OntarioTech Business & IT

Software Protection

By

Dr. Madani

Winter 2024

Learning Objectives

Upon completion of this material, you should be able to

- Explain the importance and components of a Software License Management system.
- Describe the difference between common software license models.
- Describe the process of software license key generation and validation.
- Explain the steps in software encryption and packing.
- Describe the relevancy of Hardware Security Module in software protection.

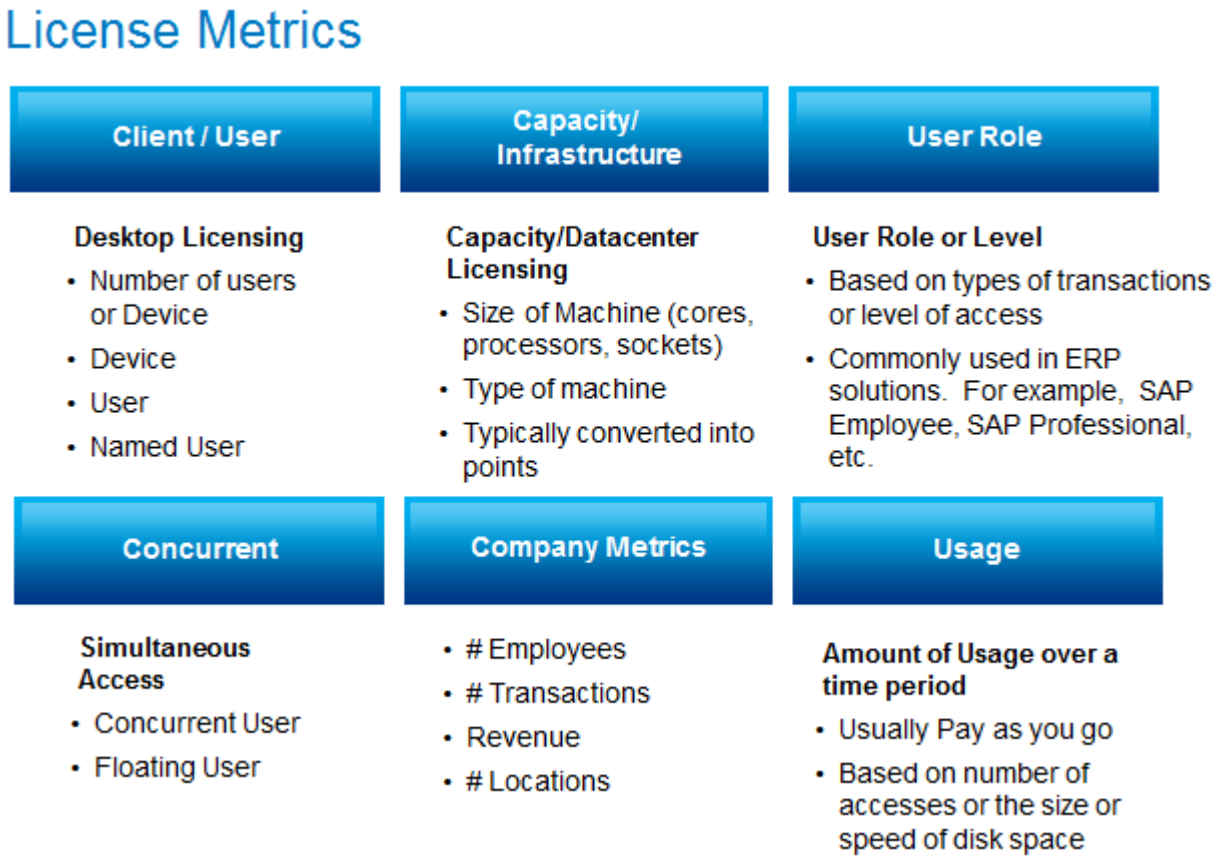
Introduction

- Software protection refers to the various methods and techniques used to prevent unauthorized access, copying, modification, or distribution of software.
  - To protect intellectual property, maintain revenue streams, and ensure software integrity.

- Methods employed in software protection that we are going to discuss:
  - **License Management**: using a licensing system that requires users to enter a unique product key.
  - **Software Encryption**: encrypting the software's executable code, making it unreadable without the correct decryption key.
  - **Code Obfuscation**: making the software code more difficult to understand by changing its structure, variable names, or control flow.
  - **Code Signing**: attaching a digital signature to the software's executable files, confirming its authenticity and integrity.
  - **Hardware-Based Protection:** involves using a physical device which must be connected to the user's computer for the software to function.

Software License

- It defines the legal terms and conditions under which the software can be used, distributed, and modified.
  - Protects intellectual property rights of software developers.
  - Limits the liability of the software vendor!
  - Limits what other parties can do with the software and software code.
  - Specifications a license might include:
    - How many times the software can be downloaded?
    - What the software will cost?
    - What level of access users will have to the source code!



Different Software License Models

https://www.flexera.com/blog/it-asset-management/software-license-models-101-2/

Software License – cont'd

- Types of Software Licenses:
  - **Proprietary**: These licenses grant specific usage rights to the end-user but typically restrict access to the software's source code. The developer retains all rights to the software, and users are generally not allowed to modify, distribute, or reverse-engineer the software. Examples of proprietary software include Microsoft Windows, Adobe Photoshop, and Apple's macOS.
  - **Freeware**: is software that is available free of charge but may still be subject to copyright and licensing restrictions. Users can download and use the software without paying, but they may not have the right to access the source code or distribute the software. Examples of freeware include Adobe Acrobat Reader and Skype.
  - **Shareware**: is a type of software licensing where the user is allowed to try the software for free for a limited time or with limited functionality. After the trial period, the user is typically required to purchase a license to continue using the software or to unlock its full features. Examples of shareware include WinZip and WinRAR.
  - **Open-source**: allow users to access, modify, and distribute the software's source code. There are many open-source licenses, each with its own set of terms and conditions. Some common open-source licenses include the GNU General Public License (GPL), the MIT License, and the Apache License. Examples of open-source software include the Linux operating system, the Apache HTTP Server, and the Python programming language.

Software License – cont'd

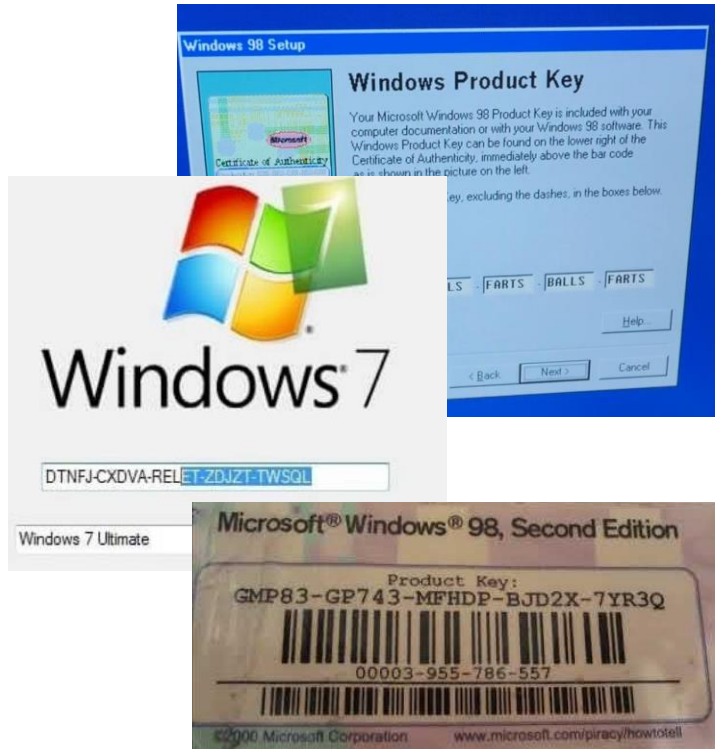
- Types of Software Licenses (cont'd):
  - **Creative Common**: although not specifically designed for software, Creative Commons licenses are sometimes used to license software projects. These licenses grant various levels of permissions and restrictions for use, distribution, and modification of the work.
  - **Site**: these licenses are designed for organizations that need multiple copies of the software for their employees or members. Site licenses typically cover an unlimited number of installations within a single physical location, while volume licenses cover a specified number of installations.
  - **Subscription**: require users to pay a recurring fee, usually monthly or annually, to continue using the software. This model has become more popular with the rise of software as a service (SaaS) and cloud-based applications.

Software License – cont'd

- Software License Management: ensures that the software is used in compliance with the licensing agreement.
  - involves administering, monitoring, and enforcing the usage and distribution of software based on the terms and conditions set by the software publisher or developer.
  - Key Components:
    - **License Model**: discussed in the previous slides!
    - **License Key**: a unique code or string that is generated by the publisher and is required for activating or installing the software. serves as proof of purchase and a way to control the number of installations or users allowed for a specific software product.
    - **Activation**: is the process of verifying the license key and granting the user access to the software.
    - **License Tracking, Audit and Enforcement** : periodic audits can be performed by organizations or third-party auditors to ensure that the software is being used according to the licensing terms.
    - **Usage Analytics** : analyzing software usage data, organizations can optimize their software investments, identify underused or unused licenses, and make informed decisions about future software purchases or renewals.

Software License – cont’d

- License Key Generation and Validation
  - Is the process of creating unique keys or codes that are used to activate, authenticate, or verify the licenses of software applications.
  - Key generation is a crucial component of software licensing and protection systems.
  - It's essential to consider the security requirements, target platforms, and user experience.
  - A well-designed key generation system can provide robust protection against unauthorized access and software piracy while minimizing the inconvenience and complexity for legitimate users.



Software License – cont'd

- License Key Generation and Validation (cont'd)
  - Aspects to consider:
    - **Generation Algorithm**: typically (not always!) involves the use of algorithms, which can be either deterministic or random, depending on the desired level of security and complexity. Common algorithms used for key generation include cryptographic hash functions (e.g., SHA-256), symmetric encryption algorithms (e.g., AES), or asymmetric encryption algorithms (e.g., RSA).
      - The generation process highly depends on activation and validate process [why? Class discussion]. Is it going to be validated in an offline or online setting?
    - **Key Length and Complexity**: play a significant role in determining its security. Longer and more complex keys are harder to guess or crack, providing better protection for the software. However, longer keys may be more difficult for users to enter or manage, so a balance must be struck between security and usability.

Software License – cont’d

- License Key Generation and Validation (cont’d)
  - Aspects to consider:
    - **Unique Identifiers**: ensure that each key is unique, some key generation algorithms incorporate unique identifiers, such as user or device information, purchase transaction details, or a combination of these. This helps prevent key duplication or unauthorized sharing.
    - **Activation Limits and Expiration**: keys can be generated with built-in activation limits or expiration dates, which restrict the number of times a key can be used or the duration for which it is valid. This can help prevent unauthorized sharing or use of the software beyond the terms of the license.
    - **Key Validation and Verification**: the software or the publisher's server must validate and verify the key to ensure its authenticity. This can involve checking the key's structure, performing cryptographic operations, or comparing the key against a database of valid keys (on the software vendor’s online database!).

Software License – cont'd

- Example of Key Generation/Verification in Windows 95! [I know …]

Windows 95 retail key

Windows 95 retail product keys take the form XXX-XXXXXXX.[2] To determine whether the key is valid, Windows 95 performs the following checks:

- The first 3 characters must not be equal to 333, 444, 555, 666, 777, 888 or 999.
- The last 7 characters must all be numbers from 0-8.
- The sum of the last 7 numbers must be divisible by 7 with no remainder.
- The fourth character is unchecked.

If all checks pass, the product key is valid.

https://en.wikipedia.org/wiki/Product_key

Software License – cont’d

- License Key Generation and Validation (cont’d)
  - Aspects to consider:
    - **Key Storage and Management**: once generated, software keys must be stored securely and managed effectively, both by the publisher and the user. Publishers must protect their key databases from unauthorized access or tampering, while users must securely store their keys to prevent loss or theft.
    - **Key Distribution**: keys must be distributed to users in a secure and efficient manner, such as via email, through a user account portal, or as part of a physical product package. Publishers must ensure that the distribution process maintains the confidentiality and integrity of the keys.

Software Encryption

- Software encryption (sometimes referred to as “Software Packing”) is a technique used to protect software applications by converting the original code and data into an unreadable format using encryption algorithms.
  - To prevent unauthorized access, reverse engineering, tampering, or piracy of the software!
  - Only users with the appropriate decryption key can convert the encrypted software back into its original form and execute it.

FULL PROGRAM ENCRYPTION

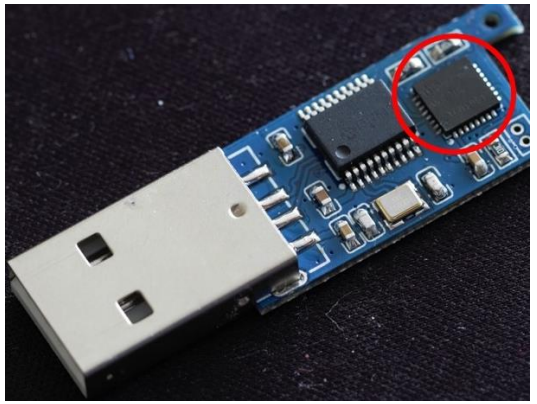
- The entire executable file, including the code, data, and resources, is encrypted.
- The software can only be executed after the user provides the correct decryption key, which is typically done during the installation or activation process.
- The entire program needs to be decrypted before execution.

FULL PROGRAM ENCRYPTION

- Only specific parts of the software are encrypted, such as critical algorithms or sensitive data.
- This technique is more efficient in terms of performance, as only the encrypted portions need to be decrypted during runtime.
- However, some parts of the software remain in plain text and could be analyzed or tampered with by an attacker.

Software Encryption – cont'd

- Steps in Software Encryption:
  1. **Selection of an encryption algorithm**
     - Common symmetric (e.g., AES or DES) and/or asymmetric (RSA). Target platform for decryption must be considered!
  2. **Key generation**
     - Used as an input to the encryption algorithm. It could be related (e.g., equal) to the customer product key.
     - It can be stored and shipped in a hardware device (e.g., dongle) that must be always connected to the executing computer!
  3. **Encryption**
     - Partial or Full encryption of the program code and resource data.
  4. **Decryption and Execution**
     - Using correct decryption key (that may be coming from a dongle), the software is then decrypted and executed on the user's system.



Software Encryption – cont'd

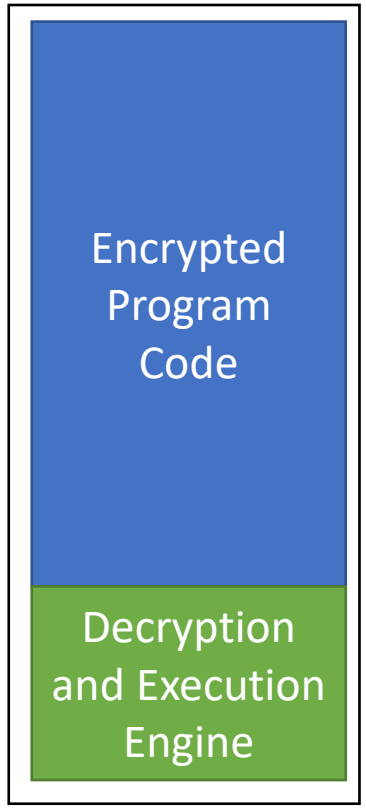
- Example

This is an actual python program (code) – encrypted!

```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from base64 import b64encode, b64decode
from binascii import unhexlify

source_encrypted = '1EHOr0VZBlmZyzL1HuLY4T/J79v6NPiUExoRCSfVJGraKpPKz22URaI6eO8wI84

iv = "7bde5a0f3f39fd658efc45de143cbc94"
iv = unhexlify(iv)

cipher = AES.new(b'3e83b13d99bf0de6c6bde5ac5ca4ae68', AES.MODE_CBC,iv)

decrypted_source  = unpad(cipher.decrypt(b64decode(source_encrypted)),
                          AES.block_size).decode('utf-8')

exec(decrypted_source)
```

Decrypt the content to get the actual python code as a "string"

This will **execute** the decrypted python "string"



Encrypted Program

Code Obfuscation

- Code obfuscation is a technique used to make software code more difficult to read, understand, and reverse-engineer.
  - Making the code more complex and less comprehensible, obfuscation deters hackers and other malicious actors from understanding the inner workings of the software, stealing proprietary algorithms, or discovering vulnerabilities that can be exploited.
  - You can also use to add extra layer of security to further protect:
    - the license management logics in your software.
    - The software encryption/decryption logics in your software.
  - Can provide a valuable layer of protection for software but it is not foolproof.

Code Obfuscation – cont'd

- Commonly used techniques:
  - **Identifier renaming**
    - This technique involves renaming variables, functions, and classes with meaningless or confusing names. This makes it harder for someone analyzing the code to understand its purpose or function.
  - **Control flow obfuscation**
    - Control flow obfuscation alters the structure and flow of the code to make it more difficult to follow. This can involve reordering, splitting, or merging functions, adding dead or redundant code, or using opaque predicates (conditional statements with outcomes that are difficult to predict).
  - **Data obfuscation**
    - Data obfuscation involves encrypting or encoding data, such as strings or numbers, within the code. This makes it harder for an attacker to identify patterns or extract sensitive information from the code.
  - **Code encryption**

Code Obfuscation – cont’d

- Commonly used techniques:
  - **Anti-debugging and anti tampering techniques**
    - These techniques are designed to hinder debugging or reverse-engineering efforts. They can include checks for the presence of debuggers, checksums to detect code modifications (code signing could also be used), or time-based or environmental triggers that alter the software's behavior when certain conditions are met.
  - **Function inlining or outlining**
    - Inlining involves combining multiple functions into a single function, while outlining involves breaking a single function into multiple smaller functions. Both techniques can be used to make the code more challenging to analyze and understand.

Code Obfuscation

- Example of Identifier Renaming

Sample source code (input)

```
source = '''
class ComputeTax(object):
    def SetupAccount(self):
        self.var1 = "some value"
        self.var2 = 1
    def ComputeMonthlyReturn(self):
        var3 = 2
    def FileWithCRA(self, var):
        var4 = var
'''
```

```
import ast
import secrets

def get_identifiers(source):
    root = ast.parse(source)

    for node in ast.walk(root):
        if isinstance(node, ast.Name) and isinstance(node.ctx, ast.Store):
            yield node.id
        elif isinstance(node, ast.Attribute):
            yield node.attr
        elif isinstance(node, ast.FunctionDef):
            yield node.name

ids = list(get_identifiers(source))
ids_replace = {id:'a' + secrets.token_hex(5) for id in ids}

obfuscated_src = source
for id in ids:
    obfuscated_src = obfuscated_src.replace(id,ids_replace[id])

print(obfuscated_src)
```

Python Identifier Obfuscator

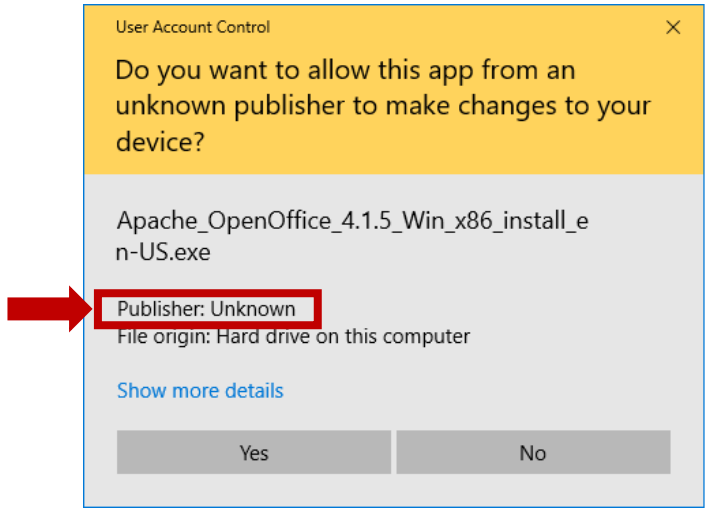
```
class ComputeTax(object):
    def a9a15925bbe(self):
        self.af4eb2f1164 = "some value"
        self.a53d04b70fb = 1
    def ace092876a4(self):
        aded2bc9257 = 2
    def aa3d17ce214(self, var):
        a3709761b9a = var
```

Sample source code (out)

OntarioTech Business & IT

Code Signing

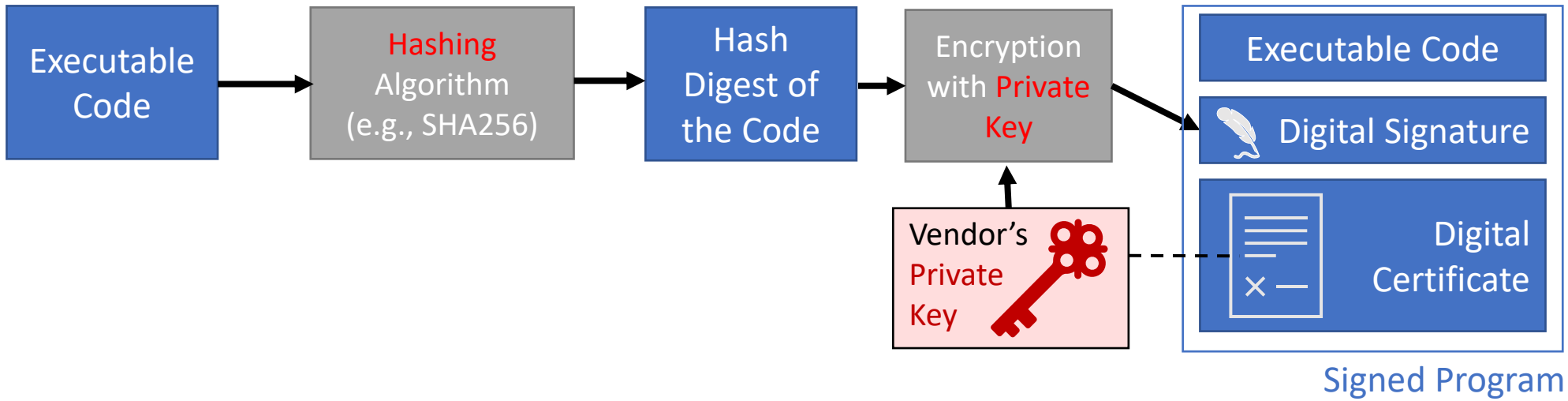
- Code signing is a technique used to verify the authenticity and integrity of software by attaching a digital signature to the executable files or scripts.
  - Ensures that the software has not been tampered with and comes from a trusted source.
  - Is especially important for software distributed over the internet, where the risk of malware or other malicious alterations is higher.
    - E.g., You cannot install unsigned apps on your iPhone (not jailbroken!)
  - Code signing does not guarantee that the software is free from vulnerabilities or malicious code, but it provides a crucial layer of trust and security in the software distribution process.



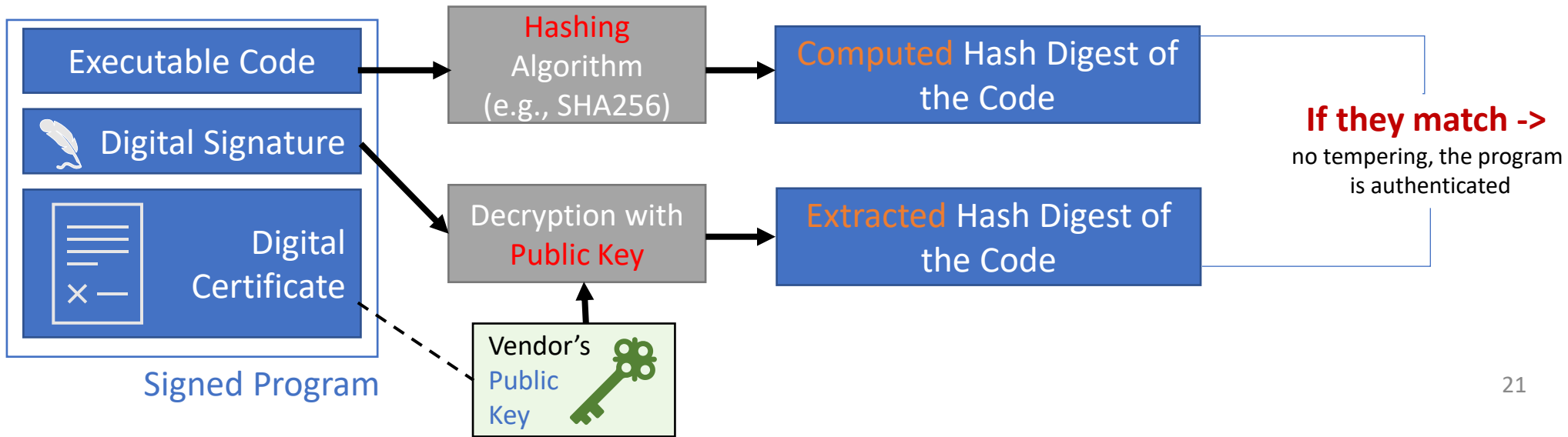
https://www.computer.org/publications/tech-news/trends/what-is-a-code-signing-certificate

Code Signing – cont'd

- Program Signing Steps:



- Program Verification Steps:



Code Signing – cont’d

- Program Signing Steps:
  1. **Obtain a code signing certificate**: The software publisher or developer needs to obtain a code signing certificate from a trusted Certificate Authority (CA). This certificate contains the publisher's public key and is digitally signed by the CA. The CA verifies the identity of the publisher before issuing the certificate, thus providing a level of trust in the software's origin.
  2. **Generate a hash of the software**: The publisher creates a hash (or a cryptographic digest) of the software's executable files or scripts using a cryptographic hashing algorithm, such as SHA-256. This hash serves as a unique representation of the software's content and will change if any part of the software is altered.
  3. **Sign the hash with a private key**: The publisher then signs the generated hash using their private key, which is kept secret and secure. This creates a digital signature that is unique to the software and the publisher.
  4. **Attach the digital signature to the software**: The digital signature, along with the code signing certificate containing the public key, is attached to the software or embedded in its metadata. The software is now considered "signed" and ready for distribution.

Code Signing – cont’d

- Program Verification Steps:
  1. Extract the digital signature and the code signing certificate from the software.
  2. Use the public key in the certificate to decrypt the digital signature, revealing the original hash of the software.
  3. Independently generate a new hash of the downloaded software using the same hashing algorithm.
  4. Compare the decrypted hash (from the signature) and the newly generated hash. If they match, it means the software has not been tampered with since it was signed.
  5. Verify that the code signing certificate is valid and comes from a trusted CA. This ensures that the software originates from a legitimate publisher.

Hardware-based Protection

- Hardware-based protection for software involves using physical devices or components to secure and manage access to the software.
  - More expensive and complex to implement and may require specialized hardware or infrastructure.

- Examples:
  - **Dongles**: A dongle is a small hardware device that connects to a computer via USB or another interface. The dongle stores licensing information, encryption keys, or other sensitive data required for the software to function. The software checks for the presence of the dongle and validates its contents before allowing the user to access the software. The software cannot be used without the corresponding dongle.

  - **Smart cards**: Smart cards are portable devices, similar to credit cards, that contain an embedded microprocessor and memory. They can store cryptographic keys, certificates, or other sensitive data and perform cryptographic operations. Smart cards can be used for software licensing or authentication, ensuring that only authorized users can access the software.

Hardware-based Protection – cont'd

- Examples (cont'd):
  - **Hardware Security Modules (HSMs)**: is a physical device that provides secure storage of cryptographic keys, as well as cryptographic operations such as encryption, decryption, and digital signature generation. HSMs are designed to be tamper-resistant.
    - Services provided:
      1. **Secure Key Storage**: store encryption keys and other sensitive information within a secure, tamper-resistant environment, preventing unauthorized access or extraction. This can be particularly useful for protecting software licensing keys, cryptographic keys used for software encryption, or keys used in digital rights management (DRM) systems.
      2. **License enforcement**
      3. **Code signing**
      4. **Secure cryptographic operations**: can perform cryptographic operations such as encryption and decryption **within their secure environment**. Thus, cryptographic keys will remain inside their internal memory and not moved into the main memory!
      5. **Secure boot and firmware protection**

Hardware-based Protection – cont'd

- Two Special HSM
  - **Trusted Platform Modules (TPMs):** TPMs are specialized microcontrollers or chips embedded in a computer's motherboard that provide hardware-based security features. TPMs can be used to store cryptographic keys, certificates, or other sensitive information, ensuring the integrity and confidentiality of the stored data. TPMs can also be used for secure boot processes, verifying the integrity of the software and operating system before execution.

  - **Secure Enclaves**: Secure enclaves, such as Intel's Software Guard Extensions (SGX) or ARM's TrustZone, are hardware-based security features built into modern processors. These secure enclaves provide isolated execution environments that protect sensitive data and code from unauthorized access, even if the operating system or other software is compromised. Secure enclaves can be used to store encryption keys, licensing information, or other sensitive data needed for software protection.

    - Very common in today's Smart Phone devices and fully supported by iOS and Android:
      - https://support.apple.com/en-ca/guide/security/secf020d1074/web