My favorites ▼ | Sign in

crypto-js

JavaScript implementations of standard and secure cryptographic algorithms

Search projects

Project Home   Downloads   Wiki   Issues   Source

Summary   People

Project Information

+19 Recommend this on Goog

Project feeds

Code license
New BSD License

Labels
Crypto, JavaScript, MD5, SHA-1, SHA-2, HMAC, PBKDF2, AES, TripleDES, Rabbit, RC4

Members
Jeff.Mott.OR
1 committer

Featured

Downloads
CryptoJS v3.0.zip
Show all »

Wiki pages
ChangeLog
License
SpecialThanksTo
Show all »

Links

Groups
CryptoJS Discussion Group

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.

If you have a problem with CryptoJS, if you want to discuss new features, or if you want to contribute to the project, you can visit the CryptoJS discussion group.

http://groups.google.com/group/crypto-js/topics

CryptoJS 3

- More algorithms.
- Progressive processing.
- More compatible with existing implementations.
- More adaptable to new and varied implementations.
- Better architecture under the hood.
- Faaaster.

Quick-start Guide

Hashers

The Hasher Algorithms

MD5

MD5 is a widely used hash function. It's been used in a variety of security applications and is also commonly used to check the integrity of files. Though, MD5 is not collision resistant, and it isn't suitable for applications like SSL certificates or digital signatures that rely on this property.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/md5.js"></script>
<script>
    var hash = CryptoJS.MD5("Message");
</script>
```

SHA-1

The SHA hash functions were designed by the National Security Agency (NSA). SHA-1 is the most established of the existing SHA hash functions, and it's used in a variety of security applications and protocols. Though, SHA-1's collision resistance has been weakening as new attacks are discovered or improved.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/sha1.js"></script>
<script>
    var hash = CryptoJS.SHA1("Message");
</script>
```

SHA-2

SHA-256 is one of the four variants in the SHA-2 set. It isn't as widely used as SHA-1, though it appears to provide much better security.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/sha256.js"></script>
<script>
    var hash = CryptoJS.SHA256("Message");
</script>
```

SHA-512 is largely identical to SHA-256 but operates on 64-bit words rather than 32.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/sha512.js"></script>
<script>
    var hash = CryptoJS.SHA512("Message");
</script>
```

CryptoJS also supports SHA-224 and SHA-384, which are largely identical but truncated versions of SHA-256 and SHA-512 respectively.

The Hasher Input

The hash algorithms accept either strings or instances of CryptoJS.lib.WordArray. A WordArray object represents an array of 32-bit words. When you pass a string, it's automatically converted to a WordArray encoded as UTF-8.

The Hasher Output

The hash you get back isn't a string yet. It's a WordArray object. When you use a WordArray object in a string context, it's automatically converted to a hex string.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/sha256.js"></script>
<script>
    var hash = CryptoJS.SHA256("Message");

    alert(typeof hash); // object

    alert(hash); // 2f77668a9dfbf8d5848b9eeb4a7145ca94c6ed9236e4a773f6dcafa5132b2f91
</script>
```

You can convert a WordArray object to other formats by explicitly calling the toString method and passing an encoder.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/sha256.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/components/enc-base64-min.js"></script>
<script>
    var hash = CryptoJS.SHA256("Message");

    alert(hash.toString(CryptoJS.enc.Base64)); // L3dmip37+NWEi57rSnFFypTG7ZI25Kdz9tyvpRMrL5E=

    alert(hash.toString(CryptoJS.enc.Latin1)); // /wf��ûøÕ���ëJqEÊ�Æí�6ä§söÜ¯¥+/�

    alert(hash.toString(CryptoJS.enc.Hex)); // 2f77668a9dfbf8d5848b9eeb4a7145ca94c6ed9236e4a773f6dcafa5132b2f91
</script>
```

Progressive Hashing

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/sha256.js"></script>
<script>
    var sha256 = CryptoJS.algo.SHA256.create();

    sha256.update("Message Part 1");
    sha256.update("Message Part 2");
    sha256.update("Message Part 3");

    var hash = sha256.finalize();
</script>
```

HMAC

Keyed-hash message authentication codes (HMAC) is a mechanism for message authentication using cryptographic hash functions.

HMAC can be used in combination with any iterated cryptographic hash function.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/hmac-md5.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/hmac-sha1.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/hmac-sha256.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/hmac-sha512.js"></script>
<script>
    var hash = CryptoJS.HmacMD5("Message", "Secret Passphrase");
    var hash = CryptoJS.HmacSHA1("Message", "Secret Passphrase");
    var hash = CryptoJS.HmacSHA256("Message", "Secret Passphrase");
    var hash = CryptoJS.HmacSHA512("Message", "Secret Passphrase");
</script>
```

Progressive HMAC Hashing

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/hmac-sha256.js"></script>
<script>
    var hmac = CryptoJS.algo.HMAC.create(CryptoJS.algo.SHA256, "Secret Passphrase");

    hmac.update("Message Part 1");
    hmac.update("Message Part 2");
    hmac.update("Message Part 3");

    var hash = hmac.finalize();
</script>
```

PBKDF2

PBKDF2 is a password-based key derivation function. In many applications of cryptography, user security is ultimately dependent on a password, and because a password usually can't be used directly as a cryptographic key, some processing is required.

A salt provides a large set of keys for any given password, and an iteration count increases the cost of producing keys from a password, thereby also increasing the difficulty of attack.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/pbkdf2.js"></script>
<script>
    var salt = CryptoJS.lib.WordArray.random(128/8);

    var key128Bits = CryptoJS.PBKDF2("Secret Passphrase", salt, { keySize: 128/32 });
    var key256Bits = CryptoJS.PBKDF2("Secret Passphrase", salt, { keySize: 256/32 });
    var key512Bits = CryptoJS.PBKDF2("Secret Passphrase", salt, { keySize: 512/32 });

    var key512Bits1000Iterations = CryptoJS.PBKDF2("Secret Passphrase", salt, { keySize: 512/32, iterations: 1000 });
</script>
```

Ciphers

The Cipher Algorithms

AES

The Advanced Encryption Standard (AES) is a U.S. Federal Information Processing Standard (FIPS). It was selected after a 5-year process where 15 competing designs were evaluated.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/aes.js"></script>
<script>
    var encrypted = CryptoJS.AES.encrypt("Message", "Secret Passphrase");

    var decrypted = CryptoJS.AES.decrypt(encrypted, "Secret Passphrase");
</script>
```

DES, Triple DES

DES is a previously dominant algorithm for encryption, and was published as an official Federal Information Processing Standard (FIPS). DES is now considered to be insecure due to the small key size.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/tripledes.js"></script>
<script>
    var encrypted = CryptoJS.DES.encrypt("Message", "Secret Passphrase");

    var decrypted = CryptoJS.DES.decrypt(encrypted, "Secret Passphrase");
</script>
```

Triple DES applies DES three times to each block to increase the key size. The algorithm is believed to be secure in this form.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/tripledes.js"></script>
<script>
    var encrypted = CryptoJS.TripleDES.encrypt("Message", "Secret Passphrase");

    var decrypted = CryptoJS.TripleDES.decrypt(encrypted, "Secret Passphrase");
</script>
```

Rabbit

Rabbit is a high-performance stream cipher and a finalist in the eSTREAM Portfolio. It is one of the four designs selected after a 3 1/2-year process where 22 designs were evaluated.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/rabbit.js"></script>
<script>
    var encrypted = CryptoJS.Rabbit.encrypt("Message", "Secret Passphrase");

    var decrypted = CryptoJS.Rabbit.decrypt(encrypted, "Secret Passphrase");
</script>
```

RC4, RC4Drop

RC4 is a widely-used stream cipher. It's used in popular protocols such as SSL and WEP. Although remarkable for its simplicity and speed, the algorithm's history doesn't inspire confidence in its security.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/rc4.js"></script>
<script>
    var encrypted = CryptoJS.RC4.encrypt("Message", "Secret Passphrase");

    var decrypted = CryptoJS.RC4.decrypt(encrypted, "Secret Passphrase");
</script>
```

It was discovered that the first few bytes of keystream are strongly non-random, and leak information about the key. We can defend against this attack by discarding the initial portion of the keystream. This modified algorithm is traditionally called RC4-drop.

By default, 192 words (768 bytes) are dropped, but you can configure the algorithm to drop any number of words.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/rc4.js"></script>
<script>
    var encrypted = CryptoJS.RC4Drop.encrypt("Message", "Secret Passphrase");

    var encrypted = CryptoJS.RC4Drop.encrypt("Message", "Secret Passphrase", { drop: 3072/4 });

    var decrypted = CryptoJS.RC4Drop.decrypt(encrypted, "Secret Passphrase", { drop: 3072/4 });
</script>
```

Custom Key and IV

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/aes.js"></script>
<script>
    var key = CryptoJS.enc.Hex.parse('000102030405060708090a0b0c0d0e0f');
    var iv = CryptoJS.enc.Hex.parse('101112131415161718191a1b1c1d1e1f');

    var encrypted = CryptoJS.AES.encrypt("Message", key, { iv: iv });
</script>
```

Block Modes and Padding

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/aes.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/components/mode-cfb-min.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/components/pad-ansix923-min.js"></script>
<script>
    var encrypted = CryptoJS.AES.encrypt("Message", "Secret Passphrase", { mode: CryptoJS.mode.CFB, padding: CryptoJS
</script>
```

CryptoJS supports the following modes:

- CBC (the default)
- CFB
- CTR
- OFB
- ECB

And CryptoJS supports the following padding schemes:

- Pkcs7 (the default)
- Iso97971
- AnsiX923
- Iso10126
- ZeroPadding
- NoPadding

The Cipher Input

For the plaintext message, the cipher algorithms accept either strings or instances of CryptoJS.lib.WordArray.

For the key, when you pass a string, it's treated as a passphrase to derive an actual key and IV. Or you can pass a WordArray that represents the actual key. If you pass the actual key, you must also pass the actual IV.

For the ciphertext, the cipher algorithms accept either strings or instances of CryptoJS.lib.CipherParams. A CipherParams object represents a collection of parameters such as the IV, a salt, and the raw ciphertext itself. When you pass a string, it's automatically converted to a CipherParams object according to a configurable format strategy.

The Cipher Output

The plaintext you get back after decryption is a WordArray object. See Hashers' Output for more detail.

The ciphertext you get back after encryption isn't a string yet. It's a CipherParams object. A CipherParams object gives you access to all the parameters used during encryption. When you use a CipherParams object in a string context, it's automatically converted to a string according to a format strategy. The default is an OpenSSL-compatible format.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/aes.js"></script>
<script>
    var encrypted = CryptoJS.AES.encrypt("Message", "Secret Passphrase");

    alert(encrypted.key);        // 74eb593087a982e2a6f5dded54ecd96d1fd0f3d44a58728cdcd40c55227522223
    alert(encrypted.iv);         // 7781157e2629b094f0e3dd48c4d786115
    alert(encrypted.salt);       // 7a25f9132ec6a8b34
    alert(encrypted.ciphertext); // 73e54154a15d1beeb509d9e12f1e462a0
    alert(encrypted);            // U2FsdGVkX1+iX5Ey7GqLND5UFUoV0b7rUJ2eEvHkYqA=
</script>
```

You can define your own formats in order to be compatible with other crypto implementations. A format is an object with two methods—stringify and parse—that converts between CipherParams objects and ciphertext strings.

Here's how you might write a JSON formatter:

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/aes.js"></script>
<script>
    var JsonFormatter = {
        stringify: function (cipherParams) {
            var jsonObj = {
                ct: cipherParams.ciphertext.toString(CryptoJS.enc.Base64)
            };
            if (cipherParams.iv) {
                jsonObj.iv = cipherParams.iv.toString();
            }
            if (cipherParams.salt) {
                jsonObj.s = cipherParams.salt.toString();
            }

            return JSON.stringify(jsonObj);
        },

        parse: function (jsonStr) {
            var jsonObj = JSON.parse(jsonStr);

            var cipherParams = CryptoJS.lib.CipherParams.create({
                ciphertext: CryptoJS.enc.Base64.parse(jsonObj.ct)
            });
            if (jsonObj.iv) {
                cipherParams.iv = CryptoJS.enc.Hex.parse(jsonObj.iv)
            }
            if (jsonObj.s) {
                cipherParams.salt = CryptoJS.enc.Hex.parse(jsonObj.s)
            }

            return cipherParams;
        }
    };

    var encrypted = CryptoJS.AES.encrypt("Message", "Secret Passphrase", { format: JsonFormatter });

    alert(encrypted); // {"ct":"tZ4MsEnfbcDOwqau68aOrQ==","iv":"8a8c8fd8fe33743d3638737ea4a00698","s":"ba06373c8f5717

    var decrypted = CryptoJS.AES.decrypt(encrypted, "Secret Passphrase", { format: JsonFormatter });

    alert(decrypted.toString(CryptoJS.enc.Utf8)); // Message
</script>
```

Progressive Ciphering

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/aes.js"></script>
<script>
    var key = CryptoJS.enc.Hex.parse('000102030405060708090a0b0c0d0e0f');
    var iv = CryptoJS.enc.Hex.parse('101112131415161718191a1b1c1d1e1f');

    var aesEncryptor = CryptoJS.algo.AES.createEncryptor(key, { iv: iv });

    var ciphertextPart1 = aesEncryptor.process("Message Part 1");
    var ciphertextPart2 = aesEncryptor.process("Message Part 2");
    var ciphertextPart3 = aesEncryptor.process("Message Part 3");
    var ciphertextPart4 = aesEncryptor.finalize();

    var aesDecryptor = CryptoJS.algo.AES.createDecryptor(key, { iv: iv });

    var plaintextPart1 = aesDecryptor.process(ciphertextPart1);
    var plaintextPart2 = aesDecryptor.process(ciphertextPart2);
    var plaintextPart3 = aesDecryptor.process(ciphertextPart3);
    var plaintextPart4 = aesDecryptor.process(ciphertextPart4);
    var plaintextPart5 = aesDecryptor.finalize();
</script>
```

Interoperability

With OpenSSL

Encrypt with OpenSSL:

```
openssl enc -aes-256-cbc -in infile -out outfile -pass pass:"Secret Passphrase" -e -base64
```

Decrypt with CryptoJS:

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/rollups/aes.js"></script>
<script>
    var decrypted = CryptoJS.AES.decrypt(openSSLEncrypted, "Secret Passphrase");
</script>
```

Encoders

CryptoJS can convert from encoding formats such as Base64, Latin1 or Hex to WordArray objects and vica versa.

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/components/core-min.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/components/enc-utf16-min.js"></script>
<script src="http://crypto-js.googlecode.com/svn/tags/3.0/build/components/enc-base64-min.js"></script>
<script>
    var words = CryptoJS.enc.Base64.parse('SGVsbG8sIFdvcmxkIQ==');
    var base64 = CryptoJS.enc.Base64.stringify(words);

    var words = CryptoJS.enc.Latin1.parse('Hello, World!');
    var latin1 = CryptoJS.enc.Latin1.stringify(words);

    var words = CryptoJS.enc.Hex.parse('48656c6c6f2c20576f726c6421');
    var hex = CryptoJS.enc.Hex.stringify(words);

    var words = CryptoJS.enc.Utf8.parse('□');
    var utf8 = CryptoJS.enc.Utf8.stringify(words);

    var words = CryptoJS.enc.Utf16.parse('Hello, World!');
    var utf16 = CryptoJS.enc.Utf16.stringify(words);
</script>
```