Cryptography for JavaScript Developer

Anish Nath

Cryptography for Java Script Developer

Anish Nath

This book is for sale at http://leanpub.com/cryptojs

This version was published on 2018-10-26

I would like to dedicate this book to my 8-year old son Arjun Nath , Grandfather Sri Rajeshwar Prasad; wife Manisha Prasad; mother Indu Sinha; and all my family members (my father Anil Kumar Sinha; chote papa Sunil Kumar Sinha; choti mummy: Poonam Sinha; and friends). Without them, this would not have been possible.

I would also thanks to Cisco where I work, I got most of the learning from there, and not least the opensource community.

I would also thanks to reader who spend their money and time to read this book, without them there is no Inspiration.

Contents

JavaScript Crypto Libraries

This book is dedicated to use WebCryptoAPI and sjcl javascript library though it's good to know other javascript Library available.

Library	Description
WebCryptoAPI[1]	The **Web Crypto API** is an interface allowing a script to use cryptographic primitives in order to build systems using cryptography
sjcl[2]	The Stanford Javascript Crypto Library is maintained on GitHub.
forge[3]	A native implementation of TLS (and various other cryptographic tools) in JavaScript.
js-nacl[4]	Pure-Javascript High-level API to Emscripten-compiled libsodium routines
jsbn[5]	basic BigInteger implementation, just enough for RSA encryption and not much more
OpenPGP.js[6]	OpenPGP.js is a JavaScript implementation of the OpenPGP protocol.
cryptoJS[7]	CryptoJS is a growing collection of standard and secure cryptographic algorithms implemented in JavaScript using best practices and patterns. They are fast, and they have a consistent and simple interface. CURRENTLY No Longer Maintained
Cifre[8]	Cifre is a fast crypto toolkit for modern client-side JavaScript. This is done by taking the best crypto code for js on the net and updating it to use modern technologies. CURRENTLY they move to FORGE.

[1] https://www.w3.org/TR/WebCryptoAPI/
[2] https://crypto.stanford.edu/sjcl/
[3] https://github.com/digitalbazaar/forge
[4] https://github.com/tonyg/js-nacl
[5] https://github.com/hoangductho/jsbn
[6] https://github.com/openpgpjs/openpgpjs
[7] http://code.google.com/p/crypto-js/
[8] https://github.com/openpeer/cifre

Library	Description
PKIjs[9]	PKIjs is a pure JavaScript library implementing the formats that are used in PKI applications. It is built on WebCrypto (Web Cryptography API) and aspires to make it possible to build native web applications that utilize X.509 and the related formats on the web without plug-ins.
ASN1js[10]	ASN1js is a pure JavaScript library implementing this standard. ASN.1 is the basis of all X.509 related data structures and numerous other protocols used on the web.
PKCS11js[11]	PKCS#11 (also known as CryptoKI or PKCS11) is the standard interface for interacting with hardware crypto devices such as Smart Cards and Hardware Security Modules (HSMs).
XAdESjs[12]	XAdES is short for "XML Advanced Electronic Signatures", it is a superset of XMLDSIG. This library aims to provide an implementation of XAdES in Typescript/Javascript that is built on XMLDSIGjs

AES - generateKey/exportKey(JWK)

In this example we are going to learn how to generate **AES** keys and export it into JWK format. In general we are going to perform the below steps.

1. First Generate the Random **Initial Vector** to be used if encryption/decryption is performed .
2. Invoke the **generateKey** method with required parameter
3. Mark this **crytokey** as extractable i.e the **true** flag
4. Define the Key Usage
5. Finally invoke the **exportKey** method using the **jwk** format.

AES-GCM - generateKey/exportKey

[9] https://github.com/PeculiarVentures/PKI.js
[10] https://github.com/PeculiarVentures/ASN1.js
[11] https://github.com/PeculiarVentures/pkcs11js
[12] https://github.com/PeculiarVentures/xadesjs

```
function AESGCM_EXPORT_KEYS() {
        var iv = crypto.getRandomValues(new Uint8Array(16));
        window.crypto.subtle.generateKey({
                name: "AES-GCM",
                length: 256, //can be  128, 192, or 256
            },
            true, // the key is extractable (i.e. can be used in exportKey)
            ["encrypt", "decrypt", "wrapKey", "unwrapKey"] //can "encrypt", "decrypt\
" or  "wrapKey", or "unwrapKey"
        ).then(function(aesgcmkey) {
            var secretKey = aesgcmkey;
            return crypto.subtle.exportKey("jwk", secretKey).then(function(result) {
            key = result;
            document.getElementById("kty").value = key.kty;
            document.getElementById("key_ops").value = key.key_ops;
            document.getElementById("key_algo").value = key.alg
            document.getElementById("key").value = btoa(key.k);
        }, failAndLog);
        })
        .catch(function(err) {
            console.error(err);
        });
    }
```

AES-CTR - generateKey/exportKey

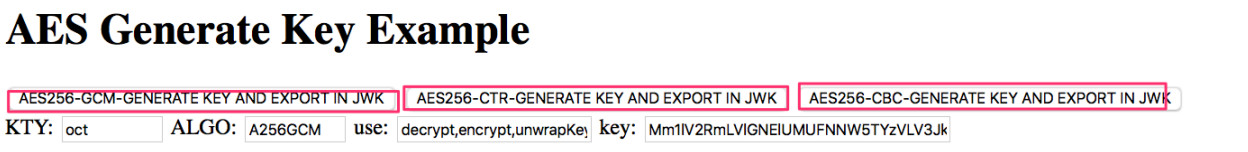
```
    function AESCTR_EXPORT_KEYS() {
        var iv = crypto.getRandomValues(new Uint8Array(16));
        window.crypto.subtle.generateKey({
                name: "AES-CTR",
                length: 256, //can be  128, 192, or 256
            },
            true, // the key is extractable (i.e. can be used in exportKey)
            ["encrypt", "decrypt", "wrapKey", "unwrapKey"] //can "encrypt", "decrypt\
" or  "wrapKey", or "unwrapKey"
        ).then(function(aesgcmkey) {
            var secretKey = aesgcmkey;
            return crypto.subtle.exportKey("jwk", secretKey).then(function(result) {
            key = result;
            document.getElementById("kty").value = key.kty;
            document.getElementById("key_ops").value = key.key_ops;
            document.getElementById("key_algo").value = key.alg
            document.getElementById("key").value = btoa(key.k);
```

```
        }, failAndLog);
        })
        .catch(function(err) {
            console.error(err);
        });
    }
```

AES-CBC - generateKey/exportKey

```
    function AESCBC_EXPORT_KEYS() {
        var iv = crypto.getRandomValues(new Uint8Array(16));
        window.crypto.subtle.generateKey({
                name: "AES-CBC",
                length: 256, //can be  128, 192, or 256
            },
            true, // the key is extractable (i.e. can be used in exportKey)
            ["encrypt", "decrypt", "wrapKey", "unwrapKey"] //can "encrypt", "decrypt\
" or  "wrapKey", or "unwrapKey"
        ).then(function(aesgcmkey) {
            var secretKey = aesgcmkey;
            return crypto.subtle.exportKey("jwk", secretKey).then(function(result) {
            key = result;
            document.getElementById("kty").value = key.kty;
            document.getElementById("key_ops").value = key.key_ops;
            document.getElementById("key_algo").value = key.alg
            document.getElementById("key").value = btoa(key.k);
        }, failAndLog);
        })
        .catch(function(err) {
            console.error(err);
        });
    }
```

The Demo Code example



aes-generate_key.html

AES - generateKey Encryption Decryption

In this example we are going to learn how to generate **AES** keys and then use this crypto key performing encryption and decryption. In general we are going to perform the below steps.

1. First Generate the Random **Initial Vector** to be used for encryption/decryption
2. Invoke **generateKey** Method, this example uses **AES-GCM** algo to derive the key
3. The **cryptokey** is not extractable and set to **false**
4. The **key** usage is set to perform only **encryption** and **decryption**

```
window.crypto.subtle.generateKey({
                name: "AES-GCM",
                length: 256, //can be  128, 192, or 256
            },
            false, //whether the key is extractable (i.e. can be used in exportKey)
            ["encrypt", "decrypt"] //can "encrypt", "decrypt" or  "wrapKey", or "unw\
rapKey"
        ).then(function(key) {
            secretKey = key;
        })
        .catch(function(err) {
            console.error(err);
        });
```

1. This is the example of performing AES encryption in GCM Mode
2. The Initial Vector is generated and passed to encrypt method
3. The input is converted into the Array
4. The encrypted text is displayed in **hexstring**

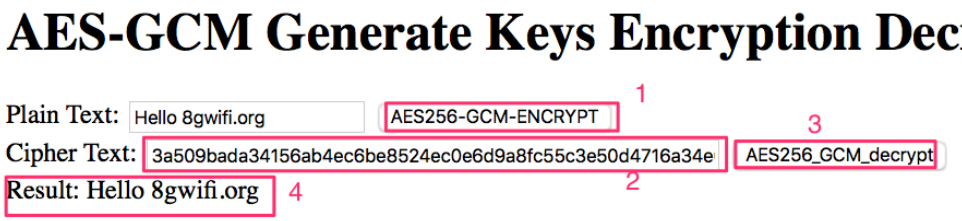
```
function AES256_GCM_ENCRYPT() {

        if (secretKey == null) {
            alert("Failed to Generate AES Keys Check The browser Comptabilty ")
            return;
        }
        var plainText = document.getElementById("plainTextGCM").value;
        return crypto.subtle.encrypt({
            name: "aes-gcm",
            iv: iv,
```

```
            additionalData: adddata,
            tagLength: 128 //can be 32, 64, 96, 104, 112, 120 or 128 (default)
        }, secretKey, asciiToUint8Array(plainText)).then(function(cipherText) {
            document.getElementById("cipherTextGCM").value = bytesToHexString(cipher\
Text);
        }, failAndLog);

    }
```

1. When performing **decryption** same **IV** and same **crytokey** is required.
2. Then invoking the decrypt method, with the same algorithm property that is used for encryption
3. The cipherText which is in hex string is converted back to ASCII String.

```
function AES256_GCM_decrypt() {

        if (secretKey == null) {
            alert("Failed to Generate AES Keys Check The browser Comptabilty ")
            return;
        }
        var cipherText = document.getElementById("cipherTextGCM").value;
        crypto.subtle.decrypt({
                name: "aes-gcm",
                iv: iv,
                additionalData: adddata,
                tagLength: 128 //can be 32, 64, 96, 104, 112, 120 or 128 (default)
            },
            secretKey,
            hexStringToUint8Array(cipherText)
        ).then(
            function(plainText) {
                document.getElementById("resultGCM").innerHTML = "Result: " + bytesT\
oASCIIString(plainText);
            },
            function(result) {
                document.getElementById("resultGCM").innerHTML = "Result: " + result;
            });
    }
```

The Demo Code example



aes-gcm-generate_key.html

AES - Encryption Decryption using Raw Key

In this example we are going to learn how to import raw **AES** keys and then use this key as crypto key to performing encryption and decryption. In general we are going to perform the below steps.

1. Convert the 128 bit hex key to Uint8Array
2. Generate random initial vector to be used for encryption and decryption
3. Perform encryption (AES-CTR,AES-GCM,AES-CBC)
4. Convert cipher-text into the hex
5. Perform decryption (AES-CTR,AES-GCM,AES-CBC)

The hardcoded **Raw** Key : This is for demo purpose only

```
var keyData = hexStringToUint8Array("1bf8be421b201adf247297140aa76cc8211620976125db5\
05d348cd60bee4198");
var iv = crypto.getRandomValues(new Uint8Array(16));
```

The **AES-CTR** encrypt Method

```
function AES_CTR_encrypt()
        {
            crypto.subtle.importKey("raw", keyData, "aes-ctr", false, ["encrypt"]).t\
hen(function(key) {
                var plainText = document.getElementById("plainTextGCM").value;
                return crypto.subtle.encrypt({name: "aes-ctr", counter: iv,length: 1\
28}, key, asciiToUint8Array(plainText));
            }, failAndLog).then(function(cipherText) {
                document.getElementById("cipherTextGCM").value = bytesToHexString(ci\
pherText);
            }, failAndLog);
        }
```

The **AES-GCM** encrypt Method

```
function AES_GCM_encrypt()
    {
        crypto.subtle.importKey("raw", keyData, "aes-gcm", false, ["encrypt"]).then(\
function(key) {
            var plainText = document.getElementById("plainTextGCM").value;
            return crypto.subtle.encrypt({name: "aes-gcm", iv: iv}, key, asciiToUint\
8Array(plainText));
        }, failAndLog).then(function(cipherText) {
            document.getElementById("cipherTextGCM").value = bytesToHexString(cipher\
Text);
        }, failAndLog);
    }
```

The **AES-GCM** encrypt Method

```
function AES_CBC_encrypt()
    {
        crypto.subtle.importKey("raw", keyData, "aes-cbc", false, ["encrypt"]).then(\
function(key) {
            var plainText = document.getElementById("plainTextGCM").value;
            return crypto.subtle.encrypt({name: "aes-cbc", iv: iv}, key, asciiToUint\
8Array(plainText));
        }, failAndLog).then(function(cipherText) {
            document.getElementById("cipherTextGCM").value = bytesToHexString(cipher\
Text);
        }, failAndLog);
    }
```

The **AES-GCM** decrypt Method

```
function AES_GCM_decrypt()
    {
        crypto.subtle.importKey("raw", keyData, "aes-gcm", false, ["decrypt"]).then(\
function(key) {
            var cipherText = document.getElementById("cipherTextGCM").value;
            return crypto.subtle.decrypt({name: "aes-gcm", iv: iv}, key, hexStringTo\
Uint8Array(cipherText));
        }, failAndLog).then(function(plainText) {
            document.getElementById("resultGCM").innerHTML = "Result: " + bytesToASC\
IIString(plainText);
        }, function(result) {
            document.getElementById("resultGCM").innerHTML = "Result: " + result;
```

```
        });
    }
```

The AES-CTR decrypt Method

```
function AES_CTR_decrypt()
        {
            crypto.subtle.importKey("raw", keyData, "aes-ctr", false, ["decrypt"]).t\
hen(function(key) {
                var cipherText = document.getElementById("cipherTextGCM").value;
                return crypto.subtle.decrypt({name: "aes-ctr", counter: iv, length:1\
28}, key, hexStringToUint8Array(cipherText));
            }, failAndLog).then(function(plainText) {
                document.getElementById("resultGCM").innerHTML = "Result: " + bytesT\
oASCIIString(plainText);
            }, function(result) {
                document.getElementById("resultGCM").innerHTML = "Result: " + result;
            });
        }
```

The AES-CBC decrypt Method

```
function AES_CBC_decrypt()
    {
        crypto.subtle.importKey("raw", keyData, "aes-cbc", false, ["decrypt"]).then(\
function(key) {
            var cipherText = document.getElementById("cipherTextGCM").value;
            return crypto.subtle.decrypt({name: "aes-cbc", iv: iv}, key, hexStringTo\
Uint8Array(cipherText));
        }, failAndLog).then(function(plainText) {
            document.getElementById("resultGCM").innerHTML = "Result: " + bytesToASC\
IIString(plainText);
        }, failAndLog);
    }
```

The Demo Code Example

AES-Encryption/Decryption using Fixed Key importKey

Click the corresponding buttons to do AES-CBC/GCM encryption/decryption. In the middle, try to modify the cipher text to see how AES-CBC/GCM responds.

Plain Text: Hello, 8gwifi.org encryptGCM encryptCBC encryptCTR

Cipher Text: 3068236e5f71ee94f2bdcc24bee9d415384d0c0780e1d35300826 decryptGCM decryptCBC decryptCTR

Result: Hello, 8gwifi.org

aes-encryption_importKey.html

ELGAMAL Encryption and Decryption

The example demonstrated in this code is performing elGamal encryption and decryption using the serialized public key and private as given elgamal input.

1. The ELGAMAL **encryption** is done with **public** key of a given plain text message
2. This demo assumes the serialized public key is using P-256 name curve. tune it for your need.

```
function performEncryption()
        {

        var elpublicKey = document.getElementById("elpublicKey").value;
        var plainText = document.getElementById("plainText").value;

        // Unserialized public key:
        var pub = new sjcl.ecc.elGamal.publicKey(
                  sjcl.ecc.curves.c256,
                  sjcl.codec.base64.toBits(elpublicKey)
                  )
        var ct = sjcl.encrypt(pub, plainText)
        document.getElementById("output").value = ct;
        }
```

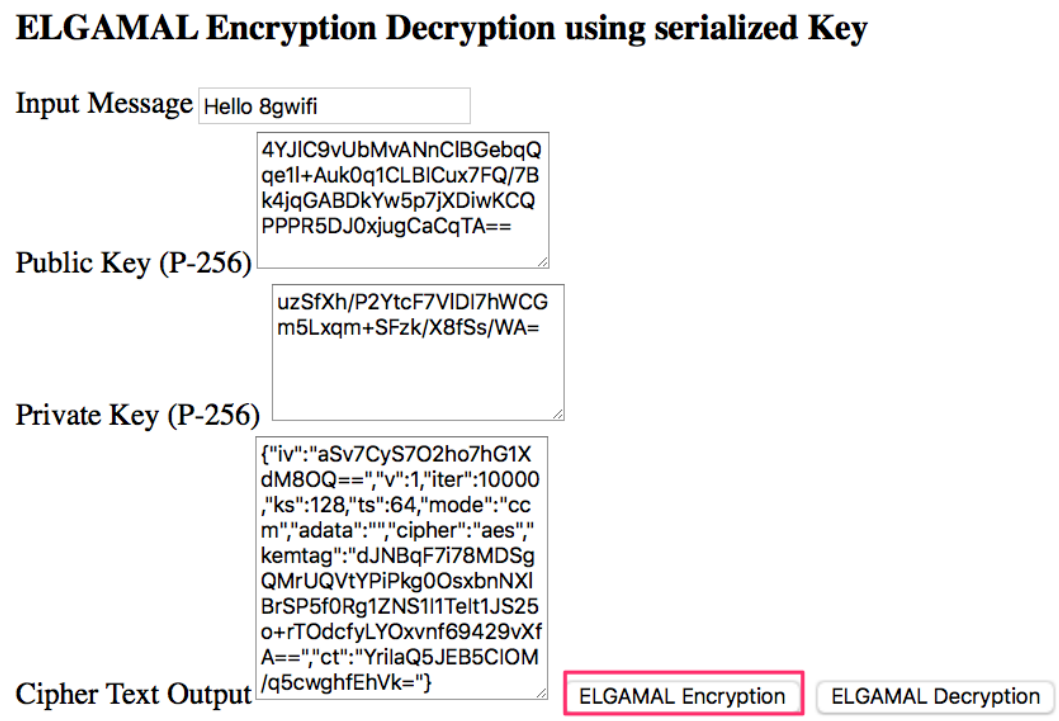
. The ELGAMAL **decryption** is done with **private** key of a given cipher text message

1. This demo assumes the serialized private key is using P-256 name curve.

```
function performDecryption()
        {
        var elprivateKey = document.getElementById("elprivateKey").value;
        var cipherText = document.getElementById("output").value;
        sec = new sjcl.ecc.elGamal.secretKey(
                  sjcl.ecc.curves.c256,
                  sjcl.ecc.curves.c256.field.fromBits(sjcl.codec.base64.toBits(elpri\
vateKey))

        )
        var pt = sjcl.decrypt(sec, cipherText)

        alert(pt);

    }
```

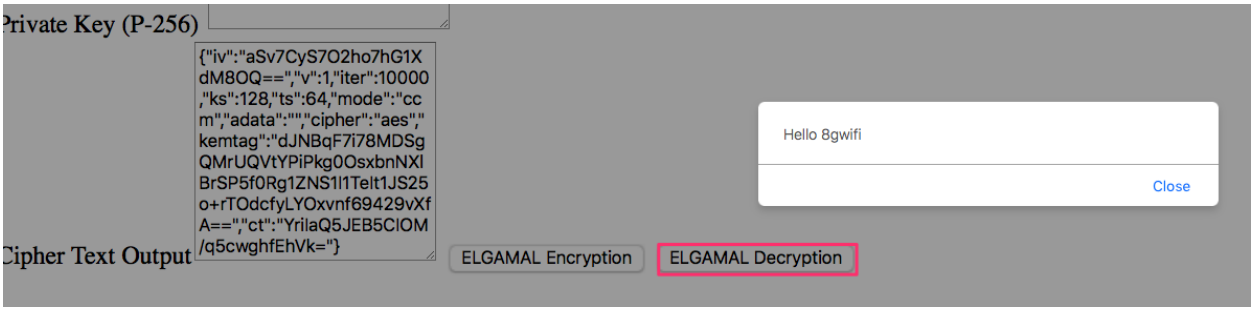
The Demo Code example

Perform encryption using elgamal serialized public keys



elgamal_encryption.html

Perform decryption using elgamal serialized private keys



elgamal_encryption.html