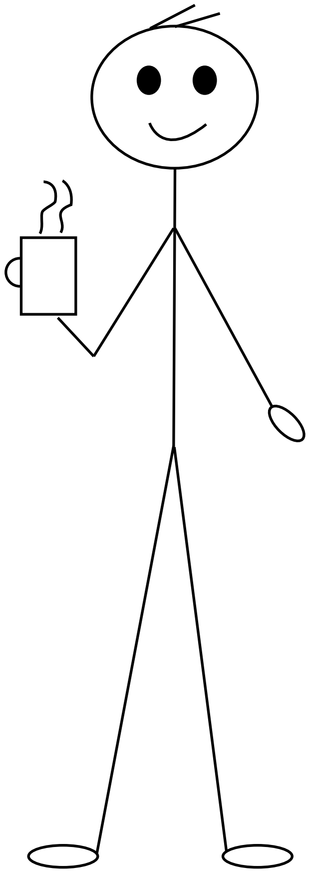
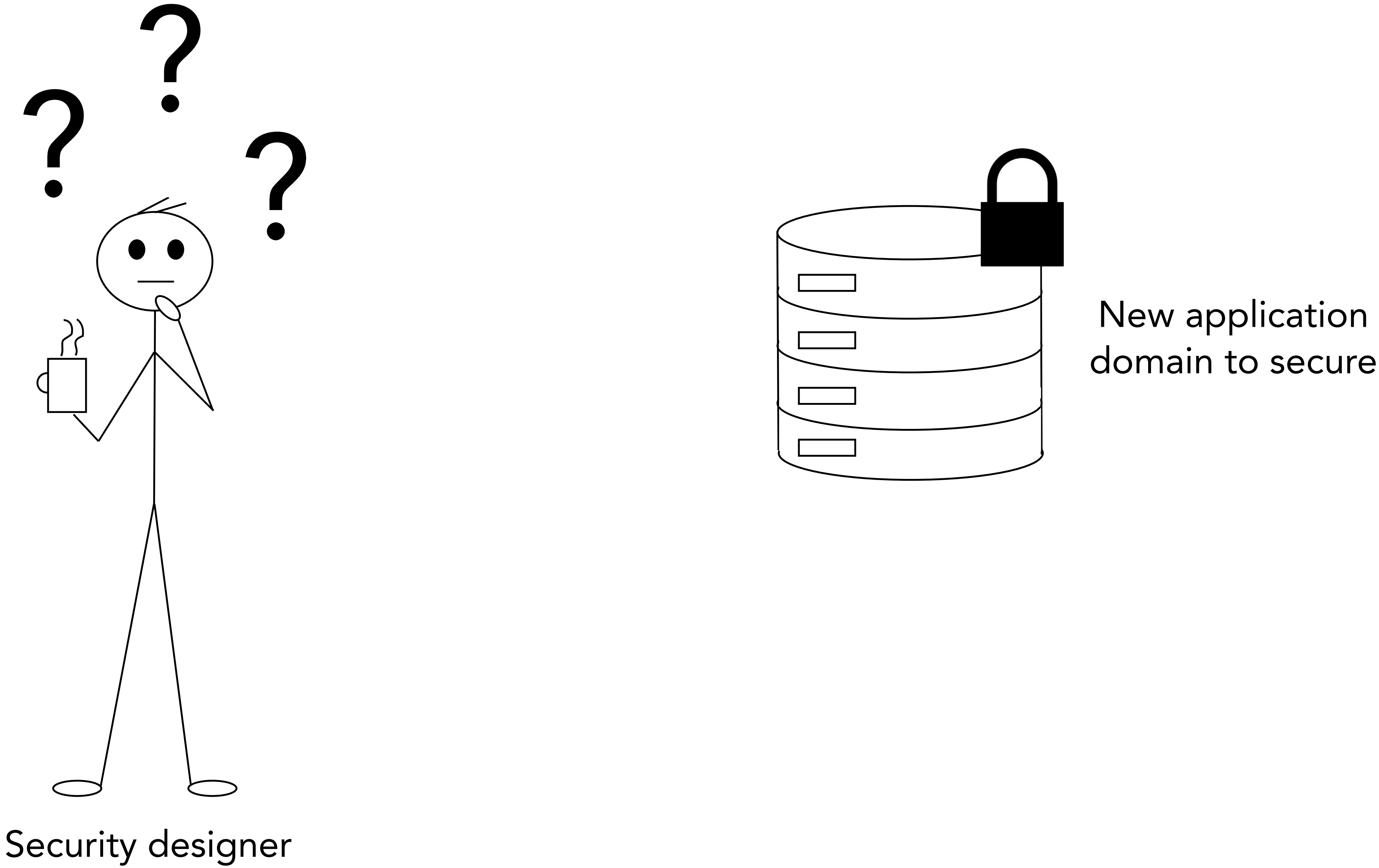
Prudent Design Principles for Information Flow Control

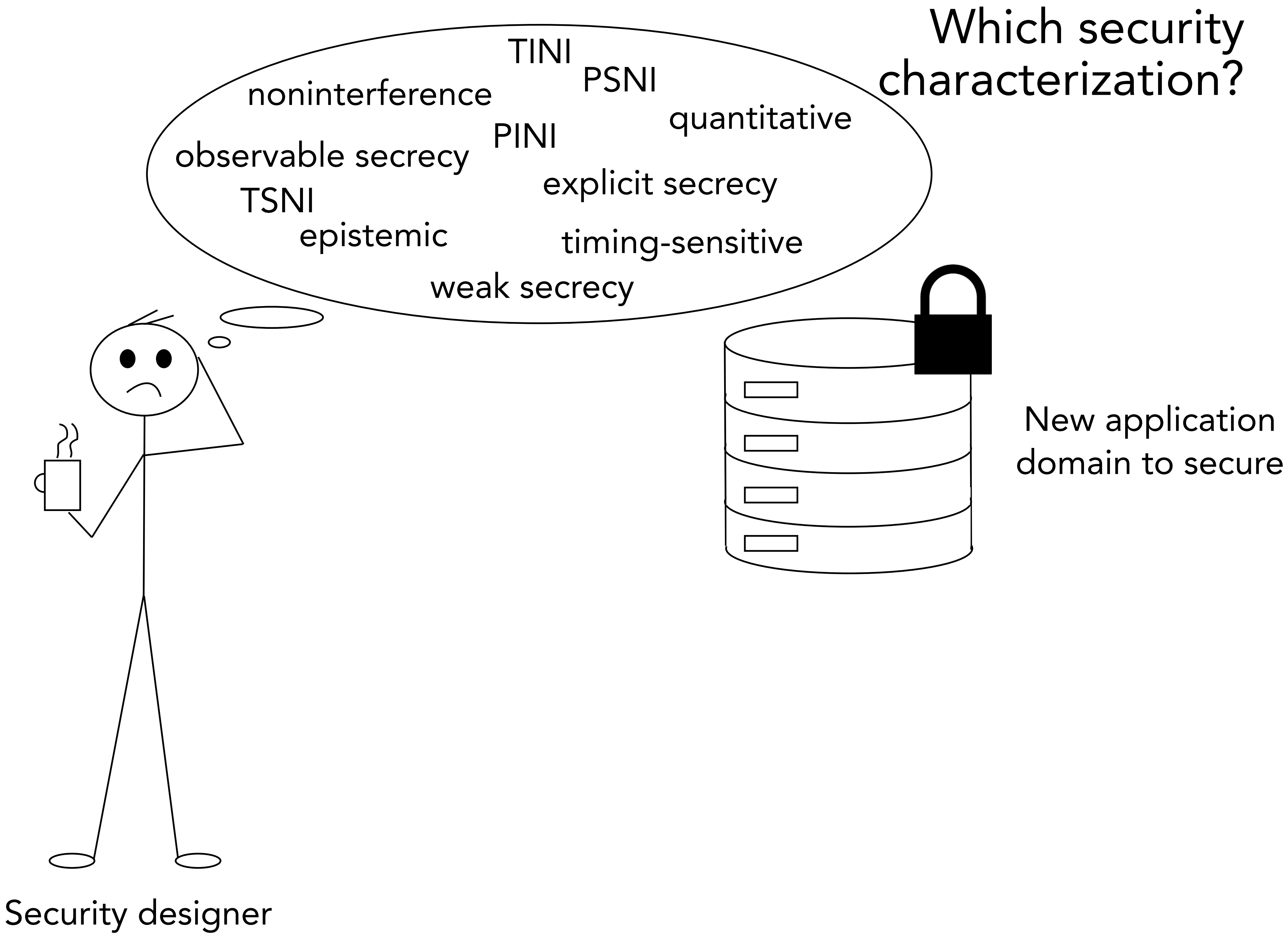
Iulia Bastys Frank Piessens Andrei Sabelfeld

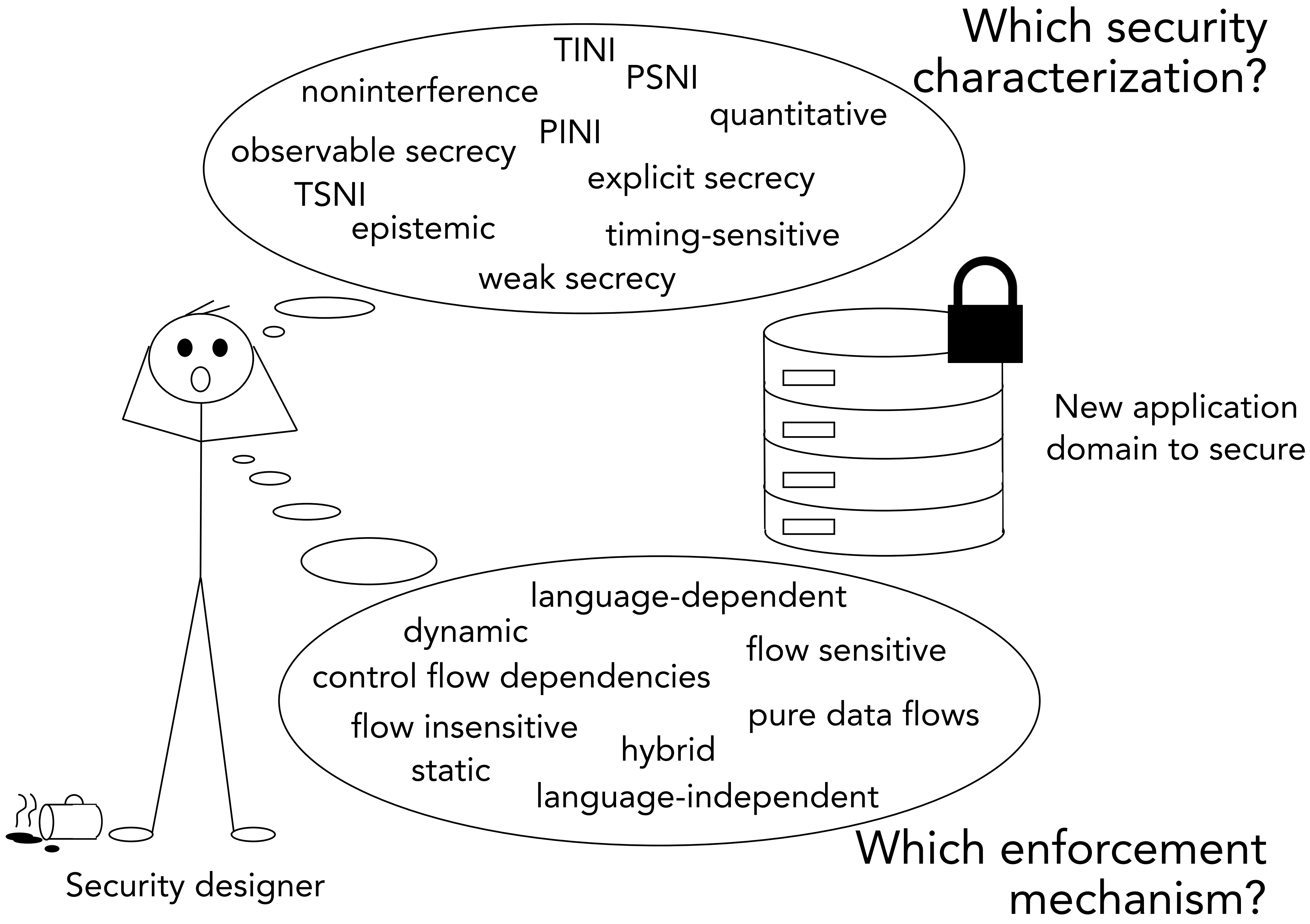
CHALMERS KU LEUVEN

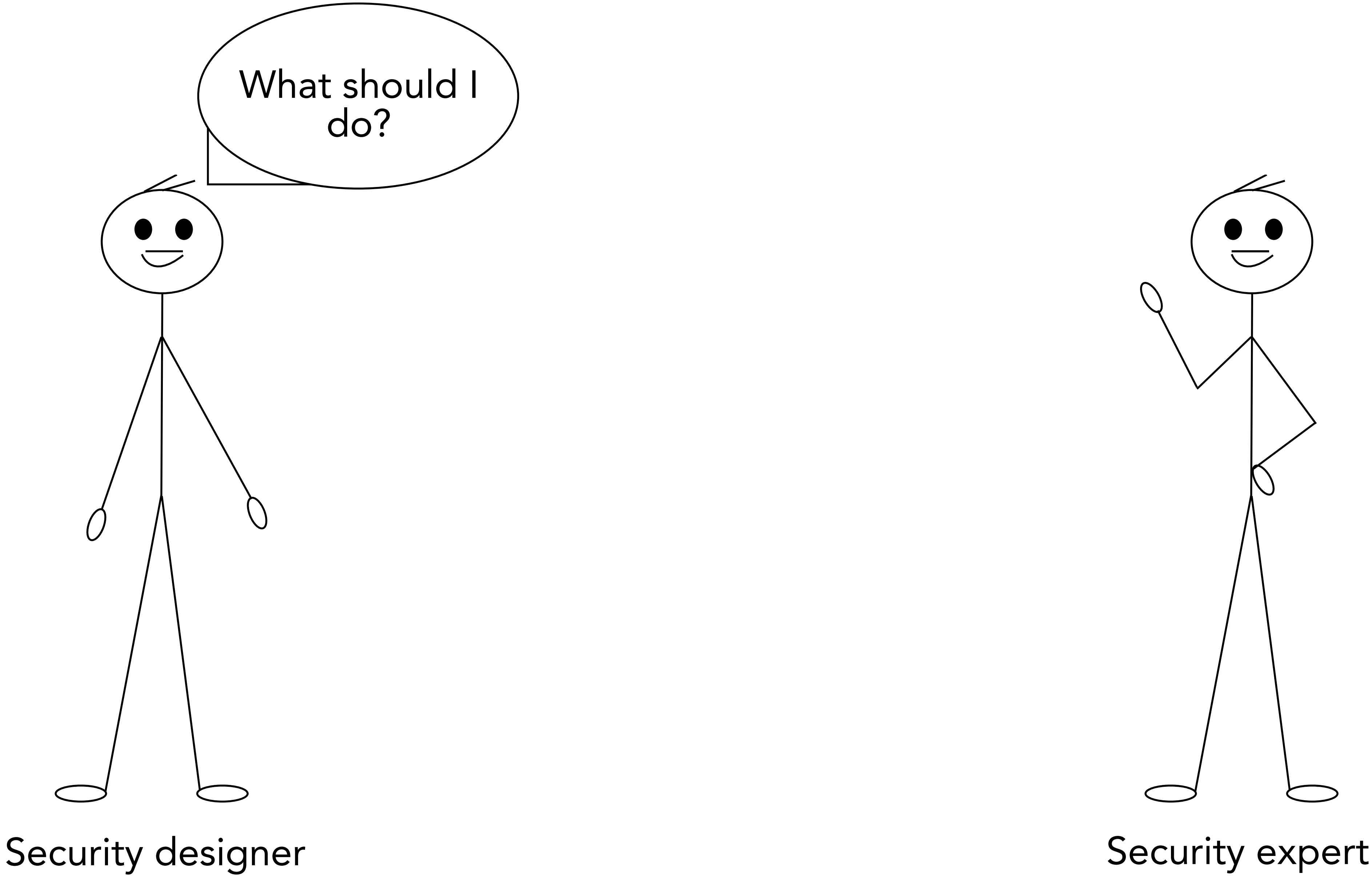


Security designer

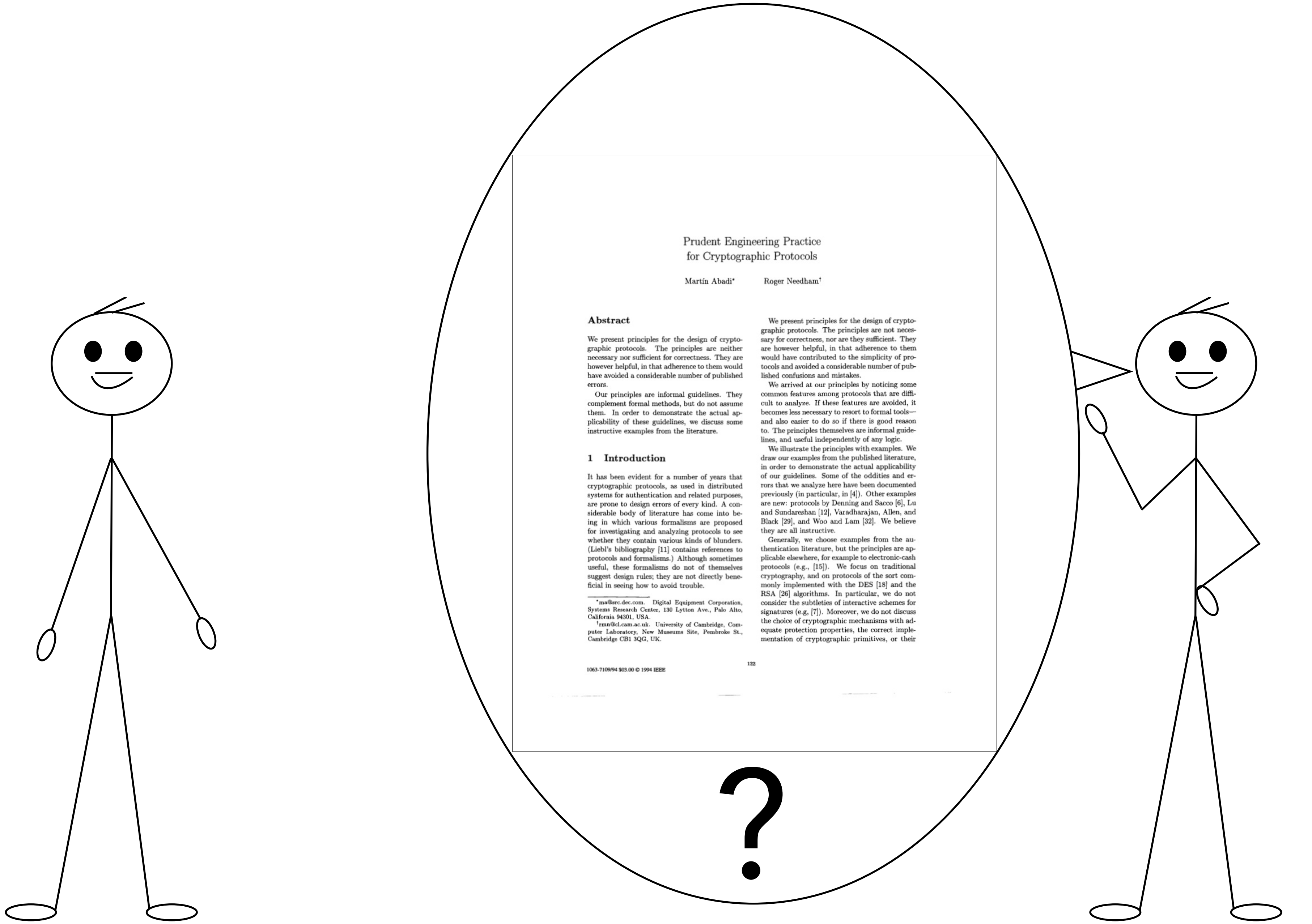

Security designer
New application domain to secure


Which security characterization?
TINI
PSNI
noninterference
quantitative
PINI
observable secrecy
explicit secrecy
TSNI
epistemic
timing-sensitive
weak secrecy
New application domain to secure
Security designer


Which security characterization?
TINI
PSNI
noninterference
quantitative
PINI
observable secrecy
explicit secrecy
TSNI
epistemic
timing-sensitive
weak secrecy
New application domain to secure
language-dependent
dynamic
flow sensitive
control flow dependencies
pure data flows
flow insensitive
hybrid
static
language-independent
Which enforcement mechanism?
Security designer



What should I do?
Security designer
Security expert



Prudent Engineering Practice
for Cryptographic Protocols

Martín Abadi* Roger Needham†

Abstract

We present principles for the design of cryptographic protocols. The principles are neither necessary nor sufficient for correctness. They are however helpful, in that adherence to them would have avoided a considerable number of published errors.

Our principles are informal guidelines. They complement formal methods, but do not assume them. In order to demonstrate the actual applicability of these guidelines, we discuss some instructive examples from the literature.

1 Introduction

It has been evident for a number of years that cryptographic protocols, as used in distributed systems for authentication and related purposes, are prone to design errors of every kind. A considerable body of literature has come into being in which various formalisms are proposed for investigating and analyzing protocols to see whether they contain various kinds of blunders. (Liebl's bibliography [11] contains references to protocols and formalisms.) Although sometimes useful, these formalisms do not of themselves suggest design rules; they are not directly beneficial in seeing how to avoid trouble.

We present principles for the design of cryptographic protocols. The principles are not necessary for correctness, nor are they sufficient. They are however helpful, in that adherence to them would have contributed to the simplicity of protocols and avoided a considerable number of published confusions and mistakes.

We arrived at our principles by noticing some common features among protocols that are difficult to analyze. If these features are avoided, it becomes less necessary to resort to formal tools—and also easier to do so if there is good reason to. The principles themselves are informal guidelines, and useful independently of any logic.

We illustrate the principles with examples. We draw our examples from the published literature, in order to demonstrate the actual applicability of our guidelines. Some of the oddities and errors that we analyze here have been documented previously (in particular, in [4]). Other examples are new: protocols by Denning and Sacco [6], Lu and Sundareshan [12], Varadharajan, Allen, and Black [29], and Woo and Lam [32]. We believe they are all instructive.

Generally, we choose examples from the authentication literature, but the principles are applicable elsewhere, for example to electronic-cash protocols (e.g., [15]). We focus on traditional cryptography, and on protocols of the sort commonly implemented with the DES [18] and the RSA [26] algorithms. In particular, we do not consider the subtleties of interactive schemes for signatures (e.g, [7]). Moreover, we do not discuss the choice of cryptographic mechanisms with adequate protection properties, the correct implementation of cryptographic primitives, or their

*ma@src.dec.com. Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave., Palo Alto, California 94301, USA.

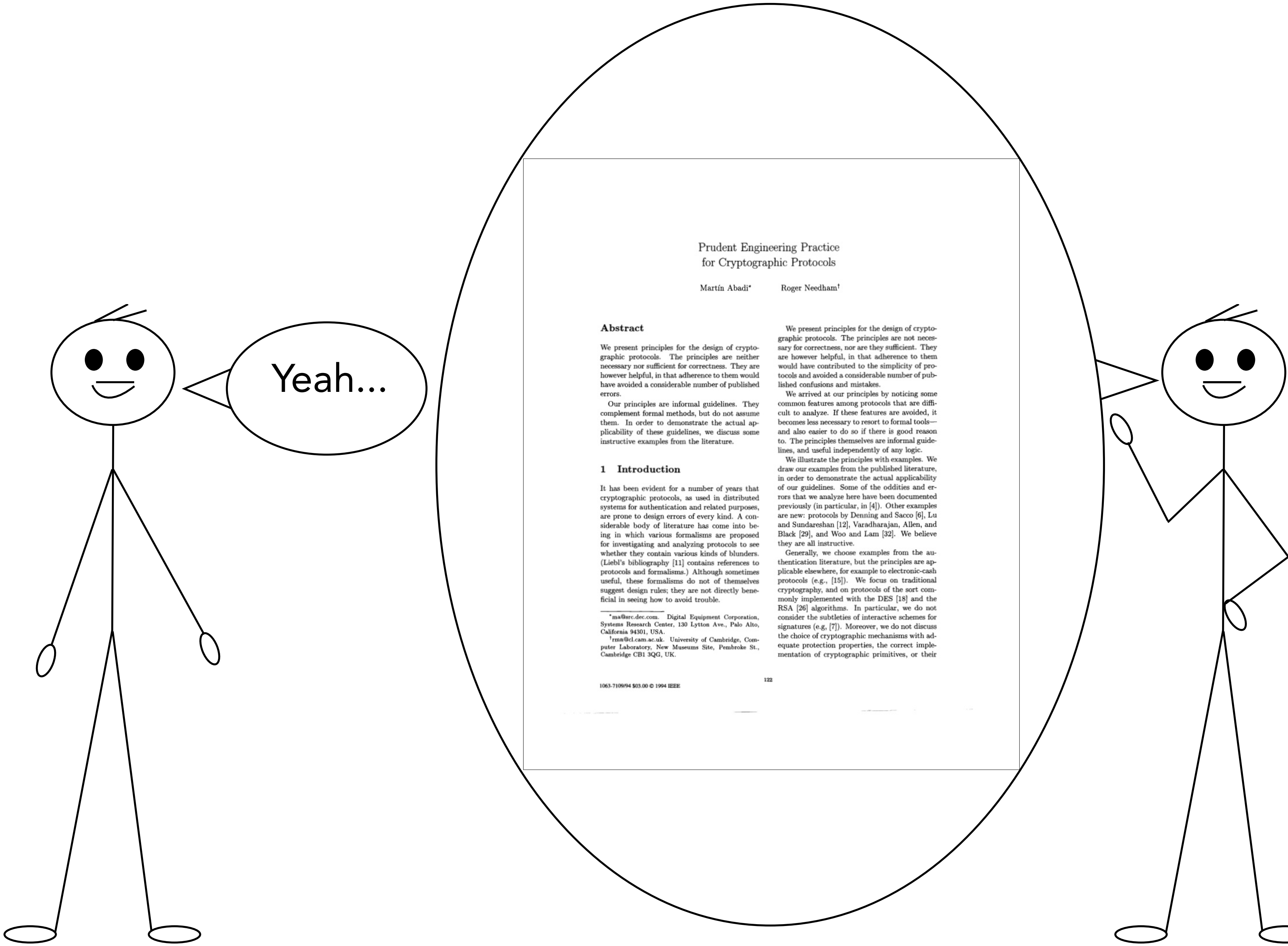
†rmn@cl.cam.ac.uk. University of Cambridge, Computer Laboratory, New Museums Site, Pembroke St., Cambridge CB1 3QG, UK.

122

1063-7109/94 $03.00 © 1994 IEEE

Security designer

Security expert



Prudent Engineering Practice
for Cryptographic Protocols

Martín Abadi*　　Roger Needham†

Abstract

We present principles for the design of cryptographic protocols. The principles are neither necessary nor sufficient for correctness. They are however helpful, in that adherence to them would have avoided a considerable number of published errors.

Our principles are informal guidelines. They complement formal methods, but do not assume them. In order to demonstrate the actual applicability of these guidelines, we discuss some instructive examples from the literature.

1 Introduction

It has been evident for a number of years that cryptographic protocols, as used in distributed systems for authentication and related purposes, are prone to design errors of every kind. A considerable body of literature has come into being in which various formalisms are proposed for investigating and analyzing protocols to see whether they contain various kinds of blunders. (Liebl's bibliography [11] contains references to protocols and formalisms.) Although sometimes useful, these formalisms do not of themselves suggest design rules; they are not directly beneficial in seeing how to avoid trouble.

We present principles for the design of cryptographic protocols. The principles are not necessary for correctness, nor are they sufficient. They are however helpful, in that adherence to them would have contributed to the simplicity of protocols and avoided a considerable number of published confusions and mistakes.

We arrived at our principles by noticing some common features among protocols that are difficult to analyze. If these features are avoided, it becomes less necessary to resort to formal tools—and also easier to do so if there is good reason to. The principles themselves are informal guidelines, and useful independently of any logic.

We illustrate the principles with examples. We draw our examples from the published literature, in order to demonstrate the actual applicability of our guidelines. Some of the oddities and errors that we analyze here have been documented previously (in particular, in [4]). Other examples are new: protocols by Denning and Sacco [6], Lu and Sundareshan [12], Varadharajan, Allen, and Black [29], and Woo and Lam [32]. We believe they are all instructive.

Generally, we choose examples from the authentication literature, but the principles are applicable elsewhere, for example to electronic-cash protocols (e.g., [15]). We focus on traditional cryptography, and on protocols of the sort commonly implemented with the DES [18] and the RSA [26] algorithms. In particular, we do not consider the subtleties of interactive schemes for signatures (e.g, [7]). Moreover, we do not discuss the choice of cryptographic mechanisms with adequate protection properties, the correct implementation of cryptographic primitives, or their

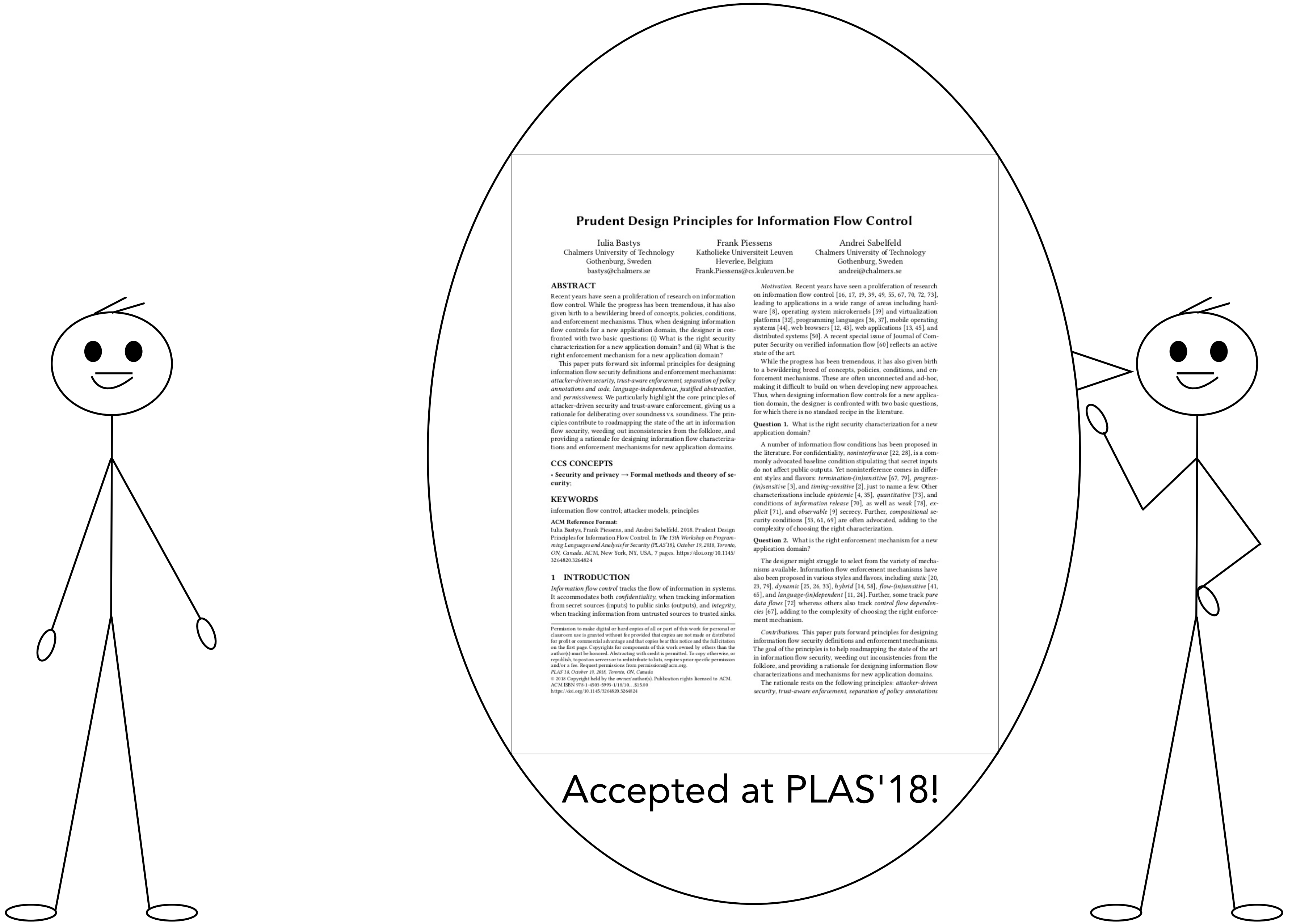
*ma@src.dec.com. Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave., Palo Alto, California 94301, USA.

†rmn@cl.cam.ac.uk. University of Cambridge, Computer Laboratory, New Museums Site, Pembroke St., Cambridge CB1 3QG, UK.

1063-7109/94 $03.00 © 1994 IEEE

Security designer

Security expert



Prudent Design Principles for Information Flow Control

Iulia Bastys
Chalmers University of Technology
Gothenburg, Sweden
bastys@chalmers.se

Frank Piessens
Katholieke Universiteit Leuven
Heverlee, Belgium
Frank.Piessens@cs.kuleuven.be

Andrei Sabelfeld
Chalmers University of Technology
Gothenburg, Sweden
andrei@chalmers.se

ABSTRACT

Recent years have seen a proliferation of research on information flow control. While the progress has been tremendous, it has also given birth to a bewildering breed of concepts, policies, conditions, and enforcement mechanisms. Thus, when designing information flow controls for a new application domain, the designer is confronted with two basic questions: (i) What is the right security characterization for a new application domain? and (ii) What is the right enforcement mechanism for a new application domain?

This paper puts forward six informal principles for designing information flow security definitions and enforcement mechanisms: *attacker-driven security, trust-aware enforcement, separation of policy annotations and code, language-independence, justified abstraction*, and *permissiveness*. We particularly highlight the core principles of attacker-driven security and trust-aware enforcement, giving us a rationale for deliberating over soundness vs. soundiness. The principles contribute to roadmapping the state of the art in information flow security, weeding out inconsistencies from the folklore, and providing a rationale for designing information flow characterizations and enforcement mechanisms for new application domains.

CCS CONCEPTS

• **Security and privacy → Formal methods and theory of security**;

KEYWORDS

information flow control; attacker models; principles

ACM Reference Format:
Iulia Bastys, Frank Piessens, and Andrei Sabelfeld. 2018. Prudent Design Principles for Information Flow Control. In *The 13th Workshop on Programming Languages and Analysis for Security (PLAS'18), October 19, 2018, Toronto, ON, Canada*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3264820.3264824

1 INTRODUCTION

Information flow control tracks the flow of information in systems. It accommodates both *confidentiality*, when tracking information from secret sources (inputs) to public sinks (outputs), and *integrity*, when tracking information from untrusted sources to trusted sinks.

Motivation. Recent years have seen a proliferation of research on information flow control [16, 17, 19, 39, 49, 55, 67, 70, 72, 73], leading to applications in a wide range of areas including hardware [8], operating system microkernels [59] and virtualization platforms [32], programming languages [36, 37], mobile operating systems [44], web browsers [12, 43], web applications [13, 45], and distributed systems [50]. A recent special issue of Journal of Computer Security on verified information flow [60] reflects an active state of the art.

While the progress has been tremendous, it has also given birth to a bewildering breed of concepts, policies, conditions, and enforcement mechanisms. These are often unconnected and ad-hoc, making it difficult to build on when developing new approaches. Thus, when designing information flow controls for a new application domain, the designer is confronted with two basic questions, for which there is no standard recipe in the literature.

Question 1. What is the right security characterization for a new application domain?

A number of information flow conditions has been proposed in the literature. For confidentiality, *noninterference* [22, 28], is a commonly advocated baseline condition stipulating that secret inputs do not affect public outputs. Yet noninterference comes in different styles and flavors: *termination-(in)sensitive* [67, 79], *progress-(in)sensitive* [3], and *timing-sensitive* [2], just to name a few. Other characterizations include *epistemic* [4, 35], *quantitative* [73], and conditions of *information release* [70], as well as *weak* [78], *explicit* [71], and *observable* [9] secrecy. Further, *compositional* security conditions [53, 61, 69] are often advocated, adding to the complexity of choosing the right characterization.

Question 2. What is the right enforcement mechanism for a new application domain?

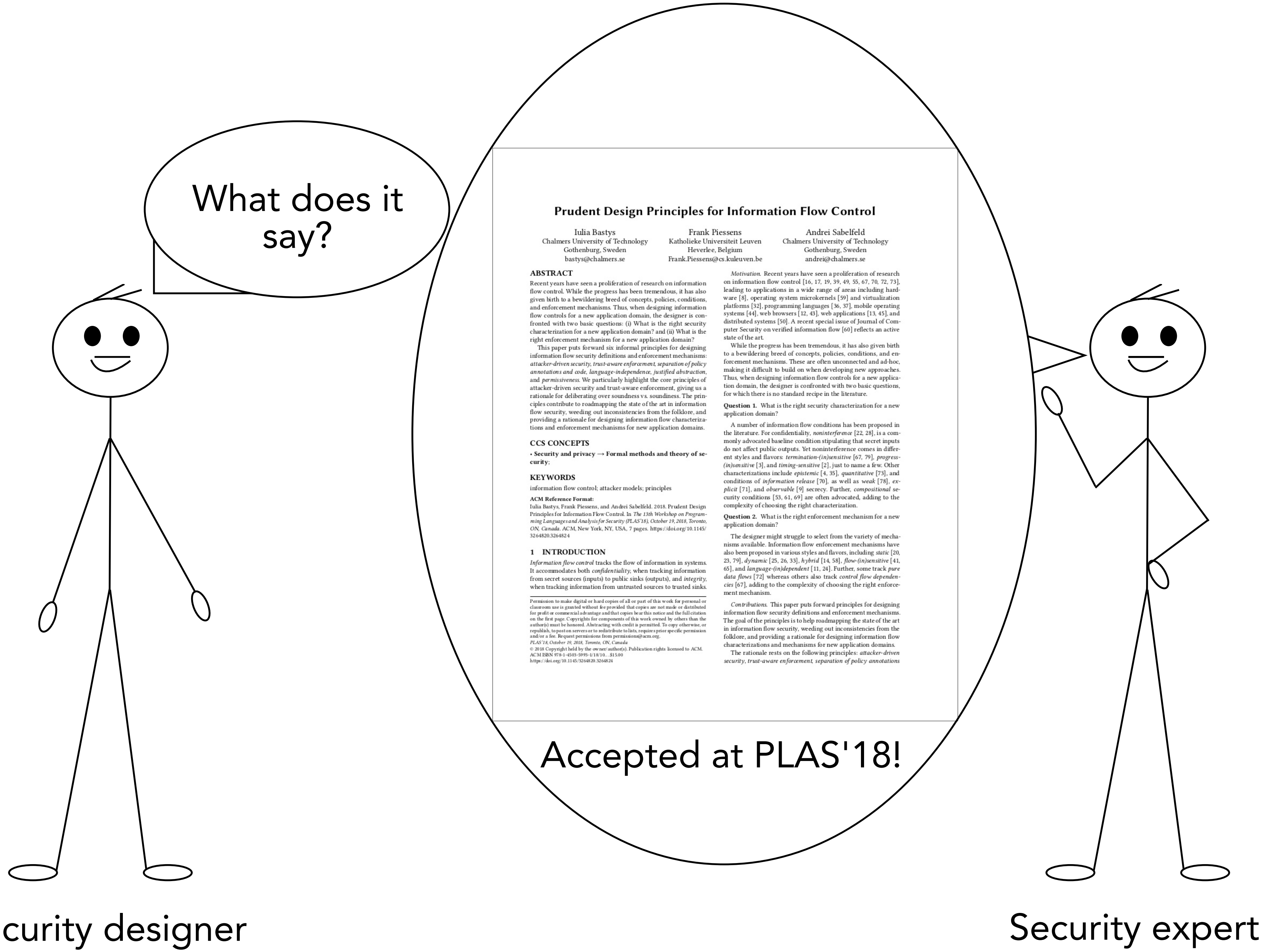
The designer might struggle to select from the variety of mechanisms available. Information flow enforcement mechanisms have also been proposed in various styles and flavors, including *static* [20, 23, 79], *dynamic* [25, 26, 33], *hybrid* [14, 58], *flow-(in)sensitive* [41, 65], and *language-(in)dependent* [11, 24]. Further, some track *pure data flows* [72] whereas others also track *control flow dependencies* [67], adding to the complexity of choosing the right enforcement mechanism.

Contributions. This paper puts forward principles for designing information flow security definitions and enforcement mechanisms. The goal of the principles is to help roadmapping the state of the art in information flow security, weeding out inconsistencies from the folklore, and providing a rationale for designing information flow characterizations and mechanisms for new application domains.

The rationale rests on the following principles: *attacker-driven security, trust-aware enforcement, separation of policy annotations*

PLAS'18, October 19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5993-1/18/10...$15.00
https://doi.org/10.1145/3264820.3264824



Prudent Design Principles for Information Flow Control

Iulia Bastys
Chalmers University of Technology
Gothenburg, Sweden
bastys@chalmers.se

Frank Piessens
Katholieke Universiteit Leuven
Heverlee, Belgium
Frank.Piessens@cs.kuleuven.be

Andrei Sabelfeld
Chalmers University of Technology
Gothenburg, Sweden
andrei@chalmers.se

ABSTRACT

Recent years have seen a proliferation of research on information flow control. While the progress has been tremendous, it has also given birth to a bewildering breed of concepts, policies, conditions, and enforcement mechanisms. Thus, when designing information flow controls for a new application domain, the designer is confronted with two basic questions: (i) What is the right security characterization for a new application domain? and (ii) What is the right enforcement mechanism for a new application domain?

This paper puts forward six informal principles for designing information flow security definitions and enforcement mechanisms: *attacker-driven security, trust-aware enforcement, separation of policy annotations and code, language-independence, justified abstraction*, and *permissiveness*. We particularly highlight the core principles of attacker-driven security and trust-aware enforcement, giving us a rationale for deliberating over soundness vs. soundiness. The principles contribute to roadmapping the state of the art in information flow security, weeding out inconsistencies from the folklore, and providing a rationale for designing information flow characterizations and enforcement mechanisms for new application domains.

CCS CONCEPTS

• **Security and privacy → Formal methods and theory of security**;

KEYWORDS

information flow control; attacker models; principles

ACM Reference Format:
Iulia Bastys, Frank Piessens, and Andrei Sabelfeld. 2018. Prudent Design Principles for Information Flow Control. In *The 13th Workshop on Programming Languages and Analysis for Security (PLAS'18), October 19, 2018, Toronto, ON, Canada*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3264820.3264824

1 INTRODUCTION

Information flow control tracks the flow of information in systems. It accommodates both *confidentiality*, when tracking information from secret sources (inputs) to public sinks (outputs), and *integrity*, when tracking information from untrusted sources to trusted sinks.

Motivation. Recent years have seen a proliferation of research on information flow control [16, 17, 19, 39, 49, 55, 67, 70, 72, 73], leading to applications in a wide range of areas including hardware [8], operating system microkernels [59] and virtualization platforms [32], programming languages [36, 37], mobile operating systems [44], web browsers [12, 43], web applications [13, 45], and distributed systems [50]. A recent special issue of Journal of Computer Security on verified information flow [60] reflects an active state of the art.

While the progress has been tremendous, it has also given birth to a bewildering breed of concepts, policies, conditions, and enforcement mechanisms. These are often unconnected and ad-hoc, making it difficult to build on when developing new approaches. Thus, when designing information flow controls for a new application domain, the designer is confronted with two basic questions, for which there is no standard recipe in the literature.

Question 1. What is the right security characterization for a new application domain?

A number of information flow conditions has been proposed in the literature. For confidentiality, *noninterference* [22, 28], is a commonly advocated baseline condition stipulating that secret inputs do not affect public outputs. Yet noninterference comes in different styles and flavors: *termination-(in)sensitive* [67, 79], *progress-(in)sensitive* [3], and *timing-sensitive* [2], just to name a few. Other characterizations include *epistemic* [4, 35], *quantitative* [73], and conditions of *information release* [70], as well as *weak* [78], *explicit* [71], and *observable* [9] secrecy. Further, *compositional* security conditions [53, 61, 69] are often advocated, adding to the complexity of choosing the right characterization.

Question 2. What is the right enforcement mechanism for a new application domain?

The designer might struggle to select from the variety of mechanisms available. Information flow enforcement mechanisms have also been proposed in various styles and flavors, including *static* [20, 23, 79], *dynamic* [25, 26, 33], *hybrid* [14, 58], *flow-(in)sensitive* [41, 65], and *language-(in)dependent* [11, 24]. Further, some track *pure data flows* [72] whereas others also track *control flow dependencies* [67], adding to the complexity of choosing the right enforcement mechanism.

Contributions. This paper puts forward principles for designing information flow security definitions and enforcement mechanisms. The goal of the principles is to help roadmapping the state of the art in information flow security, weeding out inconsistencies from the folklore, and providing a rationale for designing information flow characterizations and mechanisms for new application domains.

The rationale rests on the following principles: *attacker-driven security, trust-aware enforcement, separation of policy annotations*

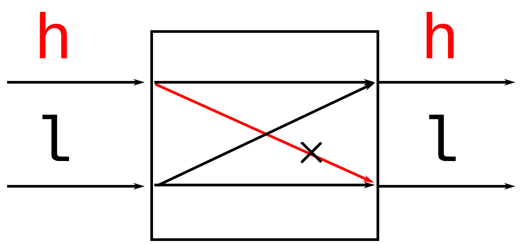
PLAS'18, October 19, 2018, Toronto, ON, Canada

ACM ISBN 978-1-4503-5993-1/18/10...$15.00
https://doi.org/10.1145/3264820.3264824

JSFlow

- information flow tracker for JavaScript
- ECMA/262 v.5 support

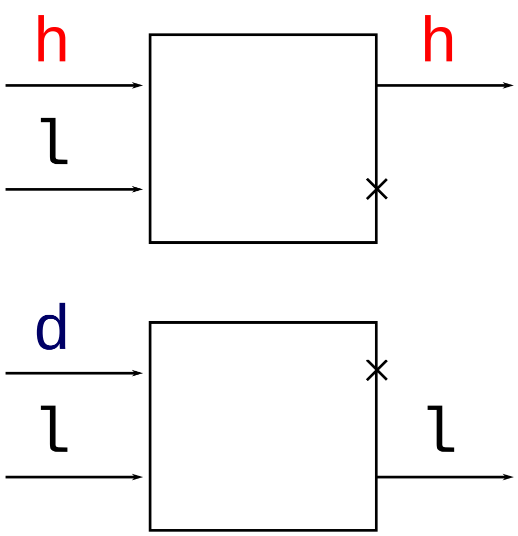
`jsflow.net`



FlowFox

- web browser with information flow control
- based on secure multi-execution

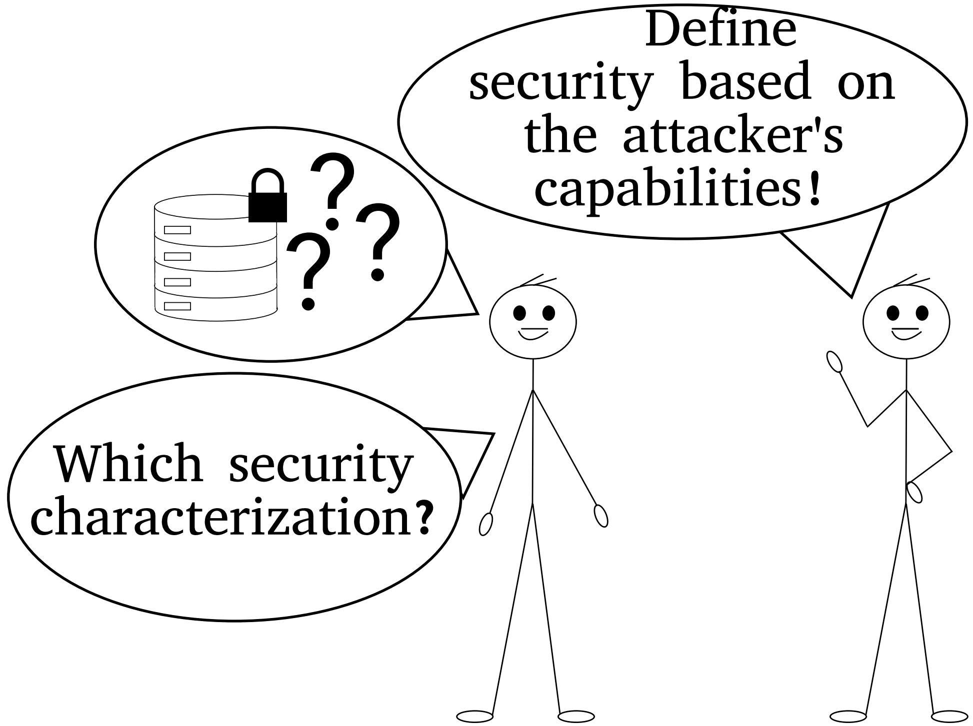
`distrinet.cs.kuleuven.be/software/FlowFox/`



Principle 1

Attacker-driven security

Security characterizations benefit from directly connecting to a behavioral attacker model, expressing (un)desirable behaviors in terms of system events that attackers can observe and trigger.



Progress-sensitive noninterference

- security condition that prevents information leakage via progress channels:
- pairwise execution in low equivalent environments results the same sequences of output, before possibly both diverging

Progress-insensitive noninterference

- security condition that ignores such channels:
- the observable behavior of the program in different low equivalent environments is independent of secrets modulo progress

```
i = 0;
while (i < Number.MAX_VALUE) {
  print(i);
  if (i == secret) {
    while (true) { }
  }
  i = i + 1;
}
```

loops forever

output trace:

```
0
1
.
.
.
secret - 1
```

```
i = 0;
while (i < Number.MAX_VALUE) {
  print(i);
  if (i == secret) {
    while (true) { }
  }
  i = i + 1;
}
```

loops forever

output trace:

```
0
1
.
.
.
secret - 1
```

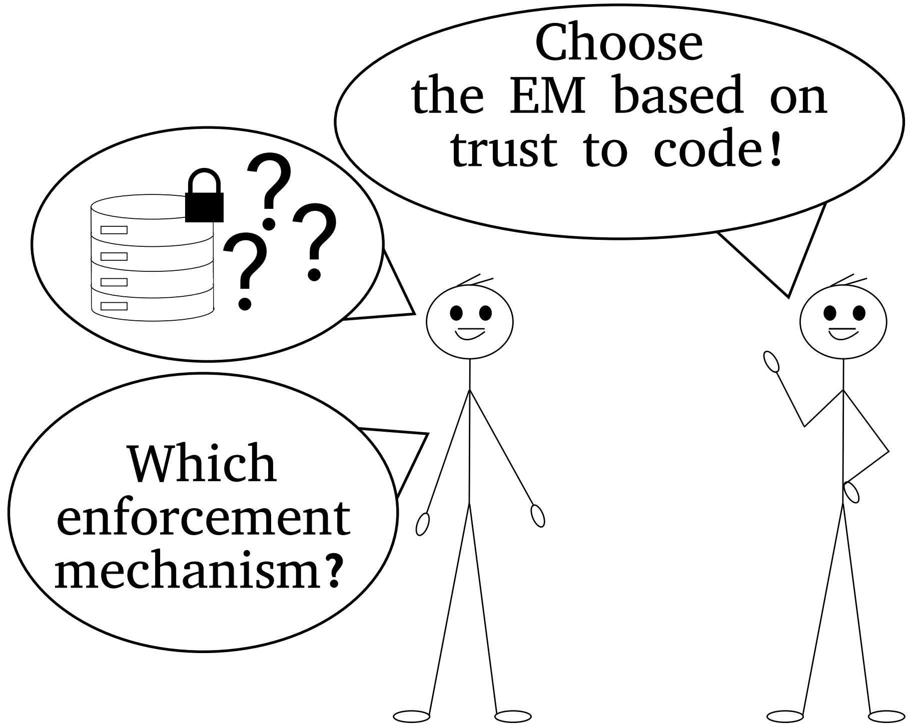
Can the attacker observe intermediate outputs?

- YES => Progress-sensitive insecure
- NO => Progress-insensitive secure (accepted by JSFlow)

Principle 2

Trust-aware security enforcement

Security enforcement benefits from explicit trust assumptions, making clear the boundary between trusted and untrusted computing base and guiding the enforcement design in accord.



```
l = true;
k = true;
if (h) { l = false; }
if (l) { k = false; }
print(42);
```

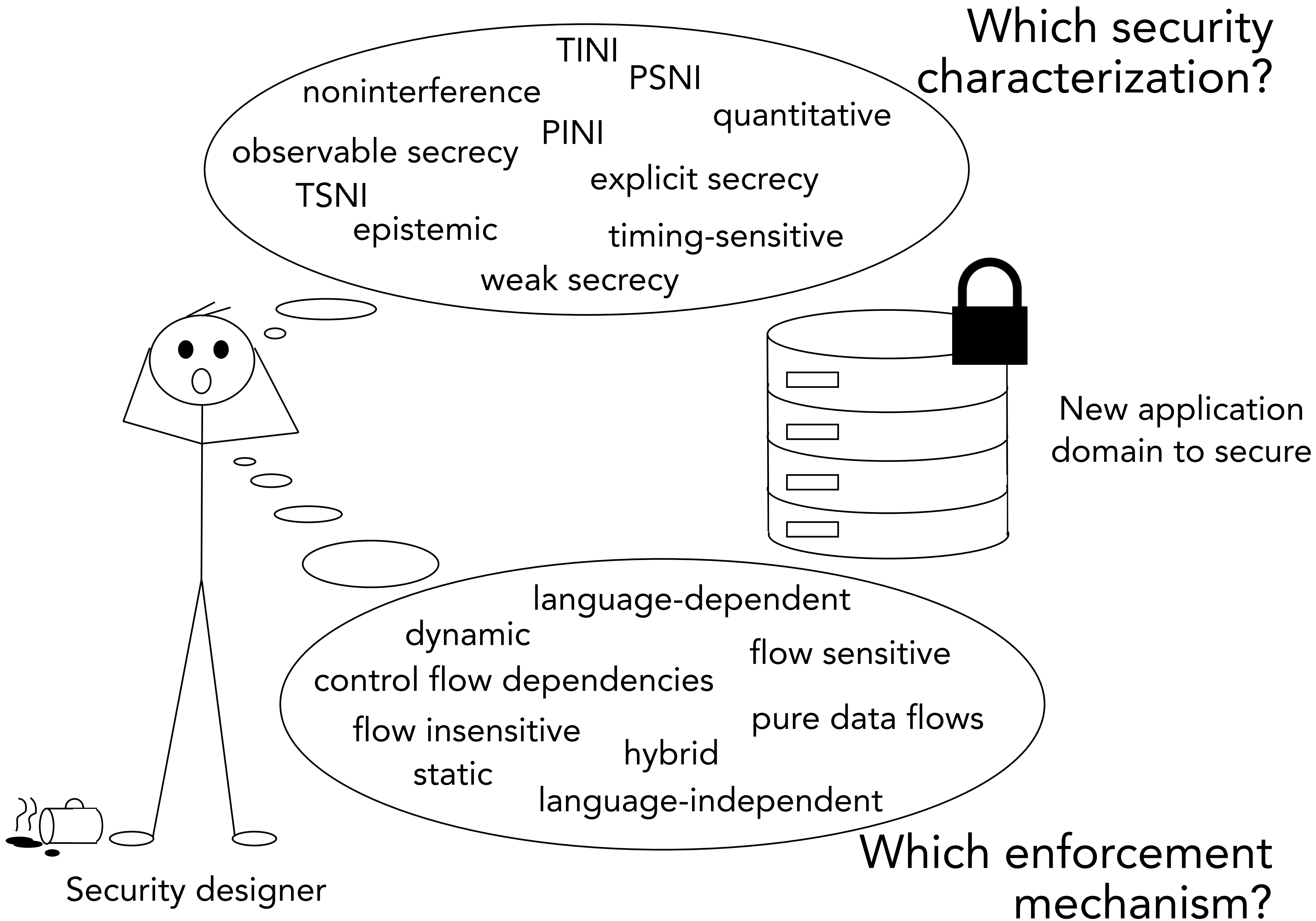
h = true	h = false
l = true	l = true
k = true	k = true
l = false	l = true
k = true	k = false
42	42

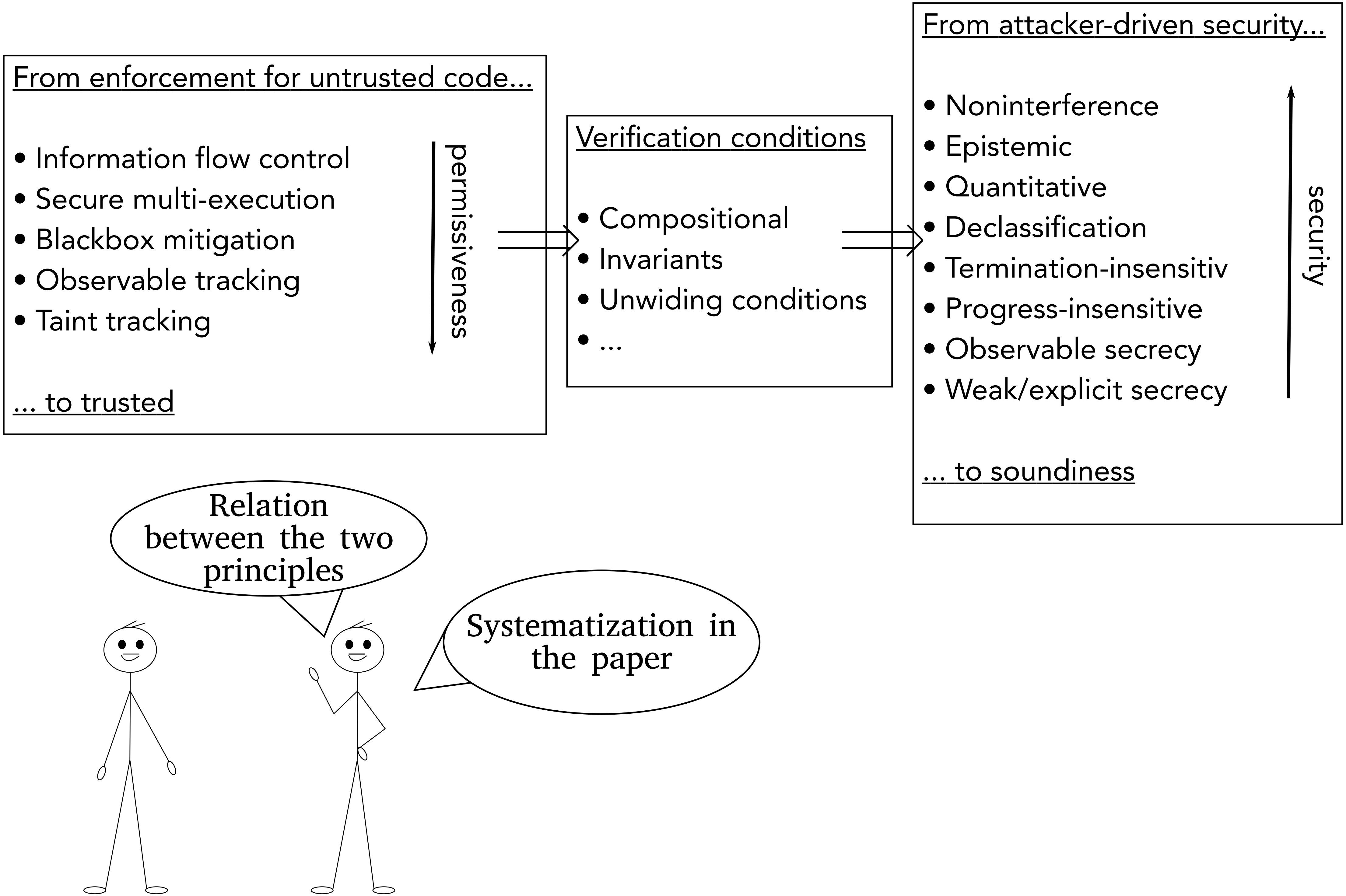
JSFlow execution

	`h = true`	`h = false`
`l = true;`	`l = true`	`l = true`
`k = true;`	`k = true`	`k = true`
`if (h) { l = false; }`	~~`l = false`~~	`l = true`
`if (l) { k = false; }`	~~`k = true`~~	`k = false`
`print(42);`	~~`42`~~	`42`

Is the code trusted?

- YES => accepted by taint tracking
- NO => blocked by JSFlow when `h = true`

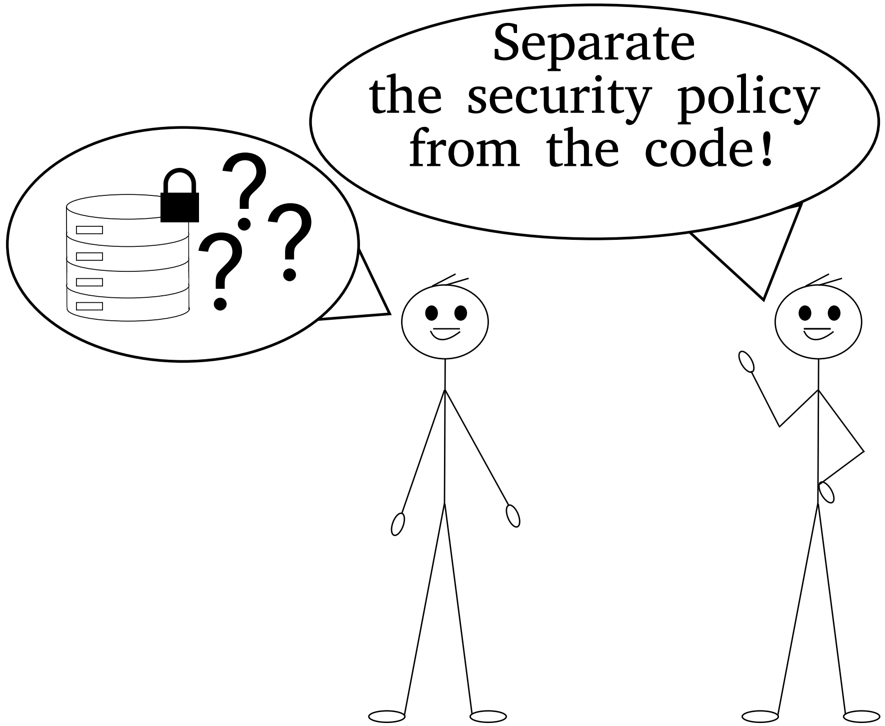

Which security characterization?
TINI
PSNI
noninterference
quantitative
PINI
observable secrecy
explicit secrecy
TSNI
epistemic
timing-sensitive
weak secrecy
New application domain to secure
language-dependent
dynamic
flow sensitive
control flow dependencies
pure data flows
flow insensitive
hybrid
static
language-independent
Security designer
Which enforcement mechanism?


From enforcement for untrusted code...
Information flow control
Secure multi-execution
Blackbox mitigation
Observable tracking
Taint tracking
permissiveness
... to trusted
Verification conditions
Compositional
Invariants
Unwiding conditions
...
From attacker-driven security...
Noninterference
Epistemic
Quantitative
Declassification
Termination-insensitiv
Progress-insensitive
Observable secrecy
Weak/explicit secrecy
security
... to soundiness
Relation between the two principles
Systematization in the paper

Principle 3

Separation of policy annotations and code

Security policy annotations and code benefit from clear separation, especially when the policy is trusted and code is untrusted.



Dimensions of declassification

what

- specifies what partial information about a secret is released
- e.g., parity of secret

where

- specifies where in a system information is released
- e.g. via declassify statements

when

- specifies when information should be released
- e.g., only after a certain time

by whom

- specifies who controls information release in a computing system

JSFlow construct:
defines secret data

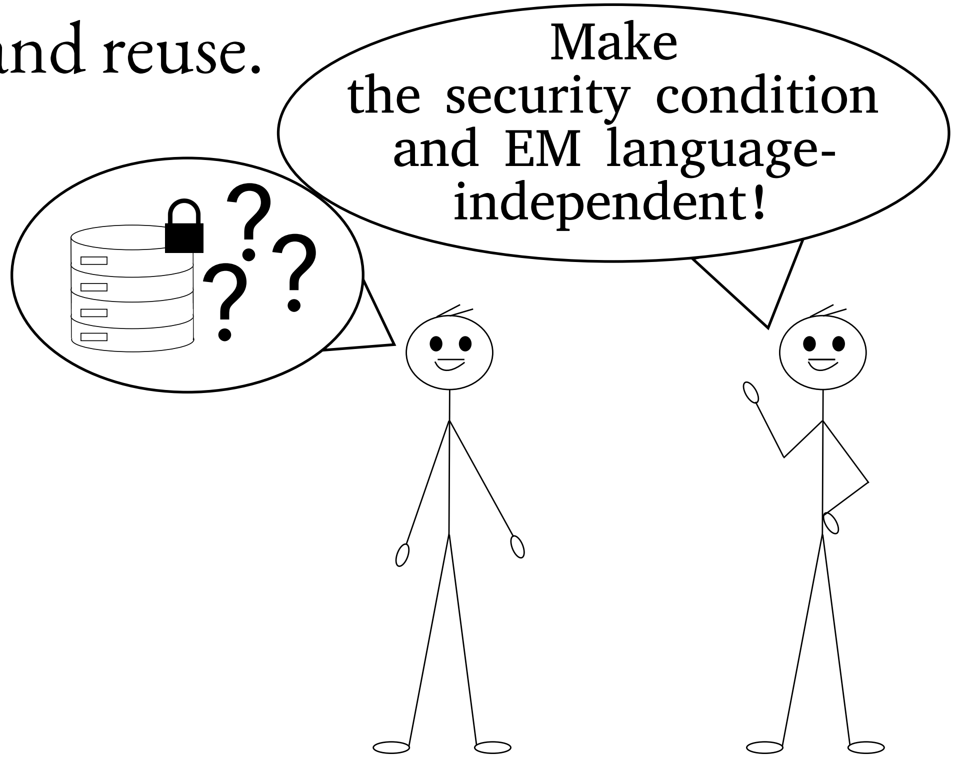
```
guess = lbl(getUserInput());
result = declassify(guess == pwd);
```

- untrusted code => `result = declassify(pwd);`
- strengthen with other dimensions: what, when, by whom
- JSFlow accepts both (only where dimension)

Principle 4

Language independence

Language-independent security conditions benefit from abstracting away from the constructs of the underlying language. Language-independent enforcement benefits from simplicity and reuse.



```
l = true;
k = true;
if (h) { l = false; }
if (l) { k = false; }
print(42);
```

```
h = true
l = true
k = true
l = false
k = true
42
```

```
h = false
l = true
k = true
l = true
k = false
42
```

```
l = true;
k = true;
if (h) { l = false; }
if (l) { k = false; }
print(42);
```

FlowFox execution

High run:	Low run:
`h = true`	`h = false`
`l = true`	`l = true`
`k = true`	`k = true`
`l = false`	`l = true`
`k = true`	`k = false`
~~42~~	42

microflows between
language constructs

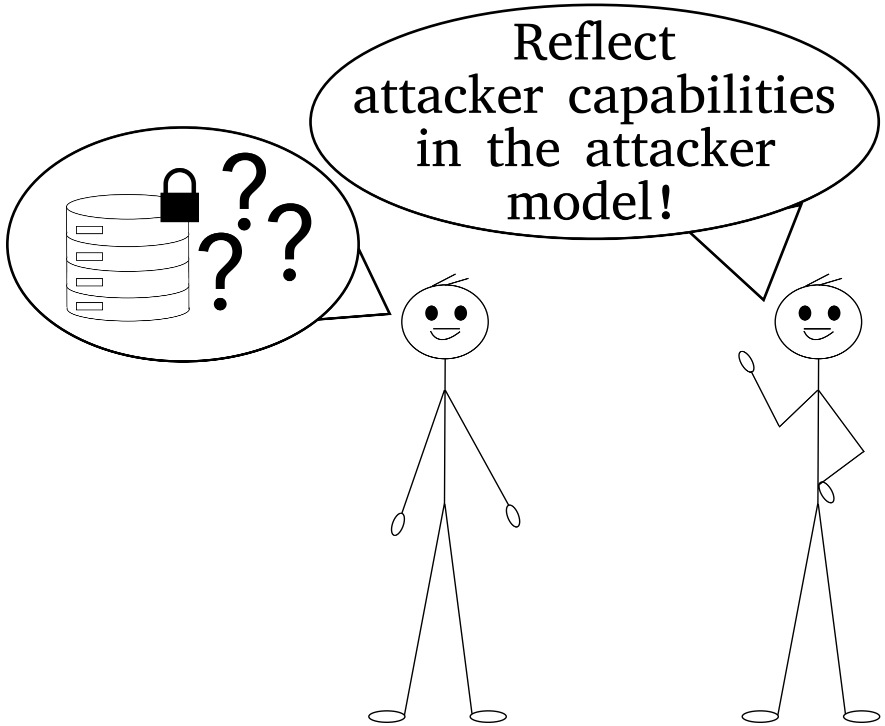
- Execution blocked by JSFlow when `h = true`
- Execution accepted by FlowFox: output 42 always produced

macroflows between
sources and sinks

Principle 5

Justified abstraction

The level of abstraction in the security model benefits from reflecting attacker capabilities.



	h = true	h = false
if (h) {		
$h' = h_1$;	$h' = h_1$	
}		
else {		
$h' = h_2$;		$h' = h_2$
}		
$h' = h_1$;	$h' = h_1$	$h' = h_1$

	h = true	h = false
`if (h) {`		
`  h' = h1;`	`h' = h1`	
`}`		
`else {`		
`  h' = h2;`		`h' = h2`
`}`		
`h' = h1;`	`h' = h1`	`h' = h1`

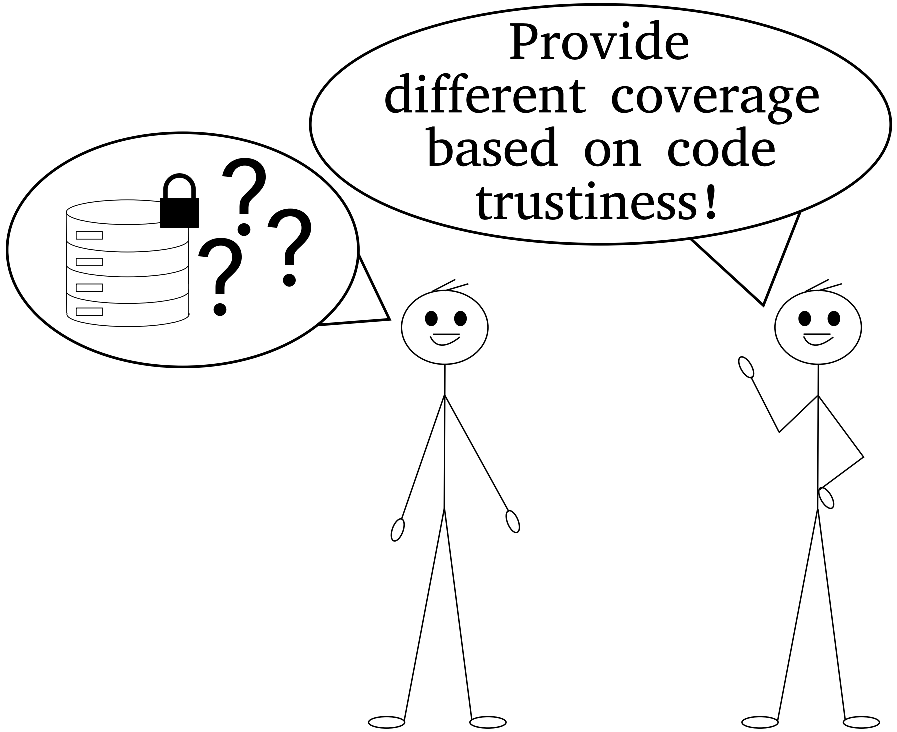
Can the attacker time the execution?

- YES => Attacker learns h: if `h = true` then h_1 in the cache
- NO => Execution accepted by JSFlow (disregards timing)

Principle 6

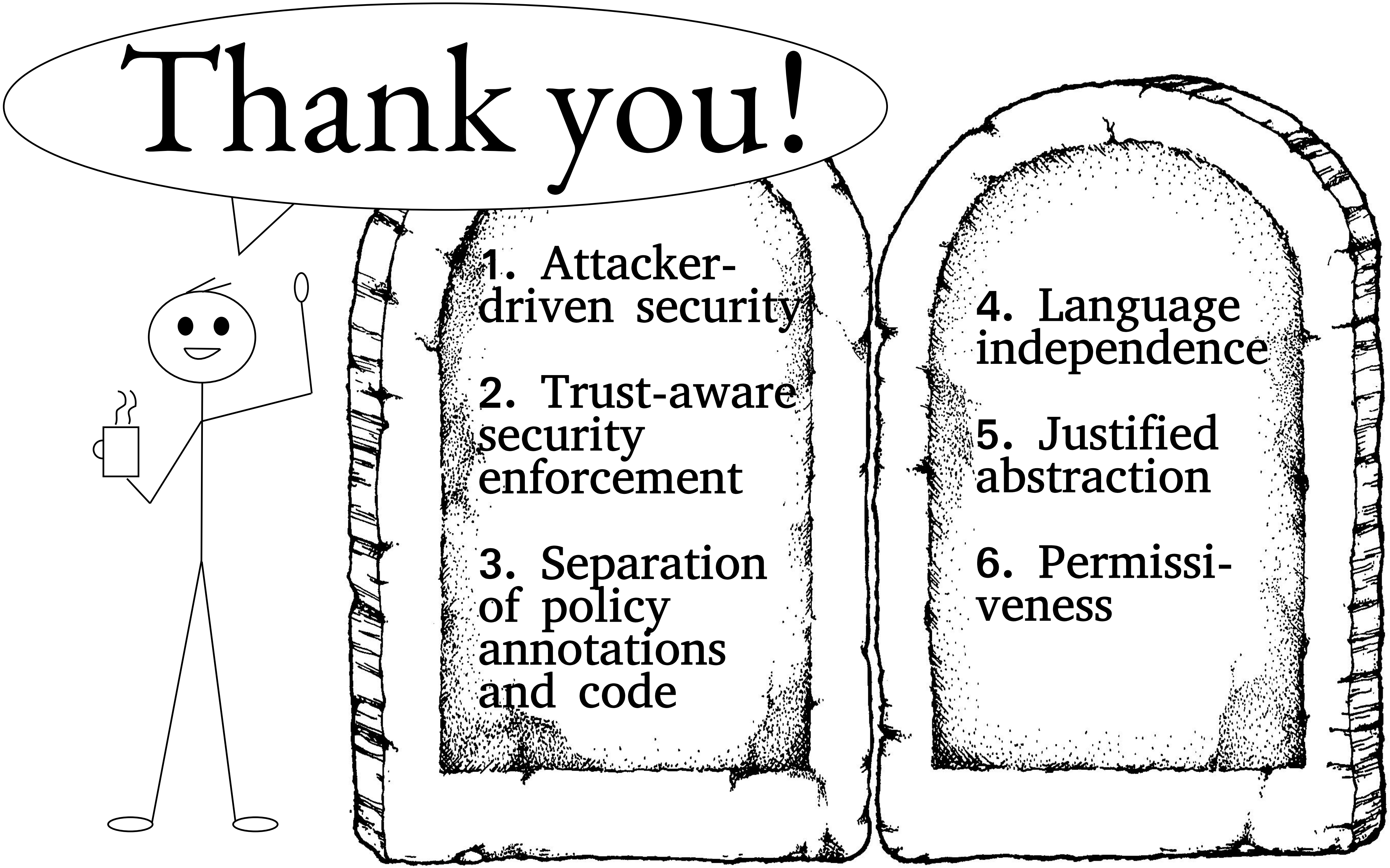
Permissiveness

Enforcement for untrusted code particularly benefits from reducing false negatives (soundness), while enforcement for trusted code particularly benefits from reducing false positives (high permissiveness).



Code	h = true	h = false
	`h = true`	`h = false`
`l = true;`	`l = true`	`l = true`
`k = true;`	`k = true`	`k = true`
`if (h) { l = false; }`	`l = false`	`l = true`
`if (l) { k = false; }`	`k = true`	`k = false`
`print(42);`	`42`	`42`

- false positive for JSFlow when `h = true`
- accepted by taint trackers


Thank you!
1. Attacker-driven security
2. Trust-aware security enforcement
3. Separation of policy annotations and code
4. Language independence
5. Justified abstraction
6. Permissiveness

> Static enforcement mechanisms:

Andrei Sabelfeld and Andrew Myers. Language-based information-flow security. In IEEE Journal on Selected Areas in Communications 21, (2003).

> Dynamic techniques:

Gurvan Le Guernic. 2007. Confidentiality Enforcement Using Dynamic Information Flow Analyses. Ph.D. Dissertation. Kansas State University. `http://tel.archives-ouvertes.fr/tel-00198621/fr/`.

> Dynamic taint analysis and symbolic execution:

Edward J. Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All You Ever Wanted to Know about Dynamic Taint Analysis and Forward Symbolic Execution (but Might Have Been Afraid to Ask). In S&P 2010.

> JavaScript security policies and their enforcement:

Nataliia Bielova. Survey on JavaScript security policies and their enforcement mechanisms in a web browser. In Journal of Log. Algebr. Program, 2013.

> Information flow techniques based on abstract interpretation:

Isabella Mastroeni. Abstract interpretation-based approaches to Security - A Survey on Abstract Non-Interference and its Challenging Applications. arXiv preprint arXiv:1309.5131 129 (2013).