CRYPTOGRAPHY

View online: https://www.construct.net/en/make-games/manuals/construct-3/plugin-reference/cryptography

The **Cryptography** object can perform cryptographic operations such as encryption and hashing.

Scripting

This object has no script interface, because when using JavaScript or TypeScript coding you can use the browser built-in Web Crypto API.

Encryption does not guarantee security

Please read this section carefully if you intend to use encryption in your project.

The Cryptography object can perform password-based encryption and decryption. This is intended for both educational purposes to help illustrate the principles of encryption, and also as a means to make it harder, although not impossible, to make unwanted modifications to sensitive data such as save game data (which could for example allow cheating).

Merely using encryption **does not mean your project is secure.** Designing truly secure systems involves careful design of the overall system and is usually done by people with training and expertise, and may well need to involve a server to host the sensitive data out of reach of clients. In particular, if you enter a long and supposedly secure password for the *Encrypt* action in the event sheet, then that password will appear in plain text in your project's exported data files. A sufficiently motivated person will likely be able to identify that password in the exported files and then be able to view or modify any data encrypted with the same password. Even if you use a dynamically generated password so that the password does not appear in the project's exported data files, a more advanced adversary will still be able to use debugging tools to identify the password at the moment it is used to encrypt data.

In short you should see encryption as a tool that makes it somewhat harder to view or modify data, but certainly not impossible. In some cases this is still sufficient. For example if you are concerned about people using browser developer tools to casually modify save game data and cheat, encrypting the data may be sufficient to deter casual modification, even if more skilled adversaries are still capable of it.

HTTPS is already encrypted

Note that if you want to send data to a server securely, the best approach is to use a secure connection over HTTPS. This is already encrypted, and so there is no need to use the Cryptography object to encrypt data sent over a secure connection.

Processing text

Since encrypted data is usually binary data that is not easily representible as text, the Cryptography object generally works with the Binary Data object for both its input and output. However this does not preclude the ability to use text: after all, text is just another kind of binary data. You can store text in a Binary Data object with the *Set from text* action, and retrieve text from it with the *GetAllText* expression. Using these allows performing tasks like encrypting and decrypting text or hashing text.

You can also reliably display and transmit binary data in a text format by encoding it as Base64 using both the *Set from base64* action and *GetBase64* expression.

Examples

See the Encryption example for a basic demonstration of encryption and decryption of a message. The Hashing example also demonstrates hashing text or files.

Encrypted data format

For advanced users who wish to interoperate encrypted data with other code or services, the encrypted binary data contains a small amount of metadata added at the start by Construct to aid in decryption. The format of the data is described below.

Bytes	Description
0-1	Reserved (must be 0)
1-17	Salt (16 bytes)
17-29	Initialization vector (aka IV, 12 bytes)
29-33	Iterations (uint32, 4 bytes)
33+	Encrypted data payload

Cryptography conditions

On encryption finished

On encryption failed

On any encryption finished

On any encryption failed

Triggered after an encryption action depending on whether the operation completed successfully. A tag can be specified to distinguish different encryption operations. The "any" variants always trigger regardless of the tag, and the associated tag can be retrieved with the *Tag* expression.

On decryption finished

On decryption failed

On any decryption finished

On any decryption failed

Triggered after a decryption action depending on whether the operation completed successfully. Decryption will fail if the incorrect password is specified. A tag can be specified to distinguish different decryption operations. The "any" variants always trigger regardless of the tag, and the associated tag can be retrieved with the *Tag* expression.

On hash finished

On any hash finished

Triggered after a hash action when the operation completes. A tag can be specified to distinguish different hashing operations. The "any" variant always triggers regardless of the tag, and the associated tag can be retrieved with the *Tag* expression.

Cryptography actions

Encrypt binary

Encrypt the contents of a given Binary Data object. The same provided password must be specified to successfully decrypt the data. The *Iterations* parameter indicates how many times to repeat a hash function when generating an encryption key; higher values use more processing power and so slow down encryption and decryption, but make it harder to use a brute-force attack to decrypt the data. An optional tag can be specified to distinguish different encryption operations. When the data has finished being encrypted, *On encryption finished* triggers.

> *Using encryption does not guarantee your project is secure. See the section Encryption does not guarantee security above.*

Decrypt binary

Decrypt the contents of a given Binary Data object. Decryption will only succeed if the same password that was used to encrypt the data is provided. An optional tag can be specified to distinguish different decryption operations. When the data has finished being decrypted, *On decryption finished* triggers; if decryption fails, including due to having the wrong password, then *On decryption failed* will trigger instead.

Hash binary

Compute a cryptographic hash function for the contents of a given Binary Data object. Currently supported hash functions include SHA-256, SHA-384 and SHA-512 from the SHA-2 family of functions. An optional tag can be specified to distinguish different hasing operations. When the hashing completes, *On hash finished* will trigger and the resulting hash will be available as a hexadecimal string in the *Hash* expression.

Cryptography expressions

Hash

After *On hash finished* triggers, a hexadecimal string of the resulting hash. For example the SHA-256 hash of the string "Construct" is `67e7c6e28e540b4fd412ad663b634a38572b98d3f4608fc5792f2600d32c1eb9`.

Tag

In a trigger such as *On any decryption finished* or *On any hash finished*, this returns the associated tag of the action that resulted in the trigger.

RandomUUID

Returns a random v4 UUID as a string using a cryptographically secure random number generator, e.g. `36b8f84d-df4e-4d49-b662-bcde71a8764f`.