/\*\*

\* Author: Peter Dragicevic [peter@petschko.org]

\* Authors-Website: http://petschko.org/

\* Date: 30.03.2017

\* Time: 23:05

\*/

/\*\*

\* Creates a new instance of the Decrypter Object

\*

\* @param {string|null} encryptionKey - Encryption-Key

\* @constructor - Decrypter

\*/

function Decrypter(encryptionKey) {

// Encryption-Fields

this.encryptCode = encryptionKey;

// Option Fields

this.ignoreFakeHeader = false;

// Fake-Header Info-Fields

this.headerLen = null;

this.signature = null;

this.version = null;

this.remain = null;

// Setup how long the PNG-Header is

this.pngHeaderLen = null;

/\*\*

\* Splits the Encryption-Code into an Array

\*

\* @returns {Array} - Encryption-Array

\*/

Decrypter.prototype.splitEncryptionCode = function() {

if(! this.encryptCode)

return [];

return this.encryptCode.split(/(.{2})/).filter(Boolean);

};

this.encryptionCodeArray = this.splitEncryptionCode();

/\*\*

\* Check if the current File-Header matches the Fake-Header

\*

\* @param {Uint8Array} fileHeader - Current-File-Header

\* @returns {boolean} - true if header matches fake header else false

\*/

Decrypter.prototype.verifyFakeHeader = function(fileHeader) {

var fakeHeader = this.buildFakeHeader();

for(var i = 0; i < this.getHeaderLen(); i++)

if(fileHeader[i] !== fakeHeader[i])

return false;

return true;

};

/\*\*

\* Builds the Fake-Header

\*

\* @returns {Uint8Array} - Fake-Header-Array

\*/

Decrypter.prototype.buildFakeHeader = function() {

var fakeHeader = new Uint8Array(this.getHeaderLen());

var headerStructure = this.getSignature() + this.getVersion() + this.getRemain();

for(var i = 0; i < this.getHeaderLen(); i++)

fakeHeader[i] = parseInt('0x' + headerStructure.substr(i \* 2, 2), 16);

return fakeHeader;

};

/\*\*

\* Do something with a RPGFile

\*

\* @param {RPGFile} rpgFile - RPGFile Object

\* @param {string} modType - Specify what to do with that file

\* @param {function} callback - Function if operation is done

\*/

Decrypter.prototype.modifyFile = function(rpgFile, modType, callback) {

var reader = new FileReader();

var that = this;

reader[window.addEventListener ? 'addEventListener' : 'attachEvent']

(window.addEventListener ? 'load' : 'onload', function() {

console.log('Try to ' + modType + ' the File "' + rpgFile.name + '.' + rpgFile.extension + '...');

switch(modType) {

case 'restore':

try {

rpgFile.content = that.restorePngHeader(this.result);

rpgFile.createBlobUrl();

} catch(e) {

callback(rpgFile, e);

return;

}

break;

case 'encrypt':

try {

rpgFile.content = that.encrypt(this.result);

rpgFile.createBlobUrl()

} catch(e) {

callback(rpgFile, e);

return;

}

break;

case 'decrypt':

default:

try {

rpgFile.content = that.decrypt(this.result);

rpgFile.createBlobUrl();

} catch(e) {

callback(rpgFile, e);

return;

}

}

callback(rpgFile, null);

}, false);

reader.readAsArrayBuffer(rpgFile.file);

};

/\*\*

\* Restores the header of an encrypted PNG-File without the key

\*

\* @param {ArrayBuffer} arrayBuffer - Array-Buffer of the File

\* @returns {ArrayBuffer} - Array-Buffer of the restored File

\*/

Decrypter.prototype.restorePngHeader = function(arrayBuffer) {

if(! arrayBuffer)

throw new ErrorException('File is empty or can\'t be read by your Browser...', 1);

var headerLen = (this.pngHeaderLen === null) ? this.getHeaderLen() : this.pngHeaderLen;

var pngStartHeader = Decrypter.getNormalPNGHeader(headerLen);

// Make sure that to long header values get the correct one

headerLen = pngStartHeader.length;

// Remove Fake-Header and encrypted Bytes

arrayBuffer = arrayBuffer.slice(headerLen \* 2, arrayBuffer.byteLength);

// Merge the file-content with the header

var tmpInt8Arr = new Uint8Array(arrayBuffer.byteLength + headerLen);

tmpInt8Arr.set(pngStartHeader, 0);

tmpInt8Arr.set(new Uint8Array(arrayBuffer), headerLen);

return tmpInt8Arr.buffer;

};

/\*\*

\* Decrypts a RPG-Make-File-ArrayBuffer & may check the header if turned on

\*

\* @param {ArrayBuffer} arrayBuffer - Array-Buffer of the File

\* @returns {ArrayBuffer} - Decrypted Array-Buffer of the File without the Fake-Header

\*/

Decrypter.prototype.decrypt = function(arrayBuffer) {

if(! arrayBuffer)

throw new ErrorException('File is empty or can\'t be read by your Browser...', 1);

if(! this.ignoreFakeHeader) {

var header = new Uint8Array(arrayBuffer, 0, this.getHeaderLen());

if(! this.verifyFakeHeader(header))

throw new ErrorException(

'Fake-Header don\'t matches the Template-Fake-Header. Make sure, that you use an Encrypted File.' +

' - If you do, turn off "Fake-Header"-Check and try again.',

2

);

}

// Remove the Fake-Header from File-arrayBuffer

arrayBuffer = arrayBuffer.slice(this.getHeaderLen(), arrayBuffer.byteLength);

// Decrypt File beginning

arrayBuffer = this.xOrBytes(arrayBuffer);

return arrayBuffer;

};

/\*\*

\* (Re)-Encrypt a RPG-Make-File-ArrayBuffer

\*

\* @param {ArrayBuffer} arrayBuffer - Array-Buffer of the File

\* @returns {ArrayBuffer} - Encrypted Array-Buffer with the Fake-Header

\*/

Decrypter.prototype.encrypt = function(arrayBuffer) {

if(! arrayBuffer)

throw new ErrorException('File is empty or can\'t be read by your Browser...', 1);

// Encrypt the File beginning

arrayBuffer = this.xOrBytes(arrayBuffer);

// Create Header

var fakeHeader = this.buildFakeHeader();

// Add Fake-Header in Front then the File

var tmpInt8Array = new Uint8Array(arrayBuffer.byteLength + this.getHeaderLen());

tmpInt8Array.set(fakeHeader, 0);

tmpInt8Array.set(new Uint8Array(arrayBuffer), this.getHeaderLen());

// Check if header is valid

var header = new Uint8Array(tmpInt8Array.buffer, 0, this.getHeaderLen());

if(! this.verifyFakeHeader(header))

throw new ErrorException(

'Fake-Header don\'t matches the Template-Fake-Header... Please report this Bug',

3

);

return tmpInt8Array.buffer;

};

/\*\*

\* XOR x Bytes (x = header-length-bytes)

\*

\* @param {ArrayBuffer} arrayBuffer - Array-Buffer where bytes should be XORed

\* @returns {ArrayBuffer} - Array-Buffer with XORed Bytes

\*/

Decrypter.prototype.xOrBytes = function(arrayBuffer) {

var view = new DataView(arrayBuffer);

if(arrayBuffer) {

var byteArray = new Uint8Array(arrayBuffer);

for(var i = 0; i < this.getHeaderLen(); i++) {

byteArray[i] = byteArray[i] ^ parseInt(this.encryptionCodeArray[i], 16);

view.setUint8(i, byteArray[i]);

}

}

return arrayBuffer;

}

}

// Class constants

Decrypter.prototype.defaultHeaderLen = 16;

Decrypter.prototype.defaultSignature = "5250474d56000000";

Decrypter.prototype.defaultVersion = "000301";

Decrypter.prototype.defaultRemain = "0000000000";

Decrypter.pngHeaderBytes = '89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52';

/\*\*

\* Returns the Header Len

\*

\* @returns {int} - Header-Len

\*/

Decrypter.prototype.getHeaderLen = function() {

if(this.headerLen === null || typeof this.headerLen !== 'number')

this.headerLen = this.defaultHeaderLen;

// Ensure int

return Math.floor(this.headerLen);

};

/\*\*

\* Returns the Signature

\*

\* @returns {string} - Signature

\*/

Decrypter.prototype.getSignature = function() {

if(this.signature === null)

this.signature = this.defaultSignature;

return this.signature;

};

/\*\*

\* Returns the Version

\*

\* @returns {string} - Version

\*/

Decrypter.prototype.getVersion = function() {

if(this.version === null)

this.version = this.defaultVersion;

return this.version;

};

/\*\*

\* Returns the Remain

\*

\* @returns {string} - Remain

\*/

Decrypter.prototype.getRemain = function() {

if(this.remain === null)

this.remain = this.defaultRemain;

return this.remain;

};

/\*\*

\* Restores a RPGMVP-File without key

\*

\* @param {RPGFile} rpgFile - RPGFile to Decrypt

\* @param {function} callback - Function if operation is done

\*/

Decrypter.prototype.restoreHeader = function(rpgFile, callback) {

this.modifyFile(rpgFile, 'restore', callback);

};

/\*\*

\* Decrypts a RPGFile

\*

\* @param {RPGFile} rpgFile - RPGFile to Decrypt

\* @param {function} callback - Function if operation is done

\*/

Decrypter.prototype.decryptFile = function(rpgFile, callback) {

this.modifyFile(rpgFile, 'decrypt', callback);

};

/\*\*

\* Encrypts a RPGFile

\*

\* @param {RPGFile} rpgFile - RPGFile to Decrypt

\* @param {function} callback - Function if operation is done

\*/

Decrypter.prototype.encryptFile = function(rpgFile, callback) {

this.modifyFile(rpgFile, 'encrypt', callback)

};

/\*\*

\* Get the normal starting PNG-Header

\*

\* @param {int} headerLen - Header Length

\* @returns {Uint8Array} - Original starting PNG-Header

\*/

Decrypter.getNormalPNGHeader = function(headerLen) {

var headerToRestore = Decrypter.pngHeaderBytes.split(' ');

// Header can't be longer than restore string

if(headerLen > headerToRestore.length)

headerLen = headerToRestore.length;

var restoredHeader = new Uint8Array(headerLen);

for(var i = 0; i < headerLen; i++)

restoredHeader[i] = parseInt('0x' + headerToRestore[i], 16);

return restoredHeader;

};

/\*\*

\* Detect the Encryption-Code from a RPGFile

\*

\* @param {RPGFile} rpgFile - RPGFile Object

\* @param {int} headerLen - Header-Length

\* @param {function} callback - Function if operation is done

\*/

Decrypter.detectEncryptionCode = function(rpgFile, headerLen, callback) {

var reader = new FileReader();

reader[window.addEventListener ? 'addEventListener' : 'attachEvent']

(window.addEventListener ? 'load' : 'onload', function() {

var key;

var fileContent;

// Try to get the key from Image

if(rpgFile.isEncryptedImg()) {

key = Decrypter.getKeyFromPNG(headerLen, this.result); // todo replace header len

if(key !== null) {

callback(key);

return;

}

}

var fileContentAsText = new TextDecoder().decode(new Uint8Array(this.result));

try {

fileContent = JSON.parse('[' + fileContentAsText + ']');

key = fileContent[0].encryptionKey;

} catch(e) {

// Try if it is LZ-String compressed

var lzUncompressed = LZString.decompressFromBase64(fileContentAsText);

try {

fileContent = JSON.parse('[' + lzUncompressed + ']');

key = fileContent[0].encryptionKey;

} catch(e) {

// Still no JSON-File...

key = null;

}

}

// Try a search

if(key === null)

key = Decrypter.searchEncryptionCode(fileContentAsText, 'rpg\_core', false);

callback(key);

}, false);

reader.readAsArrayBuffer(rpgFile.file);

};

/\*\*

\* Gets the Key from PNG File

\*

\* @param {int} headerLen - Header Length

\* @param {ArrayBuffer} fileContent - File-Content which should be tried to get the key from

\* @return {string|null} - Key or null of not found/valid

\*/

Decrypter.getKeyFromPNG = function(headerLen, fileContent) {

if(! fileContent)

return null;

if(fileContent.byteLength < headerLen \* 2)

return null;

var fileHeader = fileContent.slice(headerLen, headerLen \* 2);

var fileHeaderU8 = new Uint8Array(fileHeader);

var maybeKeyBytes = new Uint8Array(headerLen);

var pngHeaderU8 = Decrypter.getNormalPNGHeader(headerLen);

// Get the potential Key

var i;

var tmpByte;

var output = '';

for(i = 0; i < headerLen; i++) {

maybeKeyBytes[i] = pngHeaderU8[i] ^ fileHeaderU8[i];

output += '' + Decrypter.byteToHex(maybeKeyBytes[i]);

}

return output;

};

/\*\*

\* Searches for the encryption-key in other places

\*

\* @param {string} fileContent - Content of the File, where to search

\* @param {string} searchParam - What method should be used to search

\* @param {boolean} lzString - Decompress LZ-String

\* @returns {null|string} - null if the key was not found else the key

\*/

Decrypter.searchEncryptionCode = function(fileContent, searchParam, lzString) {

var result = null;

fileContent = (lzString) ? LZString.decompressFromBase64(fileContent) : fileContent;

// Exit on empty File-Content (Usually caused if LZ-String-Decompress was not an LZ-String)

if(! fileContent)

return null;

switch(searchParam) {

case 'rpg\_core':

var innerFunctionCodeMatches = null;

var lines = fileContent.split('\n');

for(var line = 0; line < lines.length; line++) {

var l = lines[line];

// Clean the line

l = l.trim();

l = l.replace(/[\r\n\t]/g, '');

innerFunctionCodeMatches = l.match(/^(.\*)this\.\_encryptionKey ?= ?"(.\*)"(.\*);(.\*)?$/);

if(innerFunctionCodeMatches !== null)

break;

}

if(innerFunctionCodeMatches && innerFunctionCodeMatches.length > 2)

result = innerFunctionCodeMatches[2];

// Verify result

if(! result || typeof result === 'undefined')

result = null;

// Also try a LZ-String search

if(result === null && ! lzString)

result = Decrypter.searchEncryptionCode(fileContent, searchParam, true);

return result;

default:

return null;

}

};

/\*\*

\* Converts a Byte to a Hex string

\*

\* @param {int} byte - Byte Value

\* @returns {string} - Hex Value

\*/

Decrypter.byteToHex = function(byte) {

return ('0' + (byte & 0xFF).toString(16)).slice(-2);

};

/\*\*

\* Check if the string only has HEX-Chars (0-9 & A-F)

\*

\* @param {string} string - String to verify

\* @returns {boolean} - true if string is valid else false

\*/

Decrypter.checkHexChars = function(string) {

var regex = new RegExp(/^[A-Fa-f0-9]+$/);

return regex.test(string);

};

/\*\*

\* Converts a Byte to Bits

\*

\* @param {number} byte - Byte

\* @returns {string} - Bits

\*/

Decrypter.helperShowBits = function(byte) {

if(isNaN(byte))

byte = 0;

if(byte > 255 || byte < 0)

throw 'Invalid Byte-Value (' + byte + ')';

var bits = byte.toString(2);

var missingZeros = 8 - bits.length;

return new Array(missingZeros + 1).join('0') + bits;

};