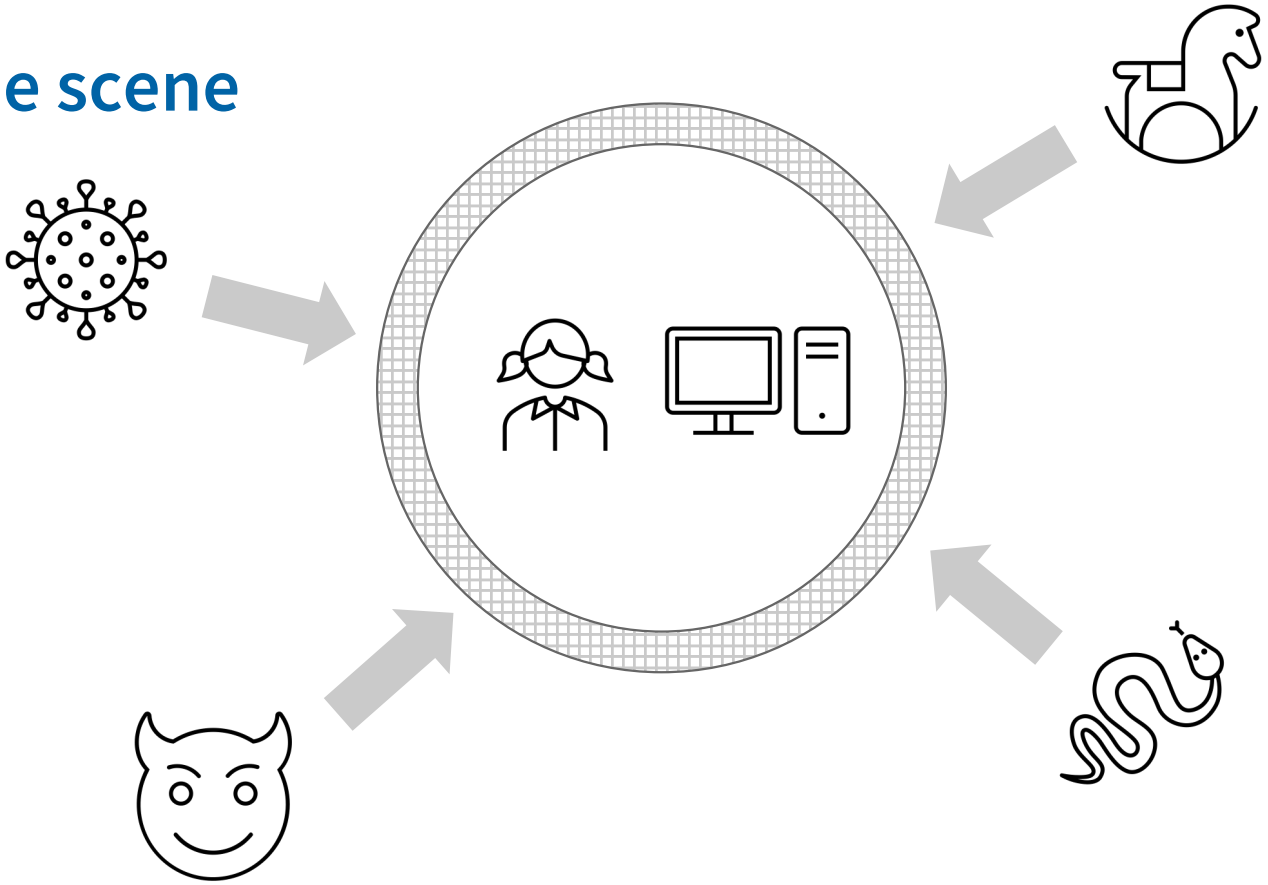
universität wien

# Software Protections

## Theory, practice, and recent advances

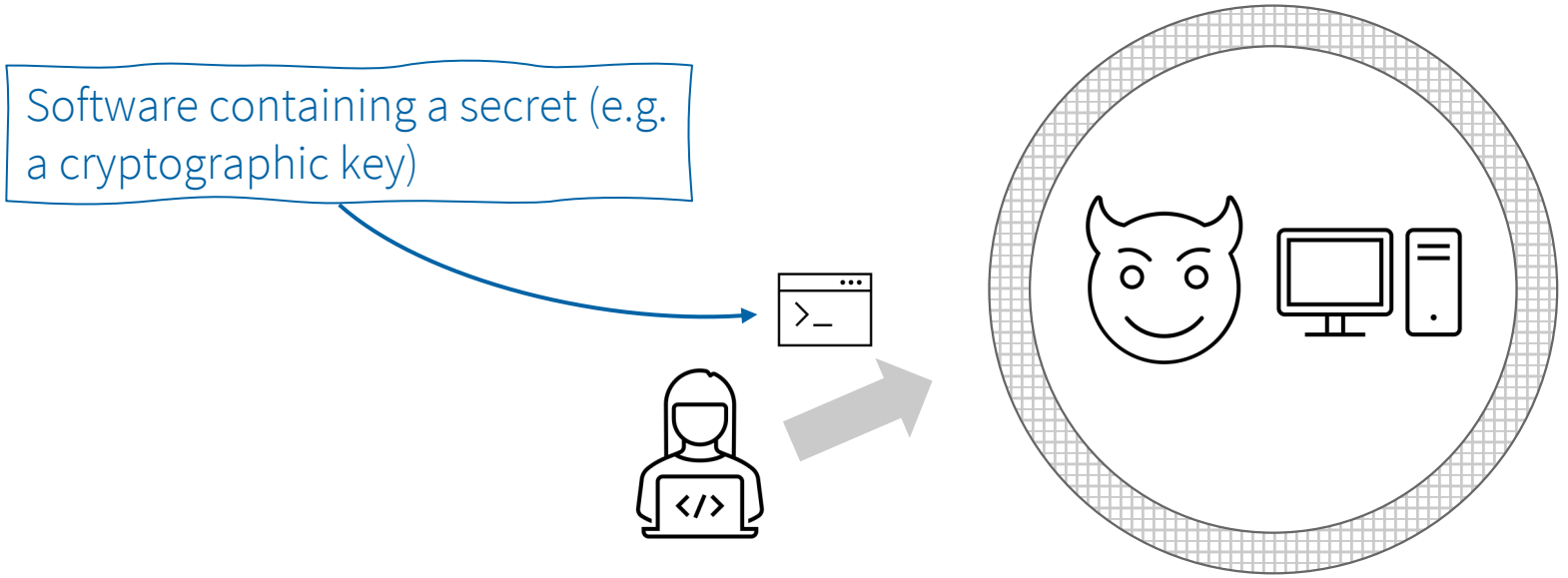
universität wien

# Setting the scene



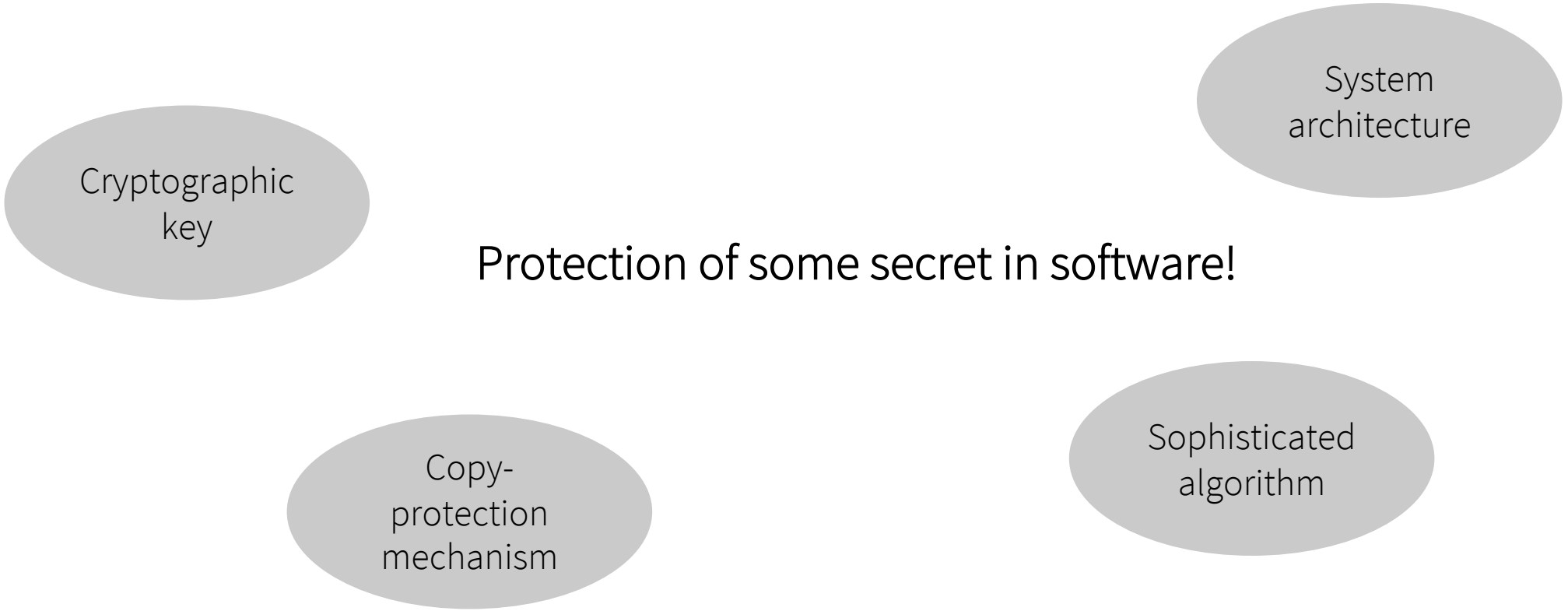
universität wien

# Setting the scene



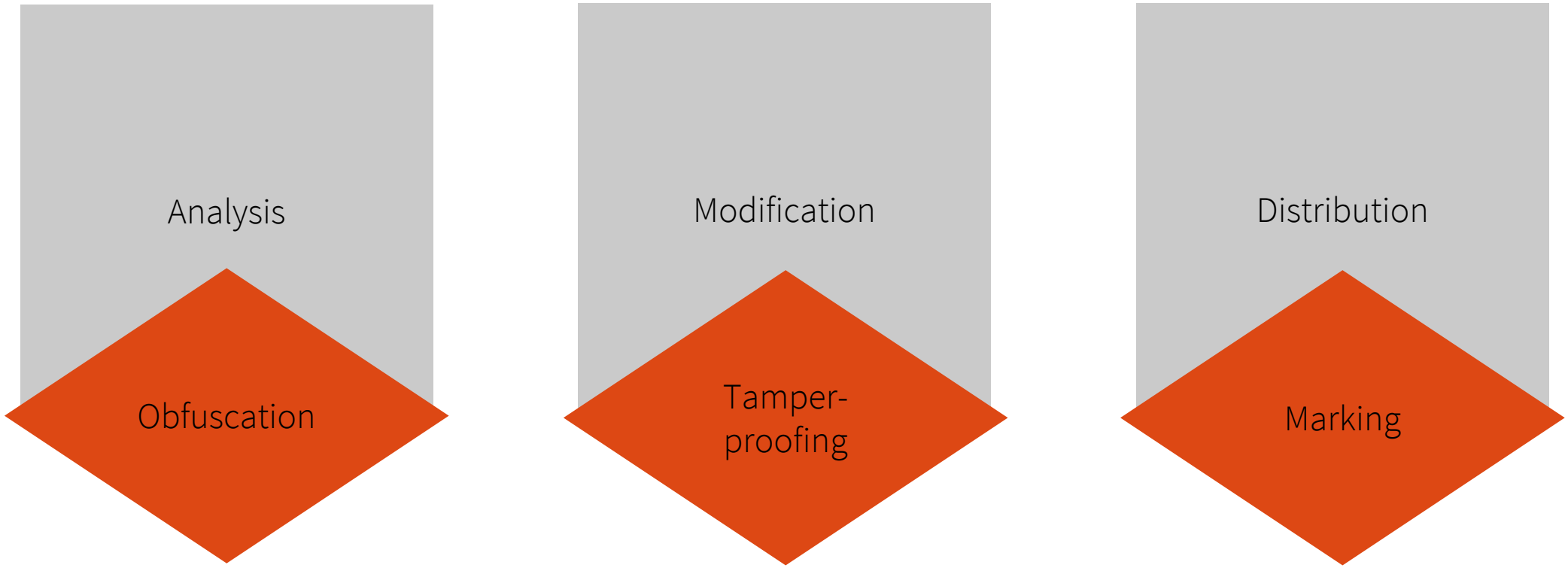
universität wien

# Why software protections?



universität wien

# Attack scenarios



universität wien

# Two major research directions

Obfuscation based on increasing code complexity

Obfuscation based on cryptographic primitives

universität wien

# Definition by Collberg et al.

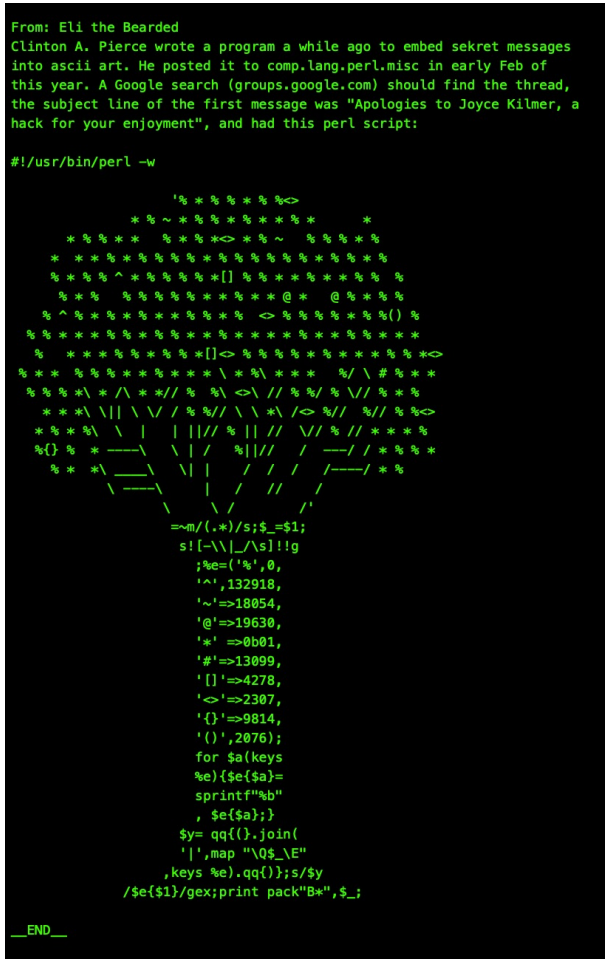
Let $P \xrightarrow{\mathcal{T}} P'$ be a transformation of a source program $P$ into a target program $P'$. The transformation $P \xrightarrow{\mathcal{T}} P'$ is an *obfuscating transformation*, if $P$ and $P'$ have the same *observable behaviour*. More precisely, in order for $P \xrightarrow{\mathcal{T}} P'$ to be a legal obfuscating transformation the following conditions must hold:

- If $P$ fails to terminate or terminates with an error condition, then $P'$ may or may not terminate.
- Otherwise, $P'$ must terminate and produce the same output as $P$.

(Collberg et a. 1997)

universität wien






Source: https://ascii.co.uk/art/perl

universität wien

# Classification of obfuscating algorithms

Data obfuscation

Static obfuscation

Dynamic obfuscation

universität wien

# Data obfuscation

- Make data look different
- Goal: An attacker is unable to locate data based on its known structure
- Example: AES key
  - 128, 192, or 256 bit
  - High entropy
  - Often: pre-calculated round keys after the main key
  - aeskeyfind: program to locate an AES key in a captured memory image

universität wien

# Data obfuscation

Reordering data

Changing encodings

Converting static data to procedures

universität wien

# Reordering data

- Variables can be split into two or more pieces
  - Mapping managed by two functions (splitting at obfuscation time, reconstruction at runtime)
- Example: Splitting booleans
  - Split a boolean variable x into two parts p and q
  - p and q shall be set in different parts of the program
  - Encoding: Choose one of the representations randomly
  - Decoding: Many possibilities
    - Check, whether p == q
    - Calculate p XOR q, with „true“ … 1

| x | p | q |
|---|---|---|
| true | false | false |
| false | false | true |
| false | true | false |
| true | true | true |

universität wien

# Classification of obfuscating algorithms

Data obfuscation

Static obfuscation

Dynamic obfuscation

universität wien

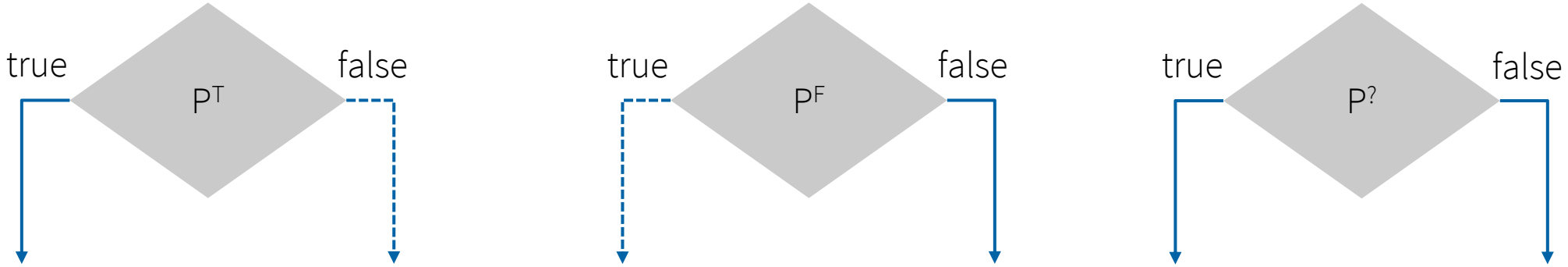
# Static code rewriting

| | | | | | |
|---|---|---|---|---|---|
| Opaque predicates | Inserting dead or irrelevant code | Replacing instructions | Reordering | Loop trans-formations | Function splitting/ recombination |
| Aliasing | Control flow obfuscation | Parallelized code | Name scrambling | Removing standard library calls | Breaking relations |

universität wien

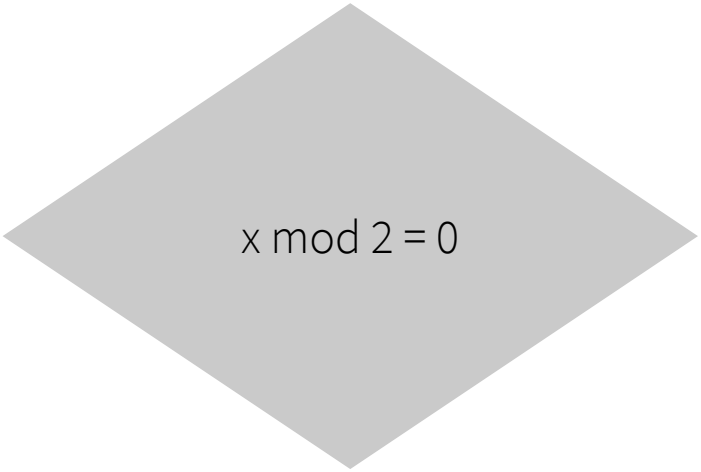
# Opaque predicates

- Opaque expression: Expression whose value is known at obfuscation time, but difficult for an attacker to figure out
- Most common are opaque predicates (boolean valued expressions)



universität wien

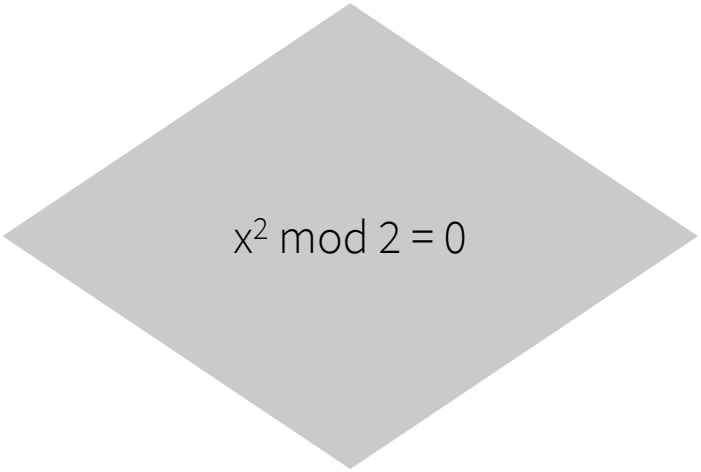
# Opaque predicates




$P^?$

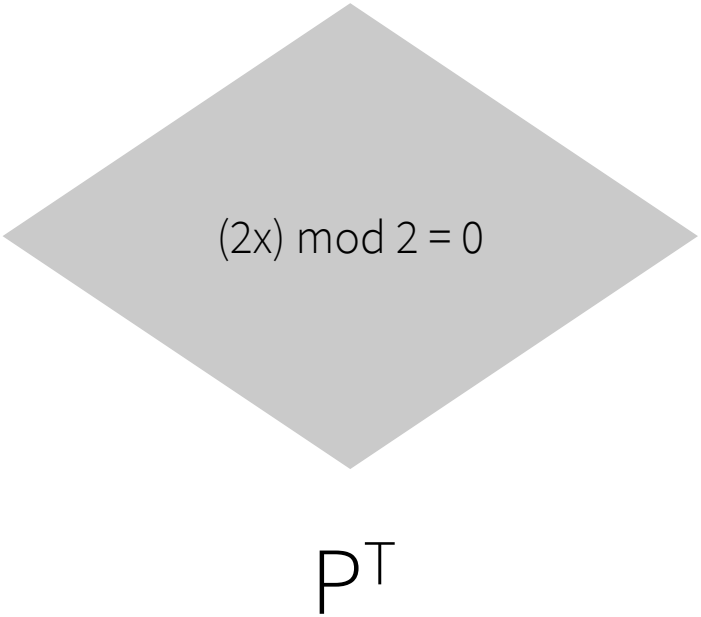
universität wien

# Opaque predicates



universität wien

# Opaque predicates



universität wien

# Name scrambling

```
public long convert(float amountDollar, float rate)
amountEuro = amountDollar * rate;
   return amountEuro;
}
```

universität wien

# Classification of obfuscating algorithms

Data obfuscation

Static obfuscation

Dynamic obfuscation

universität wien

# Dynamic code rewriting

| | | |
|---|---|---|
| Packing/ Encryption | Dynamic code modifications | Environmental requirements |
| Hardware-assisted obfuscation | Virtualization | Anti-debugging techniques |

universität wien

# Virtualization

- One of the most advanced techniques for binary obfuscation
- Converting the program's functionality into byte code for a custom virtual machine interpreter that is bundled with the program
- The virtual machine interpreter and payload can be different for each instance of the program (polymorphism)

universität wien

| | Degree of automation | | | | | Extracted artifacts | Analysis methods | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Understand input | Find context switch | Understand fetcher/vpc | Understand dispatcher | Understand bytecode | | static | dynamic | Distributions/Statistics | Slicing | Symbolic execution | Taint analysis | Trace | Abstract interpretation |
| Rolles [36] | M | M | M | M | M | simplified code | ✓ | - | - | - | - | - | - | - |
| Rotalumé [41] | M | ✓ | ✓ | ✓ | ✓ | CFG, trace | - | ✓ | ✓ | - | - | ✓ | ✓ | - |
| METASM [20] | M | M | M | M | ✓ | bytecode mapping, lifting, recompilation | ✓ | - | - | ✓ | ✓ | - | - | - |
| METASM [21] | M | ✓ | ✓ | ✓ | ✓ | simplified code, decompiled code | ✓ | ✓ | - | ✓ | ✓ | - | ✓ | - |
| Coogan et al. [16] | M | ✓ | ✓ | ✓ | ✓ | simplified trace | - | ✓ | - | ✓ | - | - | ✓ | - |
| Kinder [26] | M | ✓ | M | ✓ | ✓ | CFG, invariants | ✓ | - | - | - | - | - | - | ✓ |
| Virtual Deobfuscator [34] | M | M | ✓ | ✓ | ✓ | simplified code | ✓ | ✓ | ✓ | - | - | - | ✓ | - |
| Yadegari et al. [49] | ✓ | ✓ | ✓ | ✓ | ✓ | simplified CFG | - | ✓ | - | - | ✓ | ✓ | ✓ | - |
| SEEAD [42] | ✓ | ✓ | ✓ | ✓ | ✓ | simplified trace, CFG, FCG | - | ✓ | - | - | - | ✓ | ✓ | - |
| VMAttack [25] | M | ✓ | ✓ | ✓ | ✓ | graded trace | ✓ | ✓ | - | ✓ | - | - | ✓ | - |
| Syntia [11] | M | M | M | M | ✓ | simplified VM instruction handler (semantics) | - | ✓ | ✓ | - | - | - | ✓ | - |
| Liang et al. [30] | M | ✓ | ✓ | ✓ | ✓ | simplified code | - | ✓ | - | - | ✓ | - | ✓ | - |
| VMHunt [47] | M | ✓ | ✓ | ✓ | ✓ | symbolic formula | - | ✓ | ✓ | ✓ | ✓ | - | ✓ | - |
| Tigress DeObf [37] | M | ✓ | ✓ | ✓ | ✓ | simplified code | - | ✓ | - | - | ✓ | ✓ | ✓ | - |
| DynOpVM [13] | M | M | M | M | ✓ | bytecode mapping | ✓ | - | ✓ | - | - | - | - | - |

*Kochberger, P., Schrittwieser, S., Schweighofer, S., Kieseberg, P., & Weippl, E. (2021). SoK: Automatic Deobfuscation of Virtualization-protected Applications. In The 16th International Conference on Availability, Reliability and Security.*

universität wien

# Protection evaluation

- In contrast to cryptography, it is very difficult to make a statement about the strength of an obfuscation
- Protection strength depends on a variety of parameters, including the motivation and creativity of a human analyst
- Collberg et al. proposed a taxonomy for obfuscations in 1997
  - Potency, resilience, cost, stealth

universität wien

# Potency

- A potent obfuscating transformation makes at least one analysis method harder to perform and no analysis easier
- Mila Dalla Preda[1] presented a potency framework based on abstract domains
  - Comparing the properties that are preserved by obfuscation transformations
  - A transformation that preserves more properties is weaker than one than preserves less
  - Often, domains of obfuscating transformations are not comparable
  - Example: (very simple) data obfuscation

[1] Mila Dalla Preda. *Code Obfuscation and Malware Detection by Abstract Interpretation*. Ph.D. thesis, Dipartimento di Informatica, Universita' di Verona, 2007.

universität wien

# Potency

$\tau_1(x) = 2 \cdot x$
$\tau_2(x) = 3 \cdot x$

$$\text{Sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{If } x > 0 \end{cases}$$

$$\text{Parity}(x) = \begin{cases} 0 & \text{if even}(x) \\ 1 & \text{if odd}(x) \end{cases}$$

Both obfuscations preserve the *Sign* property

Only $\tau_2$ preserves the *Parity* property

Two simple data obfuscations

Properties *Sign* and *Parity*

$\tau_1$ preserves less, thus is more potent

Source: Surreptitious Software, C. Collberg and J. Nagra

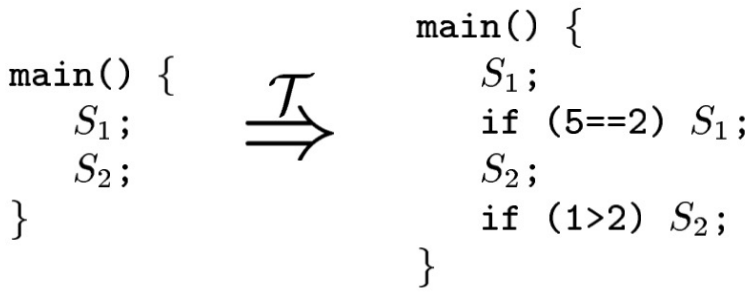
universität wien

# Potency

- How much more obscure (complex, unreadable) is an obfuscated representation of a program (for humans)
- Often evaluated with *software complexity metrices*
  - E.g., counting textual properties of the source code, cyclomatic complexity
  - Usually, the goal in software engineering is to make code *less* complex
  - A potent obfuscating transformation makes code *more* complex
- QMOOD (Quality Model for Object-Oriented Design) for Java code
  - Metric for understandability including abstraction, encapsulation, etc.
  - Relative metric that can only be used to compare two program versions
- Visual Studio Code Metrics

universität wien

# Potency




- This obfuscating transformation is potent (it makes the code more complex)
- However, is it almost useless, because it can be undone easily

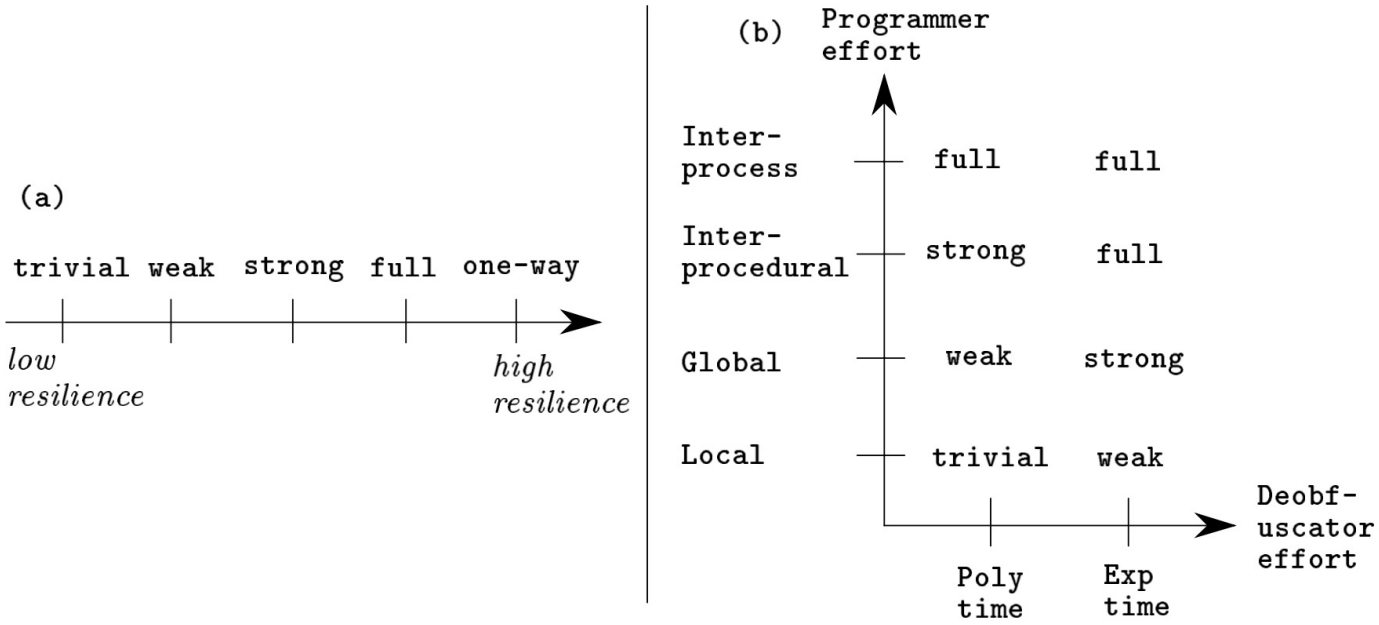
Source: Collberg et al. (1997). A taxonomy of obfuscating transformations. University of Auckland

universität wien

# Resilience

- Strength of a transformation against an automatic deobfuscator program
- Two properties
  - Programmer effort
    - The amount of time required to construct an automatic deobfuscator for a particular obfuscating transformation to effectively reduce its potency
  - Deobfuscator effort
    - Execution time and space required to run the deobfuscator

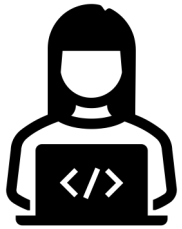
# Resilience

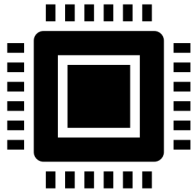



Source: Collberg et al. (1997). A taxonomy of obfuscating transformations. University of Auckland

universität wien

# Potency and resilience in a nutshell

Potency → confuses →  humans

Resilience → confuses →  deobfuscator programs

universität wien

# Cost

- Computational overhead (runtime, memory consumption, etc.) of an obfuscating transformation
- Measurement easy compared to potency and resilience
- However, meaningless without potency/resilience measurements
- What are acceptable costs?
  - Highly depending on the concrete use case

universität wien

# Cost

$$\mathcal{T}_{\text{cost}}(P) \stackrel{\text{def}}{=} \begin{cases} \textit{dear} & \text{if executing } P' \text{ requires } \textit{exponentially} \text{ more resources than } P. \\ \textit{costly} & \text{if executing } P' \text{ requires } \mathcal{O}(n^p),\ p > 1, \text{ more resources than } P. \\ \textit{cheap} & \text{if executing } P' \text{ requires } \mathcal{O}(n) \text{ more resources than } P. \\ \textit{free} & \text{if executing } P' \text{ requires } \mathcal{O}(1) \text{ more resources than } P. \end{cases}$$

Source: Collberg et al. (1997). A taxonomy of obfuscating transformations. University of Auckland

universität wien

# Potency, resilience, and cost

| Obfuscation | | | Quality | | | | |
|---|---|---|---|---|---|---|---|
| Target | Operation | Transformation | Potency | Resilience | Cost | Metrics | Section |
| Layout | | Scramble Identifiers | medium | one-way | free | | 5.5 |
| | | Change Formatting | low | one-way | free | | 5.5 |
| | | Remove Comments | high | one-way | free | | 5.5 |
| Control | Computations | Insert Dead or Irrelevant Code | Depends on the quality of the opaque predicate and the nesting depth at which the construct is inserted. | | | $\mu_1,\mu_2,\mu_3$ | 6.2.1 |
| | | Extend Loop Condition | | | | $\mu_1,\mu_2,\mu_3$ | 6.2.2 |
| | | Reducible to Non-Reducible | | | | $\mu_1,\mu_2,\mu_3$ | 6.2.3 |
| | | Add Redundant Operands | | | | $\mu_1$ | 6.2.6 |
| | | Remove Programming Idioms | medium | strong | † | $\mu_1$ | 6.2.4 |
| | | Table Interpretation | high | strong | costly | $\mu_1$ | 6.2.5 |
| | | Parallelize Code | high | strong | costly | $\mu_1,\mu_2$ | 6.2.7 |

Source: Collberg et al. (1997). A taxonomy of obfuscating transformations. University of Auckland

universität wien

# Two major research directions

Obfuscation based on increasing code complexity

Obfuscation based on cryptographic primitives

universität wien

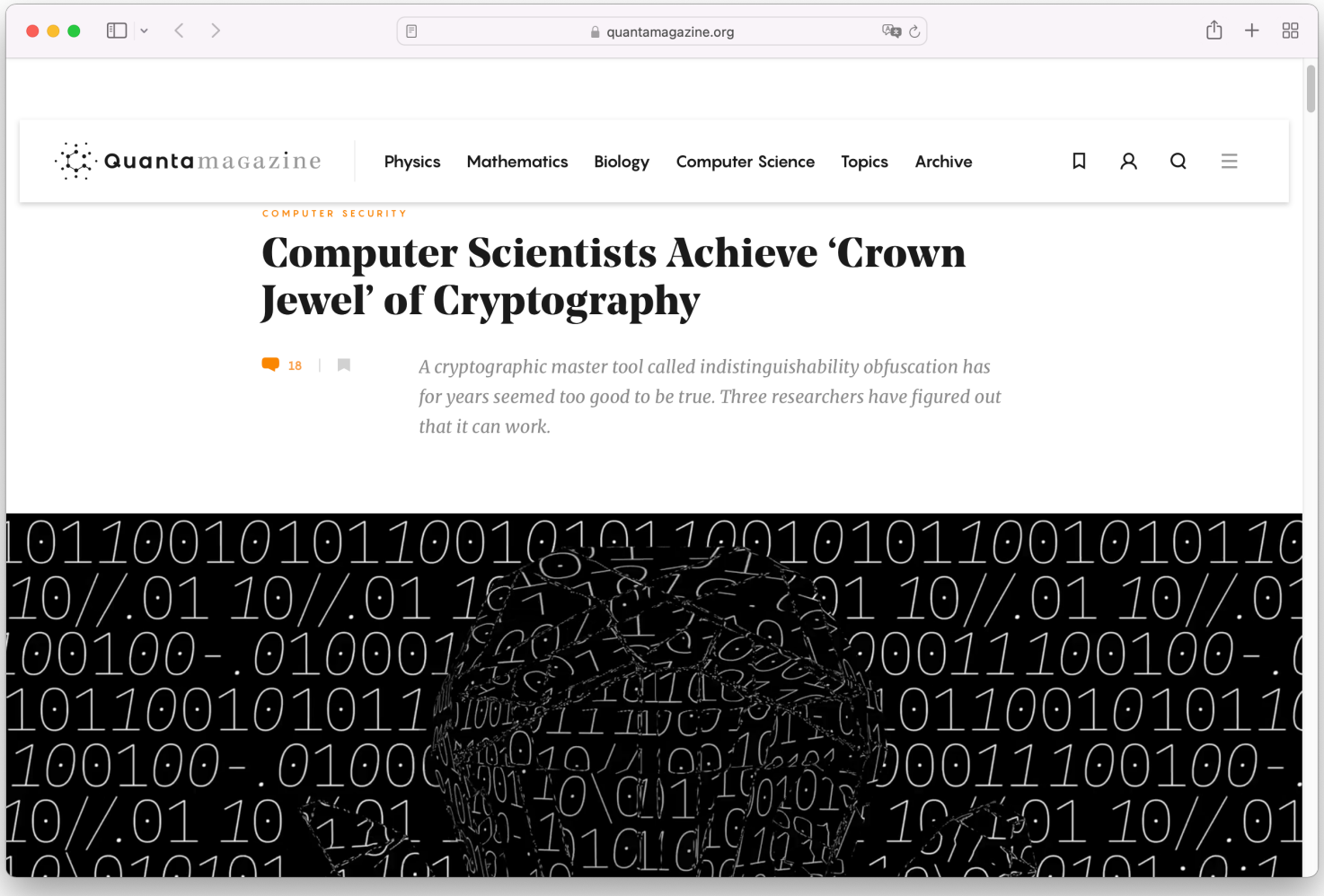
# Definition

- Formal definition by Barak et al [2001]
- An obfuscator *O* is a “compiler” which takes as input a program *P* and produces a new program *O(P)* such that for every *P*:
  - **Functionality:** *O(P)* computes the same function as *P*
  - **Polynomial Slowdown:** The description length and running time of *O(P)* are at most polynomially larger than that of *P*.
  - **“Virtual black box” property:** “Anything that can be efficiently computed from *O(P)* can be efficiently computed given oracle access to *P*“

universität wien

# Virtual black box

- Gaind tremendous attention since 2014
  - Papers until 2013: 12
  - Papers 2014-2021: 117



universität wien

# Indistinguishability obfuscation

- „One cryptographic primitive to rule them all” (Barak, Harvard University)
  - First proposed by Barak et al [2001]
  - First candidate indistinguishability obfuscation from assumptions over multilinear maps in 2013
  - Since then, many more concepts were published
- $C_1$ und $C_2$ are two different circuits that both compute the same functionality
  - Obf($C_1$) and Obf($C_2$) are indistinguishable
  - Meaning: If there is more than one way of implementing a particular functionality, the obfuscated version doesn't reveal anything about the chosen implementation of the functionality

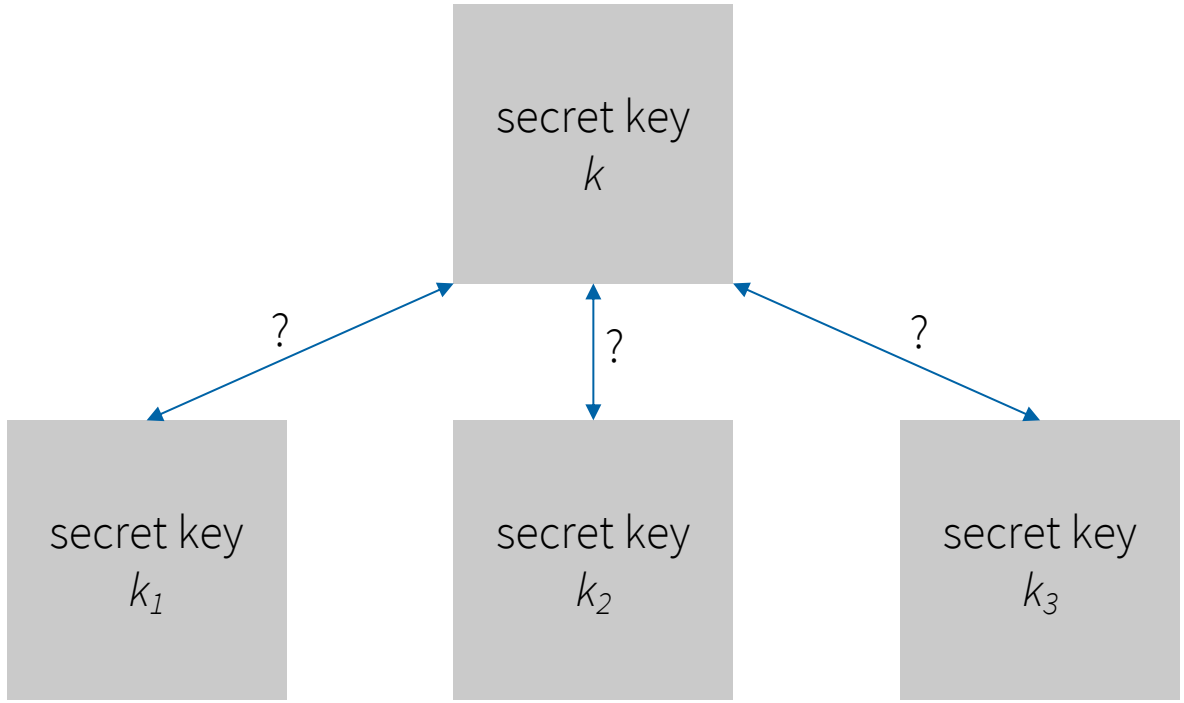
universität wien

# Equivalence problem

- Basic idea of Turing's halting problem
  - Boolean method *HALTS*
  - *HALTS* performs arbitrarily complex analysis of a program to find out if the program halts or runs forever
  - *HALTS* itself must be a method that halts
- Turing showed that computing whether a program halts is impossible for some programs with a counterexample
  - From this negative result, the **equivalence problem** can be derived
    - We have two programs: one always halts, the other one is Turing's counterexample – are these programs equivalent?

universität wien

# Provably secure obfuscation?



universität wien

# Virtual black-box property in real-life use cases

```
boolean isValidPassword(String password){
     if (password.equals('mySecretPassword'))
          return true;
     else
          return false;
}
```

universität wien

# Virtual black-box property in real-life use cases

```
boolean isValidPassword(String password){
     if (sha512(password).equals('d32b568cd1[…]08eab'))
          return true;
     else
          return false;
}
```

universität wien

# Point functions

- Return 1 for one specific input
- Return 0 for all other inputs
- Can be used for obfuscation
  - Hierarchical access control
  - Regular expressions
  - Database relations
- Always based on challenge/response

universität wien

# Future of indistinguishability obfuscation

- Seemed to be impossible a few years ago
- Still, research is not even close to something useful in practice
- Often compared to progress in homomorphic encryption
  - Allows performing computations on encrypted data without decrypting it
  - Also seemed impossible, but real-world use-cases do exist now
- Interesting read from 2020
  - https://www.quantamagazine.org/computer-scientists-achieve-crown-jewel-of-cryptography-20201110/

universität wien

# Conclusions

- Obfuscations based on increasing the complexity of the code are heavily used for more than 3 decades both to protect benign software an well as malware
- Measuring the strength of on obfuscation is challenging as it depends on multiple factors including an attacker‘s skills
- Indistinguishability obfuscation is based on cryptographic primitives and might revolutionize software protections in the future
  - Gained a lot of attraction after a break-through paper in 2013 presenting a first candidate indistinguishability obfuscation
  - Today, however, no practical concept exists

universität wien

# Thank you for your attention!