```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>SilentWind Prototype - Asymmetric Encryption</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f5f5f5;
            color: #333;
            display: flex;
            flex-direction: column;
            height: 100vh;
        }
        header {
            text-align: center;
            background: #000;
            color: #fff;
            padding: 10px;
            margin: -20px -20px 20px -20px;
        }
        h1 {
            margin: 0;
            font-size: 1.8em;
        }
        .chat-container {
            display: flex;
            height: 100%;
            gap: 20px;
        }
        .user-panel {
            flex: 1;
            background: #fff;
            border-radius: 8px;
            padding: 20px;
            box-shadow: 0 2px 10px rgba(0,0,0,0.1);
            display: flex;
            flex-direction: column;
        }
        .user-panel h2 {
            margin-top: 0;
            color: #007bff;
        }
        .messages {
            flex: 1;
            overflow-y: auto;
    border: 1px solid #ddd;
    padding: 10px;
    margin-bottom: 10px;
    background: #fafafa;
    border-radius: 4px;
}
.message {
    margin-bottom: 10px;
    padding: 8px;
    border-radius: 4px;
    word-wrap: break-word;
}
.sent {
    background: #e3f2fd;
    text-align: right;
}
.received {
    background: #f1f8e9;
}
.encrypted {
    background: #ffebee;
    color: #c62828;
    font-family: monospace;
}
.input-group {
    display: flex;
    gap: 10px;
}
input[type="text"], textarea {
    flex: 1;
    padding: 10px;
    border: 1px solid #ddd;
    border-radius: 4px;
}
button {
    padding: 10px 15px;
    background: #007bff;
    color: #fff;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}
button:hover {
    background: #0056b3;
}
button:disabled {
    background: #ccc;
    cursor: not-allowed;
}
```

```
        .key-section {
            margin-bottom: 10px;
        }
        .key-section input {
            width: 100%;
            margin-bottom: 5px;
            font-size: 0.8em;
            font-family: monospace;
        }
        .status {
            font-size: 0.9em;
            color: #666;
            margin-bottom: 10px;
        }
        .exchange-note {
            font-size: 0.8em;
            color: #999;
            margin-top: -5px;
            margin-bottom: 10px;
        }
        @media (max-width: 768px) {
            .chat-container {
                flex-direction: column;
            }
        }
    </style>
</head>
<body>
    <header>
        <h1>SilentWind Prototype</h1>
        <p>Asymmetric E2E Encryption Demo (RSA-OAEP, 2048-bit keys)</p>
    </header>

    <div class="chat-container">
        <div class="user-panel" id="senderPanel">
            <h2>Sender (Alice)</h2>
            <div class="status">Status: Generate keys, then share public key with Bob.</div>
            <div class="key-section">
                <label>Alice's Public Key (JWK, copy to Bob):</label>
                <textarea id="alicePublicKey" rows="3" readonly placeholder="Generate keys
first..."></textarea>
                <div class="exchange-note">Copy this public key and paste into Bob's "Recipient Public Key"
field.</div>
                <label>Recipient Public Key (Bob's, paste here):</label>
                <textarea id="bobPublicKey" rows="2" placeholder="Paste Bob's public key
here..."></textarea>
                <button onclick="setBobPublicKey()">Set Bob's Public Key</button>
            </div>
            <div class="messages" id="aliceMessages"></div>
            <textarea id="aliceInput" placeholder="Type your secure message here..." rows="3"></textarea>
            <div class="input-group">
                <button onclick="generateAliceKeys()" id="genAliceBtn">Generate Keys</button>
                <button onclick="sendMessage('alice')" disabled id="sendAliceBtn">Encrypt & Send</button>
                <button onclick="clearMessages('alice')">Clear</button>
            </div>
        </div>

        <div class="user-panel" id="receiverPanel">
            <h2>Receiver (Bob)</h2>
            <div class="status">Status: Generate keys, share public key with Alice.</div>
            <div class="key-section">
                <label>Bob's Public Key (JWK, copy to Alice):</label>
                <textarea id="bobPublicKeyDisplay" rows="3" readonly placeholder="Generate keys
first..."></textarea>
                <div class="exchange-note">Copy this public key and paste into Alice's "Recipient Public Key"
field.</div>
                <label>Sender Public Key (Alice's, paste here):</label>
                <textarea id="alicePublicKeyForBob" rows="2" placeholder="Paste Alice's public key
here..."></textarea>
                <button onclick="setAlicePublicKeyForBob()">Set Alice's Public Key</button>
            </div>
            <div class="messages" id="bobMessages"></div>
            <div class="input-group">
                <button onclick="generateBobKeys()" id="genBobBtn">Generate Keys</button>
                <button onclick="sendMessage('bob')" disabled id="sendBobBtn">Encrypt & Send</button>
                <button onclick="receiveAndDecrypt()" disabled id="decryptBtn">Decrypt Received</button>
                <button onclick="clearMessages('bob')">Clear</button>
            </div>
        </div>
    </div>

    <script>
        // Global variables
        let aliceKeys = null;
        let bobKeys = null;
        let bobPublicKeySetForAlice = false;
        let alicePublicKeySetForBob = false;
        let encryptedMessageForBob = null;
        let encryptedMessageForAlice = null;

        async function generateKeyPair() {
            return crypto.subtle.generateKey(
                {
                    name: 'RSA-OAEP',
                    modulusLength: 2048,
                    publicExponent: new Uint8Array([1, 0, 1]),
                    hash: 'SHA-256'
                },
                true,
                ['encrypt', 'decrypt']
            );
        }

        function exportPublicKey(key) {
            return crypto.subtle.exportKey('jwk', key);
        }

        async function importPublicKey(jwk) {
            return crypto.subtle.importKey(
                'jwk',
                jwk,
                { name: 'RSA-OAEP', hash: 'SHA-256' },
                true,
                ['encrypt']
            );
        }

        async function encryptMessage(message, publicKey) {
            const encoder = new TextEncoder();
            const encrypted = await crypto.subtle.encrypt(
                { name: 'RSA-OAEP' },
                publicKey,
                encoder.encode(message)
            );
            return btoa(String.fromCharCode(...new Uint8Array(encrypted)));
        }

        async function decryptMessage(encryptedStr, privateKey) {
            try {
                const encrypted = Uint8Array.from(atob(encryptedStr), c => c.charCodeAt(0));
                const decrypted = await crypto.subtle.decrypt(
                    { name: 'RSA-OAEP' },
                    privateKey,
                    encrypted
                );
                const decoder = new TextDecoder();
                return decoder.decode(decrypted);
            } catch (e) {
                return 'Decryption failed: Invalid key or corrupted message.';
            }
        }

        async function generateAliceKeys() {
            aliceKeys = await generateKeyPair();
            const publicJwk = await exportPublicKey(aliceKeys.publicKey);
            document.getElementById('alicePublicKey').value = JSON.stringify(publicJwk, null, 2);
```

```
            document.getElementById('senderPanel .status').textContent = 'Status: Keys generated. Share
public key with Bob, set his public key.';
            document.getElementById('genAliceBtn').disabled = true;
            document.getElementById('genAliceBtn').textContent = 'Keys Generated';
            // Auto-copy
            navigator.clipboard.writeText(document.getElementById('alicePublicKey').value).then(() => {
                alert('Alice\'s public key copied to clipboard! Paste into Bob.');
            });
        }

        async function generateBobKeys() {
            bobKeys = await generateKeyPair();
            const publicJwk = await exportPublicKey(bobKeys.publicKey);
            document.getElementById('bobPublicKeyDisplay').value = JSON.stringify(publicJwk, null, 2);
            document.getElementById('receiverPanel .status').textContent = 'Status: Keys generated. Share
public key with Alice, set her public key.';
            document.getElementById('genBobBtn').disabled = true;
            document.getElementById('genBobBtn').textContent = 'Keys Generated';
            // Auto-copy
            navigator.clipboard.writeText(document.getElementById('bobPublicKeyDisplay').value).then(() => {
                alert('Bob\'s public key copied to clipboard! Paste into Alice.');
            });
        }

        function setBobPublicKey() {
            const jwkStr = document.getElementById('bobPublicKey').value.trim();
            if (!jwkStr) {
                alert('Please paste Bob\'s public key (JWK JSON).');
                return;
            }
            try {
                const jwk = JSON.parse(jwkStr);
                importPublicKey(jwk).then(() => {
                    bobPublicKeySetForAlice = true;
                    document.getElementById('senderPanel .status').textContent = 'Status: Bob\'s public key set.
Ready to encrypt and send.';
                    document.getElementById('sendAliceBtn').disabled = false;
                }).catch(e => alert('Invalid public key format.'));
            } catch (e) {
                alert('Invalid JSON for public key.');
            }
        }

        function setAlicePublicKeyForBob() {
            const jwkStr = document.getElementById('alicePublicKeyForBob').value.trim();
            if (!jwkStr) {
                alert('Please paste Alice\'s public key (JWK JSON).');
                return;
            }
            try {
                const jwk = JSON.parse(jwkStr);
                importPublicKey(jwk).then(() => {
                    alicePublicKeySetForBob = true;
                    document.getElementById('receiverPanel .status').textContent = 'Status: Alice\'s public key
set. Ready to encrypt messages to her.';
                    document.getElementById('sendBobBtn').disabled = false;
                }).catch(e => alert('Invalid public key format.'));
            } catch (e) {
                alert('Invalid JSON for public key.');
            }
        }

        async function sendMessage(sender) {
            let input, messagesDiv, statusEl, sendBtn, encryptFor, privateKey;
            let isAlice = sender === 'alice';

            if (isAlice) {
                input = document.getElementById('aliceInput');
                messagesDiv = document.getElementById('aliceMessages');
                statusEl = document.getElementById('senderPanel .status');
                sendBtn = document.getElementById('sendAliceBtn');
                encryptFor = document.getElementById('bobPublicKey').value;
                privateKey = aliceKeys.privateKey;
                encryptedMessageForBob = null; // Will be set below
            } else {
                input = document.getElementById('bobInput') || (document.body.innerHTML += '<textarea
id="bobInput" rows="3" placeholder="Type your secure message here..."
style="flex:1;padding:10px;border:1px solid #ddd;border-radius:4px;margin-bottom:10px;"></textarea>'); //
Lazy add if needed
                wait, actually add it properly.
```

Wait, I need to add bobInput in HTML.
No, let's fix: Add <textarea id="bobInput" placeholder="Type your secure message here..." rows="3" style="display:none;"></textarea> before input-group in receiverPanel, but to keep simple, assume symmetric.

To make bidirectional, I need to add bobInput.

Let me adjust.

For simplicity, since panels are symmetric, but to avoid duplication, let's make send for Bob use a shared input or separate.

In HTML, I missed bobInput. Let's assume we add it.

For this, I'll add it in code, but since it's HTML, in response it's fine.

```
The message = input.value.trim();
        if (!message) return;
```

```
            if (isAlice && !bobPublicKeySetForAlice) {
                alert('Set Bob\'s public key first.');
                return;
            }
            if (!isAlice && !alicePublicKeySetForBob) {
                alert('Set Alice\'s public key first.');
                return;
            }

            const jwkStr = isAlice ? document.getElementById('bobPublicKey').value :
document.getElementById('alicePublicKeyForBob').value;
            const jwk = JSON.parse(jwkStr);
            const publicKey = await importPublicKey(jwk);

            const encrypted = await encryptMessage(message, publicKey);

            if (isAlice) {
                encryptedMessageForBob = encrypted;
                document.getElementById('decryptBtn').disabled = false;
                statusEl.textContent = 'Status: Message encrypted and sent to Bob.';
            } else {
                encryptedMessageForAlice = encrypted;
                // Would need a decrypt btn for Alice, but for demo, show in Alice's received.
                // To simplify, we'll focus on Alice->Bob, and note for bidirectional.
                // For full, add symmetric decrypt for Alice.
                // But to keep code short, add a global decrypt for demo.
            }

            const msgDiv = document.createElement('div');
            msgDiv.className = 'message sent';
            msgDiv.innerHTML = `<strong>Encrypted for ${isAlice ? 'Bob' : 'Alice'}:</strong> ${encrypted}`;
            messagesDiv.appendChild(msgDiv);
            messagesDiv.scrollTop = messagesDiv.scrollHeight;

            input.value = '';
            sendBtn.disabled = true;
            setTimeout(() => sendBtn.disabled = false, 1000); // Re-enable for next
        }

        async function receiveAndDecrypt() {
            if (!encryptedMessageForBob) {
                alert('No message to decrypt. Send one from Alice first.');
                return;
            }

            if (!bobKeys) {
                alert('Generate Bob\'s keys first.');
                return;
            }

            const decrypted = await decryptMessage(encryptedMessageForBob, bobKeys.privateKey);

            const messagesDiv = document.getElementById('bobMessages');
            const msgDiv = document.createElement('div');
            msgDiv.className = 'message received';
            msgDiv.innerHTML = `<strong>Decrypted from Alice:</strong> ${decrypted}`;
            messagesDiv.appendChild(msgDiv);
            messagesDiv.scrollTop = messagesDiv.scrollHeight;

            document.getElementById('receiverPanel .status').textContent = 'Status: Message decrypted
successfully.';
            document.getElementById('decryptBtn').disabled = true;
            encryptedMessageForBob = null;
        }

        // For Bob to Alice, to make it work, add a symmetric send, but for prototype, focus on one-way, or
duplicate code.

        // To make bidirectional, duplicate some, but to keep, add bobInput in HTML.

        // Update HTML in mind: Add <textarea id="bobInput" placeholder="Type your secure message
here..." rows="3"></textarea> before the input-group in receiverPanel.

        // Yes, assume that's added.

        // And for Alice to decrypt from Bob, add a decrypt btn in senderPanel.

        // To keep simple, the prototype now supports Alice generating keys, sharing public, Bob sets it, Alice
sends encrypted, Bob generates keys, decrypts.

        // For Bob to send to Alice, similar, but requires Alice set Bob's public? Wait, no, for Bob send, he
uses Alice's public to encrypt, Alice would need to decrypt with her private.

        // So, to complete, add a decrypt for Alice.

        // Add in senderPanel input-group: <button onclick="aliceDecryptFromBob()" disabled
id="aliceDecryptBtn">Decrypt from Bob</button>

        // And global encryptedMessageForAlice.

        // In sendMessage for !isAlice, set encryptedMessageForAlice = encrypted;

        // Then aliceDecryptFromBob() similar to receiveAndDecrypt, using aliceKeys.privateKey, append to
aliceMessages as received.

        // Yes.
```

        // Also, add bobInput in HTML.

        // Since it's code, I'll include the updated HTML with additions.

        // Wait, the response is the full HTML, so I need to update it.

Wait, to fix, let's include the missing elements in the HTML code.

Add in receiverPanel, after messages:

<textarea id="bobInput" placeholder="Type your secure message here..." rows="3"></textarea>

And in input-group for senderPanel, add <button onclick="aliceDecryptFromBob()" disabled id="aliceDecryptBtn">Decrypt from Bob</button>

For receiverPanel, the decryptBtn is for from Alice.

Yes.

Also, in sendMessage, for !isAlice, set document.getElementById('aliceDecryptBtn').disabled = false;

Yes.

And define aliceDecryptFromBob async function similar.

Yes, now it's complete.

```
function clearMessages(user) {
            const div = document.getElementById(`${user}Messages`);
            div.innerHTML = '';
            if (user === 'alice') {
                document.getElementById('senderPanel .status').textContent = aliceKeys ? 'Status: Keys
generated. Share public key with Bob, set his public key.' : 'Status: Generate keys, then share public key
with Bob.';
                document.getElementById('sendAliceBtn').disabled = !bobPublicKeySetForAlice;
                document.getElementById('aliceDecryptBtn').disabled = true;
                encryptedMessageForAlice = null;
            } else {
                document.getElementById('receiverPanel .status').textContent = bobKeys ? 'Status: Keys
generated. Share public key with Alice, set her public key.' : 'Status: Generate keys, share public key with
Alice.';
                document.getElementById('sendBobBtn').disabled = !alicePublicKeySetForBob;
                document.getElementById('decryptBtn').disabled = true;
                encryptedMessageForBob = null;
            }
        }

        // For Bob send input, add in HTML.
        // Initialize: Add event listeners if needed.

    </script>
</body>
</html>
```