Config encryption

Organization

Before speaking of encryption, let's talk quickly about config file organization.

As you know, you can include as many config files you want from a single file (cf. config includes).
In addition to allowing an easy development process (gathering config by type, for example), it is also very useful when you release your game and you want the end-user to be able to tweak some parts of the config but not all of them.

For example, in the main config file, you can include a file that doesn't exist during the development (the include will be simply ignored), but that can be used by the end-user to store its own config values.

For example, here's a simple scheme that can be used in the main config file.

```
; First we define here all our default settings for values that can be overidden by the end-user
[...]

; We now include the end-user modifiable file that will be stored in config.ini in this example
@config.ini@

; Finally we include the file containing all the settings that we don't want to end user to mess up with
; If the end-user did provide values for them, they'll be overidden here anyway
@encrypted.ini@
```

In the same way, we can add included files for customized inputs, for example. We would add it just after the inclusion of config.ini.
If the file exist it will be loaded, otherwise, if there's no custom saved controles, it will simply be ignored.
As a reminder, you can save the whole input settings by calling:

```
orxInput_Save("FileName");
```

So in the end, we would have this kind of scheme for our main config file.

```
; Default settings that are allowed to be overridden by end-user
[...]

; End-user custom settings
@config.ini@

; End-user custom inputs
```

```
@inputs.ini@

; Non-overridable config stuff (ie. everything else)
@encrypted.ini@
```

And, of course, you might not want the end-user to mess up with this file, so we now need to encrypt it when releasing our game.

Encryption

Let's clear something about the encryption itself right now: it is *not* a safe encryption. It merely is a visual encryption that prevents human reading of your file. But it can be easily decrypted by someone that has your encryption key as it's a very fast, non-robust encryption.

Well, now that we've said that, let's see how we can use it! 😀

First of all, we need an encryption key (or pass phrase). The longer, the better. If you don't provide any, orx's default one will be used, so you might want to change it for one of yours so that others could not (too easily) decrypt your stuff. 😉

To set the encryption key, you need to call:

```
orxConfig_SetEncryptionKey("MyVeryLongEncryptionKey");
```

Of course, as orx will load its main config file **when** it is initialized, you need to issue the call to `orxConfig_SetEncryptionKey()` before initializing orx. It would look something like this:

```
int main(int argc, char **argv)
{
  // Sets our encryption key
  orxConfig_SetEncryptionKey("MyVeryLongEncryptionKey");

  // Executes orx for our game
  orx_Execute(argc, argv, MyInit, MyRun, MyExit);

  // Done!
  return EXIT_SUCCESS;
}
```

But one question remains: how can we encrypt our config files? There are two possible answers: either doing it by code (calling `orxConfig_Save("FileName", orxTRUE, MyFilterCallback)`) or by using the command line tool called orxCrypt. 😀

orxCrypt

orxCrypt is a command line tool available for all supported development platforms. [1)]

It can be used to merge multiple config files into a single one and perform encryption/decryption if requested.

orxCrypt accepts a number of command line parameters:

- [MANDATORY] a list of input files
- [OPTIONAL] a name for the merged output file
- [OPTIONAL] a key for encryption/decryption
- [OPTIONAL] a switch to save output in a non-encrypted way (by default output'll be encrypted)

Here's its syntax:

```
orxcrypt -f InputFile [+ ...] [-o OutputFile] [-k EncryptionKey] [-d]
```

You can display its help with

```
orxcrypt -h
```

For any parameter, help can be diplayed using its long name:

```
orxcrypt -h ParameterLongName
```

Let's now see the parameters in details.

Input file list

-f / --filelist

The file list is mandatory. At least one file has to be provided and multiple files have to be separated by spaces. Unfortunately input files can't include spaces for now.

Syntax:

```
-f InputFile1 [InputFile2 ... InputFileN]
```

If input files are encrypted with a user-provided key, you need to pass it to orxcrypt using its encryption key parameter.

Encryption key

-k / --key

The encryption key parameter is optional. If it isn't provided, orx's default one will be used. The key is used for both decryption and encryption. OrxCrypt will figure by itself if your input files are encrypted

or not.

Syntax:

```
-k EncryptionKey
```

*NB: If your encryption key contains spaces, you won't be able to provide it via the command line. In this case, you'll have to provide it through orxCrypt's config file (*orxcrypt.ini*):*

```
[Param]
key = My encryption key contains spaces and can be provided here
```

Output file

-o / --output

The output parameter is optional. If none is provided, the merged output will be stored in `orxcrypt.out`.

Syntax:

```
-o OutputFile
```

Decryption

-d / --decrypt

By default orxCrypt will encrypt the merged file, either by using the encryption key the user provided or by orx's default one.
However, if you want the output to be stored in a human readable format (ie. non-encrypted), you'll have to provide this parameter.

NB: You can still provide a custom encryption key if one or many of your input files are encrypted with that key.

Syntax:

```
-d
```

[1)]
Windows, Linux, Mac OS X

From:
https://www.orx-project.org/wiki/ - **Orx Learning**

Permanent link:
https://www.orx-project.org/wiki/en/orx/config/encryption

Last update: **2025/09/30 17:26 (14 months ago)**