# Comparing HTML and WinRT base64 Encryption APIs

Kraig Brockschmidt

*Note: A DP2 or later build is needed to use the HTML/DOM btoa and atob APIs.*

Mozilla has introduced two simple methods to the DOM API for encrypting and decrypting base64 values. These are [*btoa*](https://developer.mozilla.org/en/DOM/window.btoa), which converts binary (or string) data to a base64 ASCII-encoded string, and [*atob*](https://developer.mozilla.org/en/DOM/window.atob), which will do the opposite. These APIs are being adopted by Internet Explorer 10 and are thus also available to Metro style apps written in JavaScript.

The basic functionality of these methods overlaps with the WinRT APIs found in the [Windows.Security.Cryptography.CryptographicBuffer](http://msdn.microsoft.com/en-us/library/windows/apps/windows.security.cryptography.cryptographicbuffer(v=VS.85).aspx) class, with the results being entirely interchangeable as you would expect. This means that base64-encrypted data created in one app using the DOM APIs can be passed to and successfully decrypted by an app using the WinRT APIs, and vice versa. This is shown in the code example below.

The difference between the two APIs thus rests in their structure and scope. As seen in the code example below, the *btoa* and *atob* methods are the simplest and most concise, but they work only with binary or basic string data, and cannot handle raw Unicode. As Mozilla’s [*btoa*](https://developer.mozilla.org/en/DOM/window.btoa)docs point out, Unicode strings will cause an exception in both methods and must be worked around with additional code. These methods are also focused solely on base64.

The WinRT APIs, on the other hand, are a little more complicated for basic usage but are much more flexible overall. For one, they handle Unicode strings just fine, with support for UTF-8, UTF-16LE, and UTF-16BE. Second, these APIs work with an intermediary buffer that can be created from a string, a byte array, or filled with random data. Finally, such buffers can then be encoded to base64 or hexadecimal, or copied to a separate byte array.

In short, the two can be used interchangeably for basic base64 operations, but developers might find it more consistent to use WinRT for other kinds of encoding.

## Code example

This example simply shows the equivalence of the APIs and their cross compatibility. Create a new JavaScript project using the Blank template and drop in the code below. When you run the app and press the Run Test button, you’ll see the original string, the base64 encryption, and the decrypted values, using WinRT exclusively, using HTML APIs exclusively, then intermixing the two. As you’ll see, the results are entirely as expected.

### HTML (body of default.html)

<body>

<h1>Encryption Comparison</h1>

<button id="btnRun">Run Test</button>

<p>Input Text: <span id="txtInput">A text string for encyption</span></p>

<p>Encrypted String (WinRT): <span id="txtEncryptedWinRT"></span></p>

<p>Decrypted String (WinRT): <span id="txtDecryptedWinRT"></span></p>

<p>Encrypted String (btoa): <span id="txtEncryptedJS"></span></p>

<p>Decrypted String (atob): <span id="txtDecryptedJS"></span></p>

<p>Decrypted String (WinRT using btoa output): <span id="txtDecryptedCross1"></span></p>

<p>Decrypted String (atob using WinRT output): <span id="txtDecryptedCross2"></span></p>

</body>

### JavaScript (entire contents of default.js)

(function () {

"use strict";

var app = WinJS.Application;

app.onactivated = function (e) {

if (e.detail.kind === Windows.ApplicationModel.Activation.ActivationKind.launch) {

WinJS.UI.processAll();

var btn = btnRun;

btnRun.addEventListener("click", runTest);

}

}

function runTest() {

var wsc = Windows.Security.Cryptography;

var input = document.getElementById("txtInput").innerText;

var buffer1a = wsc.CryptographicBuffer.convertStringToBinary(input, wsc.BinaryStringEncoding.utf8);

var encrypt1 = wsc.CryptographicBuffer.encodeToBase64String(buffer1a);

document.getElementById("txtEncryptedWinRT").innerText = encrypt1;

var buffer1b = wsc.CryptographicBuffer.decodeFromBase64String(encrypt1);

document.getElementById("txtDecryptedWinRT").innerText = wsc.CryptographicBuffer.convertBinaryToString(wsc.BinaryStringEncoding.utf8, buffer1b);

var encrypt2 = btoa(input);

document.getElementById("txtEncryptedJS").innerText = encrypt2.toString();

document.getElementById("txtDecryptedJS").innerText = atob(encrypt2);

document.getElementById("txtDecryptedCross1").innerText = wsc.CryptographicBuffer.convertBinaryToString(wsc.BinaryStringEncoding.utf8,

wsc.CryptographicBuffer.decodeFromBase64String(encrypt2));

document.getElementById("txtDecryptedCross2").innerText = atob(encrypt1);

}

app.start();

})();