addActionListener(e -> measureAndRun(() -> {
            try {
                Cipher c = Cipher.getInstance("RSA");
                c.init(Cipher.ENCRYPT_MODE, kp.getPublic());
                byte[] b = c.doFinal(getInputText().getBytes());
                outputArea.setText(Base64.getEncoder().encodeToString(b));
            } catch (Exception ex) {
                throw new RuntimeException(ex.getMessage());
            }
        }, true));

        dec.addActionListener(e -> measureAndRun(() -> {
            try {
```

```
                Cipher c = Cipher.getInstance("RSA");
                c.init(Cipher.DECRYPT_MODE, kp.getPrivate());
                byte[] b = c.doFinal(Base64.getDecoder().decode(getInputText()));
                outputArea.setText(new String(b));
            } catch (Exception ex) {
                throw new RuntimeException(ex.getMessage());
            }
        }, false));

        p.add(gen);
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\Salsa20Panel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class Salsa20Panel extends BaseCipherPanel {
    public Salsa20Panel() {
        super("Salsa20 (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ScytaleCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class ScytaleCipherPanel extends BaseCipherPanel {
    private JTextField diameterField;

    public ScytaleCipherPanel() {
        super("Scytale Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Diameter:</font></html>"));
        diameterField = createTextField("4");
        p.add(diameterField);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        diameterField.setText("4");
    }

    private void process(boolean encrypt) {
        try {
            String text = getInputText();
            int diameter = Integer.parseInt(diameterField.getText());
            if (diameter <= 0)
                throw new IllegalArgumentException("Diameter must be positive");

            outputArea.setText(encrypt ? encryptScytale(text, diameter) : decryptScytale(text, diameter));
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }

    private String encryptScytale(String text, int diameter) {
        StringBuilder result = new StringBuilder();
        int len = text.length();
        for (int i = 0; i < diameter; i++) {
            for (int j = i; j < len; j += diameter) {
```

```
                result.append(text.charAt(j));
            }
        }
        return result.toString();
    }

    private String decryptScytale(String text, int diameter) {
        int len = text.length();
        // Inverse of encryption logic
        char[] decrypted = new char[len];
        int idx = 0;
        for (int i = 0; i < diameter; i++) {
            for (int j = i; j < len; j += diameter) {
                decrypted[j] = text.charAt(idx++);
            }
        }
        return new String(decrypted);

    }
}
```

File: .\ciphers\SeedPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SeedPanel extends BaseCipherPanel {
    public SeedPanel() {
        super("Seed (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SerpentPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SerpentPanel extends BaseCipherPanel {
    public SerpentPanel() {
        super("Serpent (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SHABasedCiphersPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SHABasedCiphersPanel extends BaseCipherPanel {
    public SHABasedCiphersPanel() {
        super("SHA-Based Ciphers (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ShiftCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ShiftCipherPanel extends BaseCipherPanel {
    public ShiftCipherPanel() {
        super("Shift Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SIGABAPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SIGABAPanel extends BaseCipherPanel {
    public SIGABAPanel() {
        super("SIGABA (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SimonPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SimonPanel extends BaseCipherPanel {
    public SimonPanel() {
        super("Simon (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SimpleSubstitutionPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class SimpleSubstitutionPanel extends BaseCipherPanel {
    private JTextField keyField;
    private static final String ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

    public SimpleSubstitutionPanel() {
        super("Simple Substitution");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key (26 chars):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(250, 30));
        p.add(keyField);

        JButton gen = new NeonButton("Random Key", new Color(100, 100, 100));
        gen.addActionListener(e -> generateKey());
        p.add(gen);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        p.add(enc);
        p.add(dec);

        generateKey();
    }

    private void generateKey() {
        List<Character> chars = new ArrayList<>();
        for (char c : ALPHABET.toCharArray()) {
            chars.add(c);
        }
        Collections.shuffle(chars);
        StringBuilder sb = new StringBuilder();
        for (char c : chars) {
            sb.append(c);
        }
        keyField.setText(sb.toString());
    }
```

```
    @Override
    protected void resetFields() {
        generateKey();
    }

    private void process(boolean encrypt) {
        try {
            String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "");
            if (key.length() != 26) {
                // Check if it has duplicates or missing chars?
                // For now just warn length
                if (key.length() < 26)
                    throw new IllegalArgumentException("Key must be 26 unique letters.");
            }

            String text = getInputText().toUpperCase();
            StringBuilder result = new StringBuilder();

            String source = encrypt ? ALPHABET : key;
            String target = encrypt ? key : ALPHABET;

            for (char c : text.toCharArray()) {
                int index = source.indexOf(c);
                if (index != -1) {
                    result.append(target.charAt(index));
                } else {
                    result.append(c);
                }
            }
            outputArea.setText(result.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\SkipjackPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class SkipjackPanel extends BaseCipherPanel {
    public SkipjackPanel() {
        super("Skipjack (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\Snow3GPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class Snow3GPanel extends BaseCipherPanel {
    public Snow3GPanel() {
        super("Snow3G (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SolitaireCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class SolitaireCipherPanel extends BaseCipherPanel {
    public SolitaireCipherPanel() {
        super("Solitaire Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SonofPermutationPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SonofPermutationPanel extends BaseCipherPanel {
    public SonofPermutationPanel() {
        super("Son-of-Permutation (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SpartanScytalePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SpartanScytalePanel extends BaseCipherPanel {
    public SpartanScytalePanel() {
        super("Spartan Scytale (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SpeckPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SpeckPanel extends BaseCipherPanel {
    public SpeckPanel() {
        super("Speck (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SPNNetworkPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SPNNetworkPanel extends BaseCipherPanel {
    public SPNNetworkPanel() {
        super("SPN Network (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\StraddlingCheckerboardPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class StraddlingCheckerboardPanel extends BaseCipherPanel {
    public StraddlingCheckerboardPanel() {
        super("Straddling Checkerboard (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(
                e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(
                e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\StreamCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class StreamCipherPanel extends BaseCipherPanel {
    public StreamCipherPanel() {
        super("Stream Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\SwagmanCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class SwagmanCipherPanel extends BaseCipherPanel {
    public SwagmanCipherPanel() {
        super("Swagman Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TabulaRectaPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class TabulaRectaPanel extends BaseCipherPanel {
    public TabulaRectaPanel() {
        super("Tabula Recta (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TapCodePanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class TapCodePanel extends BaseCipherPanel {
    public TapCodePanel() {
        super("Tap Code (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TEAPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class TEAPanel extends BaseCipherPanel {
    public TEAPanel() {
        super("TEA (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TemplarCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class TemplarCipherPanel extends BaseCipherPanel {
    public TemplarCipherPanel() {
        super("Templar Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\ThreeSquareCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class ThreeSquareCipherPanel extends BaseCipherPanel {
    public ThreeSquareCipherPanel() {
        super("Three-Square Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TrifidCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class TrifidCipherPanel extends BaseCipherPanel {
    public TrifidCipherPanel() {
        super("Trifid Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TripleDESPanel.java

```
package ciphers;

import components.*;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.swing.*;
import java.awt.*;
import java.security.SecureRandom;
import java.util.Base64;

public class TripleDESPanel extends BaseCipherPanel {
    private JTextField keyField;
    private JTextField ivField;

    public TripleDESPanel() {
        super("3DES");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Row 1: Key
        JPanel keyPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        keyPanel.setOpaque(false);
        keyPanel.add(new JLabel("<html><font color='white'>Key (Base64):</font></html>"));
        keyField = createTextField("");
        keyField.setPreferredSize(new Dimension(200, 30));
        keyPanel.add(keyField);
        container.add(keyPanel);

        container.add(Box.createVerticalStrut(5));

        // Row 2: IV
        JPanel ivPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        ivPanel.setOpaque(false);
        ivPanel.add(new JLabel("<html><font color='white'>IV (Base64):</font></html>"));
        ivField = createTextField("");
        ivField.setPreferredSize(new Dimension(200, 30));
        ivPanel.add(ivField);
        container.add(ivPanel);

        container.add(Box.createVerticalStrut(10));

        // Row 3: Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
```

```
    JButton gen = new NeonButton("Gen Key/IV", new Color(100, 100, 100));
    gen.addActionListener(e -> generateKeyAndIV());
    btnPanel.add(gen);

    JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
    enc.addActionListener(e -> measureAndRun(() -> process(true), true));
    btnPanel.add(enc);

    JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
    dec.addActionListener(e -> measureAndRun(() -> process(false), false));
    btnPanel.add(dec);

    container.add(btnPanel);

    p.add(container);

    generateKeyAndIV();
}

private void generateKeyAndIV() {
    try {
        KeyGenerator kg = KeyGenerator.getInstance("DESede");
        kg.init(168); // 168 bits for 3-key 3DES
        SecretKey sk = kg.generateKey();
        keyField.setText(Base64.getEncoder().encodeToString(sk.getEncoded()));

        byte[] iv = new byte[8]; // DES block size is 8 bytes
        new SecureRandom().nextBytes(iv);
        ivField.setText(Base64.getEncoder().encodeToString(iv));
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void process(boolean encrypt) {
    try {
        byte[] keyBytes = Base64.getDecoder().decode(keyField.getText());
        byte[] ivBytes = Base64.getDecoder().decode(ivField.getText());
        SecretKeySpec keySpec = new SecretKeySpec(keyBytes, "DESede");
        IvParameterSpec ivSpec = new IvParameterSpec(ivBytes);

        Cipher cipher = Cipher.getInstance("DESede/CBC/PKCS5Padding");
        cipher.init(encrypt ? Cipher.ENCRYPT_MODE : Cipher.DECRYPT_MODE, keySpec, ivSpec);

        byte[] input;
        if (encrypt) {
            input = getInputText().getBytes("UTF-8");
            byte[] output = cipher.doFinal(input);
            outputArea.setText(Base64.getEncoder().encodeToString(output));
        } else {
            input = Base64.getDecoder().decode(getInputText());
            byte[] output = cipher.doFinal(input);
            String result = new String(output, "UTF-8");
```

```
                outputArea.setText(result);
                System.out.println("Decrypted Text: " + result);
            }
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

File: .\ciphers\TrithemeCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class TrithemeCipherPanel extends BaseCipherPanel {
    public TrithemeCipherPanel() {
        super("Tritheme Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> {
            String text = inputArea.getText().toUpperCase();
            StringBuilder sb = new StringBuilder();
            int shift = 0;
            for (char c : text.toCharArray()) {
                if (Character.isLetter(c)) {
                    sb.append((char) ((c - 'A' + shift) % 26 + 'A'));
                    shift++;
                } else
                    sb.append(c);
            }
            outputArea.setText(sb.toString());
        }, true));

           dec.addActionListener(e -> measureAndRun(() -> outputArea.setText("Tritheme Decrypt (Simulated)"),
false));

        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TwofishPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class TwofishPanel extends BaseCipherPanel {
    public TwofishPanel() {
        super("Twofish (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(
                () -> outputArea.setText("Twofish Encrypt (Simulated)"), true));

        dec.addActionListener(e -> measureAndRun(
                () -> outputArea.setText("Twofish Decrypt (Simulated)"), false));

        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\TypexCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class TypexCipherPanel extends BaseCipherPanel {
    public TypexCipherPanel() {
        super("Typex Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\UltraCipherPanel.java

```
package ciphers;
import components.*;
import javax.swing.*;
import java.awt.*;

public class UltraCipherPanel extends BaseCipherPanel {
    public UltraCipherPanel() {
        super("Ultra Cipher (Simulated)");
    }

    @Override
    protected void addControls(JPanel p) {
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));
        enc.addActionListener(e -> measureAndRun(() -> runSimulation(true), true));
        dec.addActionListener(e -> measureAndRun(() -> runSimulation(false), false));
        p.add(enc);
        p.add(dec);
    }
}
```

File: .\ciphers\VernamCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class VernamCipherPanel extends BaseCipherPanel {
    JTextField keyField;

    public VernamCipherPanel() {
        super("Vernam Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key:</font></html>"));
        keyField = createTextField("");
        p.add(keyField);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(this::process, true));
        dec.addActionListener(e -> measureAndRun(this::process, false));

        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        keyField.setText("");
    }

    private void process() {
        try {
            String text = getInputText();
            String key = keyField.getText();
            if (text.length() != key.length())
                throw new RuntimeException("Key length must equal text length");
            StringBuilder res = new StringBuilder();
            for (int i = 0; i < text.length(); i++)
                res.append((char) (text.charAt(i) ^ key.charAt(i)));
            outputArea.setText(res.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\VigenereCipherPanel.java

```
package ciphers;

import components.*;

import javax.swing.*;
import java.awt.*;

public class VigenereCipherPanel extends BaseCipherPanel {
    JTextField keyField;

    public VigenereCipherPanel() {
        super("Vigenere Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        JPanel container = new JPanel();
        container.setLayout(new BoxLayout(container, BoxLayout.Y_AXIS));
        container.setOpaque(false);

        // Input
        JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        inputPanel.setOpaque(false);
        inputPanel.add(new JLabel("<html><font color='white'>Key:</font></html>"));
        keyField = createTextField("KEY");
        keyField.setPreferredSize(new Dimension(150, 30));
        inputPanel.add(keyField);
        container.add(inputPanel);

        container.add(Box.createVerticalStrut(10));

        // Buttons
        JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 5, 0));
        btnPanel.setOpaque(false);
        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(() -> process(true), true));
        dec.addActionListener(e -> measureAndRun(() -> process(false), false));

        btnPanel.add(enc);
        btnPanel.add(dec);
        container.add(btnPanel);

        p.add(container);
    }

    @Override
    protected void resetFields() {
        keyField.setText("KEY");
    }
```

```
    private void process(boolean encrypt) {
        try {
            String text = getInputText().toUpperCase();
            String key = keyField.getText().toUpperCase().replaceAll("[^A-Z]", "");
            StringBuilder res = new StringBuilder();
            for (int i = 0, j = 0; i < text.length(); i++) {
                char c = text.charAt(i);
                if (c < 'A' || c > 'Z') {
                    res.append(c);
                    continue;
                }
                int m = c - 'A';
                int k = key.charAt(j % key.length()) - 'A';
                int val = encrypt ? (m + k) % 26 : (m - k + 26) % 26;
                res.append((char) (val + 'A'));
                j++;
            }
            outputArea.setText(res.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\ciphers\XORCipherPanel.java

```
package ciphers;

import components.*;
import javax.swing.*;
import java.awt.*;

public class XORCipherPanel extends BaseCipherPanel {
    JTextField keyField;

    public XORCipherPanel() {
        super("XOR Cipher");
    }

    @Override
    protected void addControls(JPanel p) {
        p.add(new JLabel("<html><font color='white'>Key (Int):</font></html>"));
        keyField = createTextField("123");
        p.add(keyField);

        JButton enc = new NeonButton("Encrypt", Theme.ACCENT_COLOR);
        JButton dec = new NeonButton("Decrypt", new Color(200, 50, 50));

        enc.addActionListener(e -> measureAndRun(this::process, true));
        dec.addActionListener(e -> measureAndRun(this::process, false));

        p.add(enc);
        p.add(dec);
    }

    @Override
    protected void resetFields() {
        keyField.setText("123");
    }

    private void process() {
        try {
            String text = getInputText();
            int key = Integer.parseInt(keyField.getText());
            StringBuilder sb = new StringBuilder();
            for (char c : text.toCharArray())
                sb.append((char) (c ^ key));
            outputArea.setText(sb.toString());
        } catch (Exception e) {
            outputArea.setText("Error: " + e.getMessage());
        }
    }
}
```

File: .\components\AnimatedSidebarButton.java

```
package components;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import javax.swing.Timer;

public class AnimatedSidebarButton extends JButton {
    private Color targetColor;
    private Color currentColor;
    private Timer animationTimer;
    private float progress = 0f;
    private boolean hovering = false;
    private boolean isSelected = false;

    private static final Color NORMAL_COLOR = Theme.TEXT_SECONDARY;
    private static final Color HOVER_COLOR = Color.WHITE;
    private static final Color SELECTED_COLOR = Theme.ACCENT_COLOR;

    public AnimatedSidebarButton(String text) {
        super(text);
        this.currentColor = NORMAL_COLOR;
        this.targetColor = NORMAL_COLOR;

        setAlignmentX(Component.LEFT_ALIGNMENT);
        setMaximumSize(new Dimension(280, 45));
        setForeground(currentColor);
        setFont(Theme.FONT_NORMAL);
        setFocusPainted(false);
        setBorder(new EmptyBorder(10, 15, 10, 15));
        setHorizontalAlignment(SwingConstants.LEFT);
        setContentAreaFilled(false);
        setCursor(new Cursor(Cursor.HAND_CURSOR));

        animationTimer = new Timer(15, e -> {
            boolean needsRepaint = false;

            // Interpolate Color
            int r = currentColor.getRed();
            int g = currentColor.getGreen();
            int b = currentColor.getBlue();
            int tr = targetColor.getRed();
            int tg = targetColor.getGreen();
            int tb = targetColor.getBlue();

            if (r != tr || g != tg || b != tb) {
                int newR = moveTowards(r, tr);
                int newG = moveTowards(g, tg);
                int newB = moveTowards(b, tb);
                currentColor = new Color(newR, newG, newB);
```

```
            setForeground(currentColor);
            needsRepaint = true;
        }

        // Interpolate Progress (for background/indicator)
        float targetProgress = (isSelected || hovering) ? 1.0f : 0.0f;
        if (Math.abs(progress - targetProgress) > 0.01f) {
            progress += (targetProgress - progress) * 0.2f;
            needsRepaint = true;
        } else {
            progress = targetProgress;
        }

        if (needsRepaint) {
            repaint();
        } else {
            ((Timer)e.getSource()).stop();
        }
    });

    addMouseListener(new MouseAdapter() {
        @Override
        public void mouseEntered(MouseEvent e) {
            hovering = true;
            updateTargetState();
        }

        @Override
        public void mouseExited(MouseEvent e) {
            hovering = false;
            updateTargetState();
        }
    });
}

private int moveTowards(int current, int target) {
    int step = 15; // Speed of color transition
    if (Math.abs(target - current) <= step) return target;
    return current + (target > current ? step : -step);
}

public void setSelected(boolean selected) {
    this.isSelected = selected;
    updateTargetState();
}

private void updateTargetState() {
    if (isSelected) {
        targetColor = SELECTED_COLOR;
        setFont(Theme.FONT_NORMAL.deriveFont(Font.BOLD)); // Make bold when selected
    } else if (hovering) {
        targetColor = HOVER_COLOR;
        setFont(Theme.FONT_NORMAL);
    } else {
```

```
            targetColor = NORMAL_COLOR;
            setFont(Theme.FONT_NORMAL);
        }

        if (!animationTimer.isRunning()) {
            animationTimer.start();
        }
    }

    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g.create();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        // Draw subtle background on hover/select
        if (progress > 0.01f) {
                            g2.setColor(new  Color(Theme.ACCENT_COLOR.getRed(),  Theme.ACCENT_COLOR.getGreen(),
Theme.ACCENT_COLOR.getBlue(), (int)(30 * progress)));
            g2.fillRoundRect(5, 2, getWidth() - 10, getHeight() - 4, 10, 10);

            // Draw a small indicator bar on the left
                            g2.setColor(new  Color(Theme.ACCENT_COLOR.getRed(),  Theme.ACCENT_COLOR.getGreen(),
Theme.ACCENT_COLOR.getBlue(), (int)(255 * progress)));
            int barHeight = (int)(20 * progress);
            int barY = (getHeight() - barHeight) / 2;
            g2.fillRoundRect(5, barY, 3, barHeight, 3, 3);
        }

        super.paintComponent(g2);
        g2.dispose();
    }
}
```

File: .\components\CardPanel.java

```
package components;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;

public class CardPanel extends JPanel {
    public CardPanel() {
        setBackground(Theme.CARD_BG);
        setOpaque(false); // Enable transparency for rounded corners
        setBorder(new EmptyBorder(20, 20, 20, 20)); // Padding
    }

    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g.create();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        g2.setColor(getBackground());
        g2.fillRoundRect(0, 0, getWidth(), getHeight(), 15, 15);

        // Subtle Border
        g2.setColor(new Color(255, 255, 255, 20));
        g2.drawRoundRect(0, 0, getWidth() - 1, getHeight() - 1, 15, 15);

        g2.dispose();
        super.paintComponent(g);
    }
}
```

File: .\components\ContentPanel.java

```
package components;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.util.HashMap;
import java.util.Map;

public class ContentPanel extends JPanel {
    private JPanel mainContainer;
    private Component currentComponent;
    private BufferedImage transitionImage;
    private float alpha = 0.0f;
    private Timer transitionTimer;
    private boolean isAnimating = false;

    public ContentPanel() {
        setLayout(new BorderLayout());
        setBackground(Theme.MAIN_BG);

        mainContainer = new JPanel(new BorderLayout());
        mainContainer.setBackground(Theme.MAIN_BG);
        add(mainContainer, BorderLayout.CENTER);

        transitionTimer = new Timer(15, e -> {
            alpha -= 0.08f; // Speed of fade
            if (alpha <= 0) {
                alpha = 0;
                isAnimating = false;
                transitionImage = null; // Free memory
                ((Timer) e.getSource()).stop();
                repaint();
            } else {
                repaint();
            }
        });
    }

    public void showPanel(JComponent panel) {
        if (currentComponent == panel)
            return;

        // If we are currently showing something, capture it
        if (currentComponent != null && getWidth() > 0 && getHeight() > 0) {
            transitionImage = new BufferedImage(getWidth(), getHeight(), BufferedImage.TYPE_INT_ARGB);
            Graphics2D g2 = transitionImage.createGraphics();
            paint(g2); // Paint everything (including current animation state if any)
            g2.dispose();

            alpha = 1.0f;
            isAnimating = true;
            transitionTimer.restart();
```

```
        }

        mainContainer.removeAll();
        mainContainer.add(panel, BorderLayout.CENTER);
        currentComponent = panel;

        mainContainer.revalidate();
        mainContainer.repaint();
    }

    // Helper for CardLayout-style usage if needed, though simpler to use showPanel
    // We will just expose showPanel for direct usage.

    @Override
    public void paint(Graphics g) {
        super.paint(g);
    }

    @Override
    protected void paintChildren(Graphics g) {
        super.paintChildren(g);

        // Draw the transition image on top
        if (isAnimating && transitionImage != null) {
            Graphics2D g2 = (Graphics2D) g.create();
            g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, alpha));
            g2.drawImage(transitionImage, 0, 0, null);
            g2.dispose();
        }
    }
}
```

File: .\components\CyberBackground.java

```
package components;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class CyberBackground extends JPanel {
    private List<Particle> particles = new ArrayList<>();
    private Timer animTimer;
    private Random random = new Random();

    public CyberBackground() {
        setOpaque(false);
        setBackground(new Color(0, 0, 0, 0)); // Transparent

        // Initialize particles
        for (int i = 0; i < 50; i++) {
            particles.add(new Particle());
        }

        animTimer = new Timer(30, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                for (Particle p : particles) {
                    p.update();
                }
                repaint();
            }
        });
        animTimer.start();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        int w = getWidth();
        int h = getHeight();

        // Draw connections
                        g2.setColor(new   Color(Theme.ACCENT_COLOR.getRed(),   Theme.ACCENT_COLOR.getGreen(),
Theme.ACCENT_COLOR.getBlue(),
                30));
        g2.setStroke(new BasicStroke(1f));

        for (int i = 0; i < particles.size(); i++) {
```

```
            Particle p1 = particles.get(i);
            for (int j = i + 1; j < particles.size(); j++) {
                Particle p2 = particles.get(j);
                double dist = Math.hypot(p1.x - p2.x, p1.y - p2.y);
                if (dist < 150) {
                    int alpha = (int) ((150 - dist) / 150.0 * 40);
                    g2.setColor(new Color(Theme.ACCENT_COLOR.getRed(), Theme.ACCENT_COLOR.getGreen(),
                            Theme.ACCENT_COLOR.getBlue(), alpha));
                    g2.drawLine((int) p1.x, (int) p1.y, (int) p2.x, (int) p2.y);
                }
            }
        }

        // Draw particles
        for (Particle p : particles) {
            g2.setColor(new Color(Theme.ACCENT_COLOR.getRed(), Theme.ACCENT_COLOR.getGreen(),
                    Theme.ACCENT_COLOR.getBlue(), (int) (p.opacity * 255)));
            g2.fillOval((int) p.x - 2, (int) p.y - 2, 4, 4);
        }
    }

    private class Particle {
        double x, y;
        double vx, vy;
        float opacity;

        public Particle() {
            reset(true);
        }

        void reset(boolean randomY) {
            x = random.nextInt(Math.max(1, getWidth() > 0 ? getWidth() : 800));
            y = randomY ? random.nextInt(Math.max(1, getHeight() > 0 ? getHeight() : 600)) : getHeight() + 10;
            vx = (random.nextDouble() - 0.5) * 1.5;
            vy = -(random.nextDouble() * 1.0 + 0.2); // Move upwards
            opacity = random.nextFloat() * 0.5f + 0.1f;
        }

        void update() {
            x += vx;
            y += vy;

            if (y < -10 || x < -10 || x > getWidth() + 10) {
                reset(false);
                y = getHeight() + 10;
            }
        }
    }
}
```

File: .\components\DarkScrollPane.java

```
package components;

import javax.swing.*;
import javax.swing.plaf.basic.BasicScrollBarUI;
import java.awt.*;

public class DarkScrollPane extends JScrollPane {
    public DarkScrollPane(Component view) {
        super(view);
        setBorder(null);
        getViewport().setOpaque(true);
        getViewport().setBackground(Theme.INPUT_BG);
        setOpaque(true);
        setBackground(Theme.INPUT_BG);
        getVerticalScrollBar().setUI(new BasicScrollBarUI() {
            protected void configureScrollBarColors() {
                this.thumbColor = new Color(80, 80, 80);
                this.trackColor = Theme.INPUT_BG;
            }

            @Override
            protected JButton createDecreaseButton(int orientation) {
                return createZeroButton();
            }

            @Override
            protected JButton createIncreaseButton(int orientation) {
                return createZeroButton();
            }
        });
    }

    private JButton createZeroButton() {
        JButton jbutton = new JButton();
        jbutton.setPreferredSize(new Dimension(0, 0));
        return jbutton;
    }
}
```

File: .\components\NeonButton.java

```
package components;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;

public class NeonButton extends JButton {
    private Color baseColor;
    private boolean isHovered = false;

    public NeonButton(String text, Color color) {
        super(text);
        this.baseColor = color;
        setFont(new Font("Segoe UI", Font.BOLD, 13));
        setForeground(Color.WHITE);
        setBackground(color);
        setFocusPainted(false);
        setBorder(new EmptyBorder(10, 25, 10, 25));
        setCursor(new Cursor(Cursor.HAND_CURSOR));
        setContentAreaFilled(false);
        setOpaque(false);

        addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseEntered(java.awt.event.MouseEvent evt) {
                isHovered = true;
                repaint();
            }

            public void mouseExited(java.awt.event.MouseEvent evt) {
                isHovered = false;
                repaint();
            }
        });
    }

    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g.create();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        if (isHovered) {
            g2.setColor(baseColor.brighter());
            // Glow effect
            g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.5f));
            g2.fillRoundRect(0, 0, getWidth(), getHeight(), 20, 20);
            g2.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 1.0f));
            g2.setColor(baseColor);
            g2.drawRoundRect(0, 0, getWidth() - 1, getHeight() - 1, 20, 20);
        } else {
            g2.setColor(baseColor.darker());
            g2.fillRoundRect(0, 0, getWidth(), getHeight(), 20, 20);
        }
```

```
        // Text
        super.paintComponent(g2);
        g2.dispose();
    }
}
```

File: .\components\PerformanceGraph.java

```
package components;

import javax.swing.*;
import java.awt.*;

public class PerformanceGraph extends JPanel {
    private long encTime = 0;
    private long decTime = 0;

    public PerformanceGraph() {
        setOpaque(true);
        setBackground(Theme.CARD_BG); // Match Card BG
        setPreferredSize(new Dimension(300, 200));
    }

    public void setEncTime(long t) {
        this.encTime = t;
        repaint();
    }

    public void setDecTime(long t) {
        this.decTime = t;
        repaint();
    }

    public void reset() {
        this.encTime = 0;
        this.decTime = 0;
        repaint();
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g); // Paint background first
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

        int w = getWidth();
        int h = getHeight();
        int barWidth = 60;
        int spacing = 80;
        int topMargin = 50;
        int maxBarHeight = h - topMargin - 40;
        int startX = (w - (2 * barWidth + spacing)) / 2;

        // Draw Grid
        g2.setColor(new Color(60, 60, 60));
        g2.drawLine(20, h - 30, w - 20, h - 30); // X-Axis

        // Summary Text
        g2.setFont(new Font("Segoe UI", Font.BOLD, 14));
```

```
        g2.setColor(Theme.GRAPH_ENC_COLOR);
        g2.drawString("Encryption", startX, 25);
        g2.setFont(new Font("Consolas", Font.PLAIN, 12));
        g2.drawString(formatTime(encTime), startX, 45);

        g2.setFont(new Font("Segoe UI", Font.BOLD, 14));
        g2.setColor(Theme.GRAPH_DEC_COLOR);
        g2.drawString("Decryption", startX + barWidth + spacing, 25);
        g2.setFont(new Font("Consolas", Font.PLAIN, 12));
        g2.drawString(formatTime(decTime), startX + barWidth + spacing, 45);

        long maxVal = Math.max(encTime, decTime);
        if (maxVal == 0)
            maxVal = 1;

        // Enc Bar
        int encHeight = (int) ((double) encTime / maxVal * maxBarHeight);
        if (encHeight < 5 && encTime > 0)
            encHeight = 5;
        g2.setColor(Theme.GRAPH_ENC_COLOR);
        g2.fillRoundRect(startX, h - 30 - encHeight, barWidth, encHeight, 10, 10);

        // Dec Bar
        int decHeight = (int) ((double) decTime / maxVal * maxBarHeight);
        if (decHeight < 5 && decTime > 0)
            decHeight = 5;
        g2.setColor(Theme.GRAPH_DEC_COLOR);
        g2.fillRoundRect(startX + barWidth + spacing, h - 30 - decHeight, barWidth, decHeight, 10, 10);
    }

    private String formatTime(long nanos) {
        if (nanos == 0)
            return "0 ms";
        if (nanos < 10000)
            return nanos + " ns";
        return String.format("%.3f ms", nanos / 1_000_000.0);
    }
}
```

File: .\components\SplashScreen.java

```
package components;

import javax.swing.*;
import java.awt.*;

public class SplashScreen extends JWindow {

    public SplashScreen() {
        setSize(500, 300);
        setLocationRelativeTo(null);
        setBackground(new Color(0, 0, 0, 0)); // Translucent window support

        JPanel content = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                Graphics2D g2 = (Graphics2D) g;
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

                // Background with rounded corners
                g2.setColor(Theme.MAIN_BG);
                g2.fillRoundRect(0, 0, getWidth(), getHeight(), 30, 30);

                // Border
                g2.setColor(Theme.ACCENT_COLOR);
                g2.setStroke(new BasicStroke(2));
                g2.drawRoundRect(1, 1, getWidth() - 2, getHeight() - 2, 30, 30);
            }
        };
        content.setLayout(null);
        content.setOpaque(false);
        setContentPane(content);

        // Logo / Text
        JLabel title = new JLabel("CRYPTO SUITE");
        title.setFont(Theme.FONT_TITLE.deriveFont(32f));
        title.setForeground(Color.WHITE);
        title.setBounds(0, 80, 500, 40);
        title.setHorizontalAlignment(SwingConstants.CENTER);
        content.add(title);

        JLabel subtitle = new JLabel("Initializing Secure Modules...");
        subtitle.setFont(Theme.FONT_NORMAL);
        subtitle.setForeground(Theme.TEXT_SECONDARY);
        subtitle.setBounds(0, 130, 500, 20);
        subtitle.setHorizontalAlignment(SwingConstants.CENTER);
        content.add(subtitle);

        // Custom Progress Bar
        JProgressBar bar = new JProgressBar();
        bar.setBounds(100, 180, 300, 6);
        bar.setBackground(new Color(40, 40, 40));
        bar.setForeground(Theme.ACCENT_COLOR);
```

```
        bar.setBorderPainted(false);
        content.add(bar);

        // Simulate Loading in a thread
        new Thread(() -> {
            try {
                for (int i = 0; i <= 100; i++) {
                    bar.setValue(i);
                    Thread.sleep(25); // 2.5 seconds load
                }
                SwingUtilities.invokeLater(() -> {
                    dispose();
                    // Callback to main would be here, but we'll handle this in main() logic
                });
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }).start();
    }

    public void showSplash() {
        setVisible(true);
        try {
            // Block partially to simulate wait, or just let the internal thread handle it?
            // Better: Blocking method for main to call
            Thread.sleep(2800);
        } catch (Exception e) {
        }
    }
}
```

File: .\components\Theme.java

```
package components;

import java.awt.Color;
import java.awt.Font;

public class Theme {
    public static final Color MAIN_BG = new Color(18, 18, 18); // Deep Dark
    public static final Color SIDEBAR_BG = new Color(30, 30, 30); // Dark Sidebar
    public static final Color CARD_BG = new Color(40, 40, 40); // Card Background
    public static final Color ACCENT_COLOR = new Color(0, 229, 255); // Neon Cyan
    public static final Color TEXT_PRIMARY = new Color(255, 255, 255);
    public static final Color TEXT_SECONDARY = new Color(176, 176, 176);
    public static final Color INPUT_BG = new Color(45, 45, 45); // Input Fields

    public static final Color GRAPH_ENC_COLOR = new Color(0, 255, 127); // Spring Green
    public static final Color GRAPH_DEC_COLOR = new Color(255, 69, 0); // Red Orange

    public static final Font FONT_TITLE = new Font("Segoe UI", Font.BOLD, 28);
    public static final Font FONT_HEADER = new Font("Segoe UI", Font.BOLD, 16);
    public static final Font FONT_NORMAL = new Font("Segoe UI", Font.PLAIN, 14);
    public static final Font FONT_MONO = new Font("Consolas", Font.PLAIN, 14);
}
```