

Keeping your Intellectual Property Safe with Python Software

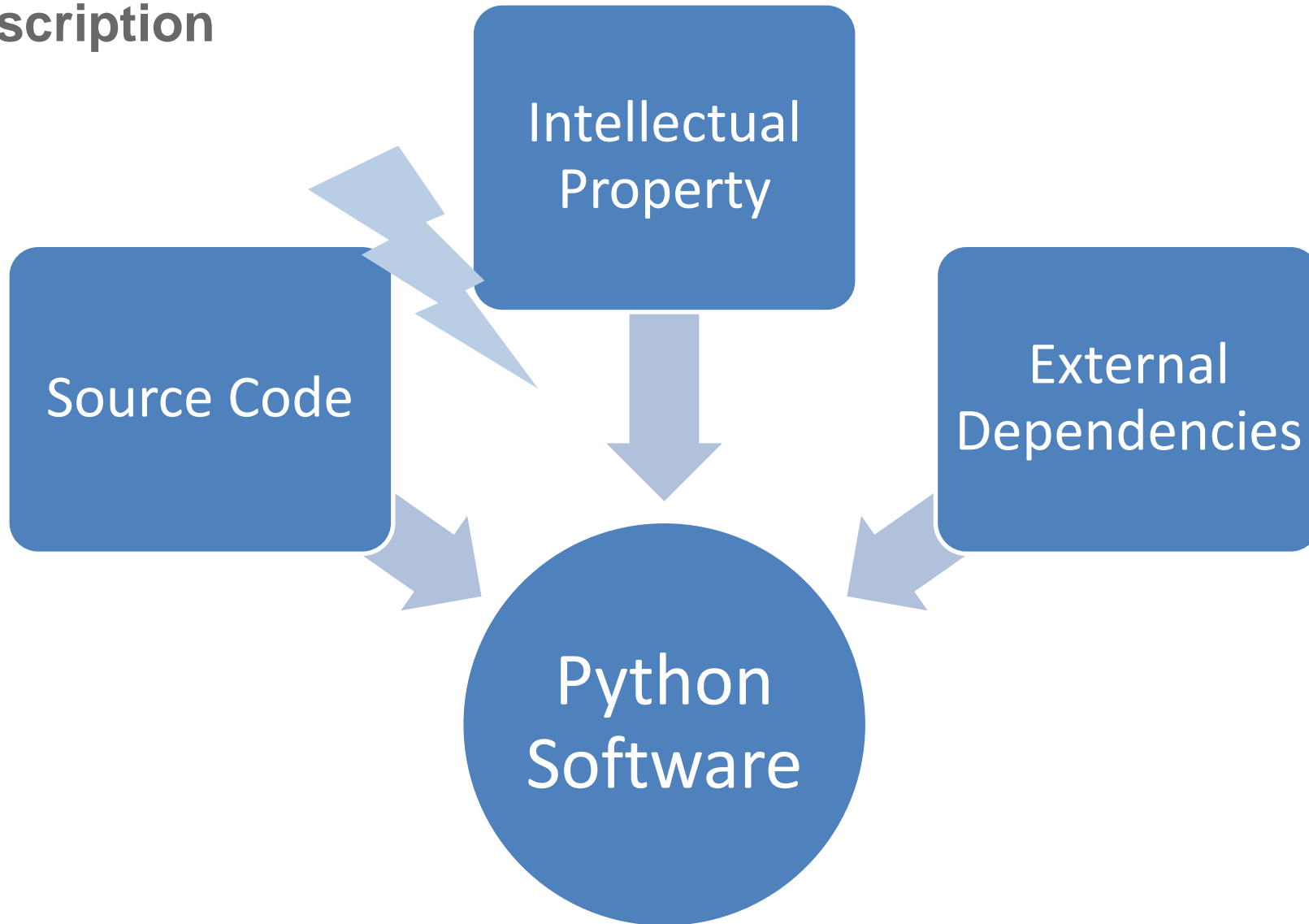
Python Users Berlin, 2022-03-10

Michael Meinel <Michael.Meinel@dlr.de>
German Aerospace Center (DLR e.V.)
Institute for Software Technology
Berlin-Adlershof

A satellite-style photograph of the Earth from space, showing the curvature of the planet, blue oceans, white clouds, and green landmasses. The image is positioned in the lower right quadrant of the slide.

Knowledge for Tomorrow

Problem Description



Disclaimer



- Most elegant way: Make it Open Source
 - ... is problematic with intellectual property
 - ... especially with small customer group
 - ... especially in science
 - ... especially for unpublished results
- Most important rule stays:
 - Don't roll your own crypto!
 - (... but I'm an expert ... somehow)



About Myself

- Dipl.-Ing. (DH) for Software Engineering
 - ... soon to be Master for IT Security
- Research Software Engineer at DLR since 2004
 - Software development
 - Trainings
 - In-house consulting

 - Programming since 1996
 - Python enthusiast
- Member of workers council
 - ... with focus on data protection



Institute for Software Technology

DLR Institute for **Software Research**,
Software Engineering, **Artificial Intelligence** and
Scientific Computation

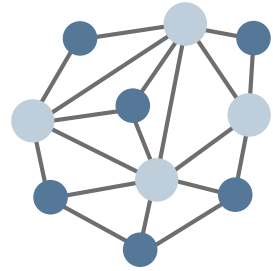
About 100 employees located at

- **Cologne-Porz**
- Berlin-Adlershof
- Brunswick
- Oberpfaffenhofen
- Bremen-Airport (ECOMAT)

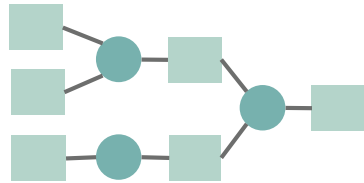
<https://www.DLR.de/sc>



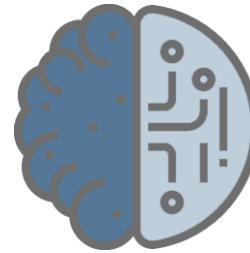
Current Research



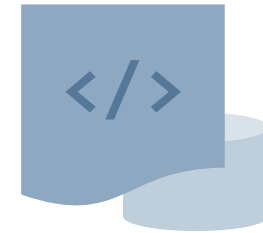
**Distributed &
Decentralized Systems**



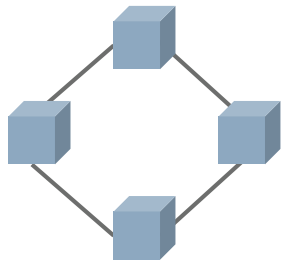
**Workflows &
Provenance**



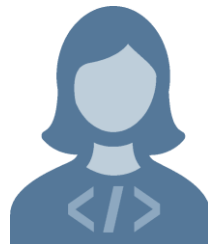
AI & Knowledge



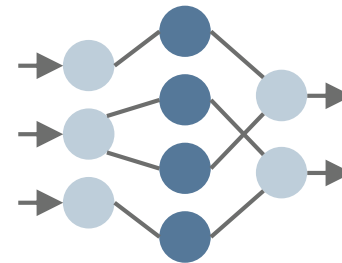
**Software
Engineering**



**Distributed Databases
& Blockchains**



**Human
Factors**



**Machine
Learning**

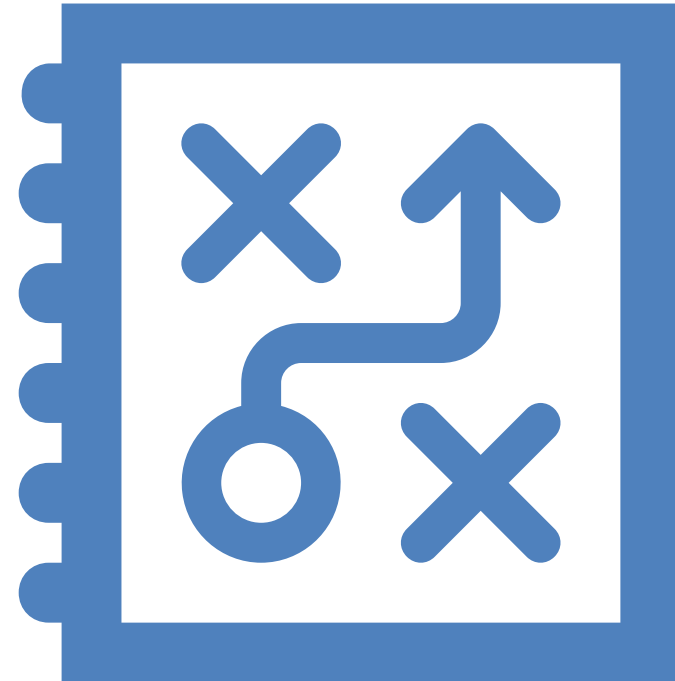


**Software
Analytics**



Outline

- Goals and strategies to keep your IP safe
- Existing solutions
 - Python byte code / Binary
 - Obfuscators
 - Encryption
- My own solution: dongle/lockup
- Discussion



Goals and Strategies

Goals

- Protect source from being read
- Enforce usage restrictions
- Protect from modification
(e. g., to deactivate license checks)
- Protect from reverse engineering and tampering
- Keep it simple, stupid

Strategies

- Compile, Obfuscate, Encrypt
- Implement license rules and checks
- Sign scripts
(and enforce signatures)
- Implement run-time checks
 - ... for running debuggers
 - ... against tampering



Compile Your Source

... using Python

- It's easy to create .pyc / .pyo from Python source.
- Platform-independent,
but depends on Python version
- Very low level of protection
→ e. g., uncompile6

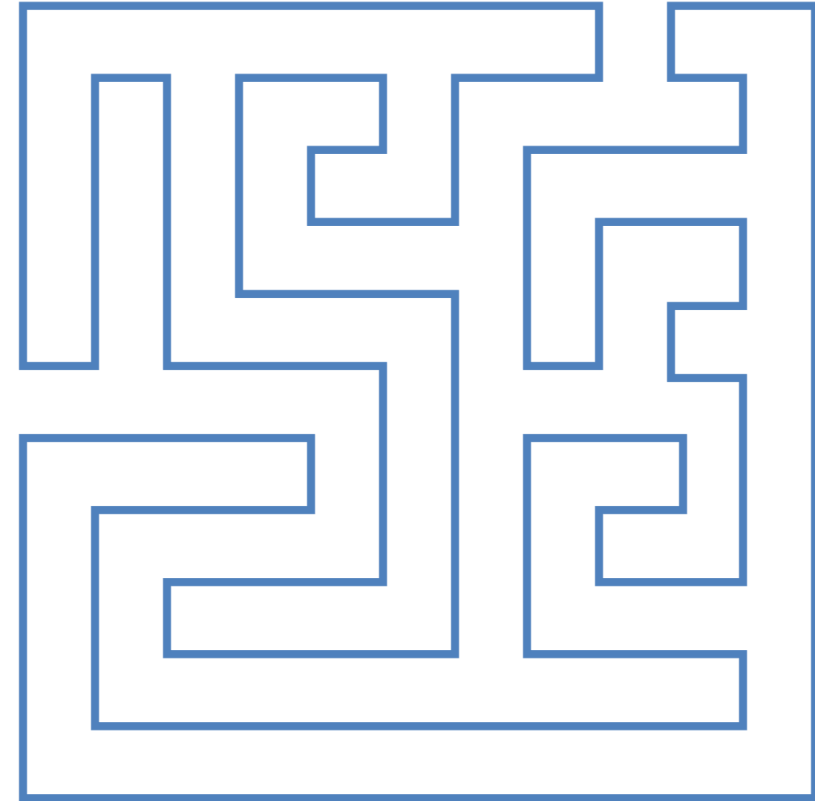
... using Cython, Nuitka

- Still rather easy...
 - (... but I did not test Nuitka yet)
- Less portable, need to build per platform.
- Less debuggable...
- Good protection for low price...
 - ... but depends on compiler –O-level
 - ... but still can be analyzed



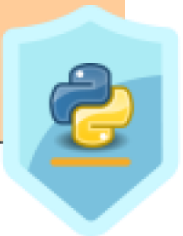
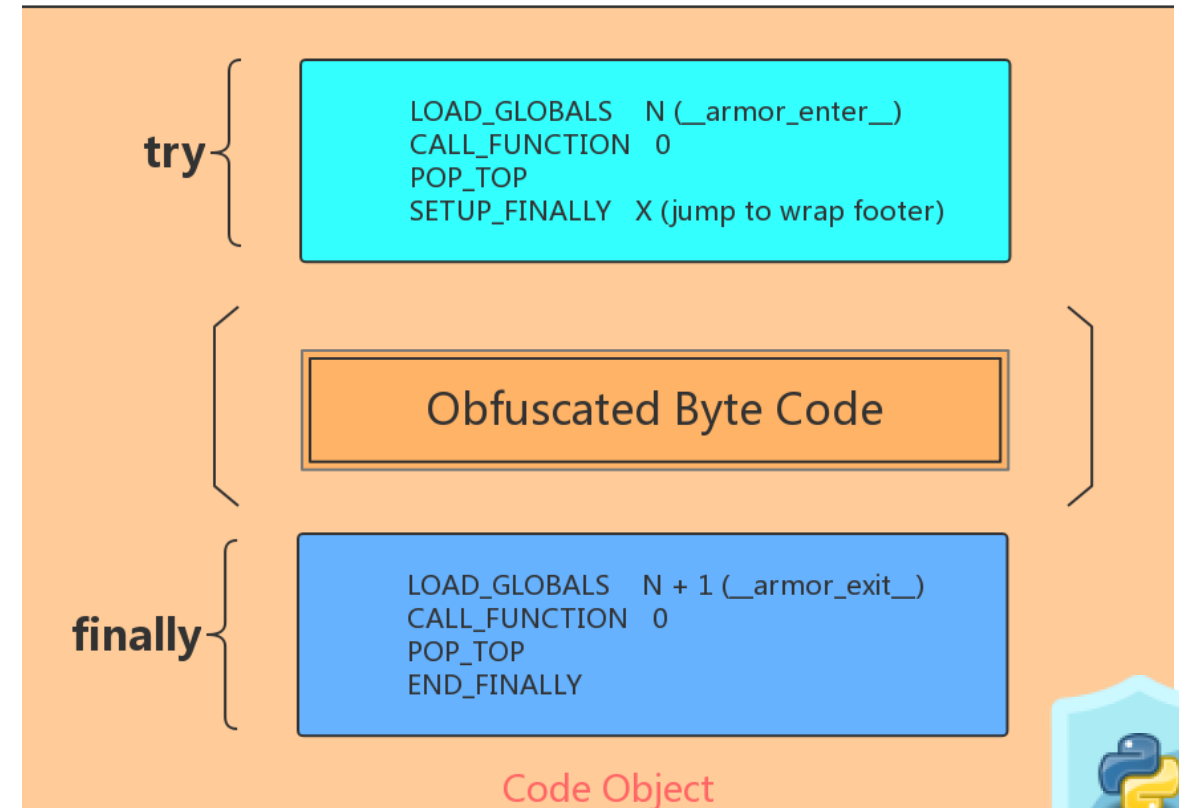
Source Code Obfuscation

- Lots of Open Source solutions available.
- Transform source code...
 - ... add red herrings
 - ... rename variables
 - ... split up lines
 - ... apply transformations (like base64, mono-alphabetic, ...)
 - ...
- Complicates problem resolution.
- Hard to read at first, but easily breakable:
 - Compile to byte code
 - Decompile
 - Run formatter / beautifier



Byte Code Obfuscation using PyArmor

- <https://pyarmor.dashingsoft.com/>
- Interesting approach:
 - Obfuscate byte code!
- Commercial product (but not expensive).
- Requires adapted runtime.
- Includes basic licensing solution.
 - Custom builds required for each license.



Source / Byte Code Encryption

- Several solutions exist...
 - ... Open Source and commercial.
- Different approaches
 - ... without asymmetric encryption
 - ... with and without customer licenses
- Most depend on binary module
 - ... platform dependent
- Some integrate transparently into Python (i. e., custom module loader).
- Other provide custom distributables.
- Only few integrate well with PEP 517.










SOURCEdefender

<https://www.sourcedefender.co.uk/>

- Commercial software (Free Trial available)
- Symmetric encryption of code with AES 256.
- Transparent integration with importer that loads *.pye files.
- Uses binary module available for broad range of platforms
- Basic licensing mechanism:
 - Can set expiration date
 - Tied to *.pye file → new translation each time
- Own distributable builder included

Features

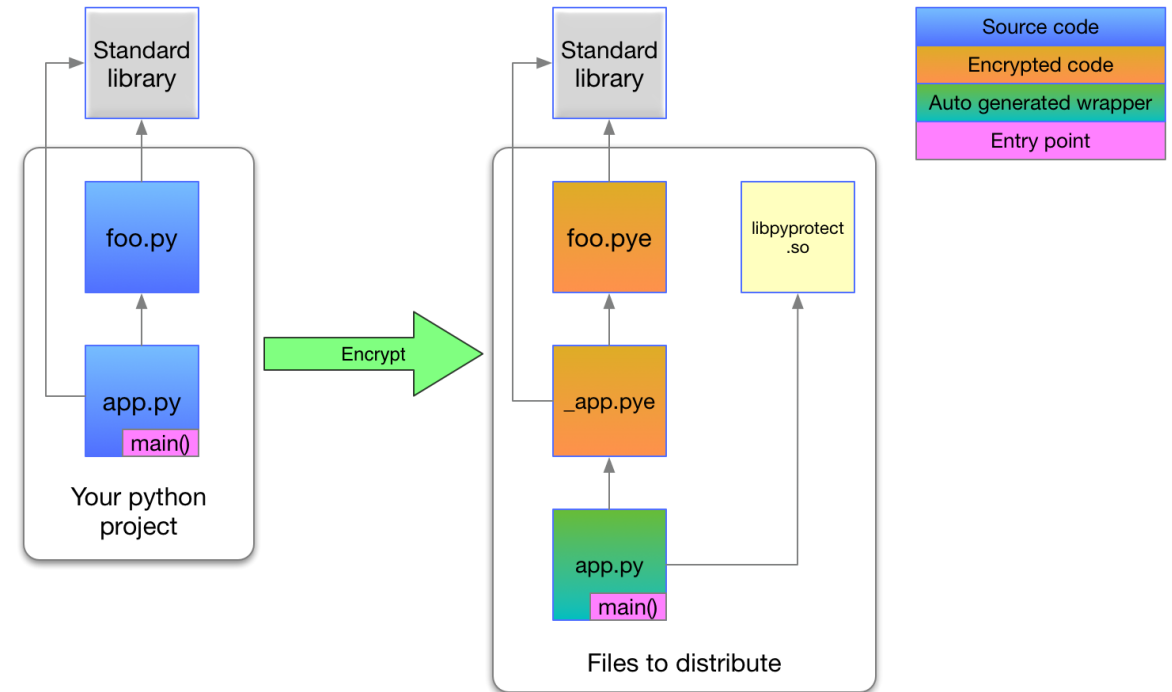
-  No end user device license required
-  Symmetric AES 256-bit encryption
-  Set your own password & salt for encryption
-  No 'Magic Number' problems changing Python version
-  Set an expiration time on encrypted code
-  Encrypted code will load on ANY supported target system
-  Bundle encrypted files or folders into a single executable binary using PyInstaller



pyprotect

<https://github.com/ga0/pyprotect>

- Open Source solution (BSD)
- Based on pybind11
- Comes with custom AES implementation
→ No extra dependencies
- Builds a custom libpyprotect.so with custom importer
- Code is stored in *.pye files (different to SOURCEdefender)
- encrypt.py as simple Interface for building
- No built-in license mechanism



pyconcrete

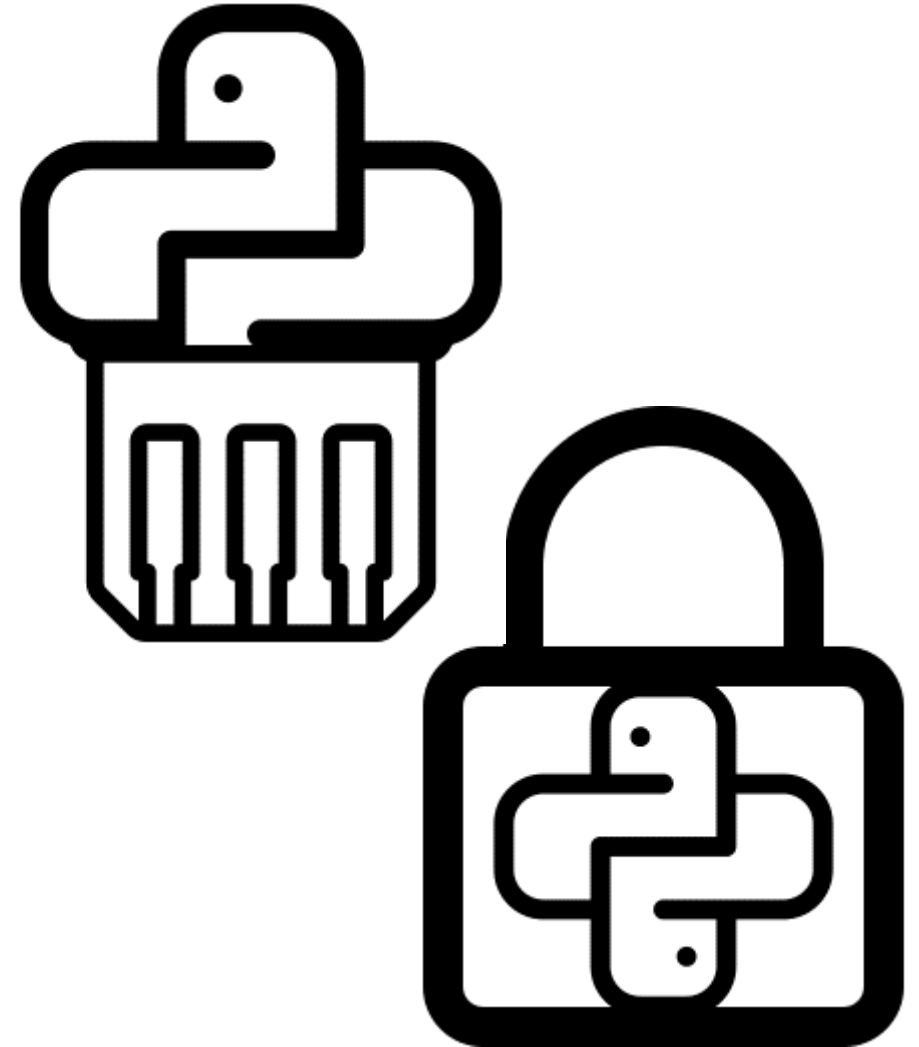
<https://github.com/Falldog/pyconcrete>

- Open Source (Apache 2.0)
 - ... with honest disclaimer
- Uses binary module and still works with 2.7
- Transparent integration
 - Produces yet another „flavor“ of *.pye files
- pyconcrete-admin.py for building encrypted mods
- Key created at installation time
- No built-in license mechanism

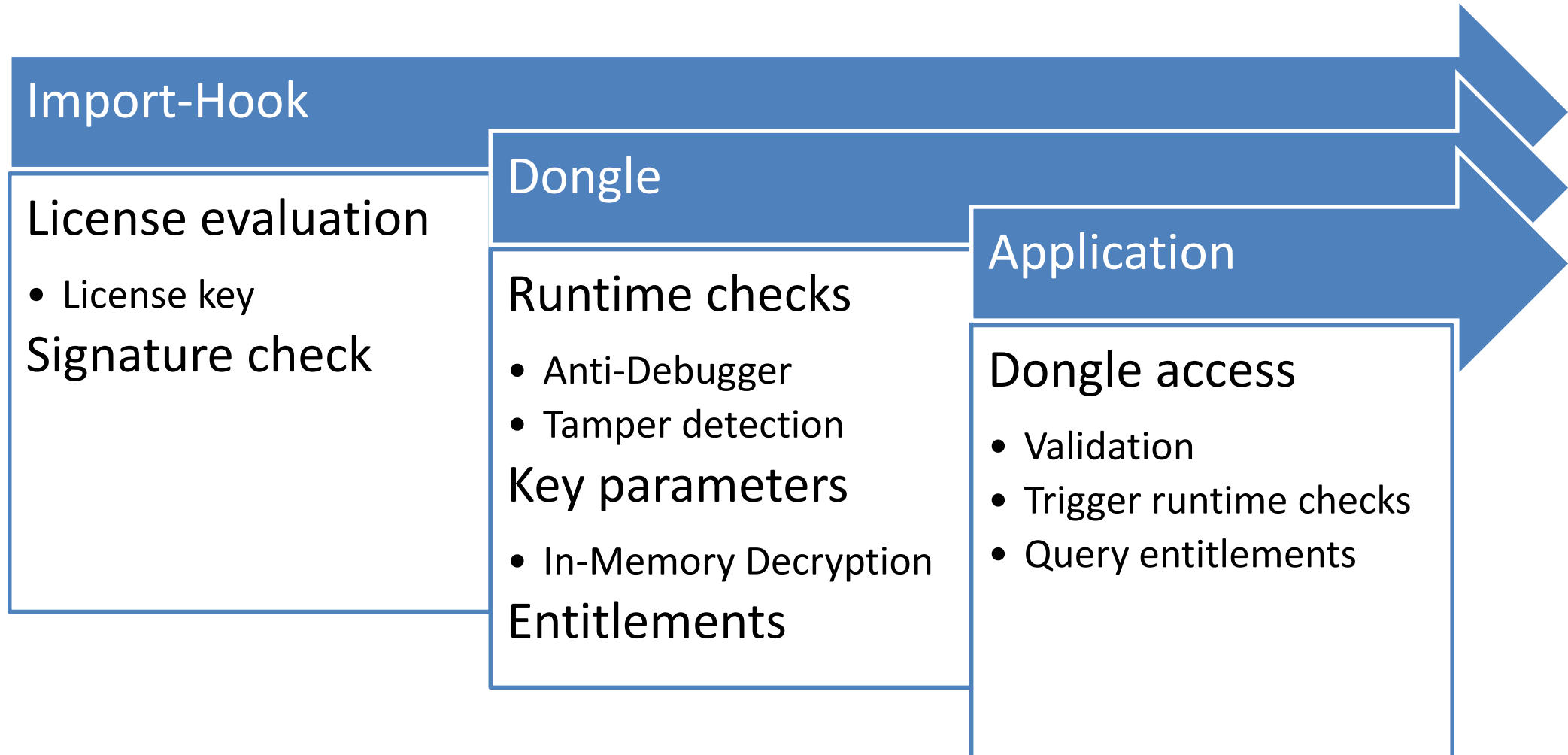


dongle/lockup

- Custom solution for DLR Technology Marketing
- Encrypted and/or signed modules
 - Possibly fully self-contained and transparent
 - License, independent of distributable
 - Implements different type of checks (extensible)
 - License conditions (node lock, expiration)
 - Environment (anti-debug, time tampering)
 - „Pure“ Python implementation (depends on „Cryptography“ package, though)
- Two parts:
 - dongle → Runtime: signed, Open Source (tbp)
 - lockup → Builder: licensed, Closed Source
 - Protected by ... dongle ;)



dongle Runtime Architecture



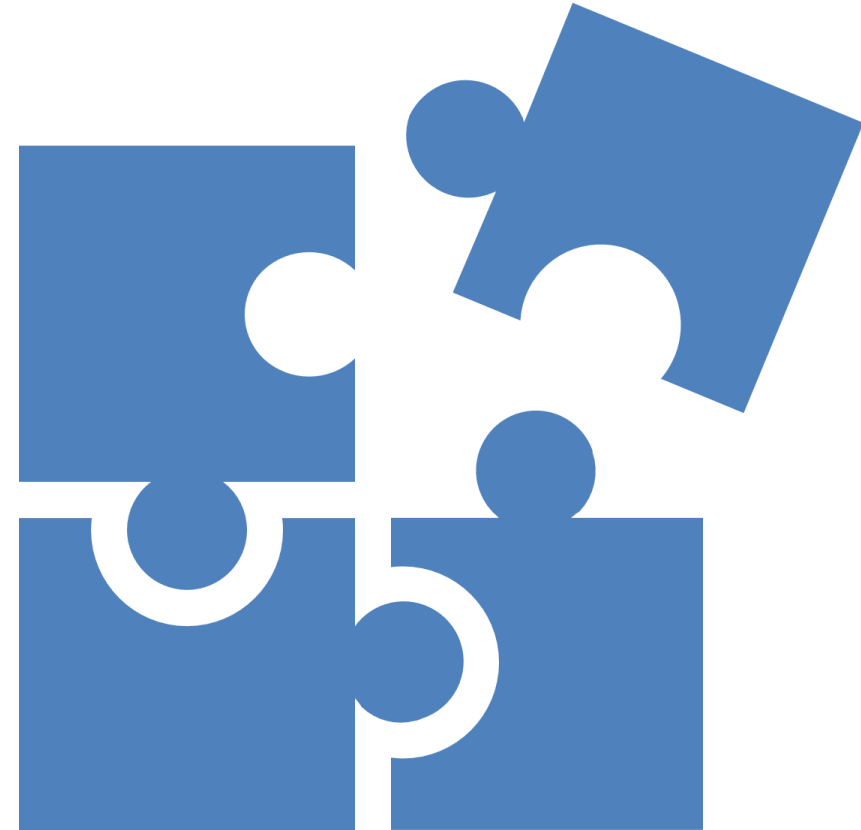
dongle Runtime

- Public key based with ECC
 - Symmetric encryption with AES
 - Using Cryptography / OpenSSL
- License independent from distributable (using a tagging system)
 - Fully offline licensing possible
 - Enforce different rules in licenses
 - Selective package licensing
- Basic mechanisms to protect access to lower layer
- Fully transparent usage possible (*.py files)
- No build tools included



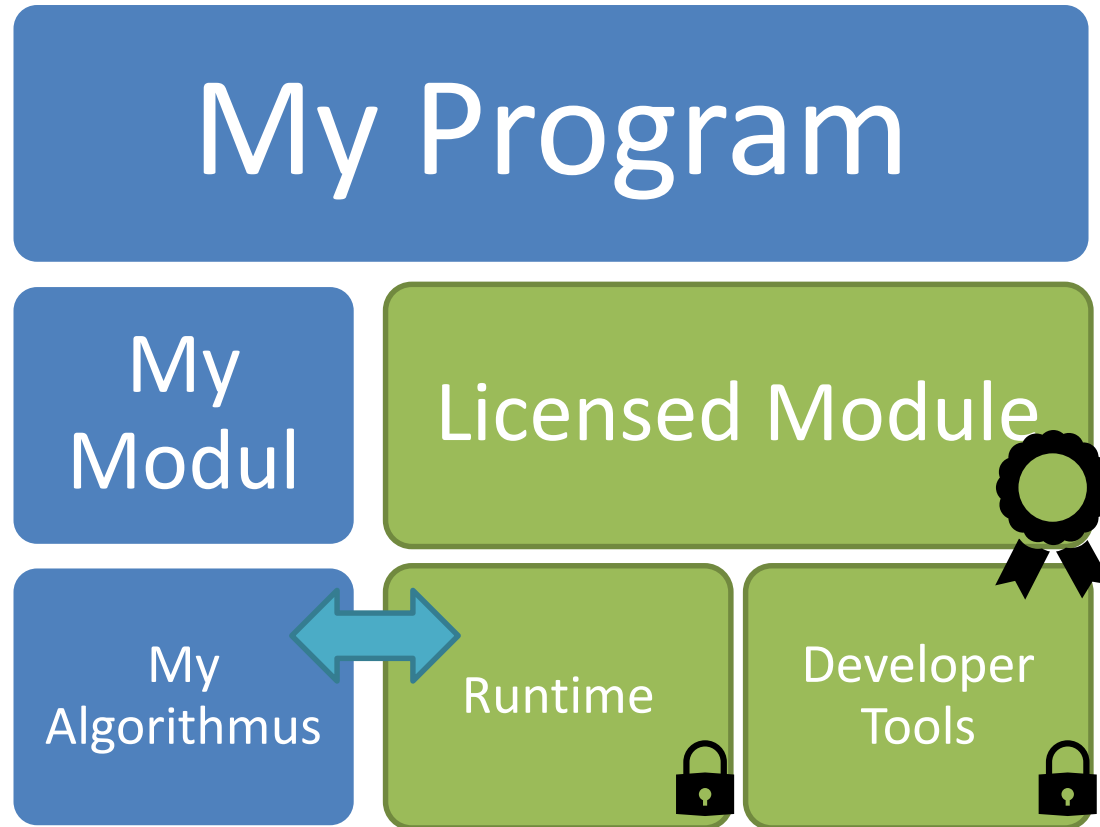
lockup License Manager

- The „missing piece“ ...
- PEP 517 build system
 - drop-in replacement for setuptools
 - (WIP for poetry-based drop-in)
- License manager to generate licenses (CLI)
- Additional checks
 - Debugger detection
 - ...
 - Sub-licensable (WIP)

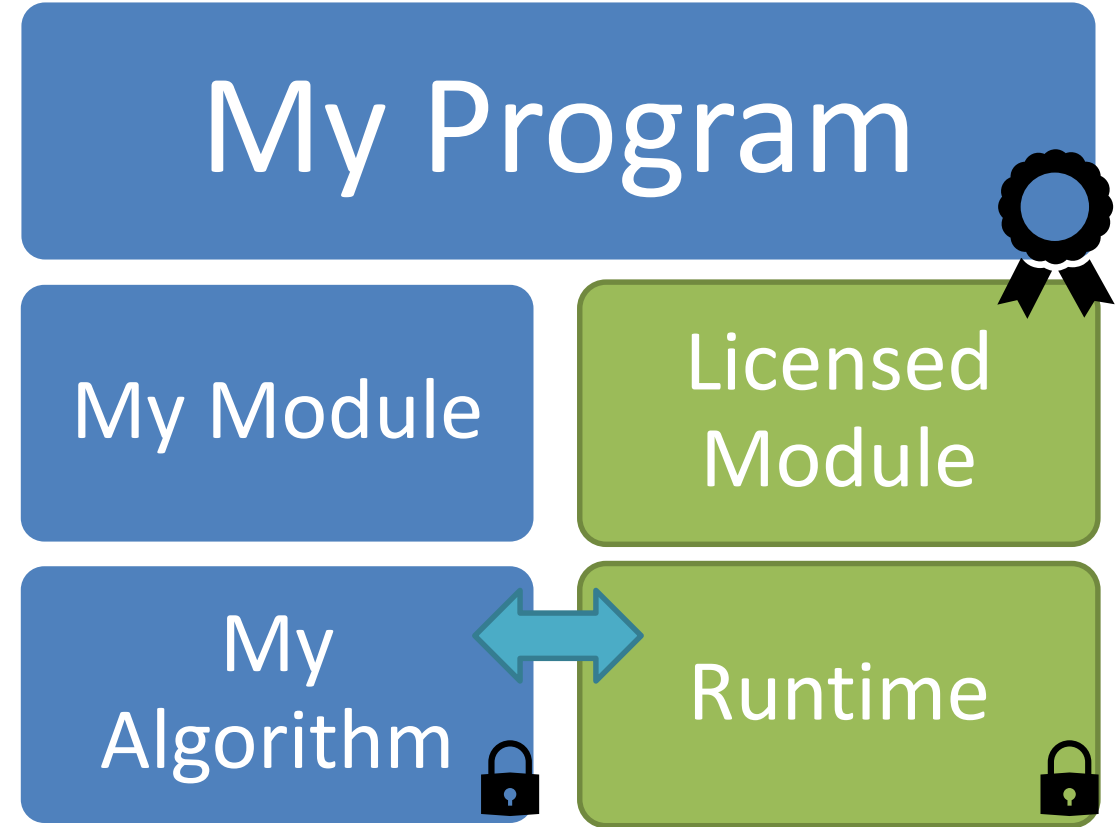


„Sub-Licensing“ Working concept

Development



Customer



Roadmap

- Fix up to make it more PEPish
 - Signed wheels (PEP 491)
 - Resource loading
 - ...
- Documentation
 - User documentation for dongle / lockup
 - Write Master Thesis
(Analysis of the Crypto System)
 - Publish dongle als Open Source (license?)
- Better User Interface for lockup
- Get funding and further users!



Question / Discussion



The Final Slide

- This presentation does not aim to be complete or exhaustive in any way.
- dongle/lockup are developed by me at the German Aerospace Center. If you are interested in this software, feel free to contact me:

Michael Meinel <michael.meinel@dlr.de>

- Image Credit:
 - Chart 11: The logo and block diagram were taken from <https://pyarmor.dashingsoft.com/>
 - Chart 13: The feature list was taken from <https://www.sourcedefender.co.uk/>
 - Chart 14: The block diagram was taken from <https://github.com/ga0/pyprotect>

