CodeCrypter and MetaCode Files **Q** & **A** (FAQ)

Q. What steps do I need to take to get my script encrypted?
A. First insert "#include MCFinclude.au3" in your script immediately above where you want the encryption to start. Then run CodeScanner on it, and make sure there are no major errors in your code (like missing #includes, for example). If your script is okay and the CodeScanner switch WriteMetaCode wasn't already on, switch it on and run a second scan. The MCF# basis files and arrays will now be written out. Afterwards, close CodeScanner and start CodeCrypter. Under the <Main> Tab, tick options <Write MCF0> and <BackTranslate>. Press the <Source> button and load your original script's name, then press <Run>. When Codecrypter completes without errors, a new file MCF0test.au3 will have been created in the same directory as your original script. Make sure it performs exactly as your original version. If so, untick the original checkboxes in CodeCrypter and enable Encrypt instead. You may also wish to set specific Encryption options (such as which key ID(s) to use!) under the <Encrypt> Tab. Finally, press <Run>, and if all goes well, a new MCF0test.au3 is produced. This is the encrypted version of your script. Test it thoroughly again. If there are any errors or other issues, see below.

Q. Do I need to press DataDump in CodeScanner, after running a scan, to write out the files for MCF or CodeCrypter?
A. No, just enable *WriteMetaCode* in the Settings panel before you (re)scan; the relevant data will be written out automatically. Activating DataDump without this setting enabled will not write out the MCF#.txt files, just the arrays (including many you don't need for MCF).

Q. I ran CodeScanner on my target script, but I cannot find the MetaCode output. Where is it?
A. CodeScanner only writes out MCF files when *WriteMetaCode* in the Settings panel is enabled. It then creates a subdirectory below your target file's directory called <targetfilename.au3>.CS_DATA to store those files. This subdirectory is often referred to as "CodeScanner's datadump directory."

Q. CodeScanner's Report complains that it cannot find include file AES.au3 or CryptoNG.au3. Where is it?
A. It is wherever ***you*** put it. AES.au3 and CryptoNG.au3 should be placed in the same directory as MCFinclude.au3 and your target script (or, if you put it elsewhere, insert its explicit path in the #include directive in MCFinclude.au3).

Q. There's something wrong with AES.au3; when I run Ward's test programme, Scite complains about several undefined Global Constants. Can this be fixed?
A. Yes, very easily. This can occur if you download AES.au3 separately, rather than using the patched version in the CodeScannerCrypter bundle. The problem occurs because AutoIt has changed since Ward wrote the original code. You'll find the instructions how to patch it in the MCFinclude.au3 remarks. It's a single

line where you have to get rid of the *Enum* keyword and assign the values 0,1,2 respectively to the three global constants.
Note: Ward has graciously allowed a patched version to be included in the codeScannerCrypter bundle, so this problem should no longer occur.

Q. Do I create MCF0 (the "Single Build") from CodeScanner or from CodeCrypter? What's the difference?
A. CodeScanner and CodeCrypter both call the same MCF function (called: _CreateSingleBuild), but CodeScanner provides only the default options. If you want more control over how much original code is kept in MCF0.txt, use CodeCrypter instead, and select options under the Tab "Single-Build."

Q. I ran CodeScanner and created MCF0.txt from there. Now do I load MCF0.txt in Codecrypter?
A. No, you specify the same target file in CodeCrypter and CodeScanner. Codecrypter then looks for CodeScanner's datadump subdirectory based on that filename. CodeCrypter doesn't actually load your original target file at all, it just needs its name to find the associated CS_data subdirectory.

Q. Sometimes when I run BackTranslate, it takes much longer than usual, and displays update messages it usually doesn't. What is happening?
A. BackTranslate always checks that a Single Build (*MCF0.txt*) exists, and that it was generated after the regular MetaCode target file (*MCF1.txt*) was written. If the Single Build does not exist or is out-of-date, it first calls MCF's _CreateSingleBuild function to regenerate *MCF0.txt*, and this produces extra update messages.

Q. Why do I need to BackTranslate anyway? Won't I just end up with the same script?
A. Strictly speaking, you don't need to BackTranslate first. But if you immediately translate your GUI strings, obfuscate your variables, and encrypt everything, and the final script (which looks like garbage) doesn't work, it's going to be hard to identify the problem. If the BackTranslated script does not work, then nothing you change afterwards is going to fix that. If your script passes BackTranslate but fails when you change things, you can start narrowing down which change is causing your script to fail. See further down on how to do this methodically.
Furthermore, BackTranslate acts as filter to create a single portable file without includes or redundant parts, so it does have its own purpose.

Q. Help! My BackTranslation produced by CodeScanner does not work. What do I do now?
A. Start up CodeCrypter, enable "Create MCF0" then navigate to Tab "Single Build" and disable all options, then press <Run> (under the "Main" Tab). Sometimes CodeScanner and CodeCrypter will make mistakes when deciding whether or not your script needs a particular global variable or UDF definition,

and throw out code you actually do need. If the problem persists, then you'll need to analyse more deeply. Does the test script start up at all? How far does it get? Has any content changed? MCF is still a work in progress, and there are valid AutoIt constructs that it cannot handle (see the *Remarks* sections in MCF and CodeCrypter). But if you find an obvious bug, please let me know, and I'll try and fix it if I can.

Q. Translation, Obfuscation, Encryption? Too many steps! I just want to make one tiny change. Can't you make it simpler?
A. Sure. Copy the text files of your arrays *Used.txt to text files *New.txt and make your change in the *New.txt file(s). Then call _CreateNewScript(*$path*, *True*) directly. You' re done. Of course, one tiny change is probably more quickly implemented by editing the original script directly...

Q. When would I be using "Create New"?
A. This option allows you to create a new script directly from the *New content arrays, without any alterations by MCF or CodeCrypter. You need it if you develop some new way to change the content arrays, and want to build a new script that incorporates those changes. Of course, you still need to start by creating the Single Build MCF0.txt

Q. I want to translate my GUI into a different language, but editing several arrays by hand is too much work. Is there no easier way?
A. Firstly, *never change more than you have to.* If you are just translating the external appearance of your script (the GUI or console the user interacts with), you don't need to translate the variables or the UDF names at all. When creating MCF0, all arrays with suffix "Transl" are prefilled with the same data as in the arrays with suffix "Used," so if you select Translate without changing anything, you'll achieve a simple BackTranslation, that is, rebuilding your original script as a single file. If you only change strings, the rest will be rebuilt with the original content.
Secondly, all arrays are written out to text file repeatedly, and read in again whenever you press <Run> in CodeCrypter, or when you call an MCF function directly with its second parameter (*force_refresh*) set to *True*. So never handle arrays directly, always edit the text files instead. For translation, just open the file *stringsTransl.txt* in your favourite editor, copy everything (or the GUI part) to the clipboard, dump it in Google Translate's left window, select and generate the output language, and copy the results back into stringsTransl.txt. If you're only translating a subset, make sure all lines still line up correctly with the original in stringsUsed.txt (line number = array index = MetaCode tag ID), and save the file. MCF/Codecrypter reloads the file into array $stringsTransl[], copies it to $stringsNew[] for script rebuilding, and you're done.

Q. I've done a string Translation, and now my script can't find any of its work files anymore. What gives?
A. Be careful which strings you translate, and always check the output of whatever automatic translator you use. Google Translate, for example, may insert an additional space before/after a (back)slash, so any string specifying a

directory path will be mangled. Same with filenames without extension that are also valid English words. Ditto for strings such as "\\PhysicalDrive0". See the earlier rule: *never change more than you have to*.

Q. I've done a string Translation, and now all my DllCalls fail. WTF?
A. Be careful which strings you translate. Dll calls require string parameters that define the next parsed parameter type, strings such as "short," "long," and "word" that are also normal words in English. So if you translate the entire stringsTransl.txt file, those words may end up as "kurz," or "longue," or "palabra," depending on your chosen language. And the dll that is handling your call is probably none too happy about that.

Q. While working with the CodeCrypter script, I changed my dynamic keytype definition in MCFinclude.au3, but the change is not implemented when I press <Run>. Why?
A. Any change made to MCFinclude.au3 has to be performed before running Code***Scanner*** (not Code***Crypter***). Otherwise it won't be incorporated in the MCF# files, nor in MCF0, nor in CodeCrypter itself (if it was already running before you edited MCFinclude.au3).

Q. While working with CodeCrypter.exe, I changed my dynamic keytype definition, but the change is not implemented when I press <Run>. Why?
A. If you compiled CodeCrypter, it will have incorporated MCFinclude.au3 in the state it was last saved in prior to compilation. Either recompile CodeCrypter or run the CodeCrypter script from Scite. See also the previous answer.

Q. I wish to use a macro as my keytype definition, but decryption has to work on the user's machine, not on my own machine where I encrypt the script. Is this possible?
A. Definitely. In CodeCrypter, navigate to Tab "Encrypt" and set your keytype ID number (the $CCkey array index where you stored your macro call, in MCFinclude.au3). Then press the "Decryptor" button. The macro will be executed and you'll see two blue text strings. The top one is what the macro just returned, the bottom one is what MCF will use to encrypt your script with. If you want this decryption key to be different from the top entry, just type the *expected response* twice (to ensure no typos) in the bottom two boxes (switch boxes by pressing Tab / Shift-Tab). Then press "Ok." You can check the result by pressing the "Decryptor" button again; now the second blue-text box should contain the string you just typed. Do not close CodeCrypter or your typed entry will be forgotten (CodeCrypter intentionally never stores decryption keys). Pressing <Run> will produce a script that won't function in your own environment; it will work only on whatever environment(s) your selected macro returns the expected response you provided. Note: be extra careful when typing case-sensitive responses.

Q. What happens if my keytype definition returns an empty string in the user's environment?
A. An empty string will trigger a password user query dialog at startup. Unless the user knows the expected response (unlikely), the script won't proceed.

Q. Why are lines containing macros @error or @extended never encrypted?
A. This is an AutoIt limitation first flagged by user *MagicSpark*. CodeCrypter (well, MCF actually) replaces original code phrases with Execute statements wrapped around a decryption call. Unfortunately the AutoIt language processor resets the @error and @extended macros to zero at the start of handling each native function, so the previous contents of @error and @extended are lost before Execute can evaluate it. Therefore, to preserve full functionality of your script, any line containing either of those two macros is kept unencrypted.

Q. What lines in my script are never encrypted with my chosen key(s)?
A. The following lines have issues that CodeCrypter cannot handle:

- All lines above MCFinclude.au3
- the contents of MCFinclude.au3 itself (a fixed-key encryption is used here; note that no decryption *keys* are stored here, only the *definitions* of how to obtain them at run-time).
- lines containing @error/@extended (these would lose their state prior to evaluation)
- object queries and direct object method assignments (but object methods are supported)
- multiple variable declaration+definition by call on a single line (
- using FileInstall (this is an AutoIt limitation)

In addition, trouble can be expected for:

- lines containing Assign, Eval, IsDeclared, Execute (self-modifying/-evaluating code); these may or may not work properly after encryption, especially if obfuscation is also enabled
- UDF parameter default strings (Func _MyFunc ($stringvar = "defaultstring"))
- multi-processing UDFs that relaunch a script subsection (these would lack MCFinclude.au3)
- applying indirection to variables that switch usage between single variant and array

Q. How much slower will my new translated/obfuscated/encrypted script execute?
A. That depends:

1. There's no extra work if you just replace strings and/or names of variables and UDF names. So translationwill not slow down your new script at all.
2. Obfuscation uses numerous permutations of a fairly long random hex-string; this implies that AutoIt's internal variable look-up will, on average, take slightly longer, as all variable and/or function names now look incredibly similar, especially at the front (which slows down the search algorithm a bit).

3. Indirection adds extra calls to a few tiny, one-line UDFs, so the overhead of indirection proper (or combined with obfuscation) will be negligible.
4. Decryption, however, is a different matter. Not only is the original code replaced with a decryption step, but the decrypted "code as string" then has to be executed (indirect call), so that's at least twice as much work as before, possibly more. If you decide to nest your encryption (fixed-key encryption of your dynamic key encryption, to hide your parsed keytype), you add a whole second decryption step plus "Execute" call. Furthermore, If you change the code structure through indirection prior to encryption, you add many more calls that are all going to be encrypted. All of this adds up. Only you can decide how much slowdown is acceptable. Unfortunately, some scripts (with tight event loops, or timed calls, like games and media players) cannot tolerate much delay.

On the positive side, you can adjust the proportion of encrypted code down to any percentage, reducing the amount of extra processing again. Also, there's no additional overhead for using multiple dynamic keys instead of one.

Q. Processing speed is not a limiting factor. Do I encrypt as much as possible, as little as possible, or somewhere in between?
A. That depends on whether you just want to stop the script from working without the key or password (encrypt a few percent), or protect as much of your intellectual property as possible (encrypt everything).

Q. Processing speed IS a limiting factor in how much I can encrypt, and I don't want to encrypt every N-th line or a random proportion, but some specific UDFs that contain all my brilliant ideas and code design. Any solution?
A. Navigate to the <Encrypt> Tab, and press <UDFs> in the bottom right panel. A new window is opened that lists all code sections (main script and UDFs) for which encryption is optional (remember that some MCFinclude UDFs are always encrypted, and any UDFs preceding MCFinclude cannot be encrypted). Next to each listed item is a checkbox (default: all enabled). Simply uncheck each UDF you do not wish to encrypt (press <Esc> to cancel without storing your settings). When you press <Return>, your new settings are stored, and will be reloaded in a future CodeCrypter session. This UDF selection list acts as a *filter* on the MCF encoding for ***phrases*** (mainly function calls; strings encryption is not affected). Note that this filter is always applied in encryption; it does not depend on the state of the Subset checkbox in the bottom-right panel of the <Encrypt> Tab. You just won't notice its effect until you start deselecting UDFs from the list.

Q. I want to encrypt only some specific individual lines, not entire UDFs. How do I do that?
A. This will require some effort on your part. First Run Codecrypter with full encryption, with subsets disabled. You now have two same-sized arrays: $phrasesUsed and $phrasesEncryp. Then you figure out at which line of *phrasesUsed.txt* your important code begins, and at which line it ends. Now you write a tiny script that:

- copies text file *phrasesUsed.txt* to file *phrasesNew.txt*

- calls _readCSdatadump() to load all arrays in memory
- fills array $phrasesNew from array $phrasesEncryp, *but only from the first to the last phrase of your important code!*
- calls _FileWriteFromArray("phrasesNew.txt",$phrasesNew,**1**) (you need #include <File.au3> for that) to write out array $phrasesNew to file *phrasesNew.txt* (don't forget the third parameter "1", or everything will be misaligned and fail).

Then you fire up CodeCrypter, enable CreateNew, and press <Run>. You're done.

Q. How secure is the encryption when you store the fixed key that encrypts *MCFinclude.au3* inside *MCFinclude.au3* itself? Won't an attacker just bootstrap-decrypt the script in two passes instead of one?
A. Not possible. The fixed-key encryption is to prevent casual inspection to reveal the keytype *definitions*, that is, *how* you obtain your decryption key at runtime. A determined attacker *will* be able to figure out that you are, for example, using keytype 1, which means a password user query dialog box will be triggered at runtime (which is obvious to figure out anyway). But that's it. The password is itself never stored anywhere (other than existing briefly in RAM memory while CodeCrypter is encrypting).

The crucial safety factor is *access*, not to the programme but to the user environment. For example, if you choose to encrypt with the C-drive serial number of your end user's machine, and the attacker is able to log in to that machine in secret and obtain that serial number, all bets are off. The same goes for user names, IP addresses, stored signature files, Registry key, etc. And a user password is only as safe as the user keeps it; if they have it written down on a note on their keyboard and they keep their office door unlocked when they're gone, game over. You'll have to be creative, and tailor a solution to the specific circumstances of your target environment. Perhaps a combination of something user-specific, something machine-specific, and your own web server's response to the programme's unique serial number?

Q. Lets say a "heavy" encryption method is used to only run on a certain computer (with an environment-dependent key), and somehow someone gets a hold of that .exe. Would the data still be injected unencrypted into memory when that person trys to run it? (from the CodeCrpyter thread, question by member CodeFOB)
A. The decryption engine itself always runs when the exe is run, regardless of where or by whom it is run, but this does not matter! The beauty of this decryption is that an attacker can study the decryption engine for a lifetime, and still be unable to discover either the decryption key or your plaintext script, providing they do not have access to the original environment where the encrypted script or .exe is supposed to run. In the worst-case scenario, a determined hacker might be able to determine the type(s) of information your programme queries from its environment to obtain one or more decryption keys, but their content (user password, user name, drive serial number, fixed IP address, any AutoIt macro or UDF you write yourself) will be different when the attacker runs their stolen .exe elsewhere. So each decryption step is still executed anywhere, but (providing you select or define your decryption key

wisely) only a single machine and/or user will produce the decryption key with which the script was originally encrypted.

Example:
You can think of the array $CCkey in MCFinclude.au3 as a list of instructions to construct a secret phrase (the decryption key). For example:

- go to the library, find the first red book on the top shelf in the cabinet left of the largest window, take the 7th word of the 3rd paragraph on page 123;
- use the name of the only black pet in the house;
- take the first letter of each word of the saying on the plaque over the kitchen door
- take the word for the material of the bedspread in the main guest room
- etcetera (you can define these instructions yourself, adding as many as you like)

When your proverbial hacker steals your executable, even though this list is itself also encrypted (with a fixed key; there's no other way), they may be able to decipher the list itself, or monitor the location in RAM where the result of these instructions is stored. But such an instruction itself no longer makes sense if it's referenced <u>in any other house</u>! There, the guestroom bedspread may be a different type of cloth, there may not be any house pet, let alone a black one, or even if so, it's name is likely different. Thus your hacker can monitor all they want, but garbage in = garbage out, meaning the decryption key does not match, therefore no sensible AutoIt code comes out of the decrpytor, just gibberish that causes the programme to halt immediately. So yes, "the data" (that is, the encrypted line and the "secret phrase" as extracted locally from the work environment) are indeed "injected in memory", but this information is completely useless when the secret phrase is different because the environment is different. There is no hint whatsoever inside your script what name your black pet actually has; there never was, and never will be. **What is absent to begin with cannot ever be extracted.** Your only responsibility is to select or define an instruction (or combination of several) that is unique (or very rare), and unobtainable without access to the original environment.

The above analogy thus highlights the importance of choosing a sufficiently strong key; a user password can be strong protection unless the user's notebook with their list of passwords was also stolen; many people will have the same username, and so forth, so it makes sense to use several keys (either strung together or used cyclically (CodeCrypter allows you to select this option too), because the chances of another house having a black pet with the same name AND a red book on the top shelf next to the window AND a woollen bedspread in the guest room are fairly remote.

In summary, low-level debugging, or access to RAM or your encrypyed exe is totally insufficient; the only way to break the encryption is to gain access to the original environment to determine those actual bits of information that your exe extracts from it at runtime to build the decryption key.

Q. My script works when encrypted in a 32bit environment, but fails to run when encrypted in a 64bit environment. Why?
A. Ward's AES encryption UDF has an unresolved issue in 64-bit environments that awaits a fix. If you have to run in a 64-bit environment, use TheXman's

CryptoNG library instead, which uses Bcrypt dll calls (part of standard Windows OS from Win7 upwards). Under the "Encrypt" Tab, ensure you've selected the second radio button at top right (labelled "x86, x64") prior to encryption.

Q. My new script does not work. How can I identify the cause?
A. Methodically eliminate all factors, dear Watson:
Start with CodeScanner. Were any issues identified that might affect the script's proper functioning? Does the original script actually run?

Next up: BackTranslation. If the BackTranslated script does not work, the problem likely lies in the initial MetaCode translation step. Create a new MCF0 while retaining all supposedly "redundant" parts (disable the Single-Build code-pruning options), and see if that works when BackTranslated (if not, you're in trouble).

If the BackTranslation is okay, switch on one alteration you implemented at a time, rebuild a new script, and see if it runs. That should tell you whether the problem lies in Translation, Obfuscation, Indirection, or Encryption.

All content alterations involve array manipulation. Check all array text files on length; the number of lines should be equal for all files with the same prefix. Even a single entry lost or added causes misalignment, often with disastrous consequences. For strings, ensure they are all enclosed in single- or double quotes. For variables, ensure that all names are prefixed with "$". For macros, ensure that all names are prefixed with "@".

If none of this resolves the issue, then the most likely cause for failure is encryption. Switch off nested keys and multikeys, and keep them off. Now try and encrypt strings only (if enabled previously), then phrases only (if enabled previously). If that does not narrow down the problem, enable Subset encryption and encrypt only a tiny fraction of the code, such as 1 line in 100, or a few percent. This should tell you whether the extra processing load is to blame. (If so, gradually increase the proportion up to the point where the script starts failing again, then maybe use half that proportion).

If you're still no nearer to a solution, it's time for more drastic measures. Create a working BackTranslated script and a non-working encrypted one in the same directory. Open both files in Scite and run AU3Check to see whether there are any syntax errors. If not, run the encrypted version to check whether it starts up at all, and if so, how far it gets. Alternatively, you can start replacing parts; first the main code section, then each clear-code (working) UDF definition with their encrpyted counterpart, one at a time. If the problem is localised, it *should* be identifiable this way, although it may take time.

Finally, some valid AutoIt code simply does not work after passing through MCF's digestive tract. A real example: Suppose a script compares the speed of different sorting algorithms. Each algorithm has its own UDF, called from the main script. Each of the UDF names ends in "Sort" ("BubbleSort", "QuickSort", "MoronSort", etc.). The main script defines an array filled only with the prefix, that is, the part that is different for each name ($name[1]="Bubble", $name[2]="Quick", ...) and calls each in turn in a For-Next loop (with counter

$n) that calls each UDF like this: **Execute($name[$n] & "Sort")**. Now suppose we obfuscate the UDF names. Neither the strings in array $name[] nor string "Sort" in the *Execute* call will match the original UDF names, so they are all kept as strings instead of being replaced by **{funcU#}** MetaCode tags. Worse, to CodeCrypter it will appear as if the main script never calls any of the defined *Sort functions, so unless prevented, it won't even include them in the Single-Build (MCF0). Of course, once this problem is identified, we can easily fix it by storing the full names in the array instead and removing the "Sort" suffix from the string parsed to Execute(). That way, the strings will be recognised as UDF names, the UDF definitions will be retained as active, and obfuscation will replace the original name both in the UDF definition and in array $name.

Latest revision: 27 Feb 2020.