Overview

Chapters:

Sometimes, you just want to ship the most optimized, most secure binaries and assets for a project to your end-users.

Also, if you don't have any particular concerns about cheating/security (such as in single-player games, or proof-of-concept projects), then the above issues may unnecessarily complicate your project.

This document assumes that you already have a working game in "Profile" configuration, complete with all the assets and the debugging of code done.

This document details shipping a CRYENGINE V based project to end-users, it is not necessary for internal development.

If you don't care about maximum performance or security, you needn't follow these steps at all. This entire document is mostly relevant for optimized use cases and for projects nearing their shipping dates.

One-time Setup

Setting up Cryptography Key(s)

Open a command prompt, and make `<root>/Tools/PakEncrypt` the current directory.

Now run: `KeyGen.exe`

This should create a `key.dat` and `key.h` file.

Never give out your key.dat file to anyone that you cannot trust.

If you lose the key.dat file, then any PAK files that have been encrypted with it can no longer be decrypted.

Crytek cannot help you to recover a lost key.dat file.

In your favorite C++ IDE open the generated `key.h` file, and also the `<root>/Code/GameSDK/GameDll/GameStartup.cpp` (or the same file in your game DLL).

Now find this line in the GameStartup file:

```
#define USE_RSA_KEY
```

Make sure that USE_RSA_KEY is set to 1.

Replace the key in `GameStartup.cpp` with the key from the generated `key.h` file.

It's not required to provide the RSA key for profile or debug configurations.

Not providing a key can reduce the initial start-up time by ~1 second, but binaries from that configuration can no longer open encrypted PAK files.

Setting up CVar Whitelist

In the file `<root>/Code/GameSDK/GameDll/GameStartup.cpp` there is a function:

```
CCVarsWhiteList::IsWhiteListed
```

Here, you can add/remove CVars that may be modified by the end-user (or via system.cfg etc).

Any CVar that is not listed will be fixed in it's default value when in Release mode.

To prevent trivial cheats, it may help to keep this list as small as possible.

However, think of the target machines of end-users and don't freeze quality CVars at too high quality levels.

Creating a Release Build

Compilation

You should compile your code in the release configuration for your target platforms.

The files will be output to `bin/<platform>_release`.

Packed Assets

Assets required by the game must be packaged into appropriate .pak files. In the simplest case, this can be achieved by zipping them and then changing the file extension. For more control over the process see Compiling Assets for Multiple Platforms.

After a release build, all assets must be inside a signed or encrypted .pakfiles.
The first test to see if your .pak files are satisfactory would be to run the non-release game after deleting all non-.pak files.

Collecting Files into Staging

ONLY copy the following files to a new folder that will contain all the files for the final version of your project (i.e. the distribution)

Files and folders marked in red should NOT be copied, they are provided just for completeness.

For the sake of an example, this will only show binaries for 64-bit Windows support. For 32-bit Windows and Linux support, substitute the appropriate platform instead of "win_x64" in the folder names below, e.g. win_x86 for 32-bit Windows or linux_x64_clang for Linux using Clang as the compiler. On Linux, note that some files will be named slightly differently (e.g. libXXX.so instead of XXX.dll).

Folder	Files	Notes
bin\win_x64_release*	**Exclude: *.pdb**	Compiled from your game code
bin\win_x64 bin\win_x64_debug		**NEVER copy these folders!** These files are debug/profile builds and not suitable for release. Additionally, the Editor is included in bin\win_x64, which you cannot redistribute to end-users according to the EULA
Engine	*.pak **EXCLUDE: *.cryasset.pak**	Shipped by Crytek in the Launcher package
GameSDK (or `sys_game_folder` CVar value)	game.cfg *.pak **EXCLUDE: *.cryasset.pak**	The exact list of .pak files will depend on how you organize your assets. *.cryasset.pak files are only for the Sandbox, and are not necessary to ship
`<sys_game_folder>`/Levels	Each level folder (e.g. Singleplayer/Woodland) may have: filelist.xml <Levelname>.dds <Levelname>.xml level.cfg level.pak Occluder.ocm <LoadingScreen>.dds (if you have defined one)	Your exported levels **Note:** Don't copy the .cry file, the terraintexture.pak, the Layers or LevelData sub-folders The files in here, with the exception of level.pak, must be inside some (signed) .pak file in order to load the levels with the Release build. However, only having the data in .pak files breaks some Editor functionality
`<sys_game_folder>`/Localization	<Language>.pak <Language>_xml.pak	The supported languages for which your project is localized
Code Editor Tools User LogBackups TestResults system.cfg	<all files and sub-folders>	These folders contain no end-user usable files, save some space by not shipping them The user folder should only have user-specific settings and will be regenerated automatically by the end-user If you have settings that you expect all users to have, it is best to set them in code rather than ship a system.cfg.

<root>	game.log editor.log server.log error.dmp	Do not ship these files: it will make it harder to identify if a log-file was user generated or not

Additional files that are specific to your project may have to be added as well, in this case you must ensure they are added in the correct location and are loadable.

Consider putting those files inside a custom .pak file, this will allow you to leverage the existing signing/encryption features.

On re-distribution - Carefully consider any legal obligations that may apply on software and assets and make sure to read license agreements etc.

Signing/Encryption of Assets

Once you have your staging folder setup you need to either sign OR encrypt your asset .pak files.

Depending on your project you should therefore pick one of the following:

- Signed assets can be loaded by your project from the .pak files, and can also be opened with other tools (as a ZIP file)
  However, the .pak file cannot be modified after it is signed (or the Engine will refuse to load the files)
- Encrypted assets can be loaded by your project from the .pak files, but cannot be opened with other tools.
  Assets inside the .pak cannot easily be accessed because they are encrypted (but can be decrypted at run-time by the Engine)

Since the Engine can decrypt the encrypted assets at run-time off-line, then this is not a 100% secure way to protect your assets if the project is intended to be run off-line.

Consider though, that if at some point an asset is displayed on screen, an attacker can recover the asset data using (for example) a graphics profiler such as RenderDoc or PIX regardless of any encryption, or read the decrypted data from main memory with a debugger.
A signed or encrypted .pak file cannot be renamed after signing/encryption, because the filename (excluding folder names) are part of the signature.

Once you have decided what level of security you want:

- Pick an output folder for this project build.
- Open a command prompt and make `<root>/Tools/PakEncrypt` the current directory (the same folder that has the `key.dat` from the one-time setup).
- Now run: `dist/ParseBuild/ParseBuild.exe <verb> "<full path to your staging folder>" "<full path to an output folder>"`
  - verb can be `sign` or `encrypt`, depending on the method that you picked.

Now the files from the staging folder will be copied from the staging folder to the output folder.

You can now delete the staging folder to recover some disk space.

As an optimization method, you can also only copy PAK files into the staging folder (keeping the correct folder structure), and copy all non-pak files directly to the output folder (with the same structure), which saves a copy operation for those files.

However, that's outside the scope of this document and since a significant relative amount of data volume is in the .pak files anyway it might not be worth the additional complexity to your build systems.

Test and Redistribute

Now you can take the contents of the output folder and test or redistribute it.

You should in general make sure that your entire project works when you directly run any of the .exe that are in the output folder and without adding any additional files.

Testing this ensures that all your code and assets are present and functional for end-users.

Try to run the project on a variety of hardware configurations to find out the quality settings for your minimum advertised hardware specifications.

If you wish to be able to debug crash-dump files that are generated by your project make sure to keep a copy of all the .pdb files that are matching the binaries that you redistribute (for each version that may be live). That way, you can load error.dmp files from end-users and have (limited) information that might point you to the cause of the crash (in addition to the (limited) log files).

These .pdb files needn't be distributed as long as you can find the matching ones when loading the end-user's dump file.

Redistributing via a specific publishing platform (i.e. Steam) is outside the scope of this document. We advise that you read the documentation provided by the publishing platform to find out how it works. In general and as long as the end-users folder structure matches the one that is described here then there shouldn't be any issues.

The same comment applies to installers of various kinds - as long as the directory structure is maintained and all the files mentioned above are present, the project should run.

It's important to also include your redistribution method in your testing process; if you publish on Steam you should test the game downloaded via Steam, if you publish with an installer, then you should also run the installer before testing. This helps in finding problems due to missing files, or files installed to a wrong folder.

Troubleshooting

Sometimes it's hard to debug issues with Release builds, because code is optimized and assets may not be easily accessible.

Here are some pointers to isolate problems:

My Game doesn't run in Release Mode

Make sure that:

- You have your .pak files signed or encrypted
- You have the correct key set in the code
- The profile/debug version of the game runs with the same assets (and the same key)

My CVars don't work in Release Mode

Make sure that:

- The CVar in question is white-listed in your code

My Assets cannot be found in Release Mode

Make sure that:

- The asset exists in a .pak file, and the .pak file is in the list of .pak files to load in the code
- The .pak file containing the asset is signed or encrypted with the correct key

I get a lot of Warnings about "Non binary XML found"

This is not a fatal error. However, you can convert your XML files to binary XML using RC, see also Binary XML conversion
The conversion is recommended for an end-user distribution

I cannot Connect to a Dedicated Server

Please note that you should connect release clients with release servers. (Also, you should connect profile/debug clients with profile/debug servers)

I have some other Problem

The default value of log_severity in release builds are set quite low. Hence, to debug an issue, then it might make sense to set the default value to 3 or 4 to get more information and messages.

Note that unless you whitelist this CVar, then setting it in system.cfg will not actually work.