

```

<!DOCTYPE html>
<html>
<head>
<title>HMAC Signature Generator Fiddle (Plain JS)</title>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<!-- Tailwind CSS CDN -->
<script src="https://cdn.tailwindcss.com"></script>
<!-- Inter Font -->
<link
href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;700;
800&display=swap" rel="stylesheet">
<!-- UUID CDN (for generating nonces) -->
<script
src="https://unpkg.com/uuid@8.3.2/dist/umd/uuid.min.js"></script>

<style type="text/tailwindcss">
  body {
    font-family: 'Inter', sans-serif;
  }
  input, textarea, select {
    @apply focus:ring-2 focus:ring-blue-500 focus:border-blue-500
transition duration-200 rounded-md;
  }
</style>
</head>
<body>


<div id="app" class="min-h-screen bg-gradient-to-br from-gray-50
to-blue-100 p-4 sm:p-6 text-gray-800 flex items-center
justify-center">
  <div class="bg-white p-6 sm:p-8 rounded-xl shadow-2xl w-full
max-w-5xl border border-blue-200">
    <h1 class="text-3xl sm:text-4xl font-extrabold text-center
text-blue-800 mb-6 flex items-center justify-center gap-3">
      🔒 HMAC Signature Generator Fiddle (Plain JS)
    </h1>
    <p class="text-center text-gray-600 mb-8 max-w-2xl mx-auto">
      Generate client-side HMAC signatures for your Spring Boot API.
      Ensure your inputs match the server's configuration!
    </p>

    <div id="errorMessage" class="hidden bg-red-100 border
border-red-400 text-red-700 px-4 py-3 rounded-md mb-6 items-center
gap-2">
      ⚠️ <span id="errorText" class="font-medium"></span>
    </div>

    <div class="grid grid-cols-1 lg:grid-cols-2 gap-8">

```

```

    <!-- Input Section -->
    <div class="bg-blue-50 p-6 rounded-lg border border-blue-200
shadow-inner">
        <h2 class="text-xl sm:text-2xl font-bold text-blue-700 mb-4
flex items-center gap-2">
             Input Parameters
        </h2>
        <div class="space-y-4">
            <div>
                <label for="accessKey" class="block text-sm font-medium
text-gray-700 mb-1">
                    Access Key (Client ID)
                </label>
                <input type="text" id="accessKey" class="w-full p-2 border
border-blue-300" value="my-client-id">
            </div>
            <div>
                <label for="secretKey" class="block text-sm font-medium
text-gray-700 mb-1">
                    Secret Key (Shared Secret)
                </label>
                <input type="text" id="secretKey" class="w-full p-2 border
border-blue-300"
value="your-super-secret-key-that-should-be-long-and-random">
            </div>
            <div>
                <label for="httpMethod" class="block text-sm font-medium
text-gray-700 mb-1">
                    HTTP Method
                </label>
                <select id="httpMethod" class="w-full p-2 border
border-blue-300 bg-white">
                    <option value="GET">GET</option>
                    <option value="POST" selected>POST</option>
                    <option value="PUT">PUT</option>
                    <option value="PATCH">PATCH</option>
                    <option value="DELETE">DELETE</option>
                </select>
            </div>
            <div>
                <label for="requestUri" class="block text-sm font-medium
text-gray-700 mb-1">
                    Request URI (e.g., /userprofile/api/data)
                </label>
                <input type="text" id="requestUri" class="w-full p-2
border border-blue-300" value="/userprofile/api/submit">
            </div>
            <div id="requestBodyContainer">
                <label for="requestBody" class="block text-sm font-medium

```





```

text-gray-700 mb-1">
    Request Body (JSON)
</label>
<textarea id="requestBody" rows="6" class="w-full p-2
border border-blue-300 font-mono text-sm">{"message": "Hello from
Fiddle", "value": 123}</textarea>
<p class="text-xs text-gray-500 mt-1">
    This JSON will be canonicalized (minified) before
hashing.
</p>
</div>
<div>
<label for="delimiter" class="block text-sm font-medium
text-gray-700 mb-1">
    Delimiter Character (Must match server)
</label>
<input type="text" id="delimiter" class="w-full p-2 border
border-blue-300" value="|" maxlength="1">
</div>
<button id="generateBtn" class="w-full bg-indigo-600
hover:bg-indigo-700 text-white font-bold py-3 px-4 rounded-lg
shadow-lg flex items-center justify-center gap-2">
    ➡ Generate HMAC Headers
</button>
</div>
</div>

<!-- Output Section -->
<div class="bg-green-50 p-6 rounded-lg border border-green-200
shadow-inner">
<h2 class="text-xl sm:text-2xl font-bold text-green-700 mb-4
flex items-center gap-2">
    ✓ Generated Output
</h2>
<div class="space-y-4">
<div>
<h3 class="text-lg font-semibold text-gray-700 mb-2 flex
items-center gap-1">
    🕒 X-Timestamp
</h3>
<div class="relative bg-white p-3 rounded-md border
border-gray-200 text-sm font-mono break-all flex items-center
justify-between">
<span id="outputTimestamp">N/A</span>
<button class="ml-2 px-2 py-1 bg-gray-200 rounded
text-xs copy-btn" data-target="outputTimestamp">Copy</button>
</div>
</div>
</div>

```

```

        <h3 class="text-lg font-semibold text-gray-700 mb-2 flex
items-center gap-1">
         X-Nonce
        </h3>
        <div class="relative bg-white p-3 rounded-md border
border-gray-200 text-sm font-mono break-all flex items-center
justify-between">
            <span id="outputNonce">N/A</span>
            <button class="ml-2 px-2 py-1 bg-gray-200 rounded
text-xs copy-btn" data-target="outputNonce">Copy</button>
        </div>
    </div>
    <div>
        <h3 class="text-lg font-semibold text-gray-700 mb-2 flex
items-center gap-1">
         X-HMAC-Signature
        </h3>
        <div class="relative bg-white p-3 rounded-md border
border-gray-200 text-sm font-mono break-all flex items-center
justify-between">
            <span id="outputSignature">N/A</span>
            <button class="ml-2 px-2 py-1 bg-gray-200 rounded
text-xs copy-btn" data-target="outputSignature">Copy</button>
        </div>
    </div>
    <div>
        <h3 class="text-lg font-semibold text-gray-700 mb-2 flex
items-center gap-1">
         X-HMAC-Data-To-Sign-Debug (for verification)
        </h3>
        <div class="relative bg-white p-3 rounded-md border
border-gray-200 text-sm font-mono whitespace-pre-wrap break-all flex
items-center justify-between">
            <span id="outputDataToSignDebug">N/A</span>
            <button class="ml-2 px-2 py-1 bg-gray-200 rounded
text-xs copy-btn" data-target="outputDataToSignDebug">Copy</button>
        </div>
    </div>
    <div>
        <h3 class="text-lg font-semibold text-gray-700 mb-2 flex
items-center gap-1">
         Generated cURL Command
        </h3>
        <div class="relative bg-gray-800 text-white p-3 rounded-md
border border-gray-600 text-sm font-mono whitespace-pre-wrap break-all
flex items-center justify-between">
            <span id="outputCurlCommand">N/A</span>
            <button class="ml-2 px-2 py-1 bg-blue-500 rounded
text-xs text-white copy-btn"

```

```

data-target="outputCurlCommand">Copy</button>
</div>
<p class="text-xs text-gray-500 mt-1">
    Copy and paste this command into your terminal to test
    against your API.
</p>
</div>
</div>
</div>
</div>
</div>

<div class="mt-8 pt-6 border-t border-gray-200 text-gray-700
text-sm">
    <h3 class="text-lg font-bold text-blue-800 mb-3">Important
    Notes:</h3>
    <ul class="list-disc list-inside space-y-2">
        <li>**Consistency is Key:** The `Access Key`, `Secret Key`,
        `Delimiter`, `HTTP Method`, `Request URI`, and `Request Body`
        (especially its canonicalization for JSON) **must exactly match** the
        expectations of your Spring Boot API's `HmacUtil.java` and
        `application.properties` configuration.</li>
        <li>**Timestamp & Nonce:** A new timestamp and unique nonce
        are generated for *each* HMAC calculation.</li>
        <li>**JSON Canonicalization:** For POST/PUT/PATCH methods, the
        `Request Body` you enter will be automatically minified
        (canonicalized) before being used in the HMAC signature calculation.
        This prevents issues with pretty-printing whitespace affecting the
        hash.</li>
        <li>**`X-HMAC-Data-To-Sign-Debug`:** This header is included
        in the generated `curl` command for debugging purposes. It shows the
        exact string that was used to calculate the HMAC on the client side,
        which can be compared with the string generated on the server if
        troubleshooting is needed.</li>
    </ul>
</div>
</div>
</div>
</div>

<script type="text/javascript">
// HmacFiddleUtil (Plain JS version): Contains utility functions for
HMAC calculation and data string building.
const HmacFiddleUtil = {
    HMAC_ALGORITHM: "HMAC", // Name for Web Crypto API. SHA-256 for
HmacSHA256
    HASH_ALGORITHM: "SHA-256", // Underlying hash algorithm for HMAC

    canonicalizeJson: (jsonString) => {
        if (!jsonString) return "";
        try {

```

```

        return JSON.stringify(JSON.parse(jsonString));
    } catch (e) {
        return jsonString;
    }
},

buildDataToSign: (httpMethod, requestUri, requestBody, timestamp,
nonce, delimiter) => {
    let canonicalRequestBody = "";
    if (['POST', 'PUT', 'PATCH'].includes(httpMethod.toUpperCase())) {
        canonicalRequestBody =
HmacFiddleUtil.canonicalizeJson(requestBody);
    }

    const parts = [
        httpMethod.toUpperCase(),
        requestUri,
        canonicalRequestBody,
        String(timestamp),
        nonce
    ];
    return parts.join(delimiter);
},

calculateHmac: async (data, secretKey, hmacAlgorithm, hashAlgorithm)
=> {
    const encoder = new TextEncoder();
    const keyData = encoder.encode(secretKey);
    const dataToSign = encoder.encode(data);

    try {
        const key = await window.crypto.subtle.importKey(
            "raw",
            keyData,
            { name: hmacAlgorithm, hash: { name: hashAlgorithm } },
            false, // not extractable
            ["sign"] // can be used to sign
        );

        const signature = await window.crypto.subtle.sign(
            { name: hmacAlgorithm, hash: { name: hashAlgorithm } },
            key,
            dataToSign
        );

        // Convert ArrayBuffer to Base64 string
        return btoa(String.fromCharCode(...new Uint8Array(signature)));
    } catch (error) {
        console.error("Web Crypto API HMAC calculation failed:", error);
    }
}

```

```

        throw new Error(`Failed to calculate HMAC: ${error.message}.
Ensure valid secret key and browser support.`);
    }
},

generateHmacHeaders: async (method, uri, body, accessKey, secretKey,
delimiter, hmacAlgorithm, hashAlgorithm) => {
    const timestamp = Date.now();
    const nonce = uuid.v4(); // Access uuid from global scope

    const dataToSign = HmacFiddleUtil.buildDataToSign(method, uri,
body, timestamp, nonce, delimiter);
    const signature = await HmacFiddleUtil.calculateHmac(dataToSign,
secretKey, hmacAlgorithm, hashAlgorithm);

    return {
        'X-Access-Key': accessKey,
        'X-Timestamp': timestamp,
        'X-Nonce': nonce,
        'X-HMAC-Signature': signature,
        'X-HMAC-Data-To-Sign-Debug': dataToSign,
        'Content-Type': 'application/json'
    };
}
};

// --- DOM Element References ---
const accessKeyInput = document.getElementById('accessKey');
const secretKeyInput = document.getElementById('secretKey');
const httpMethodSelect = document.getElementById('httpMethod');
const requestUriInput = document.getElementById('requestUri');
const requestBodyContainer =
document.getElementById('requestBodyContainer');
const requestBodyTextarea = document.getElementById('requestBody');
const delimiterInput = document.getElementById('delimiter');
const generateButton = document.getElementById('generateBtn');

const outputTimestampSpan =
document.getElementById('outputTimestamp');
const outputNonceSpan = document.getElementById('outputNonce');
const outputSignatureSpan =
document.getElementById('outputSignature');
const outputDataToSignDebugSpan =
document.getElementById('outputDataToSignDebug');
const outputCurlCommandSpan =
document.getElementById('outputCurlCommand');
const errorMessageDiv = document.getElementById('errorMessage');
const errorTextSpan = document.getElementById('errorText');

```

```

// --- Helper Functions for DOM Updates ---
function showError(message) {
    errorTextSpan.textContent = message;
    errorMessageDiv.classList.remove('hidden');
    errorMessageDiv.classList.add('flex');
}

function hideError() {
    errorMessageDiv.classList.add('hidden');
    errorMessageDiv.classList.remove('flex');
}

function updateOutput(timestamp, nonce, signature, dataToSign,
curlCmd) {
    outputTimestampSpan.textContent = timestamp;
    outputNonceSpan.textContent = nonce;
    outputSignatureSpan.textContent = signature;
    outputDataToSignDebugSpan.textContent = dataToSign;
    outputCurlCommandSpan.textContent = curlCmd;
}

function handleCopy(text) {
    const textArea = document.createElement('textarea');
    textArea.value = text;
    document.body.appendChild(textArea);
    textArea.focus();
    textArea.select();
    try {
        document.execCommand('copy');
        alert('Copied to clipboard!'); // Simple feedback
    } catch (err) {
        console.error('Failed to copy text:', err);
        alert('Failed to copy to clipboard!');
    } finally {
        document.body.removeChild(textArea);
    }
}

// --- Event Listeners ---
document.addEventListener('DOMContentLoaded', () => {
    // Attach copy listeners to all copy buttons
    document.querySelectorAll('.copy-btn').forEach(button => {
        button.addEventListener('click', (event) => {
            const targetId = event.target.dataset.target;
            const targetSpan = document.getElementById(targetId);
            if (targetSpan) {
                handleCopy(targetSpan.textContent);
            }
        });
    });
});

```



```

});

// Toggle requestBody visibility based on HTTP method
httpMethodSelect.addEventListener('change', () => {
  const method = httpMethodSelect.value.toUpperCase();
  if (['POST', 'PUT', 'PATCH'].includes(method)) {
    requestBodyContainer.style.display = 'block';
  } else {
    requestBodyContainer.style.display = 'none';
  }
});

// Initial check for requestBody visibility
httpMethodSelect.dispatchEvent(new Event('change'));

// Attach generate function to button and input changes
generateButton.addEventListener('click', generateHmac);
[accessKeyInput, secretKeyInput, httpMethodSelect,
requestUriInput, requestBodyTextarea, delimiterInput].forEach(input =>
{
  input.addEventListener('input', generateHmac);
});

// Initial generation on page load
generateHmac();
});

// --- Main Generation Function ---
async function generateHmac() {
  hideError();

  const accessKey = accessKeyInput.value;
  const secretKey = secretKeyInput.value;
  const httpMethod = httpMethodSelect.value;
  const requestUri = requestUriInput.value;
  const requestBody = requestBodyTextarea.value;
  const delimiter = delimiterInput.value;
  const hmacAlgorithm = HmacFiddleUtil.HMAC_ALGORITHM;
  const hashAlgorithm = HmacFiddleUtil.HASH_ALGORITHM;

  if (!accessKey || !secretKey || !requestUri) {
    showError('Access Key, Secret Key, and Request URI are
required.');
```

updateOutput('N/A', 'N/A', 'N/A', 'N/A', 'N/A');

return;

}

```

  if (!delimiter || delimiter.length !== 1) {
    showError('Delimiter must be a single character.');
```

```

        updateOutput('N/A', 'N/A', 'N/A', 'N/A', 'N/A');
        return;
    }

    if (['POST', 'PUT', 'PATCH'].includes(httpMethod.toUpperCase()) &&
requestBody) {
        try {
            JSON.parse(requestBody);
        } catch (e) {
            showError('Request Body must be valid JSON for POST/PUT/PATCH
methods.');
```

methods.');

```

            updateOutput('N/A', 'N/A', 'N/A', 'N/A', 'N/A');
            return;
        }
    }

    try {
        const headers = await HmacFiddleUtil.generateHmacHeaders(
            httpMethod,
            requestUri,
            requestBody,
            accessKey,
            secretKey,
            delimiter,
            hmacAlgorithm,
            hashAlgorithm
        );

        const generatedTimestamp = String(headers['X-Timestamp']);
        const generatedNonce = headers['X-Nonce'];
        const generatedSignature = headers['X-HMAC-Signature'];
        const dataToSignDebug = headers['X-HMAC-Data-To-Sign-Debug'];

        let curlCommand = `curl -X ${httpMethod}
"http://localhost:8080${requestUri}" \\\`;
        for (const key in headers) {
            if (key !== 'Content-Type' && key !==
'X-HMAC-Data-To-Sign-Debug') {
                curlCommand += `\n -H "${key}: ${headers[key]}" \\\`;
            }
        }
        if (['POST', 'PUT', 'PATCH'].includes(httpMethod.toUpperCase()) &&
requestBody) {
            curlCommand += `\n -H "Content-Type: application/json" \\\`;
            // Escape single quotes within the JSON string for shell
            curlCommand += `\n -d '${requestBody.replace(/'/g, "'\\'")}'`;
        }

        updateOutput(generatedTimestamp, generatedNonce,

```

```
generatedSignature, dataToSignDebug, curlCommand);

    } catch (error) {
        console.error("HMAC generation failed:", error);
        showError(`Error generating HMAC: ${error.message}`);
        updateOutput('N/A', 'N/A', 'N/A', 'N/A', 'N/A');
    }
}
</script>

</body>
</html>
```