

# JWT & Encrypt Symetric - SSJS

- [JWT](#)
  - [SSJS Library](#)
  - [1- SFMC Symmetric Key](#)
  - [2- Creation and use of JWT](#)
  - [3- Decode JWT](#)
- [Encrypt/Decrypt Symmetric](#)
- [1- SFMC Keys](#)
  - [2- Encrypt Symmetric](#)
  - [3- Decrypt Symmetric](#)
- [JWT and Encrypt/Decrypt Symetric](#)
  - [1- Encrypted JWT](#)
  - [3- Decrypt and use JWT](#)

## JWT

JSON Web Token (JWT) is an open standard ([RFC 7519](#)) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

## SSJS Library

function jwt() used in the code is a function on the library that was created by **Sascha Huwald**, in this link you can find the entire library and how to install it <https://github.com/email360/ssjs-lib>.

### 1- SFMC Symmetric Key

Symmetric encryption requires you to create a passphrase for use with the key.

1. Top right corner (User Name) > Setup
2. In the Setup page go to Administration > Data Management > Key Management
3. Create
4. Select Symmetric and fill the details
  - a. Name
  - b. External Key (Which is gonna be used later)
  - c. Pre-Shared Key (Password 25 char max), use this page to create one <https://passwordsgenerator.net/>
  - d. Reenter Pre-Shared Key
  - e. Save

### 2- Creation and use of JWT

Use this code in Email or Cloud Page use the following code to create a JWT

```
1 <script runat="server">
2     Platform.Load("Core","1.1.1");
3     var Payload = {};
4     var Key ="KEY_GENERATED";
5     var TempTargetUrl = "https://osf.digital/en-au?token="
6
7     // Example data to be in the JWT
```

```

8
9     Payload["ContactId"] = "003123123123123123";
10    Payload["FirstName"] = "OSF";
11    Payload["LastName"] = "Digital";
12    Payload["Email"] = "info@osf.digital";
13
14
15    try{
16        // JWT Generation
17        var Token = GetJWTByKeyName(Key, "HS256", Stringify(Payload));
18
19        // Preview JWT -- need to be deleted or commented in production
20        Write("JWT: <br>");
21        Write(Token);
22        Write("<br><br>");
23
24        // Pass value into an ampscript variable
25        var TargetUrl = TempTargetUrl + Token;
26        Variable.SetValue("@TargetUrl", TargetUrl);
27
28    } catch(e) {
29        Write(e);
30    }
31
32    function GetJWTByKeyName(key, algorithm, payload) {
33        var varName = '@amp__GetJWT';
34
35        // AMP decleration
36        var amp = "%\%[ set "+varName+" = GetJWTByKeyName('"+key+"', '"+algorithm+"', '"+payload+"') output(conce
37
38        return Platform.Function.TreatAsContent(amp);
39    }
40
41 </script>
42 <!-- USE URL WITH THE JWT-->
43 <a href="%=RedirectTo(@TargetUrl)=%" target="_blank">Click Here</a>

```

### 3- Decode JWT

Use this code in the Cloud Page that want to get and use the data in the JWT

```

1 <script runat="server">
2     Platform.Load("Core", "1.1.1");
3     var Token = Request.GetQueryStringParameter("token");
4     var Key = "KEY_GENERATED";
5
6     try{
7
8         // JWT Decode
9         var jwt = new jwt();
10        var payload = jwt.decode(Token, Key);
11
12        // Preview Data in the JWT
13        Write("JWT Decoded: <br>");
14        Write(Stringify(payload));
15        Write("<br><br>");
16
17        // You can pass the values in Ampscript Variables if you need to use them

```

```

18     Variable.SetValue("@ContactId", payload.ContactId);
19     Variable.SetValue("@FirstName", payload.FirstName);
20     Variable.SetValue("@LastName", payload.LastName);
21     Variable.SetValue("@Email", payload.Email);
22
23 } catch(e) {
24     Write(e);
25 }
26
27 function jwt() {
28     /**
29      * Decode a jwt token
30      *
31      * @param {string} token A valid JWT token
32      * @param {string} [key] A symmetric key belonging to that MID Key Management to verify the tok
33      * BE AWARE: With no verification the integrity of the payload cannot be
34      * @returns {object} Returns the decoded payload
35      */
36     this.decode = function(token, key) {
37         // verify token
38         // Write("<br><br>");
39
40         // Write("THIS IS TOKEN " + token + "<br>");
41         if (key) {
42             this.verify(token, key);
43         }
44         // Write("THIS IS KEY " + key + "<br>");
45
46         // get payload
47         return Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(token.split(".")[1])))
48     };
49
50     /**
51      * Encode a jwt token
52      *
53      * @param {string} alg Name of a JWT standard hash algorithm from among HS256, HS384, or HS
54      * @param {string} key A symmetric key belonging to the MID Key Management
55      * @param {object} payload The payload, typically a JSON object with name-value pairs. The payl
56      * @param {number} [exp] Expiration time in seconds.
57      * @param {number} [nbf] Defines when the token will be active in seconds. NBF must be small
58      * @returns {string} A new JWT token.
59      */
60     this.encode = function(alg, key, payload, exp, nbf) {
61
62         // Check key
63         if (!alg) {
64             throw({message: "Require algorithm", code: 400, method: "jwt_encode"});
65         }
66
67         // Check key
68         if (!key) {
69             throw({message: "Require key", code: 400, method: "jwt_encode"});
70         }
71
72         // Check payload
73         if (!isObject(payload)) {
74             throw({message: "Require payload", code: 400, method: "jwt_encode"});
75         }

```

```

76
77 // build payload
78 payload.iat = this.getUnixTimestamp();
79
80 // check nbF
81 if (Number.isInteger(exp) && Number.isInteger(nbf) && nbf > exp) {
82     throw({message:"nbf cannot be after expiration of the token",code:400,method:"jwt_encode"});
83 }
84
85 // add expire time
86 if (exp && Number.isInteger(exp)) {
87     payload.exp = (this.getUnixTimestamp() + exp);
88 }
89
90 // add nbf
91 if (Number.isInteger(nbf)) {
92     payload.nbf = (this.getUnixTimestamp() + nbf);
93 }
94
95 // create JWT
96 var token = GetJWTByKeyName(key,alg,Stringify(payload));
97 return token;
98 };
99
100 /**
101  * Verify a jwt token
102  *
103  * @param {string} token A valid JWT token
104  * @param {string} key The symmetric key of which the signature has been encrypted with
105  * @returns {boolean}
106  */
107 this.verify = function(token, key) {
108     // check token
109     if (!token) {
110         throw({message:"No token was supplied",code:400,method:"jwt_verify"});
111     }
112
113     // check segments
114     var segments = token.split(".");
115
116     if (segments.length !== 3) {
117         throw({message:"Token structure is invalid",code:400,method:"jwt_verify"});
118     }
119
120     // verify signature
121     if (!this.verifySignature(token, key)) {
122         throw({message:"Signature verification failed",code:401,method:"jwt_verify"});
123     }
124
125     // base64 decode and parse JSON
126     var payload = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(segments[1])));
127
128
129     // Support for nbf and exp claims
130     if (payload.nbf) {
131         // check if nbf is in milliseconds or seconds
132         var unixTimestamp = (String(payload.nbf).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
133         if (unixTimestamp < payload.nbf) {

```

```

134         throw({message:"Token not yet active",code:400,method:"jwt_verify"});
135     }
136 }
137
138 if (payload.exp) {
139     // check if exp is in milliseconds or seconds
140     var unixTimestamp = (String(payload.exp).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
141     if (unixTimestamp > payload.exp) {
142         throw({message:"Token expired",code:400,method:"jwt_verify"});
143     }
144 }
145
146 return true;
147 };
148
149 /**
150     * Verify the JWT signature against a private secret
151     *
152     * @param {string} token A valid JWT token
153     * @param {string} key The symmetric key of which the signature has been encrypted with
154     * @returns {boolean}
155     */
156 this.verifySignature = function(token, key) {
157
158     return (token.split(".")[2] === this.sign(token.split(".")[0], token.split(".")[1], key));
159 };
160
161 /**
162     * Create a signature for a JWT token
163     *
164     * @param {string} header The JWT header base64 encoded
165     * @param {string} payload The JWT payload base64 encoded
166     * @param {string} key A symmetric key belonging to that MID Key Management
167     * @returns {string} The signature
168     */
169 this.sign = function(header, payload, key) {
170
171     var p = Platform.Function.ParseJSON( Platform.Function.Base64Decode( this.base64pad(payload) ) ),
172     h = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(header))),
173     alg = h.alg;
174
175
176     var jwt = GetJWTByKeyName(key,alg,Stringify(p));
177
178
179     // return signature
180     return jwt.split(".")[2];
181 };
182
183 /**
184     * Pad a base64 string to its correct length
185     *
186     * @param {string} The base64 string
187     *
188     * @return {string} The padded base64 string
189     */
190 this.base64pad = function (str) {
191     var padding = 4 - str.length % 4

```

```

192         if (padding < 4) {
193             for (var i = 0; i < padding; i++) {
194                 str += '='
195             }
196         }
197         return str;
198     };
199
200     /**
201      * Get the current UnixTimestamp
202      *
203      * @param {boolean} ms Return the UnixTimestamp in ms
204      *
205      * @returns {number} The current UnixTimestamp in UTC
206      */
207     this.getUnixTimestamp = function (ms) {
208         var ts = Math.round((new Date()).valueOf());
209         return (ms) ? ts : Math.floor(ts / 1000);
210     }
211 }
212
213 function GetJWTByKeyName(key, algorithm, payload) {
214     var varName = '@amp__GetJWT';
215
216     // AMP decleration
217     var amp = "\%\%[ set "+varName+" = GetJWTByKeyName('"+key+"', '"+algorithm+"', '"+payload+"') output(concat
218
219
220     return Platform.Function.TreatAsContent(amp);
221 }
222
223 </script>

```

## Encrypt/Decrypt Symmetric

### 1- SFMC Keys

To use 3 Keys need to be created

#### 1. Symmetric Key

- a. Symmetric encryption requires you to create a passphrase for use with the key.

#### 2. Salt Key

- a. Salt encryption requires a hex value longer than 8 bits for use as a salt value. The encryption uses random bits generated along with a password or passphrase. The salt value doesn't include a maximum length value. Use Salt keys to generate JWTs for custom Journey Builder activities. See [Encode Custom Activities Using a JWT](#) in for more details.

#### 3. IV Key

- a. Initialization vector encryption requires you to enter the block of bits to be used as the initialization vector. You can specify the 16-byte IV yourself. If you don't specify an IV, the application derives the IV from the password and salt via the protocols specified in RFC 2898.

To create them follow the next steps:

#### 1. Symmetric Key

- a. Top right corner (User Name) > Setup

- b. In the Setup page go to Administration > Data Management > Key Management
  - c. Create
  - d. Select Symmetric and fill the details
    - i. Name
    - ii. External Key (Which is gonna be used later)
    - iii. Pre-Shared Key (Password 25 char max), use this page to create one <https://passwordsgenerator.net/>
    - iv. Reenter Pre-Shared Key
    - v. Save
2. Salt Key
- a. Top right corner (User Name) > Setup
  - b. In the Setup page go to Administration > Data Management > Key Management
  - c. Create
  - d. Select Salt and fill the details
    - i. Name
    - ii. External Key (Which is gonna be used later)
    - iii. Salt (16-bit HEX) - use <https://www.browserling.com/tools/random-hex>
3. IV Key
- a. Top right corner (User Name) > Setup
  - b. In the Setup page go to Administration > Data Management > Key Management
  - c. Create
  - d. Select Initialization Vector and fill the details
    - i. Name
    - ii. External Key (Which is gonna be used later)
    - iii. IV (128-bit HEX)- use <https://www.allkeysgenerator.com/Random/Security-Encryption-Key-Generator.aspx>

## 2- Encrypt Symmetric

The EncryptSymmetric AMPscript function encrypts plain text data using the supplied algorithm and encryption values.

```

1  <script runat="server">
2      Platform.Load("Core","1.1.1");
3
4
5      var Payload = {};
6      var TempTargetUrl = "https://osf.digital/en-au?token="
7
8      // Example data to be encrypted
9
10     Payload["ContactId"] = "003123123123123123";
11     Payload["FirstName"] = "OSF";
12     Payload["LastName"] = "Digital";
13     Payload["Email"] = "info@osf.digital";
14
15     var PEK = "KEY_GENERATED"
16     var SAK = "SALT_KEY_GENERATED"
17     var IVK = "IV_KEY_GENERATED"
18
19
20     try{
21         var encrypted = EncryptEncodeString(Stringify(Payload),PEK,SAK,IVK);
22

```

```

23     Write("Data ENCRYPTED: <br>");
24     Write(encrypted);
25     Write("<br><br>");
26
27     // Pass value into an ampscript variable
28     var TargetUrl = TempTargetUrl + encrypted;
29     Variable.SetValue("@TargetUrl", TargetUrl);
30
31
32 } catch(e) {
33     Write(e);
34 }
35
36 function EncryptEncodeString(payload, PEK, SAK, IVK){
37
38     var amp = '%' + "%[";
39     amp += " SET @PEK = '"+PEK+"'";
40     amp += " SET @SAK = '"+SAK+"'";
41     amp += " SET @IVK = '"+IVK+"'";
42     amp += "set @encrypted = Concat(EncryptSymmetric('" + payload + "', 'AES',";
43     amp += "@PEK, @null,";
44     amp += "@SAK, @null,";
45     amp += "@IVK, @null))";
46     amp += "set @enc = Base64Encode(@encrypted)";
47     amp += "output(concat(@enc))";
48     amp += ']' + '%';
49
50     var val = Platform.Function.TreatAsContent(amp);
51     return val;
52
53 }
54
55 </script>
56
57 <!-- USE URL WITH THE ENCRYPTED DATA-->
58 <a href="%%=RedirectTo(@TargetUrl)=%%" target="_blank">Click Here</a>

```

### 3- Decrypt Symmetric

The DecryptSymmetric AMPscript function decrypts encrypted data using the supplied algorithm and encryption values.

```

1 <script runat="server">
2     Platform.Load("Core", "1.1.1");
3     // var Token = Request.GetQueryStringParameter("token");
4     var Token = "L0E3M3UrRUFTRTVJL2VHRzEwTjd5bFBHcURJavZPVU9VmRET2Jjs0pYQVc2MTZMN3QyZWpxam8wOHppbm1nYU9lOVNXTXNpZ"
5
6     var PEK = "KEY_GENERATED"
7     var SAK = "SALT_KEY_GENERATED"
8     var IVK = "IV_KEY_GENERATED"
9
10    try{
11        var decrypted = DecryptDecodeString(Token, PEK, SAK, IVK);
12
13        Write("Data DECRYPTED: <br>");
14        Write(decrypted);
15        Write("<br><br>");
16
17        var payload = Platform.Function.ParseJSON(decrypted);

```



```

18
19     // You can pass the values in Ampscript Variables if you need to use them
20     Variable.SetValue("@ContactId",payload.ContactId);
21     Variable.SetValue("@FirstName",payload.FirstName);
22     Variable.SetValue("@LastName",payload.LastName);
23     Variable.SetValue("@Email",payload.Email);
24
25 } catch(e) {
26     Write(e);
27 }
28
29 function DecryptDecodeString(payload,PEK,SAK,IVK){
30     var amp = '%' + "%[";
31     amp += " SET @PEK = '"+PEK+"'";
32     amp += " SET @SAK = '"+SAK+"'";
33     amp += " SET @IVK = '"+IVK+"'";
34     amp += "set @dec = Base64Decode('" + payload + "')";
35     amp += "set @decrypted = Concat(DecryptSymmetric(@dec, 'AES',";
36     amp += "@PEK, @null,";
37     amp += "@SAK, @null,";
38     amp += "@IVK, @null))";
39     amp += "output(concat(@decrypted))";
40     amp += ']' + '%';
41
42     var val = Platform.Function.TreatAsContent(amp);
43     return val;
44
45 }
46
47 </script>
48

```

## JWT and Encrypt/Decrypt Symetric

The next is combining the both to give more security with the difference the encryption will be not Encoded

### 1- Encrypted JWT

Code hte be used in the Email or Ccloud Page that will redirect to the next one

```

1 <script runat="server">
2     Platform.Load("Core","1");
3
4     var Payload = {};
5     var TempTargetUrl = "https://osf.digital/en-au?token="
6
7     // Example data to be in the JWT
8     Payload["ContactId"] = "003123123123123123";
9     Payload["FirstName"] = "OSF";
10    Payload["LastName"] = "Digital";
11    Payload["Email"] = "info@osf.digital";
12
13    var PEK = "KEY_GENERATED"
14    var SAK = "SALT_KEY_GENERATED"
15    var IVK = "IV_KEY_GENERATED"
16
17    try{

```

```

18
19 // JWT Generation
20 var Token = GetJWTByKeyName(PEK, "HS256",Stringify(Payload));
21
22 Write("JWT: <br>");
23 Write(Token);
24 Write("<br><br>");
25
26 var encrypted = EncryptString(Token,PEK,SAK,IVK);
27
28 Write("JWT ENCRYPTED: <br>");
29 Write(encrypted);
30 Write("<br><br>");
31
32 // Pass value into an ampscript variable
33 var TargetUrl = TempTargetUrl + encrypted;
34 Variable.SetValue("@TargetUrl",TargetUrl);
35
36 } catch(e) {
37     Write(e);
38 }
39
40 function GetJWTByKeyName(key,algorithm,payload) {
41     var varName = '@amp__GetJWT';
42
43     // AMP decleration
44     var amp = "%\%[ set "+varName+" = GetJWTByKeyName('"+key+"','"+algorithm+"','"+payload+"') output(concat(
45
46
47     return Platform.Function.TreatAsContent(amp);
48 }
49
50 function EncryptEncodeString(payload,PEK,SAK,IVK){
51
52     var amp = '%' + "%[";
53     amp += " SET @PEK = '"+PEK+"'";
54     amp += " SET @SAK = '"+SAK+"'";
55     amp += " SET @IVK = '"+IVK+"'";
56     amp += "set @encrypted = Concat(EncryptSymmetric('"+ payload + "', 'AES',";
57     amp += "@PEK, @null,";
58     amp += "@SAK, @null,";
59     amp += "@IVK, @null))";
60     amp += "set @enc = Base64Encode(@encrypted)";
61     amp += "output(concat(@enc))";
62     amp += ']' + '%';
63
64     var val = Platform.Function.TreatAsContent(amp);
65     return val;
66
67 }
68
69
70 </script>
71
72 <a href="%=RedirectTo(@TargetUrl)=%" target="_blank">Click Here</a>

```

### 3- Decrypt and use JWT

Code to be used in the Target Cloud page to Decrypt and Decode the JWT

```
1  <script runat="server">
2      Platform.Load("Core","1");
3      var Token = Request.GetQueryStringParameter("token");
4
5      var PEK = "KEY_GENERATED"
6      var SAK = "SALT_KEY_GENERATED"
7      var IVK = "IV_KEY_GENERATED"
8
9      try{
10
11          // JWT
12          var jwt = new jwt();
13          var decrypted = DecryptString(Token,PEK,SAK,IVK);
14
15          Write("JWT DECRYPTED: <br>");
16          Write(decrypted);
17          Write("<br><br>");
18
19          var payload = jwt.decode(decrypted,PEK);
20
21          Write("JWT VALUES: <br>");
22          Write(Stringify(payload));
23          Write("<br><br>");
24
25          // You can pass the values in Ampscript Variables if you need to use them
26          Variable.SetValue("@ContactId",payload.ContactId);
27          Variable.SetValue("@FirstName",payload.FirstName);
28          Variable.SetValue("@LastName",payload.LastName);
29          Variable.SetValue("@Email",payload.Email);
30
31
32      } catch(e) {
33          Write(e);
34      }
35
36      function jwt() {
37          /**
38              * Decode a jwt token
39              *
40              * @param {string} token A valid JWT token
41              * @param {string} [key] A symmetric key belonging to that MID Key Management to verify the token
42              * BE AWARE: With no verification the integrity of the payload cannot be guaranteed
43              * @returns {object} Returns the decoded payload
44              */
45          this.decode = function(token, key) {
46              // verify token
47              // Write("<br><br>");
48
49              // Write("THIS IS TOKEN "+ token + "<br>");
50              if (key) {
51                  this.verify(token,key);
52              }
53              // Write("THIS IS KEY "+ key + "<br>");
54
55              // get payload
```

```

56     return Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(token.split(".")[1])))
57 };
58
59 /**
60     * Encode a jwt token
61     *
62     * @param {string} alg      Name of a JWT standard hash algorithm from among HS256, HS384, or HS
63     * @param {string} key      A symmetric key belonging to the MID Key Management
64     * @param {object} payload  The payload, typically a JSON object with name-value pairs. The payl
65     * @param {number} [exp]    Expiration time in seconds.
66     * @param {number} [nbf]    Defines when the token will be active in seconds. NBF must be smaller
67     * @returns {string}        A new JWT token.
68     */
69 this.encode = function(alg, key, payload, exp, nbf) {
70
71     // Check key
72     if (!alg) {
73         throw({message:"Require algorithm",code:400,method:"jwt_encode"});
74     }
75
76     // Check key
77     if (!key) {
78         throw({message:"Require key",code:400,method:"jwt_encode"});
79     }
80
81     // Check payload
82     if (!isObject(payload)) {
83         throw({message:"Require payload",code:400,method:"jwt_encode"});
84     }
85
86     // build payload
87     payload.iat = this.getUnixTimestamp();
88
89     // check nbf
90     if (Number.isInteger(exp) && Number.isInteger(nbf) && nbf > exp) {
91         throw({message:"nbf cannot be after expiration of the token",code:400,method:"jwt_encode"});
92     }
93
94     // add expire time
95     if (exp && Number.isInteger(exp)) {
96         payload.exp = (this.getUnixTimestamp() + exp);
97     }
98
99     // add nbf
100    if (Number.isInteger(nbf)) {
101        payload.nbf = (this.getUnixTimestamp() + nbf);
102    }
103
104    // create JWT
105    var token = GetJWTByKeyName(key,alg,Stringify(payload));
106    return token;
107 };
108
109 /**
110     * Verify a jwt token
111     *
112     * @param {string} token  A valid JWT token
113     * @param {string} key    The symmetric key of which the signature has been encrypted with

```

```

114         * @returns {boolean}
115         */
116     this.verify = function(token, key) {
117         // check token
118         if (!token) {
119             throw({message:"No token was supplied",code:400,method:"jwt_verify"});
120         }
121
122         // check segments
123         var segments = token.split(".");
124
125         if (segments.length !== 3) {
126             throw({message:"Token structure is invalid",code:400,method:"jwt_verify"});
127         }
128
129         // verify signature
130         if (!this.verifySignature(token, key)) {
131             throw({message:"Signature verification failed",code:401,method:"jwt_verify"});
132         }
133
134         // base64 decode and parse JSON
135         var payload = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(segments[1])));
136
137
138         // Support for nbf and exp claims
139         if (payload.nbf) {
140             // check if nbf is in milliseconds or seconds
141             var unixTimestamp = (String(payload.nbf).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
142             if (unixTimestamp < payload.nbf) {
143                 throw({message:"Token not yet active",code:400,method:"jwt_verify"});
144             }
145         }
146
147         if (payload.exp) {
148             // check if exp is in milliseconds or seconds
149             var unixTimestamp = (String(payload.exp).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
150             if (unixTimestamp > payload.exp) {
151                 throw({message:"Token expired",code:400,method:"jwt_verify"});
152             }
153         }
154
155         return true;
156     };
157
158     /**
159     * Verify the JWT signature against a private secret
160     *
161     * @param {string} token A valid JWT token
162     * @param {string} key The symmetric key of which the signature has been encrypted with
163     * @returns {boolean}
164     */
165     this.verifySignature = function(token, key) {
166
167         return (token.split(".")[2] === this.sign(token.split(".")[0], token.split(".")[1], key));
168     };
169
170     /**
171     * Create a signature for a JWT token

```

```

172      *
173      * @param {string} header The JWT header base64 encoded
174      * @param {string} payload The JWT payload base64 encoded
175      * @param {string} key A symmetric key belonging to that MID Key Management
176      * @returns {string} The signature
177      */
178      this.sign = function(header, payload, key) {
179
180          var p = Platform.Function.ParseJSON( Platform.Function.Base64Decode( this.base64pad(payload) ) ),
181              h = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(header))),
182              alg = h.alg;
183
184
185          var jwt = GetJWTByKeyName(key,alg,Stringify(p));
186
187
188
189          // return signature
190          return jwt.split(".")[2];
191      };
192
193      /**
194       * Pad a base64 string to its correct length
195       *
196       * @param {string} The base64 string
197       *
198       * @return {string} The padded base64 string
199       */
200      this.base64pad = function (str) {
201          var padding = 4 - str.length % 4
202          if (padding < 4) {
203              for (var i = 0; i < padding; i++) {
204                  str += '='
205              }
206          }
207          return str;
208      };
209
210      /**
211       * Get the current UnixTimestamp
212       *
213       * @param {boolean} ms Return the UnixTimestamp in ms
214       *
215       * @returns {number} The current UnixTimestamp in UTC
216       */
217      this.getUnixTimestamp = function (ms) {
218          var ts = Math.round((new Date()).valueOf());
219          return (ms) ? ts : Math.floor(ts / 1000);
220      }
221  }
222
223  function GetJWTByKeyName(key,algorithm,payload) {
224      var varName = '@amp__GetJWT';
225
226      // AMP declaration
227      var amp = "%\%[% set "+varName+" = GetJWTByKeyName('"+key+"','"+algorithm+"','"+payload+"') output(concat
228
229

```

```
230     return Platform.Function.TreatAsContent(amp);
231 }
232
233 function DecryptString(payload,PEK,SAK,IVK){
234
235     var amp = '%' + "[% Output(Concat(DecryptSymmetric('" + payload + "', 'AES',"
236     amp += "'"+PEK+"', @null,";
237     amp += "'"+SAK+"', @null,";
238     amp += "'"+IVK+"', @null)))";
239     amp += ']%' + '%';
240
241     var val = Platform.Function.TreatAsContent(amp);
242     return val;
243
244 }
245
246 </script>
247
```