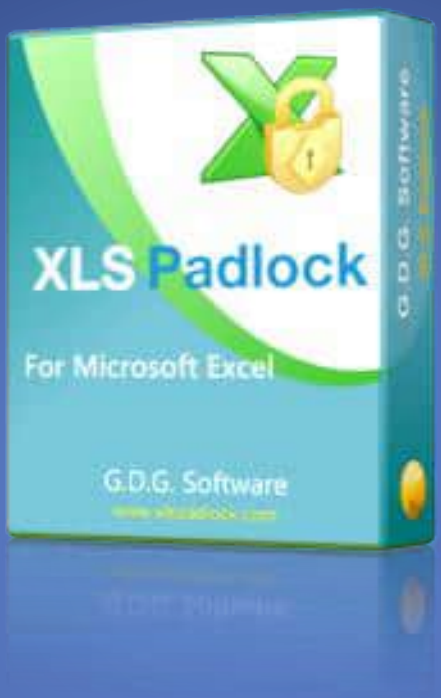
V 2025



XLS Padlock™ Guide

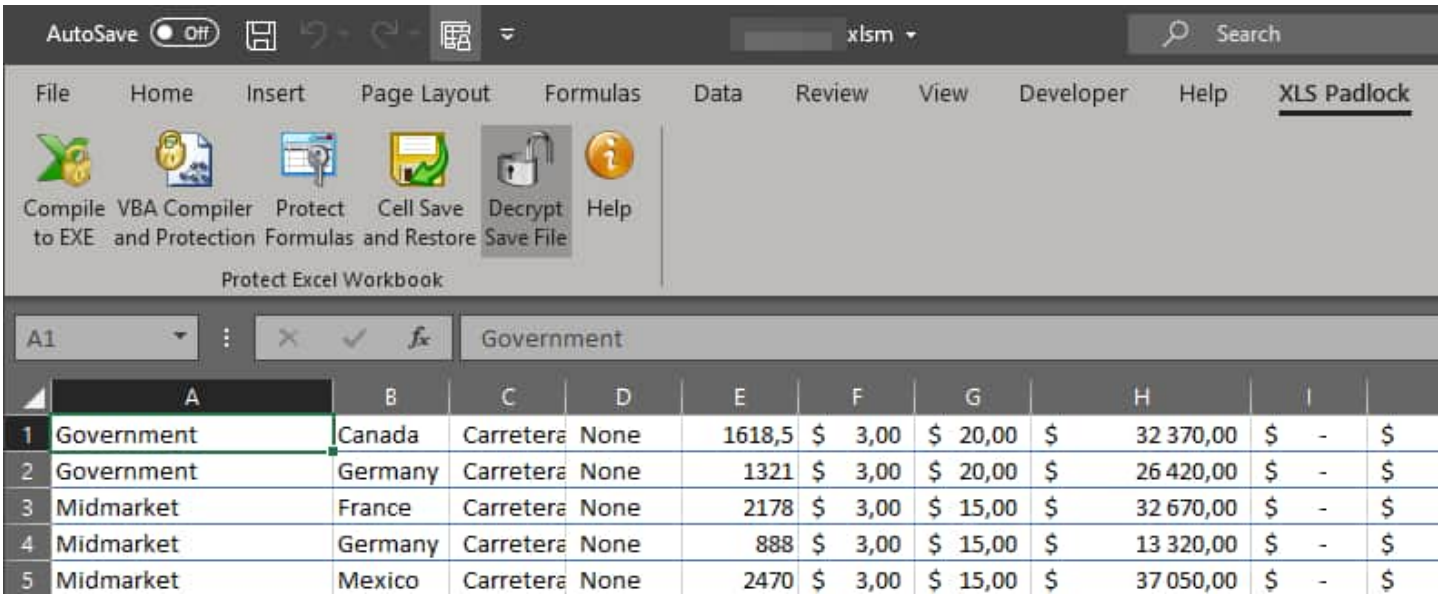
XLS PADLOCK - EXCEL PROTECTION SOFTWARE

G.D.G. SOFTWARE

Table of Contents

1. Introduction to XLS Padlock

XLS Padlock is **protection and licensing software for Microsoft Excel** to copy-protect Excel workbooks. It works as a compiler that allows you to turn your Excel workbooks into secure applications.



	A	B	C	D	E	F	G	H	I	
1	Government	Canada	Carretera	None	1618,5	$ 3,00	$ 20,00	$ 32 370,00	$ -	$
2	Government	Germany	Carretera	None	1321	$ 3,00	$ 20,00	$ 26 420,00	$ -	$
3	Midmarket	France	Carretera	None	2178	$ 3,00	$ 15,00	$ 32 670,00	$ -	$
4	Midmarket	Germany	Carretera	None	888	$ 3,00	$ 15,00	$ 13 320,00	$ -	$
5	Midmarket	Mexico	Carretera	None	2470	$ 3,00	$ 15,00	$ 37 050,00	$ -	$

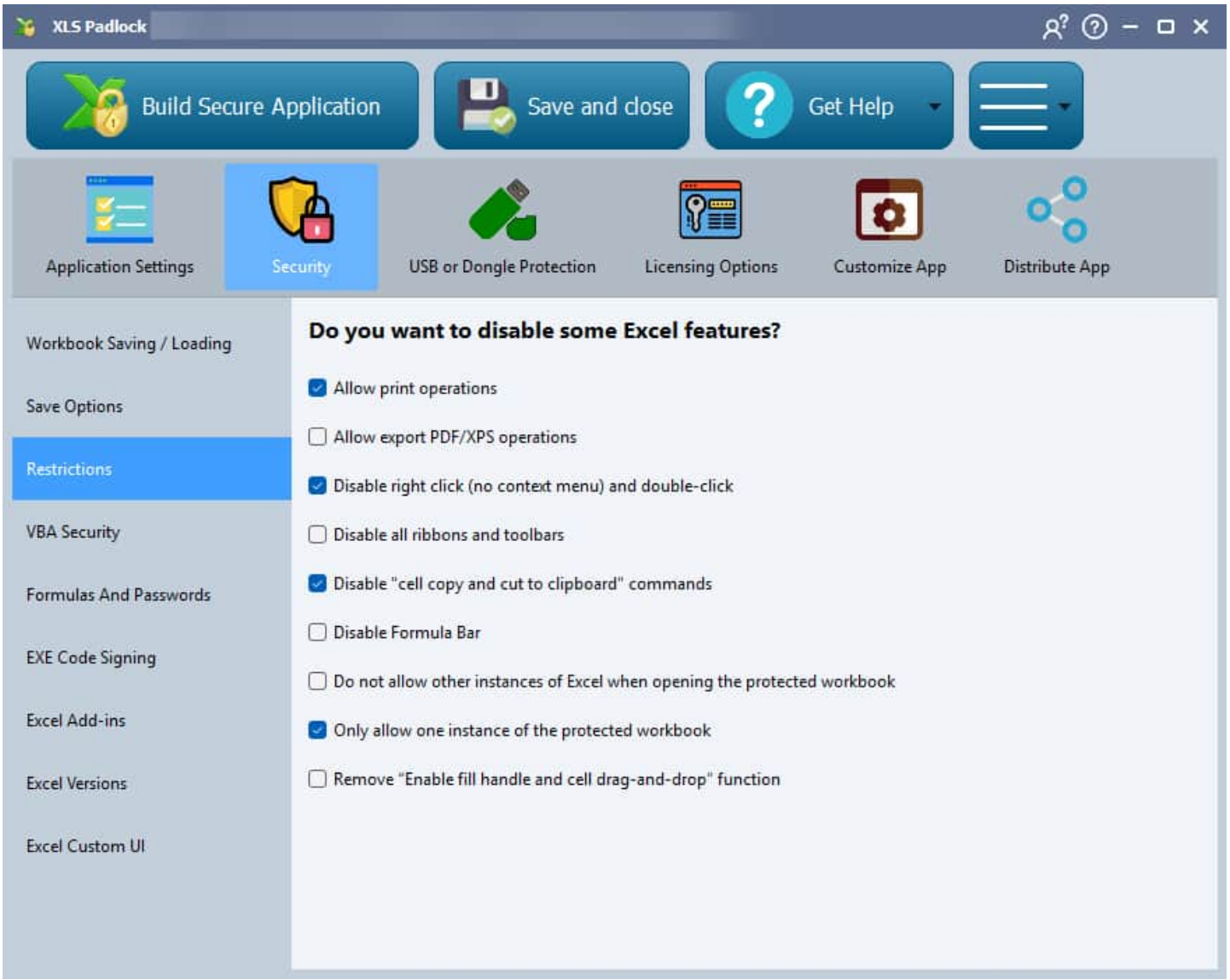
Thus, you can safely distribute your Excel files, control which user can access your Excel workbooks and prevent copying of sheets. Sheet context menu (right-click), copy to clipboard, formula bar, save, printing, access to VBA project can be disabled.

Besides, XLS Padlock allows you to secure your Excel files with USB stick, with strong dongle protection (even with dongle remote update), and/or with hardware-locked activation keys only working on a given computer. You can also decide which formulas should be protected, so that an end user can use them, but without viewing them or being able to copy them in another Excel file.

VBA Code Protection

With the built-in VBA code compiler of XLS Padlock, **compile your sensitive VBA code into secret byte-code not accessible to final users**. Your VBA macros are protected and cannot be studied/copied because the original VBA code does not exist anymore. The compiler is not a simple obfuscator: it completely turns VBA code into binary code and stores it directly into the application. Finally, you can lock or password protect your VBA project: since the original XLS file cannot be recovered, Excel password-cracking tools are useless. Learn more about VBA Code Protection.

Lots of Security Options



Applications can expire after a given number of days or runs, or on a specific date. A nag screen can be displayed for trial versions. Online activation also lets you automate licensing of your applications and control who may access your workbooks remotely.

Secured workbook applications are stand-alone and only need Microsoft Excel to work. All Excel functionalities are supported.

Applications can have their own icon and copyright information. They can also be translated into the language of your choice. A license agreement (EULA) and/or a splash screen with semi-transparency can be optionally displayed at startup.

You can digitally sign compiled workbooks with Authenticode, and even create an installer for distribution.

Finally, customers can be notified of new versions of your compiled workbooks thanks to web updates.

Compatible with Office 365 and Excel 2024, 2021, 2019, 2016, 2013, 2010, 2007 (SP3), 2003, 2002 - 32-bit or 64-bit versions.

A Windows x64 edition is required to create 64-bit applications.

- Download and install XLS Padlock

- How to protect an Excel workbook
- XLS Padlock Website

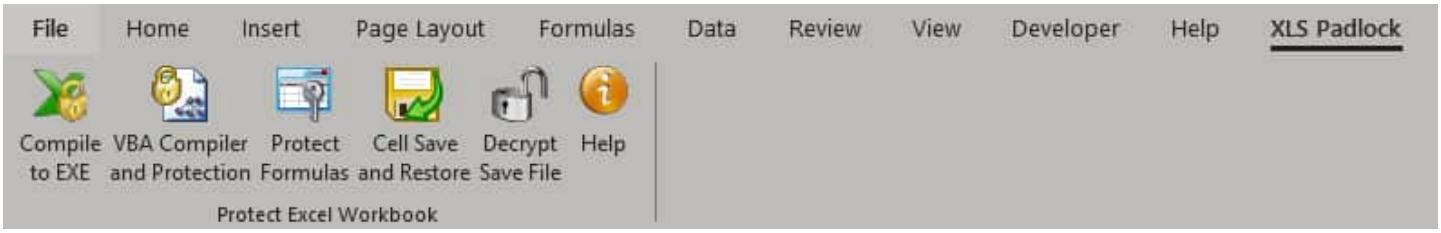
2. Installing XLS Padlock

2.1. Download and install XLS Padlock

On our website at https://www.xlspadlock.com/download, you must select the version you want to download, install and use: 32-bit or 64-bit. Choose the one according to the Excel version you have. The installer will check whether you install the correct version anyway.

For advanced users, installing can be customized.

After successful installation, **XLS Padlock is integrated into Excel**: it appears as another tab or menu of the software. To use XLS Padlock, you need to open your Excel workbook first:



Two shortcuts are created on the desktop: one for this guide and another for the XLS Padlock Manager.

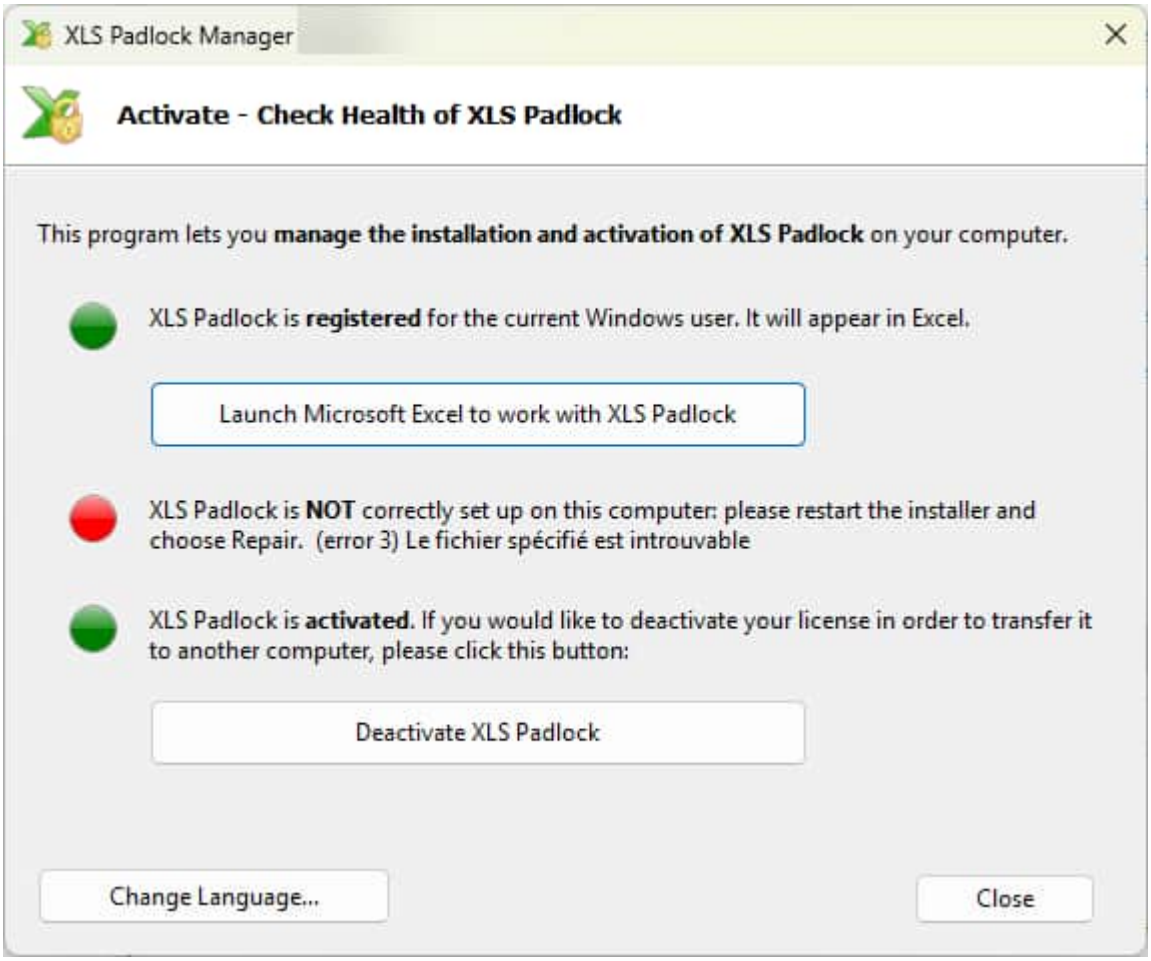
- XLS Padlock Manager
- How to protect an Excel workbook

2.2. XLS Padlock Manager

XLS Padlock Manager is a stand-alone application that lets you **check whether XLS Padlock is correctly set up and registered with the current Windows user**.

In fact, XLS Padlock **must be registered for each Windows user who wants to work with it**. When you install XLS Padlock on a computer, the administrator account is used: XLS Padlock will only be registered for the administrator. For other accounts, you have to use the XLS Padlock Manager.

- To register XLS Padlock for your Windows user account, launch "XLS Padlock for Excel – Manager" from your Windows desktop. The following window will appear:



If all LEDs are green, then your installation is fine.

If the first LED is red, click "**Enable XLS Padlock for the current Windows user**": XLS Padlock will be automatically registered, and you will be able to use it. This operation does not require any administrative rights:

Note: if you already have a license for XLS Padlock, you can activate it.

- How to protect an Excel workbook

3. How to protect an Excel workbook

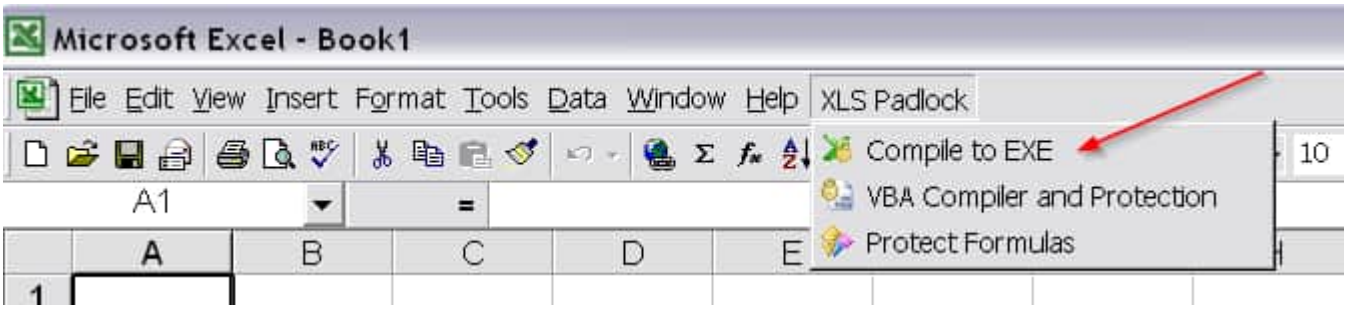
Check the following **video tutorials** about XLS Padlock made by an Excel expert:
https://excelvbaisfun.com/?ref=5

1. Open the workbook you want to protect, and save changes.

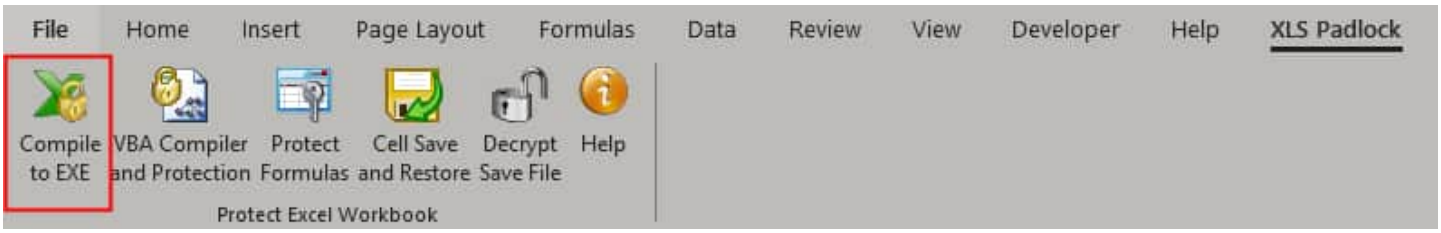
2. Configure general and security options in XLS Padlock.

Choose Compile to Exe in the XLS Padlock menu or ribbon:

- *Before Excel 2007*: "XLS Padlock" menu



- *Starting from Excel 2007*: "XLS Padlock" ribbon

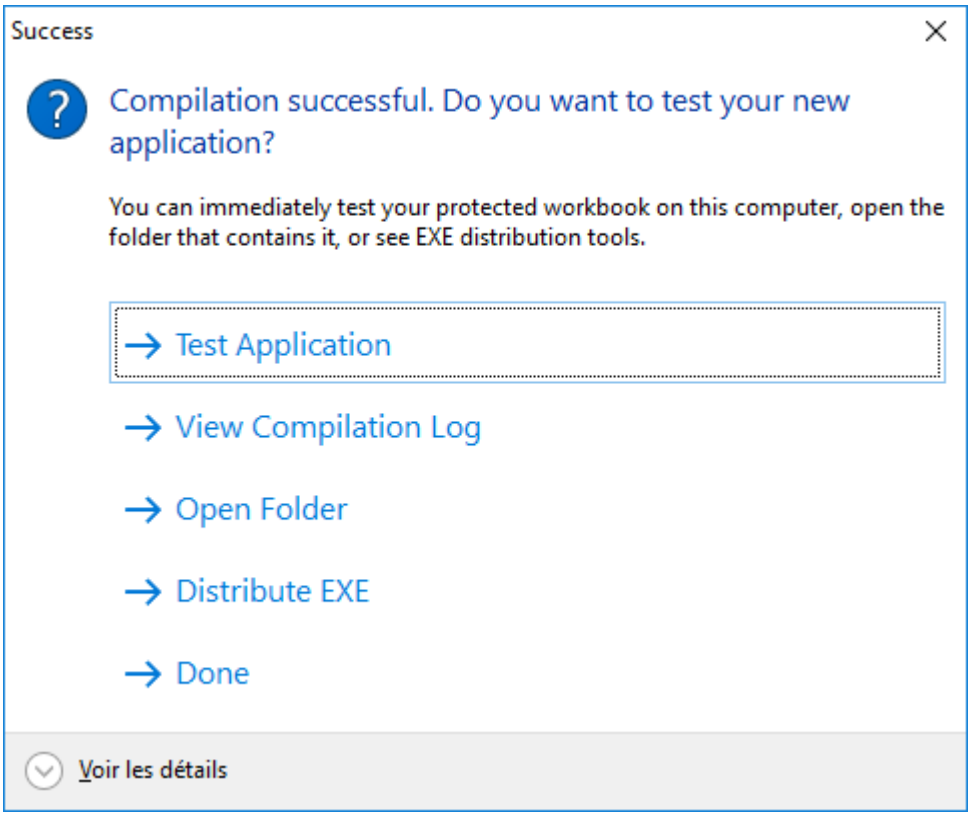


Make sure you have saved your Excel document before starting XLS Padlock.

3. Build your application

After configuring your application's settings, you can build your secure application with the following button or by pressing **F5**: the resulting application EXE file is created.

After compilation, the following message is shown:



You can start the application, view compilation log (especially useful if some error occurred), open the destination folder…

XLS Padlock does **not** modify the Excel original workbook file during its compilation into an application EXE file. Moreover, do not delete your original workbook file afer compilation.

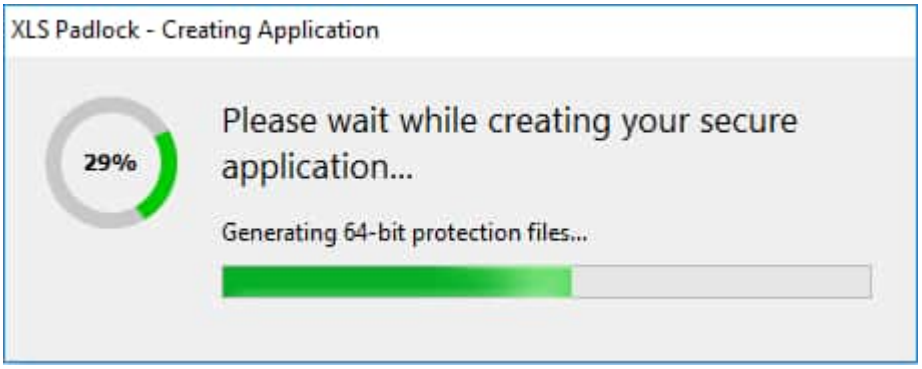
When you run the compiled workbook, if you see strange results or a malformated workbook, try to turn the following option on: "Use Excel automation for formula protection" available in Formulas and Passwords page.

- How to distribute a protected workbook

4. How to distribute a protected workbook

In XLS Padlock, when you click "**Build Secure Application**" 

, the progress dialog box is displayed:



And that's all. **Your secure application is ready.**

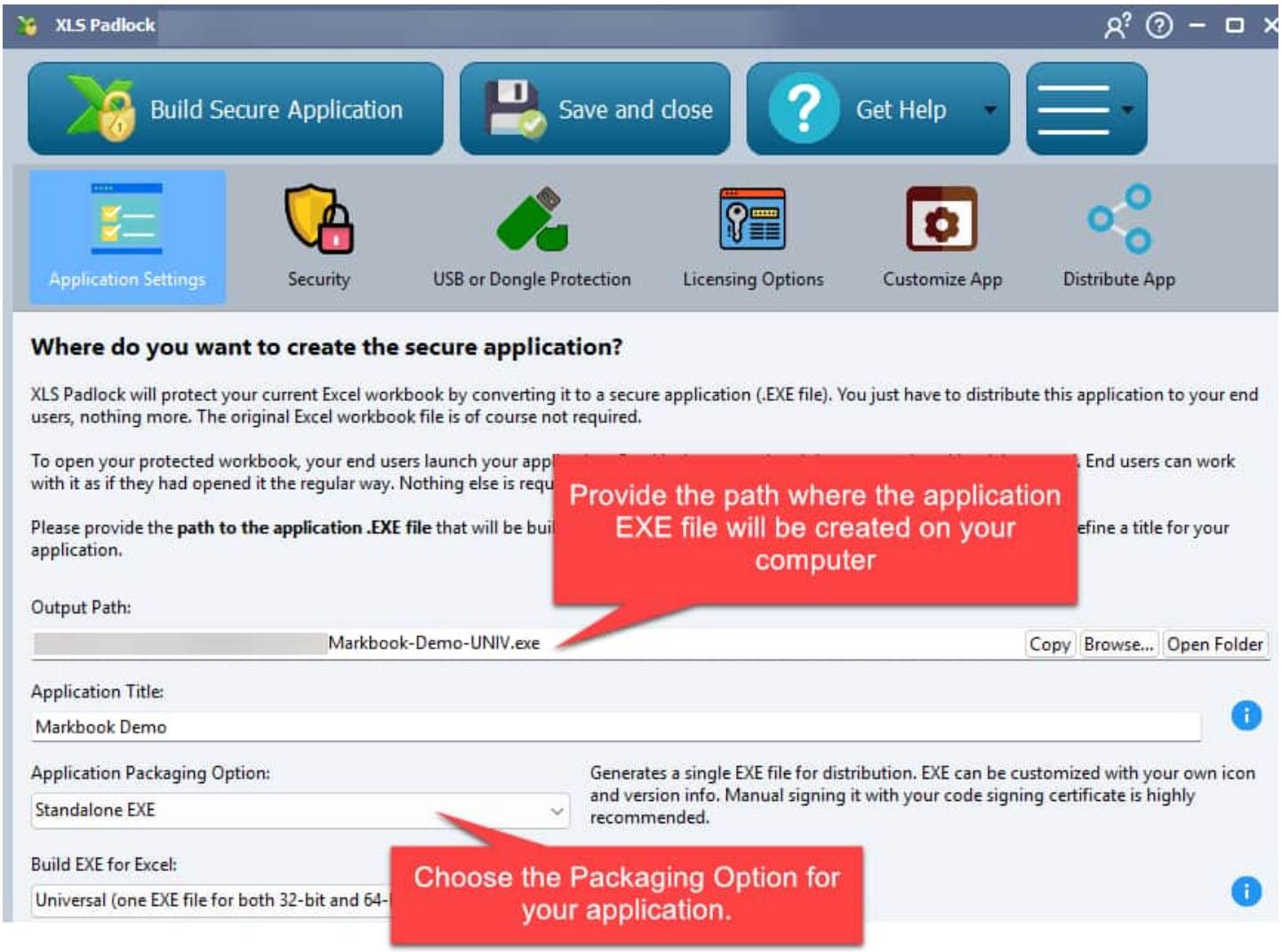
XLS Padlock generated an executable file (.EXE) from your Excel workbook, recognized by Windows.

To access the protected workbook, end users launch this .EXE file. A local copy of Microsoft Excel is the only other requirement.

- Upon launching the .EXE file, Excel starts, and the protected workbook is opened. End users can interact with it as if they had opened it in the usual manner.
- Certain options like "New Workbook", Open, Save (optionally) are disabled for security reasons.
- If Excel is not found on the user's computer, an error message will be displayed. You can configure your application to require specific versions of Excel.

Which Application Packaging Option?

XLS Padlock offers you the choice to generate your secure Excel workbook application in two formats, thanks to the **Application Packaging Option**. This choice directly influences how you distribute your application and manage security concerns such as Windows Smartscreen and trusted distribution.



Standalone EXE

XLS Padlock creates a single executable file (.EXE) from your Excel workbook, recognized by Windows. This .EXE file is generated in the location you specify with the "Output Path".

- You simply distribute this .EXE file to your end users. The original Excel workbook file is, of course, not required.
- We highly advise digitally signing your EXE files in today's digital environment. XLS Padlock offers an automated feature for this if you have a code signing certificate. This step is especially crucial for the Standalone EXE option, as distributing an unsigned EXE file might prompt Windows SmartScreen to show the standard "Unknown Application" warning. Nonetheless, rest assured, your EXE file will operate flawlessly even without a digital signature (it's just a warning).

EXE + application XPLAPP Bundle

Choosing this option generates an EXE file, a .bin64 companion file and a separate .xplapp data file. The EXE, pre-signed by our company and recognized by Windows SmartScreen and major antivirus companies, facilitates trusted distribution without the immediate need for a personal paid code signing certificate.

- Distribute both the EXE, the .bin64 and .xplapp files to your end users. The EXE will know to open the .xplapp file containing the user data, provided they are in the same directory and with the same filename. The .bin64 companion file is used for Excel 64-bit.
- XLS Padlock can pack these three files into a single Zip archive or a single installer for easier distribution.
- This option greatly reduces the risk of unknown application warning from Microsoft Windows

SmartScreen and antivirus false positives, ensuring a smoother execution of your application for your users.

Recommendations

1. To avoid possible false positives of antivirus programs, we recommend you **digitally sign your EXE files**. XLS Padlock can automatically do it for you if you have a code signing certificate.

2. If you have an **active antivirus program**, disable it before compiling your workbooks. Otherwise, you may get false positives, because your antivirus may not understand why EXE files are suddenly created on your computer. You can enable it again later.

3. Directly emailing EXE files is not recommended, as email providers might block them due to potential virus threats. Instead, opt for uploading the EXE file (and .bin64 + .xplapp files, if applicable) to a file hosting service, web server, or an FTP server. Services such as WeTransfer, Dropbox, Google Drive, OneDrive, or cloud solutions like Amazon S3 can be effective alternatives. You may also consider putting your application files into a Zip archive.

5. How to save and restore settings – Using Templates

In XLS Padlock, your project settings are automatically saved when you click 

.

- Project settings are stored in an **XLS Padlock project file** whose extension is .XPLP. The project file lies in the same folder as your source Excel workbook file.

- When you open XLS Padlock again, settings are automatically restored.

In the App menu button 
, you have additional project commands for importing/exporting settings as **templates**.

For instance, you can import all settings from another project file (.XPLP) thanks to the "**Load settings from template**" file.

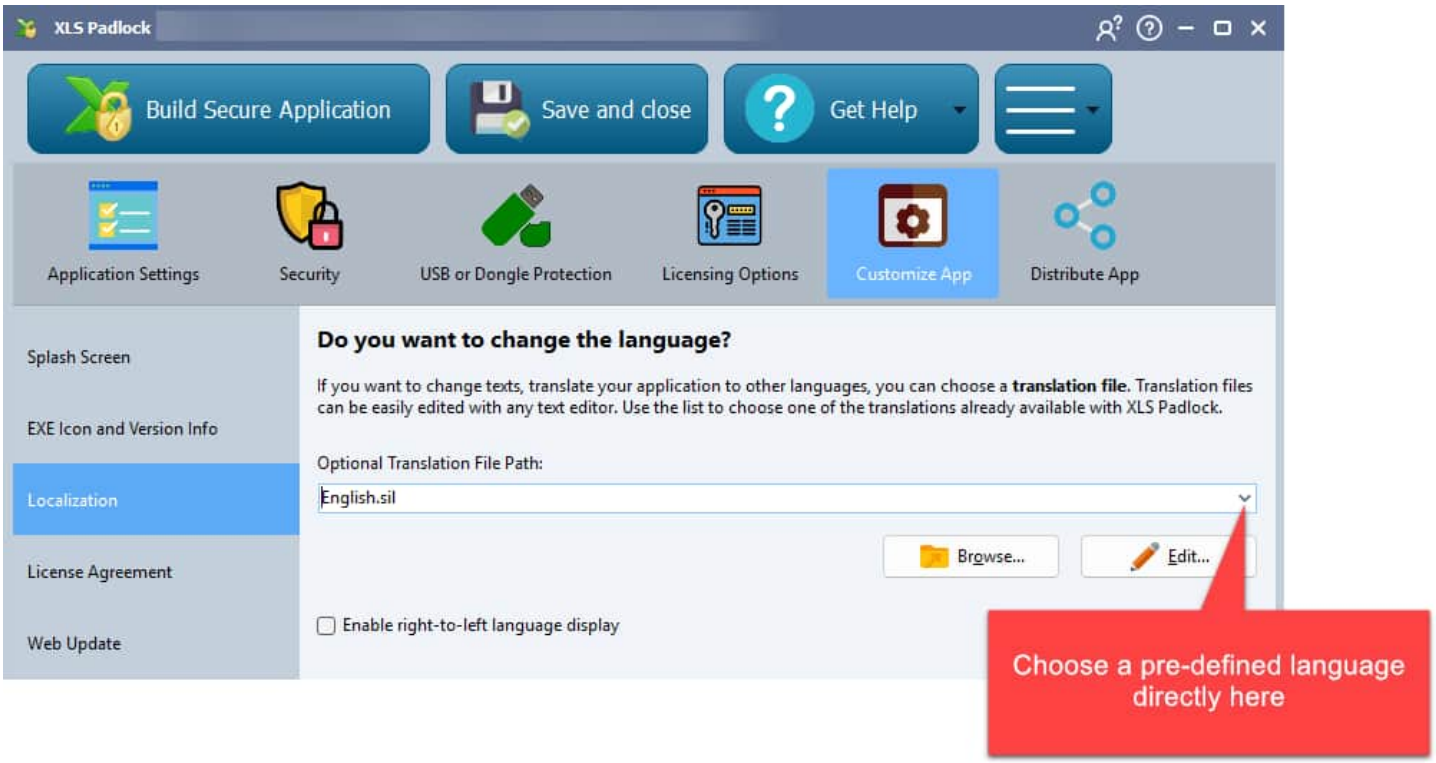
> Importing a template will **overwrite all settings**, except the output path field. Use it with care. Especially if you allow save for your workbooks: we recommend that you backup the Application GUID and Application Secret Key before importing a template.

Similarly, you can export all current settings to a **template file** for reuse.

Localization and Translation File

XLS Padlock provides **localization support for your protected workbooks**. Explanatory field text and dialog captions can display in the language of your choice when your application is run.

You can configure localization for your protected workbook app here:



XLS Padlock loads all localization texts and resources from translation files whose extension is ".sil". By default, XLS Padlock uses the translation file name "**English.sil**" available in the "XLS Padlock" installation folder.

XLS Padlock comes with some language files available in the "XLS Padlock" installation folder: the software loads them into the predefined list when it is launched. Use the combo box to choose one of the existing language files, for example German, French, etc.

The following languages are currently available:

English, French, Spanish, German, Portuguese (Brazil), Dutch, Slovenian, Croatian, Arabic.

If you would like an additional language, please contact us.

Translation files are in **text format** (same as .TXT), and their encoding should be **UTF-8**. Thus, you can edit translation files with any text editor such as Notepad or Notepad++.

How to define a language for your workbook application

To choose an existing translation file, either use the combo box, click "**Browse**" or enter its path in the

associated field.

- To customize a translation file, just click "**Edit**" and your default text editor software will be run.

  If no language file is specified in the field, the default language file is opened. You cannot modify it directly without administrative rights: we recommend you save the default language file as a new file, make your modifications and then specify the path to the new language file.

The selected translation file should remain available as an external file when the application is being compiled.

7. Improve protection of your workbooks

Why just compiling a workbook into EXE is not enough?

When you compile a workbook into a "**Secure Application**," the original workbook file (XLSX, XLSM, etc) becomes inaccessible in its raw form. Users can only open and interact with it by running the compiled .EXE file.

However, simply compiling your workbook as a secure application doesn't provide sufficient protection. It's comparable to closing a door without locking it. To enhance the security of your workbook, it's essential to **activate the additional robust protection features offered by XLS Padlock**.

XLS Padlock temporarily loads your workbook into memory to allow Excel to interact with it. **Without enabling the extra security layers of XLS Padlock**, experienced hackers could potentially extract the original workbook from the EXE file. While this is not a straightforward process, it becomes a possibility if the stronger protection features of XLS Padlock are not utilized.

When you correctly apply XLS Padlock's protection features, even if someone manages to extract the workbook from the EXE, **it won't function as expected!** The protection is particularly strong for workbooks that contain formulas and/or VBA code. These elements are well-shielded, ensuring that unauthorized access or tampering is effectively prevented..

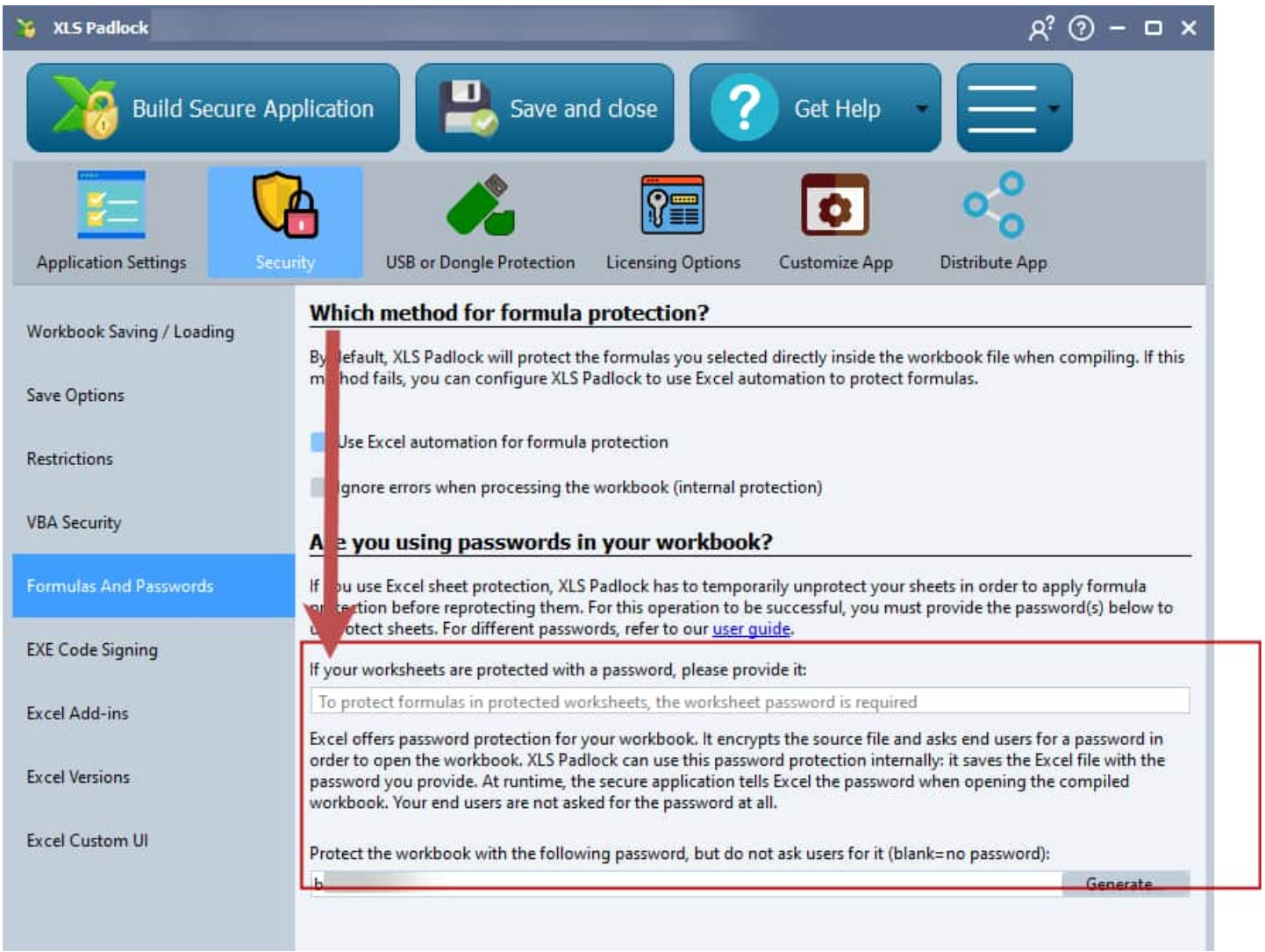
Which XLS Padlock security features are the most effective

You will find some suggestions for stronger protection in the following topics. Note that all XLS Padlock options available are described in detail in the All XLS Padlock options topics.

- Prevent common VBA and OLE hacks
- Password protect your workbook
- Forbid access to the VBA editor (VBE)
- Disable common Excel add-ins
- Protect your formulas with XLS Padlock's formula protection
- Real VBA code protection with VBA compiler

7.1. Password protect your workbook

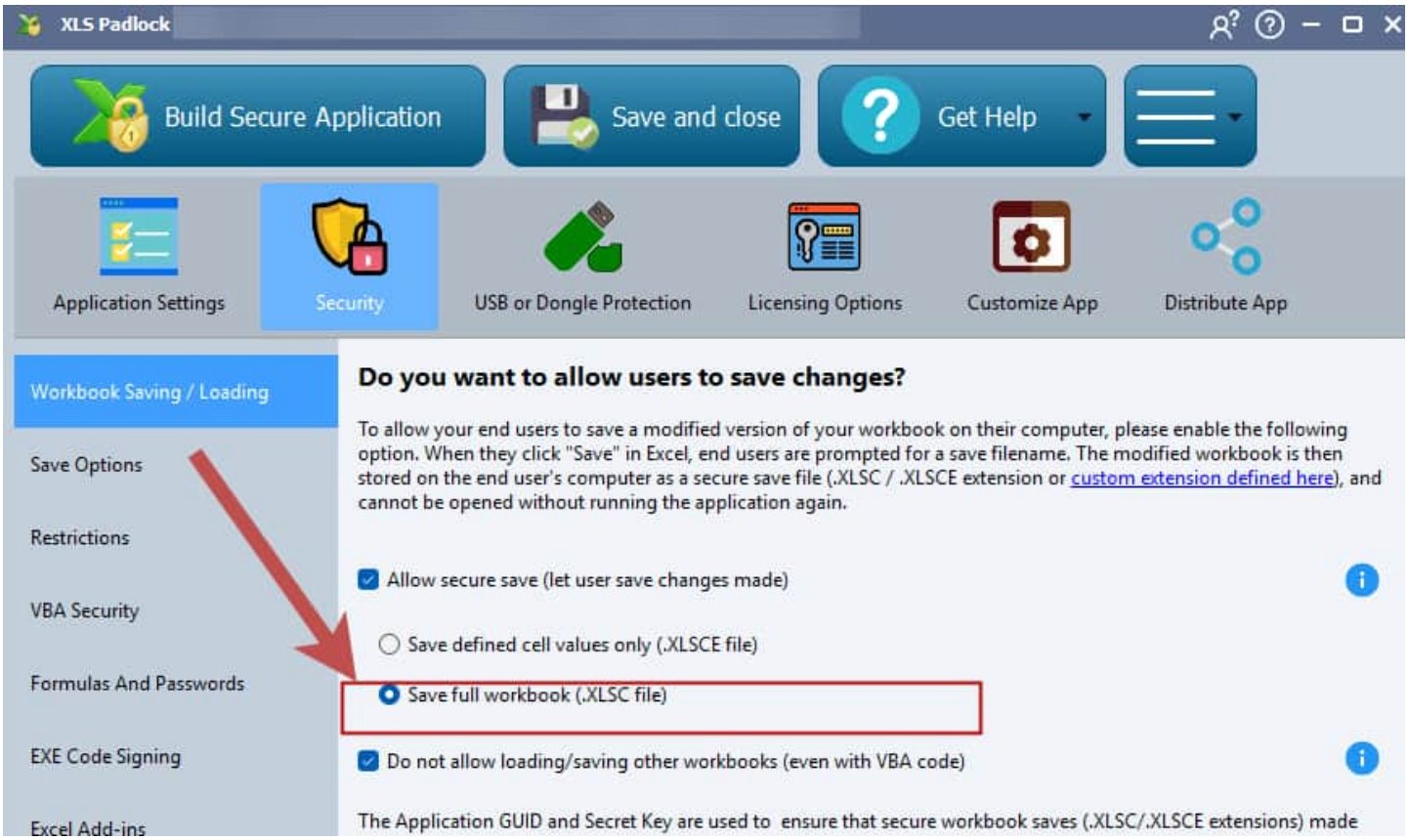
You should let Excel encrypt your workbook file with a password. If you don't already use a password, XLS Padlock can generate one. If you specify a password in the field highlighted below, **end users are not asked for the password**: it is automatically provided by XLS Padlock.



Prevent common VBA and OLE hacks

7.2. Prevent common VBA and OLE hacks

Excel workbook files can be saved as files on disks with VBA and/or OLE commands. To prevent this hack, XLS Padlock has a security option named "Do not allow loading/saving other workbooks".



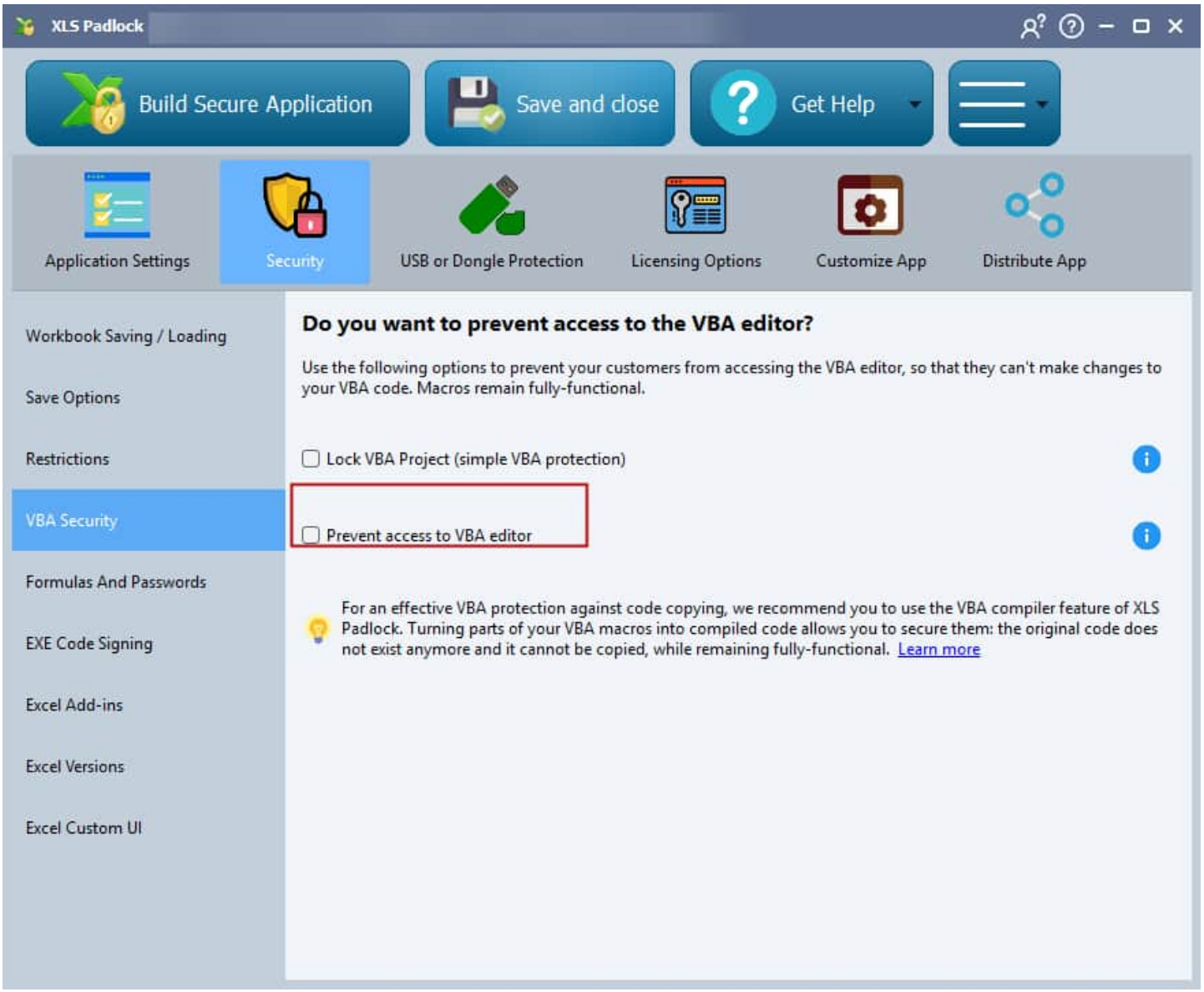
Turn this feature on. If you are having issues with that feature (for instance, your workbook must save other workbooks with VBA), some solutions exist.

- Password protect your workbook | Forbid access to the VBA editor (VBE)

7.3. Forbid access to the VBA editor (VBE)

The Secure Application will **prevent end users from accessing the Visual Basic editor**. This does not really protect your VBA code, contrary to the XLS Padlock's VBA compiler. However, **it is useful to limit possible VBA hacks** (password retrieval, export of the workbook to an external file…)

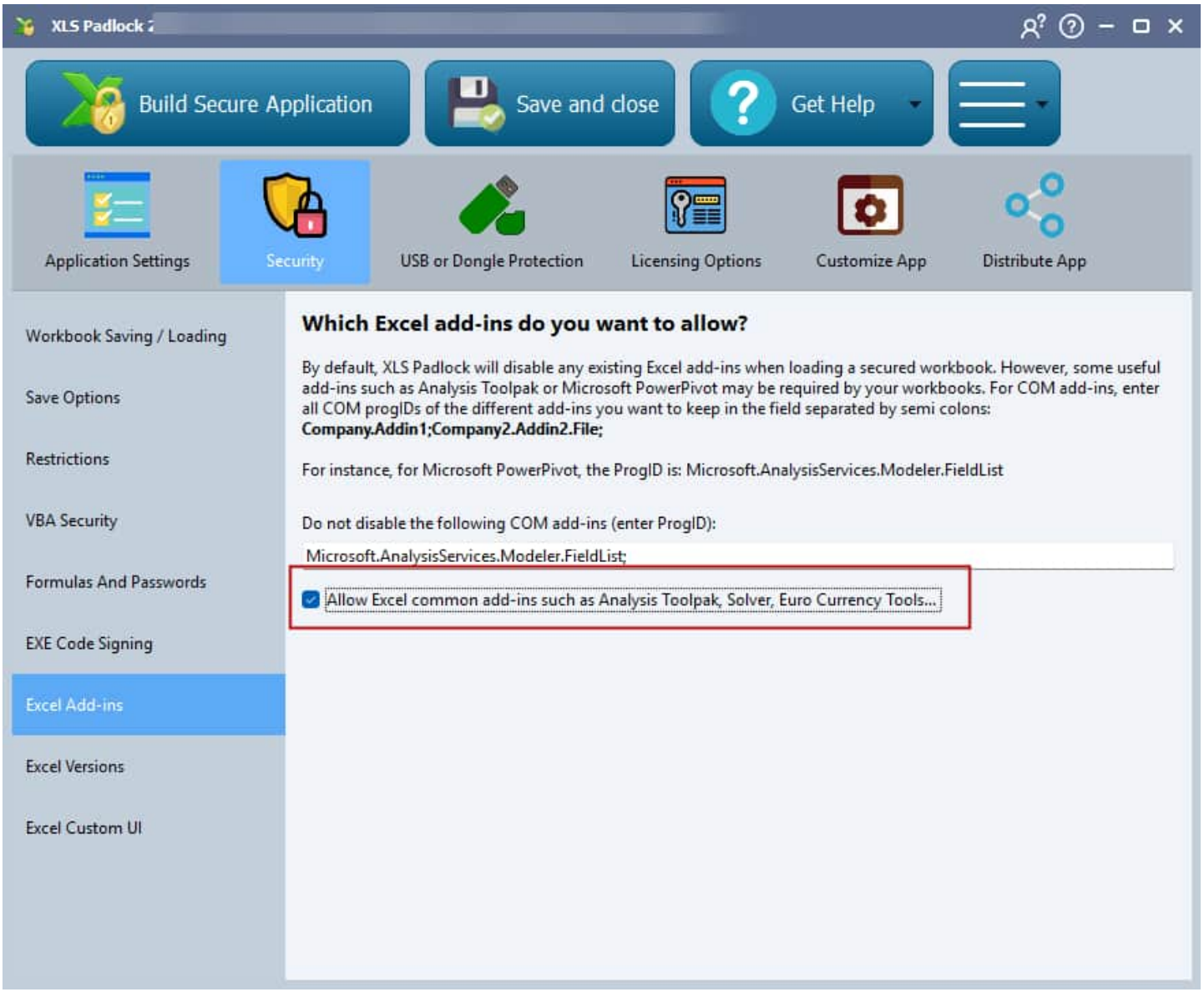
- Learn more about VBA Code Protection in XLS Padlock
- Please also refer to the dedicated topic about stopping access to VBA editor.



Prevent common VBA and OLE hacks | Disable common Excel add-ins

7.4. Disable common Excel add-ins

Unless your workbook uses Excel add-ins, they are not necessary. Uncheck the following option "Allow Excel common add-ins":

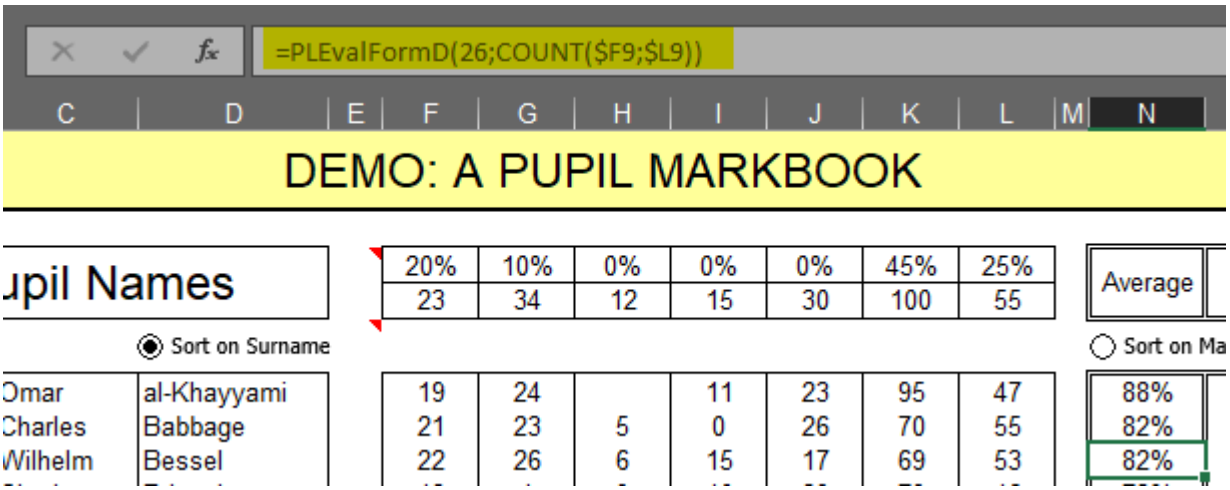


Forbid access to the VBA editor (VBE) | Protect your formulas with XLS Padlock's formula protection

7.5. Protect your formulas with XLS Padlock's formula protection

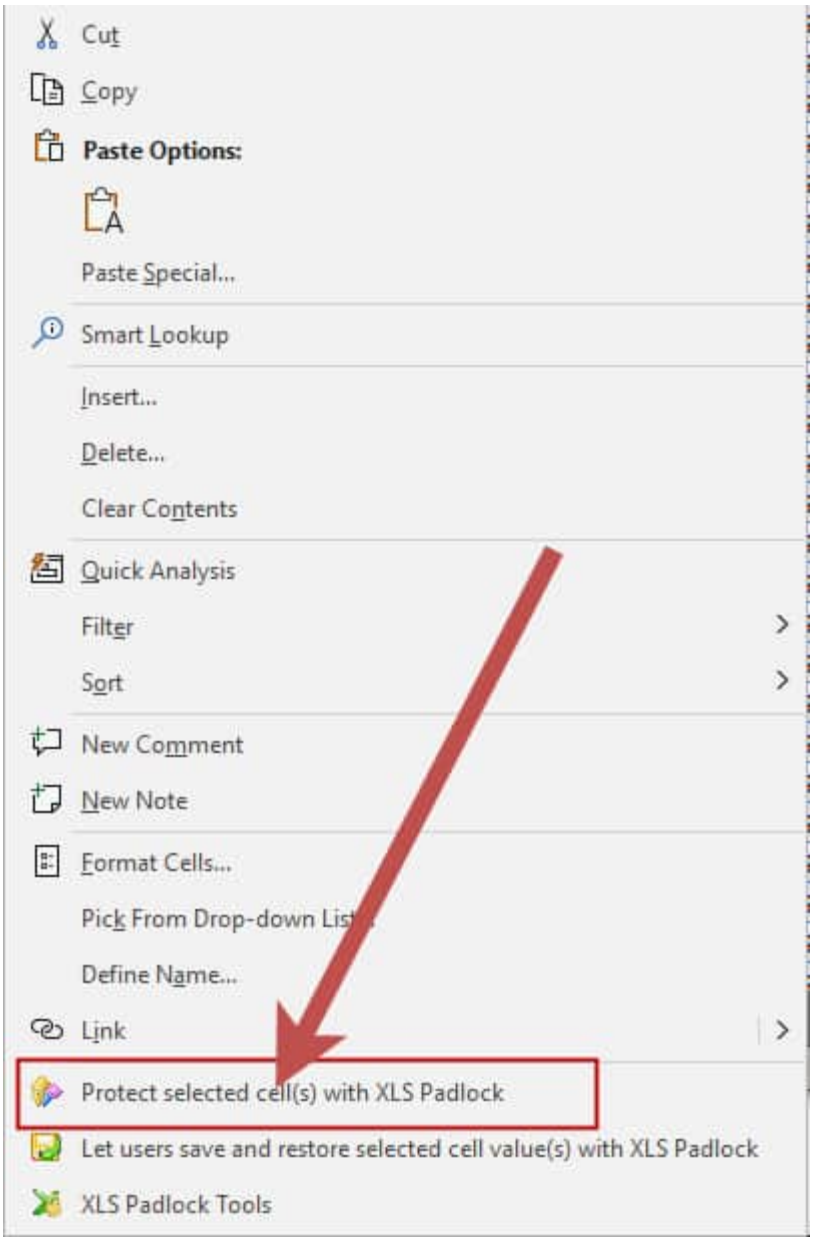
In additional to Excel's cell protection, **we strongly recommend you secure important formulas of your workbook with XLS Padlock's own formula protection**.

After protection with XLS Padlock, all formulas do not appear anymore **while remaining functional**. Instead a cryptic call PLEvalFormD is shown:



The goal is that if someone manages to recover the workbook from the EXE shell as explained in Improve protection of your workbooks, protected formulas will stop working. Thus, the workbook should become useless, because its **formulas protected by XLS Padlock are no more in the workbook file**!

XLS Padlock lets you decide which cells you want to protect (it's possible to **select several cells to protect in one time**).



- Learn more about formula protection and how to protect your formulas efficiently
- Disable common Excel add-ins | Real VBA code protection with VBA compiler

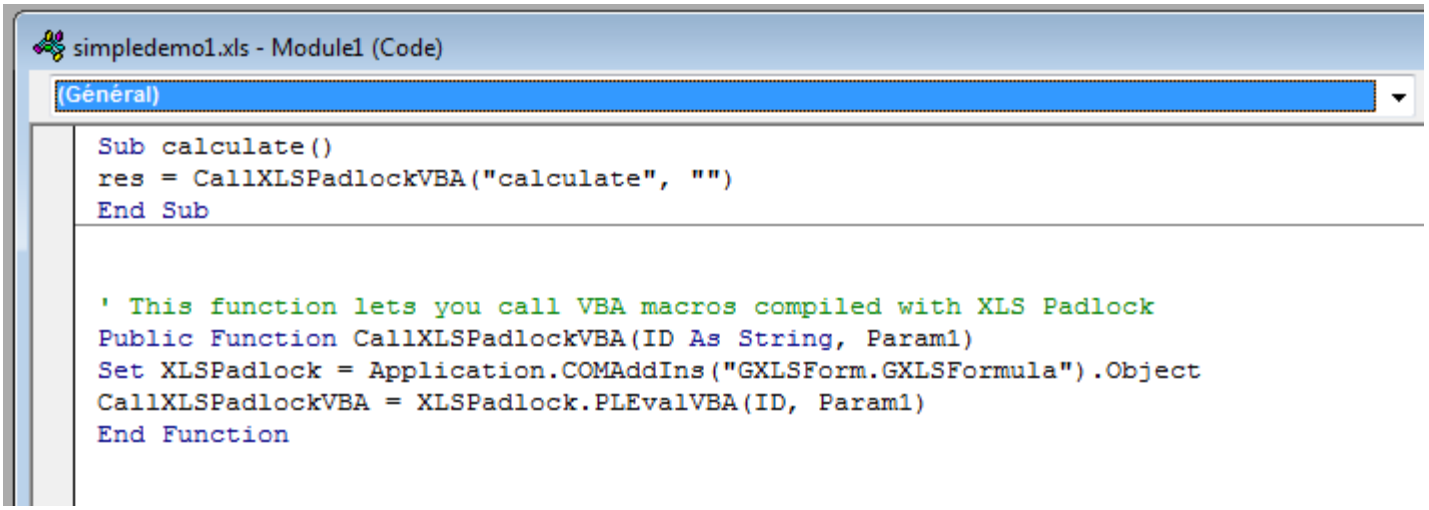
7.6. Real VBA code protection with VBA compiler

You can find some VBA obfuscators on the market, but obfuscation does not hide the logic of your VBA code.

XLS Padlock goes further: **it lets you remove VBA code from your workbook, while keeping the latter functional**! This is possible thanks to the integrated VBA compiler of XLS Padlock.

The XLS Padlock's VBA compiler turns parts of your VBA code into byte-code that can only be executed inside the secure application. Thus, if someone manages to gain access to the VBA code stored in your protected workbook as explained here, they won't see the entire original code, because some parts are simply missing: the equivalent byte-code generated by the VBA compiler is stored in the EXE shell.

On the screenshot below, the calculate() macro had original secret VBA code. We moved that entire sub code into the XLS Padlock VBA compiler. Now, in our compiled workbook, the calculate() macro just tells Excel to run the compiled bytecode thanks to a special internal macro provided by XLS Padlock (CallXLSPadlockVBA).



Thus, nobody can guess what was inside the calculate macro! The original VBA code does not exist in the workbook anymore. It's no more VBA code: it is bytecode that can't be copied to another Excel workbook. **That's a real protection!**

The counterpart is that you must manually transfer parts of VBA code you want to protect into a special VBA editor provided by XLS Padlock, as described in this manual §9 below.

The best way is to break your code into parts, and the most important parts (the ones you want to keep secret) should be compiled into bytecode to prevent your customers from copying them.

So, you don't have to move your entire VBA project to the VBA compiler. Only parts of code that should make

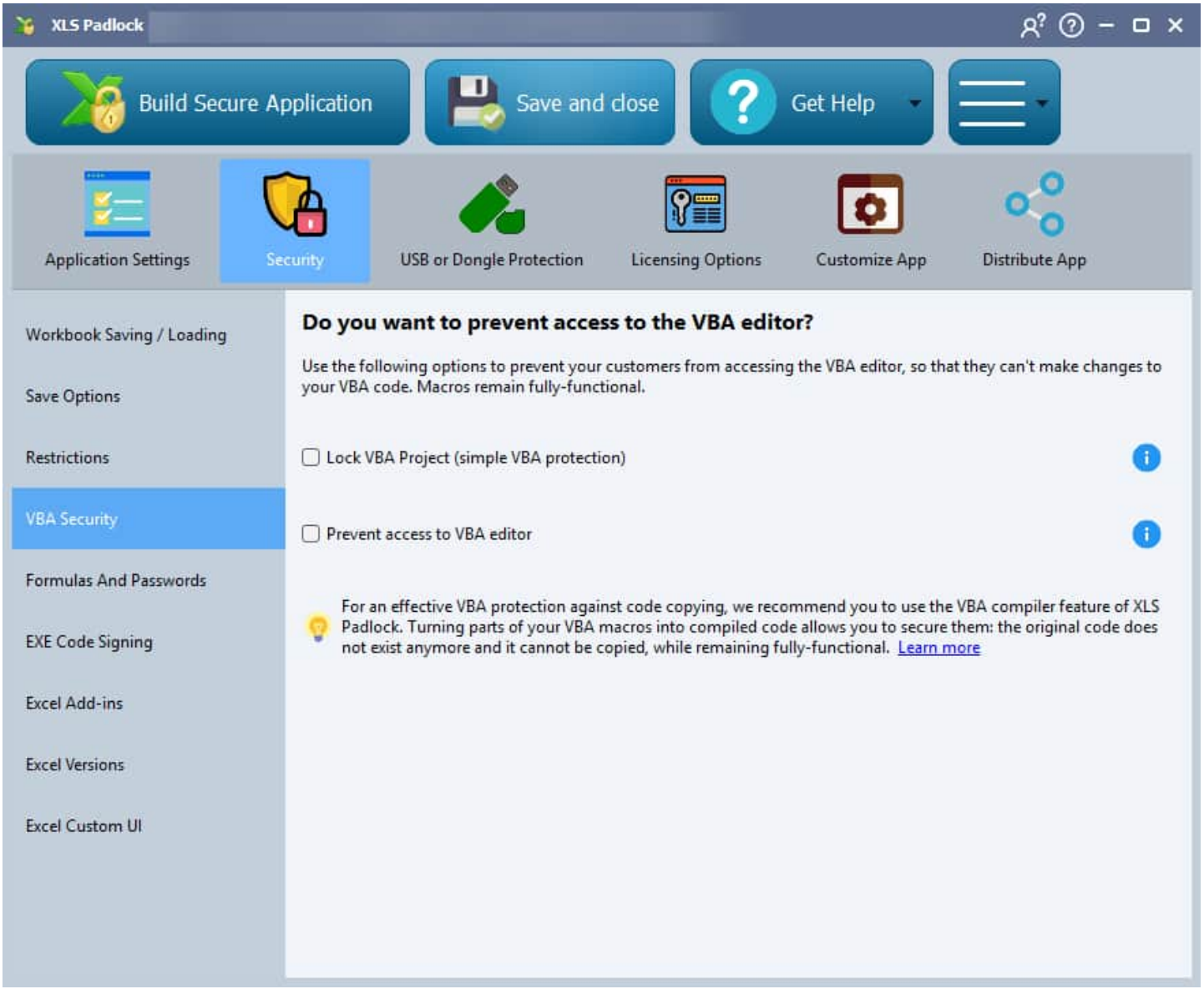
your Excel workbook non-functional if they are missing.

- Learn more about how to work with our VBA compiler
- Protect your formulas with XLS Padlock's formula protection

8. VBA Code Protection

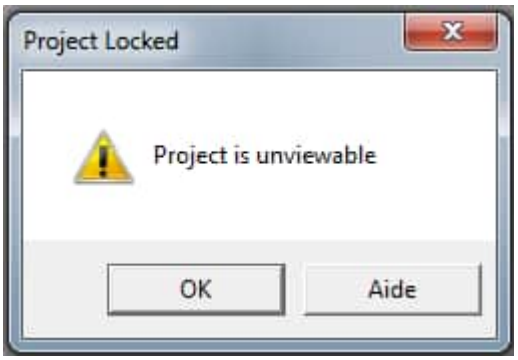
How to Protect VBA Code with XLS Padlock

VBA code can be vulnerable to tampering or easily copied, but XLS Padlock offers robust tools to secure it effectively. Below are two simple yet effective options of XLS Padlock for protecting your VBA code, plus an advanced solution for maximum security.



1. Lock Your VBA Project

XLS Padlock allows you to lock your VBA project, preventing users from viewing or editing your code in the Visual Basic Editor. This is a basic and quick method to ensure your code remains confidential. If someone tries to access the project, they'll receive an error message. This step is essential to deter casual attempts at viewing your code.



2. Prevent Access to the VBA Editor

This feature automatically blocks any attempt to access the VBA editor by closing it. While locking the project is effective, this step adds an extra layer by stopping the editor from opening entirely, even for users who know their way around Excel.

Ultimate Protection: VBA Code Compilation

For even further VBA code protection, XLS Padlock offers the option to **compile your VBA code into bytecode**. Unlike locking or restricting access, this converts your VBA code into a non-human-readable format, making it difficult to extract or reverse-engineer. This method goes beyond simple obfuscation, giving you the confidence that your code is fully protected.

Learn more about our VBA code compiler

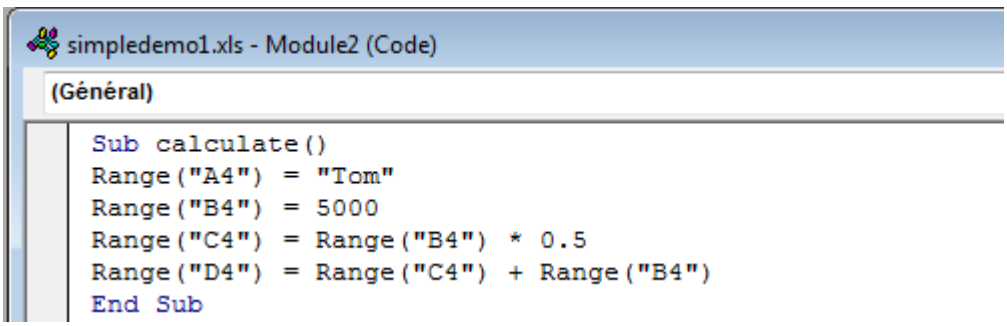
8.1. About the built-in VBA compiler

XLS Padlock features an **integrated VBA compiler: you can write Basic scripts and compile them into working byte-code not accessible to final users**.

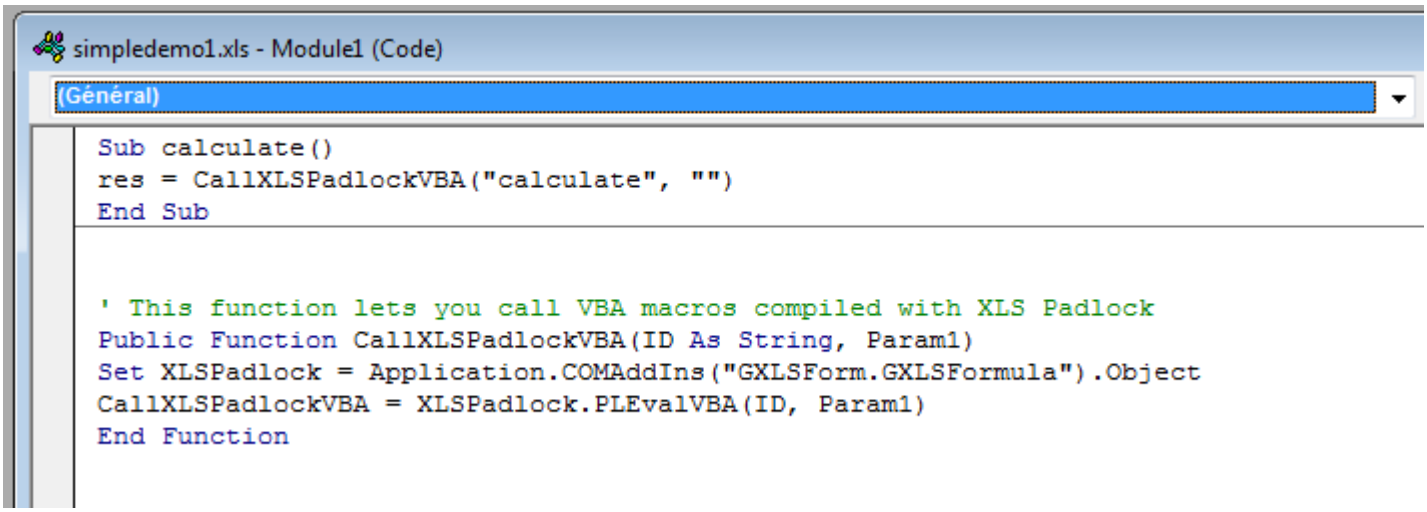
Turning parts of your VBA macros into compiled code allows you to secure them: the original code does not exist anymore, and it cannot be copied.

For a tutorial about compiling a VBA function, you can watch this video:
https://www.xlspadlock.com/video/vbacompiler

For instance, see this original code:

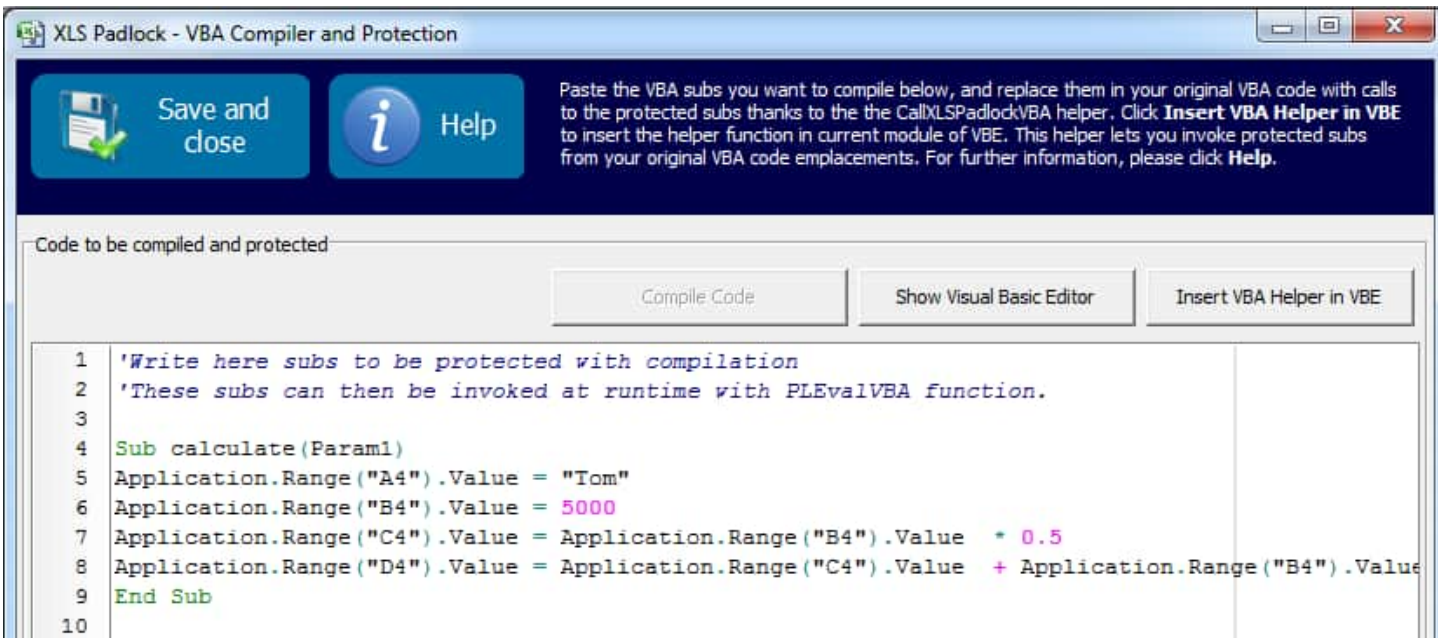


Once protected, it becomes:



As you can see, the original code in calculate() has been replaced by a call to XLS Padlock's built-in function named PLEvalVBA. This function executes the compiled bytecode.

The original code of calculate() has been moved into XLS Padlock VBA Editor and compiled there, as you can see below:



Comparing to the original code above, we had to make some modifications: indeed, the compiler requires the use of the **Application** object to access Excel objects (see Accessing Excel objects from compiled VBA code).

- Writing and compiling secure VBA code

Application's properties and methods accessible by the XLS Padlock VBA compiler are described in Microsoft Office Dev Center documentation: http://msdn.microsoft.com/en-us/library/office/ff194565.aspx

The compiler is not a simple obfuscator: it completely turns your VBA code into binary code and stores it securely in the application. Finally, you can optionally password protect your VBA project : since the original XLS file cannot be recovered, Excel password-cracking tools are useless.

Some limitations:

- You may have to make changes to your original VBA code in order to compile it. The XLS Padlock VBA

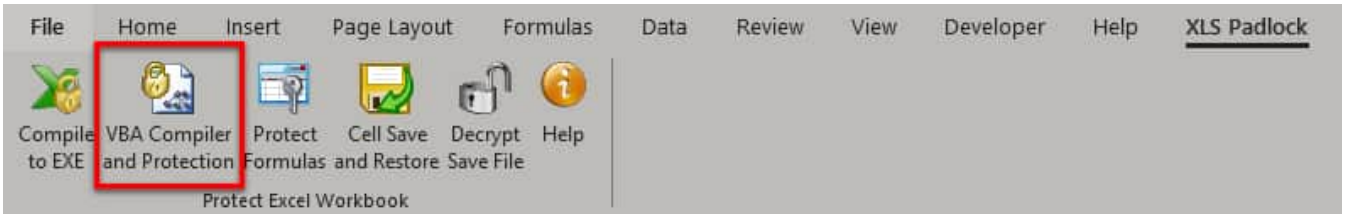
compiler is not as advanced as Microsoft VB interpreter.

- Entire macros generally cannot be compiled: only parts of them.
- Some objects and default properties are not accessible.

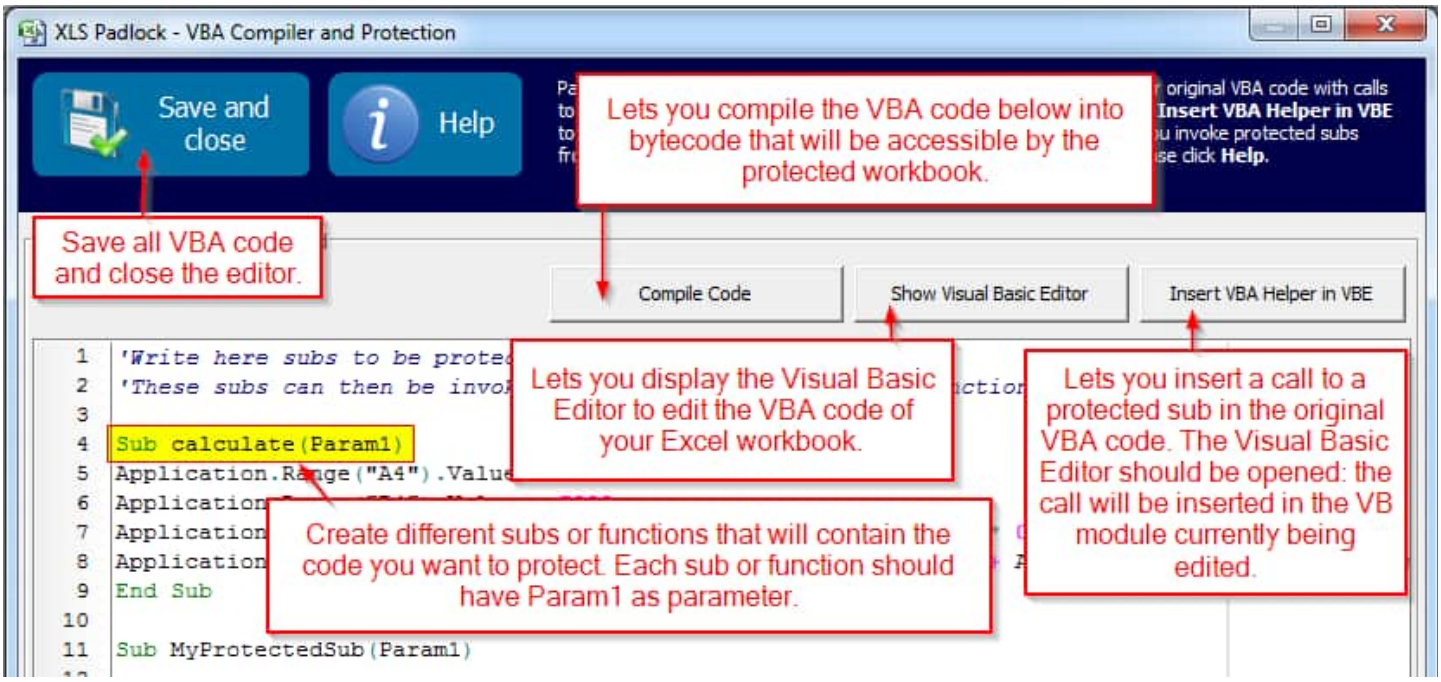
8.2. Writing and compiling secure VBA code

The goal is to **protect sensitive parts of your VBA code by compiling them into bytecode with XLS Padlock**. We recommend you to mix existing VBA code and compiled bytecode.

In Excel, click "VBA Compiler and Protection" to open the XLS Padlock VBA Editor and Compiler:



The XLS Padlock VBA editor is displayed:



In the text area with syntax highlighting, enter the VBA code that you want to compile. **You can group it into subs and functions**. It is possible to open the Visual Basic Editor by clicking "Show Visual Basic Editor" and to copy/paste your code.

By default, all subs and functions have one parameter named Param1. However, it's possible to use more

parameters.

When you are done, click "**Compile Code**". The VBA code is instantly compiled into bytecode that XLS Padlock will store until you compile your application EXE file.

If an error is detected, it is displayed, and the responsible code is underlined in red.

Click "Save and close" to close the editor.

Invoking compiled VBA code at runtime

8.3. Invoking compiled VBA code at runtime

XLS Padlock provides you with VBA functions to invoke the compiled VBA code in your workbook.

Open the Visual Basic Editor and copy/paste the following helper function into a module of your workbook:

```
' This function lets you call VBA macros compiled with XLS Padlock
Public Function CallXLSPadlockVBA(ID As String, Param1)
Dim XLSPadlock As Object
On Error Resume NextSet XLSPadlock =
Application.COMAddIns("GXLSForm.GXLSFormula").Object
CallXLSPadlockVBA = XLSPadlock.PLEvalVBA(ID, Param1)
End Function
```

Note: this function can be automatically inserted in your active VB module with the "Insert VBA Helper in VBE" button of the XLS Padlock's VBA editor.

The inserted CallXLSPadlockVBA helper function here takes two parameters:

- **ID** is the name of the compiled sub or function you want to invoke (the one you pasted in the VBA compiler).
- **Param1** is an optional parameter you want to pass to the compiled code.

Example: to invoke the "calculate" protected sub defined above in the VBA compiler, we open the original VBA module in Excel that contains the following code:

```
Sub calculate()
Range("A4") = "Tom"
Range("B4") = 5000
Range("C4") = Range("B4") * 0.5
Range("D4") = Range("C4") + Range("B4")
End Sub
```

And we replace it with a call to the CallXLSPadlockVBA helper function:

```
Sub calculate()
res = CallXLSPadlockVBA("calculate", "")
End Sub
```

As you can see, we provide the name of the protected sub we want to call and leave the second parameter to "" because it is not used.

The **res variable in the code above** holds the result of the call. It is only really used if the invoked code is a function. In that situation, it will contain the result of the function.

If you click Run to execute calculate() in VBE, calculate() sub still works, even if you replaced it. It is because **XLS Padlock can execute protected code even at design time**.

Note: the helper uses the `Application.COMAddIns("GXLSForm.GXLSFormula")` object. This COM object is always made accessible in a protected workbook on any computer, even if XLS Padlock is not installed.

Passing more parameters to the compiled VBA code

8.4. Passing more parameters to the compiled VBA code

By default, the helper function seen in "Writing and compiling secure VBA code" and used to run the compiled VBA code supports one parameter only.

```
' This function lets you call VBA macros compiled with XLS Padlock
Public Function CallXLSPadlockVBA(ID As String, Param1)
Dim XLSPadlock As Object
On Error Resume NextSet XLSPadlock =
Application.COMAddIns("GXLSForm.GXLSFormula").Object
CallXLSPadlockVBA = XLSPadlock.PLEvalVBA(ID, Param1)
End Function
```

The highlighted line shows that the PLEvalVBA method of the XLSPadlock object takes two parameters: **ID** and **Param1**.

This XPSPadlock object has other methods if you want to pass more parameters to the compiled VBA code:

- XLSPadlock.PLEvalVBA2(ID, Param1, Param2) for two parameters.
- XLSPadlock.PLEvalVBA3(ID, Param1, Param2, Param3) for three parameters.

It is even possible to pass more parameters by using arrays.

For instance, the helper function to pass two parameters to the compiled VBA code would be:

```
' This function lets you call VBA macros compiled with XLS Padlock
Public Function CallXLSPadlockVBA(ID As String, Param1, Param2)
Dim XLSPadlock As Object
On Error Resume NextSet XLSPadlock =
Application.COMAddIns("GXLSForm.GXLSFormula").Object
CallXLSPadlockVBA = XLSPadlock.PLEvalVBA2(ID, Param1, Param2)
End Function
```

- Passing arrays to the compiled VBA code

8.5. Passing arrays to the compiled VBA code

You can pass different variable types to the compiled VBA code, even static arrays.

Suppose you have the following VBA code in the VBA compiler:

```
Function TestMultipleParams(Param1, Param2, Param3)
  MsgBox( Param2[1] )
  TestMultipleParams = Param3 ^ 2
End Function
```

In the Excel VBA code, we can invoke it by passing the array.

Important: the array must be defined as Variant.

```
Sub MySubSample4()
Dim XLSPadlock As Object
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
Dim NomTableau(2) As Variant
NomTableau(0) = "a"
NomTableau(1) = "b"
NomTableau(2) = "c"
MsgBox XLSPadlock.PLEvalVBA3("TestMultipleParams", "Param1", NomTableau, 3)
Set XLSPadlock = Nothing
End Sub
```

- Accessing Excel objects from compiled VBA code

8.6. Accessing Excel objects from compiled VBA code

The compiler requires use of the **Application** object if you want to access Excel objects.

Application's properties and methods accessible by the XLS Padlock VBA compiler are described in Microsoft Office Dev Center documentation: http://msdn.microsoft.com/en-us/library/office/ff194565.aspx

Examples

- The Range object is the representation of a cell (or cells) on your worksheet. Use **Application.Range**

  In VBA: Range("A1:A4").Value = 2

  In compiled VBA: **Application.**Range("A1:A4").Value = 2

- Excel worksheet functions can be used thanks to **Application.WorksheetFunction**

```
Str1 = Application.WorksheetFunction.VLookup(Param1.ComboBox1,
Application.Range("A2:C8"), 2, False)
```

- Excel functions like InputBox **require all parameters to be specified** (default parameters are ignored). For InputBox, three parameters are required.

Original code:

```
Sub test()
Dim qty As Integer
Dim price, amount As Single
Range("A5") = "Item"
Range("B5") = "UnitPrice"
Range("C5") = "Quantity"
Range("D5") = "Amount"
ActiveCell.Offset(1, 0).Select
ActiveCell = InputBox("Enter the name of item")
ActiveCell.Offset(0, 1).Select
price = InputBox("Enter the price")
ActiveCell = price
ActiveCell.Offset(0, 1).Select
qty = InputBox("Enter the qty")
ActiveCell = qty
ActiveCell.Offset(0, 1).Select
amount = price * qty
ActiveCell = amount
ActiveCell.Offset(0, -3).Select
End Sub
```

Code entered in the XLS Padlock VBA editor and modified to be compatible with XLS Padlock compiler:

```
Sub test(Param1)
Dim qty As Integer
Dim price, amount As Single
Application.Range("A5").Value = "Item"
Application.Range("B5").Value = "UnitPrice"
Application.Range("C5").Value = "Quantity"
Application.Range("D5").Value = "Amount"
Application.ActiveCell.Offset(1, 0).Select
Application.ActiveCell = Application.InputBox("Enter the name of item", "Name", "")
Application.ActiveCell.Offset(0, 1).Select
```

```
price = Application.InputBox("Enter the price", "Price", "")
Application.ActiveCell = price
Application.ActiveCell.Offset(0, 1).Select
qty = Application.InputBox("Enter the qty", "Qty", "")
Application.ActiveCell = qty
Application.ActiveCell.Offset(0, 1).Select
amount = price * qty
Application.ActiveCell = amount
Application.ActiveCell.Offset(0, -3).Select
End Sub
```

Then in the original VB module, the sub test() is replaced by:

```
Sub test()
 res = CallXLSPadlockVBA("test", "")
End Sub
```

Another example which requires all parameters:

```
Application.Worksheets("My
data").Protect(1454511212,False,False,False,True,False,False,False,False,False,False,False,False,Fal
se,False,False)
```

Keywords and operators

8.7. Supported VBA syntax by compiler

8.7.1. Keywords and operators

Current Basic syntax supports:

- **sub .. end** and **function .. end** declarations
- **byref** and **dim** directives
- **if .. then .. else .. elseif .. end** constructor
- **for .. to .. step .. next** constructor
- **do .. while .. loop** and **do .. loop .. while** constructors
- **do .. until .. loop** and **do .. loop .. until** constructors
- **^ , * , / , and , + , - , or , <> , >=, <= , = , > , < , div , mod , xor , shl , shr** operators
- **try .. except** and **try .. finally** blocks
- **try .. catch .. end try** and **try .. finally .. end try** blocks

select case .. end select constructor

array constructors (x:=[1, 2, 3];)

exit statement

access to object properties and methods (**ObjectName.SubObject.Property**)

- Script structure

8.7.2. Script structure

Script structure is made of function and sub declarations.

```
SUB DoSomething
   CallSomething
END SUB

FUNCTION MyFunction
   MyFunction = "Ok!"
END FUNCTION

SUB DoSomethingElse
   CallSomething
END SUB
```

Statements in a single line can be separated by ":" character.

- Identifiers

8.7.3. Identifiers

Identifier names in script (variable names, function and procedure names, etc.) follow the most common rules in basic: begin with a character (a..z or A..Z), or '_', and can be followed by alphanumeric chars or '_' char. Names cannot contain any other character or spaces.

- Assign statements

8.7.4. Assign statements

Assign statements (assign a value or expression result to a variable or object property) are built using "=".

Examples:

```
MyVar = 2
Application.Range("C4").Value = "This " + "is ok."
```

- Comments

8.7.5. Comments

Comments can be inserted inside script. You can use ' chars or REM. Comment will finish at the end of line.

Examples:

```
' This is a comment before ShowMessage
ShowMessage("Ok")
REM This is another comment
ShowMessage("More ok!")
' And this is a comment
' with two lines
ShowMessage("End of okays")
```

- Variables

8.7.6. Variables

There is no need to declare variable in script but, if you want to, you declare variable just using DIM directive and its name.

```
SUB Msg(Param1)
   DIM S
```

```
   S = "Hello world!"
   ShowMessage(S)
END SUB

Sub Make(Param1)
DIM A
A = 0
A = A+1
ShowMessage(A)
END SUB
```

You can also declare global variables as private or public using the following syntax:

```
PRIVATE A
PUBLIC B
B = 0
A = B + 1
ShowMessage(A)
```

Variable declared with DIM statement are public by default. Private variables are not accessible from other scripts.

Variables can be default initialized with the following syntax

```
DIM A = "Hello world"
DIM B As Integer = 5
```

- Indexes

8.7.7. Indexes

Strings, arrays and array properties can be indexed using "[" and "]" chars. For example, if Str is a string variable, the expression Str[3] returns the third character in the string denoted by Str, while Str[I + 1] returns the character immediately after the one indexed by I.

More examples:

```
MyChar = MyStr[2]
MyStr[1] = "A"
MyArray[1,2] = 1530
```

- Arrays

8.7.8. Arrays

The compiler offers basic support for array constructors and variant arrays. To construct an array, use "[" and "]" chars. You can construct multi-index array nesting array constructors. You can then access arrays using indexes. If array is multi-index, separate indexes using ",".
If variable is a variant array, the compiler automatically supports indexing in that variable.

Arrays in the compiler are 0-based index. Some examples:

```
NewArray = [ 2,4,6,8 ]
Num = NewArray[1] //Num receives "4"
MultiArray = [ ["green","red","blue"] , ["apple","orange","lemon"] ]
Str = MultiArray[0,2] //Str receives 'blue'
MultiArray[1,1] = "new orange"
```

Dynamic arrays:

```
Sub MyProtectedSub(Param1)
' Create a dynamic array
DIM PTIM = VarArrayCreate([0,3000,0,5], 12)
' Assign a value:
PTIM[1,2] = 1530
'Show the value:
MsgBox (PTIM[1,2])
End Sub
```

If statements

8.7.9. If statements

There are two forms of if statement: if...then..end if and the if...then...else..end if. Like normal basic, if the if expression is true, the statements are executed. If there is else part and expression is false, statements after else are executed.

Examples:

```
IF J <> 0 THEN
 Result = I/J
END IF

IF J = 0 THEN
 Exit
ELSE
 Result = I/J
END IF
```

```
IF J <> 0 THEN
   Result = I/J
   Count = Count + 1
ELSE
   Done = True
END IF
```

If the IF statement is in a single line, you don't need to finish it with END IF:

```
IF J <> 0 THEN Result = I/J
IF J = 0 THEN Exit ELSE Result = I/J
```

- while statements

8.7.10. while statements

A while statement is used to repeat statements, while a control condition (expression) is evaluated as true. The control condition is evaluated before the statements. Hence, if the control condition is false at first iteration, the statement sequence is never executed. The while statement executes its constituent statement repeatedly, testing expression before each iteration. As long as expression returns True, execution continues.

Examples:

```
WHILE (Data[I] <> X)
 I = I + 1
END WHILE

WHILE (I > 0)
  IF Odd(I) THEN Z = Z * X
  X = Sqr(X)
END WHILE
```

- loop statements

8.7.11. loop statements

Possible syntax are:

```
DO WHILE expr statements LOOP
```

```
DO UNTIL expr statements LOOP
DO statements LOOP WHILE expr DO statement LOOP UNTIL expr
```

Statements will be executed WHILE expr is true, or UNTIL expr is true. if expr is before statements, then the control condition will be tested before iteration. Otherwise, control condition will be tested after iteration. Examples:

```
DO
   K = I mod J
I = J
J = K
LOOP UNTIL J = 0

DO
I = I mod J
I = J
J = K
LOOP WHILE J <> 0

DO WHILE I < 0
 ...
LOOP
```

- for statements

8.7.12. for statements

VBA Code Protection > Supported VBA syntax by compiler > for statements

Syntax:

```
FOR counter = initialValue TO  finalValue STEP stepValue
Statements
NEXT
```

For statement set counter to initialValue, repeats execution of statement until "next" and increment value of counter by stepValue, until counter reachs finalValue.
Step part is optional, and if omitted stepValue is considered 1.

Examples:

SCRIPT 1:

```
FOR c = 1 TO 10 STEP 2
 a = a + c
NEXT
```

SCRIPT 2:

```
FOR I = a TO b
 j = i ^ 2
sum = sum + j
NEXT
```

- select case statements

8.7.13. select case statements

Syntax:

```
SELECT CASE selectorExpression
  CASE caseexpr1      statement1
  …
  CASE caseexprn      statementn
CASE ELSE
   elsestatement
END SELECT
```

If selectorExpression matches the result of one of caseexprn expressions, the respective statements will be executed. Otherwise, elsestatement will be executed. Else part of case statement is optional.
Example:

```
SELECT CASE uppercase(Fruit)
CASE "lime"
 ShowMessage("green")
CASE "orange"
 ShowMessage("orange")
CASE "apple"
 ShowMessage("red")
CASE ELSE
 ShowMessage("black")
END SELECT
```

- function and sub declaration

8.7.14. function and sub declaration

Declaration of functions and subs are similar to basic. In functions to return function values, use implicit declared variable which has the same name of the function, or use Return statement. Parameters by reference can also be used, using **BYREF** directive.

Some examples:

```
S
UB HelloWord
   ShowMessage("Hello world!")
END SUB

SUB UpcaseMessage(Msg)
   ShowMessage(Uppercase(Msg))
END SUB

FUNCTION TodayAsString
  TodayAsString = DateToStr(Date)
END FUNCTION

FUNCTION Max(A,B)
   IF A>B THEN
      MAX = A
   ELSE
      MAX = B
   END IF
END FUNCTION

SUB SwapValues(BYREF A, B)
  DIM TEMP
  TEMP = A
  A = B
  B = TEMP
END SUB
```

You can also declare subs and functions as private or public using the following syntax:

```
PRIVATE SUB Hello
END SUB


PUBLIC FUNCTION Hello
END FUNCTION
```

Subs and functions are public by default.
You can use **Return** statement to exit subs and functions. For functions, you can also return a valid value.

Examples:

```
SUB UpcaseMessage(Msg) ShowMessage(Uppercase(Msg))
 Return
 'This line will be never reached
 ShowMessage("never displayed")
END SUB
```

```
FUNCTION TodayAsString
 Return DateToStr(Date)
END FUNCTION
```

Handling errors in VBA compiler

8.8. Handling errors in VBA compiler

Runtime errors can occur when your compiled VBA code is executed. In VBA, you use the VBA "**On Error**" statement for error handling. This statement performs some action when an error occurs during runtime.

XLS Padlock's VBA compiler does not rely on the "On Error" event but provides a simply construct for wrapping code with error handling. When an error occurs in the wrapped code (or anything it calls), the code will jump to the error handling part of the wrapping code:

```
Try
     ...
     The code we want to execute
     ...
Except
     ...
     This code gets executed if an exception occurs in the above block
     ...
End
```

An error which is not handled by your code will be displayed by the VBA compiler at runtime, unless you disable this option.

Code example

```
NumberStr =""
if InputQuery("Input", "Type an integer number from 1 to 7", NumberStr) then

   try
     Number=StrToFloat(NumberStr)
   except
      raise("Not a valid number")
   end

   select case Number
    case  1
      ShowMessage("One")
    case 1+1
      ShowMessage("Two")
```

```
    case 4.5/1.5
     ShowMessage("Three")
    case 2*2
     ShowMessage("Four")
    case Length("xxxxx")
     ShowMessage("Five")
   case 3+3, 3+4
      ShowMessage("Six or Seven")
   case else
      ShowMessage("You did not type an integer from 1 to 5")
   end select

end if
```

- Hide and lock your VBA code in Excel

8.9. Hide and lock your VBA code in Excel

XLS Padlock lets you **lock the VBA project**: see Lock VBA Project (simple VBA protection).

If you do not want to use this feature, you can still hide and lock your VBA code in Excel by following these steps:

1. Click “Tools”, “Macro” and then “Visual Basic Editor”. On recent Office versions, use the “Developer” ribbon.
2. In the Visual Basic editor window, choose “Tools” => “VBA Project Properties”.
3. In that dialog, click the “Protection” tab and enable the “Lock project for viewing” option and enter a password. Once the workbook is saved and closed, the protection will take effect, and the user will not be able to view or edit the VBA code.

It is said that protection in Excel is very weak and there are tools to remove password protection.

Thanks to XLS Padlock, these tools are useless because they require access to the original XLS file. However, once compiled to an EXE with XLS Padlock, your XLS file is secured and can’t be cracked by these password remover tools.

Thus, your VBA code remains “safe”.

- Disable debug information in case of compiler error

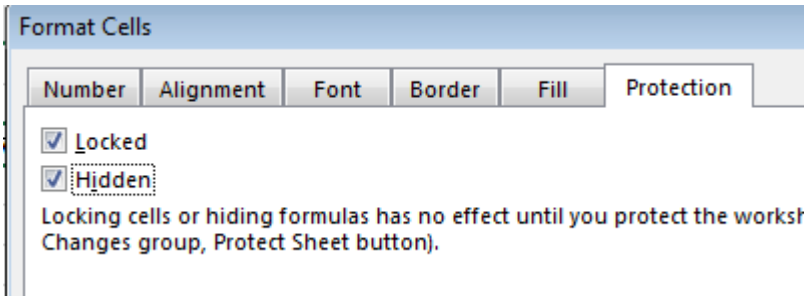
8.10. Disable debug information in case of compiler error

By default, if an error is detected in the compiled code by the VBA compiler, an error is displayed at runtime with a lot of information in order to help you to fix the problem. If you do not want to display this information, for some reason, you can disable it by going to Configure Advanced Options and turn off this option: "**Do not show VBA compiler exception messages**".

9. Protect Formulas

9.1. Excel versus XLS Padlock cell protection

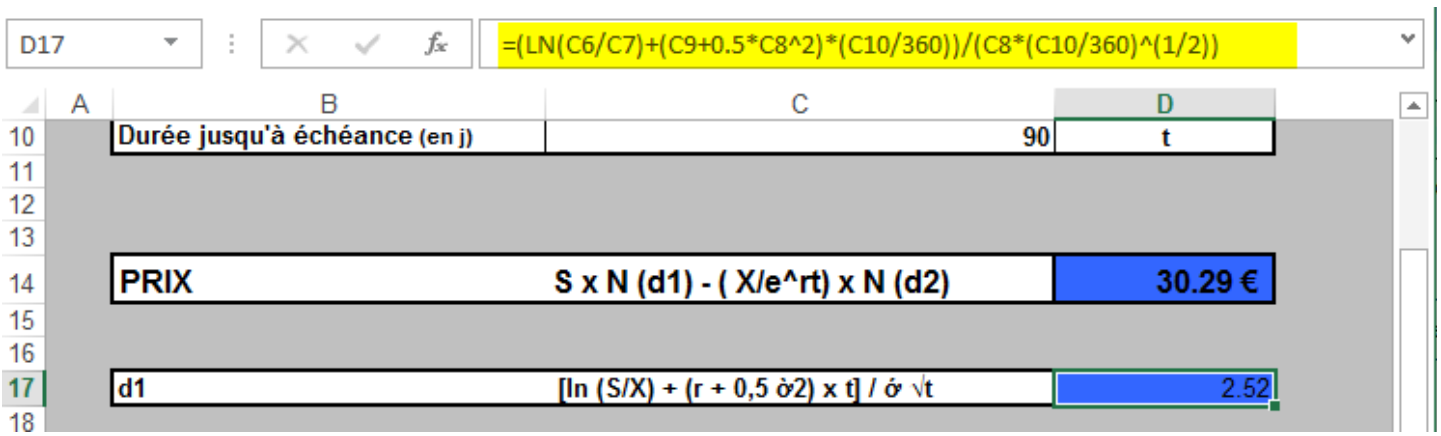
Excel lets you hide formulas thanks to a password:



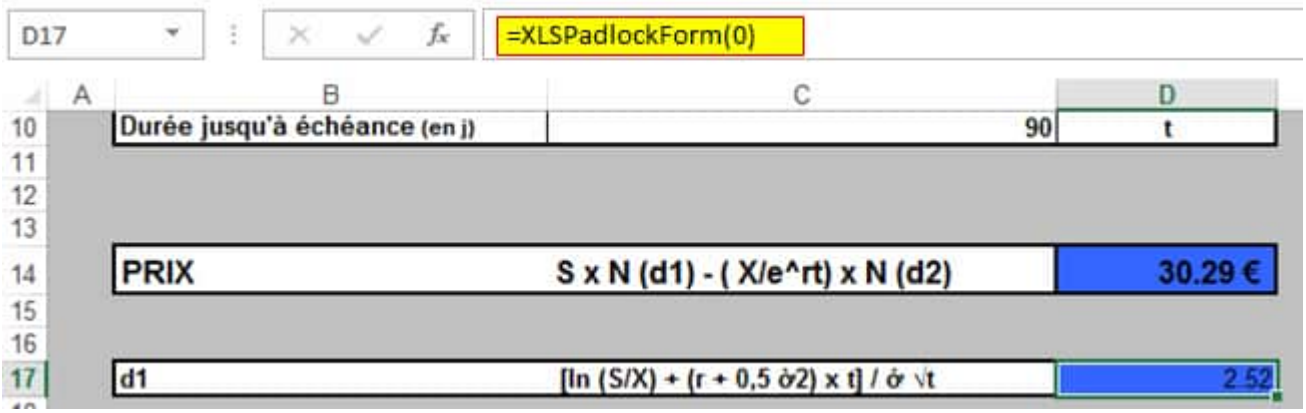
XLS Padlock lets you go further: it can protect **any of your formulas in any worksheet** by replacing it with a security function **only available when your secure application runs**.

In the compiled workbook, formulas will be replaced by XLS Padlock in the Formula Bar by anonymous functions such as **`=PLEvalFormD(0;0), =PLEvalFormD(1;COUNT($F7))…`**
However, calculations will still work as expected!

Before protection, anyone can see your formula:

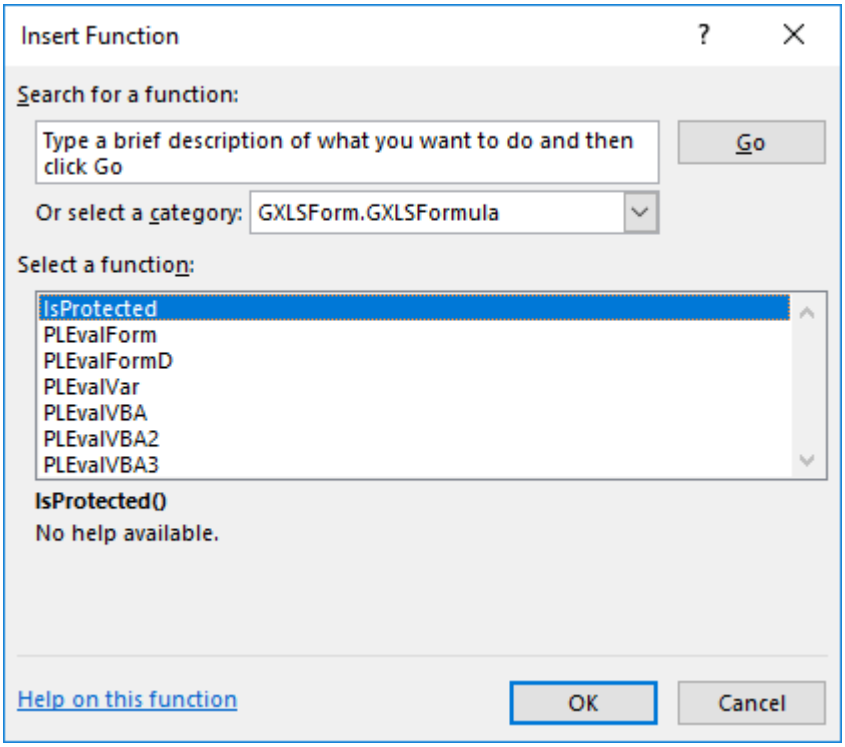


After protection with XLS Padlock, the formula is automatically replaced while remaining functional:



This increases the security of your workbook: protected cells can only work properly if the workbook is opened in the secure application made with XLS Padlock.

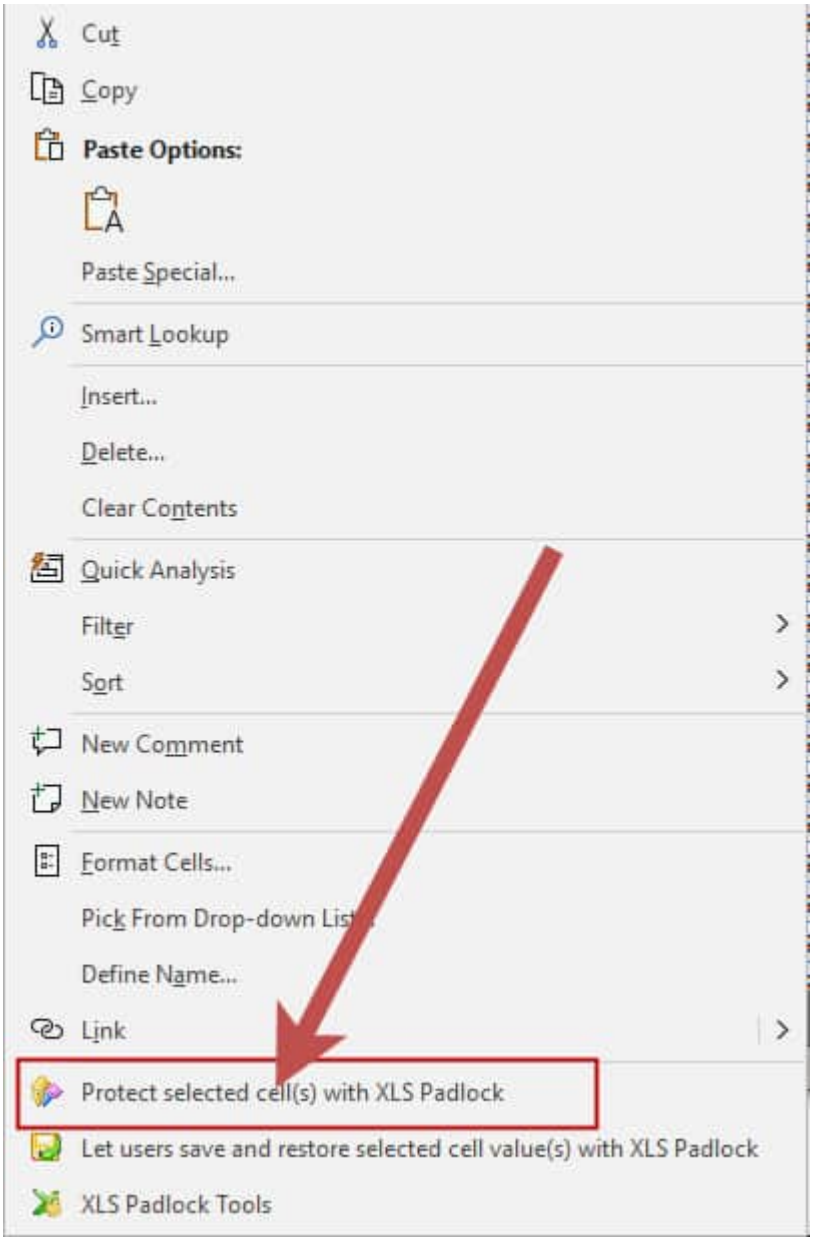
Although XLS Padlock automatically manages protected cells itself, you can see that the security functions as PLEvalForm and PLEvalFormD are also recognized by Excel at design time:



Protecting cells with XLS Padlock

9.2. Protecting cells with XLS Padlock

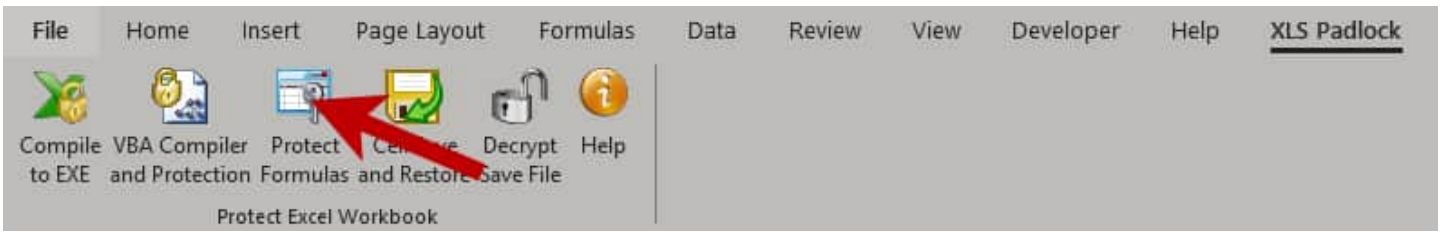
To protect a formula, you just have to **right click on one or more cells** with the formulas you want to hide and select "**Protect selected cell(s) with XLS Padlock**" in the mouse's context menu.



XLS Padlock will then tell you that the cells will be protected.

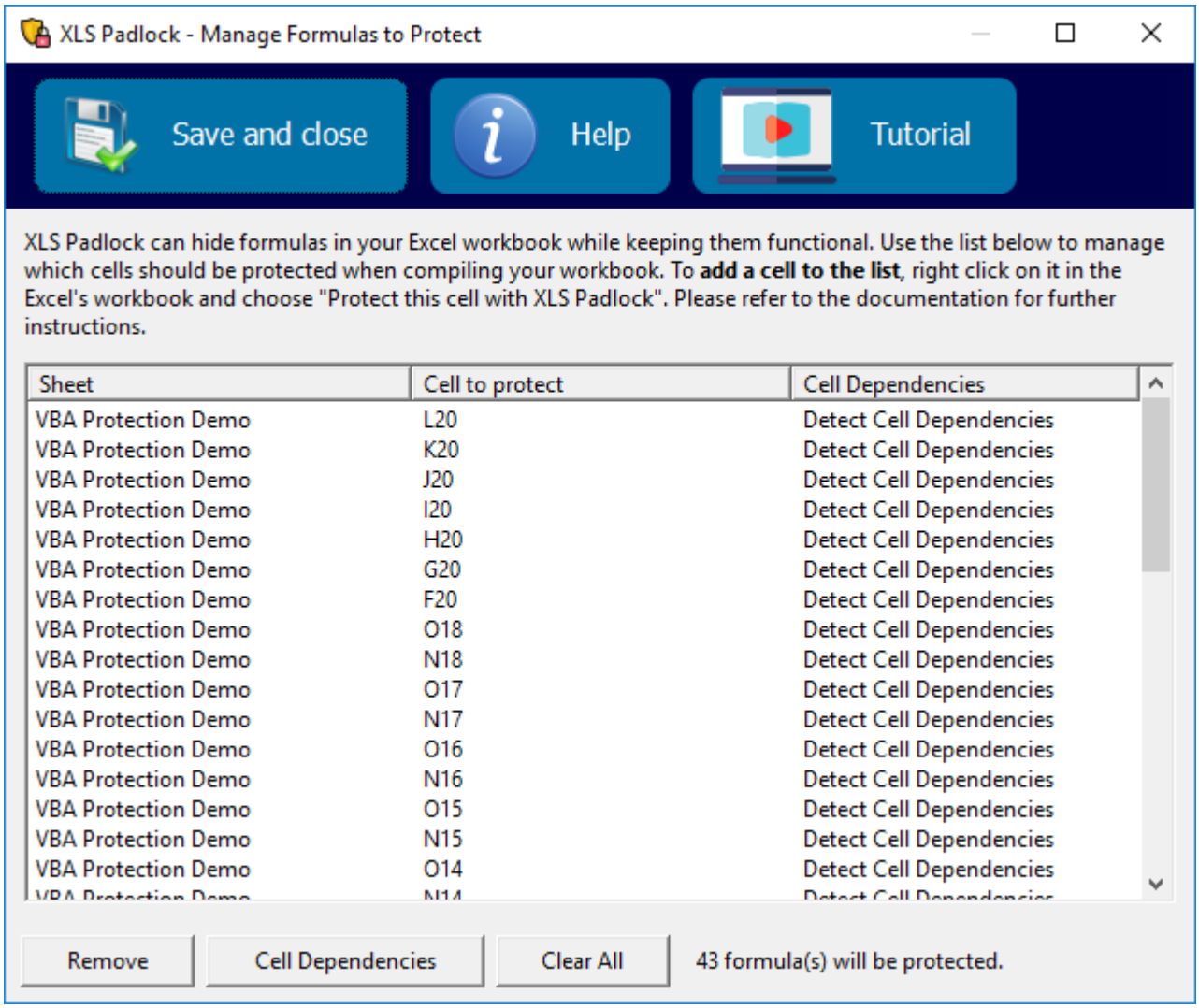
Overview of protected cells

You can have an overview of all the protected cells by clicking "**Protect Formulas**" in XLS Padlock tab or menu:



Clicking "**Protect Formulas**" will open the list of the cells you have configured to be protected. In this window, you can configure the behavior of the protection, remove cell protection or clear the entire list if needed.

When compiling your workbook, XLS Padlock will replace all listed cells with generic `PLEvalForm(N)` and `PLEvalFormD(N, ....)` function calls. **Your cells remain functional, but end users can't discover formulas behind**. In fact, original formulas do not exist in the compiled workbook anymore: they are managed by the EXE itself.



Cell Dependencies

The **Cell Dependencies** button lets you control how the protection is applied, two choices being available: "Detect Cell Dependencies" and "No".

By default, XLS Padlock will detect all cell references and range names in formulas (called cell dependencies) and generate an anonymous function that contains these cell references. Thus, Excel knows how to recalculate protected cells properly. For instance, suppose the formula to protect is `= A3^2`, XLS Padlock will generate this function: `PLEvalFormD(1, COUNT(A3))`.
If XLS Padlock fails to protect a cell, you can choose "**No**" for its "Cell Dependencies" setting. Thus, a simple generic PLEvalForm(N) function will be used.

Some complex formulas are not supported by the protection, and will fail (displaying #Error! or #Value! in the cell). Please test your formulas before distributing your protected workbook. Especially:

- The formula length must be less than 256 characters.
- INDIRECT is not supported.
- Formulas must not contain any VBA user-defined function, only regular Excel functions.

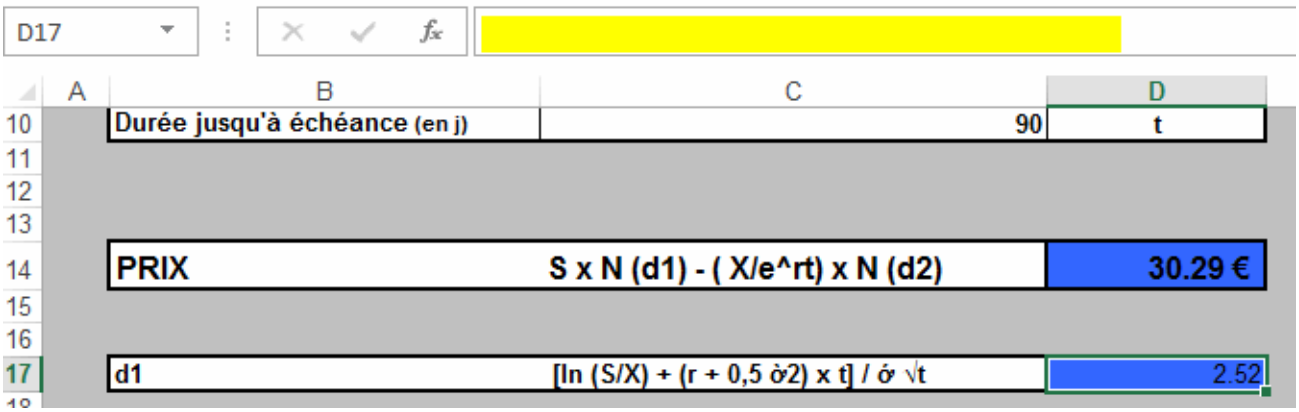
- Formula protection should be used sparingly. Only protect important formulas on which your Excel workbook depends. Protecting several thousands of formulas is not recommended, because it will make your protected XLS file larger and can slow Excel on old computers.

- **Cell data validation and Conditional Formatting Rules are not supported**.

Combining Excel sheet protection and XLS Padlock protection

9.3. Combining Excel sheet protection and XLS Padlock protection

XLS Padlock protection also works with Excel sheet protection (hidden or locked cells). In that situation, protected cells will not display anything in the Formula Bar. Generic PLEvalForm(N) / PLEvalFormD(N) function calls are not visible.



Note that protected formulas that have dependencies **will fail if the dependencies are hidden**. **Avoid the Excel's hidden attribute** in this case.

Disable Formula protection

9.4. Disable Formula protection

Formula protection is incompatible with Excel 2000. If you want to create applications that work on Excel 2000, you should tick this option in Compile to Exe → Security page.

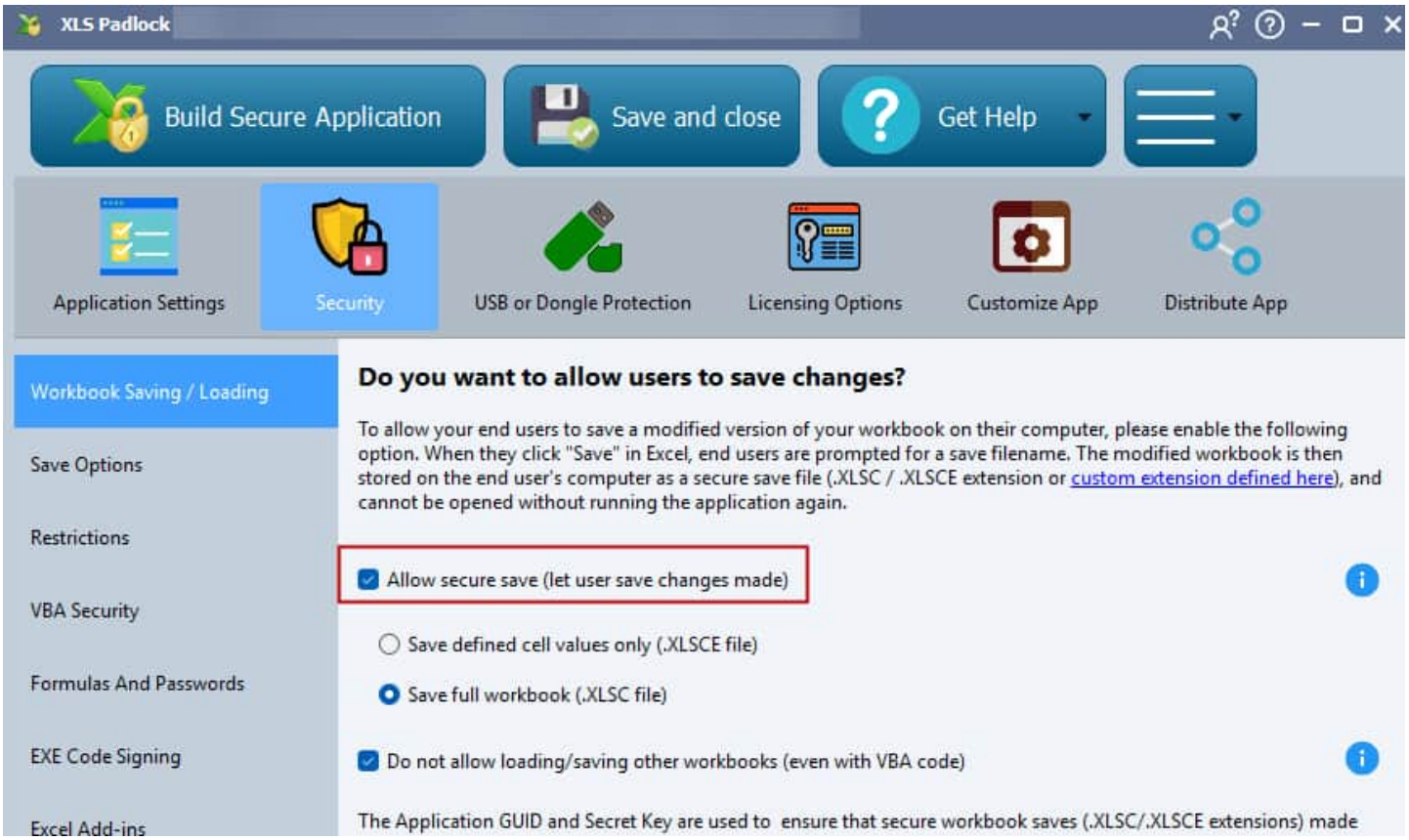
Note: this will also disable the VBA code compiler.

10. Workbook Saving / Loading

With XLS Padlock, you decide whether you let users save their changes made to your Excel workbook or not.

How XLS Padlock saves and reloads user changes

- Tick or untick the option "**Allow secure save**" in the Security page:



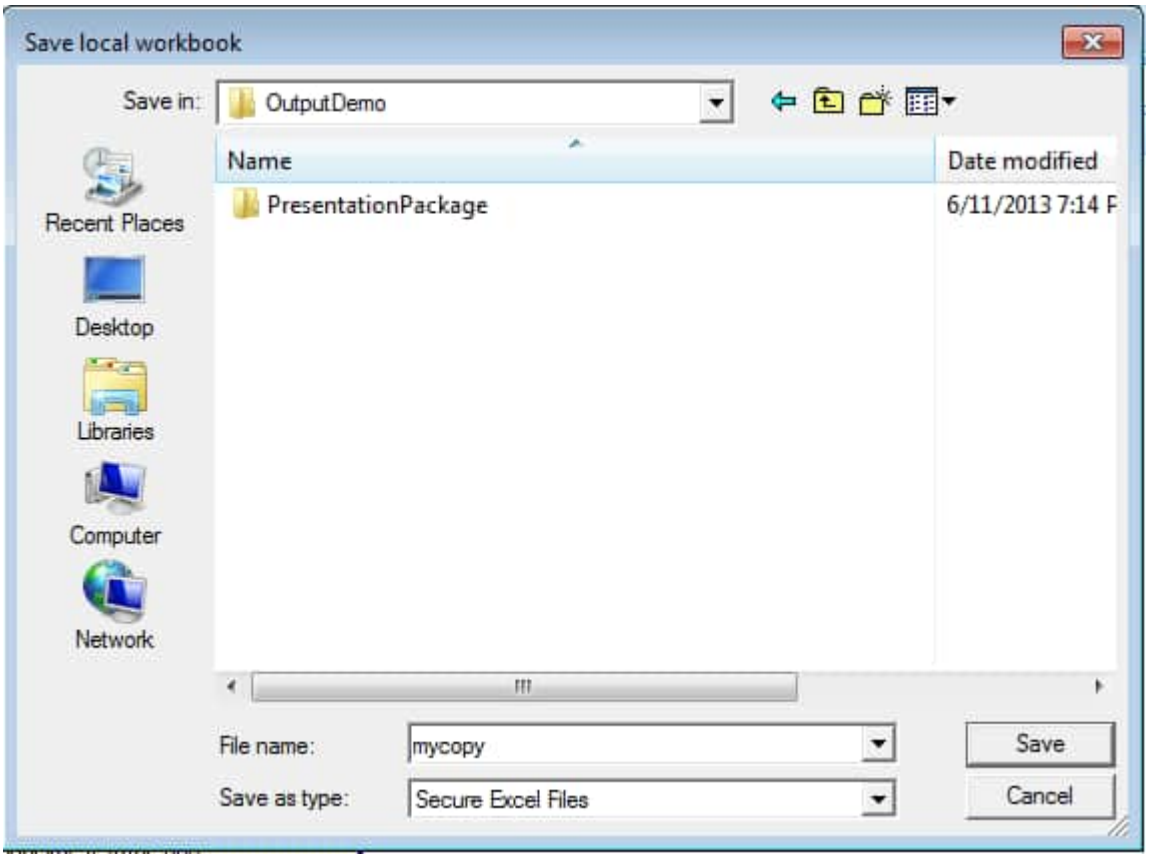
→ *If you disable this option*, your application will never be modified. Customers can use worksheets, write, delete… but when they close the application, changes are discarded. Note that the Save button is not disabled in some Excel versions, users can click it and the saving process seems to work; but in reality, changes are ignored.

→ *If you enable this option*, your customers will be able to modify your workbook and save the changes.

You will then have to select the saving mode.

- To save changes once the compiled workbook is opened, end users can click the usual Save button in Excel or use "File => Save" menu.

The "Save As" dialog box will then appear, asking customers where they want to save a secure copy of the workbook:



The workbook with changes made by your customers will be securely stored at the location specified by the customer in a file.

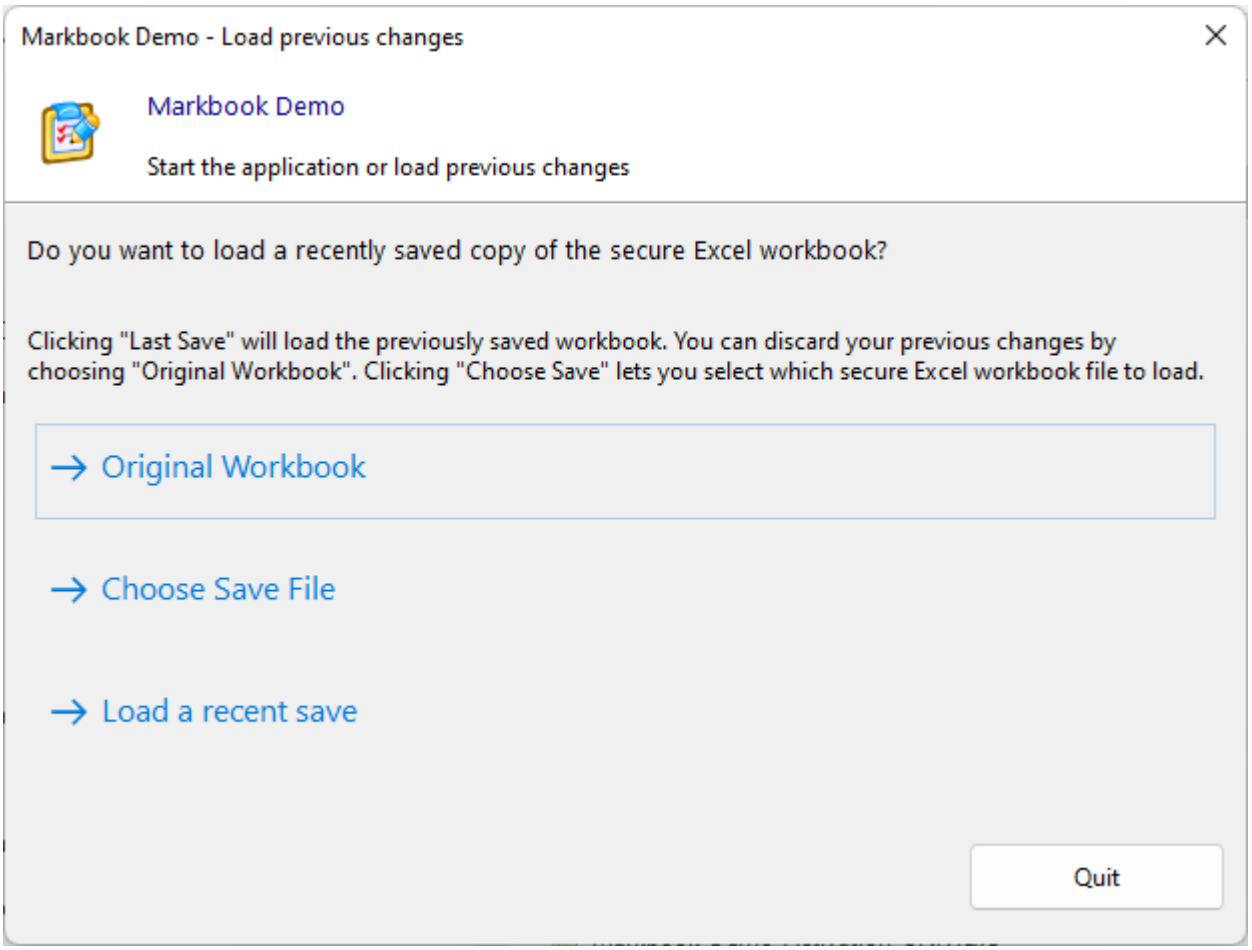
This save file is given the extension .XLSC (or .XLSCE) and **can't be opened without running the secure application again**.

By default, the dialog box will ask end users if they want to overwrite their changes or not. You can disable this behavior in the "Advanced Options".

You can define your **own custom extension** for save files. Use the "Advanced Options" to define it.

How can end users load their changes?

The next time they run your application, customers will have the choice to open your original workbook or to open a version they already saved: "Choose Save File" or load a recent save. Clicking Load a recent save displays a menu listing the recent saves made by the end user. Only existing save files are listed. If the user removed a save file, it will not appear in the list.



By default, the "Save As" dialog box is displayed each time a user clicks . The last file saved is automatically remembered.

Note: only the Save icon is working in Excel ("Save As" is always disabled or not working).

VBA Code Protection | Workbook Access Control

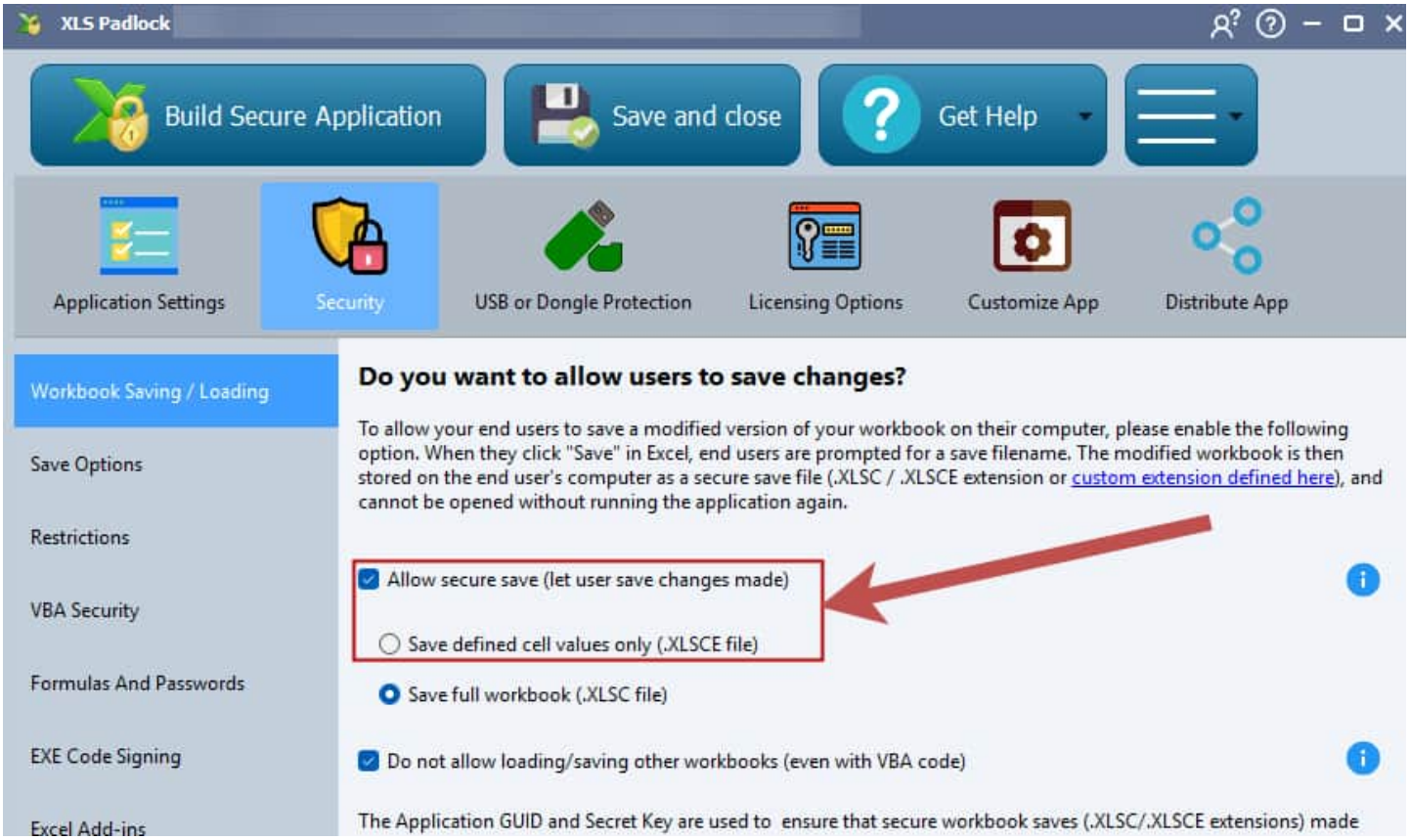
10.1. Save Mode: Full or Cell Values

If you let users save their changes made to your Excel workbook, you have to **define the way XLS Padlock will store (and restore) these changes**.

The XLS Padlock software **offers two different saving modes:**

- Save full workbook (.XLSC file)
- Save defined cell values only (.XLSCE file)

The mode can be selected on the Workbook Saving/Loading page. **Let's review them:**



Save full workbook (.XLSC file)

A full copy of the workbook with changes made by your customers will be securely stored in a file at the location specified by them. This file is given the extension **.XLSC** (this can be customized) and can't be opened without running the compiled workbook again. **Thus, your workbook remains safe**. Excel itself is unable to recognize the .XLSC proprietary format.

The **Full Save mode** encrypts and saves the entire workbook exactly as it is at the moment of the user's save. So, if an end user modifies their workbook, saves it and then you distribute a new EXE with updates, when the user opens their previous save file, they will retrieve their modifications, not your updates. This is because their saved file is an exact snapshot of the workbook at the time of the save. Your updates will be in the compiled workbook embedded in the new EXE, but not in the end user's saved file.

Full save files can be shared between users (unless you decide to lock them to a particular machine). It is even quite possible for customers to send you their save files, so that you can decrypt them and reuse data provided by your customers.
Again, these save files can only be opened by your application and not from someone's else: this is possible because XLS Padlock uses a **unique secret key** (that you will have defined as explained here) to generate save files specific to each application. So, another user of XLS Padlock will not be able to open your save files unless he has the secret key (though, of course, you will not reveal it to anyone).

The Full workbook saving mode is the default choice for a majority of XLS Padlock users, especially it should be used for workbooks on which your final users will make a lot of changes.

The main drawback of this saving mode is that, when you deploy an updated EXE file, XLSC save files made previously by customers won't be automatically upgraded. Instead, customers will see the last changes they made and not yours. That's the expected behavior since we restore the full workbook saved by the customer. In

that situation, you can rely on the second saving mode as explained below.

Save defined cell values only (.XLSCE file)

With this save mode selected, **only the values of certain cells that you defined before compiling** the workbook will be kept and restored when a save file is loaded.

Thus, if you frequently modify your source workbook and distribute these new versions to your customers, they will benefit from the updates while having the possibility to reload the values of the cells they themselves have modified.

The main disadvantage is that you have to tell XLS Padlock which cells you want your customers to be able to save. This must be done before compiling and distributing your application thanks to the Excel's context menu command named "Let users save and restore selected cell value(s) with XLS Padlock".

When customers click the Excel's Save button, the application will read all specified cells and store their values securely in a file at the location specified by them. This file is given the extension **.XLSCE** (this can be customized except the final E letter) and can't be opened without running the compiled workbook again. **Thus, even in this specific save mode, your workbook remains safe**. Excel itself is also unable to recognize the .XLSCE proprietary format.

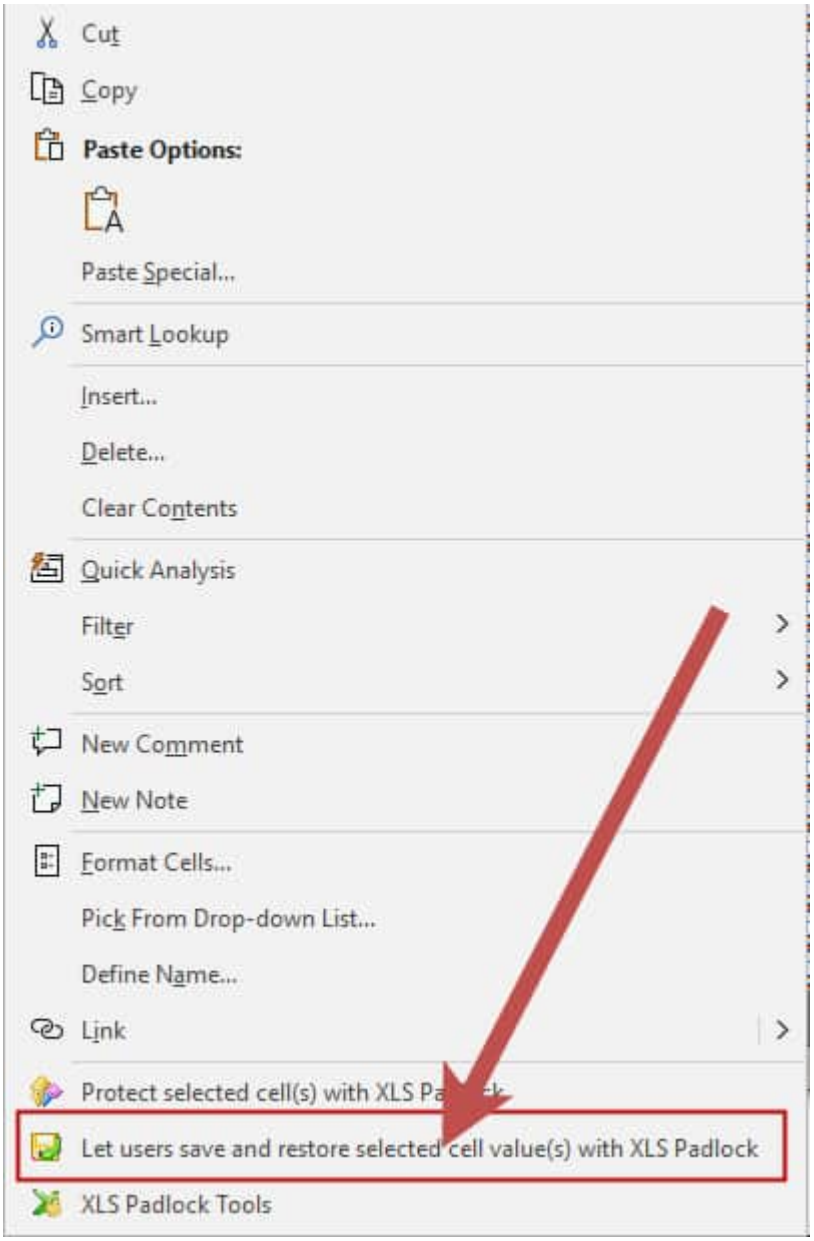
It is moreover possible for customers to send you their save files, so that you can decrypt them and load cell values saved by your customers into the currently loaded Excel workbook.

Finally, these XLSCE save files can only be opened by your application and not from someone's else, since XLS Padlock uses a **unique secret key** (that you will have defined as explained here) to generate save files specific to each application. So, another user of XLS Padlock will not be able to open your save files unless he has the secret key (though, of course, you will not reveal it to anyone).

10.1.1. How to define cells to be saved and restored

If you chose the Save defined cell values only mode, you must select the cells that should be saved and restored by your application.

To select cells to be saved, you just have to **right click on one or more cells** and click "**Let users save and restore selected cell value(s) with XLS Padlock**" in the mouse's context menu.

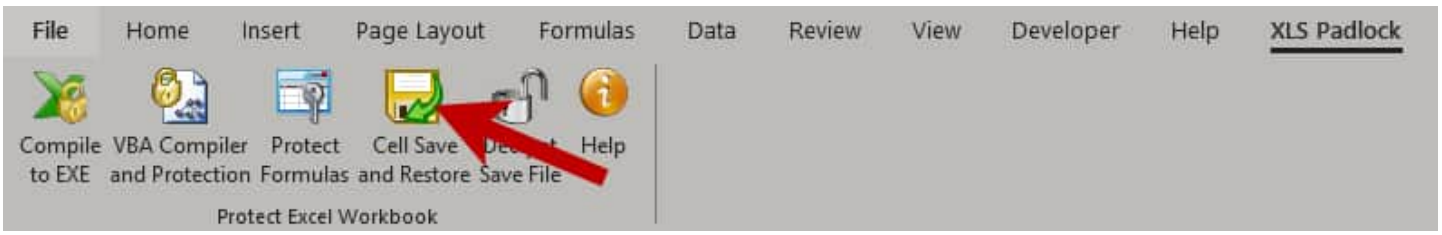


XLS Padlock will then tell you that the cells will be saved and restored.

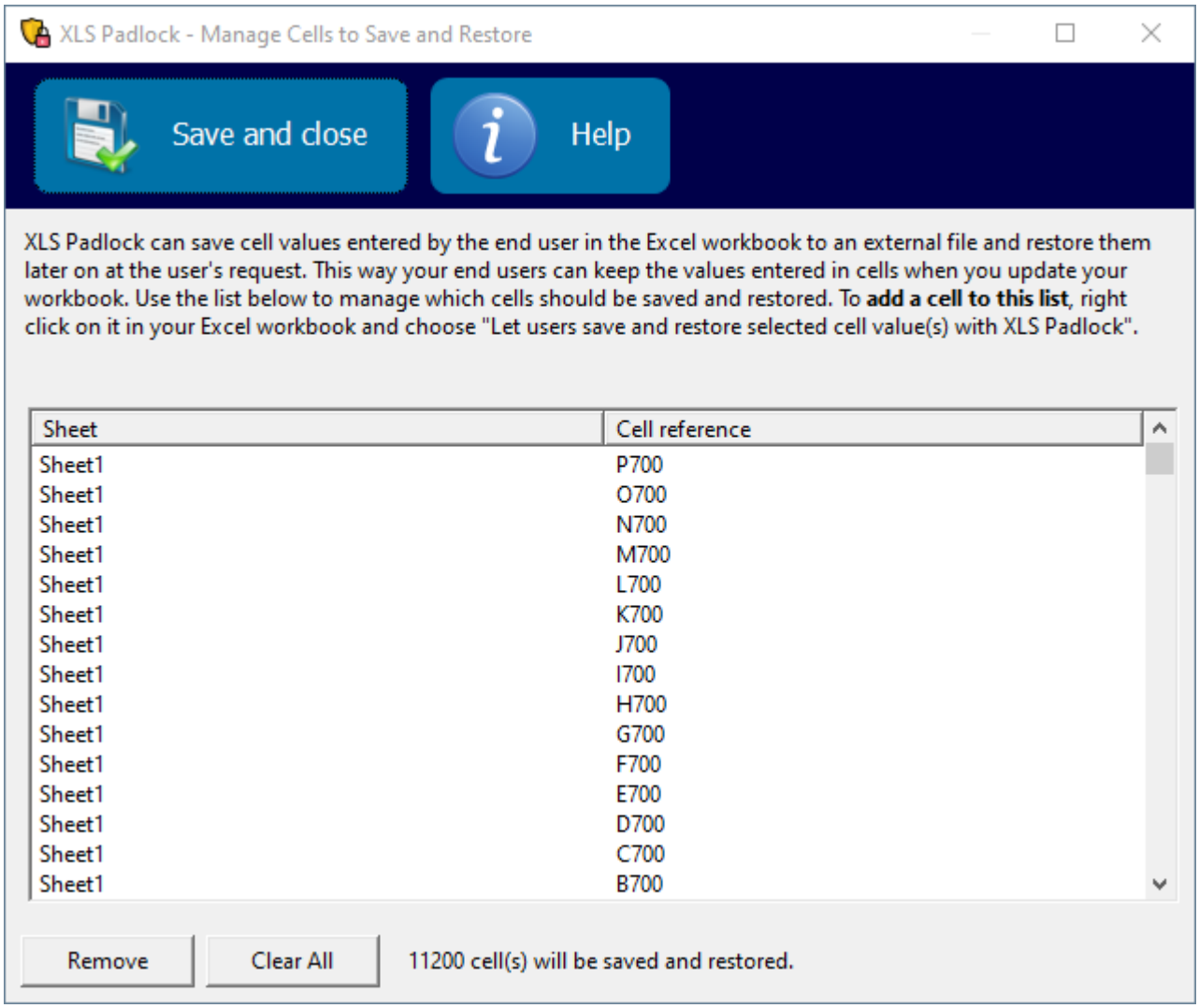
It is even possible for you to store custom values and restore them later with VBA code.

Overview of cells to be saved or restored

You can have an overview of all the saved cells by clicking "**Cell Save And Restore**" in XLS Padlock tab or menu:



Clicking "**Cell Save And Restore**" will open the list of the cells you have configured to be saved and restored In this window, you can remove some cells or clear the entire list if needed:



10.1.2. Programmatically restore/save custom values with VBA code

With the Save defined cell values only mode on, XLS Padlock lets you save and restore custom values in addition to pre-defined cell values.

To do this, XLS Padlock relies on two VBA events as well as two VBA API functions allowing you to read and write values when creating or reading customer's save files.

The two VBA events must be placed in a module of your Excel workbook:

```
Sub XLSPadlock_RestoreCustomValues()
 MsgBox ("Restoring values...")
End Sub

Sub XLSPadlock_SaveCustomValues()
 MsgBox ("Storing values")
```

```
End Sub
```

To write custom values, use the VBA API function WriteCustomCellValue.

```
Dim XLSPadlock1 As Object
On Error Resume Next
Set XLSPadlock1 = Application.COMAddIns("GXLS.GXLSPLock").Object
XLSPadlock1.WriteCustomCellValue("Unique ID", "Value you want to store")
```

To read custom values, use the VBA API function ReadCustomCellValue.

```
Dim XLSPadlock1 As Object
On Error Resume Next
Set XLSPadlock1 = Application.COMAddIns("GXLS.GXLSPLock").Object
Dim Val As String
Val = XLSPadlock1.ReadCustomCellValue("Unique ID", "Default value if no entry")
```

WriteCustomCellValue and ReadCustomCellValue will only work if invoked in the appropriate events `XLSPadlock_XYZCustomValues`.

If you want a dictionnary, it is also possible if you pass an empty unique ID to `ReadCustomCellValue`.

⚠ Be sure to add “Microsoft Scripting Runtime” in that case (using Tools->References from the VBE menu).

Here is the code to do so:

```
Sub XLSPadlock_RestoreCustomValues()

Dim XLSPadlock1 As Object
On Error Resume Next
Set XLSPadlock1 = Application.COMAddIns("GXLS.GXLSPLock").Object

Dim Dict As Object 'Scripting.Dictionary

Set Dict = XLSPadlock1.ReadCustomCellValue("", "")

For Each Key In Dict.Keys
        MsgBox "Key: " & Key & ", Value: " & Dict(Key)
Next Key

End Sub
```

Full example with comments

The entire following code must be placed into a module.

This sub restores the entire column A from a string with different values separated by the comma character.

```
Sub RestoreFromValue(value As String)
    Dim r As Range, i As Long, ar
    Set r = Worksheets(2).Range("A:A")
    r.ClearContents

    ar = Split(value, ",")
    For i = 1 To UBound(ar) + 1
     r.Cells(i).value = ar(i - 1)
    Next
End Sub
```

The following event will be invoked by the compiled workbook when customers load a save file:

```
Sub XLSPadlock_RestoreCustomValues()

Dim XLSPadlock1 As Object
On Error Resume Next
Set XLSPadlock1 = Application.COMAddIns("GXLS.GXLSPLock").Object

Dim Val As String
Val = XLSPadlock1.ReadCustomCellValue("MyEntireRow", "")

RestoreFromValue (Val)

End Sub
```

This function generates a comma-separated string from values of a given cell range.

```
Function csvRange(myRange As Range)
    Dim csvRangeOutput
    Dim entry As Variant
    For Each entry In myRange
        If Not IsEmpty(entry.value) Then
            csvRangeOutput = csvRangeOutput & entry.value & ","
        End If
    Next
    csvRange = Left(csvRangeOutput, Len(csvRangeOutput) - 1)
End Function
```

The following event will be invoked by the compiled workbook when customers save their changes to a file:

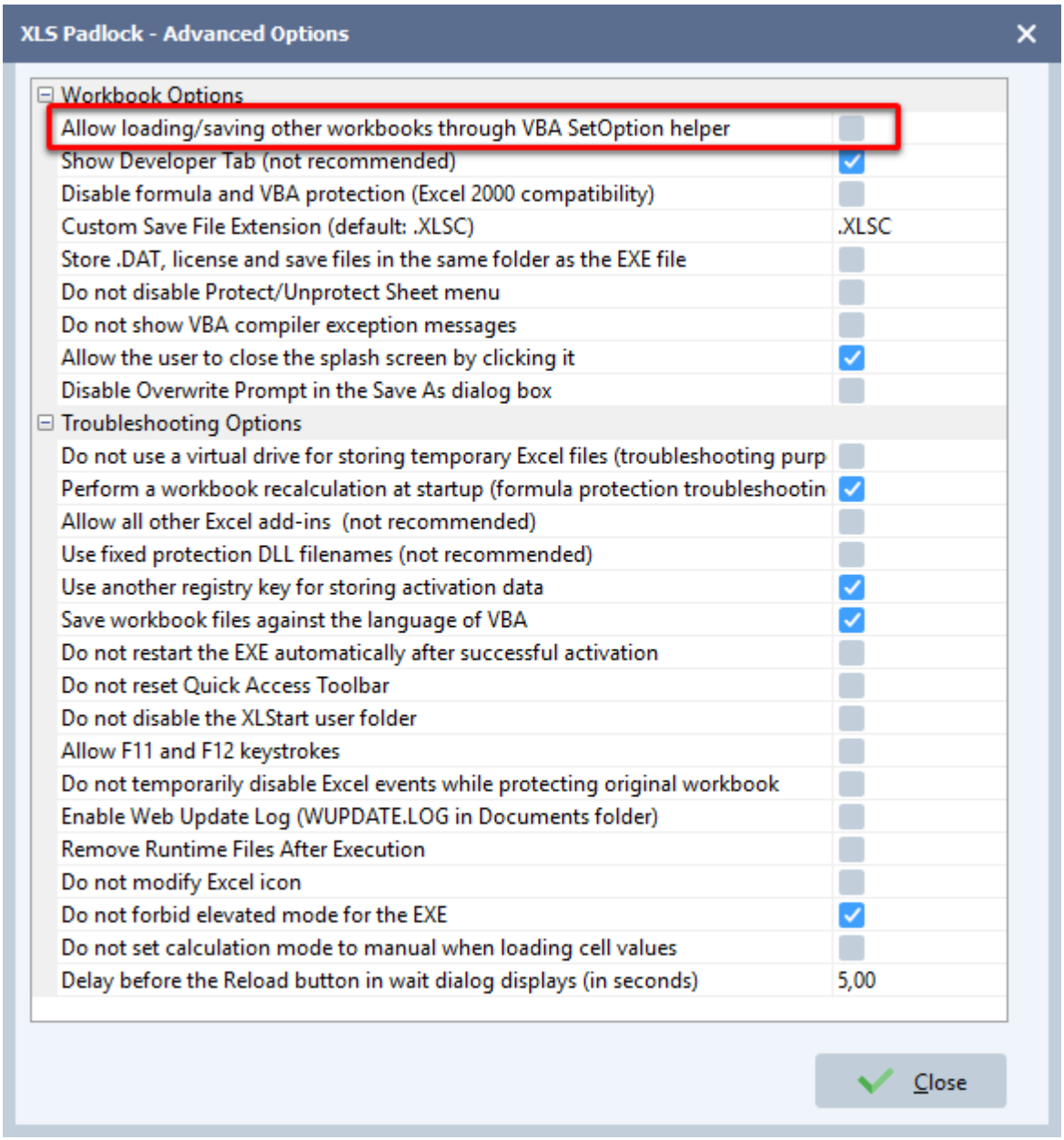
```
Sub XLSPadlock_SaveCustomValues()

Dim XLSPadlock1 As Object
On Error Resume Next
Set XLSPadlock1 = Application.COMAddIns("GXLS.GXLSPLock").Object

Dim rng As Range
Set rng = Worksheets(2).Range("A:A")

Dim myString As String
myString = csvRange(rng)

p = XLSPadlock1.WriteCustomCellValue("MyEntireRow", myString)
```

```
End Sub
```

10.2. Do not allow loading/saving other workbooks

This ultimate security feature will make Excel **unable to load and save any workbook file**, except of course the secured workbook. End users will not be able to load other existing workbooks in the same Excel instance. This also prevents possible hacks through VBA (for instance to extract values from workbook and save them as a new workbook).

This feature stops loading/saving workbooks with VBA code too. Since this could be inconvenient for VBA programmers, we provide a VBA code extension that temporarily switches the feature off: you must first enable it by going to Advanced Options and turn on "**Allow loading/saving other workbooks through VBA SetOption helper**". Then use the VBA code helper as shown in Loading/Saving workbooks through VBA SetOption helper.



See also Prevent common VBA and OLE hacks

10.3. Application GUID and Secret Key

The **Application GUID** is used by the application to store its settings on the end user's computer and to manage save files.

The **Application Secret Key** is used to create secure save files (modifications to your compiled workbook saved by customers). It ensures that secure workbook save files (.XLSC or .XLSCE extensions) made with your application can only be opened by **your** application and not in someone else's application.

If you change the Application GUID and/or Secret Key, you and your customers **will not be able to reopen previous saves**.

If you update your workbook, and if you want older versions to be ignored, generate a new Application GUID. Otherwise, you should never change it.

Do not not reveal your application GUID or Secret Key to anyone.

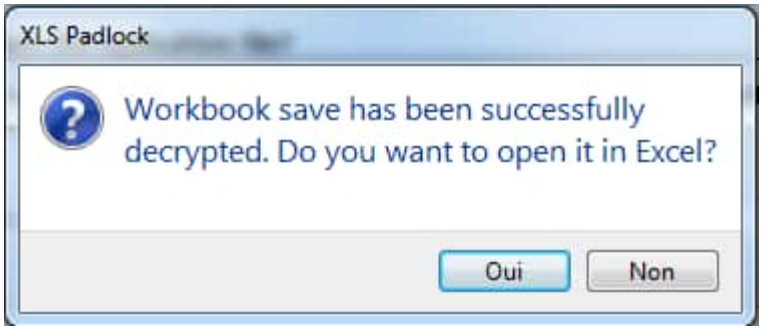
10.4. Opening a save yourself – decrypt saves

Save files made by your application are encrypted and cannot be opened directly in Excel without running the protected application. In fact, Excel itself is unable to recognize the .XLSC and .XLSCE proprietary formats.

If you are the original author of the secure workbook, **you can decrypt any save file created by your application**. This can be useful if you have customers who send you back modified versions of your secure workbook.

To open a save file (.XLSC or .XLSCE extension) created with your application, choose "Decrypt Save File" from the XLS Padlock ribbon:

If you select an XLSC file (full save mode), XLS Padlock then asks you which save file you want to decrypt. After that, you get a "successfully decrypted" message:



Formulas protected with XLS Padlock can't be recovered when decrypting a save file, and thus, the decrypted workbook may not completely work. This feature should be used only to recover data or changes made by users. If you need to use the completely working workbook, you should instead run the application and load the save yourself.

Same warning for the VBA project if you have locked it.

If you select an XLSCE file (cell values saving mode), XLS Padlock then asks you whether you want to modify the current workbook opened in Excel by loading cell values: "File ### will be modified or overwritten. Please confirm: do you want to overwrite it?" **Warning: this will definitively overwrite values in your existing source workbook!** If you confirm the operation, XLS Padlock will load cell values and finally display "Cell values were restored from the selected save file".

This decryption feature is useless **without the original XLS Padlock project file** (see how to save and restore settings). XLS Padlock does not let you decrypt saves made by other users of XLS Padlock. In fact, XLS Padlock uses your project settings to encrypt / decrypt save files. Without the project file, you can't decrypt saves. **That's also why you should never share your XLS Padlock project files (.XPLP files) with third-parties, and make regular backups.**

11. Workbook Access Control

XLS Padlock comes with a lot of security features. In particular, you can **restrict access to your Excel workbooks to authorized users only**. To do so, you can generate activation keys to be given to your customers, set up time-limited usages, and much more.

One of the objectives of XLS Padlock is to allow you to **sell licenses for your workbooks and thus earn money**, while giving you functionality to reduce workbook piracy, such as simple workbook sharing between third parties.

This topic gives an overview of security features dedicated to workbook access control:

- You want to control who may use your workbook → set up activation keys.
- You want your workbook to be accessible on one computer only, which makes the sharing of your workbook impossible → use hardware-locked activation keys.
- You want to remotely control the number of activations of your workbook and disable access in case of refunds → Use online activation, deactivation and validation.
- You want to create a Trial version of your workbook, for instance you want your workbook to expire at a given date → How to create trial workbooks
- You want to restrict the actions your customers can do: you can forbid printing, PDF or XPS export; disable right-click; hide toolbars and ribbons; disable Excel add-ins... Use "Only one instance of the workbook can be run at a time" to prevent your customers from running several times your application at the same time. You may also forbid several instances of Excel.
- You want your workbook to be accessible only when a dongle is inserted → Link the workbook application to a dongle or USB stick in the USB or Dongle Protection page.
- You want to sell licenses for your workbooks and earn money → set up your own online shop with WooCommerce and XLS Padlock thanks to the WooCommerce Integration Kit available. It is even possible to sell subscriptions and earn monthly incomes from your workbooks.
- You ant to earn money regularly by selling subscriptions for your workbooks → set up the FS subscription kit to sell subscriptions with FastSpring for your Excel workbooks.

And much more is possible thanks to XLS Padlock.

11.1. How to set up activation keys

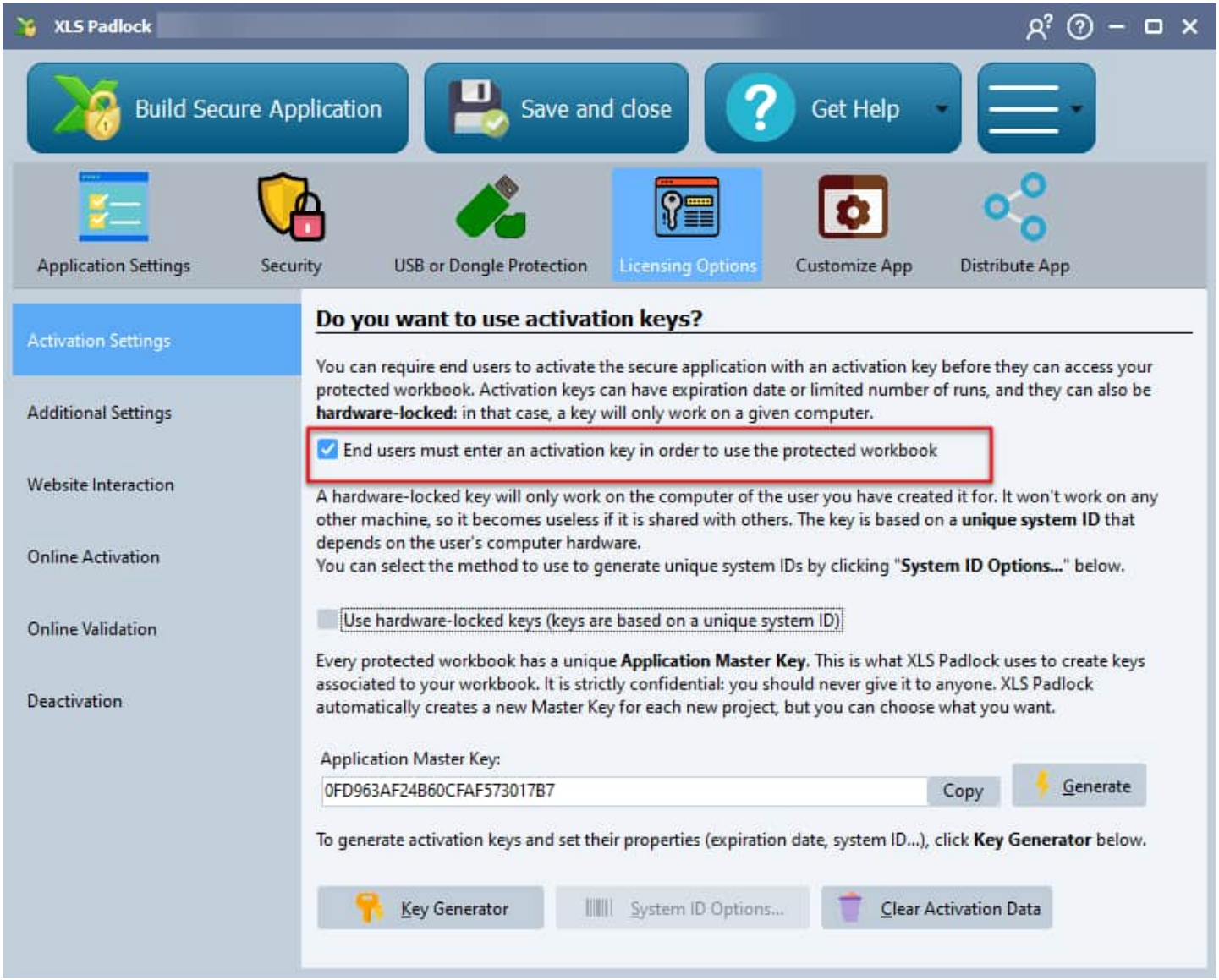
You want to give access to your Excel workbooks to some people only. That's possible with XLS Padlock's activation keys.

Activation keys offer a powerful licensing system for Excel workbooks: **you can require end users to unlock the secure application with an activation key before they can access your protected workbook.**

This topic shows you how to set up activation keys for your workbook application.

1. Enable activation keys

Tick "End users must enter an activation key in order to use the protected workbook"

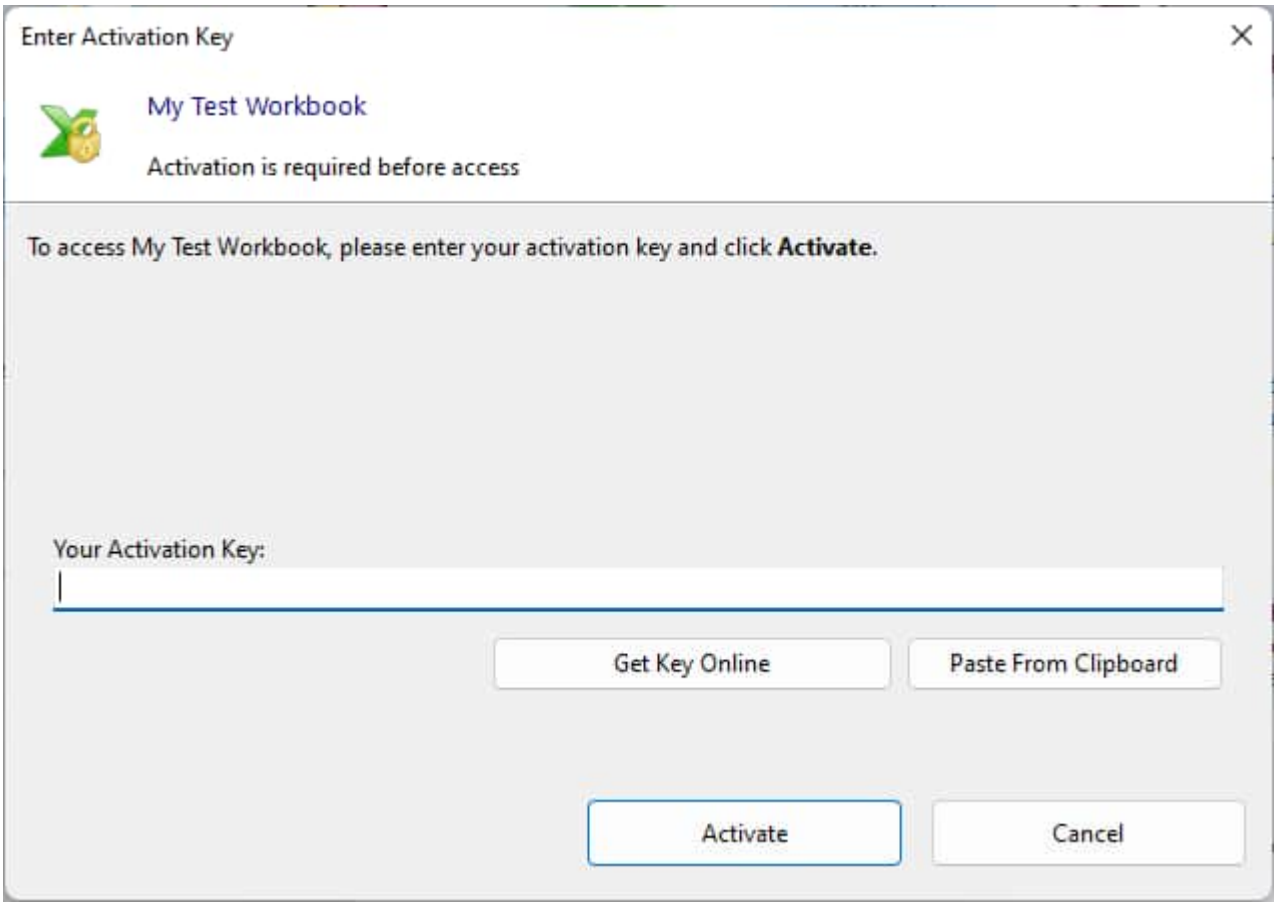


You can optionally define a new Application Master Key, but do not change it after you started deploying your application to others.

Rebuild your application. That's all.

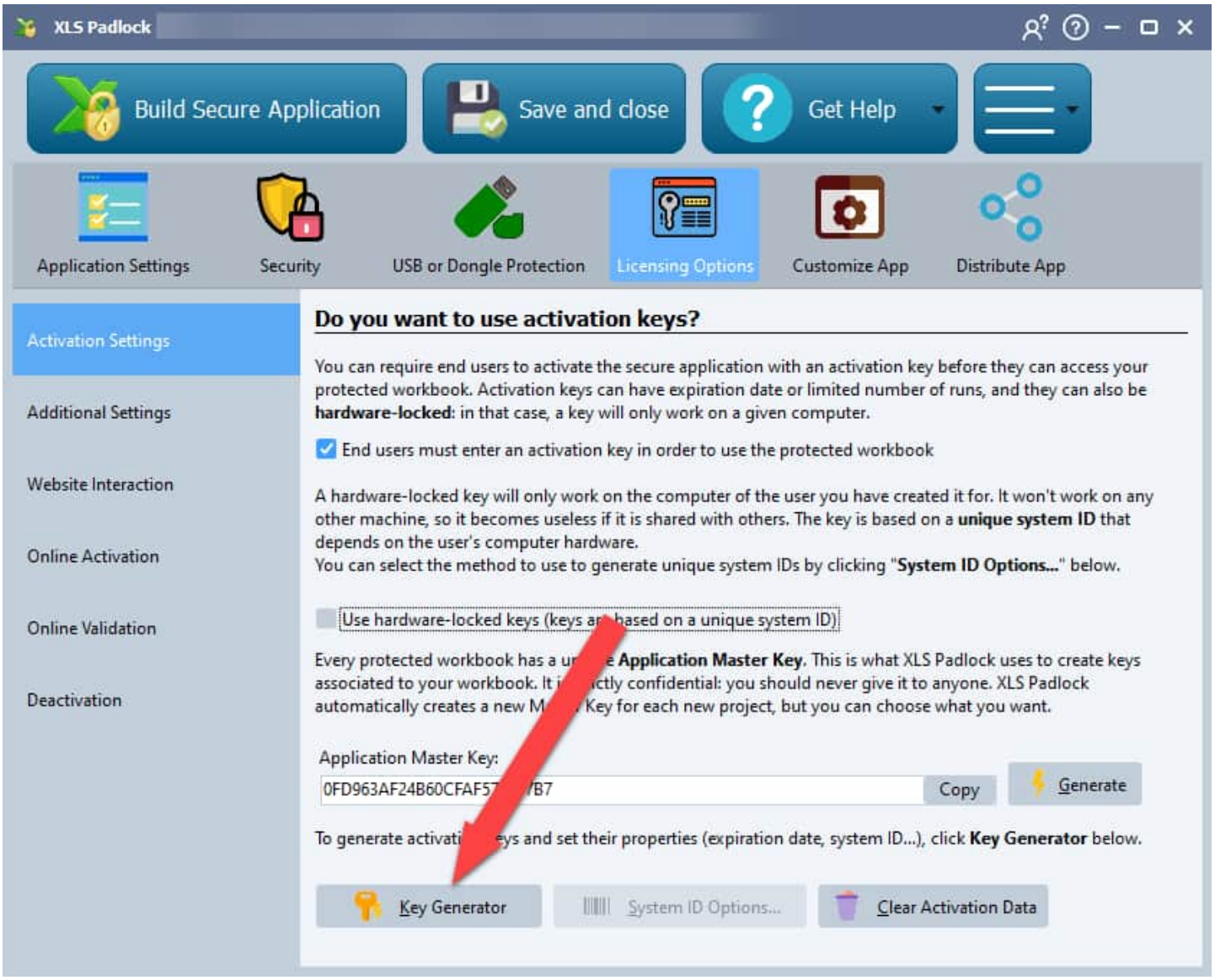
2. Test your application

Run the application. You are now prompted to enter an activation key before you may access the workbook:



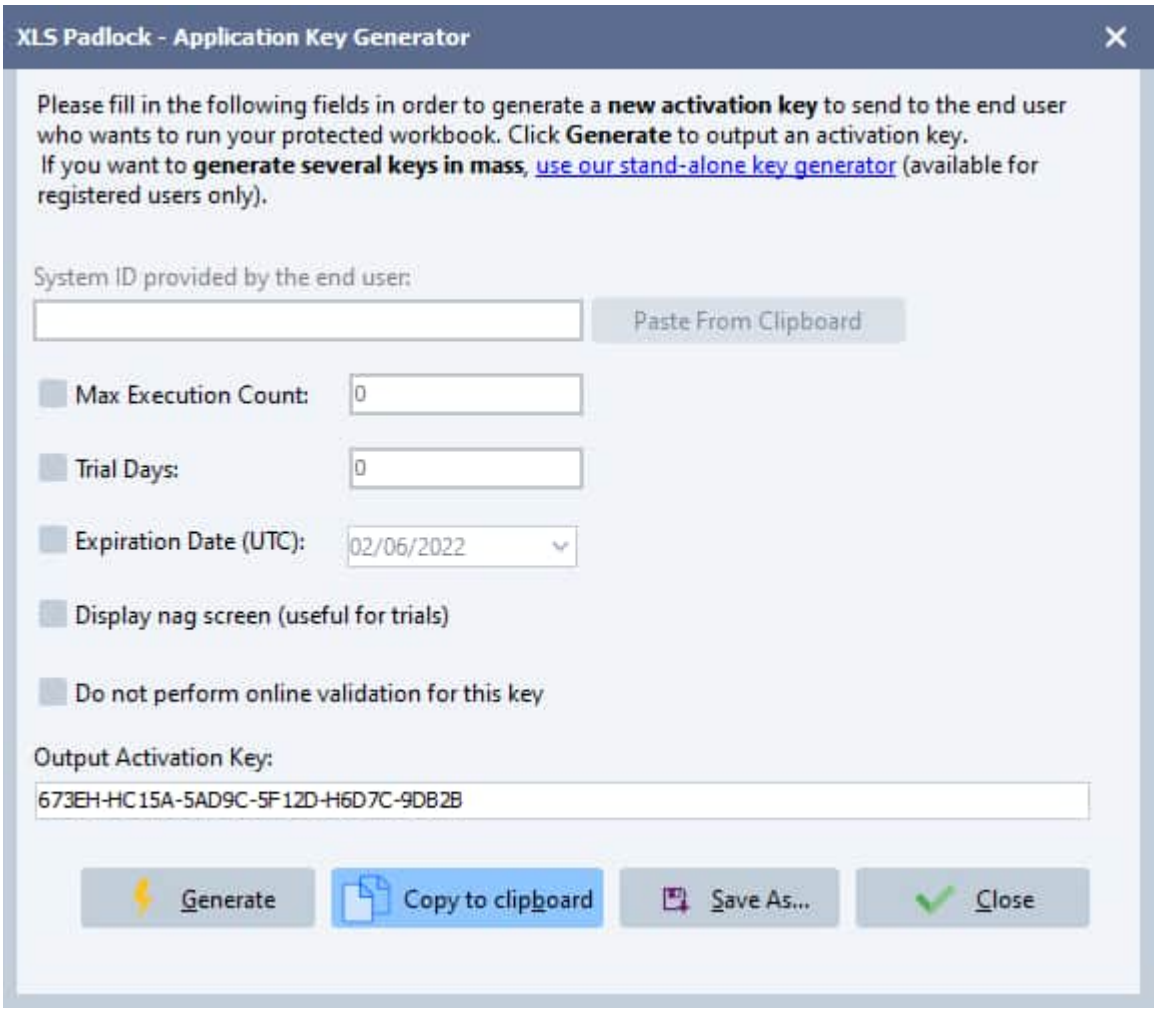
3. Generate activation keys

You can create activation keys thanks to the Key Generator in XLS Padlock, click the **Key Generator** button:

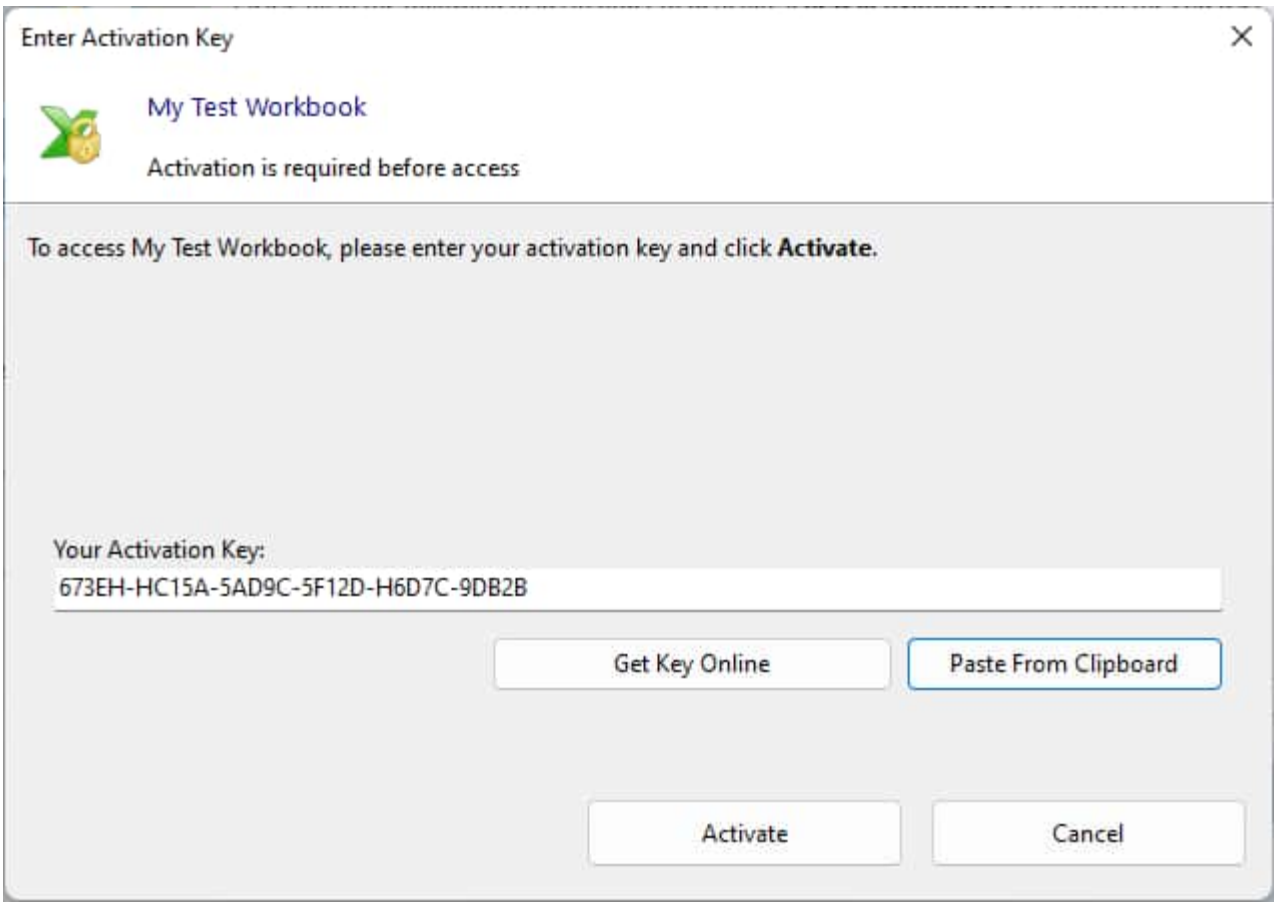


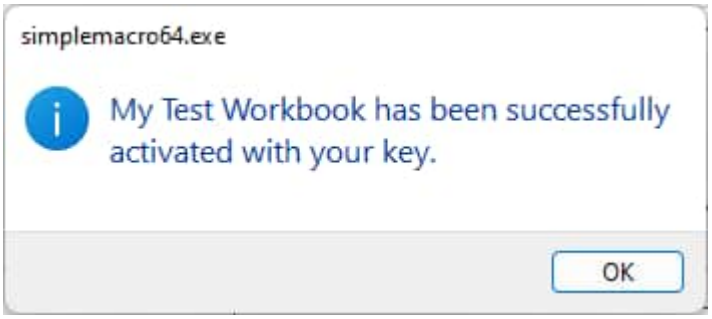
If you wish, you can **set restrictions on the key**: limit the number of times your Excel workbook can be run, make your key expire after a given number of days or after a given date.

Click the **Generate** button. A key is instantly created. You can copy it to clipboard to send to your customer, and/or save it as a .txt file:



Your customers run your Excel workbook again and will get the same window as before. They will just have to enter the key and click **Activate**:





The customer will be granted access and your workbook will open. The application will not ask for the key again, unless you set an expiration date for instance.

Generate activation keys automatically

A stand-alone key generator for workbook applications is also available for registered customers of XLS Padlock. With it, you can create activation keys for your protected workbooks without having to run XLS Padlock and Excel.

Finally, we also provide free online web applications for generating activation keys for your workbooks.

11.2. How to set up hardware-locked activation keys

In a previous topic, we saw how to set up activation keys. You want to give access to your Excel workbook to some customers only and **do not want them to share it with others**?

Using hardware-locked keys is one solution.

Watch our video tutorial about hardware-locking your Excel spreadsheets

See how this feature works online thanks to our Live Workbook Demonstration:
https://www.xlspadlock.com/excel-workbook-demo#01ce39abeb1

About hardware-locked keys

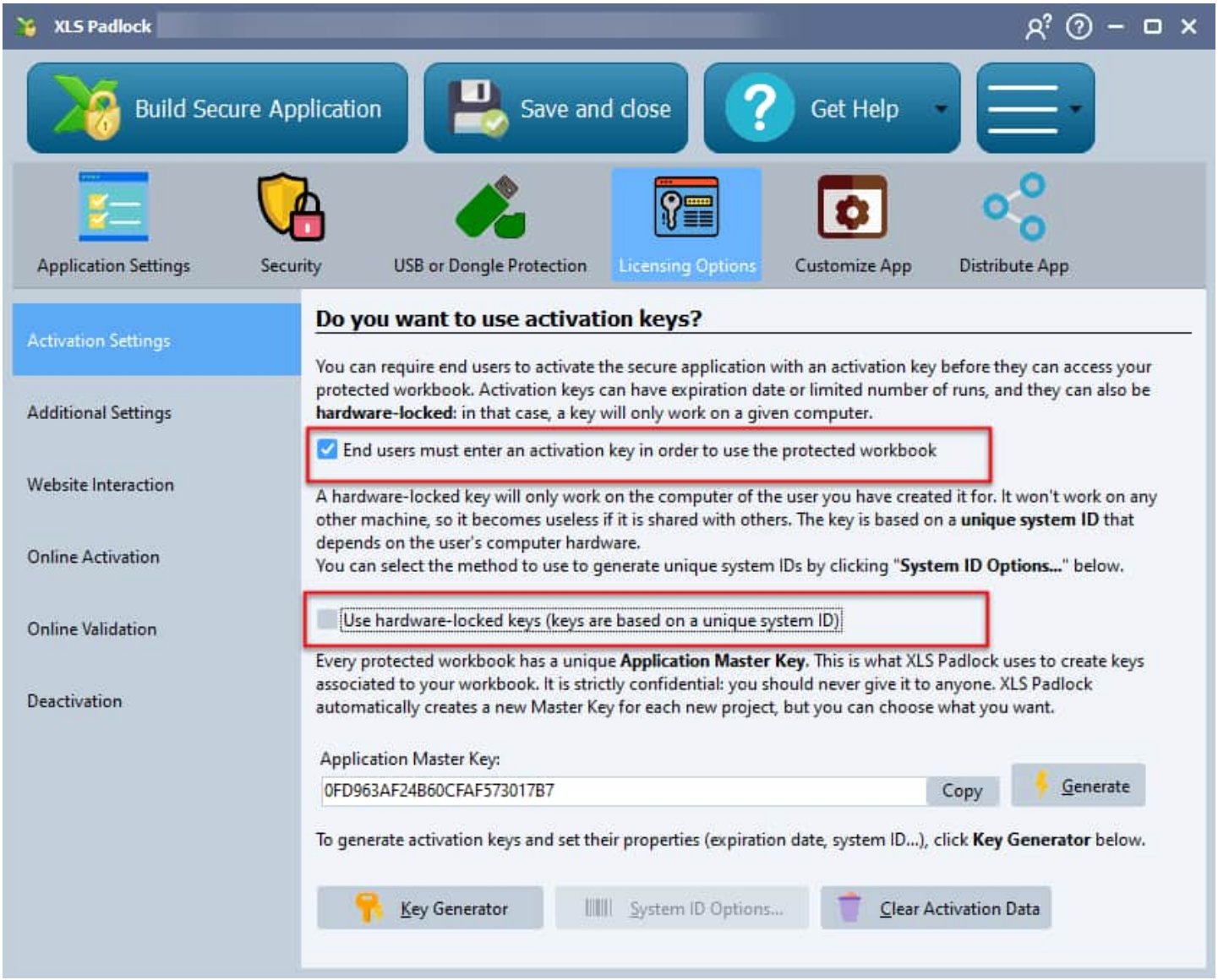
To open your Excel workbook, customers will have to enter an activation key. A hardware-locked key is a key delivered to customers **that will only work on their computer**. It won't work on any other computer than the expected one, so that it becomes useless if it is shared with others.

The key is based on a unique system ID that depends on the customer's computer hardware.

This topic shows you how to set up hardware-locked activation keys for your workbook application.

1. Enable activation keys

Tick "End users must enter an activation key in order to use the protected workbook" and “Use hardware-locked keys”:

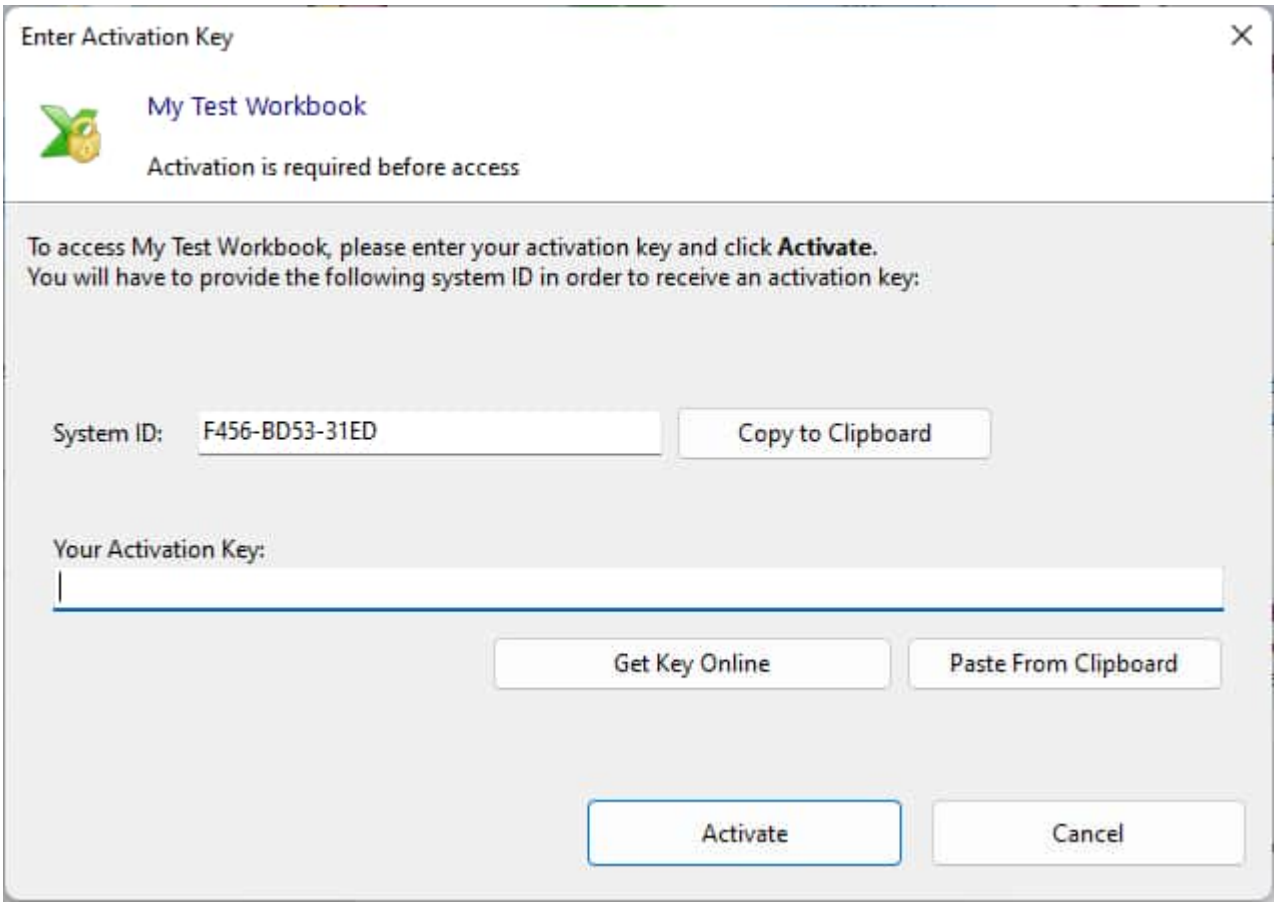


You can optionally choose the hardware components for the System ID by clicking "System ID Options", but do not change them after you started deploying your application to others.

Rebuild your application. That's all.

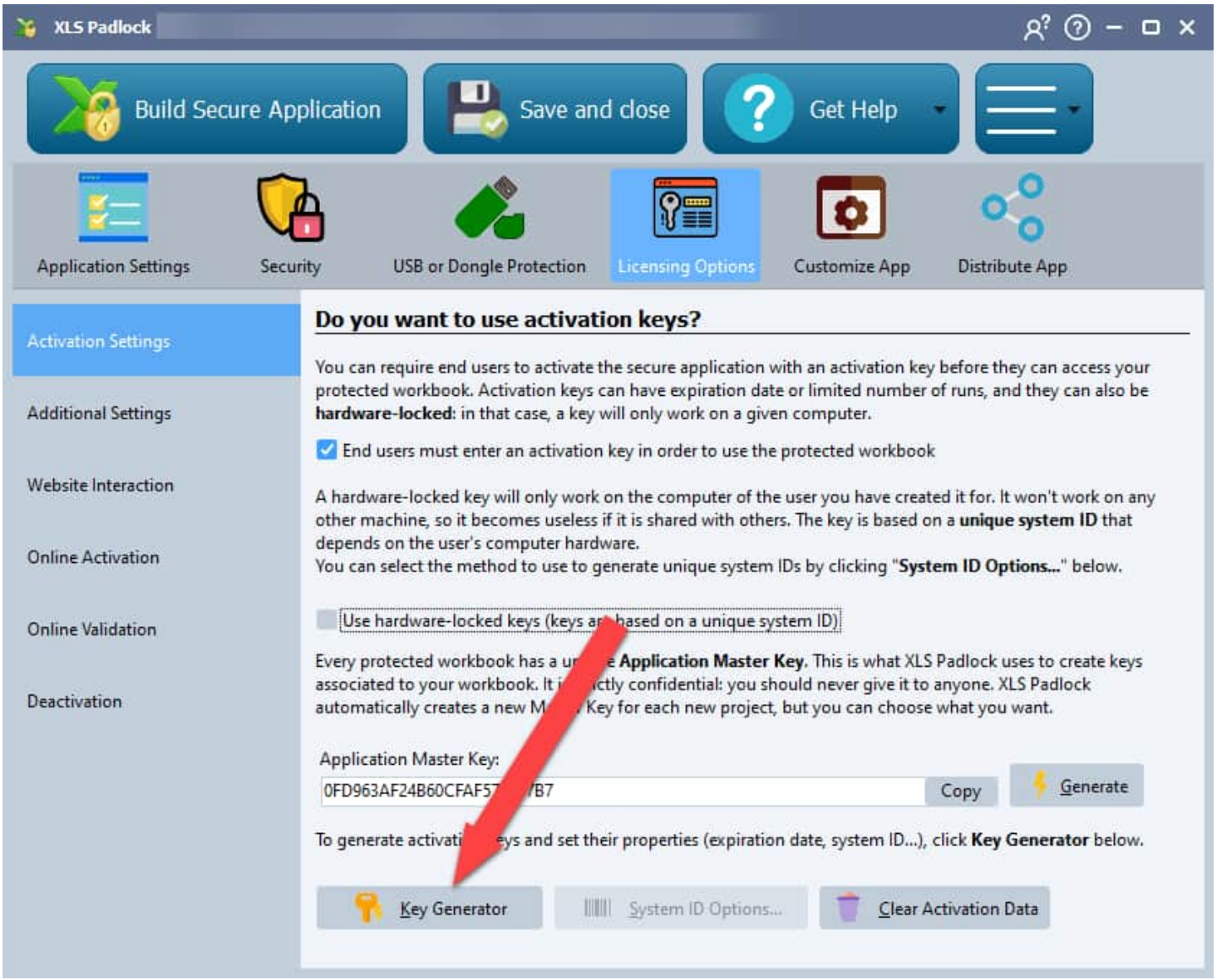
2. Test your application

Run the application. You are now prompted to enter an activation key before you may access the workbook. **Customers must copy their system ID and send it to you**. Alternatively, if you have a website, it is possible to configure a "Get Key Online" button so that the user can retrieve a key directly from your website.

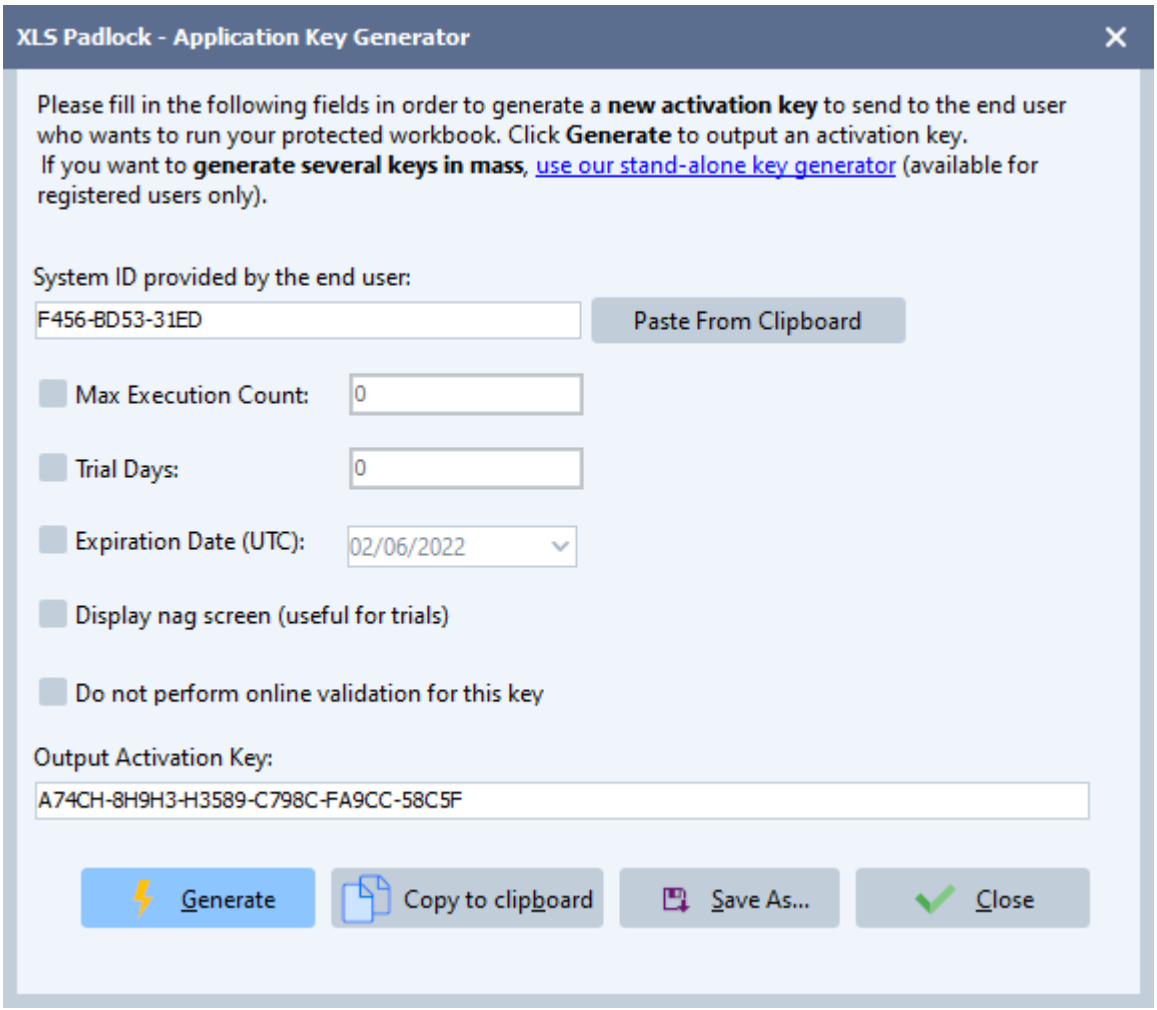


3. Generate activation keys

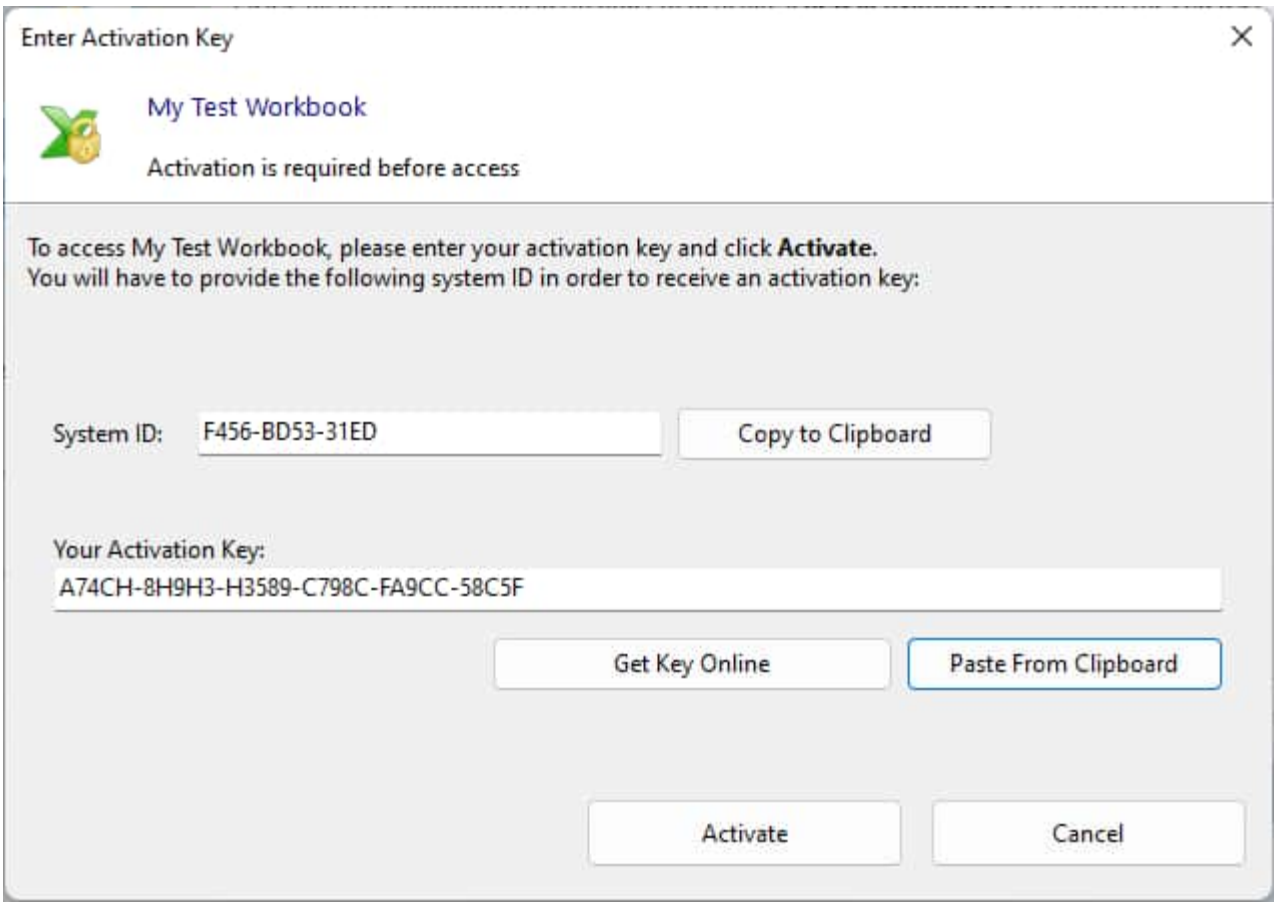
You can create activation keys thanks to the Key Generator in XLS Padlock, click the **Key Generator** button:

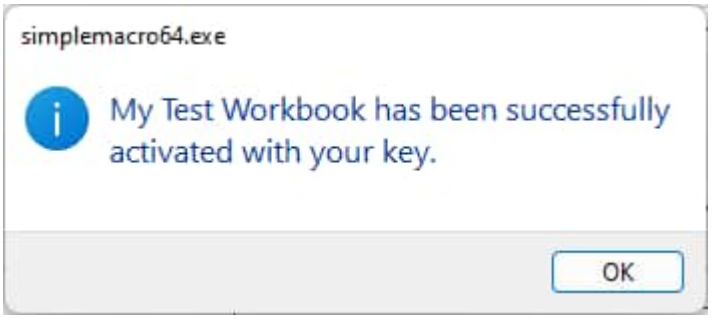


You just have to enter the system ID your customer sent to you. Then, click the **Generate** button. A key is instantly created. You can copy it to clipboard to send to your customer, and/or save it as a .txt file:



Your customers run your Excel workbook again and will get the same window as before. They will just have to enter the key (or use Paste From Clipboard) and click **Activate**:





The customer will be granted access and your workbook will open. The application will not ask for the key again, unless you set an expiration date for instance.

Moreover, **the key is associated to the system ID of the customer**, so that it will activate your Excel workbook only on the computer with this system ID. Sharing that activation key with another person is useless because the system ID will differ.

Generate activation keys automatically

The main drawback you will encounter is that you need to obtain your customers' system ID in order to generate their personalized key. If you have several hundred customers, this can cause problems. Fortunately, XLS Padlock provides you with a feature called online activation to automate the activation process and hardware-key distribution.

Finally, we also provide free online web applications for generating hardware-locked activation keys for your workbooks.

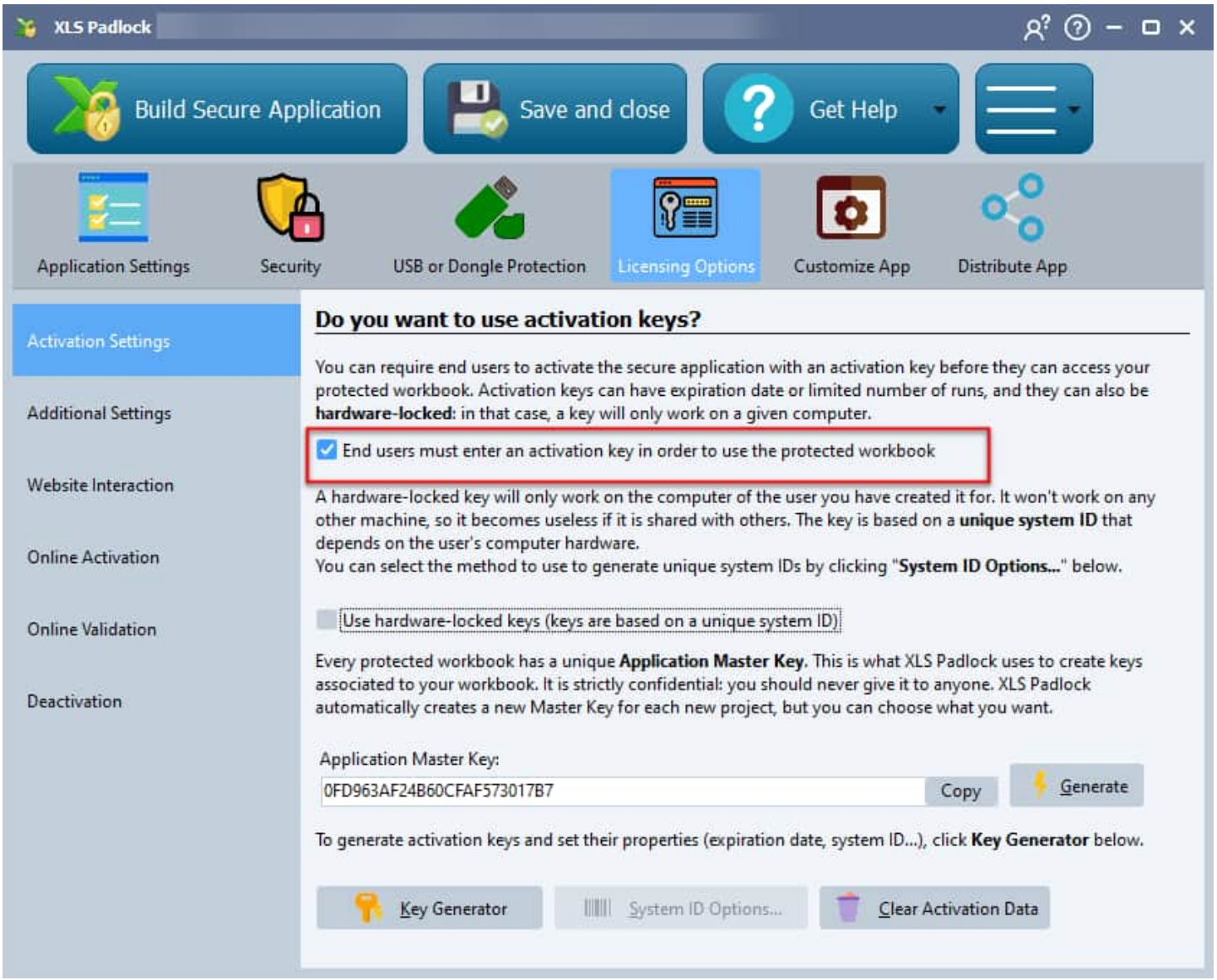
11.3. How to create trial workbooks

You want to give a preview of your Excel workbooks to your future customers and ask them to purchase a license after the trial period. That's possible with XLS Padlock's activation keys.

This topic shows you how to set up a trial period for your workbook application.

1. Enable activation keys

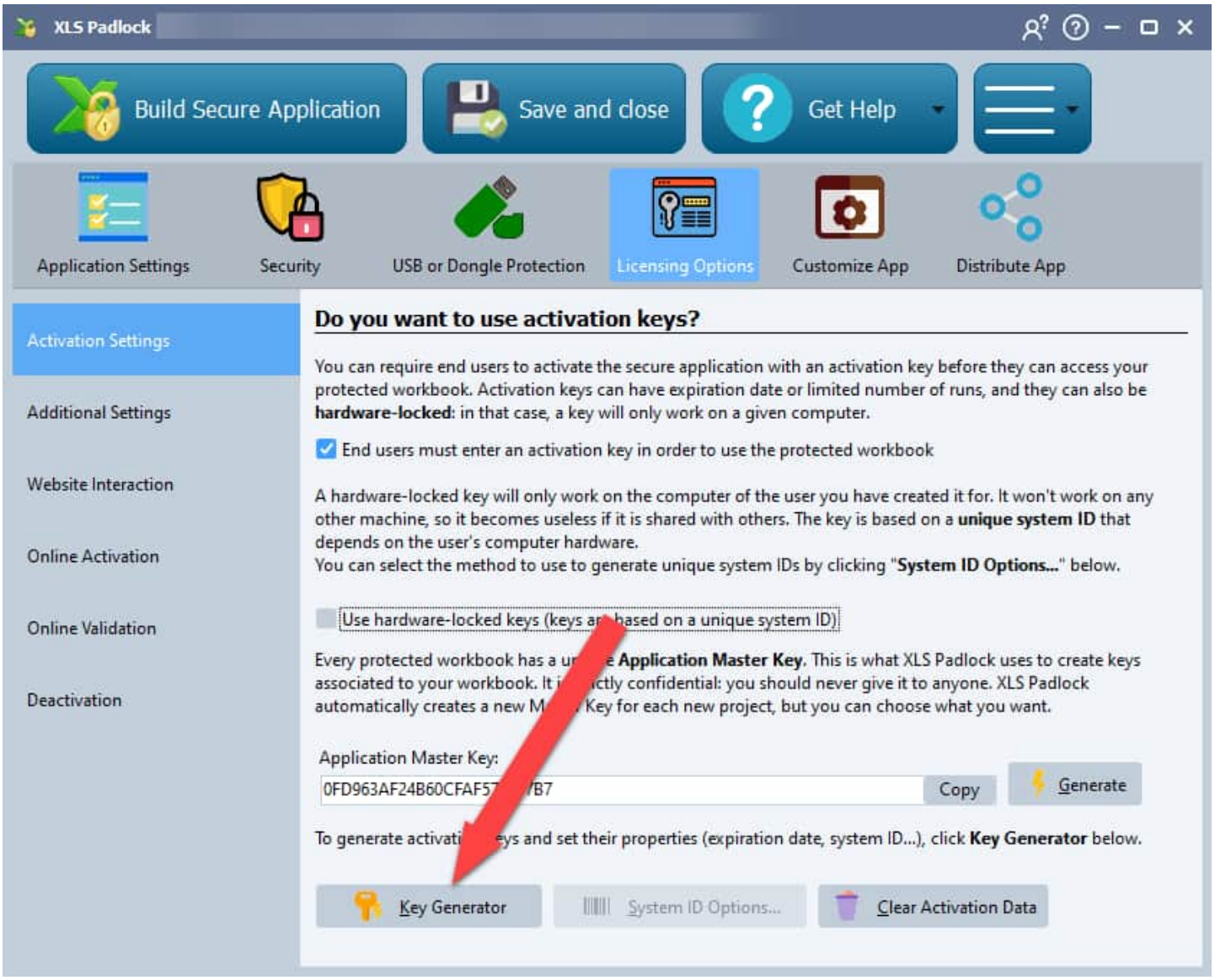
Tick "End users must enter an activation key in order to use the protected workbook"



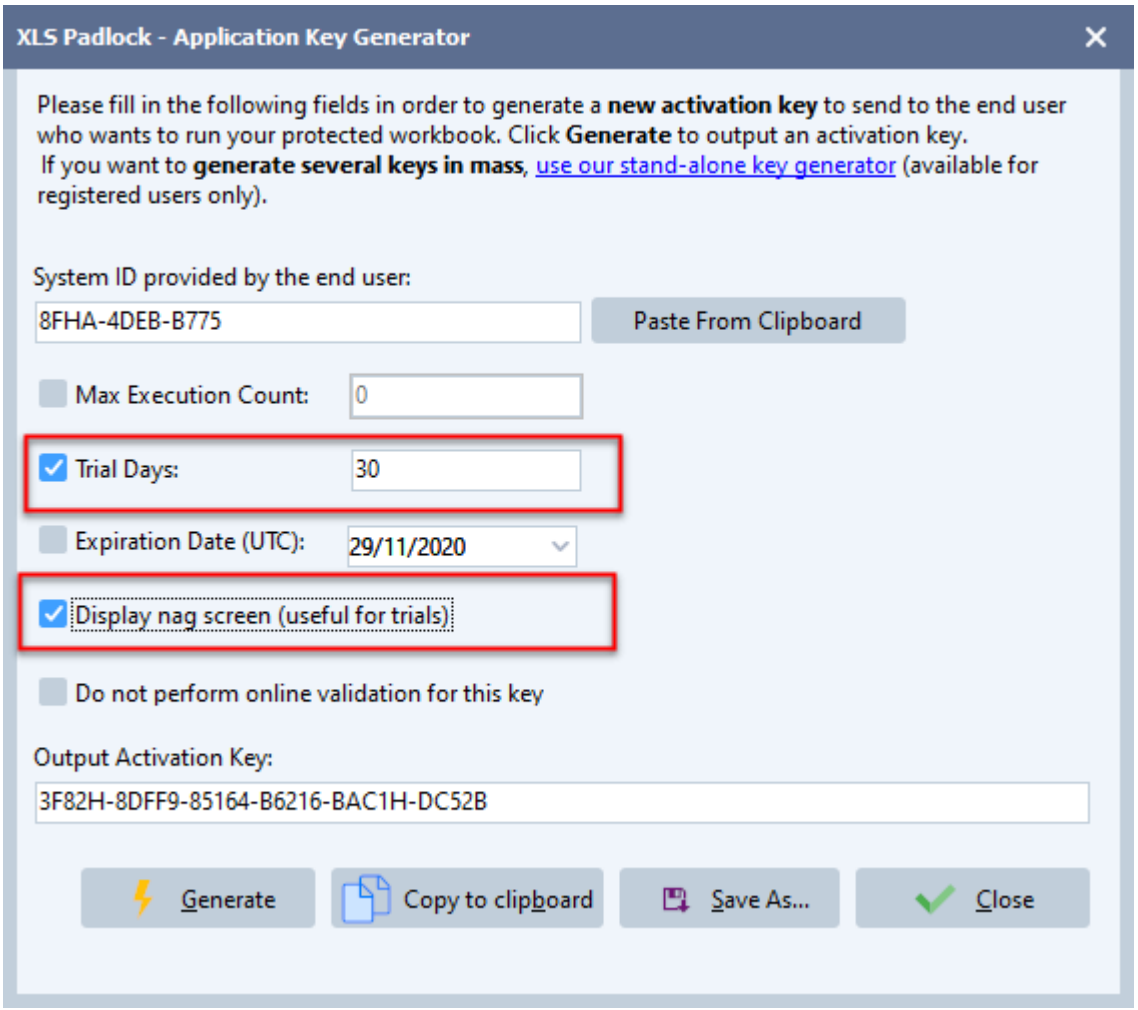
Rebuild your application. That's all.

2. Generate activation keys for trial version

You can create activation keys thanks to the Key Generator in XLS Padlock, click the **Key Generator** button:



In the key generator, **define a trial period**, for instance set "**Trial days**" to 15 to allow customers to test your workbook for 15 days. You can also choose an absolute expiration date, or a maximum number of runs:

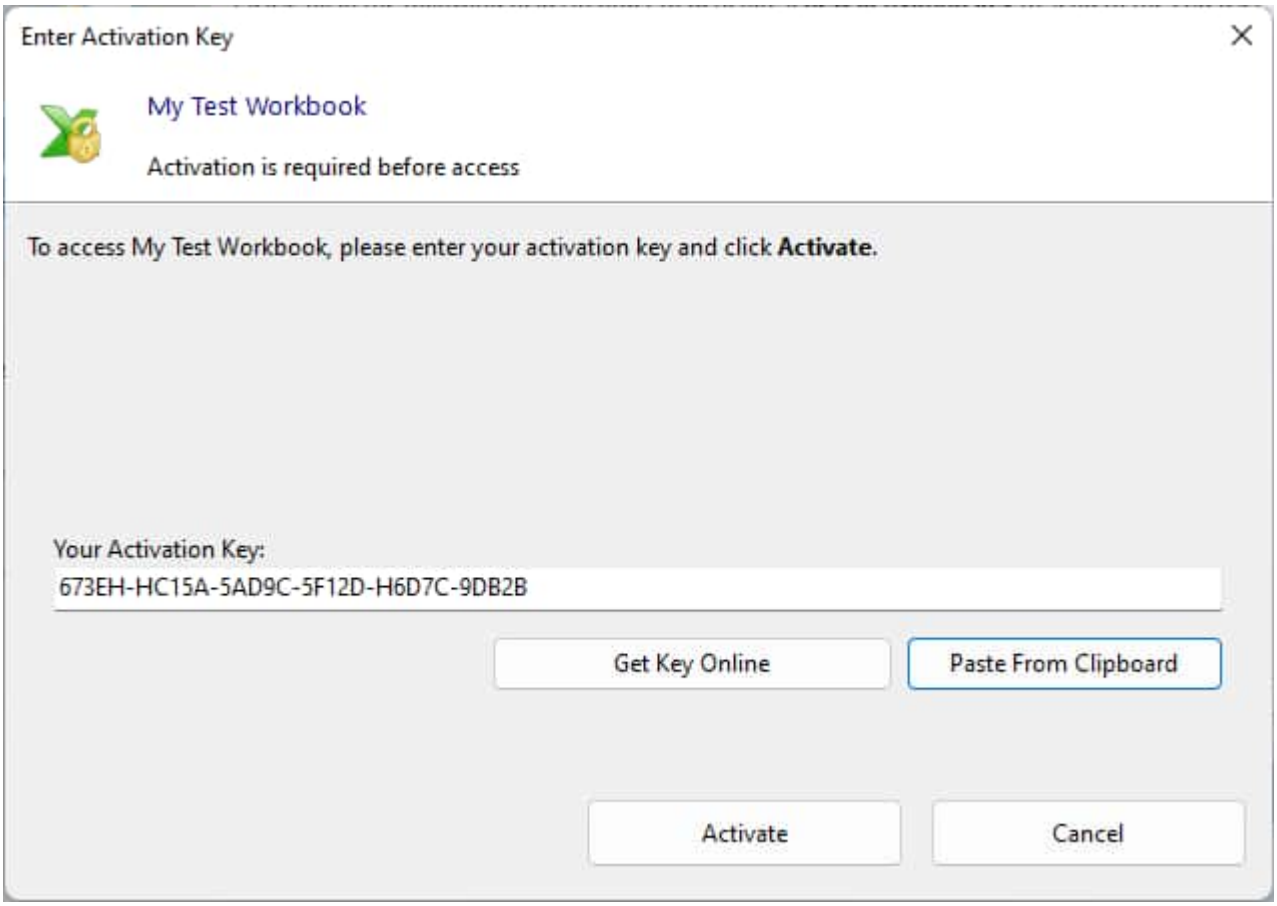


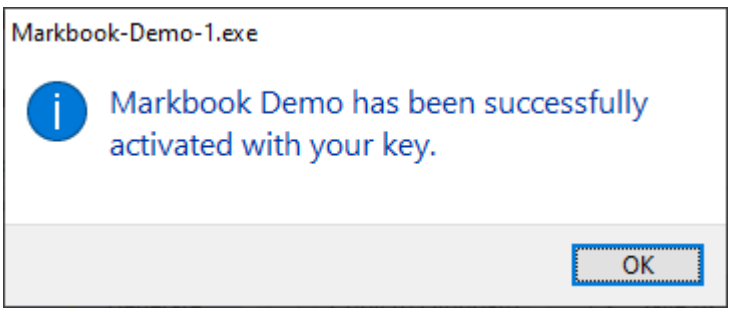
Finally, to remind your customers that they are working with a trial, tick "**Display nag screen**" so that a nag screen is shown each time the customer runs your workbook application.

Click the **Generate** button. A key is instantly created. You can copy it to clipboard to send to your customer, and/or save it as a .txt file.

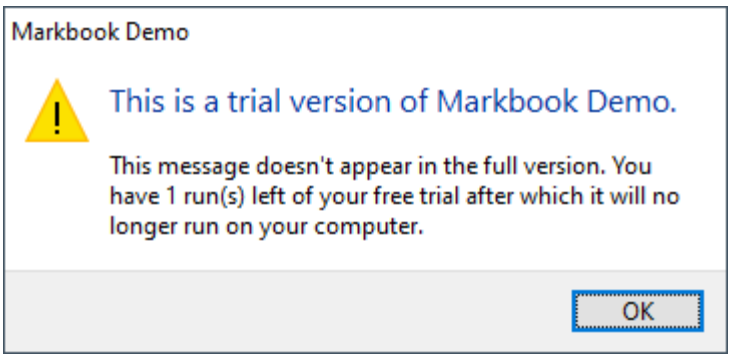
How the application works in trial mode

Your customers run your Excel workbook again and will get the same window as before. They will just have to enter the key and click **Activate**:

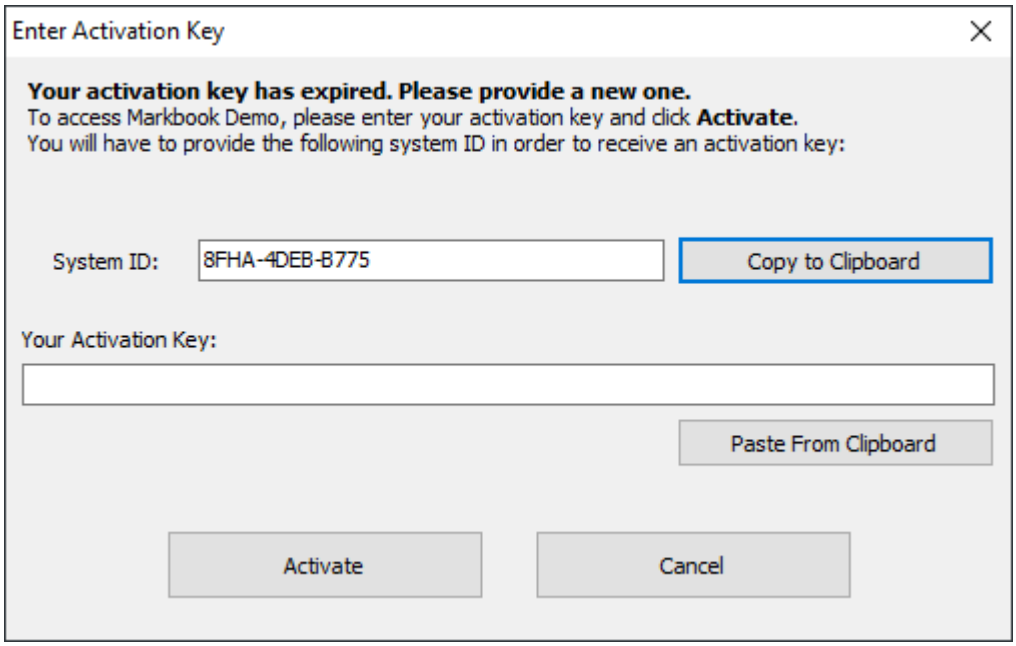




The customer will be granted access and your workbook will open, but **it will show the nag screen**:



When the trial period is over, the application will ask for a new activation key. Without it, they cannot access the workbook anymore.



How to check whether the workbook is in trial state?

Thanks to VBA, you can also check if the compiled workbook is in trial state.

11.4. How to close the workbook after a given amount of time?

> For instance, one can get a trial activation key for 1 day, open the exe file and leave it open forever. It will continue to work!
> Any Idea how to prevent it?

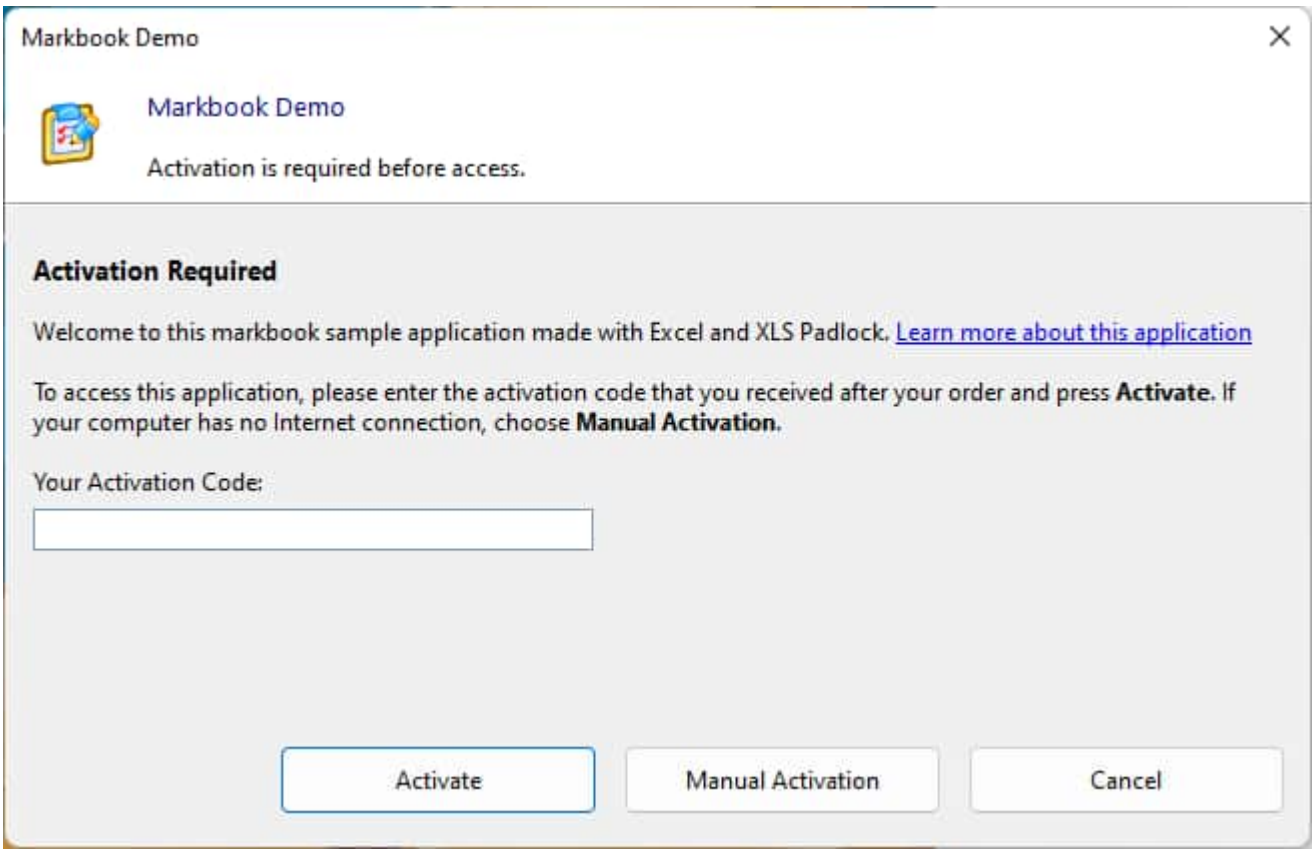
Solution: in VBA, you could set up a timer with Application.OnTime to close the workbook after a given amount of time. And forbid access to the VBA editor in XLS Padlock, so that customers cannot remove that code.

15.3.5. Online Activation

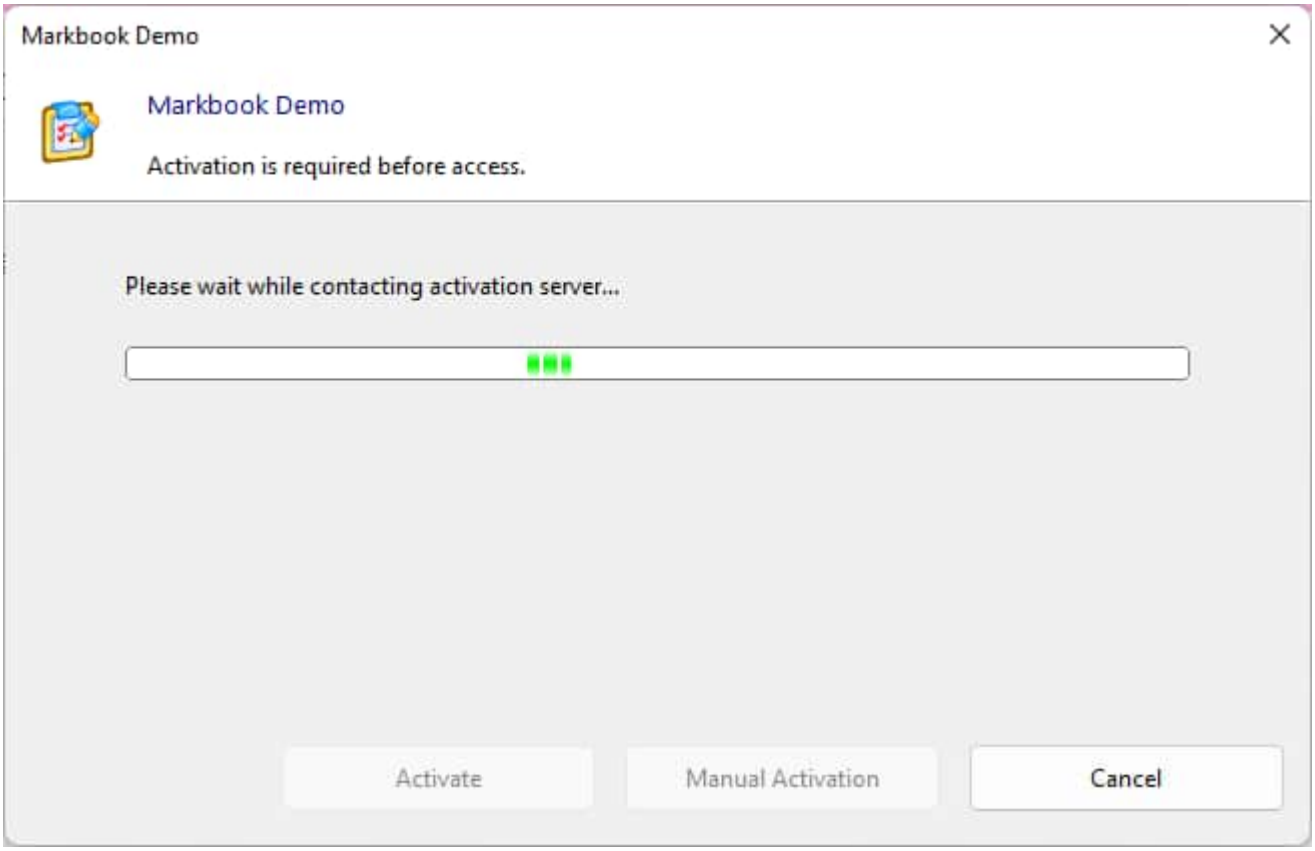
Online activation is a feature to automate the retrieval of activation keys thanks to the Internet: the application can communicate with your web server and download activation keys directly from the server instead of prompting end users.

You must have set up the **XLS Padlock Activation Kit** or the **XLS Padlock WooCommerce Integration Kit** or the FS subscription kit on your web server. Please go to https://www.xlspadlock.com/account/downloads for further information.

When the end user starts the application, a dialog box appears telling the user that an activation is required. This dialog box supersedes the "Enter Activation Key" dialog box. It can be customized at will: you can even add your own fields and have the application send provided data to your web server. In return, the web server checks whether the user has the right to access the workbook or not. If not, the workbook will not open. If yes, the web server sends the activation key back.



When the user clicks Activate, data is sent to the activation server:



After a successful activation, a confirmation message is shown, and the application is restarted. If an error occurs, the corresponding message box is shown, and end users can try to activate again.

- Learn about making regular money with online activation and subscriptions for Excel spreadsheets
- Show "Purchase Online" button on nag screen that opens the user's web browser to the following URL: Base Activation URL

15.3.7. Deactivation

Introduction to Deactivation

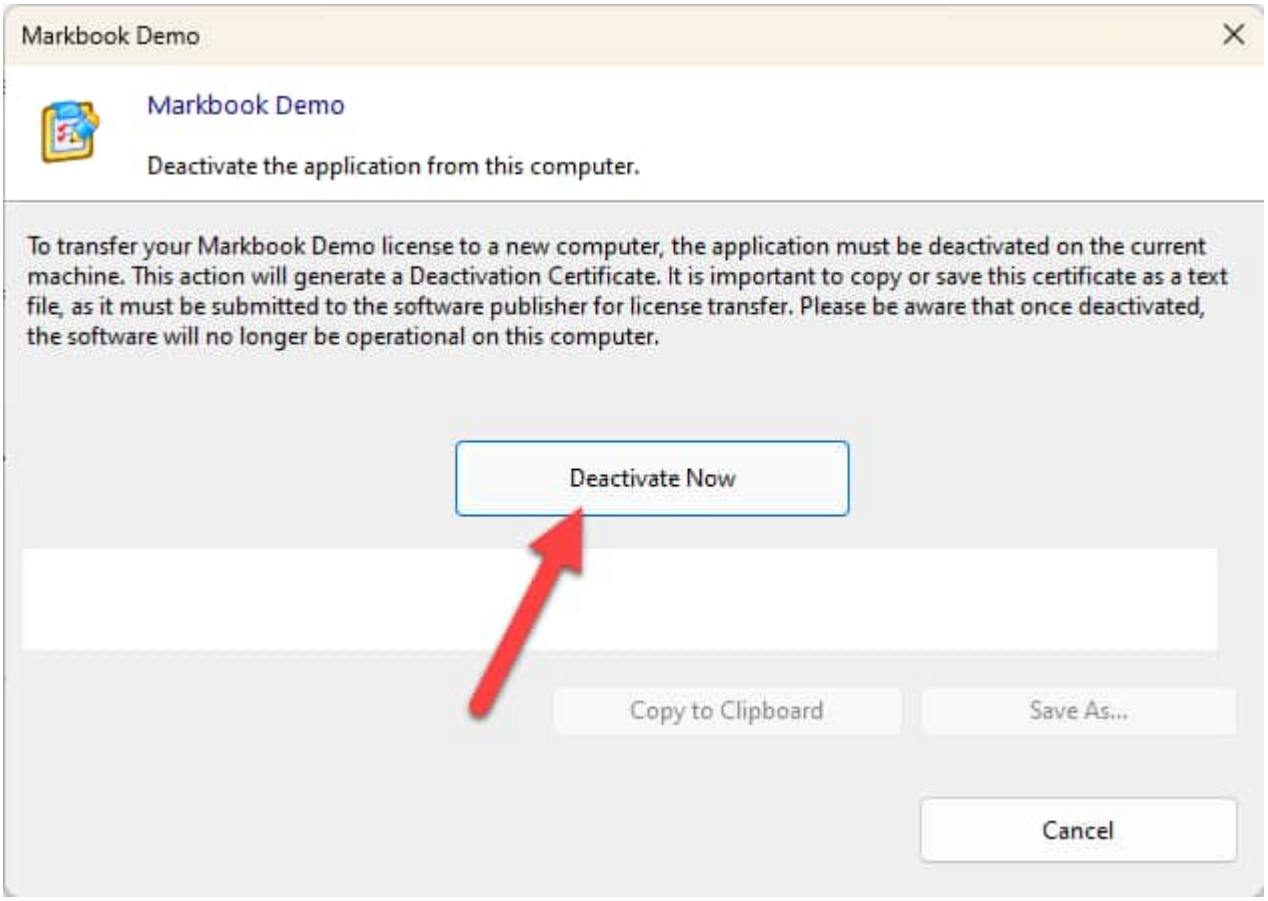
XLS Padlock provides a robust deactivation system, allowing end users to deregister their activated workbook application from a computer. This is especially useful if your customers need to transfer their secure application to a new machine or need to re-register due to system changes.

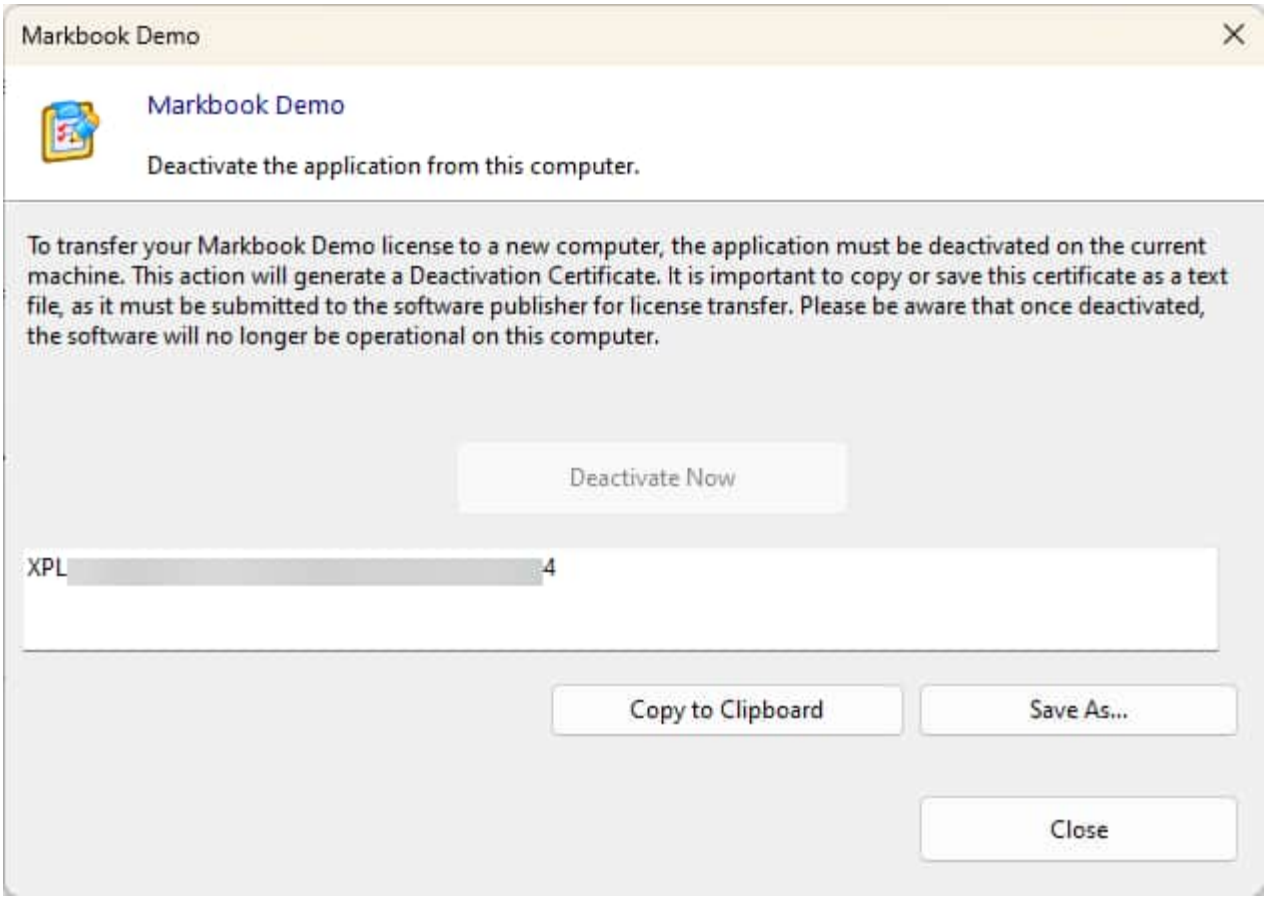
How Deactivation Works

Deactivation generates a certificate that the end user must forward to you, the vendor. Using XLS Padlock, you can verify the authenticity of this certificate thanks to the **Test Deactivation Certificate** feature. It is crucial to note that once deactivated, the current activation key will be rendered invalid by XLS Padlock, necessitating the issue of a new key to the user.

Manual Deactivation

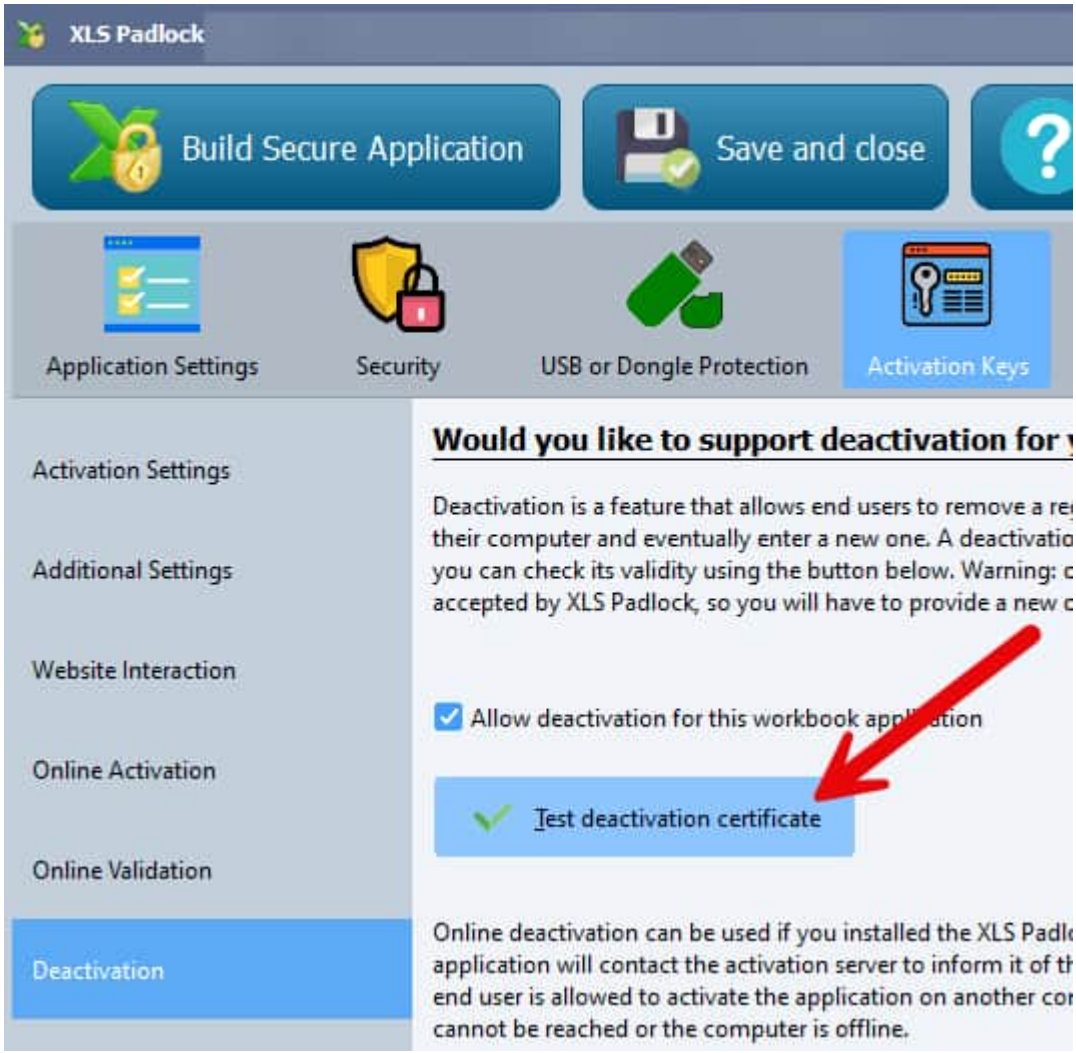
When a user decides to deactivate their workbook app, they must generate a deactivation certificate and send it to you:





Upon receipt, you can use the 'Test deactivation certificate' button in XLS Padlock to verify the certificate. If the

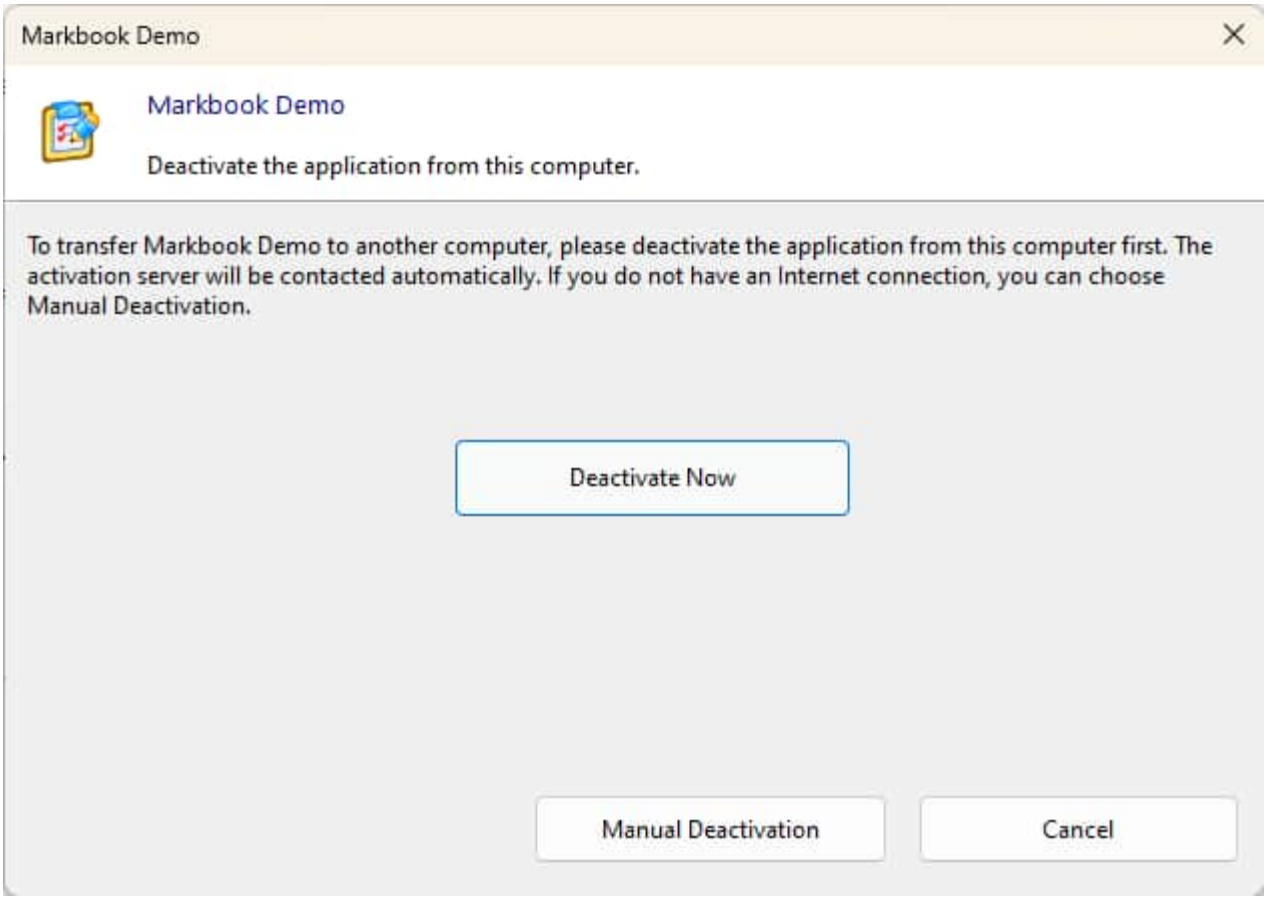
deactivation is successful, XLS Padlock will display the deactivation date and the unique identifier of the user's computer:



This information enables you to manage new activations for your client or handle deactivations manually.

Online Deactivation

For a more seamless experience, XLS Padlock supports online deactivation if you have the XLS Padlock Activation Kit installed on your web server. The application will communicate with the server to unregister the activation. If the server is unreachable or the computer is offline, a deactivation certificate will still be created.

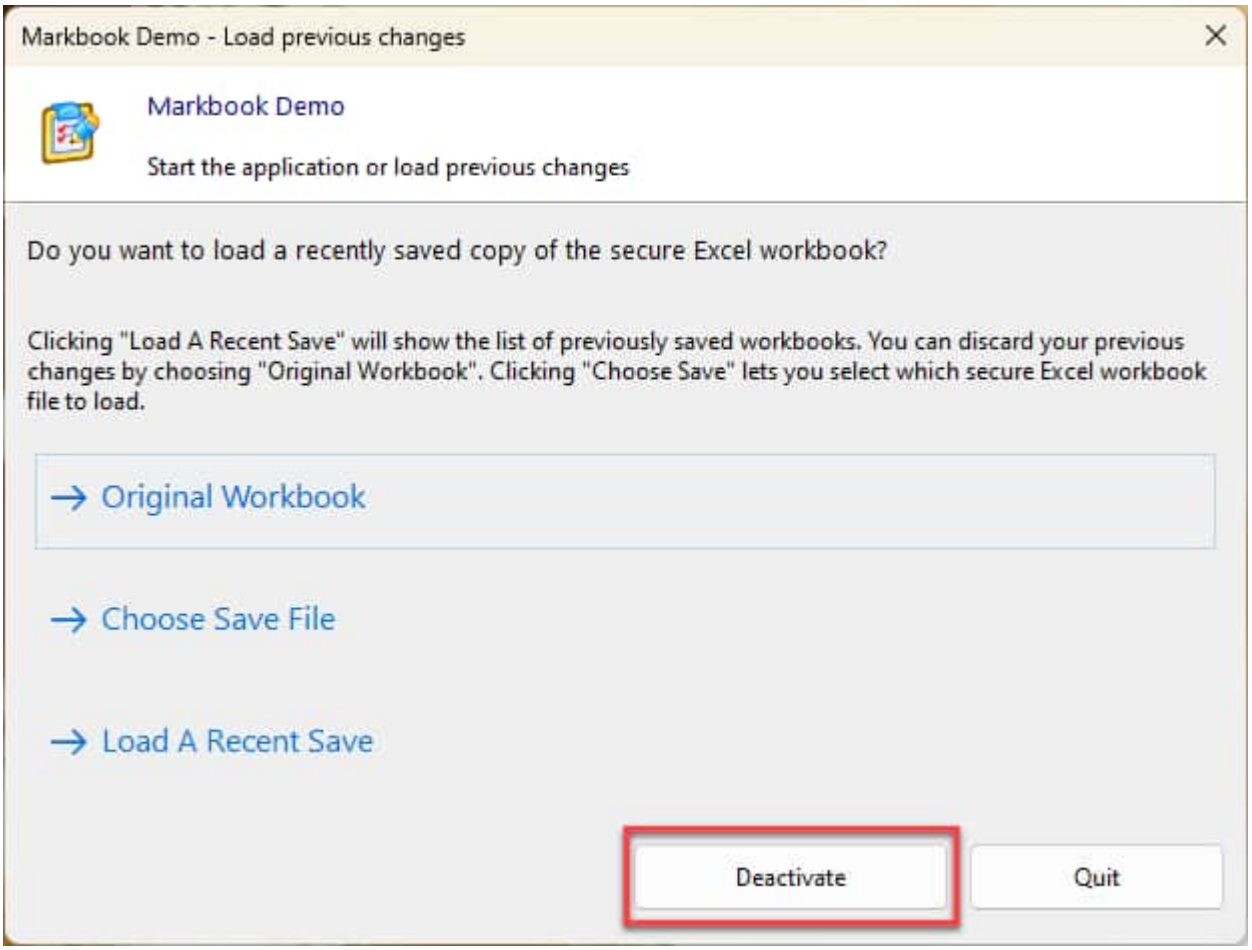


With online deactivation, everything is automated, this is easier for both your customers and you.

How to Start Deactivation

Initiating a deactivation in a secure application protected by XLS Padlock provides users with the necessary flexibility to manage their licenses. There are three main methods to start this process:

- The first option is through the integrated Welcome dialog of the application. Upon launching the secure application, the welcome dialog appears, presenting various options, including a "Deactivate" button. By clicking this button, users can begin the deactivation process in a user-friendly and guided manner:



- The second method involves the use of the command line. Advanced users may prefer this method for system automation or integration. To do this, the user needs to open the Windows command prompt, navigate to the directory where the secure application's executable file is located, and then enter the name of the executable followed by the `**-deact**` parameter. For instance, if the executable is named `Application.exe`, the complete command would be:

```
C:\Path\to\application\Application.exe -deact
```

  By pressing Enter, the parameter is passed to the executable file, triggering the deactivation process.

- The third method employs the VBA API provided by XLS Padlock. For more details on using VBA APIs for deactivation, please consult the article Start Deactivation of the Workbook Application With VBA.

Please be advised that once a key is deactivated, it will be locally blacklisted and can no longer be reused on the computer from which it was deactivated. Users must understand that deactivation is a final action for the specific machine, and the deactivated key will not be eligible for reactivation on the same system. If you wish to use the application on the same computer in the future, a new key will be required. Please also note that we speak about activation keys, not activation tokens (as defined in the WooCommerce Integration Kit).

Conclusion

Deactivation is a powerful feature that adds flexibility and control for both the vendor and the end user. By understanding and effectively managing deactivation, you can provide better service to your customers and

streamline the activation process.

- Allow deactivation for this workbook application

12. Workbook Updates

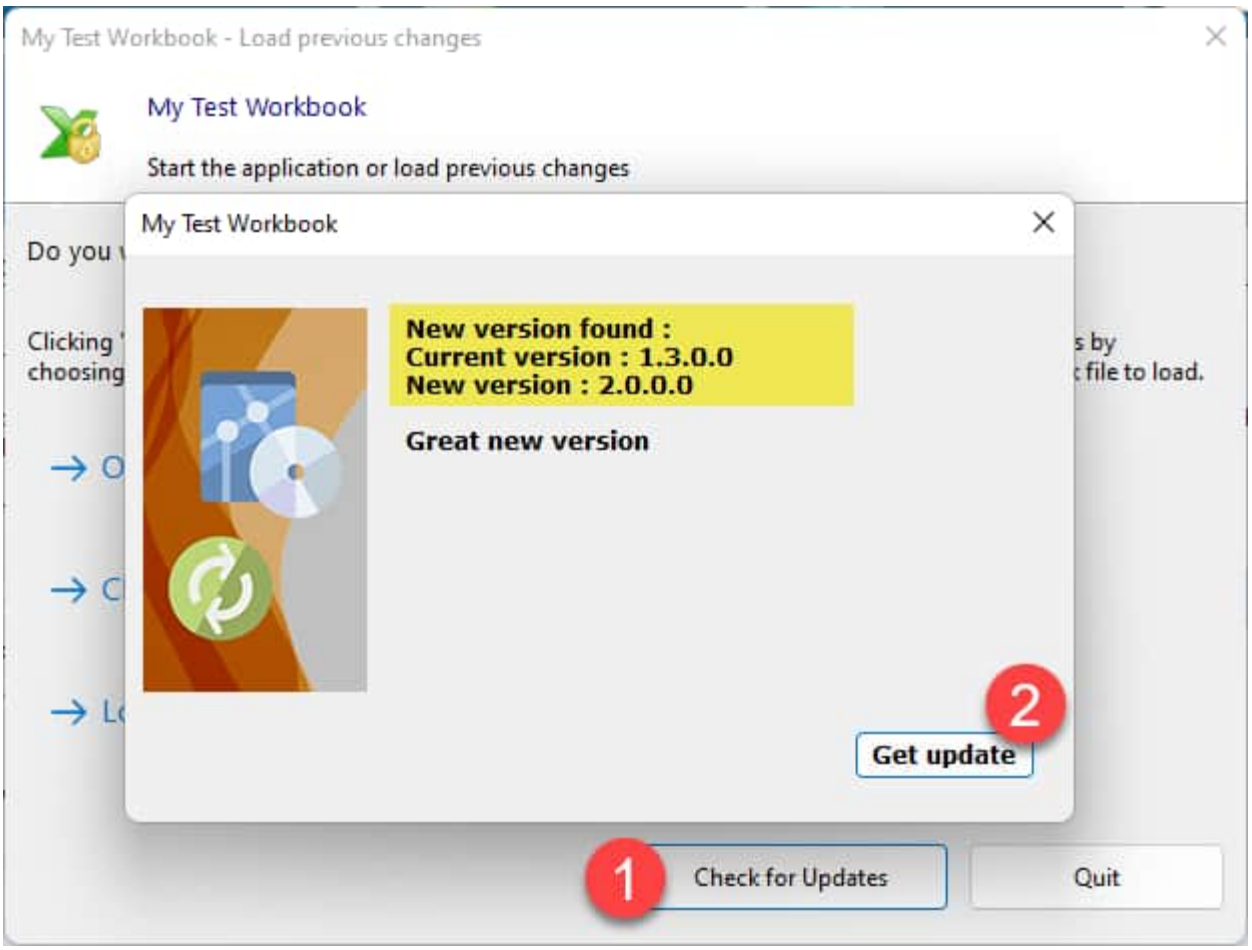
When you distribute an Excel workbook to other users, it is quite expected that you will have to make updates and provide them to your users.
XLS Padlock provides you with several tools to facilitate the distribution of updates to your compiled workbooks, let's find out which ones.

Before you start, you should already be aware of the different save modes offered by XLS Padlock.

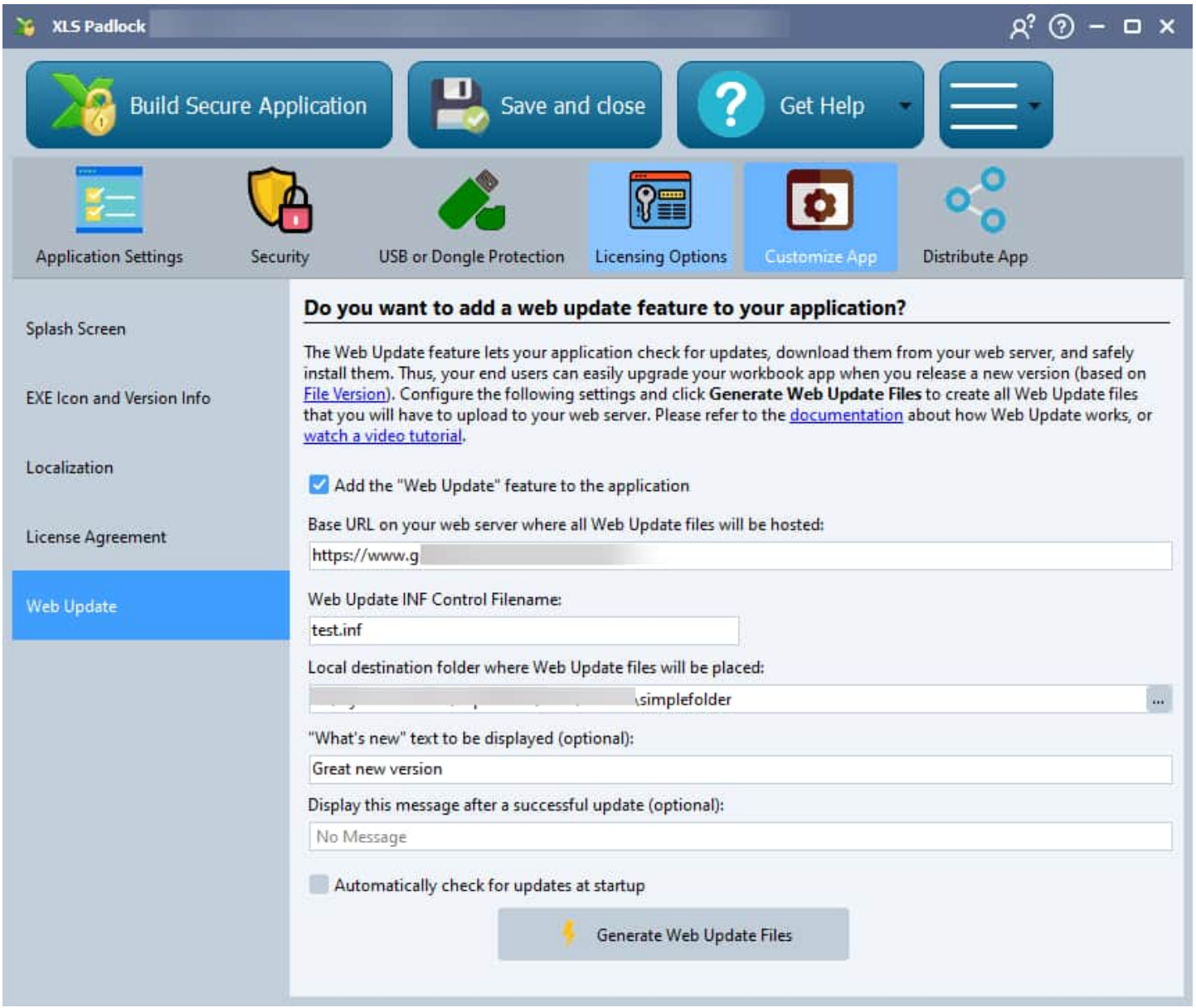
If your workbook is going to be modified often and your users only have to modify a few cell values in it when working with it, the save mode Save defined cell values only (.XLSCE file) is the most appropriate.
In fact, this allows you to update the main workbook file without the users having to re-enter all their data, because XLS Padlock with this save mode only keeps the cell values that you have designated and restores them later. This way you can modify all the logic of your workbook and your users keep the values of their cell data.

If you update your source workbook, the easiest way consists in recompiling it as an EXE file and deploying this new EXE to your customers (generally, they just download it again).

As it can be tedious for end users to redownload the EXE file and put it back in the right folder, XLS Padlock provides the possibility of an automatic update via the Web. Moreover, a regular control of possible update and notifications to end users can be set up:



Everything can be configured quickly and directly in XLS Padlock:



To set up your own update system, you must have a web hosting space available on the Internet (e.g. your own web hosting account or a dedicated server) where you can place files for direct download.
Then you just have to follow the instructions available on the how to set up automatic web updates page.

Watch a video tutorial about how to set up automated web updates for Excel workbooks

12.1. How to set up automatic web updates

XLS Padlock allows you to add a Web Update feature to your Excel workbook compiled apps: when you release

a new version, the Web Update feature can automatically download the updated EXE file from your web hosting provider or server, and safely install it. This allows end users to always stay with the latest version and easily upgrade your workbook app when you release a new version.

No third-party software is required. However, **you must own or rent a web space to host web update files generated by XLS Padlock.**

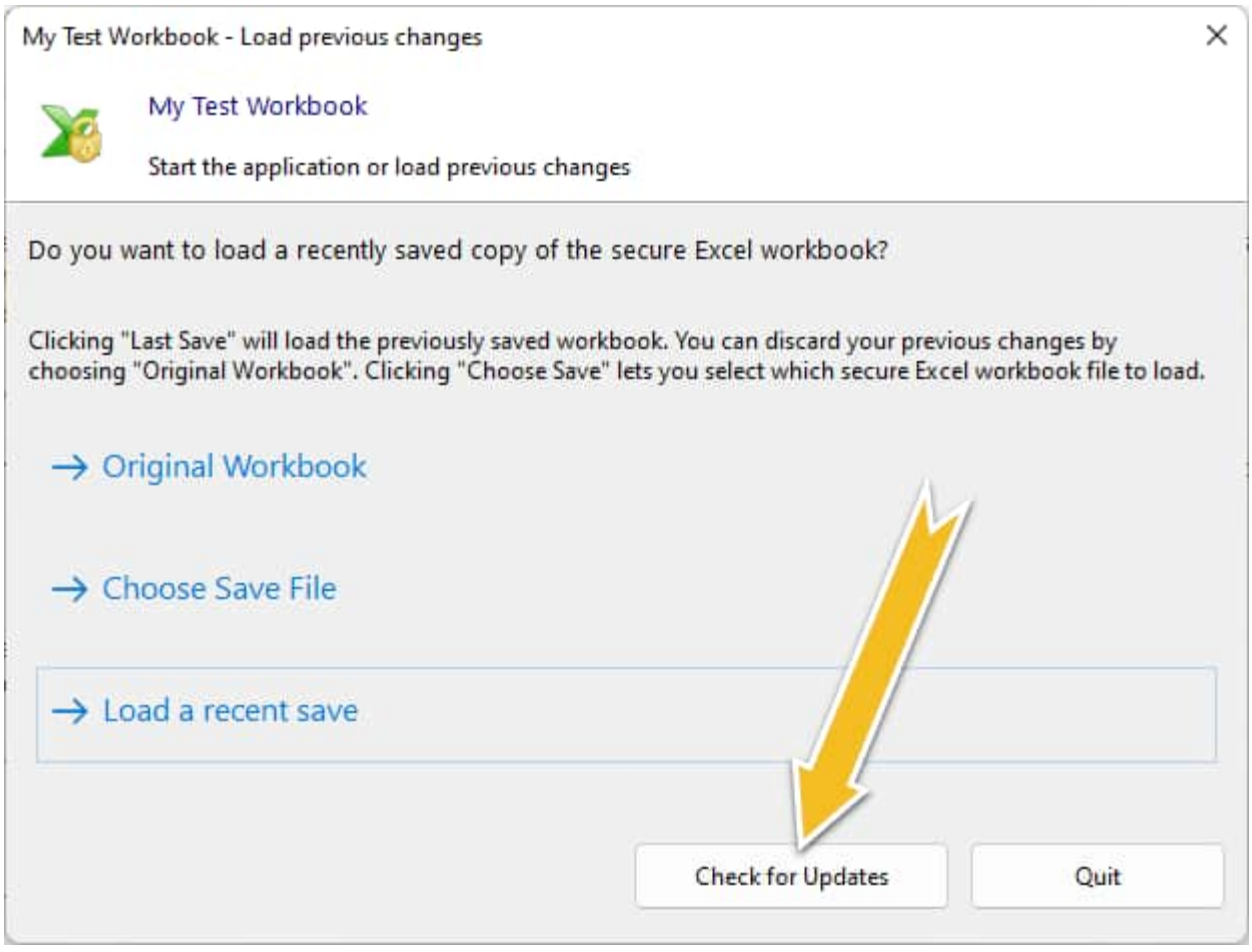
Watch a video tutorial about how to set up automated web updates for Excel workbooks

How does the Web Update proceed?

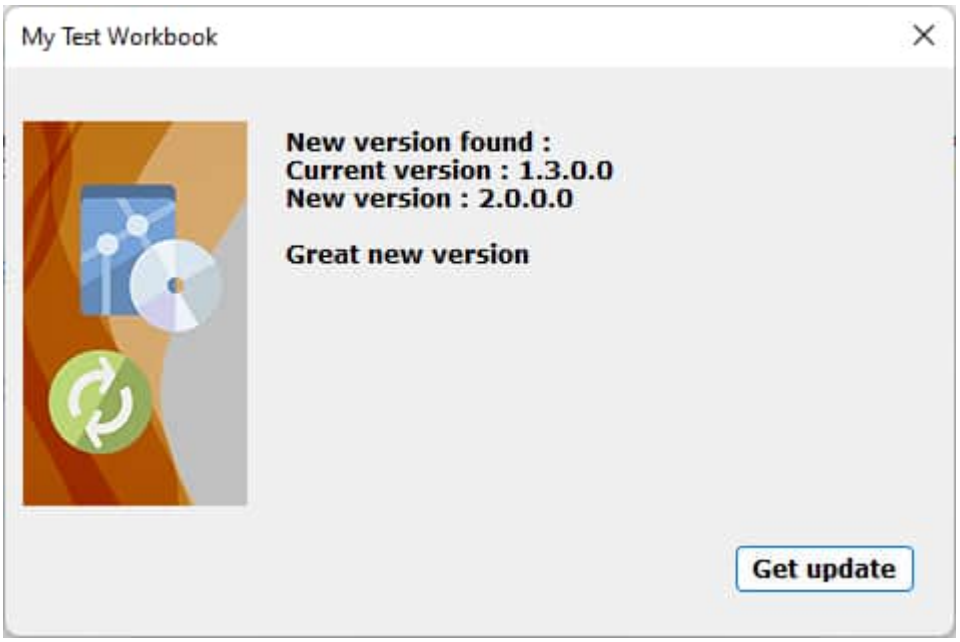
The Web Update process happens in several steps. First step consists in obtaining the control file (.INF file) from your web space (HTTP/HTTPS based location). The next step is the processing of the .INF file by verifying new files, new versions and if necessary downloading the files. Update files that are not part of the application running process (.EXE) can be immediately extracted to the proper location. If necessary, in the last step, the app is closed, the new EXE file is extracted and the updated app is automatically restarted.

Checking for new updates

You can configure your compiled workbook application to **automatically check for updates at startup** or add a "**Check for Updates**" button to the Welcome screen:



When a new version is detected, end users are informed and guided through the Web Update wizard:



They can choose to install or not the update.

End users can also start a web update through the command line (-webupdate switch).

How a new version is detected?

The File Version number specified in the EXE Icon and Version Info page of XLS Padlock is used to determine whether an upgrade is necessary or not. So, be sure to increase the File Version number when you release an update of your Excel workbook app. The File Version value is both used in standalone EXE applications and Bundle applications.

Configuration of Web Update Settings

Base URL on your web server where all Web Update files will be hosted

All files necessary for the web update (control and CAB files) must be made available for download through HTTP. You must give XLS Padlock the base URL to the location of all Web Update files hosted on your web space.

For instance, enter https://www.yourwebsite.com/myfolder

We recommend you to use secure HTTPS URLs as they are supported by XLS Padlock.

Web Update INF Control Filename

The control file (.INF format) contains necessary instructions and data about the web update. It's the file that the publication will download first.

Local destination folder where Web Update files will be placed

Provide the path to the folder on your local disk where XLS Padlock will generate web update files. The entire contents of the folder must then be uploaded to your web space (for instance, you can do it with a FTP client) and made available at the base URL specified above.

The destination folder should be empty. If it is not empty, you'll be asked whether you want to delete its contents before generating files.

"What's new" text to be displayed (optional)

The news that you may want to display to your end users on the Web Update Wizard (see screenshot above) when a new upgrade is available.

To put a carriage-return-line-feed between "Line 1" and "Line 2", insert \n, such as Line1\nLine2.

Automatically check for updates at startup

Allows the publication to check for updates at startup without user interaction. The Web Update Wizard is only displayed if a new version is detected.

Generating Web Update Files

XLS Padlock will generate all files necessary for the web update (control and CAB files) in the local destination folder. You can then upload these files to your web space.

There are two types of files:

- the .INF control file with instructions for the publication;
- the .CAB compressed file that contains the new publication EXE file and its associated external files.

The .CAB file is a standard Microsoft Cabinet file. You can open/edit it with 7-Zip for instance if you want to add more files.

Note

If the publication EXE file is installed in a Windows restricted folder such as Program Files, administrator privileges are required and this will trigger a UAC prompt.

Troubleshooting

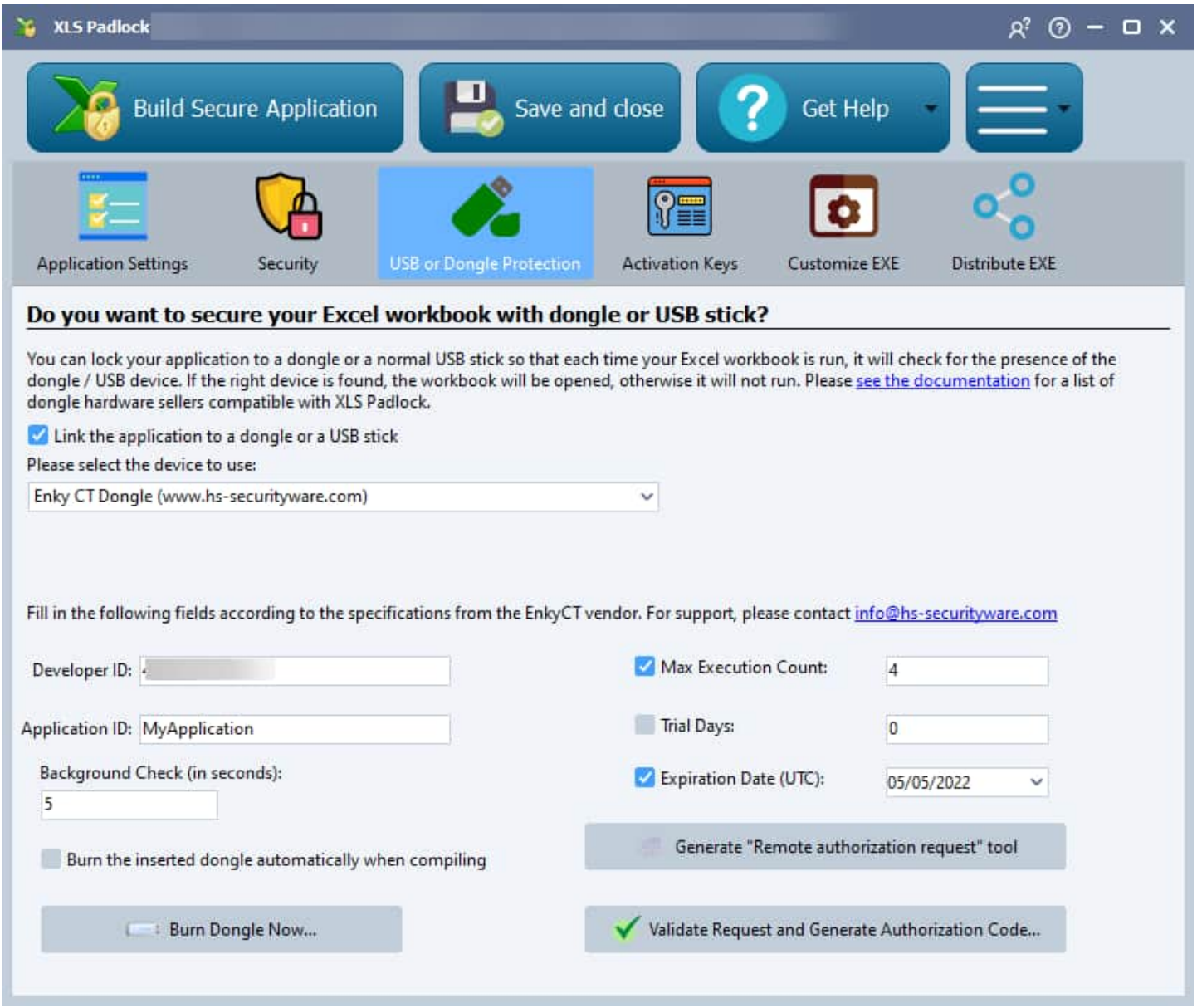
To troubleshoot the web update feature, you can create a log file by checking the "Enable WebUpdate Log" in the Advanced Options. When this option is activated, a file named WUPDATE.LOG will be created in the user's Documents directory. This log file will record all the steps undertaken by the update process, providing detailed information that can be invaluable for diagnosing any issues that might arise during the web update process.

Watch a video tutorial about how to set up automated web updates for Excel workbooks

13. USB or Dongle Protection for Excel

Thanks to XLS Padlock, you can lock your Excel workbooks to a USB stick or dongle, which means that **the compiled workbook application will not work if the correct USB device or dongle is not inserted**. The Excel workbook application will check the USB stick or dongle's presence at startup. It can also check for dongles regularly during execution if you wish.

If you want to use this protection, tick the option "**Link the application to a dongle or a USB stick**" and select the device model you have.



- Enky CT dongle
- Enky LC2 dongle
- Generic USB stick
- Enky SL dongle (support removed in XLS Padlock 2025 as deprecated, please use CT dongles)

13.1. Enky CT dongle

According to the provider, Enky CT is an easy-use, stabilized and flexible 32bit smart-card based time clock dongle, mainly used in software protection and time limitation.
For support, please contact info@hs-securityware.com or visit https://hs-securityware.com

How to use Enky CT dongle

Enter the **Developer ID** provided by HS Securityware (beware, it is case sensitive!).

Application ID = anything you want to be used to identify the dongle and it should be related to the application you are creating.

Background Check

The application will check the dongle's presence at startup. It can also check it regularly while executing.

You can define the frequency of these verifications in this field: if you set 20, the application will check the dongle's presence every 20 seconds.

If the dongle is not found, a nag screen pops up requiring insertion of the dongle in the 15 seconds. If the dongle is not inserted in time, the application exits.

Set restrictions on the application

You can limit the number of times your application can be run: tick "**Max Execution Count**" and enter the required number.

You can also make your application expire after a given number of days (tick "**Trial Days**" and enter the required number); alternatively, after a given date (tick "**Expiration Date**" and enter the required date).

After the given date, number of executions or days, when running your application, your customers will get an error message saying, "license expired".

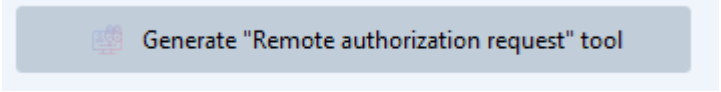
Remote dongle update features

This dongle model offers you the possibility to remotely upgrade the dongles already provided to your customers. For instance, you can extend restrictions: set a new expiration date or usage count. Customers do not have to send you their dongles physically for upgrading them. This remote update feature is based on request and authorization codes. Moreover, XLS Padlock lets you generate a stand-alone application named "remote authorization request tool" to help your customers send you the request code and enter the proper authorization code later.

By default, this feature is available and it does not require extra configuration.

How to remotely update the dongle of your customer:

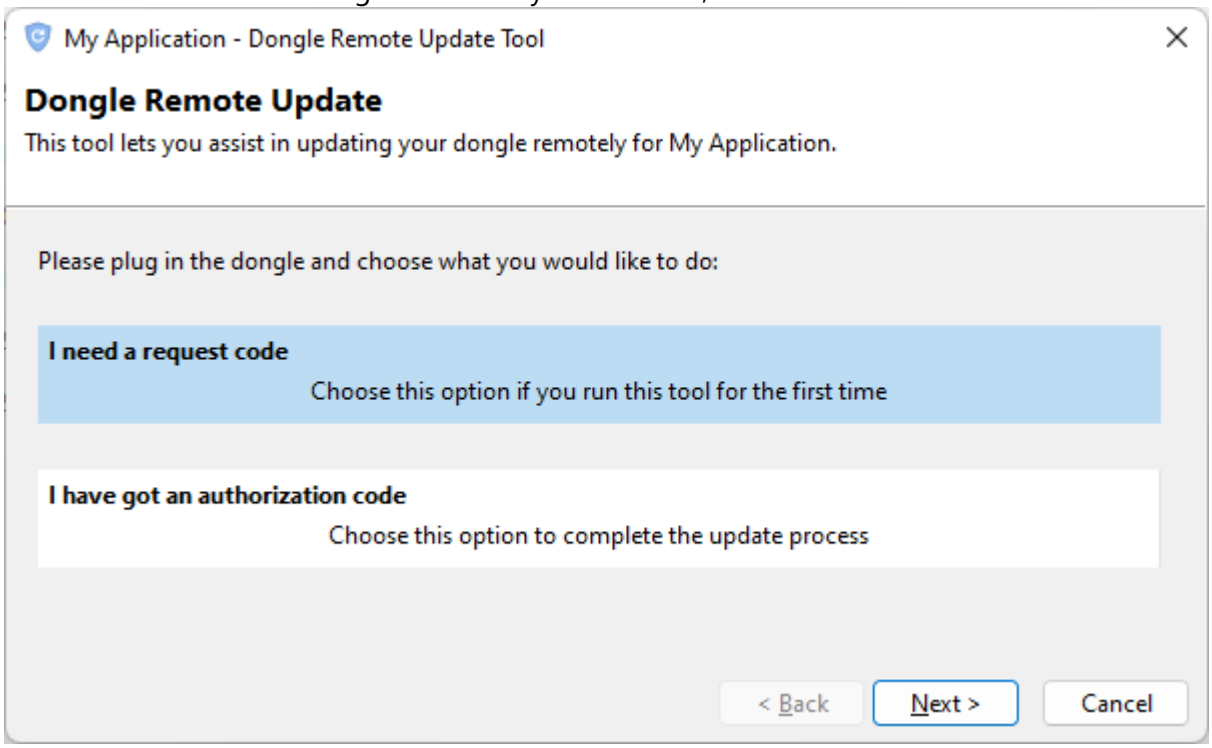
First, you have to generate the "remote authorization request tool" to send to the customer. Click the appropriate button in XLS Padlock:

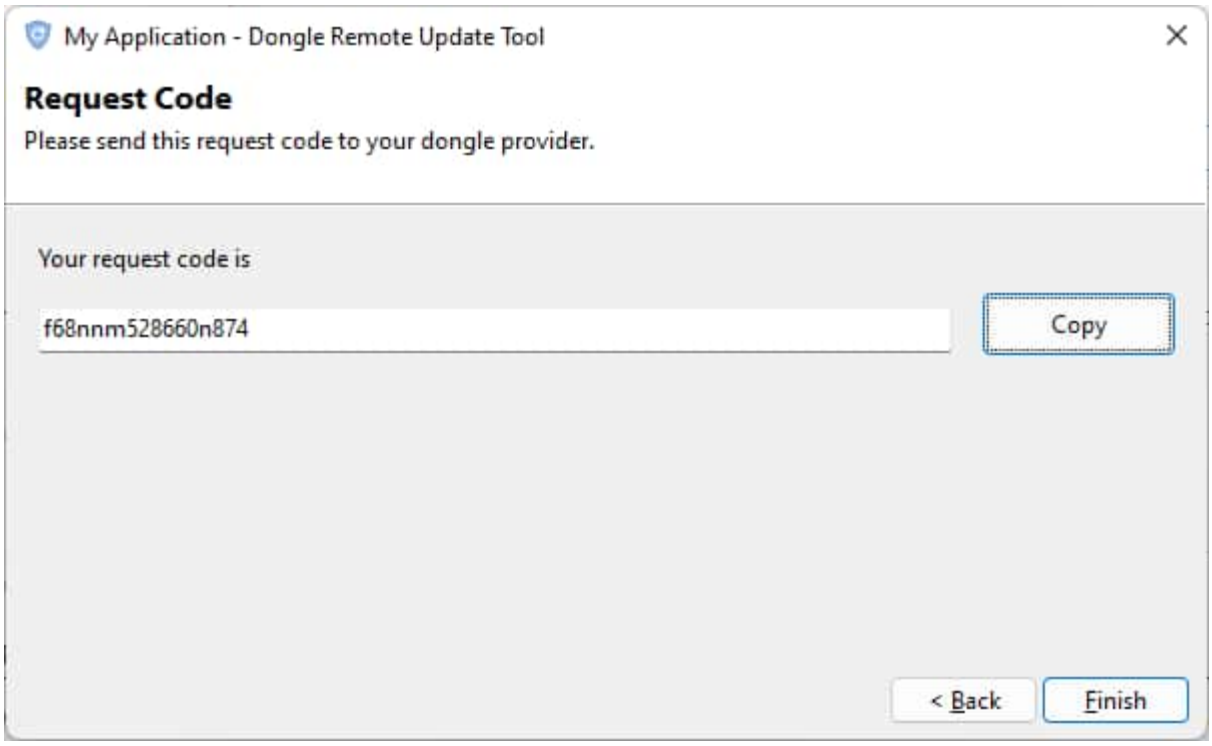


You will be prompted for the filename of the "remote authorization request tool" EXE file that will be created. Then, send this portable application EXE and its companion .DAT file to the end user who owns the dongle to be remotely updated. For better delivery, do not hesitate to zip them together.

These two files (.EXE and .DAT) must remain together or the Enky CT remote authorization request tool will not work.

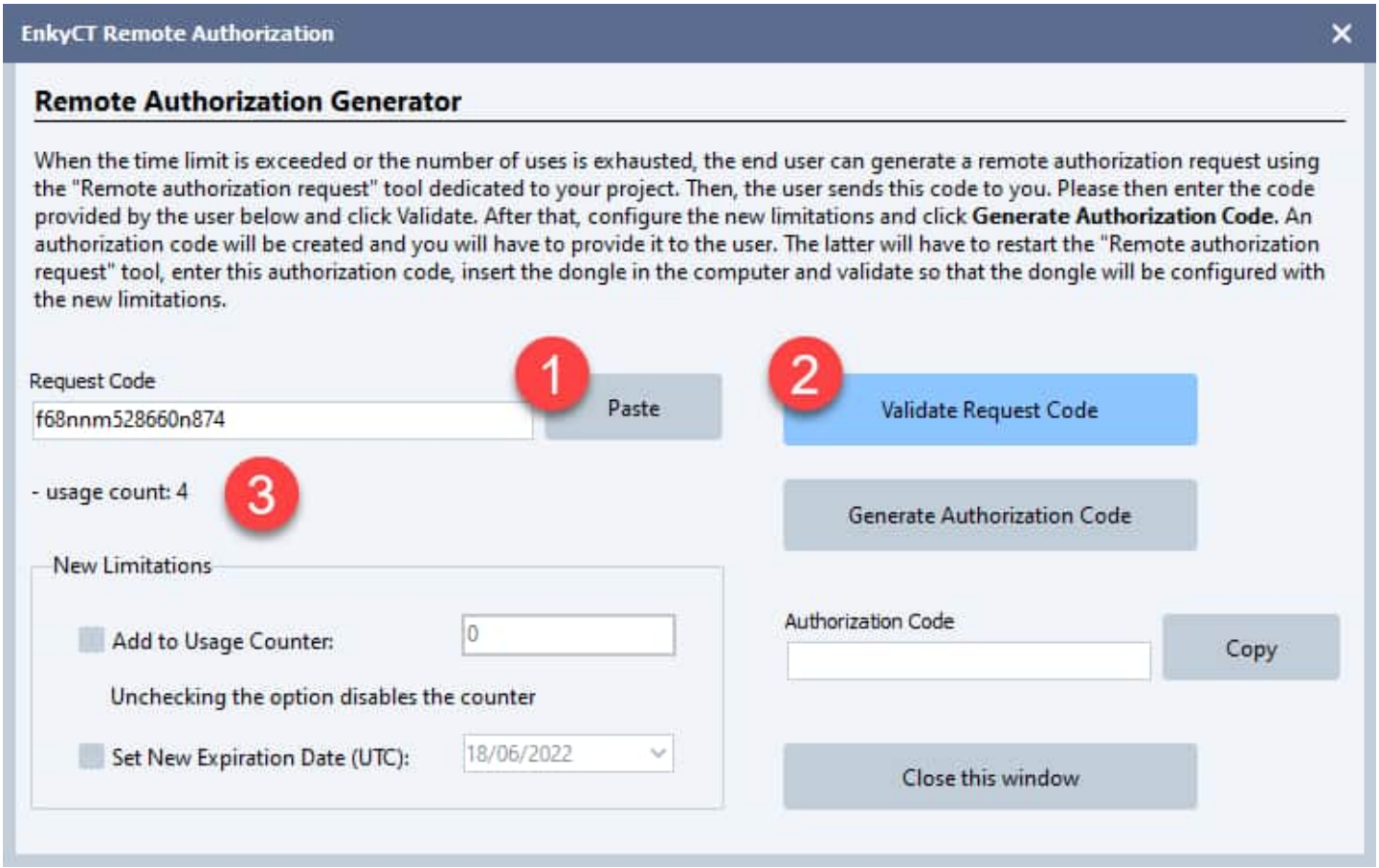
The Remote authorization request tool is a stand-alone app that you send to your customer in order to get the request code associated to their dongle. When they run the tool, it looks like this:





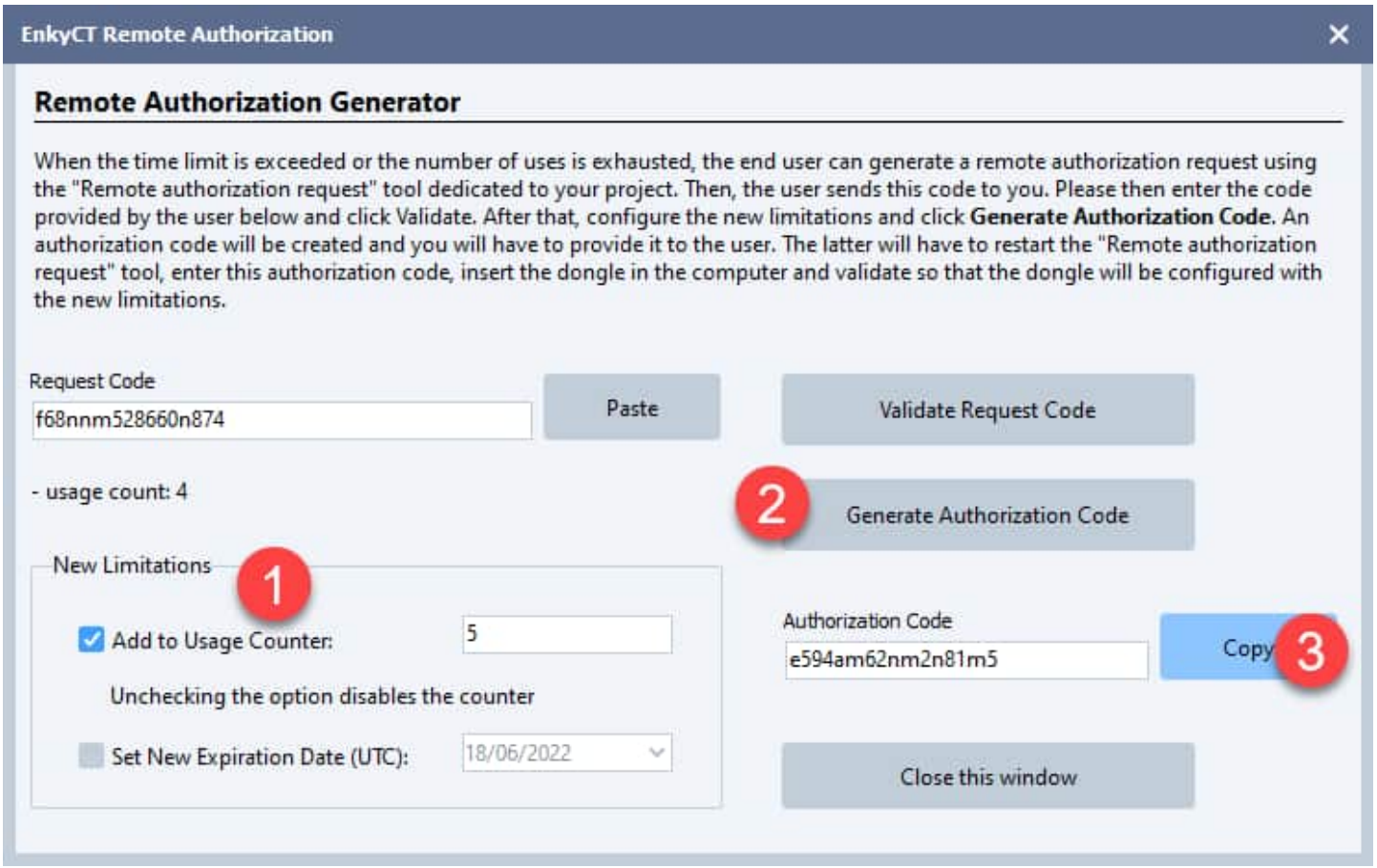
The customer then sends this request code to you.

In XLS Padlock, choose “Validate Request and Generate Authorization Code”, paste the request code received and click "Validate Request Code":



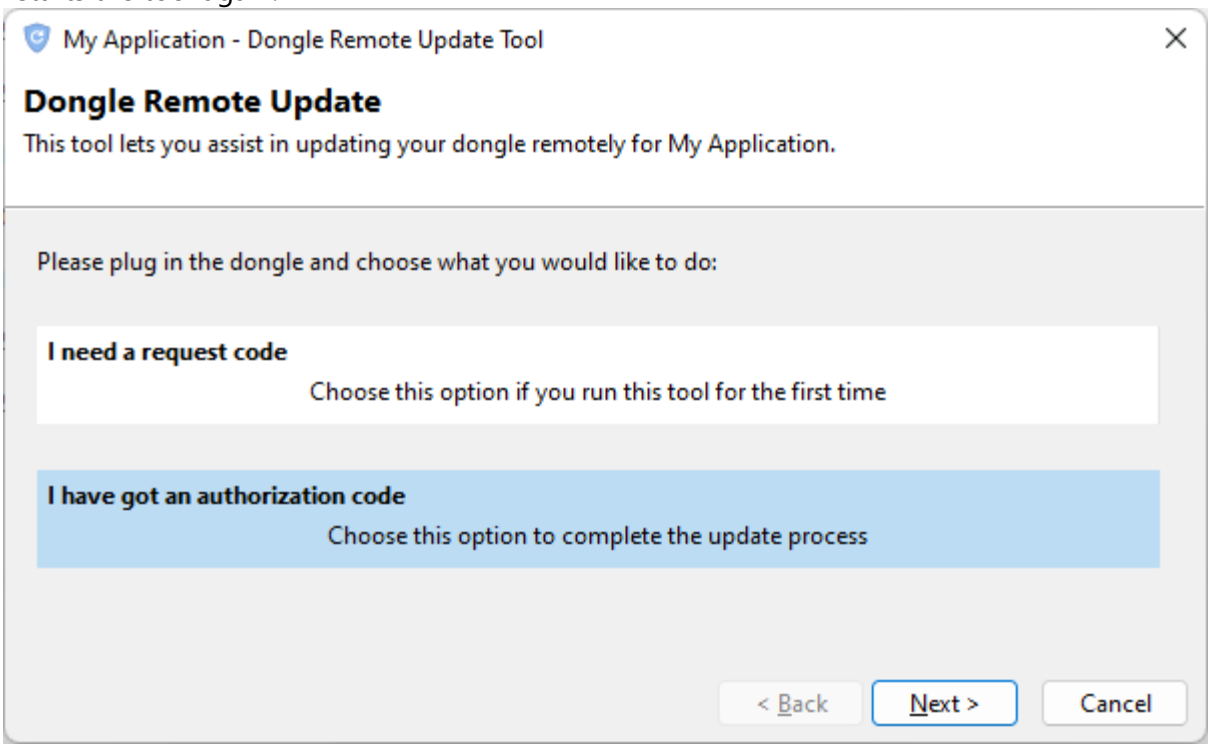
You can see the expiration data about the dongle of your customer and set new limitations. The dongle API offers only two update possibilities: usage counter and new date expiration. No trial days.

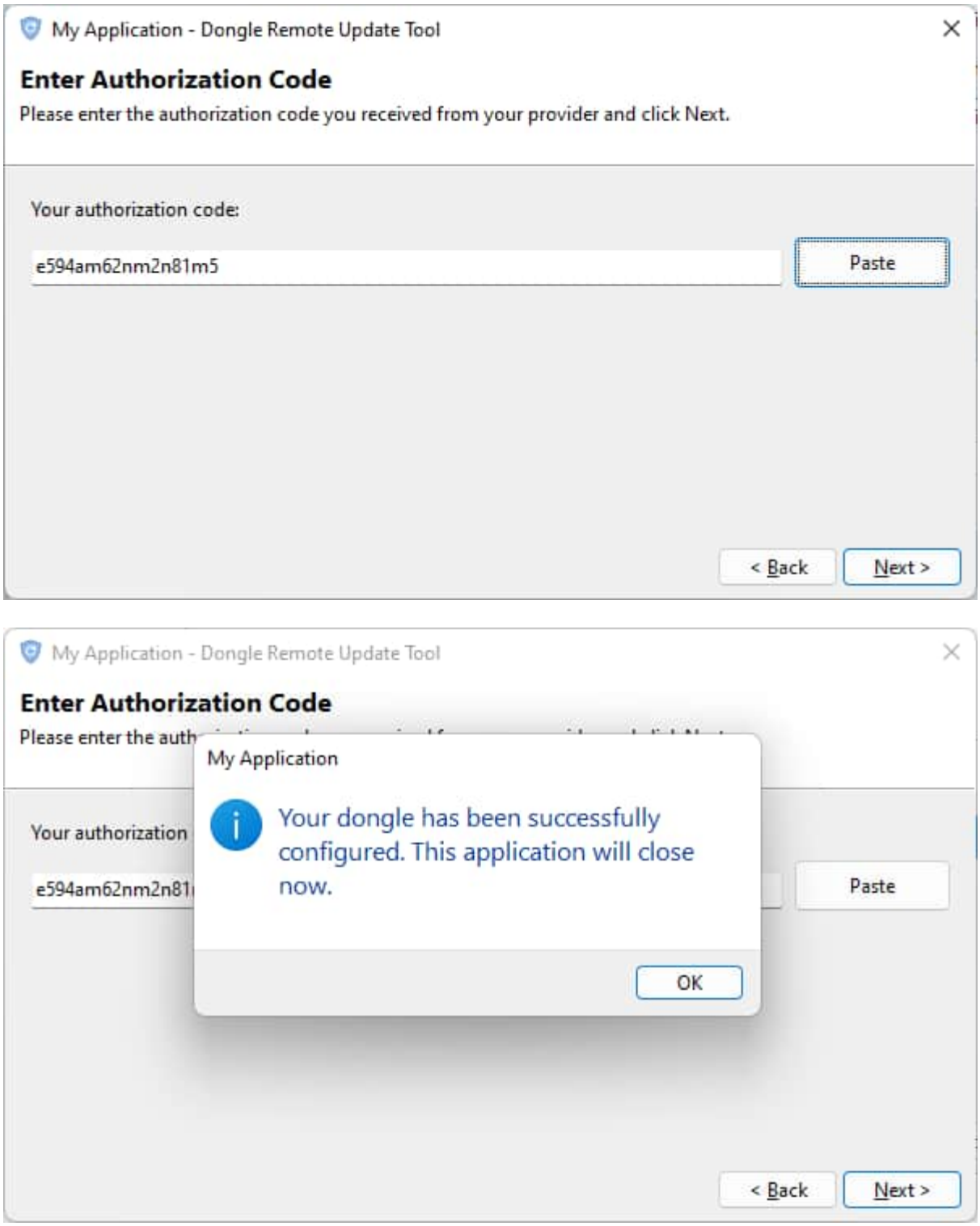
For instance, we choose to add 5 new runs:



Afterwards, you generate the authorization code and send it back to the customer.

The latter starts the tool again:





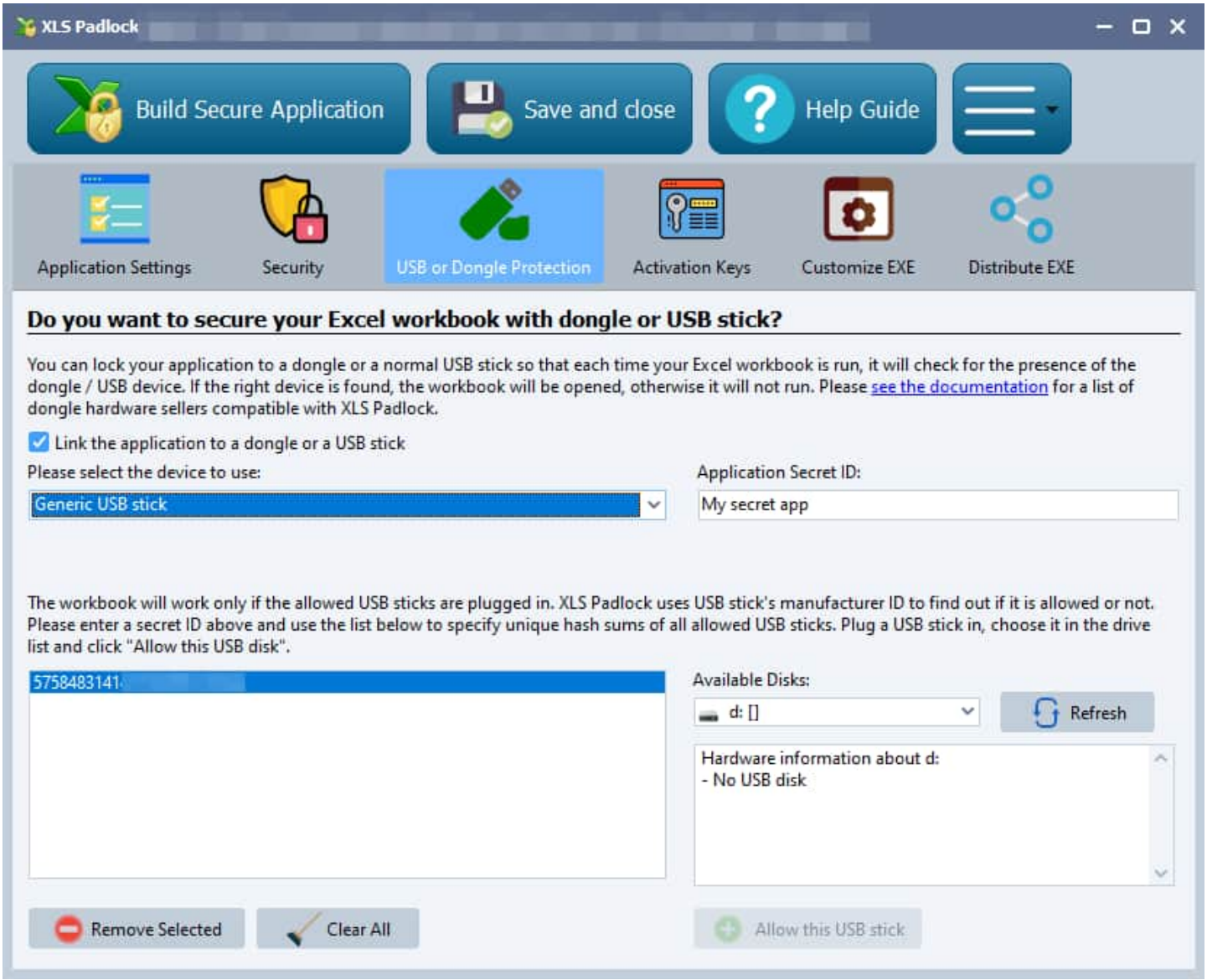
The customer's dongle has been successfully updated with the new restrictions you decided.

13.2. Generic USB stick

The workbook will open **only if the allowed USB sticks are plugged in**. This protection is based on the built-in manufacturer ID each USB stick has got.

- First, you must enter an Application Secret ID. This ID is combined with the USB stick's manufacturer ID to generate a unique hash sum for each USB stick.

When the application is launched, it scans all inserted USB disks and computes all hash sums. If one of these hash sums match the authorized one(s), the protected workbook is opened. Otherwise, the following error message is displayed: "The required security USB stick is not found. Please insert it and restart this application" and the application exits.



To authorize a given USB stick in XLS Padlock:

- Plug the USB stick you want to authorize in, choose it in the drive list and click "Allow this USB disk". XLS Padlock lets you know whether the inserted USB stick is compatible or not.
- The hash sum is computed and added to the list on the left side.
- Compile your application.

Notes:

- You can authorize several USB sticks.
- USB stick protection has less flexibility and security features than dongle protection.
- To edit an existing USB stick, just double-click on its manufacturer ID in the list and you will be prompted for any modification.

13.3. Enky LC2 dongle

For support, please contact info@hs-securityware.com or visit https://hs-securityware.com

From the manufacturer: the Enky LC dongle is one of the most cost-effective software protection dongles. Enky LC has a matured 8-bit chip developed by top class manufacturer and is a standard USB interface-based HID device, which can be used driverless under Windows.

How to use Enky LC2 dongle

1. Enter the **Developer ID** you received from HS-Security Ware when purchasing your dongles.
2. Enter a unique **Product ID** associated to your workbook. This ensures that only Enky LC dongles with the correct product ID will be accepted by your application.
3. Build your secure application.

Steps to configure an Enky LC2 dongle

Dongles received from HS-Security Ware are configured to work with XLS Padlock. In order to link the dongle to your application, you have to burn it first. This can be done with the "**Burn Dongle Now**" button. Plug in the dongle you want to burn, and press the button. The dongle is configured and a confirmation message is displayed. You now have a dongle configured to unlock your application and you can distribute it to your customers along with the application EXE file.

The Burn Dongle process can be automatically done by XLS Padlock when compiling your application, provided that a compatible dongle is plugged in (it will be listed in the compilation log).

Background check

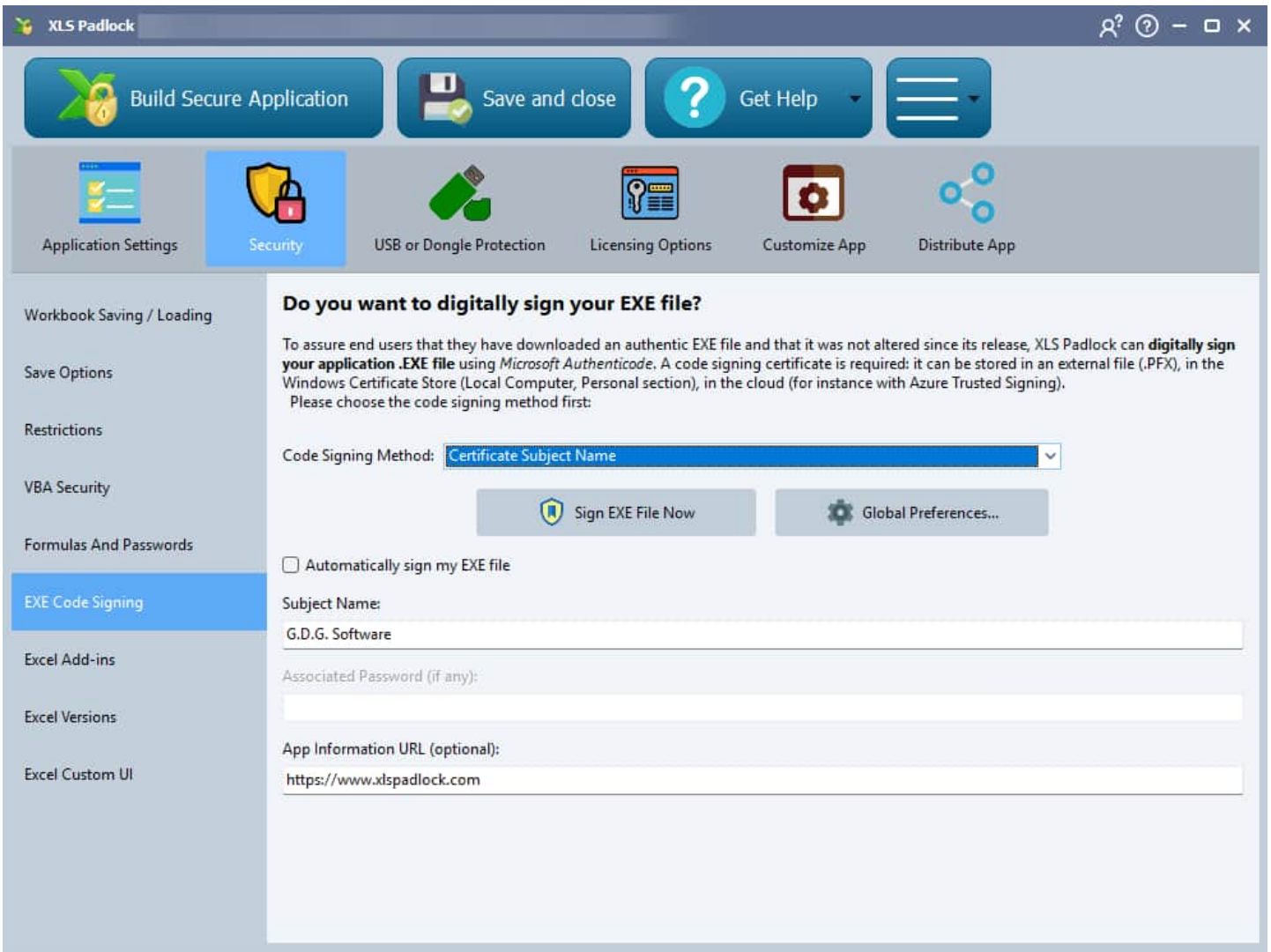
In order to open the protected Excel workbook, a dongle with the correct Developer ID and Product ID has to be plugged in. Moreover, in order to prevent users from removing the dongle while still running the application, you can enable a background check. In that case, the dongle will be checked every X seconds - if the dongle is not available, the application will freeze and request it. The customer has the choice to insert the dongle again or Excel will be terminated.

To do so, enter the X interval in seconds or leave the field blank to disable. For instance, you can enter "15" and the dongle will be checked every 15 seconds.

14. Code sign your EXE files (digital signature)

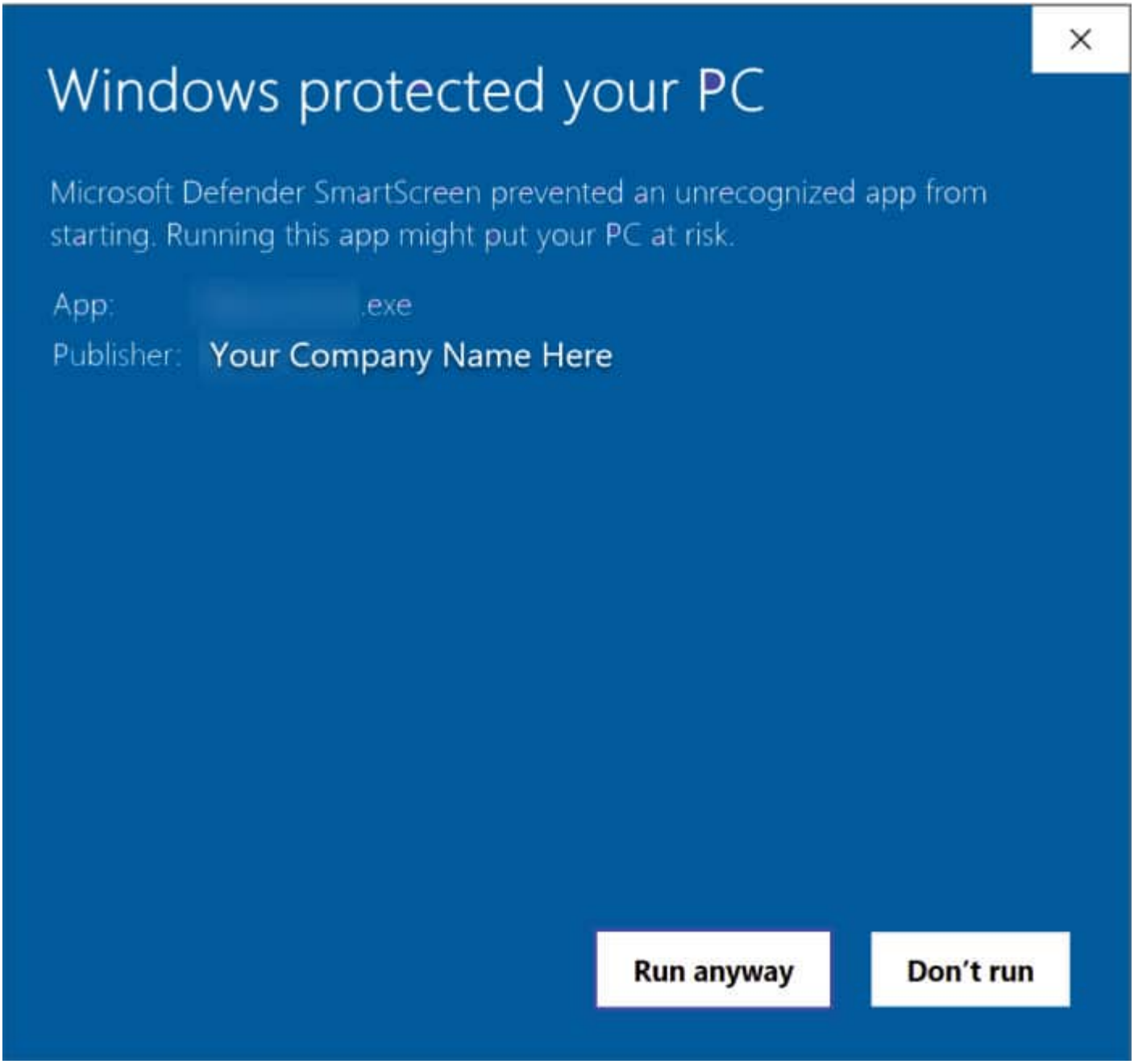
Code sign your EXE files (digital signature)

When you **digitally sign a standalone workbook EXE file** (a process known as code signing), you provide assurance to end users that the file they downloaded is authentic and hasn't been altered since its release. Digital signing relies on Microsoft Authenticode® technology, enabling both end users and the operating system to verify that the program code originates from a trusted publisher.



XLS Padlock simplifies the code signing process by handling the necessary steps internally.

If you plan to distribute your standalone workbook EXE files over the Internet, **code signing is strongly recommended**. It helps prevent web browsers and Windows from displaying "Unidentified Publisher" warnings (see example below) and can also reduce the likelihood of false positives from antivirus software.



How to Obtain a Code Signing Certificate

To sign your application, you need a valid code signing certificate from a trusted Certificate Authority (CA). CAs, such as Sectigo or Digicert, act like digital notaries, verifying your identity and issuing certificates. Be sure to purchase a **Code Signing Certificate**, as other types (like SSL/TLS certificates) are not compatible with Authenticode.

Traditional code signing certificates can be expensive, often costing several hundred dollars per year.

💡 An increasingly popular and often more cost-effective alternative is **Azure Trusted Signing**, a cloud-based code signing service provided by Microsoft. It eliminates the need for physical USB tokens and streamlines the management process. XLS Padlock fully supports signing with Azure Trusted Signing.

Token-Based Certificates and Hardware Security Modules (HSMs)

Following changes mandated by the Certificate Authority/Browser (CA/B) Forum, effective June 1, 2023, code signing certificate private keys must be stored on secure hardware. This means using a hardware security module (HSM) or a cryptographic token that meets standards like FIPS 140-2 Level 2 or Common Criteria EAL

4+. This requirement enhances security by preventing the misuse of stolen private keys.

Due to this change, the traditional method of exporting/importing certificates using password-protected PFX files is becoming obsolete for newly issued certificates. Instead, certificates are typically installed directly into the Windows Certificate Store from the hardware token.

XLS Padlock works seamlessly with token-based certificates. Simply ensure the hardware token containing your private key is physically connected to your computer when you initiate the signing process.

ⓘ If you use a token managed by the Safenet client software, you might be prompted for your token password each time you sign. To avoid repetitive prompts during a session, you can often configure the Safenet client's **Enable single logon** option. This allows you to enter the password once per session.

Configuring Code Signing in XLS Padlock

In the XLS Padlock interface, navigate to the **Security** tab and then select **EXE Code Signing**. To enable signing, first choose your preferred **Code Signing Method**:

- **PFX File:** This method uses a certificate stored in a Personal Information Exchange (.pfx) file. While still supported by XLS Padlock for older certificates, this format is being phased out for new certificates due to security mandates.
  - **PFX File Path:** Browse to or enter the full path to your .pfx file.
  - **Associated Password:** Enter the password required to access the private key within the .pfx file, if applicable.
- **Certificate Subject Name:** This method locates your certificate in the Windows Certificate Store (either Local Computer or Current User store) based on its Subject Name (often your organization's name). This is a common method when using certificates installed from hardware tokens.
  - **Subject Name:** Enter the exact Subject Name of the certificate as it appears in the Windows Certificate Store (e.g., "CN=Your Company Name, O=Your Company Name, L=City, S=State, C=Country").
  - **Associated Password:** Enter the password for your hardware token if prompted during the signing process.
- **Certificate Thumbprint:** Similar to Subject Name, this method uses the Windows Certificate Store but identifies the certificate by its unique Thumbprint (a SHA-1 hash). This is often the most reliable way to specify a certificate in the store, as thumbprints are guaranteed to be unique.
  - **Thumbprint:** Enter the certificate's thumbprint. You can find this in the certificate details in the Windows Certificate Store Manager (certmgr.msc or certlm.msc).
  - **Associated Password:** Enter the password for your hardware token if prompted during the signing process.
- **SignTool Commands:** This advanced method allows you to provide custom commands for Microsoft's SignTool.exe utility. It offers the greatest flexibility, allowing you to specify different hash algorithms (including SHA-1 if absolutely necessary, though SHA-256 is strongly recommended), timestamp servers, or other specific parameters. XLS Padlock will automatically detect SignTool.exe if it's installed with Visual Studio or the

Windows SDK.

Enter the command-line arguments for SignTool.exe, one command per line. Use the marker {$OUTPUTFILES} to represent the full path to the EXE file being signed. XLS Padlock will substitute this marker automatically.

Example:

```
sign /tr http://timestamp.sectigo.com /td sha256 /fd sha256 /a /v "{$OUTPUTFILES}"
```

Azure Trusted Signing: Use this method to sign your application with Microsoft's Azure Trusted Signing service. You'll need an active Azure subscription and a configured Trusted Signing account. XLS Padlock will check for the required SignTool.exe and the Azure Trusted Signing DLL (Azure.CodeSigning.Dlib.dll).

- **Trusted Signing Account Endpoint:** Enter the endpoint URL provided by your Azure Trusted Signing account.
- **Trusted Signing Account Name:** Enter the name of your Azure Trusted Signing account.
- **Certificate Profile Name:** Enter the name of the certificate profile you want to use within your Trusted Signing account.

▸ Read our Signing with Azure Trusted Signing And XLS Padlock Tutorial.

⚠ Before using this, ensure you have installed the Microsoft Azure CLI and logged in using **az login**. See the Microsoft documentation for setup details.

Digest Algorithm (SHA-256)

For enhanced security and compatibility with modern operating systems, XLS Padlock uses the **SHA-256 digest algorithm** by default when signing your EXE file using the PFX File, Certificate Subject Name, Certificate Thumbprint, or Azure Trusted Signing methods. Timestamping is also performed using an RFC 3161 SHA-256 timestamp server.

If you have specific legacy requirements to use a different algorithm like SHA-1 (not recommended), you must use the **SignTool Commands** method and specify the desired algorithm (e.g., `/fd sha1`) and corresponding timestamp server (`/t`) in your custom command.

Application Information URL

The optional **App Information URL** field allows you to embed a web link directly into your digital signature. Users can access this link from the certificate details, typically directing them to your product page or company website for more information.

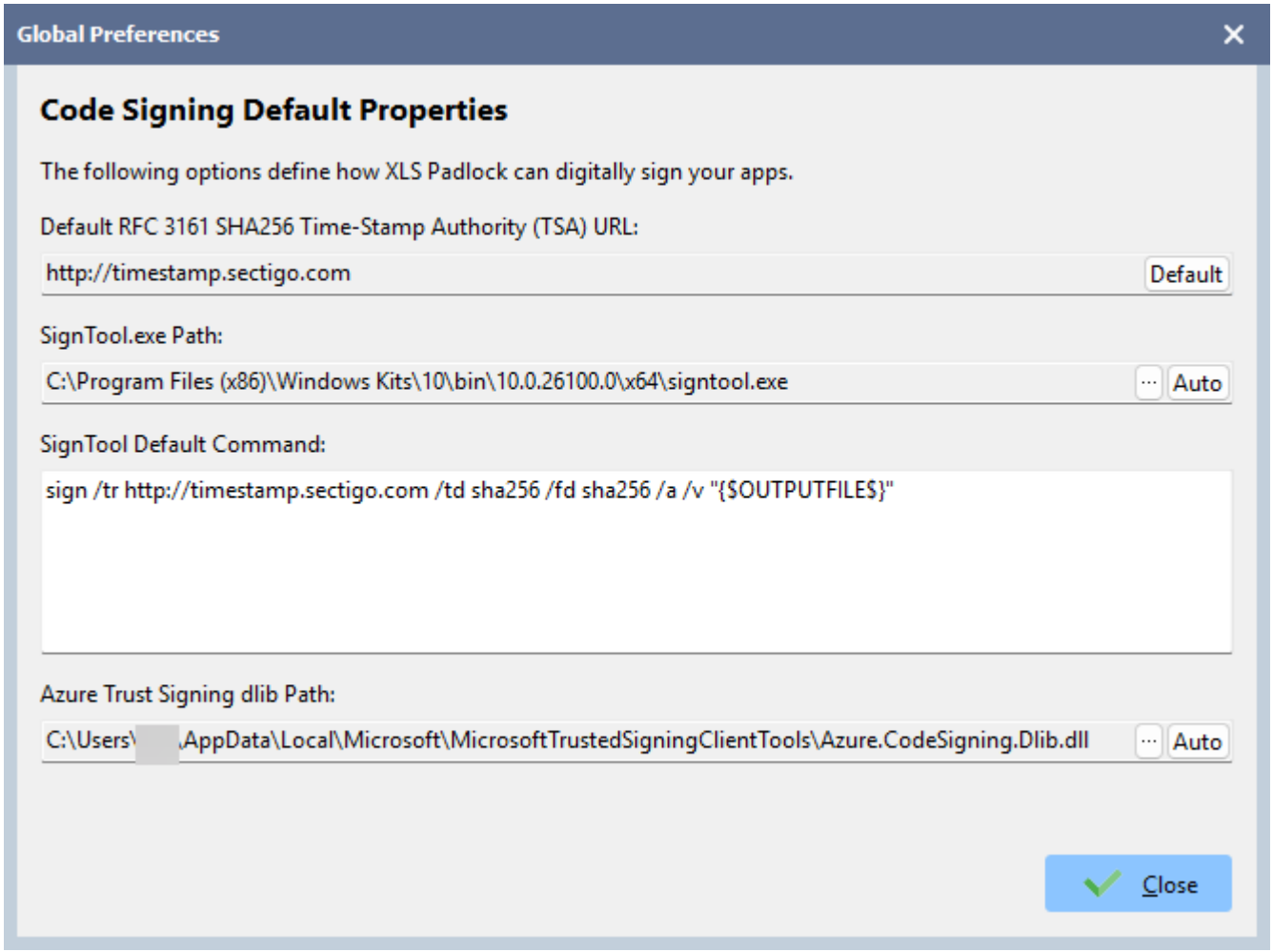
Performing the Signing

Manual Signing: After configuring your chosen method, click the **Sign EXE File Now** button to immediately sign the last built EXE file for the current project.

Automatic Signing: Check the **Automatically sign my EXE file** option. With this enabled, XLS Padlock will automatically sign the EXE file each time you build your secure application (by clicking "Build Secure Application").

Global Preferences

The **Global Preferences...** button opens a dialog box where you can configure default settings related to code signing that apply across your XLS Padlock projects.



Default RFC 3161 SHA256 Time-Stamp Authority (TSA) URL: This is the URL of the timestamp server used to add a secure timestamp to your signature. Timestamping ensures the signature remains valid even after the certificate expires. The default is often sufficient (e.g., http://timestamp.sectigo.com). This URL is used automatically by the PFX, Subject Name, Thumbprint, and Azure methods unless overridden by custom SignTool commands.

SignTool.exe Path: The full path to the Microsoft SignTool.exe utility. Click **Auto** to let XLS Padlock attempt to automatically locate it (usually in the Windows SDK directories).

SignTool Default Command: This command template is used internally by XLS Padlock when signing via the PFX File, Certificate Subject Name, or Certificate Thumbprint methods. It defines the standard parameters like the timestamp server (`/tr`), digest algorithm (`/td`, `/fd`), and the file to sign (`{$OUTPUTFILES}`). You generally don't need to change this unless you have very specific global signing requirements.

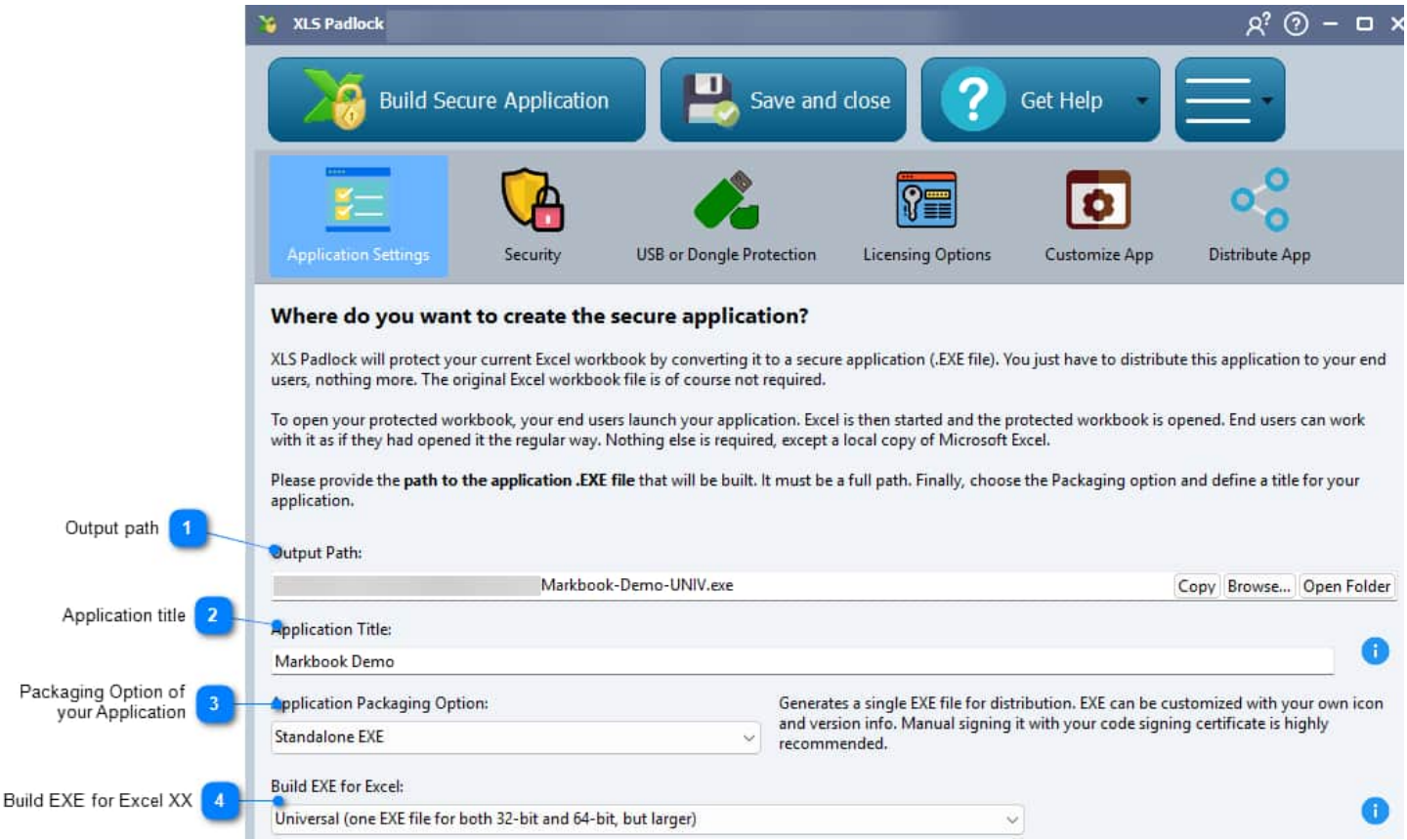
Azure Trust Signing dlib Path: The full path to the Azure.CodeSigning.Dlib.dll file required for the Azure Trusted Signing method. Click **Auto** to let XLS Padlock attempt to locate it (usually in the user's AppData directory after installing Azure CLI tools).

If an error occurs during the code signing process (e.g., certificate not found, incorrect password, SignTool failure, timestamp server unreachable), check the detailed messages in the XLS Padlock compilation log. The log file is typically named [Your Workbook Filename].xplcompil.log and is located in the same directory as your workbook.

15. All XLS Padlock Options

15.1. Application Settings

The mandatory settings for your project are on this page:



1 Output path

2 Application title

3 **Packaging Option of your Application**

4 Build EXE for Excel XX

- Configure Advanced Options
- Add Companion Files

15.1.1. Output path

XLS Padlock generates a single EXE file. Select the output path where you want your application EXE file to be created. **Do not forget the .exe extension.**

Relative paths are also accepted: the base folder will be the folder that contains the workbook file.

By default, XLS Padlock displays the path where the Excel file is saved; you can change it of course.

- Application Title

15.1.2. Application Title

All XLS Padlock Options > Application Settings > Application Title

Give a title to your application. This title displays after the file name on the upper ribbon of your application, replacing "Microsoft Excel" mention.

If you want to display the name of the save file loaded by the user, use %SAVEFILENAME% or %SAVEFULLNAME% in the title.
For instance, enter "`My Application %SAVEFILENAME%`".

If the original workbook is loaded, %SAVEFILENAME% will be empty.


%SAVEFULLNAME% will be replaced by the full path to the save file, while %SAVEFILENAME% will only be replaced by the filename without the folder path.

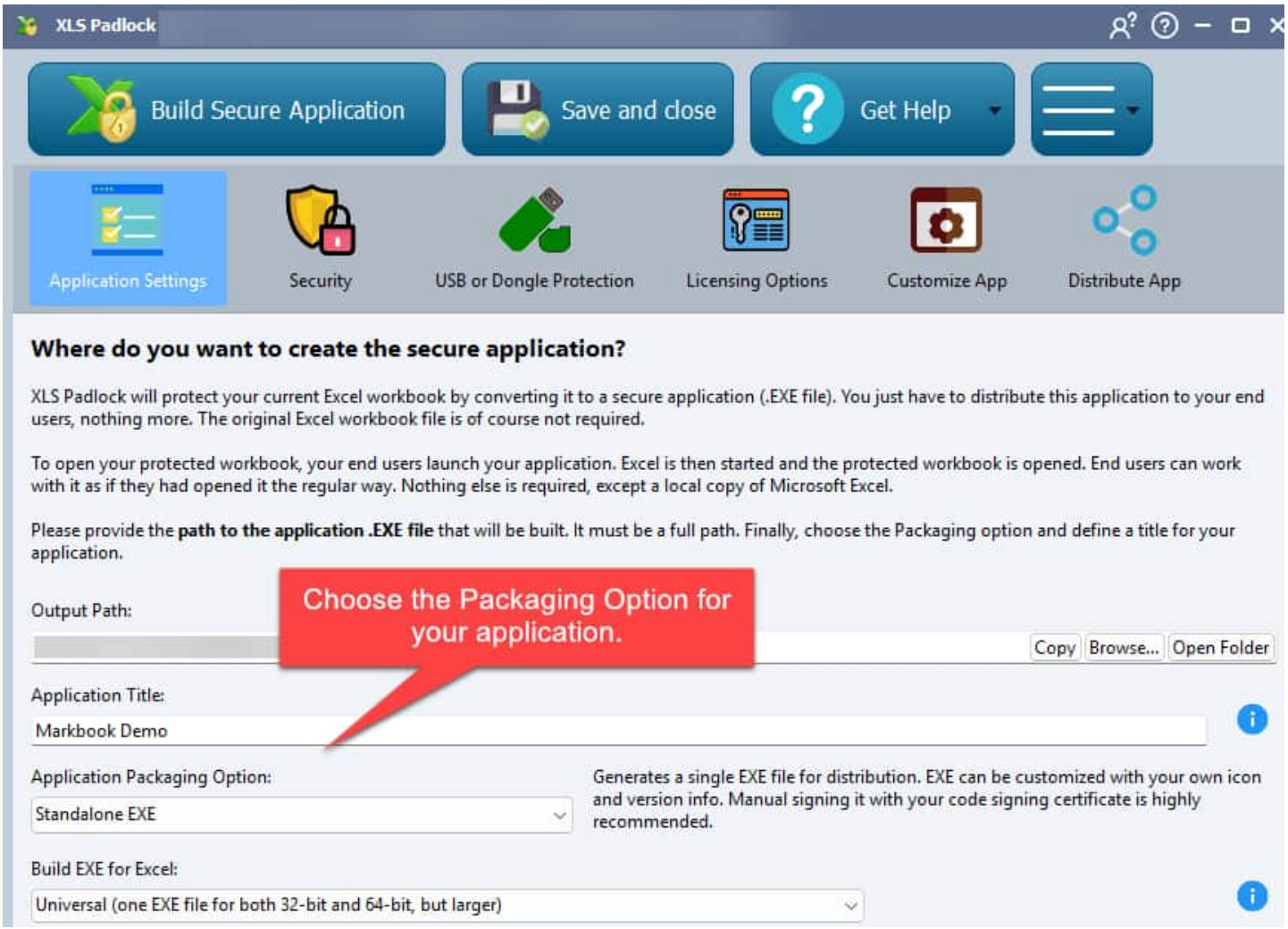
- Application Packaging Option

15.1.3. Application Packaging Option

All XLS Padlock Options > Application Settings > Application Packaging Option

The **Application Packaging Option** allows you to build your secure Excel workbook applications in two distinct formats.


We recommend you to choose **Standalone EXE** if you have a digital signing certificate. This is not mandatory, but strongly recommended to avoid the unknown app warning from Windows SmartScreen and/or false positives that may occur due to some antivirus software.



Standalone EXE

With XLS Padlock, you can transform your Excel workbook into a standalone executable (.EXE) file, seamlessly recognized by Windows.

Distributing this .EXE file to your users is straightforward. The necessity for the original Excel workbook is eliminated, thus it is protected against copy.

EXE + XPLAPP Application Bundle

Opting for this configuration results in the creation of an EXE file coupled with a .bin64 companion file and a .xplapp data file. The EXE, already signed by our firm, gains immediate recognition from Windows SmartScreen and leading antivirus solutions, offering a path to trusted distribution.

- This distribution ensures that the EXE automatically locates and accesses the user data within the .xplapp file, given both are stored in the same folder and share the same file name.
- For convenience, XLS Padlock is capable of bundling these files into a singular Zip archive or installer.
- Employing this method substantially diminishes the chances of encountering "Unknown Application" alerts from Windows SmartScreen and avoids antivirus false positives.
- The .bin64 companion file provides support for Excel 64-bit and it is required by the EXE file.

Create 64-bit Only

The EXE generated is a pure 64-bit EXE file. It is for Excel 64-bit only. No additional .bin64 companion file is

generated in that case.

15.1.4. Build EXE for Excel XX

All XLS Padlock Options > Application Settings > Build EXE for Excel XX

Recent Excel releases offer two versions: 32-bit or 64-bit. Learn more about Office 32-bit and 64-bit versions

If your end users have Office 64-bit, you must provide them with a 64-bit executable file. XLS Padlock can generate 64-bit .exe files.

This option is for standalone executable files only.

Choose what type of EXE XLS Padlock will generate when creating your secure application:

- **32-bit only**: XLS Padlock generates an EXE file for Excel 32-bit versions.
- **64-bit only**: XLS Padlock generates an EXE file for Excel 64-bit versions.
- **Universal**: XLS Padlock creates a single EXE file for both Excel 32-bit and 64-bit versions. Thus, you do not have to manage and deliver two separate EXE files to your customers. The other side of the coin is a clear increase of the EXE file size: the EXE will double in size because we merge 32-bit and 64-bit code into one EXE.
- **32-bit and 64-bit**: XLS Padlock generates two separate EXE files. When you hit "Build Secure Application", XLS Padlock first makes an EXE for Excel 32-bit versions and then an EXE for 64-bit versions at once. To prevent EXE files from overwriting themselves, XLS Padlock automatically adds "32" and "64" suffixes to the Output path. For instance, "My App.exe" will result in "My App32.exe" and "My App64.exe".

If end users run a 64-bit .EXE file with a 32-bit Office installation (and vice-versa), an error message will be displayed, asking end users to contact you for a correct EXE version.

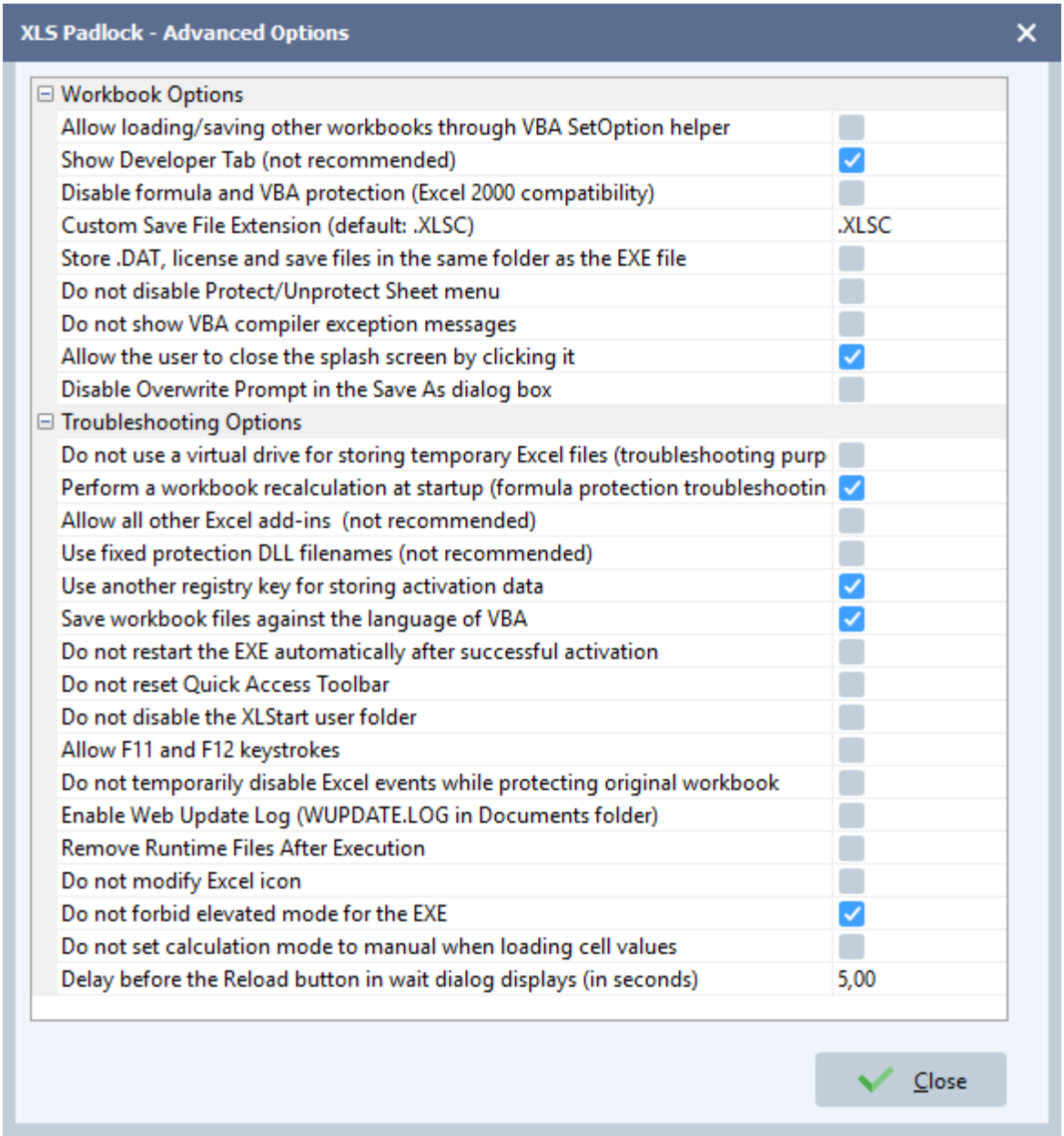
We strongly recommend you to code sign your application EXE file if you use the Universal mode, because a temporary EXE file will be created at runtime and some antivirus dislike this behavior. Code signing will definitively help to lower false positives.

By default, applications distributed in the bundle format are already Universal.

Configure Advanced Options

15.1.5. Configure Advanced Options

In “Application Settings”, click “Configure Advanced Options” button to display the following window:



These options are for advanced users: use them only on advice of technical support.

Available Advanced Options

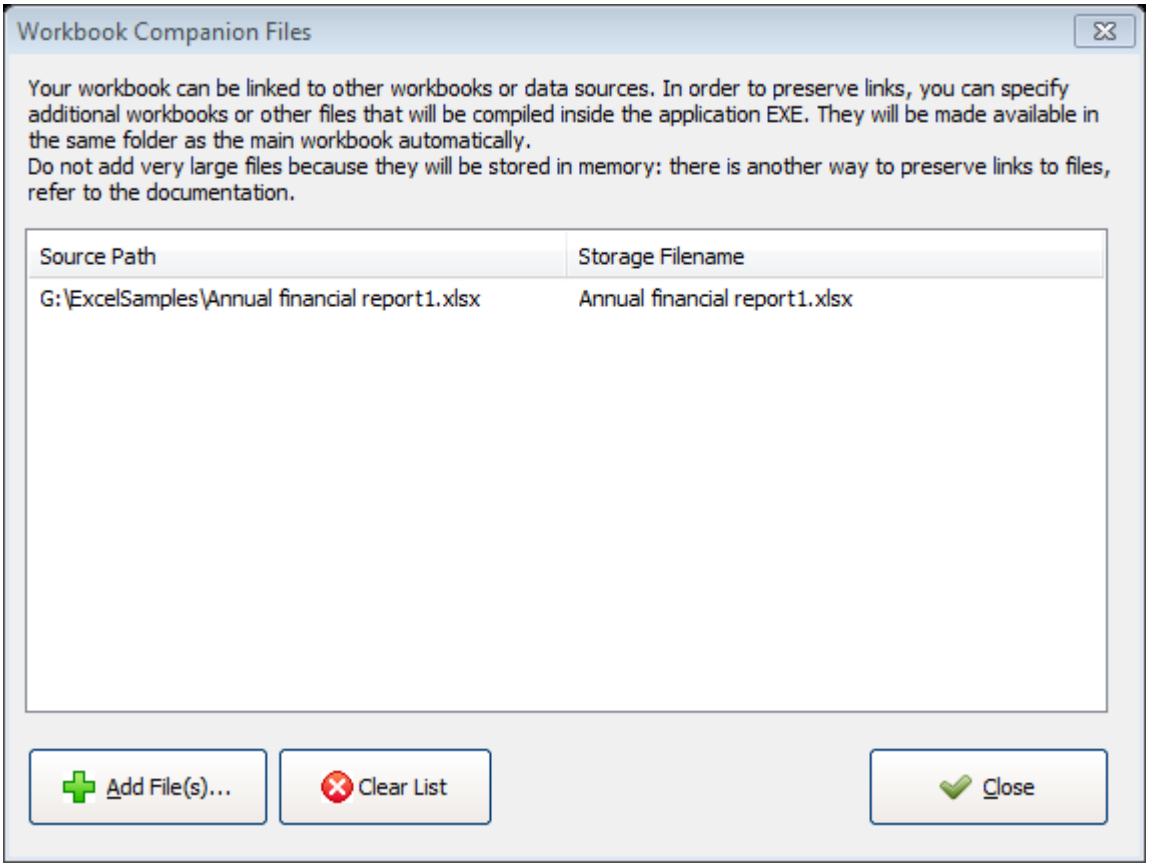
- Allow loading/saving other workbooks through VBA SetOption helper
- Show Developer Tab (not recommended)
- Disable formula and VBA protection (Excel 2000 compatibility)

- Custom Save File Extension (default: XLSC)
- Store .DAT, license and save files in the same folder as the EXE file (useful for portable applications)
- Do not disable Protect/Unprotect Sheet menu
- Do not show VBA compiler exception messages
- Allow the user to close the splash screen by clicking it
- Disable Overwrite Prompt in the Save As dialog box
- Do not use a virtual drive for storing temporary Excel files (troubleshooting purposes).
- Perform a workbook recalculation at startup (formula protection troubleshooting purposes).
- Allow all other Excel add ins (not recommended)
- Use fixed protection DLL filenames (not recommended)
- Use another registry key for storing activation data
- Save workbook files against the language of VBA
- Do not restart the EXE automatically after successful activation
- Do not reset Quick Access Toolbar
- Do not disable the XLStart user folder
- Allow F11 and F12 keystrokes
- Do not temporarily disable Excel events while protecting original workbook
- Enable Web Update Log (WUPDATE.LOG in Documents folder)
- Remove Runtime Files After Execution
- Do not modify Excel icon
- Do not forbid elevated mode for the EXE
- Do not set calculation mode to manual when loading cell values
- Delay before the Reload button in wait dialog displays (in seconds)

15.1.6. Add Companion Files

Your workbook can be linked to other workbooks or data sources. To preserve links, you can specify additional workbooks or other files that will be compiled inside the application EXE. They will be made available in the same virtual folder as the main workbook automatically. It is even possible to include XLL add-ins.

For instance, you have a workbook with a data source available as an external text file. You can add the text file as a **Companion file**, so that Excel will be able to find the external text file when the compiled workbook is opened. This ensures that companion files are always available on other computers.



To add one or more files, click **Add Files** and select them. They are added to the list.

The Storage Filename column tells you with which filename they will be compiled inside the application EXE file and restored at runtime.

Do not add very large files because they will be stored in memory. If you would like to link to other workbooks or large data files, you can leave them in the same folder as the application EXE file and use references to these files. References can be set up with the XLS Padlock built-in Excel function named **PLEvalVar**("XLSPath"). Please refer to the section: Use external references and hyperlinks.

If you want to access companion files with VBA code, you can use the following VBA code snippet (copy and paste it into a module):

```
Public Function PathToCompiledFile(Filename As String)
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
PathToCompiledFile = XLSPadlock.PLEvalVar("XLSPath") & Filename
Exit Function
Err:
PathToCompiledFile = ""
End Function
```

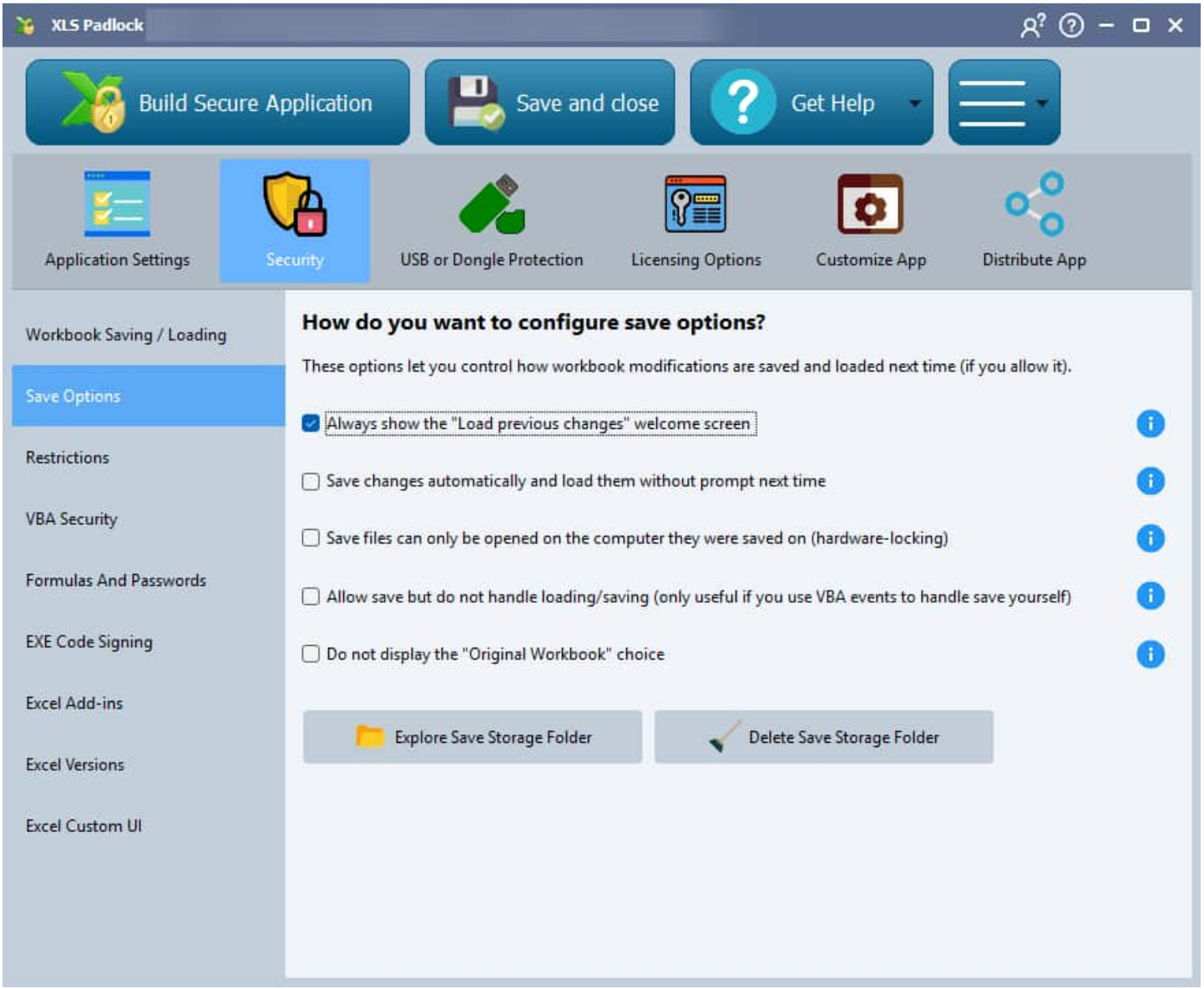
Then, you can access a companion file like this:

```
Dim wk As Workbook
```

```
Set wk = Workbooks.Open(PathToCompiledFile("Test File.xlsx"), False, False)
MsgBox wk.Sheets(1).Cells(1, 1).Value
wk.Close
```

15.2. Security

XLS Padlock offers lots of means to protect your Excel workbook: save options, restrictions, conditions to open your application (activation key, or presence of a dongle or a USB stick)... Learn more about how to secure your workbook with XLS Padlock's options: click the page and you will be redirected to the corresponding topic.

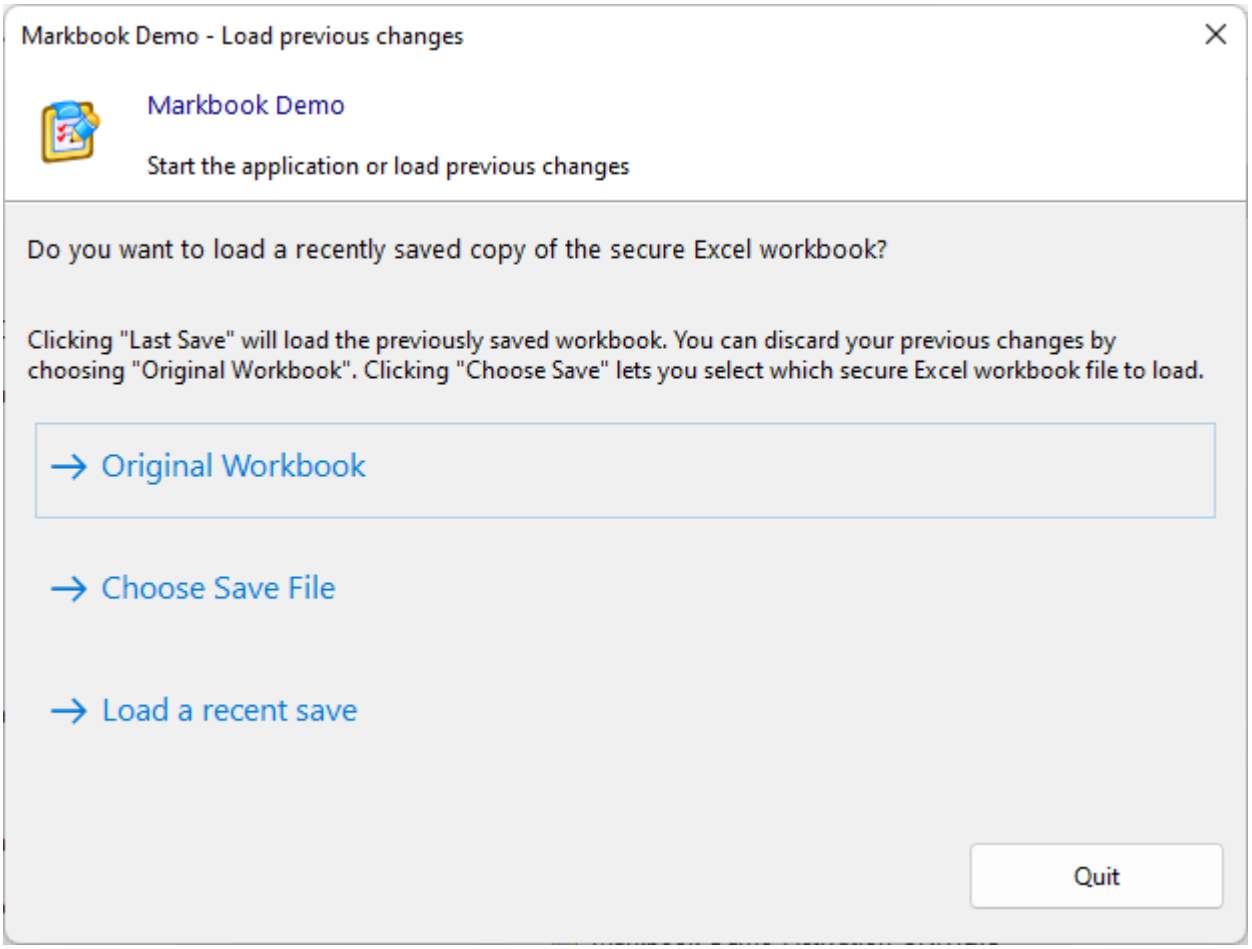


15.2.1. Save Options

15.2.1.1. Always show the "Load previous changes" welcome screen

All XLS Padlock Options > Security > Save Options > Always show the "Load previous changes" welcome screen

When end users start the secure application, if no save was previously made, the secure workbook is directly opened without any prompt. With this option enabled, a welcome dialog box is shown instead, allowing users to start the secure workbook or load an existing save file:



This is useful only if, for instance, you provide your users with both the EXE file and an existing save file, or if you move the EXE to another computer and want to load an existing save file.

15.2.1.2. Save changes automatically and load them without prompt next time

All XLS Padlock Options > Security > Save Options > Save changes automatically and load them without prompt next time

This option offers a basic load/save mechanism: user changes are automatically saved when Excel is closed and restored when the secure application is started again.

No "File Save As" dialog box nor initial prompt are displayed. On the other hand, your end users can not create several saves for the same workbook.

15.2.1.3. Save files can only be opened on the computer they were saved on (hardware-locking)

All XLS Padlock Options > Security > Save Options > Save files can only be opened on the computer they were saved on (hardware-locking)

Enable this option to prevent end users from sharing save files. The application will create save files that are **hardware-locked: they can only be opened on the computer they were created on**. If the user tries to share a save file and open it with your application on another computer, it will fail.

The application stores the current unique system ID in the save file and checks it when a user attempts to load a hardware-locked save. You can define system ID options.

As the owner of the application, you are still able to decrypt any hardware-locked save file made by your application yourself thanks to the Decrypt Save File feature of XLS Padlock

This feature is only available if you chose the Full Save mode.

15.2.1.4. Allow save but do not handle loading/saving

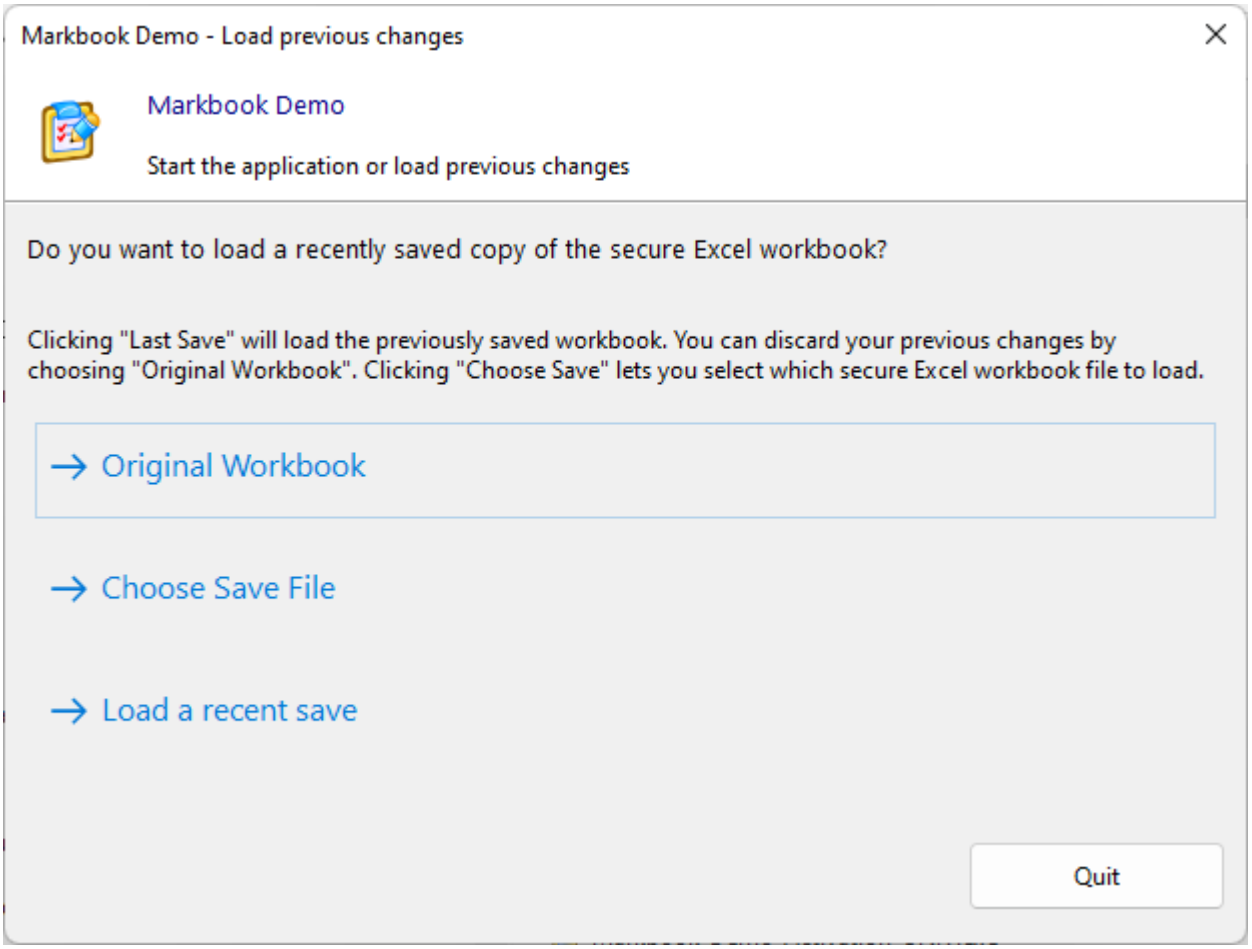
All XLS Padlock Options > Security > Save Options > Allow save but do not handle loading/saving

If you do not want the “Save as” dialog box to appear and handle saving yourself through VBA, you can tick “**Allow save but do not handle loading/saving**”. Otherwise, leave this option unchecked if you don’t want modifications to be lost when Excel closes.
VBA developers can also use VBA code extension to automatically save secure workbooks without user prompt. See 11.3 Saving a secure copy of the workbook without prompt.

15.2.1.5. Do not display the "Original Workbook" choice

All XLS Padlock Options > Security > Save Options > Do not display the "Original Workbook" choice

The "Do not display the 'Original Workbook' choice" option in XLS Padlock removes the possibility to load the original, unprotected workbook on the Welcome screen:



With this setting enabled, the welcome dialog will only offer the possibility to load an existing secure file. This option is particularly beneficial when custom save files are provided to end users at the outset, ensuring that only the secured versions of the workbook are accessible upon startup.

15.2.1.6. Explore Save Storage Folder and Delete Save Storage Folder

All XLS Padlock Options > Security > Save Options > Explore Save Storage Folder and Delete Save Storage Folder

These Explore Save Storage Folder and Delete Save Storage Folder buttons let you explore and delete the folder on your PC where the secure application stores save files (in case of automatic saves) and the save history file. If the portable mode is activated in the Advanced Options, then it's the folder holding the .EXE file (in that case, Delete will not be possible).

15.2.2. Restrictions

15.2.2.1. Allow print operations

All XLS Padlock Options > Security > Restrictions > Allow print operations

Deactivate this "Allow print operations" option if you want to disable print operations in Excel. If disabled, end users will get an error message “Print is not allowed” and will not be able to print documents.

15.2.2.2. Allow export PDF/XPS operations

All XLS Padlock Options > Security > Restrictions > Allow export PDF/XPS operations

Deactivate this "Allow export PDF/XPS operations" option if you want to disable exporting workbook content to PDF or XPS files.

15.2.2.3. Disable right click (no context menu)

All XLS Padlock Options > Security > Restrictions > Disable right click (no context menu)

Enable this "Disable right click" option to disable the mouse’s context menu displayed when you right-click on a cell. This option is generally used to prevent copying or changing cell’s data.

15.2.2.4. Disable all ribbons and toolbars

All XLS Padlock Options > Security > Restrictions > Disable all ribbons and toolbars

This "Disable all ribbons and toolbars" option will hide all built-in tabs on the Ribbon and most commands on the Microsoft Office Menu.
It will only work with Excel 2007 and later. You can use “Minimum version of Excel required for your workbook” to verify that the workbook is only opened in Excel 2007 or later.

15.2.2.5. Disable "cell copy and cut to clipboard" commands

All XLS Padlock Options > Security > Restrictions > Disable "cell copy and cut to clipboard" commands

This lets you disable copy and cut commands. Thus, end users cannot copy data from cells to clipboard for reuse in other workbooks.

15.2.2.6. Disable Formula Bar

All XLS Padlock Options > Security > Restrictions > Disable Formula Bar

The Disable Formula Bar" option will make the application hide the formula bar of Excel. Moreover, the Excel's « Formula Bar» option is grayed out so that end users can't reactivate the formula bar. We also recommend that you disable access to the VBA editor.

15.2.2.7. Do not allow other instances of Excel when opening the protected workbook

All XLS Padlock Options > Security > Restrictions > Do not allow other instances of Excel when opening the protected workbook

The "Do not allow other instances of Excel when opening the protected workbook" option will check whether Excel is already running when the end user starts the compiled workbook. If this is the case, a message asking the user to close Excel is displayed.
Thus, this ensures the first running instance of Excel is the one associated to the protected workbook.

15.2.2.8. Only allow one instance of the protected workbook

All XLS Padlock Options > Security > Restrictions > Only allow one instance of the protected workbook

This "Only allow one instance of the protected workbook" option stops end users from running another instance of the protected workbook if one is already opened. If the end user tries to run the EXE again, the initial instance gets focused in the task bar. This inhibits end users from opening several instances of the same protected workbook simultaneously.

On Excel versions lower than 2013, a message box with "An instance of this application is already running" will be shown instead of the window being focused.

15.2.2.9. Remove "Enable fill handle and cell drag-and-drop" function

All XLS Padlock Options > Security > Restrictions > Remove "Enable fill handle and cell drag-and-drop" function

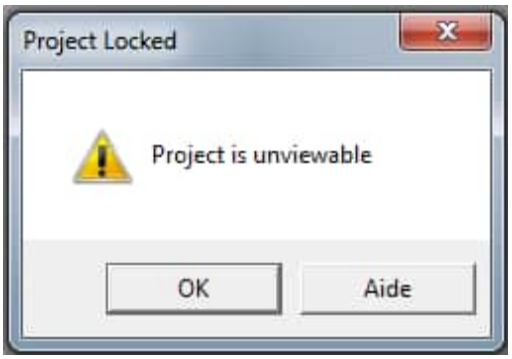
This "Remove "Enable fill handle and cell drag-and-drop" function" option stops end users from using the fill handle and cell drag-and-drop in the protected workbook. It is the same as in the Excel's general options (Review" – "Check accessibility" – "Options accessibility" – "Advanced" – "Enable fill handle and cell drag-and-drop") but it lets you override the local user's choice.

15.2.3. VBA Security

15.2.3.1. Lock VBA Project (simple VBA protection)

All XLS Padlock Options > Security > VBA Security > Lock VBA Project (simple VBA protection)

The "Lock VBA Project" option must be used if **you do not want your end users to access your VBA project**. This feature does not use password protection, it marks the VBA project as locked: it can not be viewed, accessed or modified.
If the end user tries to access a locked VBA project, the following error message is displayed (Project Locked – Project is unviewable):



If you use this option, we recommend you first to remove the password of your VBA project (if any). In fact, this password is actually useless if the project is in locked state. Moreover, Excel 2007 can crash if you use a long password while activating the Lock VBA Project option.

If you prefer to keep a password, try the option "Prevent access to VBA editor" instead.

Note that macros remain functional.

- This option is compatible with our VBA compiler, to increase the security of your VBA code.

15.2.3.2. Prevent access to VBA editor

All XLS Padlock Options > Security > VBA Security > Prevent access to VBA editor

"Prevent access to VBA editor" is an additional security measure that will **automatically close the VBA editor if the end user tries to open it**. A regular check is performed.
This option is compatible with VBA password protection and our VBA compiler.

15.2.4. Formulas and Passwords

15.2.4.1. Formula Protection Method

All XLS Padlock Options > Security > Formulas and Passwords > Formula Protection Method

When you configure XLS Padlock to protect your formulas, the software has two methods to do so. Either it modifies formulas directly in the source file of your workbook (this is the recommended method that works in the vast majority of cases). Either it controls Excel by automation.

If the recommended method fails, you can force XLS Padlock to use the method by automation: to do so, enable "**Use Excel automation for formula protection**". The latter will be automatically chosen if you protect a workbook in binary format (XLSB) or very large.

You can also consider to activate "Ignore errors when processing the workbook (internal protection)" but, in that case, be sure to test your compiled workbook.

15.2.4.2. Ignore errors when processing the workbook

All XLS Padlock Options > Security > Formulas and Passwords > Ignore errors when processing the workbook

The "Ignore errors when processing the workbook (internal protection)" option in XLS Padlock allows the protection process to continue even if errors are encountered while opening an Excel workbook. This feature is designed to streamline the protection workflow by suppressing error messages that could otherwise disrupt the securing of formulas and workbook structure.

When this option is enabled, XLS Padlock will not halt or display error prompts if it encounters issues during the workbook's opening phase.

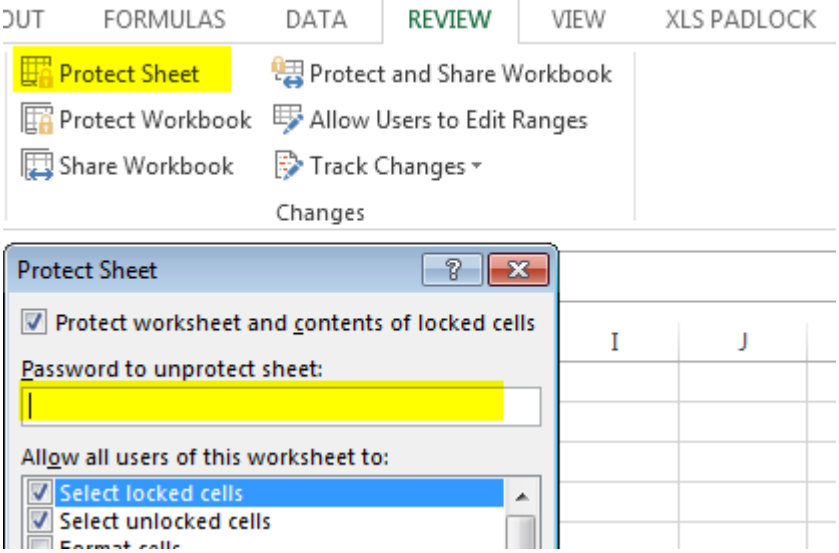
⚠ Enabling this option does not resolve any underlying issues within the workbook; it merely omits them from the protection operation. Post-protection, please test the workbook to ensure that all features are functioning as required and that the end-user experience is not negatively impacted.

Should errors manifest in the workbook after protection, try to enable the alternative option, "Use Excel automation for formula protection." This setting utilizes Excel's own mechanisms to safeguard formulas, which may mitigate issues that are not addressed by the internal protection method.

15.2.4.3. Worksheet password protection

All XLS Padlock Options > Security > Formulas and Passwords > Worksheet password protection

The "Worksheet password protection" field should be filled in **if you use Excel sheet protection**:



In case of Excel automation, XLS Padlock has to temporarily unprotect your sheets in order to apply formula protection, before re-protection. For this operation to be successful, in XLS Padlock, you must enter the password(s) you chose to unprotect sheets in Excel.

💡 Note that XLS Padlock will automatically apply the same sheet protection options as you defined in the original workbook.

In case you have different passwords for your sheets, you must pass them to XLS Padlock thanks to the following JSON format.

```
{"worksheet 1 name": "Password1","worksheet 2 name": "Password2","worksheet 3 name": "Password3"…}
```

Example of 2 passwords to be entered in the field:

```
{"worksheet1": "Hello World","worksheet2": "Password2"}
```

15.2.4.4. Protect the workbook with the following password, but do not ask users for it

All XLS Padlock Options > Security > Formulas and Passwords > Protect the workbook with the following password, but do not ask users for it

Excel offers password protection for your workbook. It encrypts the source file and asks end users for a password in order to open the workbook.
XLS Padlock can use this password protection internally: it saves the Excel file with the password you provide. At runtime, the EXE file tells Excel the password when opening the compiled workbook. Your end users are not asked for the password at all.

Thus, you have increased protection: Excel password protection combined with security features offered by XLS Padlock.

You can generate a random password with the **Generate** button.

Leave the field empty to disable password protection, or if you already use Excel password protection and want to ask end users for the password.

Do not change the password once provided, otherwise end users won't be able to open previous secure save file.

15.2.5. Excel add-ins

15.2.5.1. Do not disable the following COM add-ins (enter ProgID):

All XLS Padlock Options > Security > Excel add-ins > Do not disable the following COM add-ins (enter ProgID):

By default, any existing Excel COM add-in is disabled when loading a secured workbook.
However, some useful add-ins like Microsoft PowerPivot may be required by your workbooks. In that situation, enter all progIDs of the different add-ins you want to keep in the field separated by semi colons:

Company.Addin1;Company2.Addin2.File;...

For instance, for Microsoft PowerPivot, the ProgID are: Microsoft.AnalysisServices.Modeler.FieldList and

PowerPivotExcelClientAddIn.NativeEntry.1

Learn more at http://support.microsoft.com/kb/291392/en-us

15.2.5.2. Allow Excel common add-ins such as Analysis Toolpak, Solver, Euro Currency Tools...

All XLS Padlock Options > Security > Excel add-ins > Allow Excel common add-ins such as Analysis Toolpak, Solver, Euro Currency Tools...

Add-ins such as Analysis Toolpak, Solver, Euro Currency Tools... are provided with Excel and can be used by your workbooks. In that situation, you should enable the option so that the secure application loads them successfully at startup.

15.2.6. Excel versions required for your workbook

All XLS Padlock Options > Security > Excel versions required for your workbook

If your workbook has been created with a recent version of Excel, you can configure your application to check whether the Excel version of the end user is recent enough.

In the same way, you can set a maximum version to check (for instance, if your workbook has not been tested with a more recent version of Excel).

Enter the version number in the fields:

- 9: Excel 2000 (version 9.0) - Office 2000
- 10: Excel 2002 (version 10.0) - Office XP
- 11: Excel 2003 (version 11.0) - Office 2003
- 12: Excel 2007 (version 12.0) - Office 2007
- 14: Excel 2010 (version 14.0) - Office 2010
- 15: Excel 2013 (version 15.0) - Office 2013
- 16: Excel 2019 and 2016 (version 16.0) – Office 2019 and 2016

Excel Custom UI

15.2.7. Excel Custom UI

As explained in Excel's documentation, you can customize the Excel UI by placing an .officeUI file in a specific location.

XLS Padlock allows you to specify your own Excel.officeUI for the secure application. It will automatically replace the user's default UI (if available).

You have just to enter the full path to the Excel.officeUI file you want to use. XLS Padlock will compile it into the final EXE, so no need to deploy the Excel.officeUI file separately.

- For further information about creating custom UI XML files, please refer to the documentation of Excel:

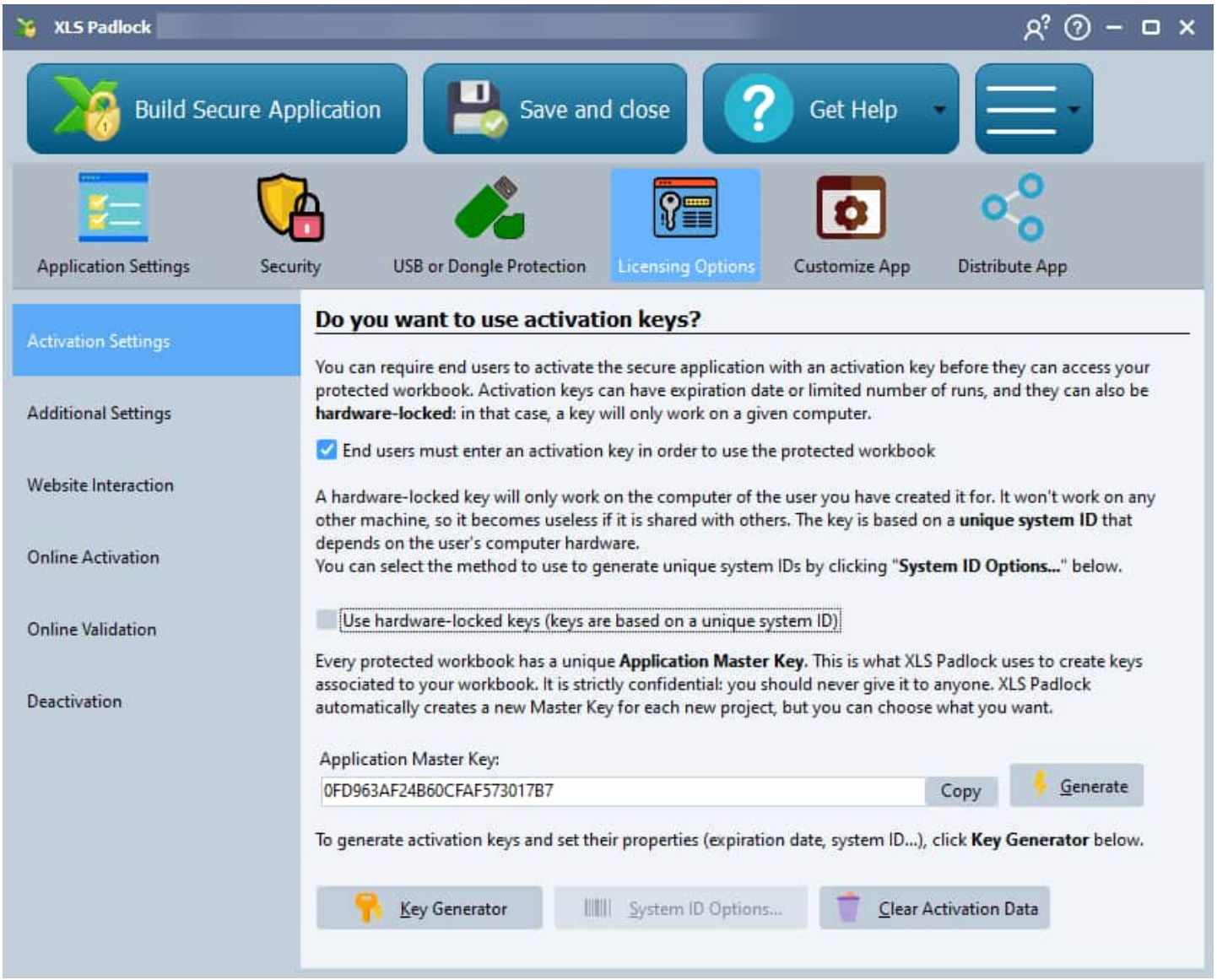
https://msdn.microsoft.com/en-us//library/office/ee704589(v=office.14).aspx

15.3. Licensing Options

Being an important feature of XLS Padlock, activation keys offer a **powerful licensing system for Excel workbooks**: you can require end users to activate the secure application with an activation key before they can access your protected workbook. Activation keys can have expiration date or limited number of runs, and they can also be hardware-locked: in that situation, a key will only work on a given computer.

In addition to XLS Padlock's built-in "Key Generator", we provide a stand-alone key generator (that can output thousands of keys at once) and a key generator SDK so that you can make keys directly on your server/website.

Finally, it is possible to work with online activation and validation features: they allow a remote control on how your users access your workbook.

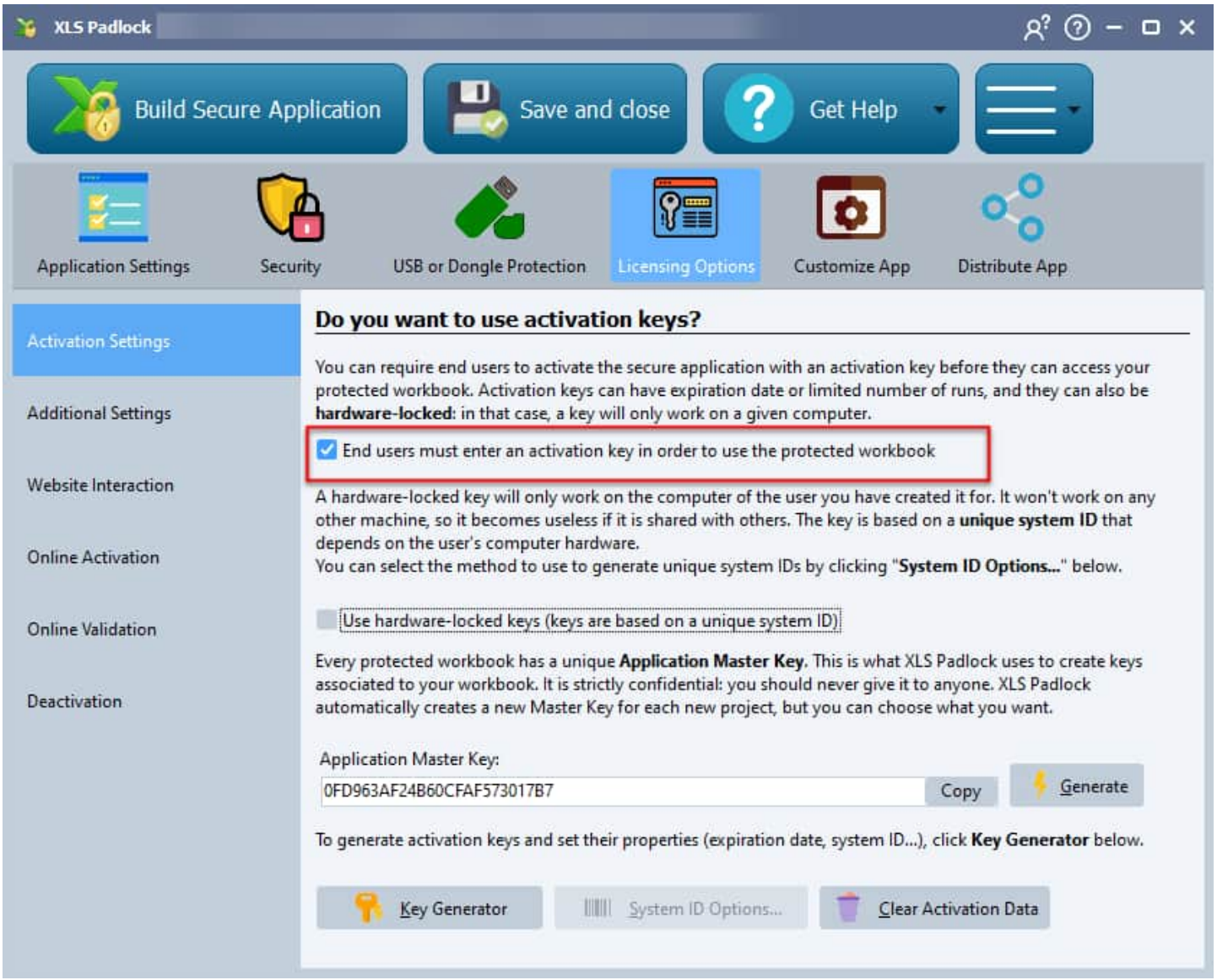


To get started with activation keys, please take a look at the "How to set up activation keys" tutorial.

15.3.1. Activation Settings

15.3.1.1. "End users must enter an activation key in order to use the protected workbook"

If you want to use activation keys to restrict access to your application, enable the following option: **"End users must enter an activation key in order to use the protected workbook**". That's all.

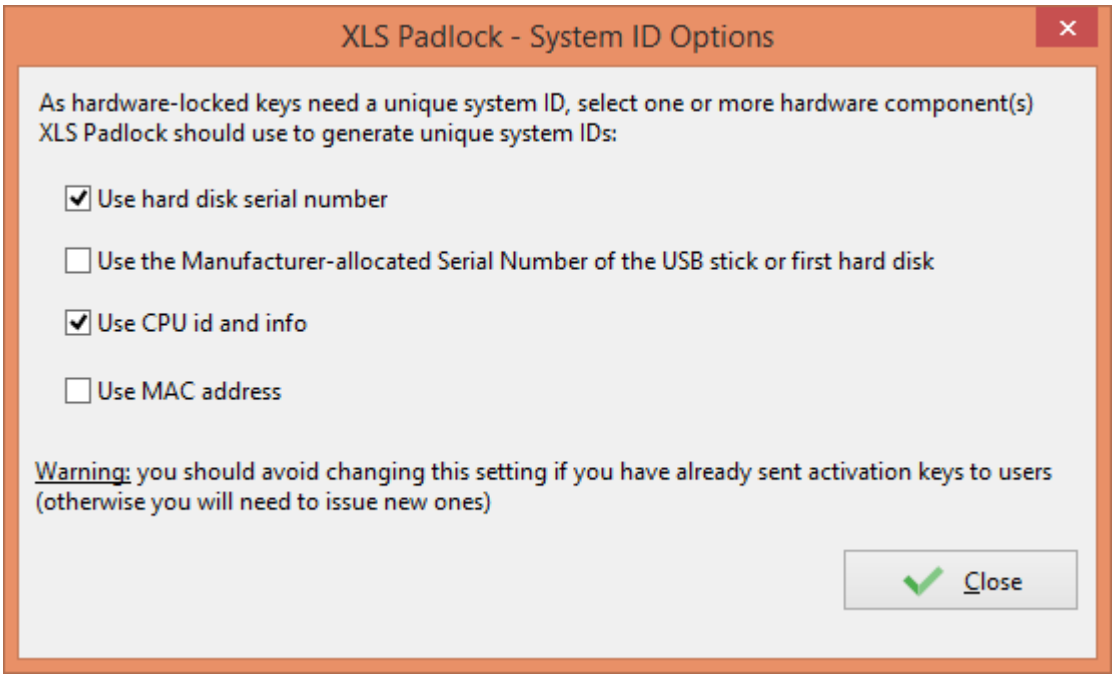


To get started with activation keys, please take a look at the "How to set up activation keys" tutorial.

15.3.1.2. "Use hardware-locked keys"

A hardware-locked key will **only work on the computer of the user** you have created it for. It won't work on any other machine, so it becomes useless if it is shared with others. The key is based on a **unique system ID** that depends on the user's computer hardware.

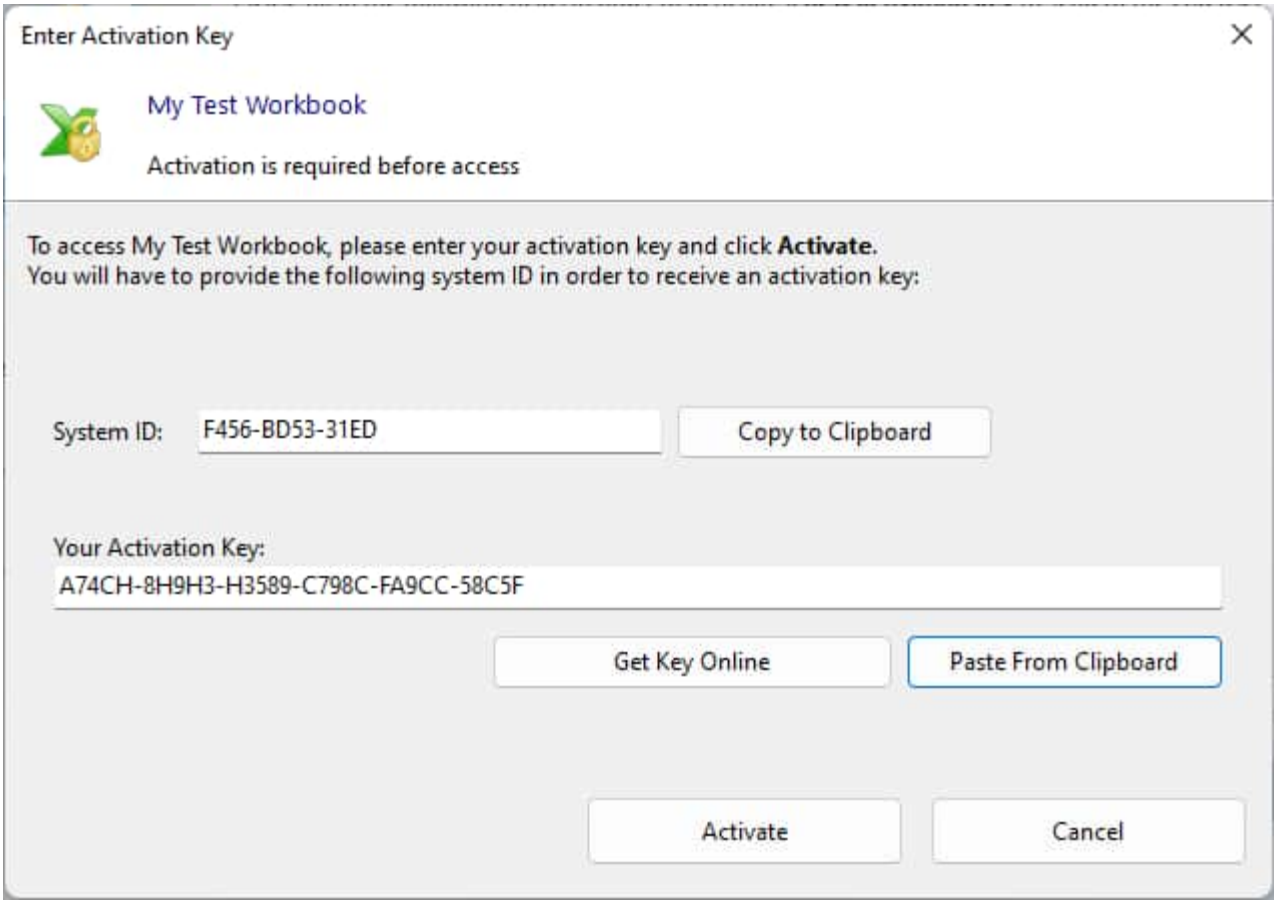
You can select which component(s) the application should use to generate unique system IDs (serial numbers, CPU, MAC address) by clicking "**System ID Options**...":



Notes:

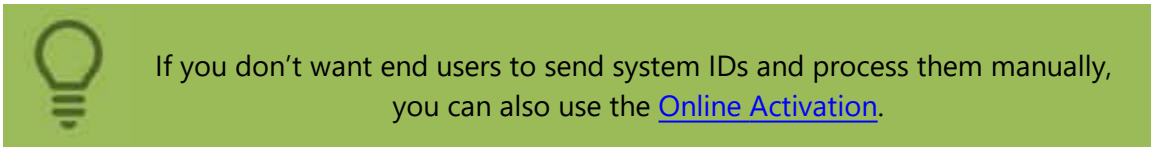
- if the CPU used has no serial ID built-in, CPU's general info is then used. For identical CPU and hardware details, you can get the same ID. That's why it would be better to combine several hardware options.
- With the "Use Mac address" option, if people have different means to connect to the Internet (wi-fi, 4G…), their Mac address can change.

When your customers run your application for the first time, they will get this window:



The customer has to copy their system ID and send it to you.

Then, you can create their key thanks to the Key Generator in XLS Padlock.



- To get started with hardware-locked activation keys, please take a look at the "How to set up hardware-locked activation keys" tutorial.

15.3.1.3. Application Master Key

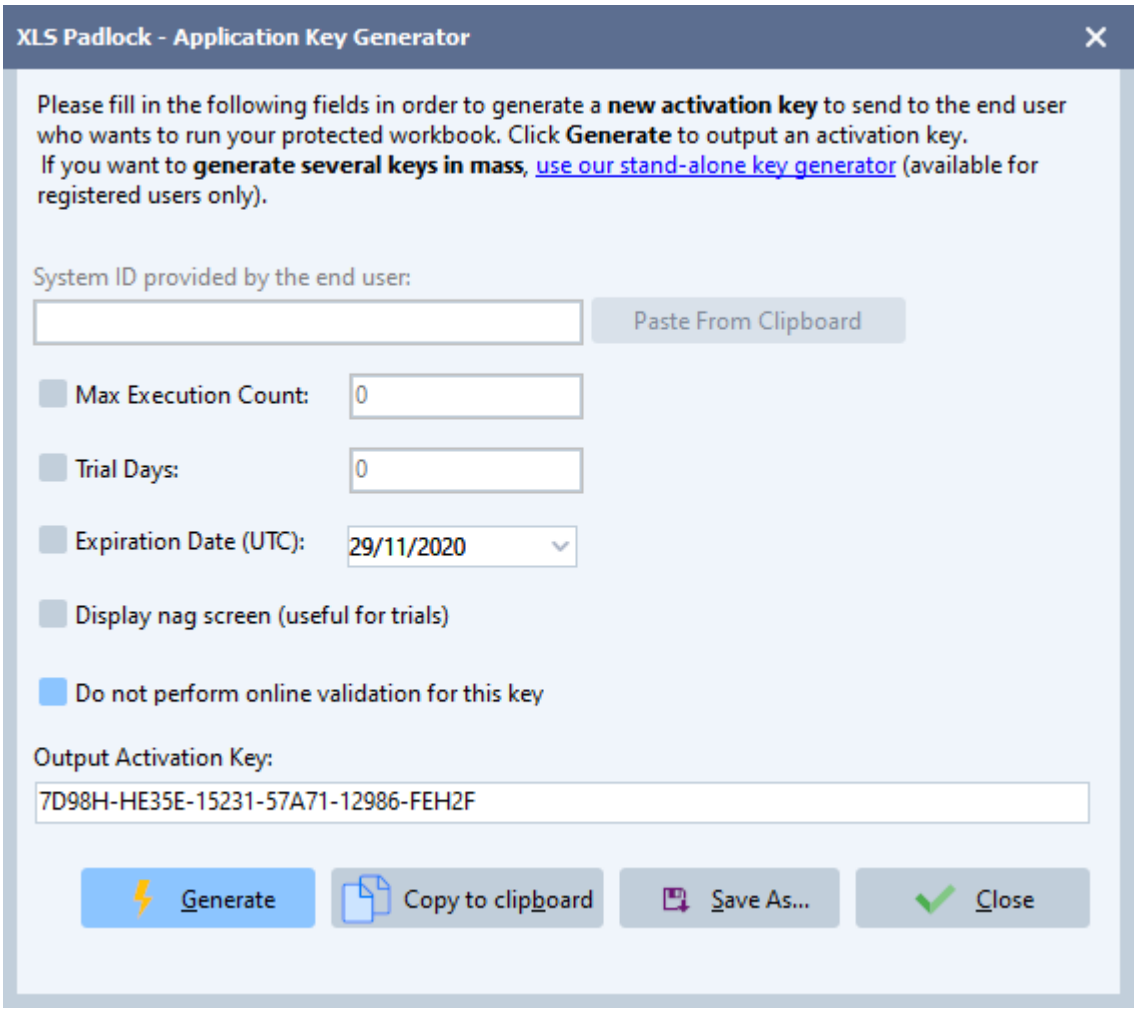
All XLS Padlock Options > Licensing Options > Activation Settings > Application Master Key

Every protected workbook has a unique **Application Master Key**. This is what XLS Padlock uses to create keys associated to your workbook. It is **strictly confidential:** you should never give it to anyone. XLS Padlock automatically creates a new Master Key for each new project, but you can choose what you want.

You can click **Copy** to copy the key to the clipboard (useful for the stand-alone key generator).

15.3.2. Key Generator (portable and remote server versions)

- In XLS Padlock, you can generate activation keys for your workbook application directly with "Key Generator". A dialog box lets then you create these activation keys:



→ *For simple activation keys*, you just click the **Generate** button. A key is instantly created.

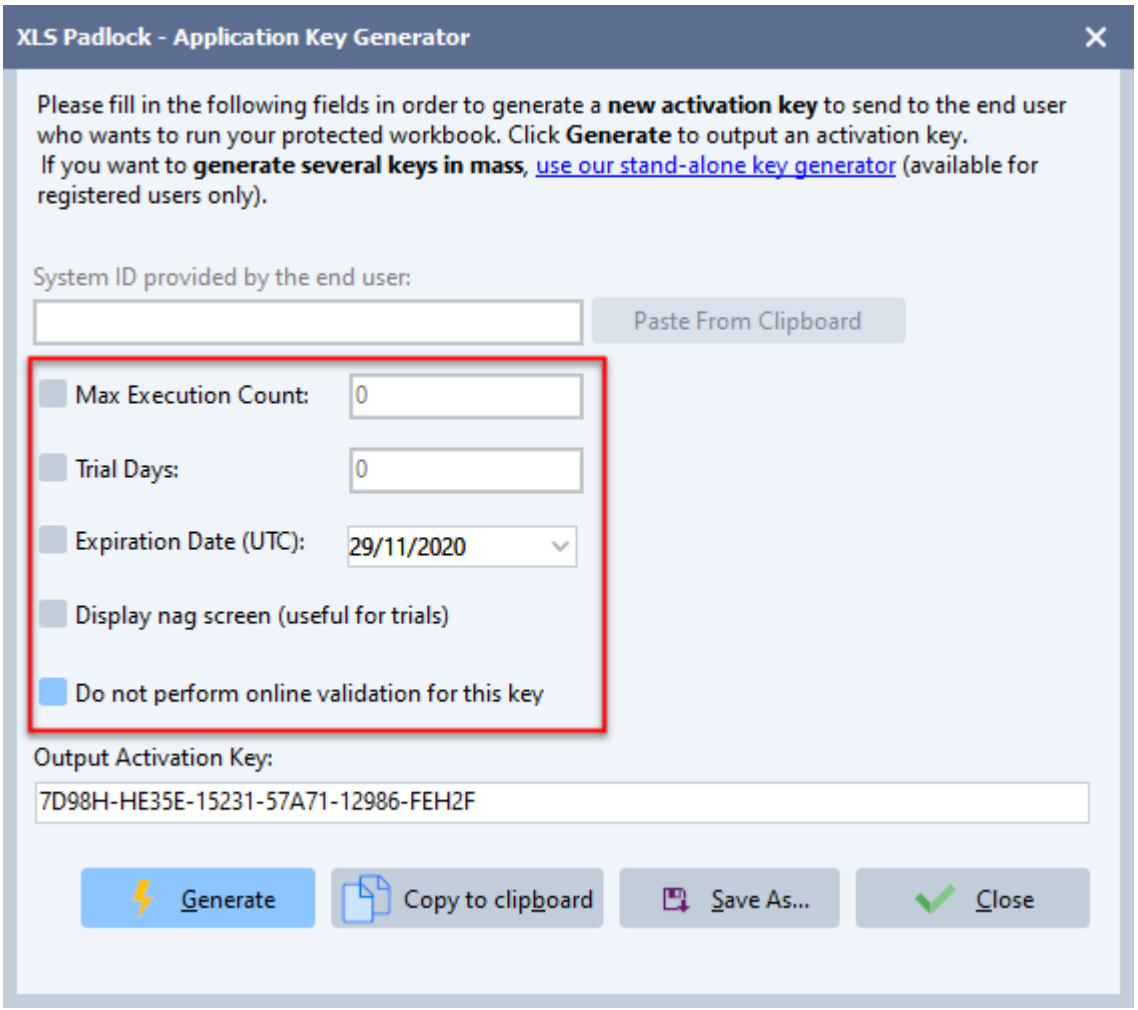
→ *For hardware-locked keys*, you must enter the system ID your user has sent to you. Click the **Generate** button. A key is instantly created. The key is associated to the system ID you entered, so that it will activate your application only on the computer with this system ID.

You can copy the key to the clipboard in order to send it to your customer, and/or save it as a .txt file.

The key generator will output activation keys for your own workbook only (be sure to have opened your source workbook in Excel). Keys generated will not work with other workbooks that do not share the same Application Master Key.

15.3.2.1. Set restrictions on keys (expiration date, max execution count...)

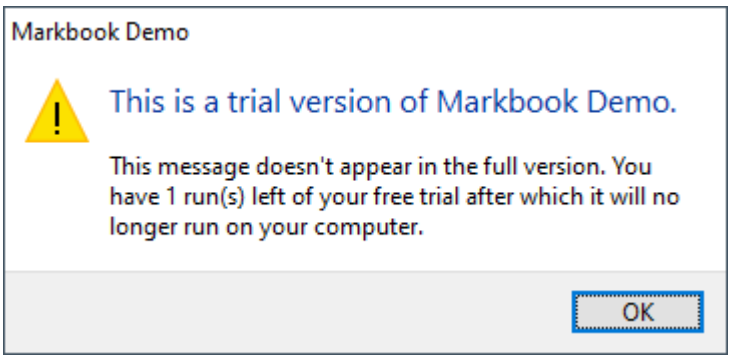
When you generate activation keys, you can set restrictions thanks to the following options:



You can limit the number of times your application can be run: tick "**Max Execution Count**" and enter the required number.

You can also make the key expire after a given number of days (tick "**Trial Days**" and enter the required number); alternatively, after a given date (tick "**Expiration Date**" and enter the required date).

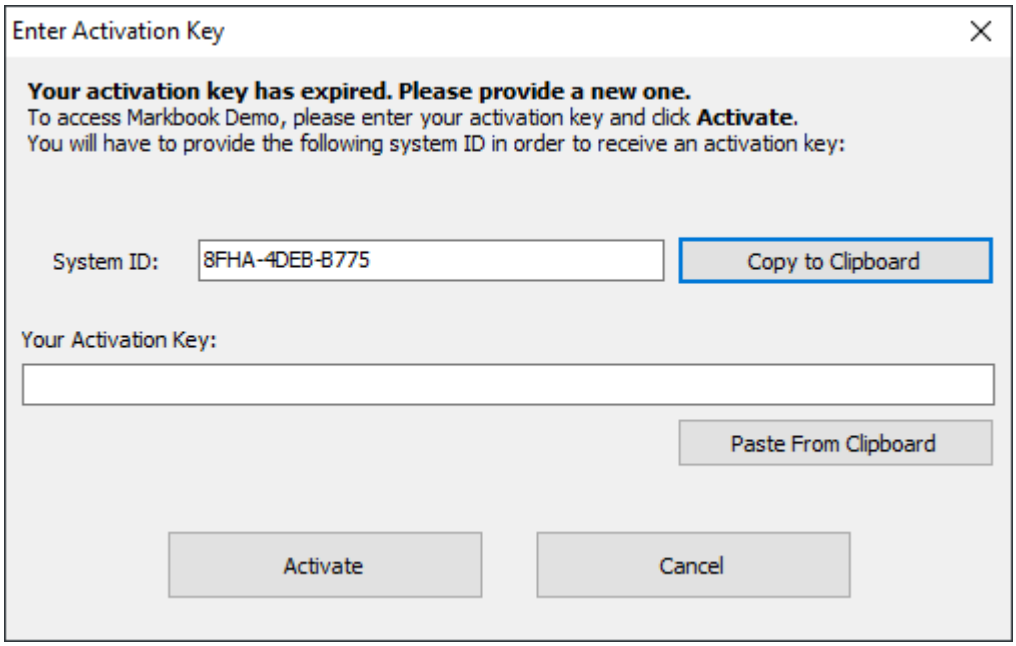
If you want to display a reminder dialog at startup, also known as nag screen, tick "**Display Nag Screen**" in the key generator (useful for trials because it shows the remaining number of days or runs to end users):



Finally, if you use online validation and you want to let users bypass it (in case of offline activation for instance), you can tick "**Do not perform online validation for this key**". This is recommended only if you also use hardware-locking for keys.

→ After the given date, number of executions or days, when running your application, your customers will get

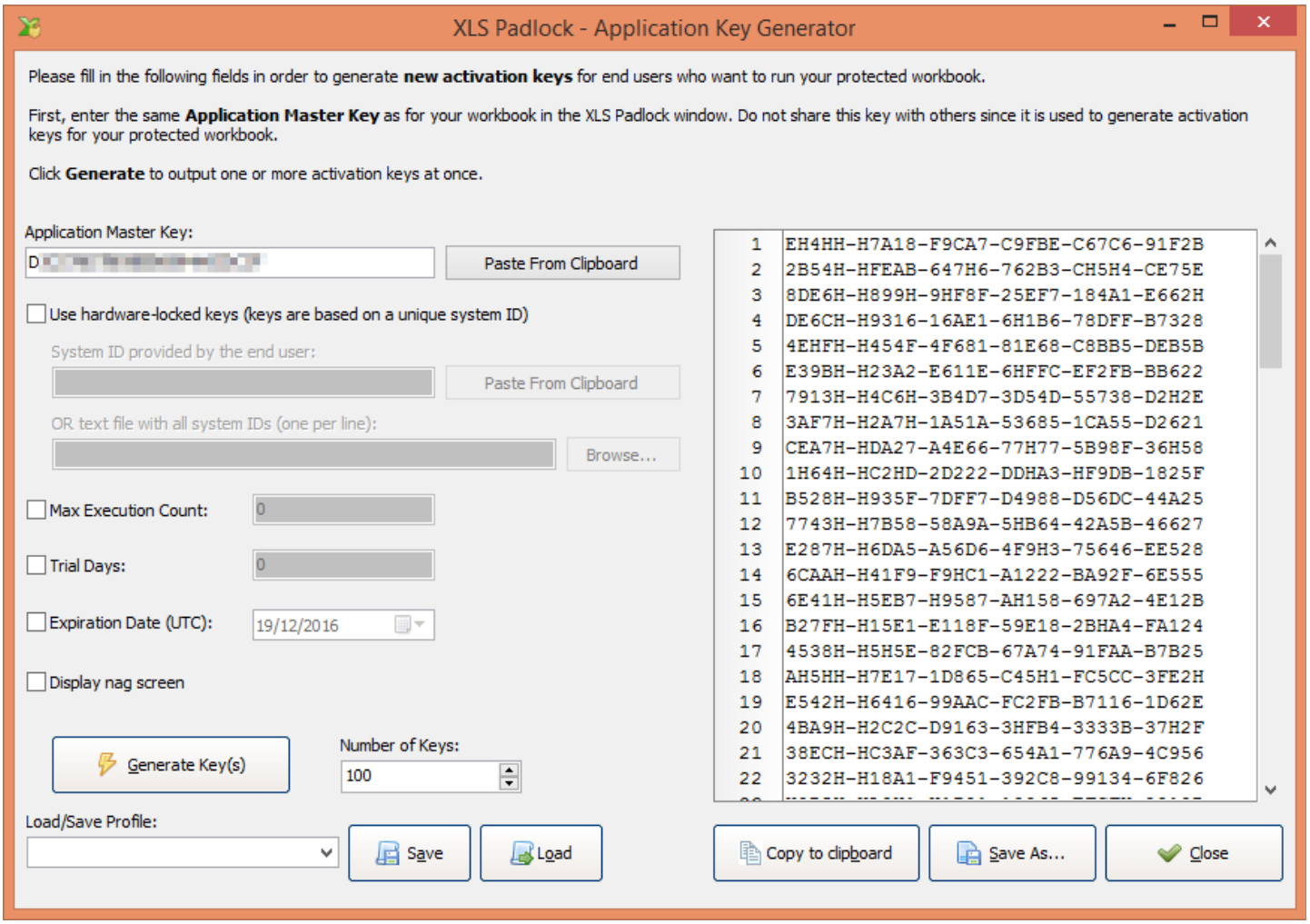
this window requiring a new key to activate your application:



15.3.2.2. Stand-alone key generator without running XLS Padlock

All XLS Padlock Options > Licensing Options > Key Generator (portable and remote server versions) > Stand-alone key generator without running XLS Padlock

A stand-alone key generator for Excel workbook applications is also available for registered customers of XLS Padlock. With it, you can create activation keys for your protected workbooks without having to run Excel and XLS Padlock. Settings can be saved to profile files. Download it at https://www.xlspadlock.com/account/downloads



15.3.2.3. Key generator SDK

All XLS Padlock Options > Licensing Options > Key Generator (portable and remote server versions) > Key generator SDK

We also provide a free online web application and associated PHP code for generating activation keys for your workbooks. This key generator SDK can be used on your own website, so that you can generate and deliver activation keys automatically.

Finally, for automated key delivery, you should consider the Online Activation feature.

Please go to https://www.xlspadlock.com/account/downloads for further information.

15.3.2.4. Clear Activation Data

All XLS Padlock Options > Licensing Options > Key Generator (portable and remote server versions) > Clear Activation Data

If you test activation keys on your local computer (for instance, key expiration), the "**Clear Activation Data**" button lets you reset activation data stored on your local computer and test new keys.

15.3.2.5. Change activation key once already activated

All XLS Padlock Options > Licensing Options > Key Generator (portable and remote server versions) > Change activation key once already activated

XLS Padlock allows your customers to change their activation key via the EXE's command line by entering the following switch: `-enterkey`

Running `MYAPP.EXE -enterkey` will ask the end user for a new activation key.

This is useful for instance to replace an old key that is going to expire. Of course, entering the same key or a key that expired will be refused (to prevent resetting trial periods).

15.3.3. Additional Settings

15.3.3.1. "Prompt the end user for the activation key each time"

All XLS Padlock Options > Licensing Options > Additional Settings > "Prompt the end user for the activation key each time"

By enabling "Prompt the end user for the activation key each time", your users will have to enter the activation key each time they open your application. It is not recommended, because it is fastidious for your customer, and you cannot set expiration features for the key.

15.3.3.2. "Do not store activation info in the registry, but in an external file (portable mode)"

All XLS Padlock Options > Licensing Options > Additional Settings > "Do not store activation info in the registry, but in an external file (portable mode)"

By default, your application will store activation data in the Windows registry. For users who do not want to modify the registry or for those who don't have enough user rights, you can enable the option "**Do not store activation info in the registry, but in an external file (portable mode)**".

With this option, the application stores **activation data in an external file**. This file is placed in the same folder

as the application EXE file, and it has the same filename as the EXE, with ".lic" as extension. This file is by default hidden by Windows.

Be careful that customers should not delete this file, otherwise their licensing info will be lost.

If you create keys that expire, you should leave this option turned off. Otherwise, deleting the ".lic" file will reset trial date

15.3.3.3. Disable the "Enter Activation Key" button on Welcome screen

All XLS Padlock Options > Licensing Options > Additional Settings > Disable the "Enter Activation Key" button on Welcome screen

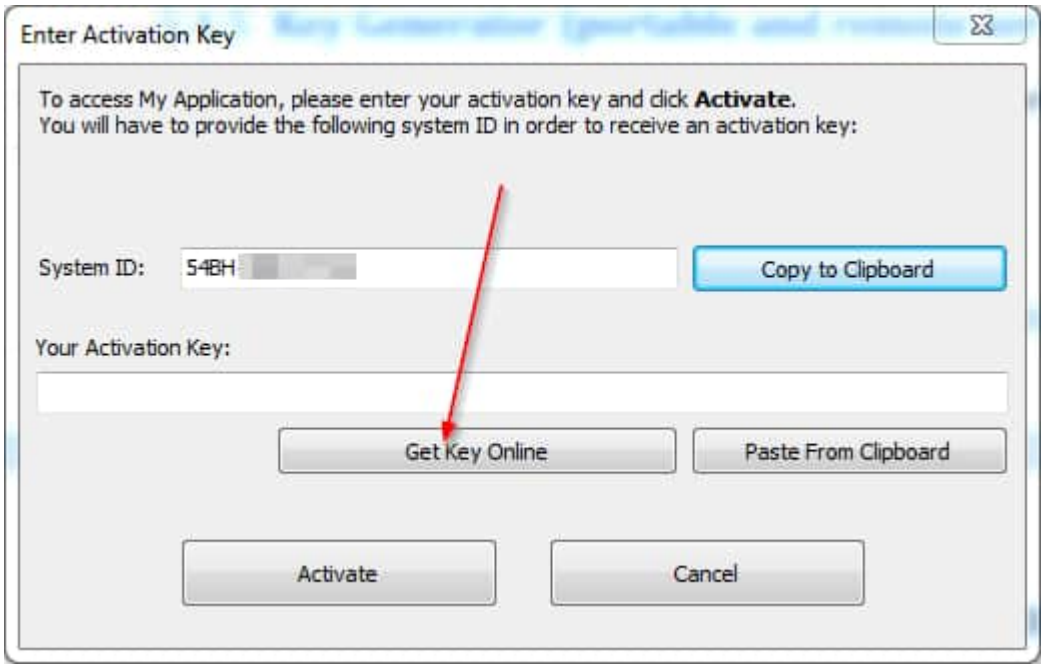
"Disable the "Enter Activation Key" button on Welcome screen" will remove the possibility for your end users to enter an activation key when they open your Excel workbook application. It is not recommended to use that option though.

15.3.4. Website Interaction

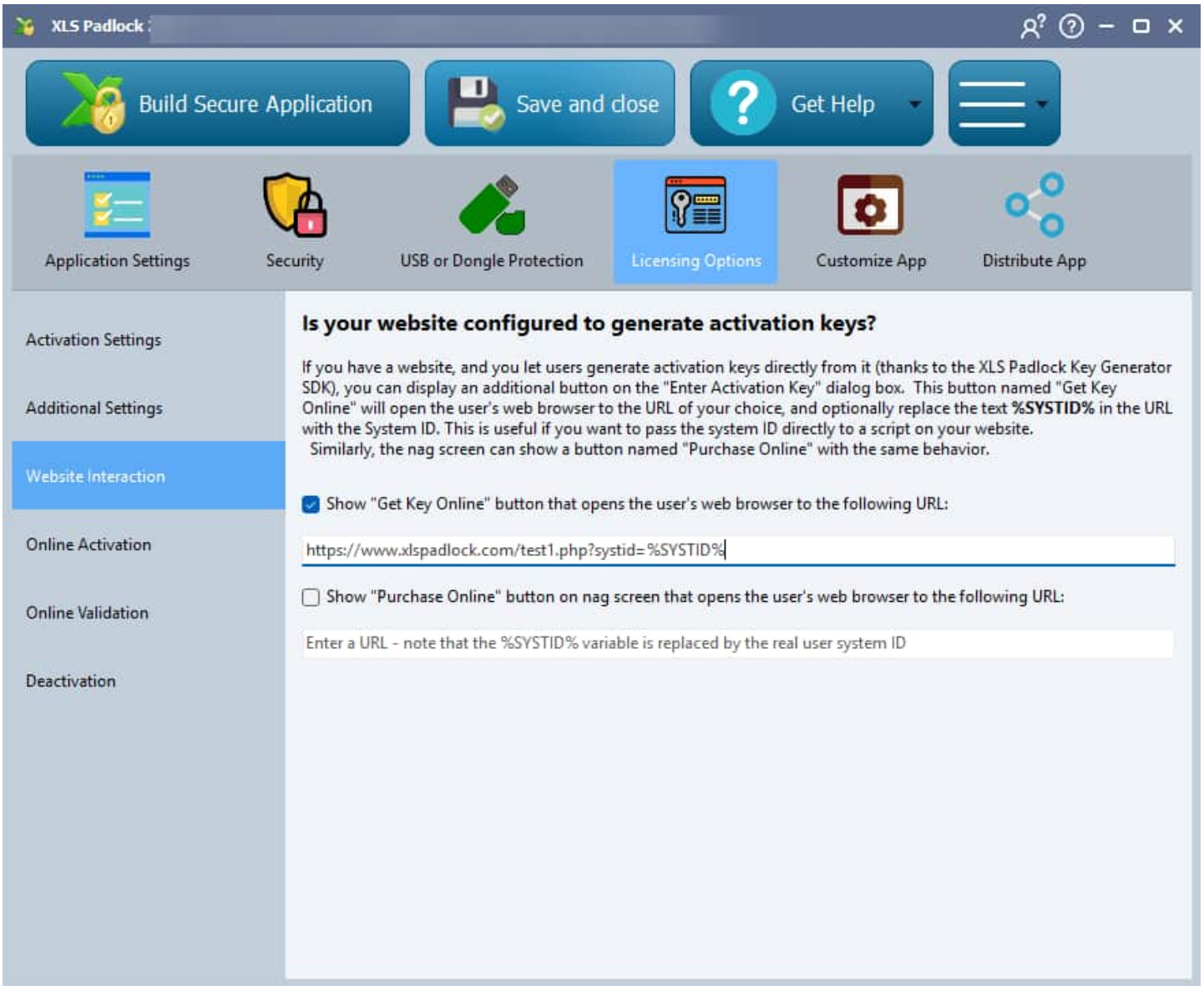
15.3.4.1. Show "Get Key Online" button that opens the user's web browser to the following URL:

All XLS Padlock Options > Licensing Options > Website Interaction > Show "Get Key Online" button that opens the user's web browser to the following URL:

If you have a website, and you let users generate activation keys directly from it, you can display an additional button on the "Enter Activation Key" dialog box:



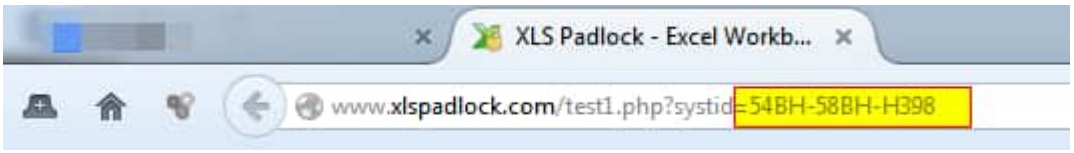
This button named "Get Key Online" will open the user's web browser to the URL of your choice, and optionally replace the text %SYSTID% in the URL with the System ID. This is useful if you want to pass the system ID directly to a script on your website.



For instance, we entered:

```
https://www.xlspadlock.com/test1.php?systid=%SYSTID%
```

Look at the **%SYSTID%** text at the end. It will be replaced by the system ID when the web browser is opened:



15.3.4.2. Show "Purchase Online" button on nag screen that opens the user's web browser to the following URL:

All XLS Padlock Options > Licensing Options > Website Interaction > Show "Purchase Online" button on nag screen that opens the user's web browser to the following URL:

If you offer trial versions of your workbook, they can display a nag screen to remind the end user that they must purchase the full version. The nag screen can optionally show a "**Purchase Online**" button that will open the user's web browser to the URL of your choice. The goal is to redirect end users to your store or purchase web page.

As in the previous topic, it is possible to use the **%SYSTID%** text in the URL to insert the System ID.

15.3.5. Online Activation

15.3.5.1. Base Activation URL

All XLS Padlock Options > Licensing Options > Online Activation > Base Activation URL

Provide the URL to the activation kit installed in your server. For instance, if you installed the activation kit in a sub folder named "activation", the URL will be http://www.yourdomain.com/activation/getactivation/

Secure connection thanks to TLS/SSL is supported by XLS Padlock applications. You can use URLs that begin with **https://**

Leave the field blank if you do not want to use online activation.

- Online Activation Security Private Key

15.3.5.2. Security Private Key

The "Security Private Key" must be a unique private key to be used to identify your application with the server. Please refer to the activation kit's documentation.

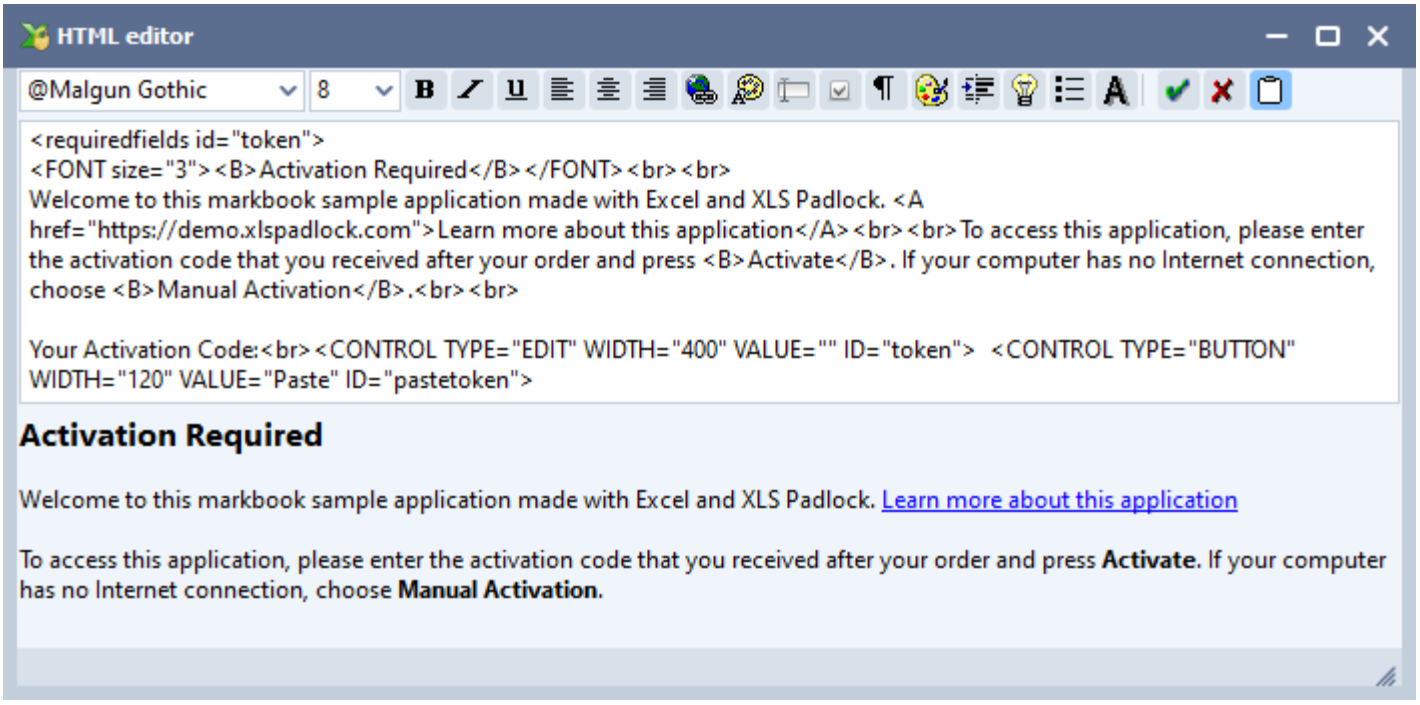
XLS Padlock should have generated a default one for you, but you can enter anything you want.

Base Activation URL Registration Form Editor

15.3.5.3. Registration Form Editor

The Registration Form Editor lets you customize the text that appears on the activation dialog box. The text supports basic HTML display so you can use HTML tags such as **<B></B>** and so on.
For further information about this functionality, please refer to the online activation's guide.

XLS Padlock shows a basic HTML editor and a preview of the dialog's text.



Two specific buttons let you add custom fields if you want to ask end users for further information and have your application send the latter to your web server.

You can add a Paste button to paste code into the token field:

```
<CONTROL TYPE="BUTTON" WIDTH="120" VALUE="Paste" ID="pastetoken">
```

You can tell XLS Padlock which fields are **mandatory** by indicating their ID in a special HTML tag named

requiredfields. For instance, the following HTML tag indicates that fields "token" and "name" are mandatory:

```
<requiredfields id="token,name">
```

Example:

```
<requiredfields id="token">
<FONT size="3"><B>Activation Required</B></FONT><br><br>
Welcome to this application.<br><br>To access this application, please enter the
activation code that you received after your order and press <B>Activate</B>. If
your computer has no Internet connection, choose <B>Manual Activation</B>.<br><br>

Your Activation Code:<br><CONTROL TYPE="EDIT" WIDTH="400" VALUE="" ID="token">
<CONTROL TYPE="BUTTON" WIDTH="120" VALUE="Paste" ID="pastetoken">
```

Security Private Key Allow Manual Activation if No Internet Connection (recommended)

15.3.5.4. Allow Manual Activation if No Internet Connection (recommended)

All XLS Padlock Options > Licensing Options > Online Activation > Allow Manual Activation if No Internet Connection (recommended)

Finally, some users may not have an active Internet connection. If you wish to allow manual registration, enable the "**Allow Manual Activation if No Internet Connection**" option.

This manual method works exactly like for registration keys. In that situation, you must be prepared to accept requests from these users without any Internet connection.

If you accept manual activations, be sure to generate activation keys that do not perform online validation (option in the key generator). Otherwise, validation will fail!

Registration Form Editor

15.3.6. Online Validation

Online validation gives you **remote control options** over activation keys: if you have set up the XLS Padlock Activation Kit or the XLS Padlock WooCommerce Integration Kit or the FS subscription kit on your web server, the application once activated **can regularly check the validity of the activation** through the Internet.

This feature is only available if online activation is also used.

Possible uses:

- you sell Excel workbooks, and you want to block access to them if end users ask for refunds.
- you want to control who can access Excel workbooks regularly.

Notes:

- Online validation must be preceded by online activation. The token used for activation is also used for validation to identify the customer.
- Activation key is not sent to the server.
- An Internet connection is mandatory for validation.

Base Validation URL

15.3.6.1. Base Validation URL

All XLS Padlock Options > Licensing Options > Online Validation > Base Validation URL

You must provide the URL to the XLS Padlock Activation Kit or the XLS Padlock WooCommerce Integration Kit or the FS subscription kit on your web server. For instance, if you installed the activation kit in a sub folder named "activation", the URL shall be https://www.yourdomain.com/activation/dovalidation/

Secure connection thanks to TLS/SSL is supported by XLS Padlock applications.
You can use URLs that begin with **https://**

Leave the field blank if you do not want to use online validation.

End users must have **an Internet connection for the validation process**.
Otherwise, the application will consider that the validation has failed.

Online Validation Validate the activation state

15.3.6.2. Validate the activation state

Choose when the application should perform a validation: at every startup, randomly, every X days or every X times. You must specify the value for X if required.

⚠ Warning: Once the application is activated, an initial validation of the activation status will be required upon the next launch. Following this initial check, the application will adhere to the selected verification frequency.

Base Validation URL If the activation cannot be validated

15.3.6.3. If the activation cannot be validated

Define what the application should do if the online validation fails:

- Exit the application
- Blacklist activation key: this causes the activation key to expire. End users are prompted to enter a new activation key. This can be the previous one: if the user enters the same activation key as before, validation is required again.
- Do nothing. Use the VBA extension feature of XLS Padlock to retrieve the state of validation: refer to the demo VBA code available.

Validate the activation state Display this error message if validation fails

15.3.6.4. Display this error message if validation fails

If validation fails, the custom error message is displayed as defined in the "Display this error message if validation fails" field.

 This message is optional but recommended.

Its goal is to inform your users that validation failed and what will happen (should they contact you or retry?).

If validation fails, the end user is also given the opportunity to enter another activation key.

- If the activation cannot be validated Always skip validation if no Internet connection is available

15.3.6.5. Always skip validation if no Internet connection is available

All XLS Padlock Options > Licensing Options > Online Validation > Always skip validation if no Internet connection is available

If this option "Always skip validation if no Internet connection is available" is enabled, the secure applications checks if the computer is connected to the Internet before trying to perform validation. If not, it will skip the validation step and try again next time it is run.

This is not recommended.

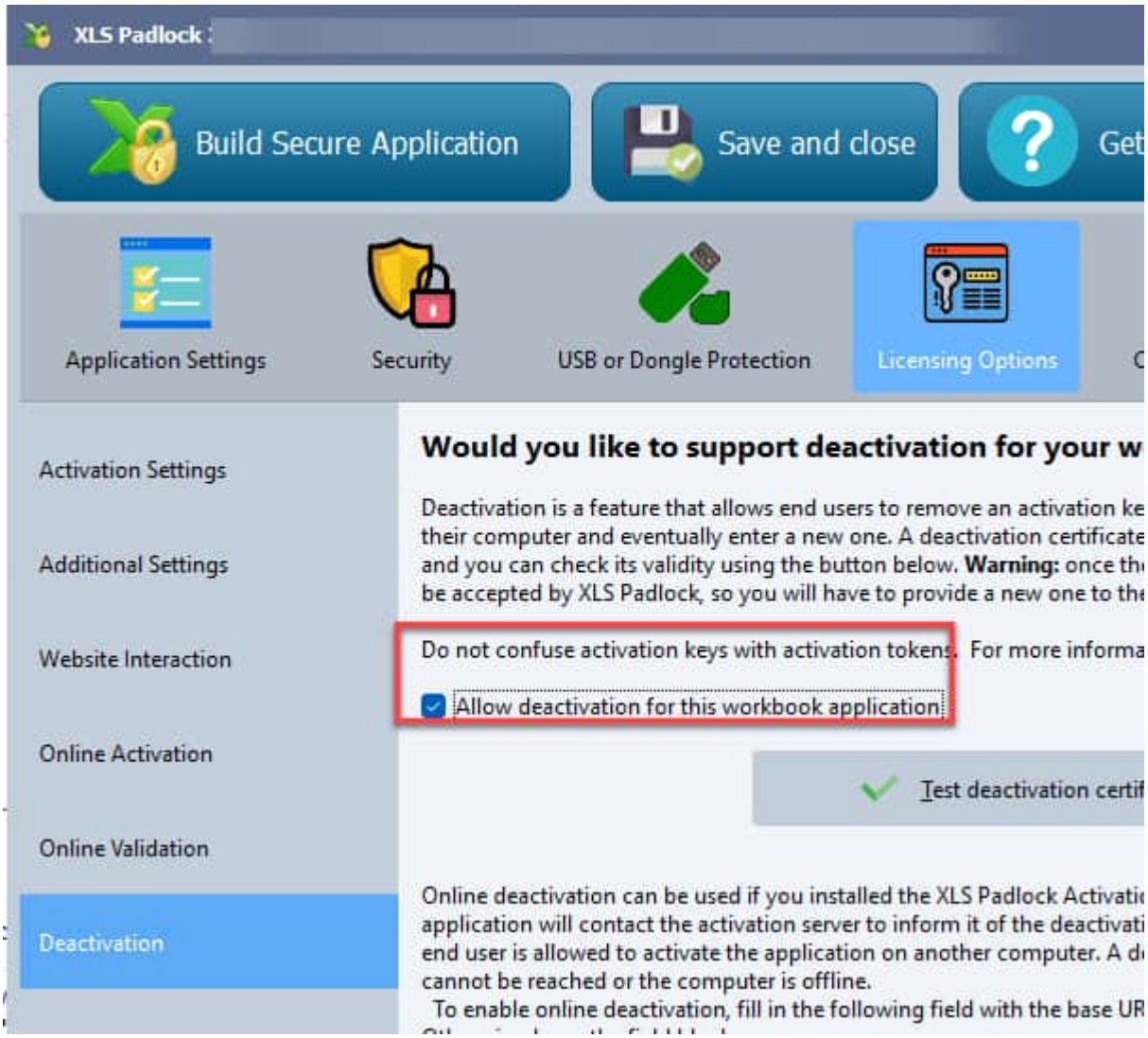
- Display this error message if validation fails

15.3.7. Deactivation

15.3.7.1. Allow deactivation for this workbook application

All XLS Padlock Options > Licensing Options > Deactivation > Allow deactivation for this workbook application

This control enables you to allow or disallow deactivation for your workbook application. By checking the 'Allow deactivation for this workbook application' box, you give users the ability to deregister their application.



Deactivation Test Deactivation Certificate

15.3.7.2. Test Deactivation Certificate

All XLS Padlock Options > Licensing Options > Deactivation > Test Deactivation Certificate

This button is used to test the deactivation certificate received from a user. Upon verification, it reveals the deactivation date and the user's computer identifier, assisting you in managing new activations or deactivations.

If the key being deactivated had an expiration date or a limited number of runs, XLS Padlock will display the remaining number of days or runs:

This can, for example, be useful in order to refund my customer for the unused portion of that license.

If the remaining number of days or runs is zero, then it is not displayed.

Allow deactivation for this workbook application Base Deactivation URL

15.3.7.3. Base Deactivation URL

All XLS Padlock Options > Licensing Options > Deactivation > Base Deactivation URL

To utilize online deactivation, you must provide the URL to the XLS Padlock Activation Kit or the XLS Padlock WooCommerce Integration Kit on your web server. For instance, if you installed the activation kit in a sub folder named “activation”, the URL shall be https://www.yourdomain.com/activation/dodeactivation

Secure connection thanks to TLS/SSL is supported by XLS Padlock applications.
You can use URLs that begin with **https://**

If you prefer to manage deactivation manually, leave this field blank.

- Test Deactivation Certificate Hide Manual Deactivation Button (in that case, it is shown if automated deactivation fails)

15.3.7.4. Hide Manual Deactivation Button (in that case, it is shown if automated deactivation fails)

All XLS Padlock Options > Licensing Options > Deactivation > Hide Manual Deactivation Button (in that case, it is shown if automated deactivation fails)

By default, XLS Padlock enables manual deactivation, particularly for situations where some users might not have an active Internet connection.

If you prefer to encourage online deactivation by the end user, you can configure your application to **hide the Manual Deactivation button**.

Please note that the application will automatically permit Manual Deactivation if online deactivation is unsuccessful (e.g., if the deactivation server is inaccessible).

This is not recommended.

- Base Deactivation URL

15.4. Customize App

15.4.1. Splash screen

During its initialization state, the application can show a **splash screen** (an image shortly displayed at startup). Splash screen is useful if you wish to display your company's logo or something related to your application's contents.

Splash Screen Settings

This can be any image in JPEG, PNG, BMP, SVG or GIF formats.

You can set how long you want this splash screen to display. Note that if the user clicks it, the screen closes immediately, whatever the time you selected. It is possible to disable that behavior in the Advanced Options (Allow the user to close the splash screen by clicking it).

XLS Padlock can display non-rectangular alpha-blended (semi-transparent) splash screens if you use 32-bit PNG files. This can give a unique look to your workbook application.

- Do not display the "Loading workbook" dialog box in Excel

15.4.1.1. Do not display the "Loading workbook" dialog box in Excel

All XLS Padlock Options > Customize App > Splash screen > Do not display the "Loading workbook" dialog box in Excel

When Excel is started, a "**Loading workbook, please wait...**" progression dialog box is displayed. Its duration can be tweaked in the Advanced Options.

You can enable the "**Do not display the "Loading workbook" dialog box in Excel**" option if you want to hide this dialog box.

It is also possible to hide this message early with VBA code.

- Excel Main Window Display at Startup

15.4.1.2. Excel Main Window Display at Startup

All XLS Padlock Options > Customize App > Splash screen > Excel Main Window Display at Startup

Thanks to the "**Excel Main Window Display at Startup**" option, you can define how the excel window will be sized when it starts: normal, minimize, maximize. By default, it's normal: the window is not modified.

Do not display the "Loading workbook" dialog box in Excel

15.4.2. EXE Icon and Version Information

15.4.2.1. Change Exe Icon

XLS Padlock can change the **default icon of the application.** You can have your own icon for your application by specifying the **path to an icon file** (it must be a true icon file, .ico extension).

XLS Padlock supports any icon image such as 32×32, 16 colors or 48×48, 256 colors: if you need to create or extract icons easily, then try GConvert: our companion tool lets you extract icons or convert images into icons.

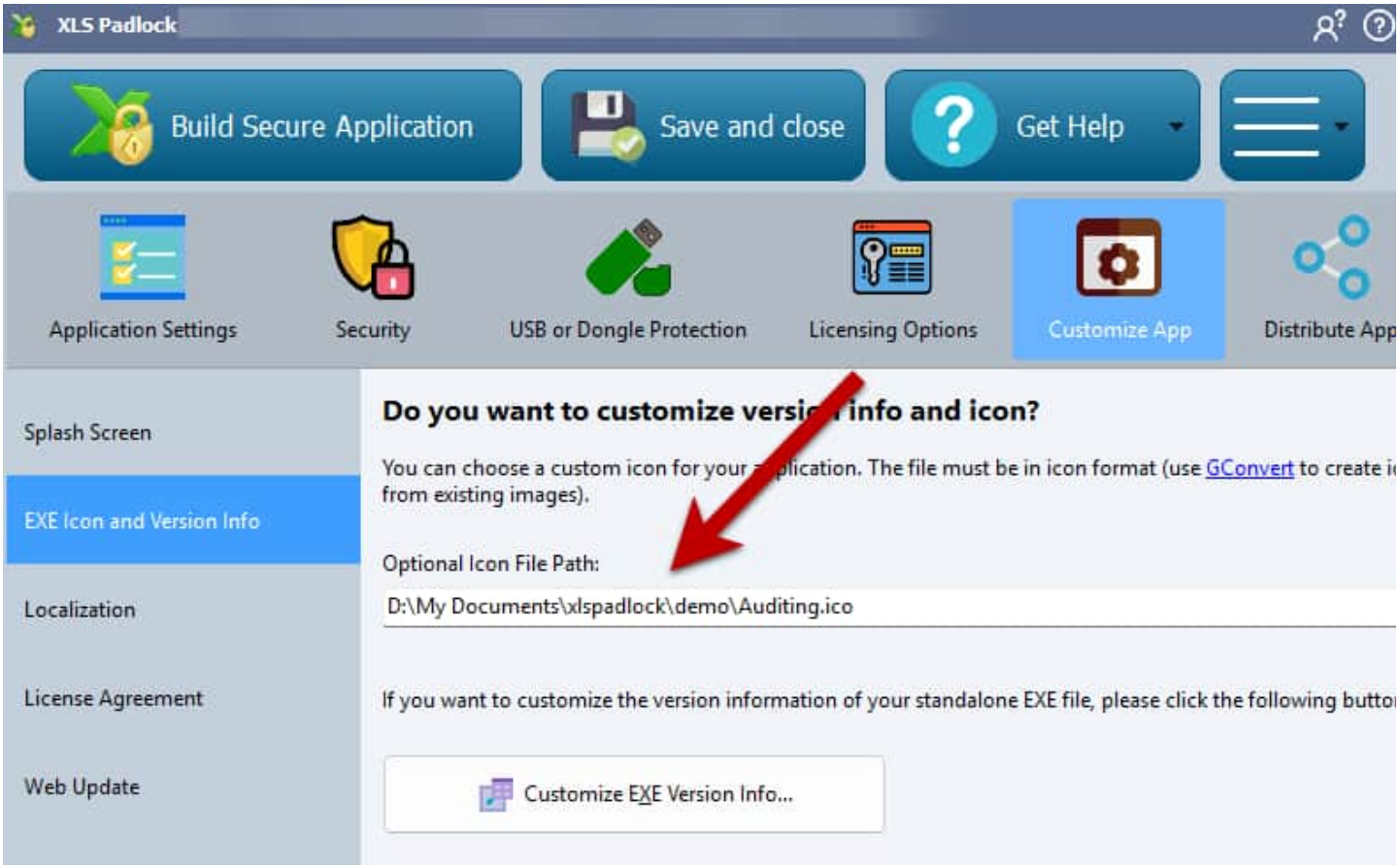
https://www.gdgsoft.com/gconvert


The icon file should be available as an external file when the application is being compiled.

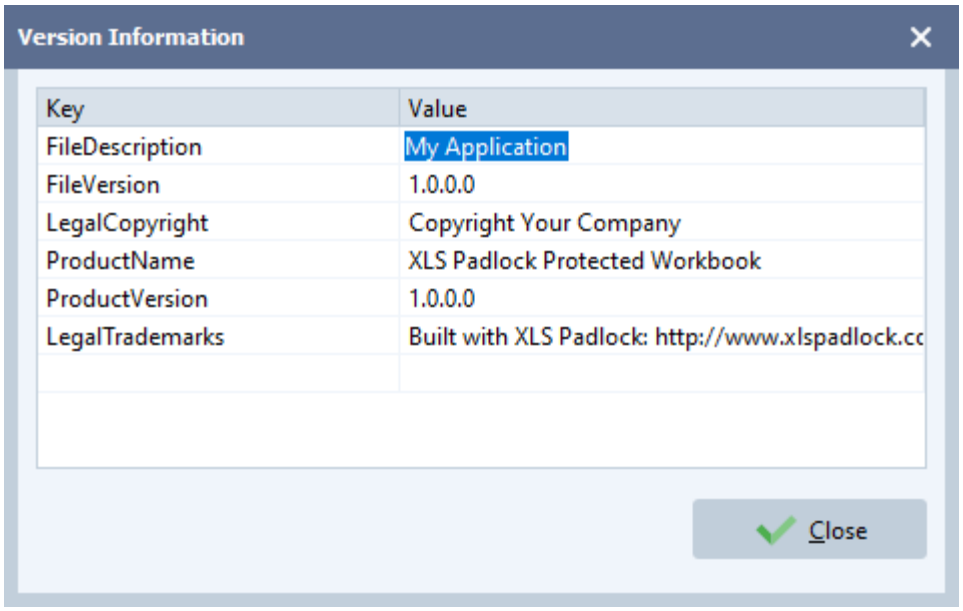
XLS Padlock will also use this icon for the standalone application's EXE file.

15.4.2.2. EXE Version Info

The **version information** of an executable file is a special resource section that contains such information about the file as its version number, its intended operating system, its original filename, its copyright information… This information is then included in the compiled code. When version information is included, end users can right-click the program icon and select Properties to display the version information (or press ALT+ENTER in Explorer).



XLS Padlock allows you to insert your own version information in the standalone EXE file. Click "**Customize EXE Version Info**" and a window with different fields appears:



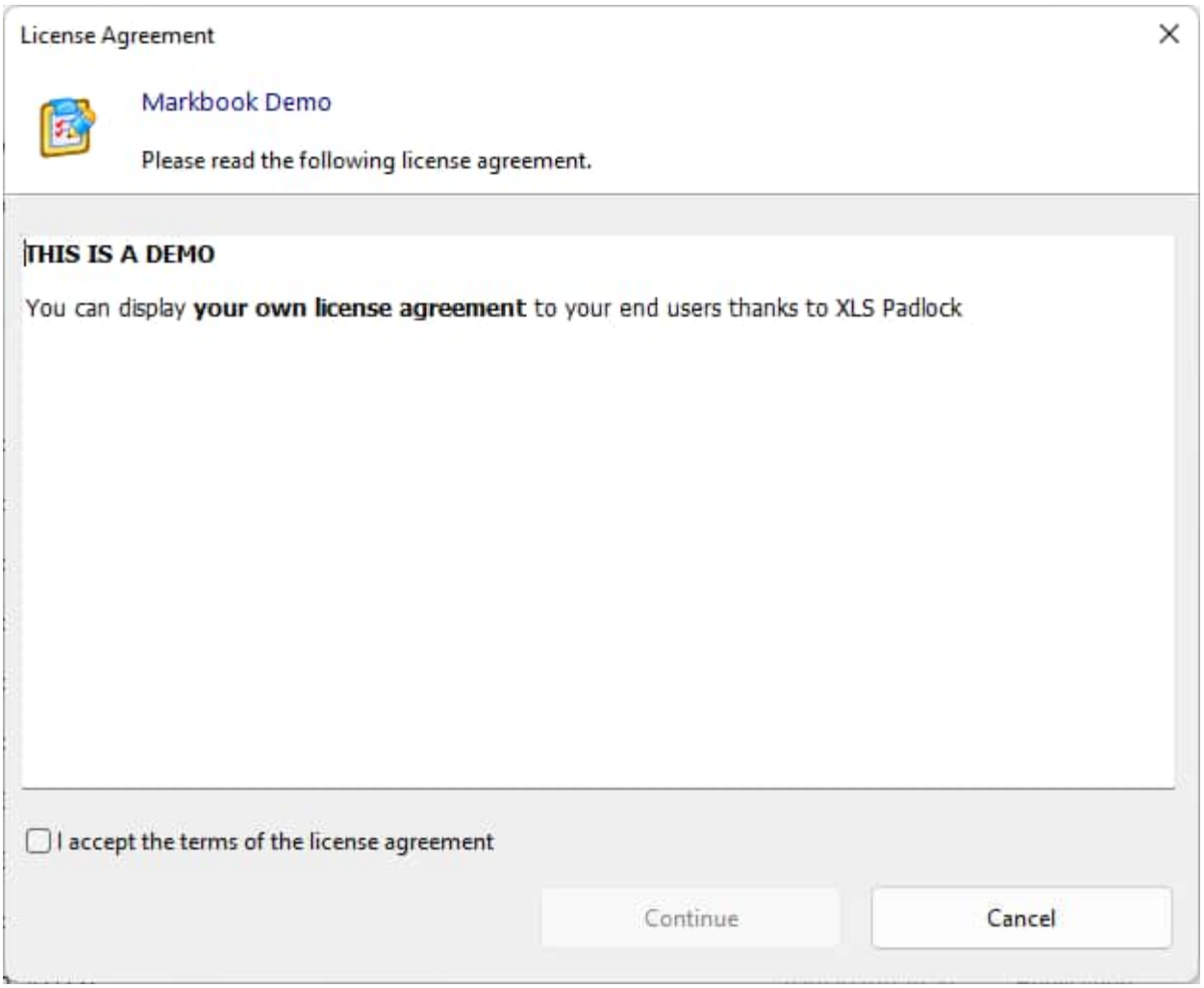
- File Description: should be the description of your application contents. Company Name: name of your company
- File Version: the current release number of your .exe file. The format must be X.X.X.X where X is a number, such as 1.20.34.45. The **File Version** value is also used by the Web Update feature to determine whether a new version of the application is available or not.
- Legal Copyright: will appear under the "Legal Copyright" entry. Enter your own copyright such as "Copyright 2019 by Yourself. All rights reserved."

- Product Name: name of your product / application. Generally, the same as your application title.
- Product version: the current release number of your application. Same format as above.
- Legal Trademarks: any trademark that you want to add.

15.4.3. License Agreement

You can **display your own EULA - End User License Agreement - before the workbook starts**. Customers must accept this agreement once prior to using the workbook.

The compiled workbook will display a dialog box as shown below:



When the user checks "I accept the terms of the license agreement", the "Continue" button activates, and he can proceed to opening the secure workbook.

To load your own EULA, click **Edit** and use the rich text editor which can open RTF files (you can also copy/paste from Microsoft Word for instance).

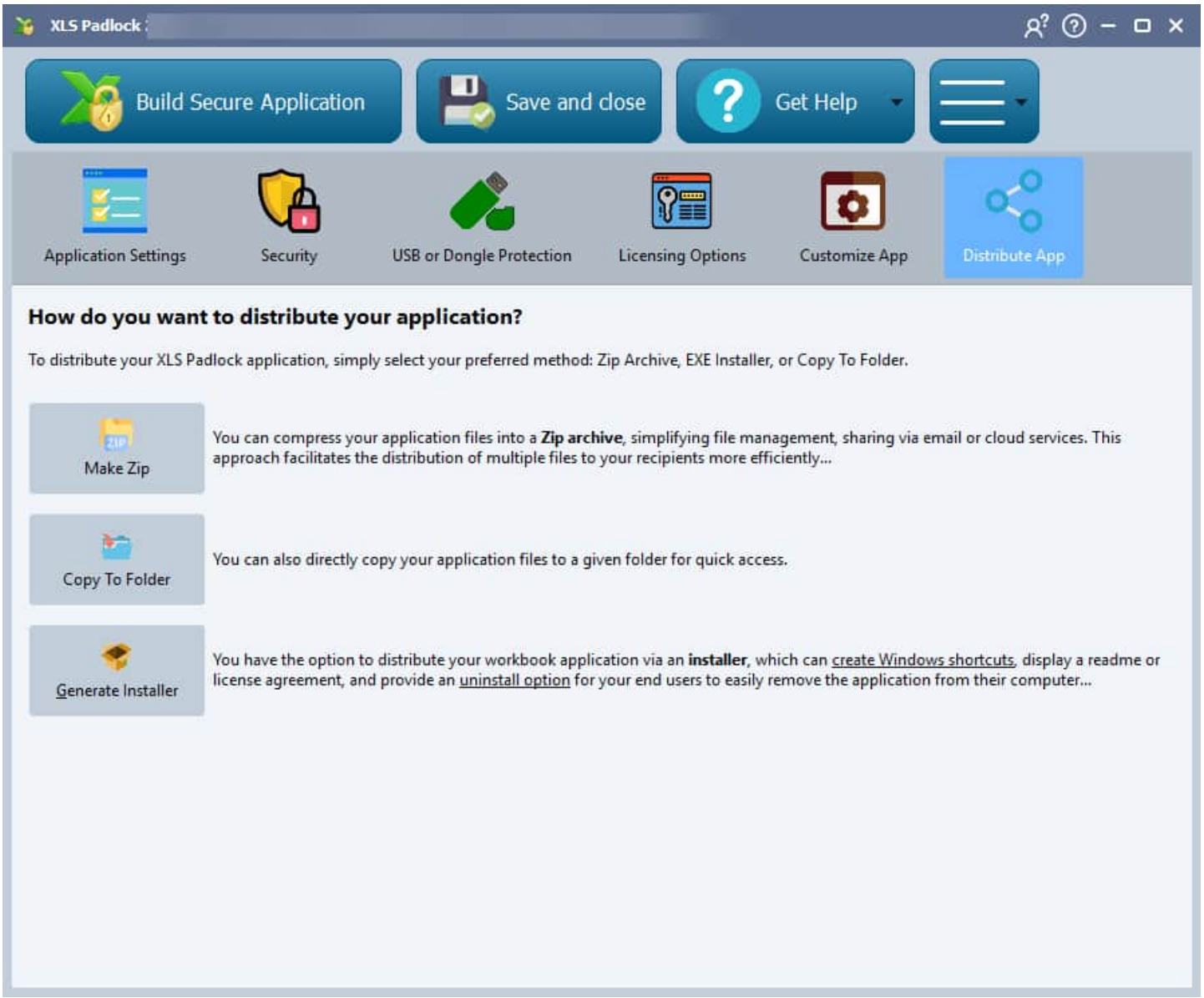
- If the field is empty, no dialog box is displayed.
- If you need the EULA to show up every time before the workbook is opened, tick the option **Ask end**

users for agreement at each startup.

15.5. Distribute App

When you have converted your workbook as an executable (.EXE) file, the next step is to distribute it to your users.

XLS Padlock offers several methods to streamline this process: through a Zip Archive, an EXE Installer, or simple Copy To Folder.



Zip Archive

A Zip archive is a popular choice for distributing applications, thanks to its simplicity and compatibility. By compressing your application files into a single Zip file, you make file management a breeze. This method is especially useful for sharing your application via email or cloud services, as it reduces the file size and bundles multiple files into one easy-to-manage package.

To create a Zip archive of your XLS Padlock application, click **Make Zip and choose where you want to create the .zip file.**

EXE Installer

For a more professional touch, consider distributing your workbook application via an EXE installer. This method enhances the user experience by providing a familiar installation process, common in Windows environments. The installer can create Windows shortcuts, display a readme or license agreement before installation, and offer an uninstall option, allowing users to easily remove the application from their computer if necessary.

To create an installer, see the dedicated topic: make an installer for your EXE file

Copy To Folder

Finally, copying your application files directly to a folder might be the fastest route. This method is as simple as it gets:

- Copy all necessary application files into a folder.
- Share this folder with your users through physical media (like a USB drive) or through network transfer.

While not as polished as the other methods, this approach offers the quickest way to get your application running on another system, without the need for zip utilities or installer software.

15.5.1. Make an installer for your application

You may want to package your compiled workbook application into an installer (or Setup program). Installers are interesting because they can install several files (additional files like readme), display additional license agreement, create shortcuts in program groups, and offer an uninstaller to let end users remove any trace of your secure application from their computer.
XLS Padlock uses our software Paquet Builder to generate custom and compact installers. **It must be installed on your computer before you can use this feature**.

Paquet Builder is a mix between a 7-Zip Self-Extracting archive maker and a Setup routine generator. Thanks to its exhaustive feature set, you can create flexible and compact self-extractors for professional file and software delivery. Package up any document or program files, visually construct simple or sophisticated multilanguage distribution and installation packages; generate updates and patches; wrap multimedia presentations or several Windows Installer MSI setups into single .exe files ready for delivery over the Internet.

Click **Generate Installer** in XLS Padlock / Distribute EXE to display a new window named "Make Setup for your compiled workbook"

First, you need to fill in the three fields "Destination path", "Setup title" and "Your Application Name": "Destination Path" is the default folder where you want your application to be installed, "Setup Title" is the title

that will appear on all Setup windows, and “Your Application Name” will appear on shortcuts, uninstaller, etc.

- Secondly, press **Generate Installer** to create the project. XLS Padlock will then launch Paquet Builder to let you modify the new project and of course create the installer for you.

- Tutorial: how to make an installer for your Excel workbook

16. Use external references and hyperlinks

XLS Padlock can protect one workbook per EXE file. If your workbook uses external references or requires additional files (or even workbooks), you'll have to update it to make it compatible with XLS Padlock.

XLS Padlock provides you with several ways to access additional resource or source files:

- Add them as Companion files (see Add Companion Files).
- Use hyperlinks or references with VBA code.

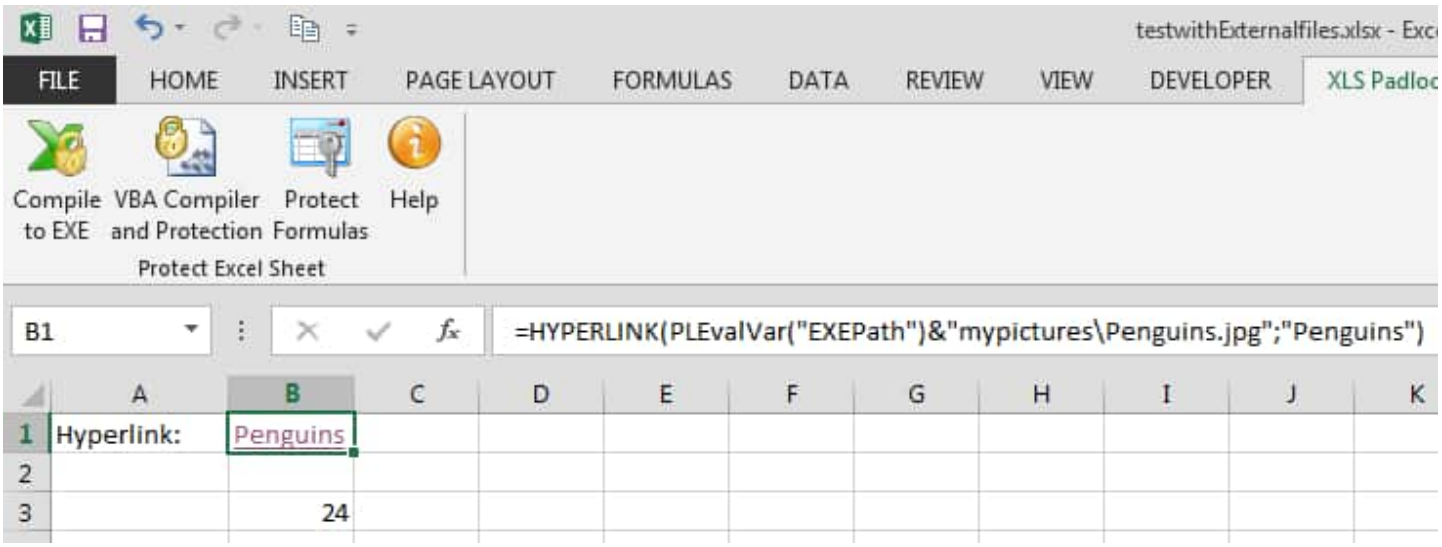
XLS Padlock offers an Excel function to retrieve paths to external files. This function is called PLEvalVar and can be called directly from Excel or with VBA code.
This function takes one argument in string format:

· =PLEvalVar("EXEPath") returns the full path to the folder that contains the application EXE file (and trailing backslash).
· =PLEvalVar("XLSPath") returns the full path to the folder that contains the compiled workbook at runtime (and trailing backslash).

Note that this folder is a virtual folder and thus, you can't place real files into it. It's useful only if you work with Companion files (see Add Companion Files).

Example 1

You have hyperlinks to external image files. These image files are in the same folder as the workbook XLS file (or in a sub folder).



You have a hyperlink in a cell which is defined by:

```
=HYPERLINK("Penguins.jpg","Penguins")
```

To make it working with XLS Padlock, you must copy all external image files in the same folder as the EXE file built with XLS Padlock. Then, you have to modify all hyperlinks to insert the PLEvalVar("EXEPath") function that returns the path to that folder.
In our case, this will become:

```
=HYPERLINK(PLEvalVar("EXEPath")&"Penguins.jpg","Penguins")
```

External files must be deployed in the same folder as the application EXE file.
Do not use spaces in filenames!

For compatibility, at design time, the PLEvalVar("EXEPath") function returns the path to the folder that contains your workbook source file. So that hyperlinks still work if you insert the function.
Note: if you have sub folders, it will work too:

```
=HYPERLINK("My Pictures\Penguins.jpg","Penguins")
```

has to be replaced by:

```
=HYPERLINK(PLEvalVar("EXEPath")&"mypictures\Penguins.jpg","Penguins")
```

Example 2

You want to access external files with VBA code. Remember that external files must be in the same folder as the EXE file. You can use this code:

```
Public Function PathToFile(Filename As String)
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
PathToFile = XLSPadlock.PLEvalVar("EXEPath") & Filename
Exit Function
Err:
PathToFile = ""
End Function
```

If you pass Myfile.ext to this function, it returns the full path to the file, provided that the file is in the same folder as the EXE file.

17. EXE Command-line Switches

The secure application EXE file made from your Excel workbook accepts some command-line switches:

- "MYAPP.EXE -load" will directly prompt the user which save he wants to load.
- "MYAPP.EXE -reset" will directly discard any previous changes (without erasing saves) and load the original workbook.
- "MYAPP.EXE -del" will erase all previous saves and load the original workbook.
- "MYAPP.EXE -enterkey" will ask the end user for a new activation key (useful for instance to replace an old key that is going to expire).
- "MYAPP.EXE -deact" will start the deactivation process.
- "MYAPP.EXE -webupdate" will launch a web update.

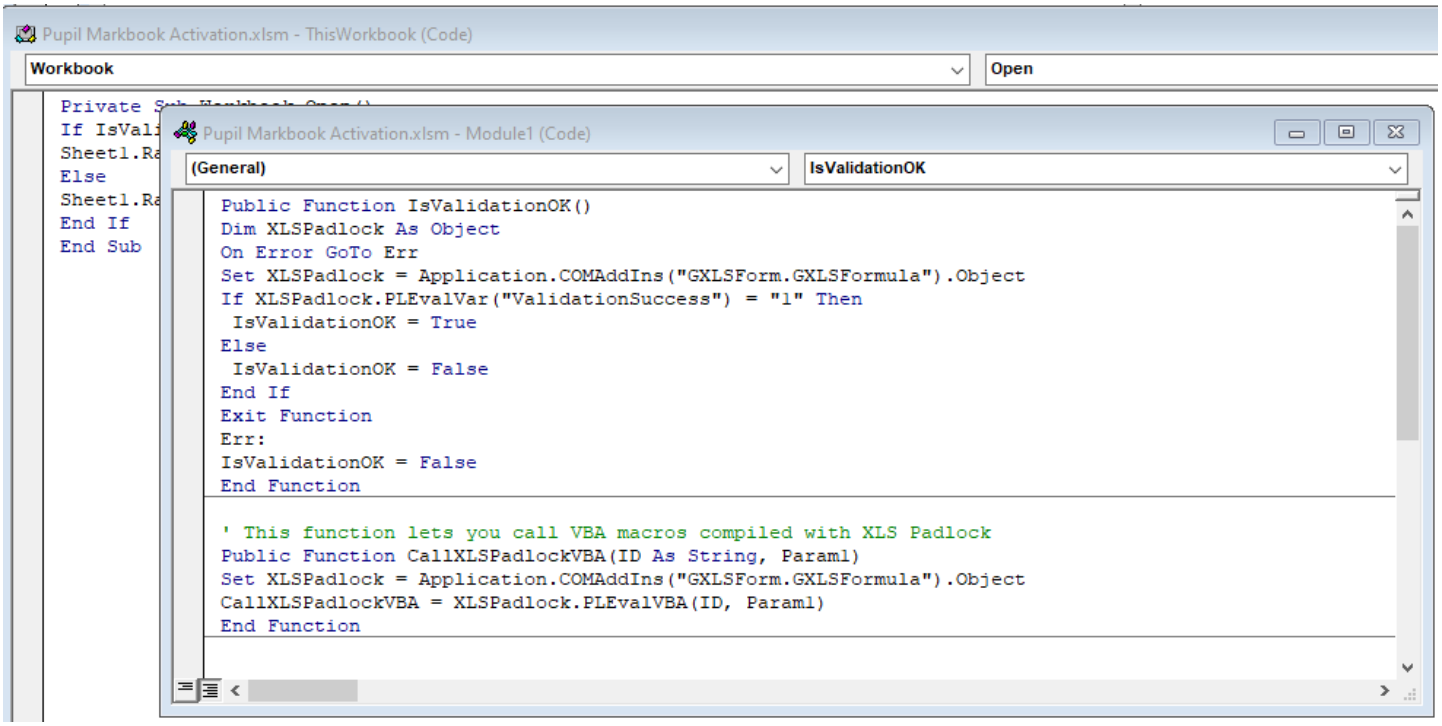
Finally, you can pass a save filename to the EXE file, so that the latter is automatically loaded without prompt:

```
MyApp.exe "D:\My Documents\123.xlsc"
```

18. XLS Padlock VBA API Extension

VBA developers can use some **VBA code extensions provided by XLS Padlock** in their protected workbooks.

All code snippets provided in this user guide can be copied/pasted directly in a VBA module for instance, as shown below:



Use the back/next buttons above on the right side to navigate between snippets or choose them directly in the TOC.

18.1. Get the path to a file in the same folder as the compiled workbook

Purpose: get the path to a file in the same folder as the compiled workbook

Insert the following code:

```
Public Function PathToFile(Filename As String)
Dim XLSPadlock As Object
On Error GoTo Err
```

```
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
PathToFile = XLSPadlock.PLEvalVar("EXEPath") & Filename
Exit Function
Err:
PathToFile = ""
End Function
```

You can then call the function:

```
Sub Test_File()
 DoSomethingWith(PathToFile("data.xls"))
End Sub
```

18.2. Test if the workbook is protected with XLS Padlock

Purpose: test if the workbook is protected with XLS Padlock

In a module, insert the following code:

```
Public Function XLSPadlockAvailable() As Boolean
Dim XLSPadlock As Object
On Error Resume Next
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
XLSPadlockAvailable = XLSPadlock.IsProtected()
End Function
```

You can then call the function:

```
Sub Test_Protected()
 If XLSPadlockAvailable() Then
  MsgBox "Protected"
 Else
  MsgBox "Not Protected"
 End If
End Sub
```

18.3. Saving a secure copy of the workbook without prompt

Purpose: saving a secure copy of the workbook without prompt

Insert the following code:

```
Public Function SaveSecureWorkbookToFile(Filename As String)
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object
```

```
SaveSecureWorkbookToFile = XLSPadlock.SaveWorkbook(Filename)Exit Function
Err:
SaveSecureWorkbookToFile = ""
End Function
```

You can then call the function:

```
Sub Test_Save()
 SaveSecureWorkbookToFile ("d:\my documents\my save.xlsc")
End Sub
```

18.4. Suggest a filename for the save dialog box

Purpose: suggest a filename for the save dialog box

```
Insert the following code:
Public Function SetSecureWorkbookFilename(Filename As String)
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object
XLSPadlock.SetDefaultSaveFilename(Filename)
SetSecureWorkbookFilename = "OK"Exit FunctionErr:
SetSecureWorkbookFilename = ""
End Function
```

You can then call the function:

```
Sub Test_SetFilename()
SetSecureWorkbookFilename ("d:\my documents\ my save.xlsc")
End Sub
```

18.5. Open an existing save file with VBA

Purpose: open an existing save file with VBA

It is possible to open a save file of your application using VBA code. However, please note that the file will be loaded in a new instance of Excel and not in the current instance. This is the same as starting the EXE file a second time.

Insert the following code (for instance in a module):

```
Public Sub LoadXLSPadlockSaveFile(FilePath As String)
On Error Resume Next
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
XLSPadlock.PLOpenSaveFile (FilePath)
```

```
End Sub
```

For instance, the following macro (to be associated to a button) will prompt end users for the XLSC save file they want to open, and then open it:

```
Sub Load_Old_Save()

strFileToOpen = Application.GetOpenFilename _
(Title:="Please choose a save file to open", _
FileFilter:="Save Files (*.xlsc),*.xlsc")

If strFileToOpen = False Then
    MsgBox "No file selected.", vbExclamation, "Error"
    Exit Sub
Else
    LoadXLSPadlockSaveFile (strFileToOpen)
End If
End Sub
```

18.6. Check if the compiled workbook is in trial state

Purpose: check if the compiled workbook is in trial state

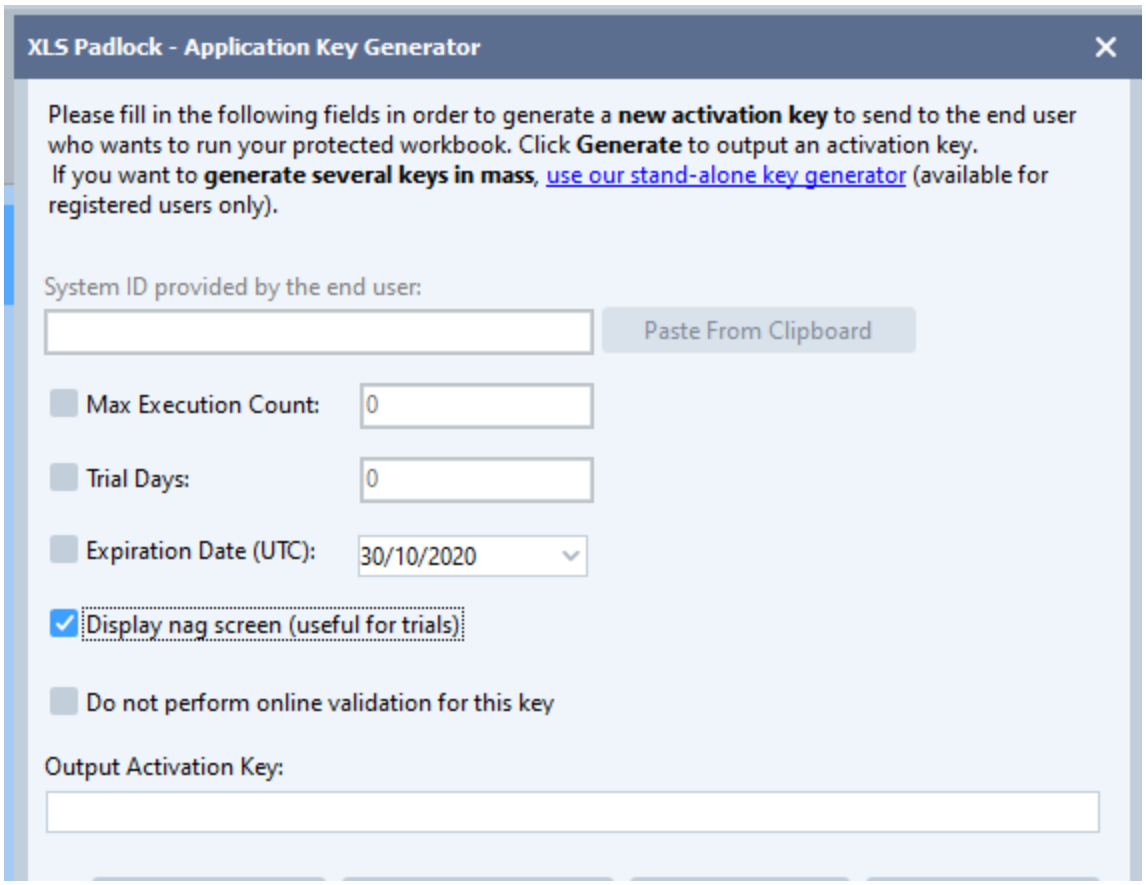
Insert the following code:

```
Public Function IsTrial()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
IsTrial = XLSPadlock.PLEvalVar("IsTrial")
Exit Function
Err:
IsTrial = False
End Function
```

You can then call the function:

```
Sub Test_IsTrial()
If IsTrial() Then
   MsgBox "Trial"
  Else
   MsgBox "Registered"
  End If
End Sub
```

Warning: the function will return true only if the activation key has the "display nag screen" flag (trial):



18.7. Getting command-line parameters passed to the EXE file

Purpose: getting command-line parameters passed to the EXE file

When you run the EXE file, you can pass command-line parameters to it. For some reason, your VBA code may need to retrieve values of these parameters.

Insert the following code:

```
Public Function ReadParamStr1()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
ReadParamStr1 = XLSPadlock.PLEvalVar("ParamStr1")
Exit Function
Err:
ReadParamStr1 = ""
End Function
```

You can then call the function:

```
Sub Test_Param()
   MsgBox ReadParamStr1()
End Sub
```

This code shows a ParamStr1 variable. This variable tells XLS Padlock to return the value of the first command-line parameter. For the 2nd, use ParamStr2 and so on. Up to 3 parameters are handled: others are ignored.

18.8. Getting EXE File Version and Product Version

Purpose: getting EXE File Version and Product Version

XLS Padlock lets you set the file version and product version of the EXE file, as explained in EXE version info.

If you want to retrieve these two values in your workbook with VBA code, use the following one:

```
Public Function ReadProductVersion()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
ReadProductVersion = XLSPadlock.PLEvalVar("ProductVersion")
Exit Function
Err:
ProductVersion = ""
End Function
```

You can then call the function:

```
Sub Test_ProductVersion()
   MsgBox ReadProductVersion()
End Sub
```

The same works for File Version. Just replace ProductVersion by FileVersion as:

```
… = XLSPadlock.PLEvalVar("FileVersion")
```

18.9. Getting EXE Filename

Purpose: getting EXE Filename

This lets you retrieve the filename of the EXE file.

Use the following VBA code:

```
Public Function GetEXEFilename()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
GetEXEFilename = XLSPadlock.PLEvalVar("EXEFilename")
Exit Function
Err:
GetEXEFilename = ""
End Function
```

You can then call the function:

```
Sub Get_EXE_Filename()
 MsgBox (GetEXEFilename())
End Sub
```

18.10. Allow saving non encrypted workbooks

Purpose: allow saving non encrypted workbooks

As a VBA programmer, if you use the instruction newworkbook.saveas, XLS Padlock saves an encrypted workbook file. If you want to save a normal workbook, you can use the following VBA code snippet:

```
Dim wkb As Workbook
' Adding New WorkbookOn Error Resume Next
Set wkb = Workbooks.Add
' Do what you need…
'Saving the Workbook
Dim XLSPadlock As Object
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object
XLSPadlock.SetOption Option:="1", Value:="1"

wkb.SaveAs "C:\My Documents\WorkbookName.xls"
XLSPadlock.SetOption Option:="1", Value:="0"
```

You can see the following line:

```
XLSPadlock.SetOption Option:="1", Value:="1"
```

This line tells XLS Padlock to allow a normal workbook to be saved. Value should be either 1 (True) or 0 (False).

After having saved the workbook, be sure to call:

```
XLSPadlock.SetOption Option:="1", Value:="0"
```

It will restore the default save behavior.

XLS Padlock will prevent saving normal workbooks if the security option "Do not allow loading/saving other workbooks" is enabled. To solve this problem, see Loading/Saving workbooks through VBA SetOption helper.

18.11. Get current workbook save file path

Purpose: get current workbook save file path

This lets you retrieve the path to the workbook save file that has been selected by the user at startup (if any).

Insert the following code:

```
Public Function GetSecureWorkbookFilename()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object
GetSecureWorkbookFilename = XLSPadlock.GetSaveFilename()
Exit Function
Err:
GetSecureWorkbookFilename = ""
End Function
```

The previous code can be tested with:

```
Sub Test_GetFilename()
 MyFile = GetSecureWorkbookFilename
 MsgBox (MyFile)
End Sub
```

18.12. Get local save folder path

Purpose: get local save folder path

This lets you retrieve the path to the local folder where XLS Padlock stores your application's settings and secure save files.

Insert the following code:

```
Public Function GetStoragePath()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
GetStoragePath = XLSPadlock.PLEvalVar("SPath")
Exit Function
Err:
GetStoragePath = ""
End Function
```

The previous code can be tested with:

```
Sub Test_GetStoragePath()
 MyPath = GetStoragePath
 MsgBox (MyPath)
End Sub
```

18.13. Test if online validation was successful or not

Purpose: test if online validation was successful or not

XLS Padlock supports online validation of activation keys. You can decide what to do if validation fails.

Use the following function to get the result of the validation process:

```
Public Function IsValidationOK()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
If XLSPadlock.PLEvalVar("ValidationSuccess") = "1" Then
 IsValidationOK = True
Else
 IsValidationOK = False
End If
Exit Function
Err:
IsValidationOK = False
End Function
```

You can then call the function. For instance, the following code sets the value of a cell:

```
Private Sub Workbook_Open()
If IsValidationOK Then
Sheet1.Range("S24") = "OK"
```

```
Else
Sheet1.Range("S24") = "FAIL"
End If
End Sub
```

18.14. Retrieve the activation token with VBA

Purpose: retrieve the activation token with VBA

XLS Padlock supports online activation for your Excel workbook app.

In some cases, you may need to know the activation token. The following VBA code allows you to get a hashsum of this activation token returned by the server on a successful activation. Note that the activation token itself is not locally stored, only its hashsum.

Use the following function to get the hashsum:

```
Public Function ReturnValidationToken()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
ReturnValidationToken = XLSPadlock.PLEvalVar("ValidationToken")
Exit Function
Err:
ReturnValidationToken = ""
End Function
```

You can then call the function. For instance, the following code sets the value of a cell:

```
Private Sub Workbook_Open()
Sheet1.Range("S24") = ReturnValidationToken()
End Sub
```

18.15. Retrieving subscription/licence information with VBA

Purpose: retrieving subscription/licence information with VBA

XLS Padlock supports online activation and license validation for your Excel workbook app.

In some cases, you may need your activation server to pass some additional data about the subscription or license of your end user. The following VBA code allows you to get a custom string value returned by the server on a successful validation.

This function will only work **after a successful validation.** It will return blank data in all other cases.

Please refer to the documentation of the XLS Padlock's Activation Kit / FastSpring subscription kit / WooCommerce Integration Kit to see how to pass custom data to the workbook app during the validation process.

```
Public Function ReturnValidationAdditionalServerData()
Dim XLSPadlock As Object

On Error GoTo Err

Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object

ReturnValidationAdditionalServerData =
XLSPadlock.PLEvalVar("ValidationAddServerData")

Exit Function

Err:

ReturnValidationAdditionalServerData= ""

End Function
```

You can then call the function. For instance, the following code sets the value of a cell:

```
Private Sub Workbook_Open()

Sheet1.Range("S24") = ReturnValidationAdditionalServerData()

End Sub
```

18.16. Test if Internet connection is available or not

Purpose: test if Internet connection is available or not

Use the following function to test whether an Internet connection is available on the end user's computer:

```
Public Function IsInternetAvailable()
```

```
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
If XLSPadlock.PLEvalVar("InternetConnected") = "1" Then
 IsInternetAvailable = True
Else
 IsInternetAvailable = False
End If
Exit Function
Err:
IsInternetAvailable = False
End Function
```

You can then call the function. For instance, the following code sets the value of a cell:

```
Private Sub Workbook_Open()
If IsInternetAvailable Then
Sheet1.Range("S24") = "Internet Available"
Else
Sheet1.Range("S24") = "No Internet"
End If
End Sub
```

18.17. Hide wait dialog programmatically with VBA

Purpose: hide wait dialog programmatically with VBA

You can hide the « Loading worbook, please wait… » dialog box with a VBA call. By default, it is hidden after a given amount of time.

Insert the following code to hide the dialog box:

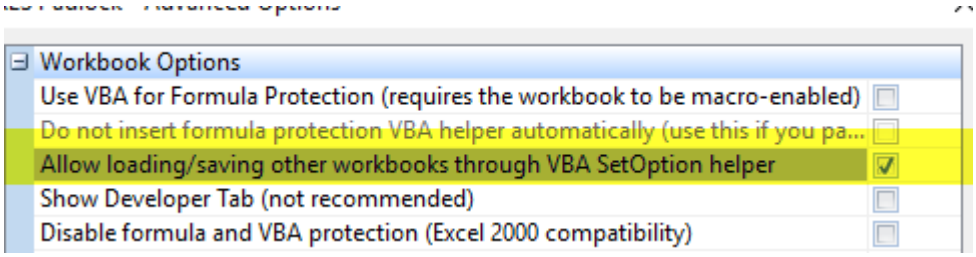
```
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object
XLSPadlock.SetOption Option:="3", Value:=""
```

18.18. Loading/Saving workbooks through VBA SetOption helper

Purpose: loading/Saving workbooks through VBA SetOption helper

If you enabled the "Do not allow loading/saving other workbooks" option (see Do not allow loading/saving other workbooks paragraph) and still need to do it through VBA, do this:

- Enable the following Advanced option:



- Then, use the following VBA code snippet:

```
Dim wkb As WorkbookOn Error Resume Next
' Adding New Workbook
Set wkb = Workbooks.Add
' Do what you need…
'Saving the Workbook
Dim XLSPadlock As Object
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object
XLSPadlock.SetOption Option:="2", Value:="0"
XLSPadlock.SetOption Option:="1", Value:="1"

wkb.SaveAs "D:\My Documents\WorkbookName.xlsx"

XLSPadlock.SetOption Option:="2", Value:="1"
XLSPadlock.SetOption Option:="1", Value:="0"
```

You can see the following line:

```
 XLSPadlock.SetOption Option:="2", Value:="0"
```

This line tells XLS Padlock to allow loading/saving workbooks as usual. Value should be either 1 (True) or 0 (False).
The second line

```
 XLSPadlock.SetOption Option:="1", Value:="1"
```

tells XLS Padlock to disable the prompt to save encrypted workbooks.

After having saved the workbook, be sure to call:

```
XLSPadlock.SetOption Option:="2", Value:="1"
XLSPadlock.SetOption Option:="1", Value:="0"
```

It will restore the default behavior.

18.19. Create other Excel instances and access secure workbook and companion files

Purpose: create other Excel instances and access secure workbook and companion files

By default, only the Excel instance run by the compiled workbook EXE file is allowed to access the workbook file and its companion files (if any).

If you start another Excel instance from VBA, it can't access the secure workbook file unless you use the following VBA code to allow access.

```
Sub RunOtherExcel()

Dim XLSPadlock As Object
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object

another_file = PathToCompiledFile("data.xlsx")

' Start a new Excel instance

Set appExcel = CreateObject("Excel.Application")


' Tell XLS Padlock to grant access to virtual files in new Excel instance

Dim Res As Long

Res = appExcel.Application.Hwnd

XLSPadlock.SetOption Option:="5", Value:=Res

' The new Excel instance can now work with virtual workbook files:

Set wbExcel = appExcel.Workbooks.Open(another_file, False, True)

Set wsExcel = wbExcel.Worksheets("Sheet1")

MsgBox (wsExcel.Cells(1, 1).Value)

' Done, be sure to quit the application.

appExcel.Application.Quit
' Release the object variable.
Set appExcel = Nothing
```

18.20. Retrieve XLS Padlock System ID from VBA

Purpose: retrieve XLS Padlock System ID from VBA

XLS Padlock lets you use hardware-locking to secure your workbook. This is based on a unique system ID, see "Use hardware-locked keys".

If you want to retrieve the value of the system ID with VBA code, use the following one:

```
Public Function ReadSystemID()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
ReadSystemID = XLSPadlock.PLEvalVar("SystemID")
Exit Function
Err:
ReadSystemID = ""
End Function
```

You can then call the function:

```
Sub Test_ReadSystemID()
   MsgBox ReadSystemID()
End Sub
```

18.21. Terminate the "Loading workbook message" early

Purpose: terminate the "Loading workbook message" early from VBA

Here is the VBA code to hide that dialog box earlier:

```
Dim XLSPadlock As Object
Set XLSPadlock = Application.COMAddIns("GXLS.GXLSPLock").Object
XLSPadlock.SetOption Option:="3", Value:="0"
```

Invoke this when you want the Loading workbook message to be hidden.

18.22. Retrieve number of remaining days in trial with VBA

Purpose: retrieve number of remaining days in trial with VBA

The following VBA code lets you retrieve the number of remaining days for a trial workbook. It will also work for non-trial keys if your key has an expiration date or number of runs.

```
Public Function ReadTrialState()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
ReadTrialState = XLSPadlock.PLEvalVar("TrialState")
Exit Function
Err:
ReadTrialState = ""
End Function
```

You can then call the function:

```
Sub Test_Trial()
 rdays = ReadTrialState()
 Worksheets("Sheet1").Range("A1").Value = rdays
End Sub
```

18.23. Determine if Current Activation Key Has an Expiration Date or Usage Limits

Purpose: determine if the activation key has an expiration date, number of days or runs

The following VBA code lets you will help you determine whether your current trial key has an expiration date or is limited by the number of days or runs.

```
Public Function ReadKeyExpState()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
ReadKeyExpState = XLSPadlock.PLEvalVar("KeyExpState")
Exit Function
Err:
ReadKeyExpState = ""
End Function
```

You can then call the function:

```
Sub Test_Trial()
 rdays = ReadKeyExpState()
 Worksheets("Sheet1").Range("A1").Value = rdays
End Sub
```

Possible values:

- 0: unavailable.
- 1: key with expiration date.
- 2: key with a given number of runs.
- 3: key with a given number of days.

ⓘ You can retrieve the number of runs or days here.

18.24. Start Deactivation of the Workbook Application With VBA

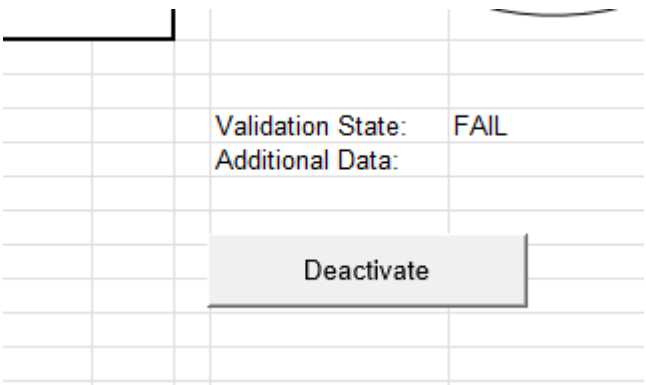
Purpose: How to Start Deactivation of the Workbook Application With VBA

The following VBA code lets users start the deactivation process on their computer.

```
Public Sub StartDeactivation()
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
XLSPadlock.PLDeactivate
Err:
End Sub
```

You can then invoke the sub thanks to a macro assigned to a button for instance:

```
Sub AskForDeactivation()
    Dim response As Integer

    ' Ask the user if they really want to deactivate the application
    response = MsgBox("Do you really want to deactivate the application?",
vbQuestion + vbYesNo, "Confirm Deactivation")

    ' Check the user's response
    If response = vbNo Then
        ' If the user selects "No," exit the procedure
        Exit Sub
    Else
        ' If the user selects "Yes," you can call the deactivation function here
         Call StartDeactivation
         Application.Quit
    End If
End Sub
```



In our workbook demo app, we added a simple Deactivate button.

□□ Please note the "Application.Quit" command in the code above to make sure that Excel is closed before we start deactivating the application.

18.25. Execute VBA Code After Saving Secure Workbook

When working with secure Excel workbooks, it's often necessary to perform specific actions immediately after a save operation. The following feature is particularly useful for developers looking to add custom post-save functionalities, such as logging activities, notifying users, or triggering other processes.

XLS Padlock invokes the following VBA callback: XLSPadlock_OnAfterSave. By including this subroutine in your VBA project, XLS Padlock will automatically invoke it each time the workbook is saved.

Implementing XLSPadlock_OnAfterSave

To leverage this feature, you simply insert a subroutine named XLSPadlock_OnAfterSave into your VBA module. The subroutine must have one parameter, SaveFilename, which is a string passed by XLS Padlock that contains the full path to the saved secure workbook file.

Here's a basic implementation of this subroutine:

```
Sub XLSPadlock_OnAfterSave(SaveFilename As String)
    MsgBox "The workbook has been successfully saved as: " & SaveFilename
End Sub
```

The example above provides a simple message box that informs the user of the save operation's success and shows the file path of the saved workbook.

19. Frequently Asked Questions

19.1. Is there any shortcut in XLS Padlock?

Yes, the following shortcuts are available:

- F1 Help
- F5 Build
- F7 Save and Close

19.2. How do users open an XSLC save file?

By default, secure applications (EXE files) can save the workbook modifications to an external encrypted save file. This is exactly the same as for a normal Excel workbook file (XLSX) except that it cannot be opened directly in a normal Excel instance. **Secure save files can only be opened by the EXE file which produced them**.

To open an existing save file, customers can:

- Run the EXE directly and click “choose save” when prompted.
- Drag an XLSC file from Windows Explorer and drop it onto the EXE file’s icon.
- Use the command line to run the EXE with the -load switch. For instance, MYAPP.EXE -load
- Pass the XLSC file directly to the EXE with the command line: MYAPP.EXE "c:\my documents\mysave.xlsc"

19.3. Is my original excel file stored on your server or will the software manipulate my files locally only?

We have never access to your original workbook file. Our software works offline on your machine, so you keep your original workbook on your own computer.

It's important to let you know that we never get access to your original workbook files. Our software works offline on your PC. Please also make backups of the original workbook, because XLS Padlock will not make backups of it and it will not upload it to our company's servers.

19.4. Can users store their changes directly to the EXE and not an external XLSC save file?

Unfortunately, EXE files must never be modified; otherwise, this will trigger antivirus false positives.

That's why XLS Padlock does not allow you to update EXE files but encrypted save files (.XLSC or .XLSCE).

You can consider the EXE file as an application, like Excel itself.

19.5. Is it possible to update the exe files which are already distributed amongst the users if I update it on my PC?

If you update your source workbook, you will have to recompile it as an EXE file and deploy this new EXE to your customers (generally, they just download it).

19.6. I updated my original workbook. What about existing saves made by end users?

The answer depends on the saving mode you selected for your XLS Padlock project.

If you choose the Full Save Mode:

When you deploy an updated EXE file, XLSC save files made by end users previously won't be automatically upgraded. Instead, end users will see the last changes they made and not yours. That's the expected behavior.

A solution consists in switching to the cell value saving mode.

You can also use VBA to save data entered by end users to a second workbook file and let them load this data back to your compiled workbook.

Example with code:

1) The macro below will generate a normal xls Excel file with the user data.

User needs to run the macro in the EXE file where the data will be copied from, so the initial EXE file must have the macro already. The user will run the macro below clicking on a button, which will copy the cells and ask the user to enter the file name. Obviously, you will need to replace the tab names and cells you would like to copy from in the macro below:

```
Sub GenerateData()
Dim strFile As String
```

```
'New workbook with 3 sheets
Workbooks.Add xlWBATWorksheet
ActiveSheet.Name = "SheetA"
Sheets.Add(After:=Sheets(1)).Name = "SheetB"
Sheets.Add(After:=Sheets(2)).Name = "SheetC"
ActiveWorkbook.Sheets("SheetA").Range("A1:C3").Value =
ThisWorkbook.Sheets("SheetA").Range("A1:C3").Value
ActiveWorkbook.Sheets("SheetB").Range("B3").Value =
ThisWorkbook.Sheets("SheetB").Range("B3").Value
ActiveWorkbook.Sheets("SheetC").Range("B1:C3").Value =
ThisWorkbook.Sheets("SheetC").Range("B1:C3").Value
strFile = Application.GetSaveAsFilename("", "Excel workbook (.xlsx),.xlsx", 1)
If strFile <> "False" Then ActiveWorkbook.SaveAs strFile, FileFormat:=51
ActiveWorkbook.Close False
End Sub
```

2) Upload the data to the new EXE file.

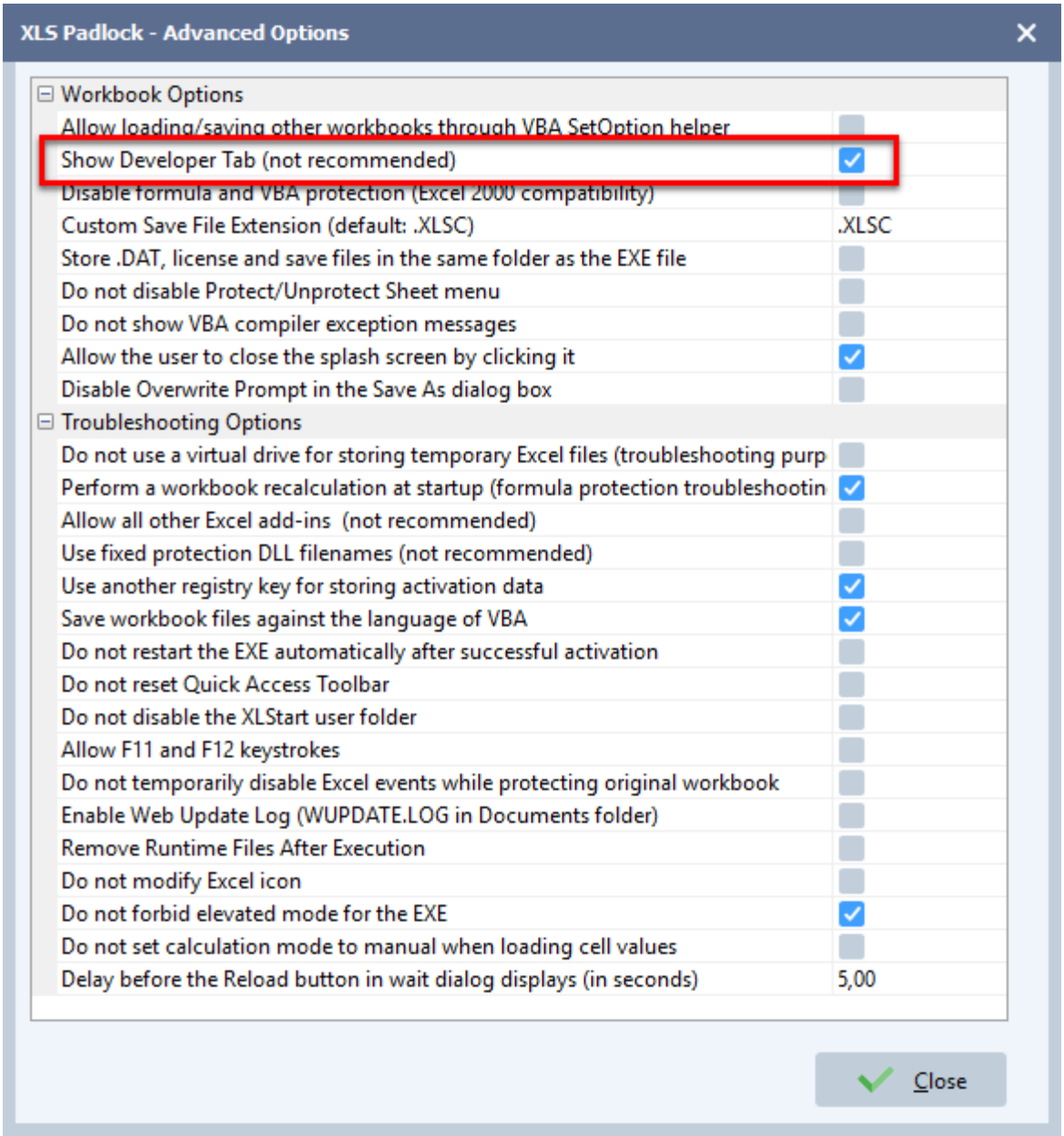
The user needs to open the new EXE and run the 3rd macro below to upload the data (link the macro to a button). Once the user runs the macro, he will be prompted to select the file (1st macro), and the data will be copied across (2nd macro). The third macro will run both macros, and that's the macro that needs to be linked to the button. Again, you will need to change the cells and tab names in the second macro, and you can also change the title in the first macro:

```
************* 1st macro:
Sub Open_Workbook_Dialog()
Dim my_FileName As Variant
my_FileName = Application.GetOpenFilename( _
FileFilter:="Excel Files,.xl;.xm", _
FilterIndex:=3, _
Title:="Select the old version of your file, where you will pull the data from", _
MultiSelect:=False)
If my_FileName <> False Then
Workbooks.Open Filename:=my_FileName
End If
End Sub

**************** 2nd macro:
Sub TransferData()
If Workbooks.Count > 1 Then
Workbooks(1).Sheets("SheetA").Range("A1:C3").Value =
Workbooks(2).Sheets("SheetA").Range("A1:C3").Value
Workbooks(1).Sheets("SheetB").Range("B3").Value =
Workbooks(2).Sheets("SheetB").Range("B3").Value
Workbooks(1).Sheets("SheetC").Range("B1:C3").Value =
Workbooks(2).Sheets("SheetC").Range("B1:C3").Value
Workbooks(2).Close savechanges:=False
Else
MsgBox "The data hasn't been transferred.", vbExclamation, "Error"
End If
End Sub

************ 3rd macro:
Sub TheTransfer()
Call Open_Workbook_Dialog
Call TransferData
End Sub
```

Note that, in order to save normal workbooks, you may also see: Loading/Saving workbooks through VBA SetOption helper

19.7. Where is the Developer ribbon?

By default, XLS Padlock hides the Developer ribbon. To enable it, go to Configure Advanced Options and enable "**Show Developer Tab**":



19.8. How to disable drag and drop?

To disable drag and drop operations for your Excel workbook, just add

```
Application.CellDragAndDrop = False
```

to the Workbook_Open VBA event, and forbid access to the VBA editor in XLS Padlock, so that customers cannot remove that line.

19.9. How to open a PDF companion file?

You can open companion files in external applications thanks to the following VBA code snippet. For instance, this will open the PDF file named UserGuide.pdf.

```
Sub OpenUserGuide()

    FileCopy PathToCompiledFile("UserGuide.pdf"), Environ("temp") & "/UserGuide.pdf")
    ActiveWorkbook.FollowHyperlink Environ("temp") & "/UserGuide.pdf"

End Sub
```

19.10. Why is the EXE file so large?

The size of the .EXE file is expected because XLS Padlock produces stand-alone EXE files that must work with different Excel and Windows versions. They also use full Unicode support, VBA interpreter and this has a "price" (large size).
Finally, EXE files have got anti-piracy protection which contributes to the size too.

To reduce the size of the EXE file:

- You can try optional UPX compression. UPX is a free open-source EXE compressor available here. However, please compress EXE files only if you use code signing too because some antivirus can trigger false positives for compressed EXE files.
- If you don't use our advanced protection features, you can disable formula protection and you'll get a smaller EXE file.

19.11. I get a name conflict "Print_Area". What can I do?

There is a known bug with some Excel versions and workbooks saved with different locales of Excel.
A workaround is available: first, be sure to remove the Print_Area name in the Defined Names of Excel before saving.
Then, add this VBA code snippet to your workbook:

```
Private Sub Workbook_Open()
Dim Sh As Worksheet
For Each Sh In ThisWorkbook.Worksheets
     With Sh
          .PageSetup.PrintArea = ""
          .PageSetup.PrintTitleRows = ""
     End With
Next
End Sub
```

And the same for:

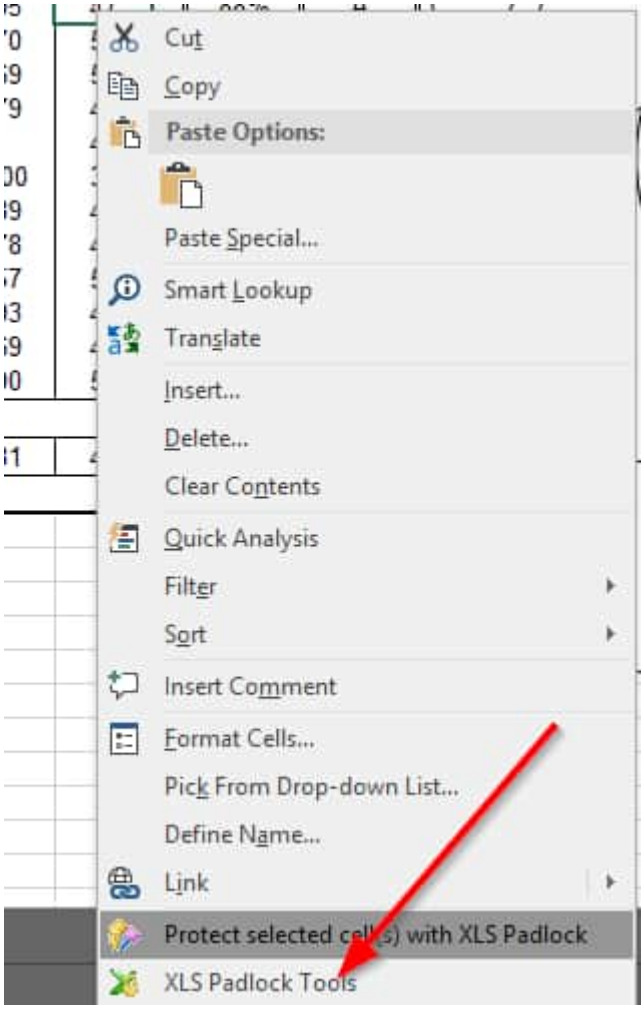
```
Private Sub Workbook_Beforeclose()
Dim Sh As Worksheet
For Each Sh In ThisWorkbook.Worksheets
     With Sh
          .PageSetup.PrintArea = ""
          .PageSetup.PrintTitleRows = ""
     End With
Next
End Sub
```

19.12. A customer gets registry error or ERegistryException error when activating the workbook

There may be a conflict with some security software or antivirus program and enable "**Use another registry key for storing activation data**".

19.13. My workbook uses a custom ribbon and thus the XLS Padlock ribbon is missing. How can I access XLS Padlock?

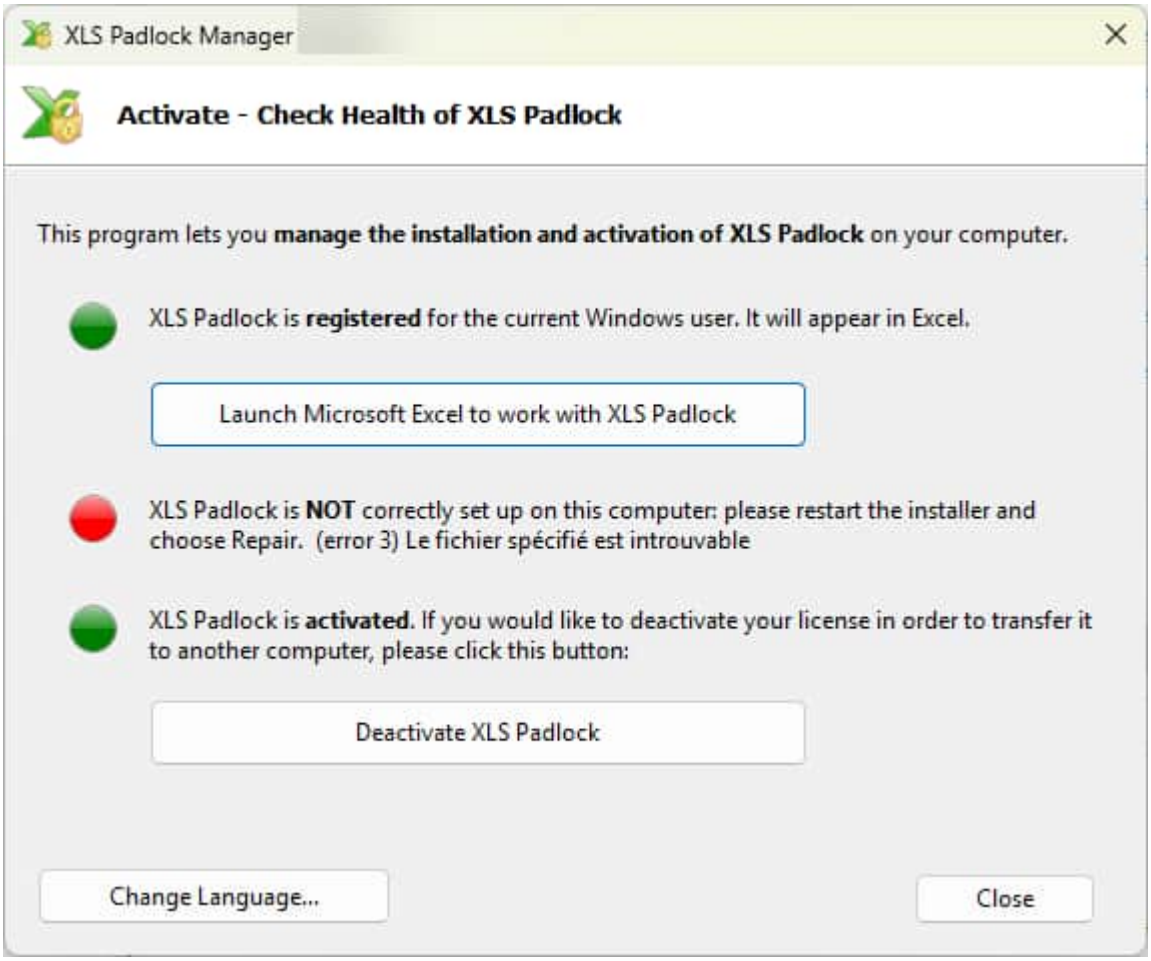
You can right-click on a worksheet and the context menu is displayed. Choose "XLS Padlock Tools" and you can access the same commands as in the missing ribbon.



19.14. XLS Padlock tab is missing in Excel

It looks like the XLS Padlock add-in is not correctly registered in Excel.

You can try to launch the XLS Padlock Manager:



The first two leds should be green. If not, try to follow the given instructions. If it still doesn't work after that, reinstall XLS Padlock.

19.15. I'm getting VBA OLE ERROR 800A03EC error when using the VBA compiler

Make sure that "Trust access to the VBA project object model" option is enabled in Excel.
See this page about how you can access this option:

https://support.office.com/en-nz/article/Enable-or-disable-macros-in-Office-documents-7b4fdd2e-174f-47e2-9611-9efe4f860b12

19.16. Getting "Run-time-error 1004. No data was imported because no elements have been mapped. "

I have a workbook with a financial model where I use an XML that I export / import to mapped cells in the workbook. When I use XLS Padlock to compile the workbook to an .exe file, the schema is no longer mapped to the cells in the workbook. (But the schema is available in the workbook, but not mapped).
I get the following message error message:
"Run-time-error 1004. No data was imported because no elements have been mapped. Use Range.XPath.SetValue to map XML elements on the worksheet".

Solution:
Turn the following option on: "Use Excel automation for formula protection" available in Formulas and Passwords.

19.17. A customer is getting unexpected error while loading protected workbook: Access violation at address 093469AF in module 'VBE6.DLL'. Read of address 00000000.

This error occurs when you use Lock VBA Project (simple VBA protection). Remove the password of your VBA project before compiling the workbook with XLS Padlock. This password is useless since the project will be marked as locked and no password will be prompted. Alternatively, use a shorter password or switch to the Prevent access to VBA editor option.

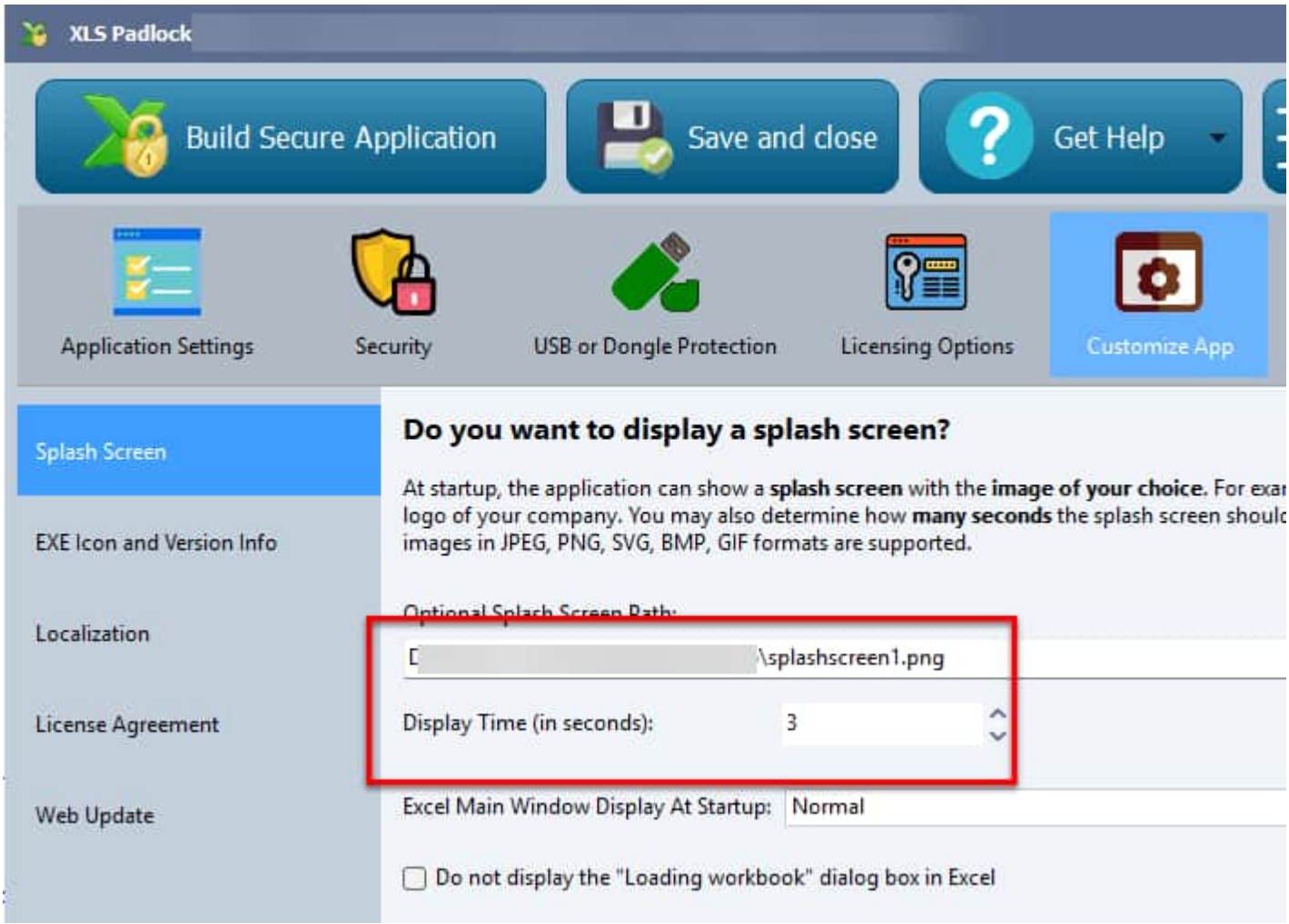
19.18. Can I restore XLS file from EXE file?

No, you can't restore from a .EXE file that doesn't support "Save As". So if you need an access to the original workbook file, be sure to enable Save As and use the "Opening a save yourself – decrypt saves" feature.

19.19. How can I hide the main Excel window at startup and only show my user form?

You have apps based only on VBA and user forms. And you want to only show your user form.

To do so, configure your XLS Padlock settings as shown below:



Then, add this VBA code to the workbook:

```
Private Sub Workbook_Open()
 Application.Visible = False
 UserForm1.Show
End Sub
```

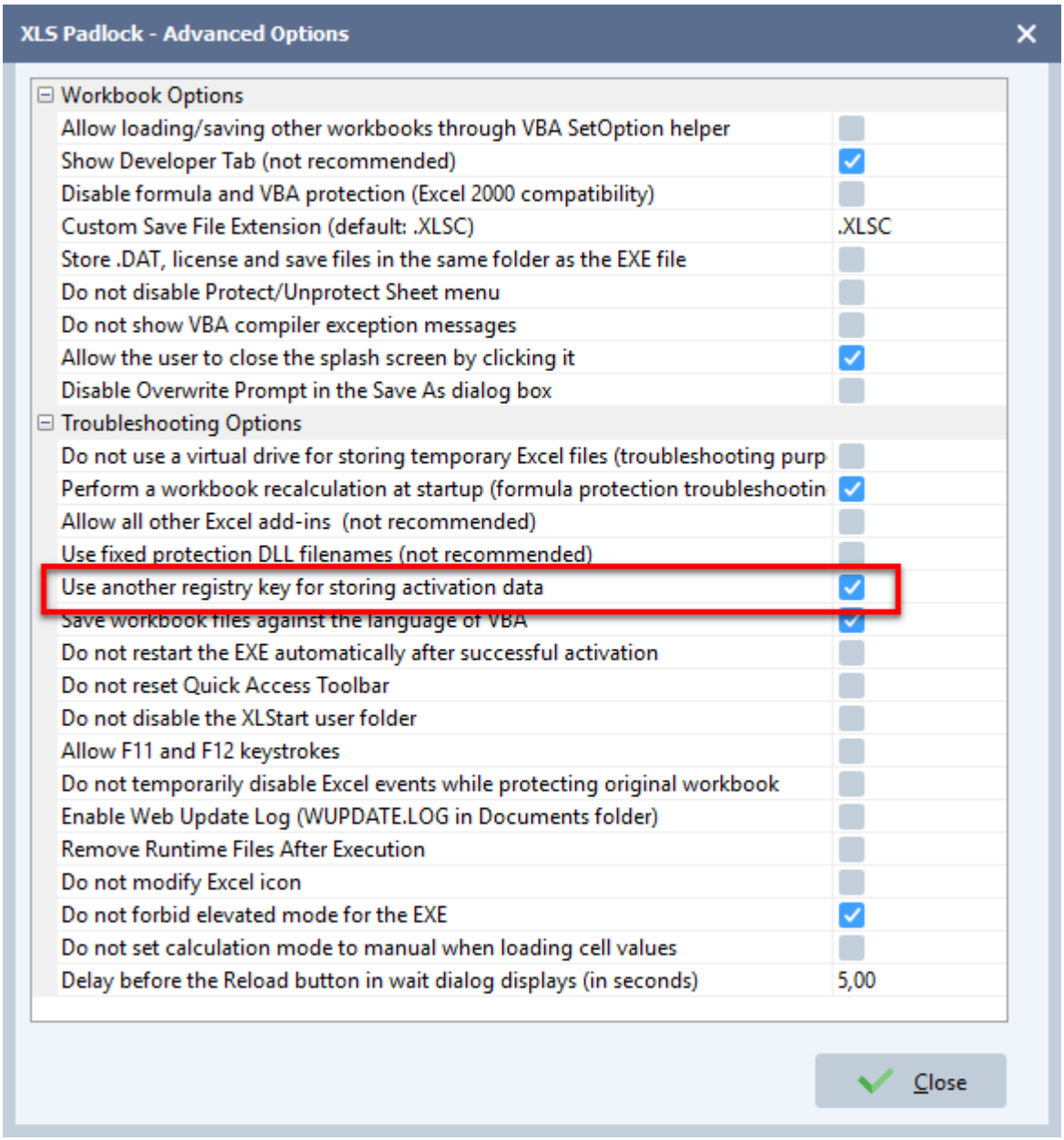
19.20. Failed to set data fot ‘Data’ error

Your end user gets this error when entering an activation key:

`Error setting data in registry - Failed to set data for 'Data'`

Reason: there is a conflict with another software which prevents the secure application to write data to registry. It is generally a security software such as an antivirus program.

Workaround: you can use the advanced option **Use another registry key for storing activation data** as shown below.



19.21. Is VBA obfuscation necessary?

The best protection is to let XLS Padlock compile parts of your VBA code into bytecode thanks to its integrated VBA compiler. You can also apply code obfuscation if you have a tool to non-compiled VBA code portions.

However, VBA obfuscation doesn't prevent someone from copying your code. True VBA compilation made with XLS Padlock does since the original VBA code doesn't exist anymore and compiled code can't be run outside your secure application made with XLS Padlock.

19.22. How to avoid error on loading xlsce file: comma instead of point in number format

On some computers, when users load a saved workbook in xlsce format, they encounter an error due to the presence of comma instead of point as decimal separator.
The excel file was created using point as decimal separator.

The error continues even if the user changes the settings in Microsoft Excel about decimal separator, choosing the point instead of comma.

Solution:
The problem is in the Windows settings for number format. Changing the decimal separator to point (.) and the thousands separator to apex (') fixes the problem.

19.23. Creating demo/trial routines with VBA

XLS Padlock provides you with some VBA routines to detect whether the EXE is run in trial version or not. Depending on that, you can let the user access VBA routines only available to registered users or not (and even display a message that this will be available only after purchase for instance).
The VBA routine to be used is in this section: Check if the compiled workbook is in trial state

19.24. Why is ThisWorkbook.Path not working?

XLS Padlock works with virtualization: when the compiled workbook is run, the Excel file is virtualized at a random location. It is never unpacked to the hard disk and thus, ThisWorkbook.Path will not point to a physical location on the disk but to the virtual location.

As a workaround, if you want to get the path to the compiled workbook (which is your EXE file), XLS Padlock provides you with VBA code that you can use in your workbooks:

```
Public Function PathToFile(Filename As String)
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
PathToFile = XLSPadlock.PLEvalVar("EXEPath") & Filename
Exit Function
Err:
PathToFile = ""
End Function
```

You can then call the function:

```
Sub Test_File()
 DoSomethingWith(PathToFile("data.xls"))
End Sub
```

If you want the path to the folder that contains the EXE, use:

```
PathToFile("")
```

19.25. I included a Word DOCX file as a companion file. How do I open it?

It's possible to include any file as a companion file: XLS Padlock will embed it into the secure application EXE file and make it available at runtime.
However, external applications are not allowed to access companion files for security reasons. For instance, if you added a DOCX file as a companion, a Word process started with CreateObject("word.Application") won't find the companion file.

The solution consists in copying the DOCX file to a temporary location and then opening it.

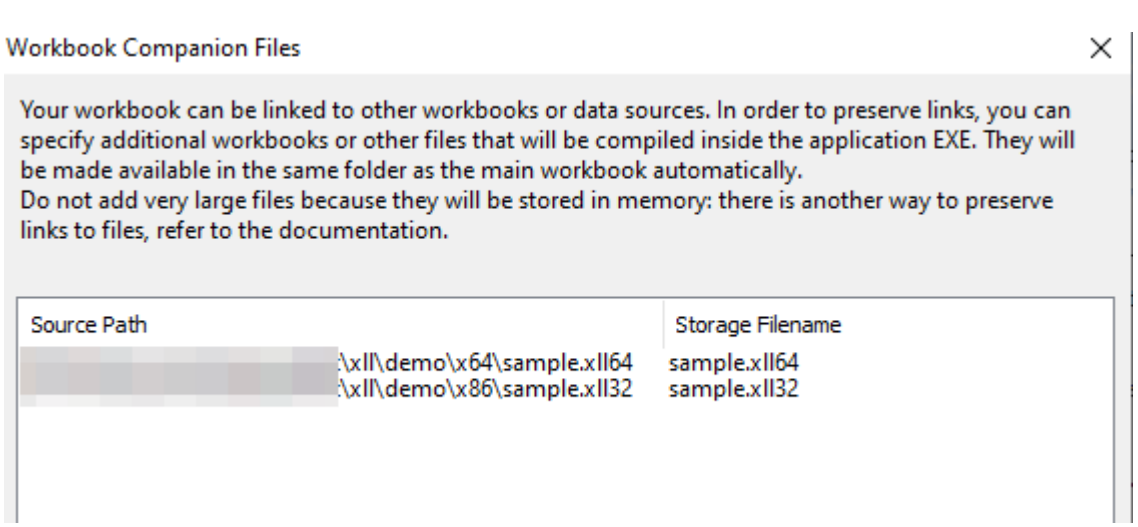
```
Sub OpenWord()
Set wordapp = CreateObject("word.Application")
FileCopy PathToCompiledFile("PADLOCKPLEASEOPENME.docx"), Environ("temp") &
"\PADLOCKPLEASEOPENME.docx"
wordapp.documents.Open Environ("temp") & "\PADLOCKPLEASEOPENME.docx"
wordapp.Visible = True
End Sub
```

The PathToCompiledFile macro has been defined in the topic about Companion Files.

19.26. How do I include XLL add-ins and register them?

It is possible to register add-ins in XLL format by listing them as companion files to your workbook.

To do this, you need to have 2 versions of your add-in: 32-bit and 64-bit formats. The filename's extensions must be modified as shown on the screenshot below, so that XLS Padlock registers the correct version with Excel at runtime:



Then, to load the add-in in your workbook, you can use the following VBA code:

```
Public Function PathToCompiledFile(Filename As String)
Dim XLSPadlock As Object
On Error GoTo Err
Set XLSPadlock = Application.COMAddIns("GXLSForm.GXLSFormula").Object
PathToCompiledFile = XLSPadlock.PLEvalVar("XLSPath") & Filename
Exit Function
Err:
PathToCompiledFile = ""
End Function

Private Sub Workbook_Open()
    Dim success As Boolean
    ' Load XLL
    success = Application.RegisterXLL(PathToCompiledFile("sample.xll"))
End Sub
```

19.27. Fixing XLS Padlock 'Workbook Modified' Warning

Question: I keep receiving the following warning when I start XLS Padlock: "The Excel workbook has been modified. You should first save it before compiling it, otherwise your changes will be lost. Are you sure you want to continue without saving?" The workbook is saved and on my desktop (not in the cloud). Can you advise on the fix?

Answer: It appears that even though your file is saved on the desktop, the desktop itself is virtual because it is linked to your OneDrive account, making the file "cloud-based," which may display this "not saved" warning. To resolve this, transfer the file from the virtual desktop to a local drive on your computer. This should correct the issue, as XLS Padlock requires the files to be stored locally, particularly during development and testing phases, to prevent issues related to cloud synchronization or virtual desktop environments.

19.28. My application says Excel not found. What should I do?

Question: My application sent to a client says Excel not found. Can you advise on the fix?

Answer:

This error may occur if the customer installed Office 365 over an older version of Office.
Tell them this:
Try performing a repair of your Office installation. If that doesn't work, use the Scrub Office utility from Microsoft, then reinstall Office 365.
👉 Get the scrub utility here.

19.29. What is the difference between activation key and activation token?

Question: In the context of XLS Padlock's WooCommerce Integration Kit, what is the difference between an activation key and an activation token, particularly in terms of their role in the activation process for compiled workbook EXE files?

Answer: Activation keys are used by end users to activate their access to the protected Excel workbook, while activation tokens are unique identifiers sent to customers, which are used by online activation to recognize the record of the customer in the database.

An activation key has this form "735DH-H12E7-DDH8F-D9BBD-BC296-F8929" while an activation token in the WooCommerce Integration Kit has thos one: "myname@myemail.com-1254".

Online activation automatically generates and activates protected workbook apps with activation keys.

□□ Activation keys can be deactivated and blocked, while an activation token will always remain valid.

19.30. How can I prevent XLS Padlock from disabling add-ins located in the XLStart folder?

Question: How can I prevent XLS Padlock from disabling add-ins located in the XLStart folder?

Answer: By default, XLS Padlock disables all add-ins in the Excel XLStart folder for security reasons. However, you can change this behavior by enabling the advanced option "Do not disable the XLStart user folder" in XLS Padlock Advanced Options. This allows add-ins in the XLStart folder to remain active while using your protected workbook.

20. About this guide

XLS Padlock User Guide

Created in January 2013 – updated in April 2025

Version 2025.0

21. Links to Support

If you have any questions or problems with XLS Padlock, you can contact us with:

- The user forum:

  https://www.gdgsoft.info

  Choose XLS Padlock as Category

- our E-Mail address:

  info@xlspadlock.com

- For Enky SL/LC dongle support:

  info@hs-securityware.com

Please provide your Excel version and if you are working with Excel 32-bit or 64-bit.

Follow us on X:

https://x.com/gdgsoft

Our other products are available at:

https://www.gdgsoft.com