JWT & Encrypt Symetric - SSJS

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

```
<script runat="server">
    Platform.Load("Core","1.1.1");
    var Payload = {};
    var Key ="KEY_GENERATED";
    var TempTargetUrl = "https://osf.digital/en-au?token="

    // Example data to be in the JWT

    Payload["ContactId"] = "003123123123123123";
    Payload["FirstName"] = "OSF";
    Payload["LastName"] = "Digital";
    Payload["Email"] = "info@osf.digital";


    try{
        // JWT Generation
        var Token = GetJWTByKeyName(Key,"HS256",Stringify(Payload));

        // Preview JWT  -- need to be deleted or commented in production
        Write("JWT: <br>");
        Write(Token);
        Write("<br><br>");

        // Pass value into an ampscript variable
        var TargetUrl = TempTargetUrl + Token;
        Variable.SetValue("@TargetUrl",TargetUrl);

    } catch(e) {
        Write(e);
    }

    function GetJWTByKeyName(key,algorithm,payload) {
        var varName = '@amp__GetJWT';

        // AMP decleration
        var amp = "\%\%[ set "+varName+" = GetJWTByKeyName('"+key+"','"+algorithm+"','"+payload+"') output(conca

        return Platform.Function.TreatAsContent(amp);
    }

</script>
<!-- USE URL WITH THE JWT-->
<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

```
<script runat="server">
    Platform.Load("Core","1.1.1");
    var Token = Request.GetQueryStringParameter("token");
    var Key ="KEY_GENERATED";

    try{

      // JWT Decode
      var jwt = new jwt();
      var payload = jwt.decode(Token,Key);

      // Preview Data in the JWT
      Write("JWT Decoded: <br>");
      Write(Stringify(payload));
      Write("<br><br>");

      // You can pass the values in Ampscript Variables if you need to use them
      Variable.SetValue("@ContactId",payload.ContactId);
      Variable.SetValue("@FirstName",payload.FirstName);
      Variable.SetValue("@LastName",payload.LastName);
      Variable.SetValue("@Email",payload.Email);

    } catch(e) {
      Write(e);
    }

    function jwt() {
      /**
             * Decode a jwt token
             *
             * @param   {string}  token  A valid JWT token
             * @param   {string}  [key]  A symmetric key belonging to that MID Key Management to verify the tok
             *                           BE AWARE: With no verification the integrety of the payload cannot be
             * @returns {object}         Returns the decoded payload
             */
      this.decode = function(token, key) {
        // verify token
        // Write("<br><br>");

        // Write("THIS IS TOKEN "+ token +"<br>");
        if (key) {
          this.verify(token,key);
        }
        //  Write("THIS IS KEY "+ key +"<br>");

        // get payload
        return Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(token.split(".")[1])))
      };

      /**
             * Encode a jwt token
             *
             * @param   {string}  alg      Name of a JWT standard hash algorithm from among HS256, HS384, or HS
             * @param   {string}  key      A symmetric key belonging to the MID Key Management
             * @param   {object}  payload  The payload, typically a JSON object with name-value pairs. The payl
             * @param   {number}  [exp]    Expiration time in seconds.
             * @param   {number}  [nbf]    Defines when the token will be active in seconds. NBF must be smalle
             * @returns {string}           A new JWT token.
             */
      this.encode = function(alg, key, payload, exp, nbf) {

        // Check key
        if (!alg) {
          throw({message:"Require algorithm",code:400,method:"jwt_encode"});
        }

        // Check key
        if (!key) {
          throw({message:"Require key",code:400,method:"jwt_encode"});
        }

        // Check payload
        if (!isObject(payload)) {
          throw({message:"Require payload",code:400,method:"jwt_encode"});
        }
```

```

        // build payload
        payload.iat = this.getUnixTimestamp();

        // check nbf
        if (Number.isInteger(exp) && Number.isInteger(nbf) && nbf > exp)  {
          throw({message:"nbf cannot be after expiration of the token",code:400,method:"jwt_encode"});
        }

        // add expire time
        if (exp && Number.isInteger(exp))  {
          payload.exp = (this.getUnixTimestamp() + exp);
        }

        // add nbf
        if (Number.isInteger(nbf))  {
          payload.nbf = (this.getUnixTimestamp() + nbf);
        }

        // create JWT
        var token = GetJWTByKeyName(key,alg,Stringify(payload));
        return token;
      };

      /**
              * Verify a jwt token
              *
              * @param   {string}   token  A valid JWT token
              * @param   {string}   key    The symmetric key of which the signature has been encrypted with
              * @returns {boolean}
              */
      this.verify = function(token, key) {
        // check token
        if (!token) {
          throw({message:"No token was supplied",code:400,method:"jwt_verify"});
        }

        // check segments
        var segments = token.split(".");

        if (segments.length !== 3) {
          throw({message:"Token structure is invalid",code:400,method:"jwt_verify"});
        }

        // verify signature
        if (!this.verifySignature(token, key)) {
          throw({message:"Signature verification failed",code:401,method:"jwt_verify"});
        }

        // base64 decode and parse JSON
        var payload = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(segments[1])));


        // Support for nbf and exp claims
        if (payload.nbf) {
          // check if nbf is in miliseconds or seconds
          var unixTimestamp = (String(payload.nbf).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
          if (unixTimestamp < payload.nbf) {
```

```
          throw({message:"Token not yet active",code:400,method:"jwt_verify"});
        }
      }

      if (payload.exp) {
        // check if exp is in miliseconds or seconds
        var unixTimestamp = (String(payload.exp).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
        if (unixTimestamp > payload.exp) {
          throw({message:"Token expired",code:400,method:"jwt_verify"});
        }
      }

      return true;
    };

    /**
           * Verify the JWT signature against a private secret
           *
           * @param   {string}   token  A valid JWT token
           * @param   {string}   key    The symmetric key of which the signature has been encrypted with
           * @returns {boolean}
           */
    this.verifySignature = function(token, key) {

      return (token.split(".")[2] === this.sign(token.split(".")[0], token.split(".")[1], key));
    };

    /**
           * Create a signature for a JWT token
           *
           * @param    {string}  header   The JWT header base64 encoded
           * @param    {string}  payload  The JWT payload base64 encoded
           * @param    {string}  key      A symmetric key belonging to that MID Key Management
           * @returns  {string}           The signature
           */
    this.sign = function(header, payload, key) {

      var p = Platform.Function.ParseJSON( Platform.Function.Base64Decode( this.base64pad(payload) ) ),
          h = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(header))),
          alg = h.alg;


      var   jwt = GetJWTByKeyName(key,alg,Stringify(p));


      // return signature
      return jwt.split(".")[2];
    };

    /**
           * Pad a base64 string to its correct length
           *
           * @param {string}  The base64 string
           *
           * @return {string} The padded base64 string
           */
    this.base64pad = function (str) {
      var padding = 4 - str.length % 4
      if (padding < 4) {
        for (var i = 0; i < padding; i++) {
          str += '='
        }
      }
      return str;
    };

    /**
           * Get the current UnixTimestamp
           *
           * @param {boolean} ms  Return the UnixTimestamp in ms
           *
           * @returns {number} The current UnixTimestamp in UTC
           */
    this.getUnixTimestamp = function (ms) {
      var ts = Math.round((new Date()).valueOf());
      return (ms) ? ts : Math.floor(ts / 1000);
    }
  }

  function GetJWTByKeyName(key,algorithm,payload) {
    var varName = '@amp__GetJWT';

    // AMP decleration
    var amp = "\%\%[ set "+varName+" = GetJWTByKeyName('"+key+"','"+algorithm+"','"+payload+"') output(concat


    return Platform.Function.TreatAsContent(amp);
  }

</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.

```
<script runat="server">
    Platform.Load("Core","1.1.1");


    var Payload = {};
    var TempTargetUrl = "https://osf.digital/en-au?token="

    // Example data to be encrypted

    Payload["ContactId"] = "003123123123123123";
    Payload["FirstName"] = "OSF";
    Payload["LastName"] = "Digital";
    Payload["Email"] = "info@osf.digital";

    var PEK = "KEY_GENERATED"
    var SAK = "SALT_KEY_GENERATED"
    var IVK = "IV_KEY_GENERATED"


    try{
        var encrypted = EncryptEncodeString(Stringify(Payload),PEK,SAK,IVK);

        Write("Data ENCRYPTED: <br>");
        Write(encrypted);
        Write("<br><br>");

        // Pass value into an ampscript variable
       var TargetUrl = TempTargetUrl + encrypted;
       Variable.SetValue("@TargetUrl",TargetUrl);


    } catch(e) {
      Write(e);
    }

    function EncryptEncodeString(payload,PEK,SAK,IVK){

        var amp = '%' + "%[";
        amp += " SET @PEK = '"+PEK+"'";
        amp += " SET @SAK = '"+SAK+"'";
        amp += " SET @IVK = '"+IVK+"'";
        amp += "set @encrypted = Concat(EncryptSymmetric('" + payload + "', 'AES',";
        amp += "@PEK, @null,";
        amp += "@SAK, @null,";
        amp += "@IVK, @null))";
        amp += "set @enc = Base64Encode(@encrypted)";
        amp += "output(concat(@enc))";
        amp += ']%' + '%';

        var val = Platform.Function.TreatAsContent(amp);
        return val;

    }

  </script>

<!-- USE URL WITH THE ENCRYPTED DATA-->
<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.

```
<script runat="server">
  Platform.Load("Core","1.1.1");
  // var Token = Request.GetQueryStringParameter("token");
  var Token = "L0E3M3UrRUFTRTVJL2VHRzEwTjd5bFBHcURJaVZPVU9WVmRET2JjS0pYQVc2MTZMN3QyZWpxam8wOHppbm1nYU9lOVNxTXNpZ

  var PEK = "KEY_GENERATED"
  var SAK = "SALT_KEY_GENERATED"
  var IVK = "IV_KEY_GENERATED"

  try{
    var decrypted = DecryptDecodeString(Token,PEK,SAK,IVK);

    Write("Data DECRYPTED: <br>");
    Write(decrypted);
    Write("<br><br>");

    var payload = Platform.Function.ParseJSON(decrypted);

    // You can pass the values in Ampscript Variables if you need to use them
    Variable.SetValue("@ContactId",payload.ContactId);
    Variable.SetValue("@FirstName",payload.FirstName);
    Variable.SetValue("@LastName",payload.LastName);
    Variable.SetValue("@Email",payload.Email);

  } catch(e) {
    Write(e);
  }

  function DecryptDecodeString(payload,PEK,SAK,IVK){
        var amp = '%' + "%[";
        amp += " SET @PEK = '"+PEK+"'";
        amp += " SET @SAK = '"+SAK+"'";
        amp += " SET @IVK = '"+IVK+"'";
        amp += "set @dec = Base64Decode('" + payload + "')";
        amp += "set @decrypted = Concat(DecryptSymmetric(@dec, 'AES',";
        amp += "@PEK, @null,";
        amp += "@SAK, @null,";
        amp += "@IVK, @null))";
        amp += "output(concat(@decrypted))";
        amp += ']%' + '%';

        var val = Platform.Function.TreatAsContent(amp);
        return val;

    }

</script>

```

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 Clloud Page that will redirect to the next one

```
<script runat="server">
    Platform.Load("Core","1");

    var Payload = {};
    var TempTargetUrl = "https://osf.digital/en-au?token="

    // Example data to be in the JWT
    Payload["ContactId"] = "003123123123123123";
    Payload["FirstName"] = "OSF";
    Payload["LastName"] = "Digital";
    Payload["Email"] = "info@osf.digital";

    var PEK = "KEY_GENERATED"
    var SAK = "SALT_KEY_GENERATED"
    var IVK = "IV_KEY_GENERATED"

    try{

      // JWT Generation
      var Token = GetJWTByKeyName(PEK,"HS256",Stringify(Payload));

      Write("JWT: <br>");
      Write(Token);
      Write("<br><br>");

      var encrypted = EncryptString(Token,PEK,SAK,IVK);

      Write("JWT ENCRYPTED: <br>");
      Write(encrypted);
      Write("<br><br>");

      // Pass value into an ampscript variable
      var TargetUrl = TempTargetUrl + encrypted;
      Variable.SetValue("@TargetUrl",TargetUrl);

    } catch(e) {
      Write(e);
    }

    function GetJWTByKeyName(key,algorithm,payload) {
      var varName = '@amp__GetJWT';

      // AMP decleration
      var amp = "\%\%[ set "+varName+" = GetJWTByKeyName('"+key+"','"+algorithm+"','"+payload+"') output(concat(


      return Platform.Function.TreatAsContent(amp);
    }

     function EncryptEncodeString(payload,PEK,SAK,IVK){

        var amp = '%' + "%[";
        amp += " SET @PEK = '"+PEK+"'";
        amp += " SET @SAK = '"+SAK+"'";
        amp += " SET @IVK = '"+IVK+"'";
        amp += "set @encrypted = Concat(EncryptSymmetric('" + payload + "', 'AES',";
        amp += "@PEK, @null,";
        amp += "@SAK, @null,";
        amp += "@IVK, @null))";
        amp += "set @enc = Base64Encode(@encrypted)";
        amp += "output(concat(@enc))";
        amp += ']%' + '%';

        var val = Platform.Function.TreatAsContent(amp);
        return val;

    }


  </script>

<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

```
<script runat="server">
    Platform.Load("Core","1");
    var Token = Request.GetQueryStringParameter("token");

    var PEK = "KEY_GENERATED"
    var SAK = "SALT_KEY_GENERATED"
    var IVK = "IV_KEY_GENERATED"

    try{

        // JWT
        var jwt = new jwt();
        var decrypted = DecryptString(Token,PEK,SAK,IVK);

        Write("JWT DECRYPTED: <br>");
        Write(decrypted);
        Write("<br><br>");

        var payload = jwt.decode(decrypted,PEK);

        Write("JWT VALUES: <br>");
        Write(Stringify(payload));
        Write("<br><br>");

        // You can pass the values in Ampscript Variables if you need to use them
        Variable.SetValue("@ContactId",payload.ContactId);
        Variable.SetValue("@FirstName",payload.FirstName);
        Variable.SetValue("@LastName",payload.LastName);
        Variable.SetValue("@Email",payload.Email);


    } catch(e) {
      Write(e);
    }

    function jwt() {
      /**
             * Decode a jwt token
             *
             * @param   {string}  token  A valid JWT token
             * @param   {string}  [key]  A symmetric key belonging to that MID Key Management to verify the tok
             *                           BE AWARE: With no verification the integrety of the payload cannot be
             * @returns {object}         Returns the decoded payload
             */
      this.decode = function(token, key) {
        // verify token
        // Write("<br><br>");

        // Write("THIS IS TOKEN "+ token +"<br>");
        if (key) {
          this.verify(token,key);
        }
        //  Write("THIS IS KEY "+ key +"<br>");

        // get payload
        return Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(token.split(".")[1])))
      };

      /**
             * Encode a jwt token
             *
             * @param   {string}  alg      Name of a JWT standard hash algorithm from among HS256, HS384, or HS
             * @param   {string}  key      A symmetric key belonging to the MID Key Management
             * @param   {object}  payload  The payload, typically a JSON object with name-value pairs. The payl
             * @param   {number}  [exp]    Expiration time in seconds.
             * @param   {number}  [nbf]    Defines when the token will be active in seconds. NBF must be smalle
             * @returns {string}           A new JWT token.
             */
      this.encode = function(alg, key, payload, exp, nbf) {

        // Check key
        if (!alg) {
          throw({message:"Require algorithm",code:400,method:"jwt_encode"});
        }

        // Check key
        if (!key) {
          throw({message:"Require key",code:400,method:"jwt_encode"});
        }

        // Check payload
        if (!isObject(payload)) {
          throw({message:"Require payload",code:400,method:"jwt_encode"});
        }

        // build payload
        payload.iat = this.getUnixTimestamp();

        // check nbf
        if (Number.isInteger(exp) && Number.isInteger(nbf) && nbf > exp)  {
          throw({message:"nbf cannot be after expiration of the token",code:400,method:"jwt_encode"});
        }

        // add expire time
        if (exp && Number.isInteger(exp))  {
          payload.exp = (this.getUnixTimestamp() + exp);
        }

        // add nbf
        if (Number.isInteger(nbf))  {
          payload.nbf = (this.getUnixTimestamp() + nbf);
        }

        // create JWT
        var token = GetJWTByKeyName(key,alg,Stringify(payload));
        return token;
      };

      /**
             * Verify a jwt token
             *
             * @param   {string}   token  A valid JWT token
             * @param   {string}   key    The symmetric key of which the signature has been encrypted with
```

```
           * @returns {boolean}
           */
    this.verify = function(token, key) {
      // check token
      if (!token) {
        throw({message:"No token was supplied",code:400,method:"jwt_verify"});
      }

      // check segments
      var segments = token.split(".");

      if (segments.length !== 3) {
        throw({message:"Token structure is invalid",code:400,method:"jwt_verify"});
      }

      // verify signature
      if (!this.verifySignature(token, key)) {
        throw({message:"Signature verification failed",code:401,method:"jwt_verify"});
      }

      // base64 decode and parse JSON
      var payload = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(segments[1])));


      // Support for nbf and exp claims
      if (payload.nbf) {
        // check if nbf is in miliseconds or seconds
        var unixTimestamp = (String(payload.nbf).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
        if (unixTimestamp < payload.nbf) {
          throw({message:"Token not yet active",code:400,method:"jwt_verify"});
        }
      }

      if (payload.exp) {
        // check if exp is in miliseconds or seconds
        var unixTimestamp = (String(payload.exp).length == 13) ? this.getUnixTimestamp(true) : this.getUnixTi
        if (unixTimestamp > payload.exp) {
          throw({message:"Token expired",code:400,method:"jwt_verify"});
        }
      }

      return true;
    };

    /**
           * Verify the JWT signature against a private secret
           *
           * @param   {string}   token  A valid JWT token
           * @param   {string}   key    The symmetric key of which the signature has been encrypted with
           * @returns {boolean}
           */
    this.verifySignature = function(token, key) {

      return (token.split(".")[2] === this.sign(token.split(".")[0], token.split(".")[1], key));
    };

    /**
           * Create a signature for a JWT token
```

```
             *
             * @param    {string}  header   The JWT header base64 encoded
             * @param    {string}  payload  The JWT payload base64 encoded
             * @param    {string}  key      A symmetric key belonging to that MID Key Management
             * @returns  {string}           The signature
             */
      this.sign = function(header, payload, key) {

        var p = Platform.Function.ParseJSON( Platform.Function.Base64Decode( this.base64pad(payload) ) ),
            h = Platform.Function.ParseJSON(Platform.Function.Base64Decode(this.base64pad(header))),
            alg = h.alg;


        var   jwt = GetJWTByKeyName(key,alg,Stringify(p));



        // return signature
        return jwt.split(".")[2];
      };

      /**
             * Pad a base64 string to its correct length
             *
             * @param {string}  The base64 string
             *
             * @return {string} The padded base64 string
             */
      this.base64pad = function (str) {
        var padding = 4 - str.length % 4
        if (padding < 4) {
          for (var i = 0; i < padding; i++) {
            str += '='
          }
        }
        return str;
      };

      /**
             * Get the current UnixTimestamp
             *
             * @param {boolean} ms  Return the UnixTimestamp in ms
             *
             * @returns {number} The current UnixTimestamp in UTC
             */
      this.getUnixTimestamp = function (ms) {
        var ts = Math.round((new Date()).valueOf());
        return (ms) ? ts : Math.floor(ts / 1000);
      }
    }

    function GetJWTByKeyName(key,algorithm,payload) {
      var varName = '@amp__GetJWT';

      // AMP decleration
      var amp = "\%\%[ set "+varName+" = GetJWTByKeyName('"+key+"','"+algorithm+"','"+payload+"') output(concat


      return Platform.Function.TreatAsContent(amp);
    }

    function DecryptString(payload,PEK,SAK,IVK){

        var amp = '%' + "%[ Output(Concat(DecryptSymmetric('" + payload + "', 'AES',";
        amp += "'"+PEK+"', @null,";
        amp += "'"+SAK+"', @null,";
        amp += "'"+IVK+"', @null)))";
        amp += ']%' + '%';

        var val = Platform.Function.TreatAsContent(amp);
        return val;

    }

</script>
```