Verification of Security Protocol Implementations

Linard Arquint

NUS — 22. April ‘25

Based on joint work with David Basin, Martin Clochard, Joey Dodds, Samarth Kishor, Jason R. Koenig, Daniel Kroening, Joseph Lallemand, Vaibhav Mehta, Peter Müller, Wytse Oortwijn, João C. Pereira, Ralf Sasse, Malte Schwerhoff, Christoph Sprenger, Sven N. Wiesner, and Felix A. Wolf

ETH zürich

Motivation

Motivation


ars TECHNICA
BIZ & IT TECH SCIENCE POLICY CARS GAMING & CULTURE STORE
A GLITCH IN THE MATRIX —
Serious vulnerabilities in Matrix's end-to-end encryption have been patched
Previously overlooked flaws allow malicious homeservers to decrypt and spoof messages.
DAN GOODIN - 9/28/2022, 9:00 AM


A Traceability Attack against e-Passports
Tom Chothia* and Vitaliy Smirnov
School of Computer Science, University of Birmingham, Birmingham, UK

Motivation

Protocol model

- Formal protocol description
- Pen and paper proofs
- Automatic tools: Tamarin, ProVerif, …
- Successfully applied to many protocols

Motivation

Implementation

Motivation

Correct protocol implementation?

Implementation

Motivation

Correct protocol implementation?

Free of buffer overflows?

Implementation

Motivation

Implementation

Correct protocol implementation?

Free of buffer overflows?

Is it safe, i.e., no panics, divisions by zero, …?

Motivation

Terrapin Attack: Breaking SSH Channel Integrity By Sequence Number Manipulation

Fabian Bäumer
Ruhr University Bochum

Marcus Brinkmann
Ruhr University Bochum

Jörg Schwenk
Ruhr University Bochum

Formal Analysis of Session-Handling in Secure Messaging: Lifting Security from Sessions to Conversations

Cas Cremers
CISPA Helmholtz Center for Information Security

Charlie Jacomme*
Inria Paris, France

Aurora Naska
CISPA Helmholtz Center for Information Security

Overview

Implementation

Part 1: Refine an existing protocol model

Part 2: Use an off-the-shelf program verifier to reason about security properties

Part 3: Outlook — Scale to large codebases and go beyond security properties

Overview

Part 0: Gobra

Implementation

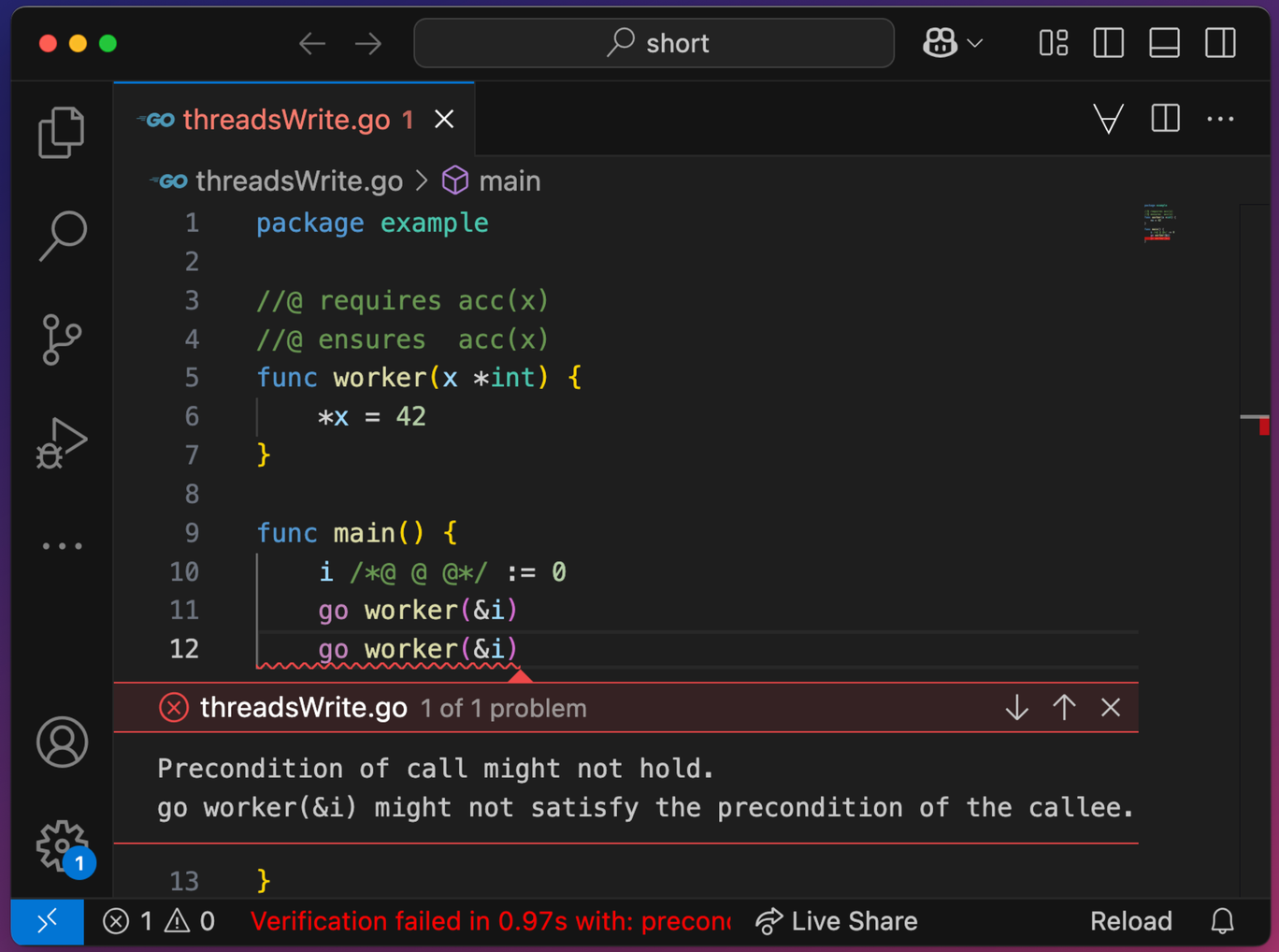
gobra

- Automated program verifier for concurrent Go code
- Modular

https://gobra.ethz.ch

gobra

- Automated program verifier for concurrent Go code
- Modular

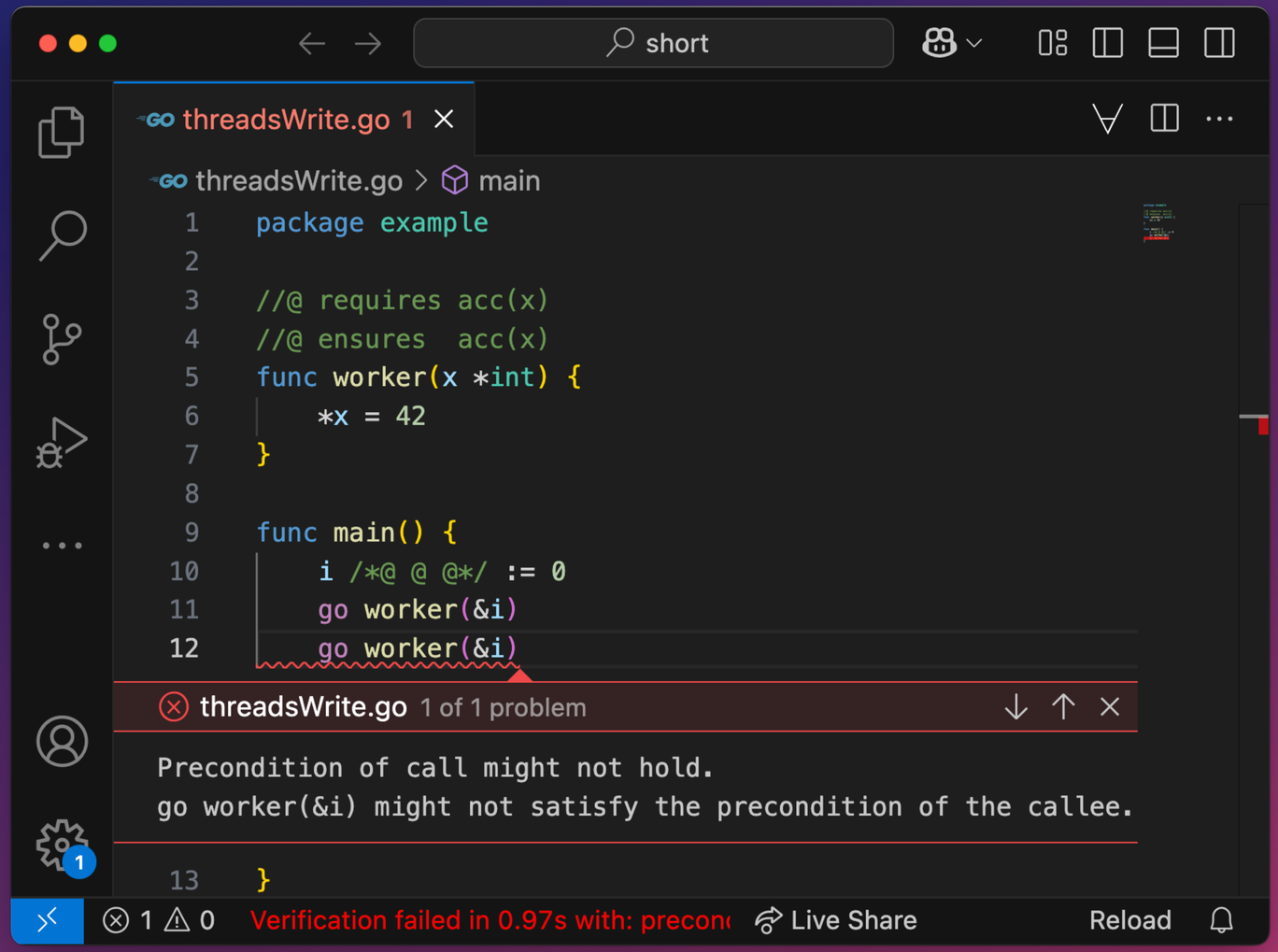


https://gobra.ethz.ch

gobra

- Automated program verifier for concurrent Go code
- Modular
- Memory safety & panic freedom
- Functional properties

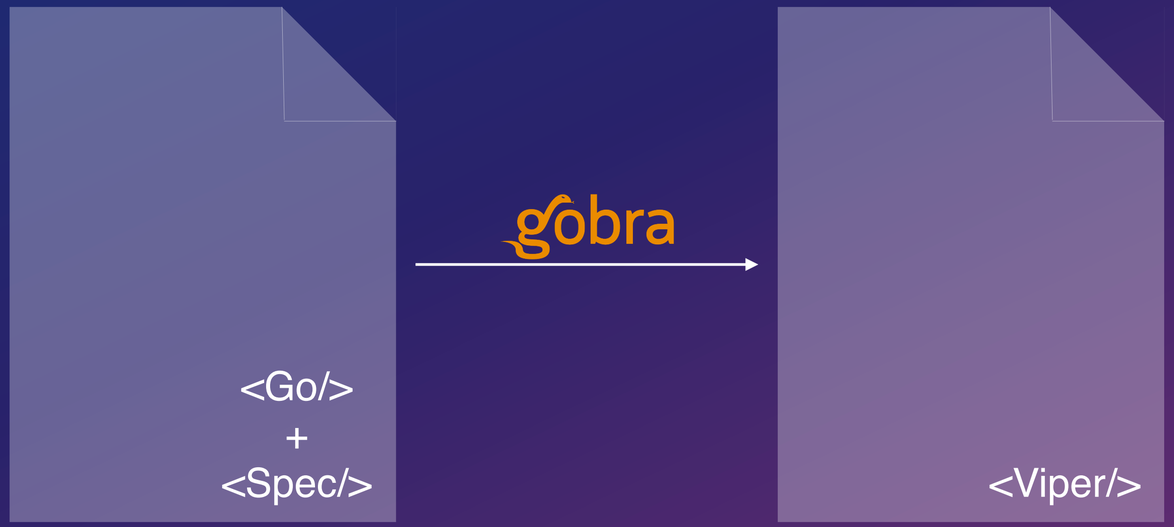
https://gobra.ethz.ch



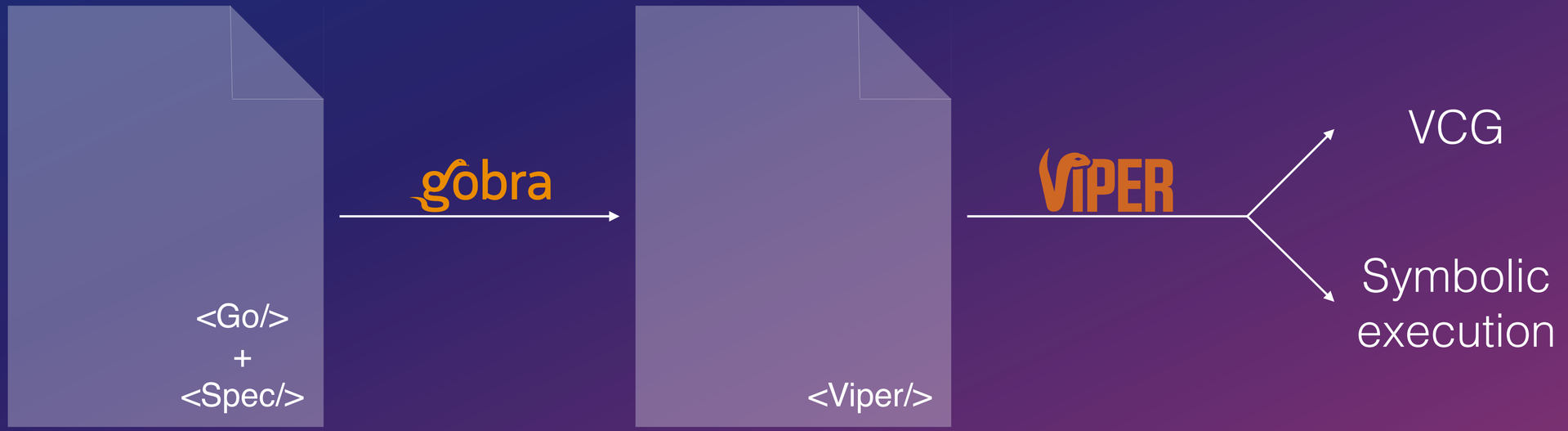
gobra

<Go/>
+
<Spec/>

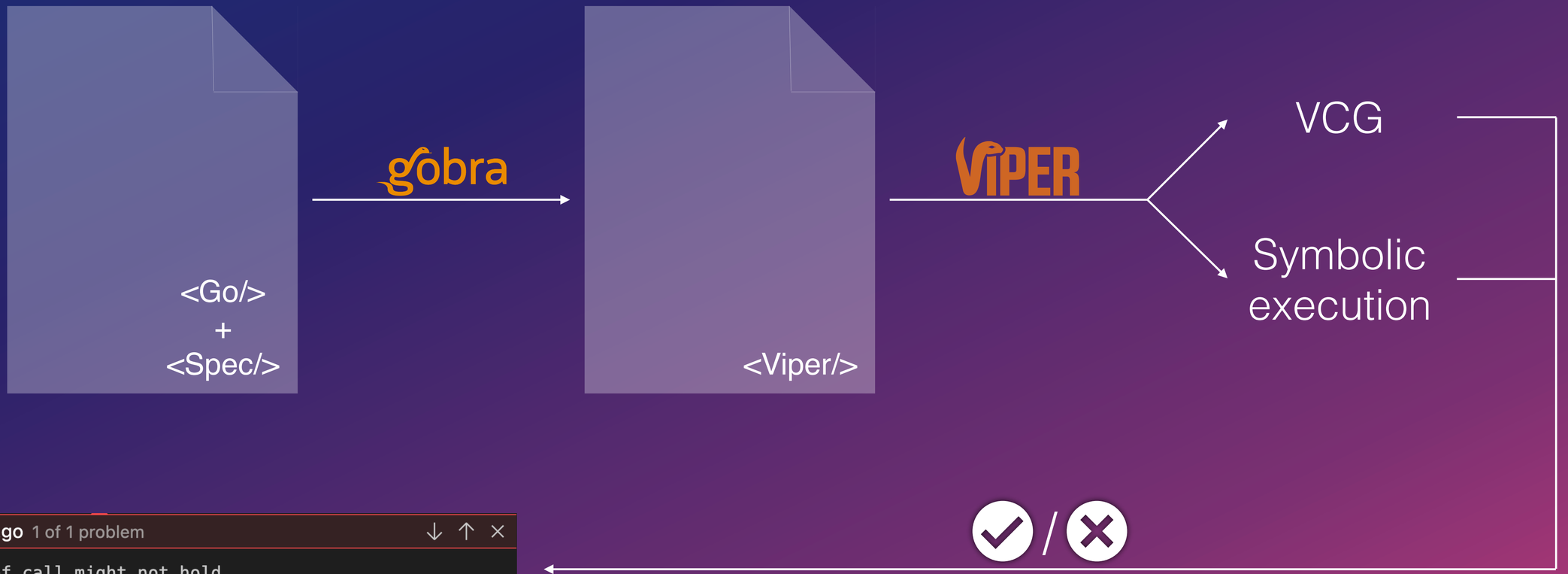
gobra



gobra



gobra


<Go/>
+
<Spec/>
gobra
<Viper/>
VIPER
VCG
Symbolic
execution
threadsWrite.go 1 of 1 problem
Precondition of call might not hold.
go worker(&i) might not satisfy the precondition of the callee.

Overview

Implementation

Part 1: Refine an existing protocol model

Our Approach

Verification of
protocol models
in Tamarin

Verification of
implementations
in program verifiers

Our Approach

Verification of
protocol models
in Tamarin

Verification of
implementations
in program verifiers

Property preservation

Protocol model
in Tamarin

1

satisfies

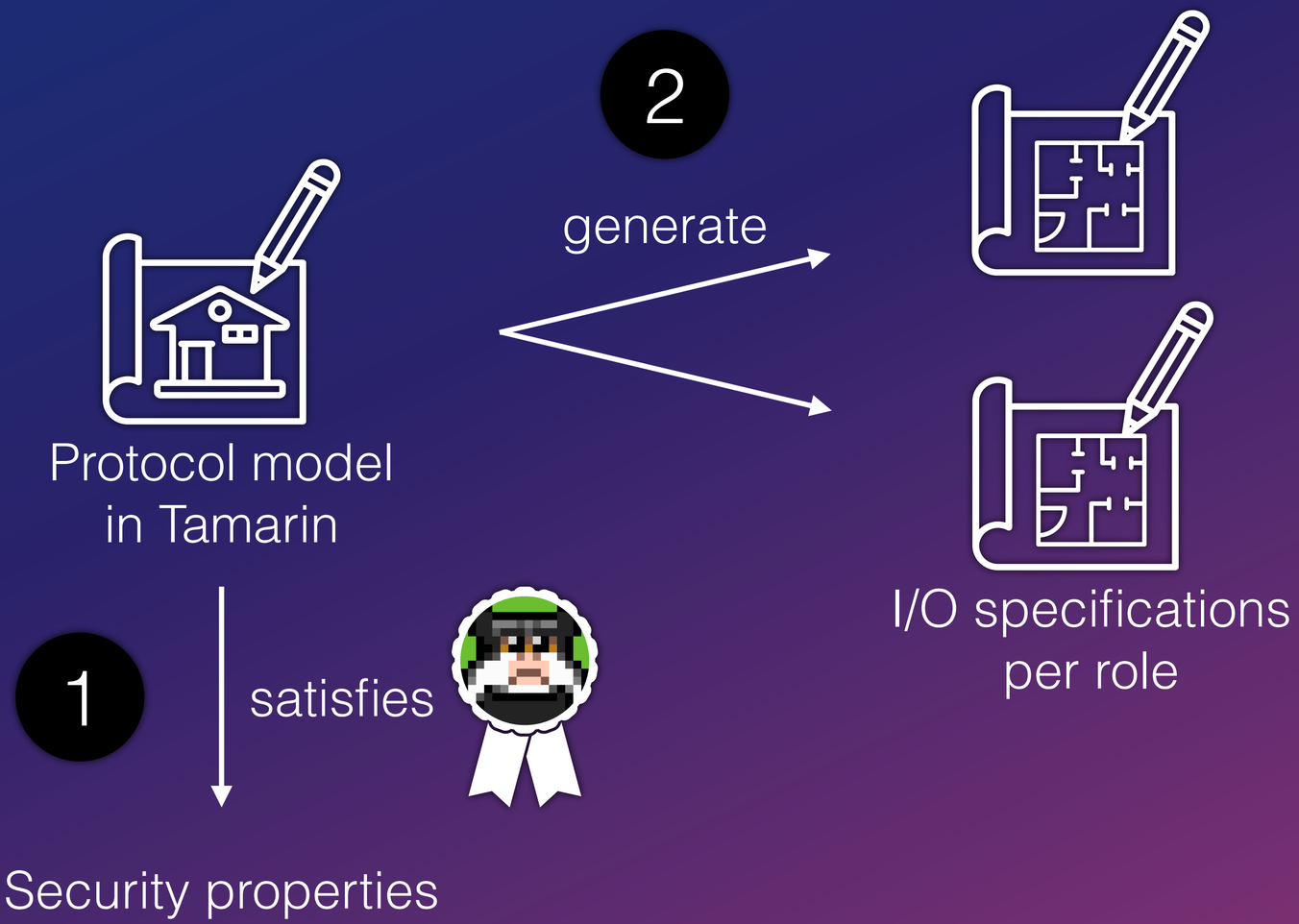
Security properties

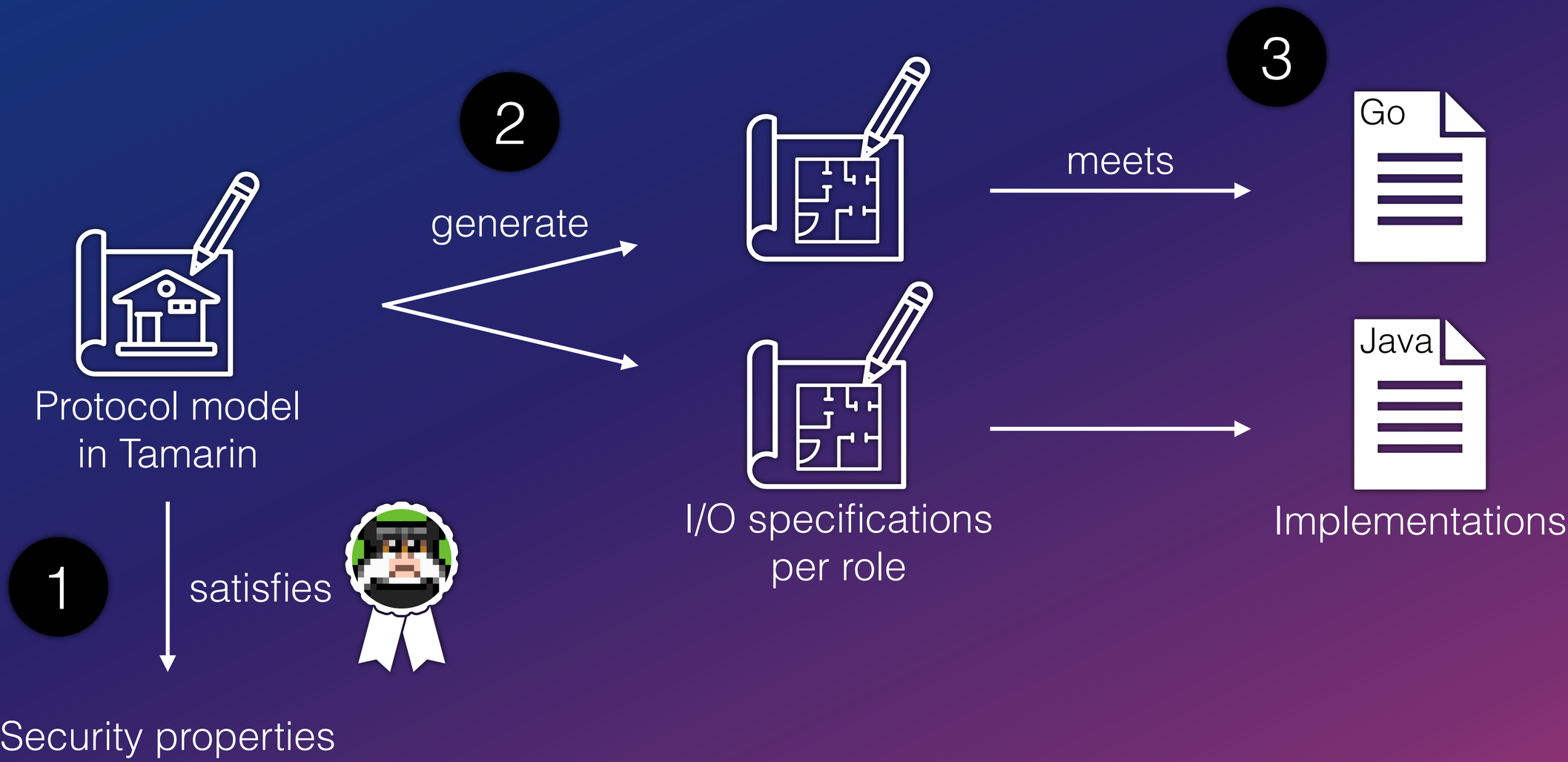
Protocol model
in Tamarin

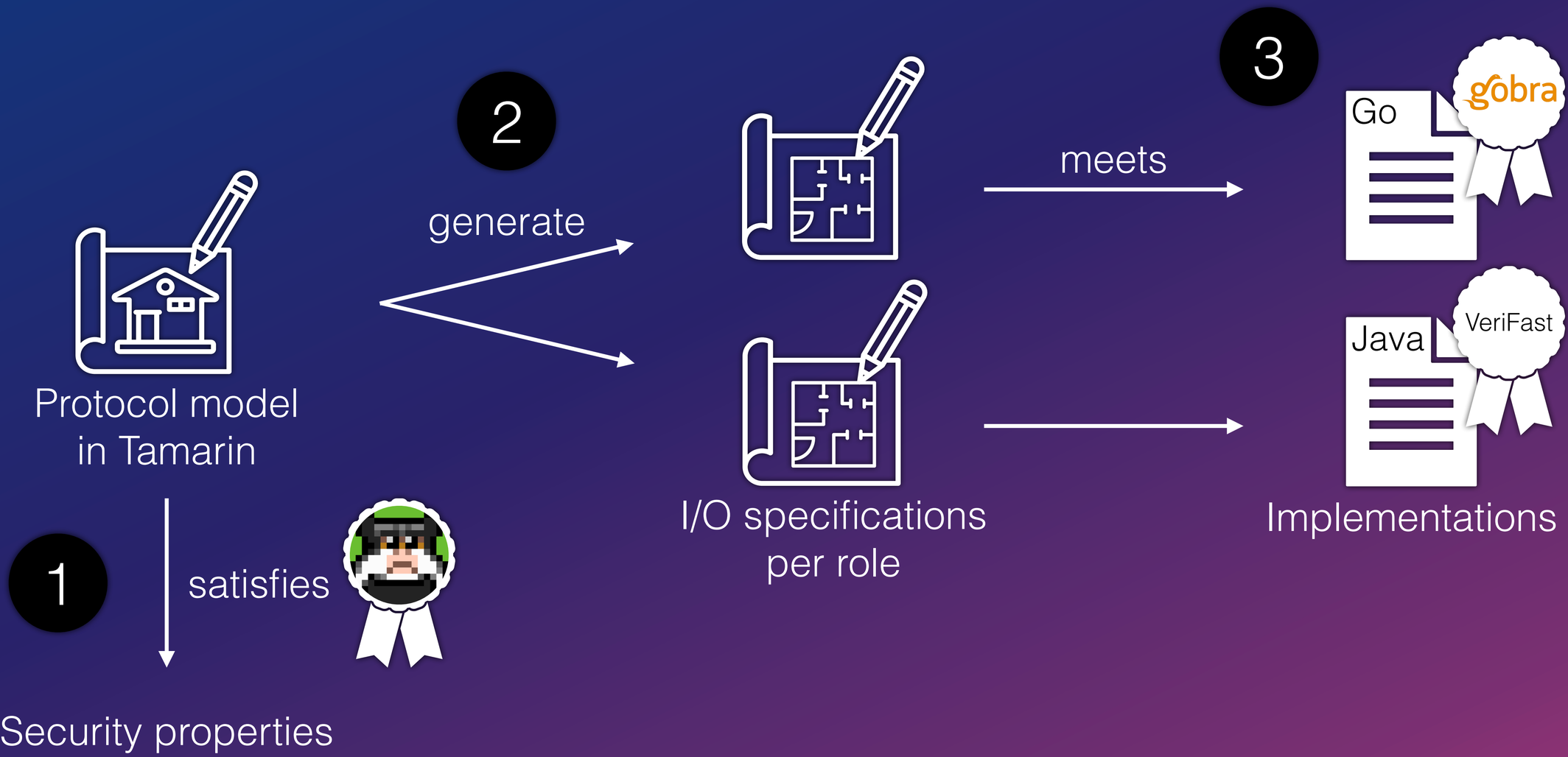
1

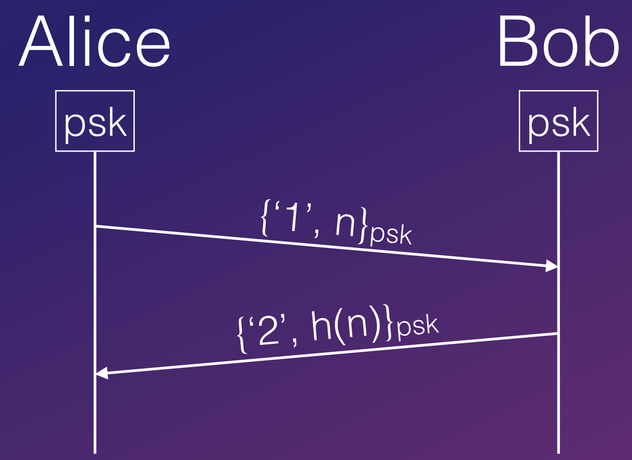
satisfies

Security properties

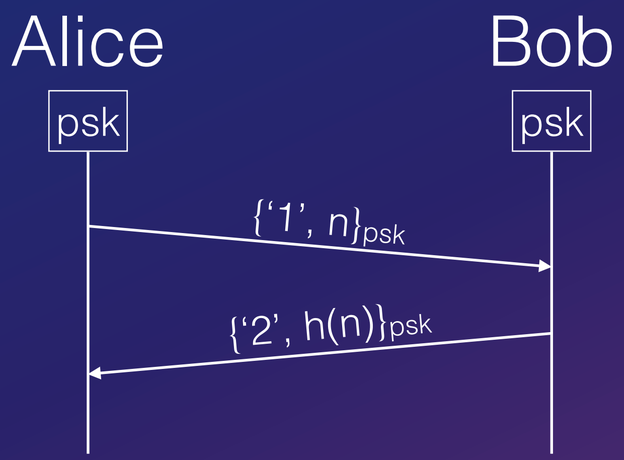

2
generate
Protocol model
in Tamarin
I/O specifications
per role
1
satisfies
Security properties

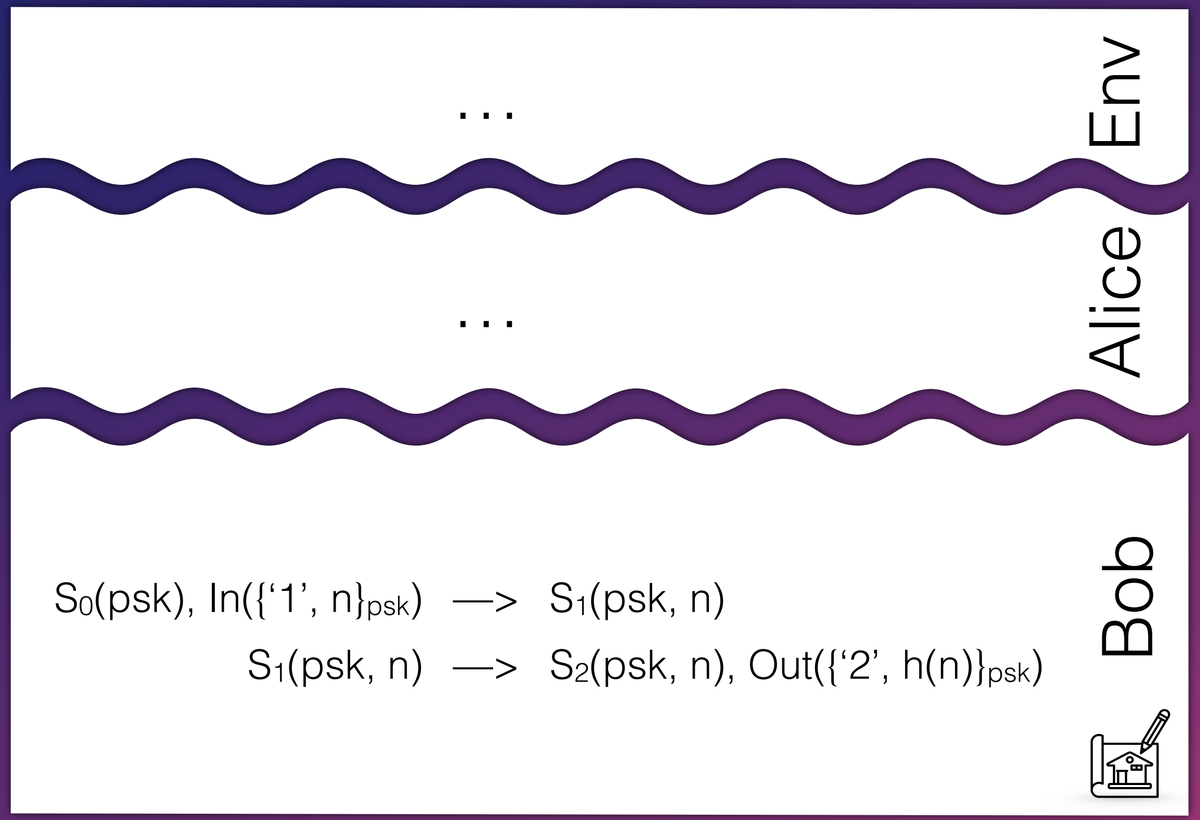

3
2
meets
Go
generate
Java
Protocol model
in Tamarin
I/O specifications
per role
Implementations
1
satisfies
Security properties


3
2
generate
meets
Go
gobra
Protocol model
in Tamarin
Java
VeriFast
I/O specifications
per role
Implementations
1
satisfies
Security properties


Alice
Bob
psk
psk
{‘1’, n}psk
{‘2’, h(n)}psk

1


Alice
Bob
psk
psk
{‘1’, n}psk
{‘2’, h(n)}psk


Env
Alice
Bob
...
...
S0(psk), In({‘1’, n}psk) —> S1(psk, n)
S1(psk, n) —> S2(psk, n), Out({‘2’, h(n)}psk)

3

$S_0(psk)$, $In(\{‘1’, n\}_{psk})$ —> $S_1(psk, n)$

$S_1(psk, n)$ —> $S_2(psk, n)$, $Out(\{‘2’, h(n)\}_{psk})$

Bob

```
func main(psk []byte) {


  m1 := recv()

  // parse m1
  p1, ok := sdecrypt(m1, psk)
  if !ok { return }

  tag1, n := destruct(p1)
  if tag1 != 1 { return }




  p2 := construct(2, hash(n))
  m2 := sencrypt(p2, psk)



  send(m2)
}
```

3

$S_0(psk)$, $In(\{‘1’, n\}_{psk})$ —> $S_1(psk, n)$

$S_1(psk, n)$ —> $S_2(psk, n)$, $Out(\{‘2’, h(n)\}_{psk})$

Bob

```
//@ requires Bob_IO_Spec()
func main(psk []byte) {


  m1 := recv()

  // parse m1
  p1, ok := sdecrypt(m1, psk)
  if !ok { return }

  tag1, n := destruct(p1)
  if tag1 != 1 { return }




  p2 := construct(2, hash(n))
  m2 := sencrypt(p2, psk)



  send(m2)
}
```

3

$S_0(psk)$, $In(\{‘1’, n\}_{psk})$ —> $S_1(psk, n)$

$S_1(psk, n)$ —> $S_2(psk, n)$, $Out(\{‘2’, h(n)\}_{psk})$

Bob

```
//@ requires Bob_IO_Spec()
func main(psk []byte) {
  // obtain permission to receive
  //@ apply_receive_transition()
  m1 := recv()

  // parse m1
  p1, ok := sdecrypt(m1, psk)
  if !ok { return }

  tag1, n := destruct(p1)
  if tag1 != 1 { return }



  p2 := construct(2, hash(n))
  m2 := sencrypt(p2, psk)



  send(m2)
}
```

3

$S_0(psk)$, $In(\{‘1’, n\}_{psk})$ —> $S_1(psk, n)$

$S_1(psk, n)$ —> $S_2(psk, n)$, $Out(\{‘2’, h(n)\}_{psk})$

Bob

```
//@ requires Bob_IO_Spec()
func main(psk []byte) {
  // obtain permission to receive
  //@ apply_receive_transition()
  m1 := recv()

  // parse m1
  p1, ok := sdecrypt(m1, psk)
  if !ok { return }

  tag1, n := destruct(p1)
  if tag1 != 1 { return }

  //@ apply_transition_1()


  p2 := construct(2, hash(n))
  m2 := sencrypt(p2, psk)



  send(m2)
}
```

3

$S_0(psk)$, $In(\{‘1’, n\}_{psk})$ —> $S_1(psk, n)$

$S_1(psk, n)$ —> $S_2(psk, n)$, $Out(\{‘2’, h(n)\}_{psk})$

Bob

```
//@ requires Bob_IO_Spec()
func main(psk []byte) {
  // obtain permission to receive
  //@ apply_receive_transition()
  m1 := recv()

  // parse m1
  p1, ok := sdecrypt(m1, psk)
  if !ok { return }

  tag1, n := destruct(p1)
  if tag1 != 1 { return }

  //@ apply_transition_1()
  //@ apply_transition_2()

  p2 := construct(2, hash(n))
  m2 := sencrypt(p2, psk)



  send(m2)
}
```

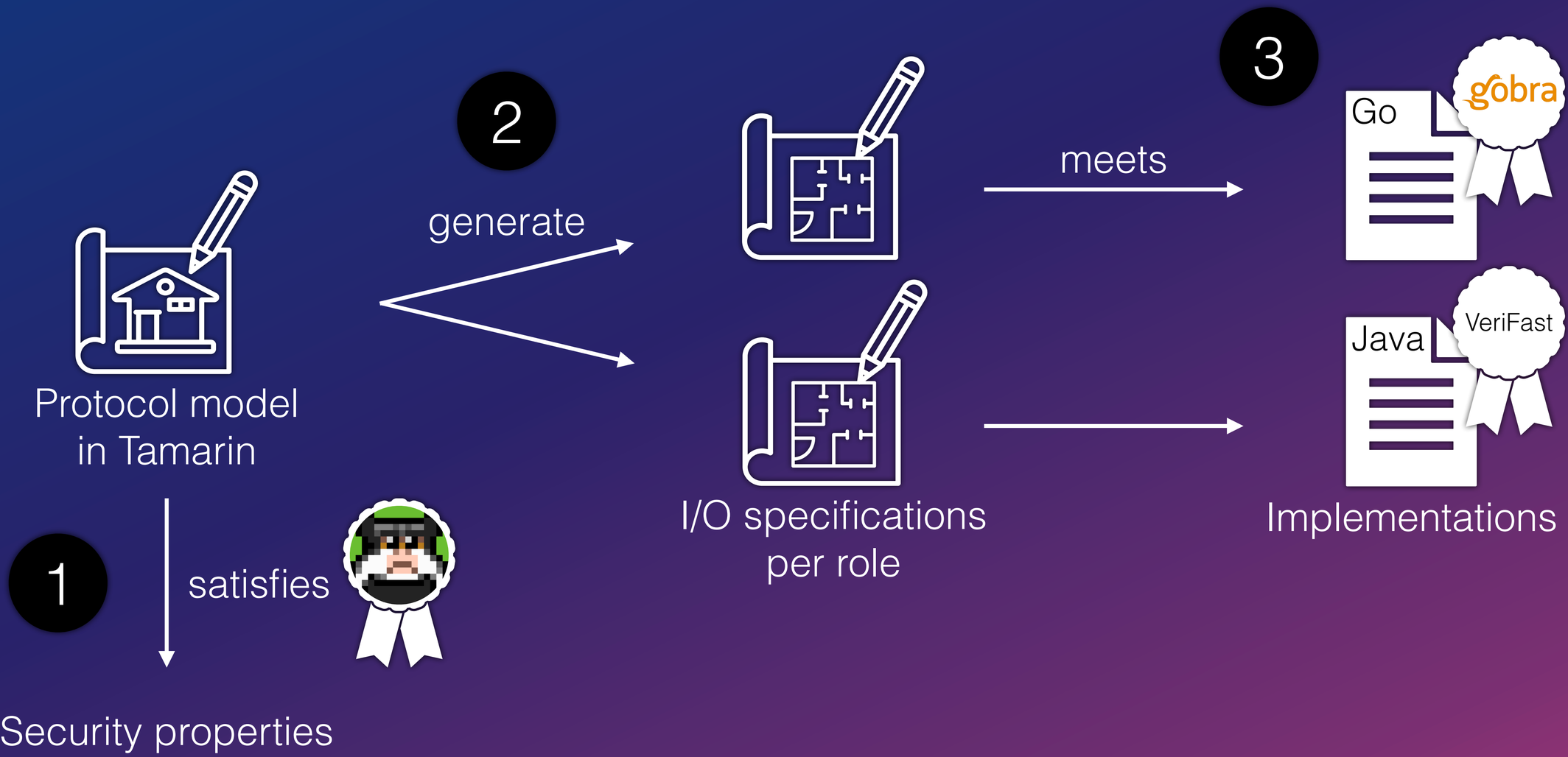
3

$S_0(psk)$, $In(\{‘1’, n\}_{psk})$ —> $S_1(psk, n)$

$S_1(psk, n)$ —> $S_2(psk, n)$, $Out(\{‘2’, h(n)\}_{psk})$

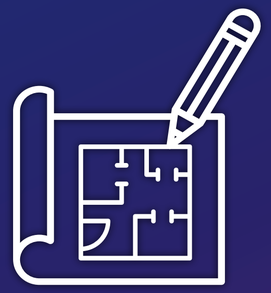
Bob

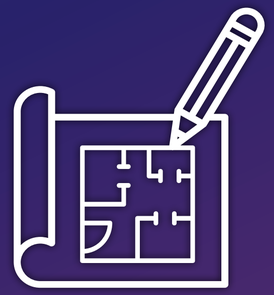
```
//@ requires Bob_IO_Spec()
func main(psk []byte) {
  // obtain permission to receive
  //@ apply_receive_transition()
  m1 := recv()

  // parse m1
  p1, ok := sdecrypt(m1, psk)
  if !ok { return }

  tag1, n := destruct(p1)
  if tag1 != 1 { return }

  //@ apply_transition_1()
  //@ apply_transition_2()

  p2 := construct(2, hash(n))
  m2 := sencrypt(p2, psk)

  // obtain permission to send
  //@ apply_send_transition()
  send(m2)
}
```


2
generate
3
meets
Protocol model
in Tamarin
I/O specifications
per role
Go
gobra
Java
VeriFast
Implementations
1
satisfies
Security properties

Protocol model
in Tamarin

generate





I/O specifications
per role

meets

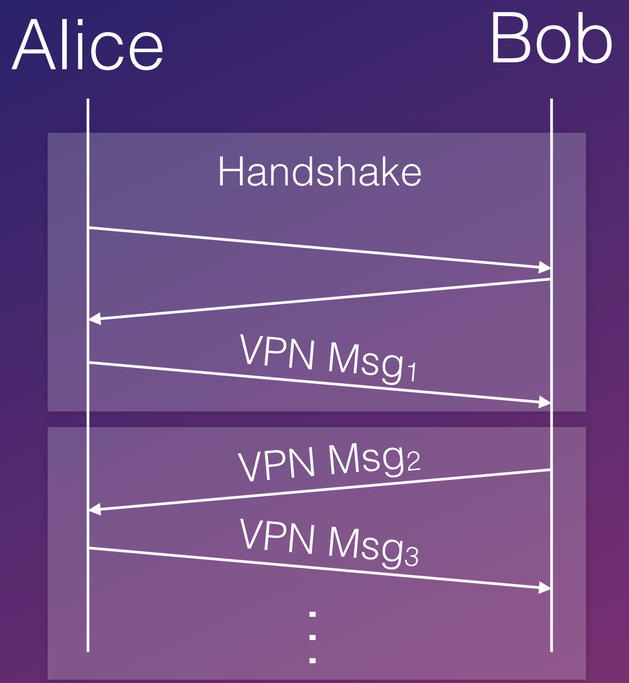
Implementations

satisfies

Security properties ⊇ Tr(Specifications) ⊇ Tr(Implementations)

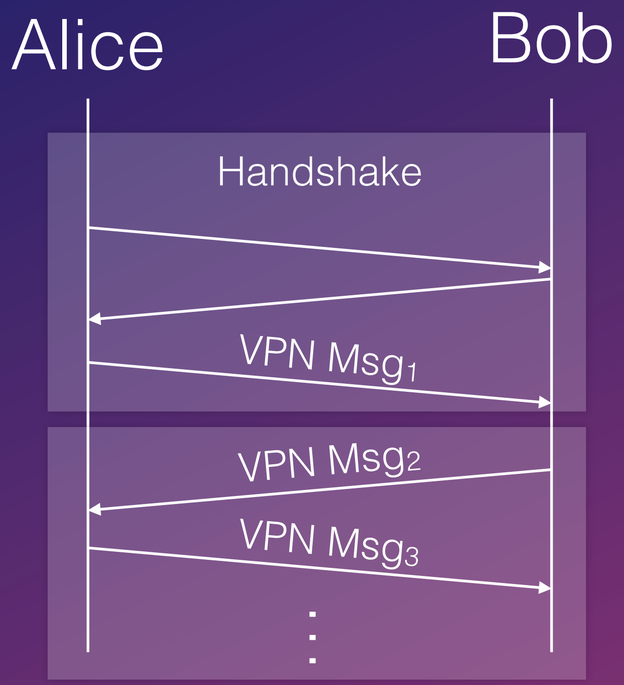
WIREGUARD®
FAST, MODERN, SECURE VPN TUNNEL

- VPN protocol consisting of handshake & transport phase
- ~350 LoC Tamarin model



WIREGUARD®
FAST, MODERN, SECURE VPN TUNNEL

- VPN protocol consisting of handshake & transport phase
- ~350 LoC Tamarin model
- ~600 LoC Go code
- ~1.2k lines of generated I/O spec
- ~2.8k lines of proof annotations



gobra

Conclusions

Verification of
protocol models
in Tamarin

Verification of
implementations
in program verifiers

Sound end-to-end verification

S&P ’23:
Sound Verification of Security Protocols:
From Design to Interoperable Implementations

Automatic specification generation

Soundness proof &
novel approach to relate
symbolic terms and bytes

Conclusions

Verification of

Verification of

What if we do not have a protocol model?

Sound end-to-end verification

S&P ’23:
Sound Verification of Security Protocols:
From Design to Interoperable Implementations

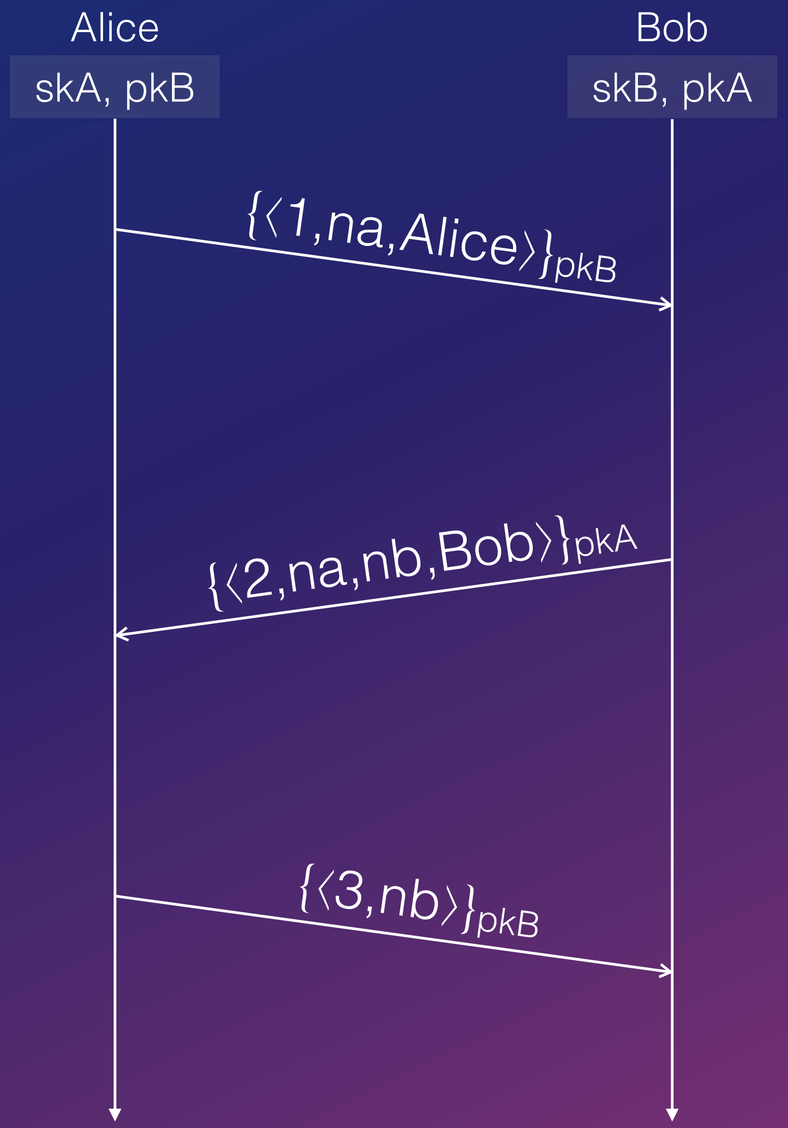
Automatic specification generation

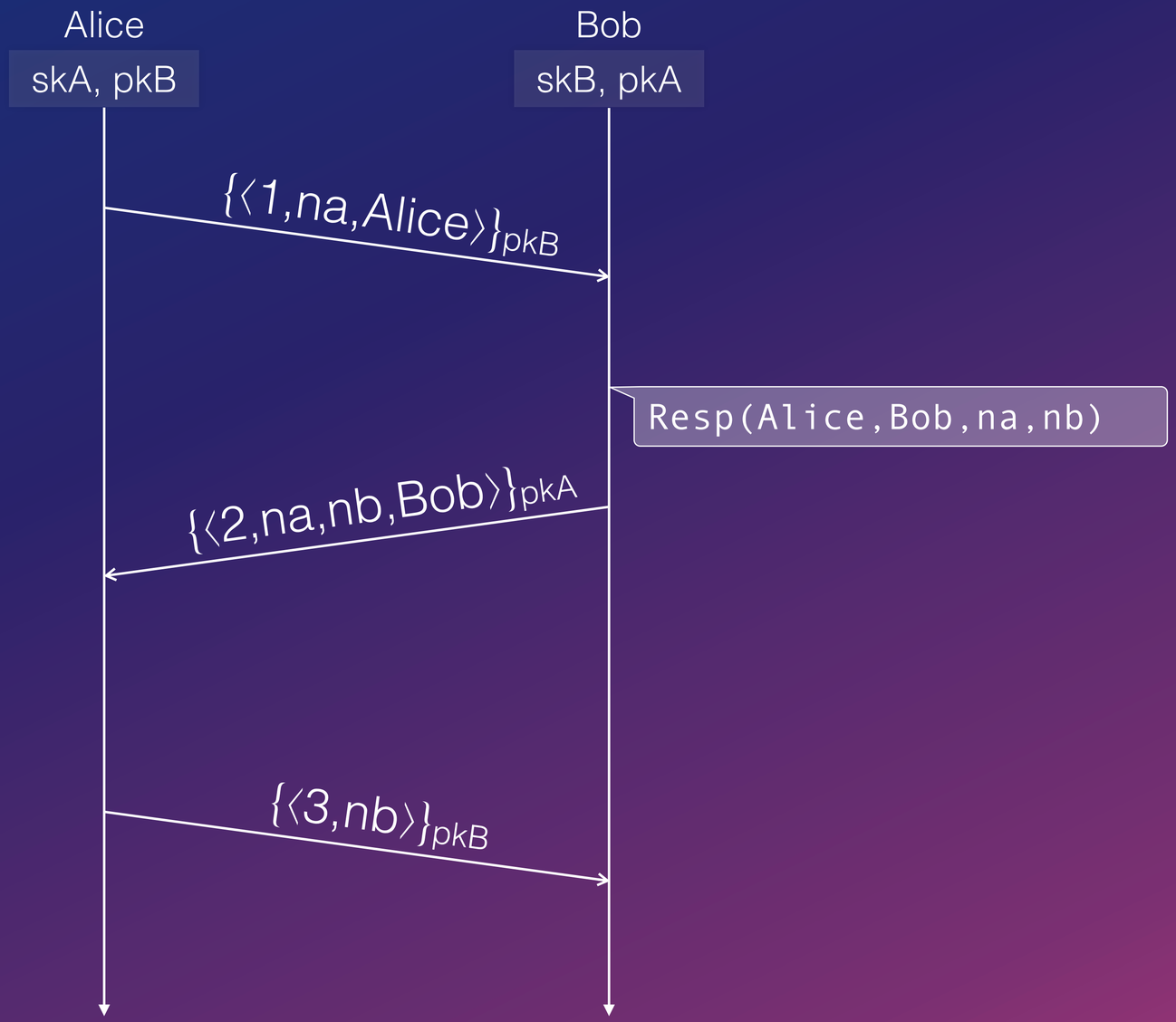
Soundness proof &
novel approach to relate
symbolic terms and bytes

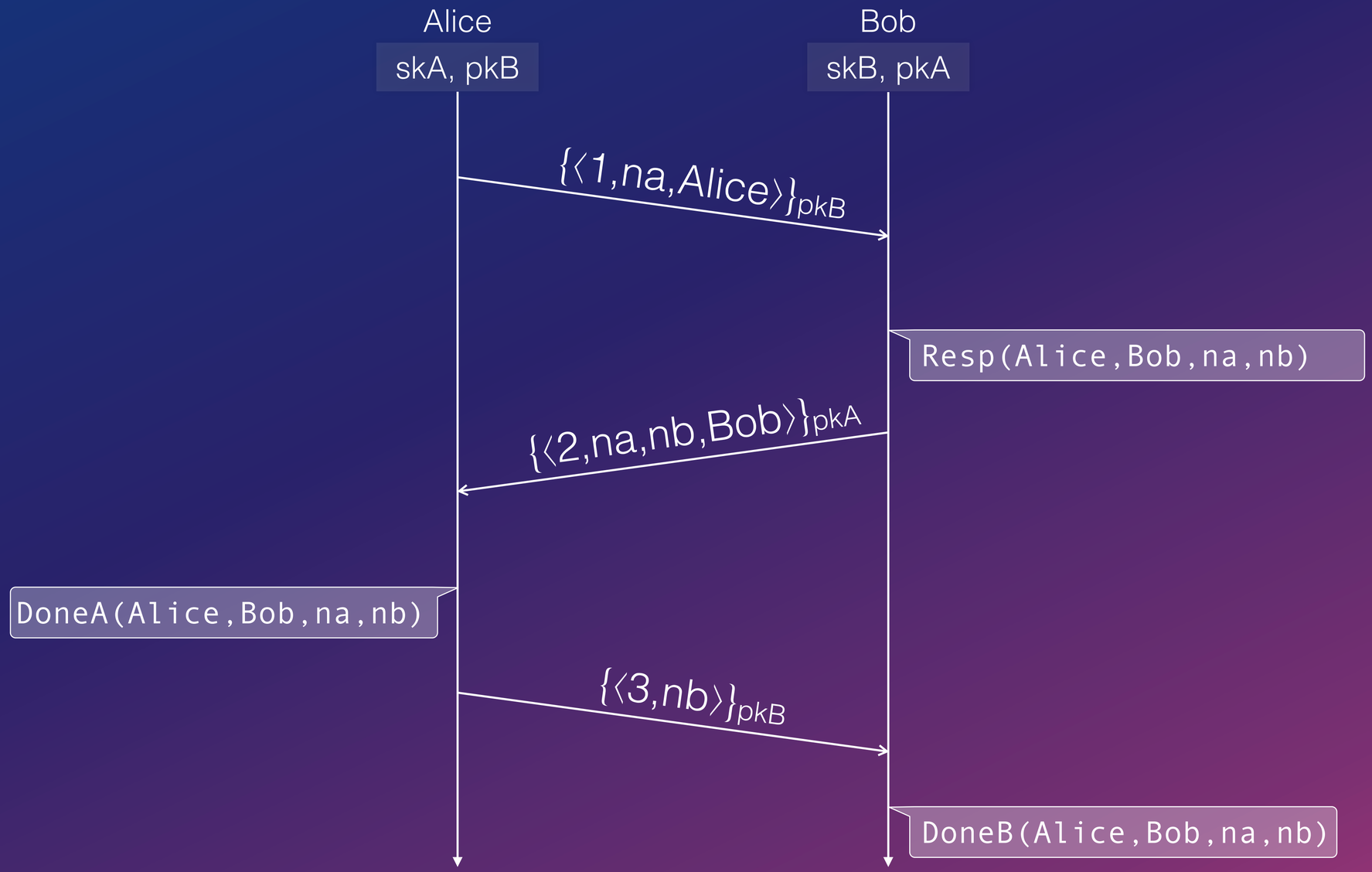
Overview

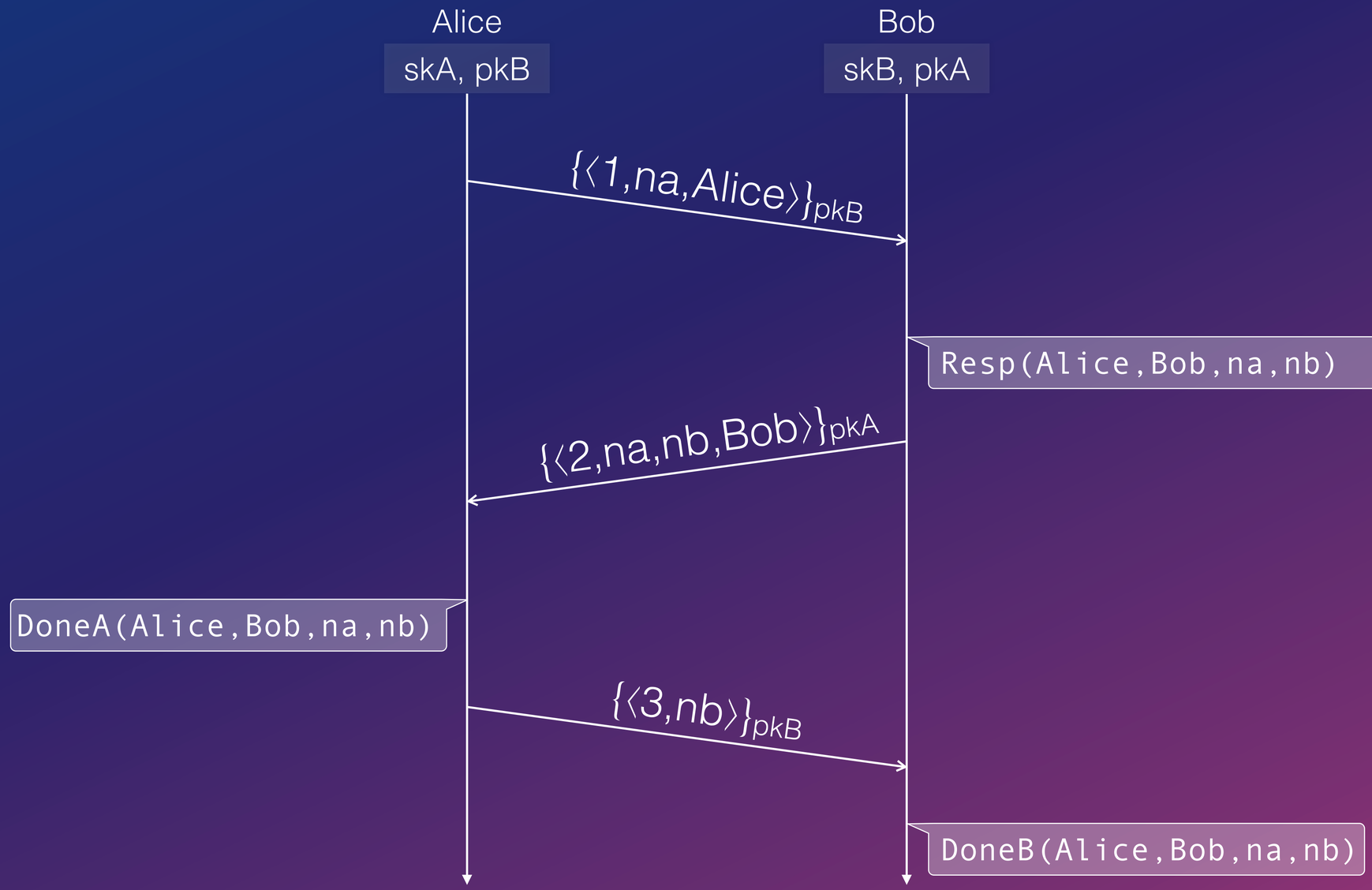
Implementation

Part 2: Use an off-the-shelf program verifier to reason about security properties


Alice
skA, pkB
Bob
skB, pkA
{⟨1,na,Alice⟩}pkB
{⟨2,na,nb,Bob⟩}pkA
{⟨3,nb⟩}pkB


Alice
skA, pkB
Bob
skB, pkA
{⟨1,na,Alice⟩}pkB
Resp(Alice,Bob,na,nb)
{⟨2,na,nb,Bob⟩}pkA
{⟨3,nb⟩}pkB


Alice
skA, pkB
Bob
skB, pkA
{⟨1,na,Alice⟩}pkB
Resp(Alice,Bob,na,nb)
{⟨2,na,nb,Bob⟩}pkA
DoneA(Alice,Bob,na,nb)
{⟨3,nb⟩}pkB
DoneB(Alice,Bob,na,nb)

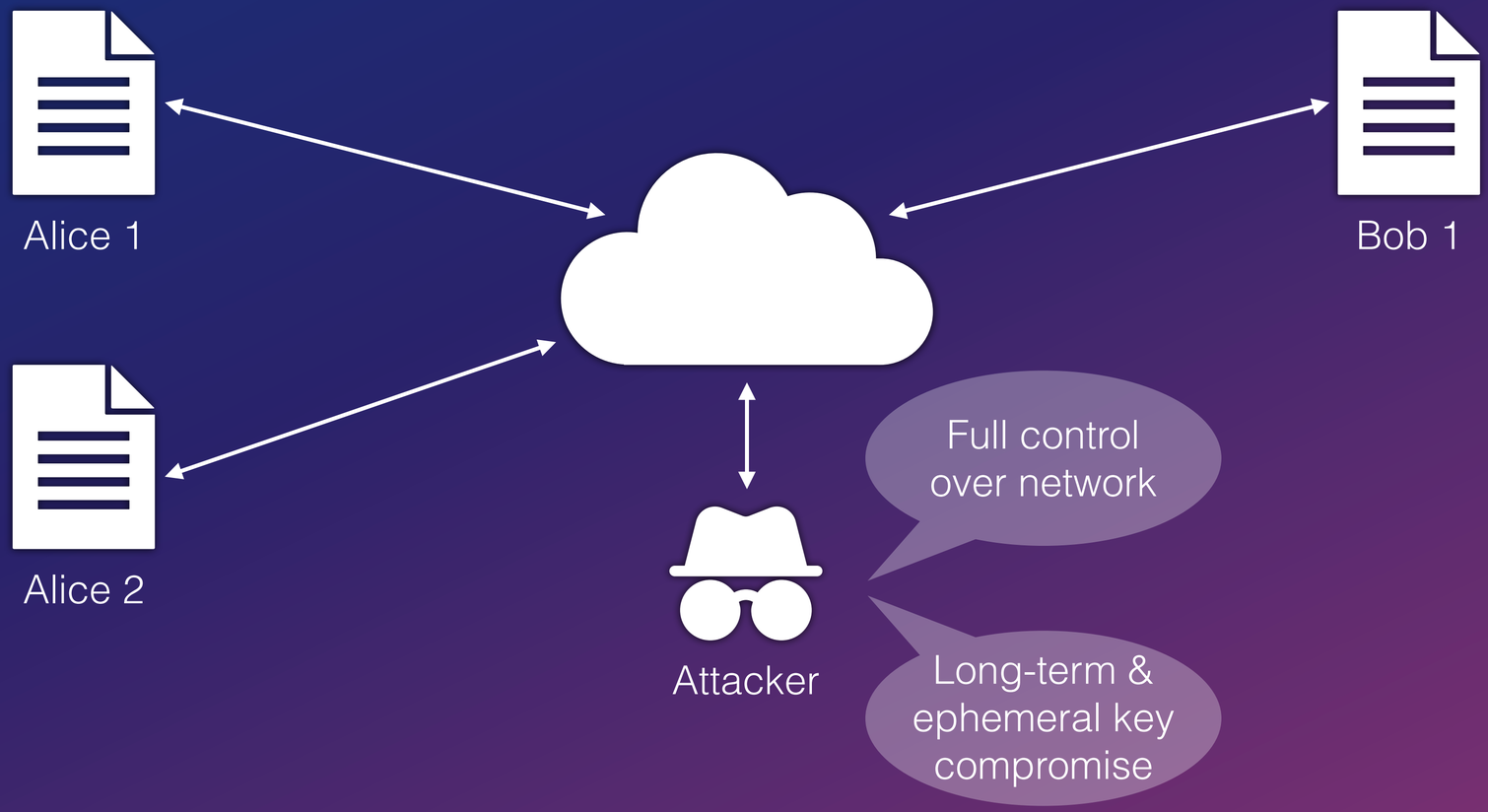


“ `DoneB(Alice,Bob,na,nb)` ⟹ Exactly one `DoneA(Alice,Bob,na,nb)`

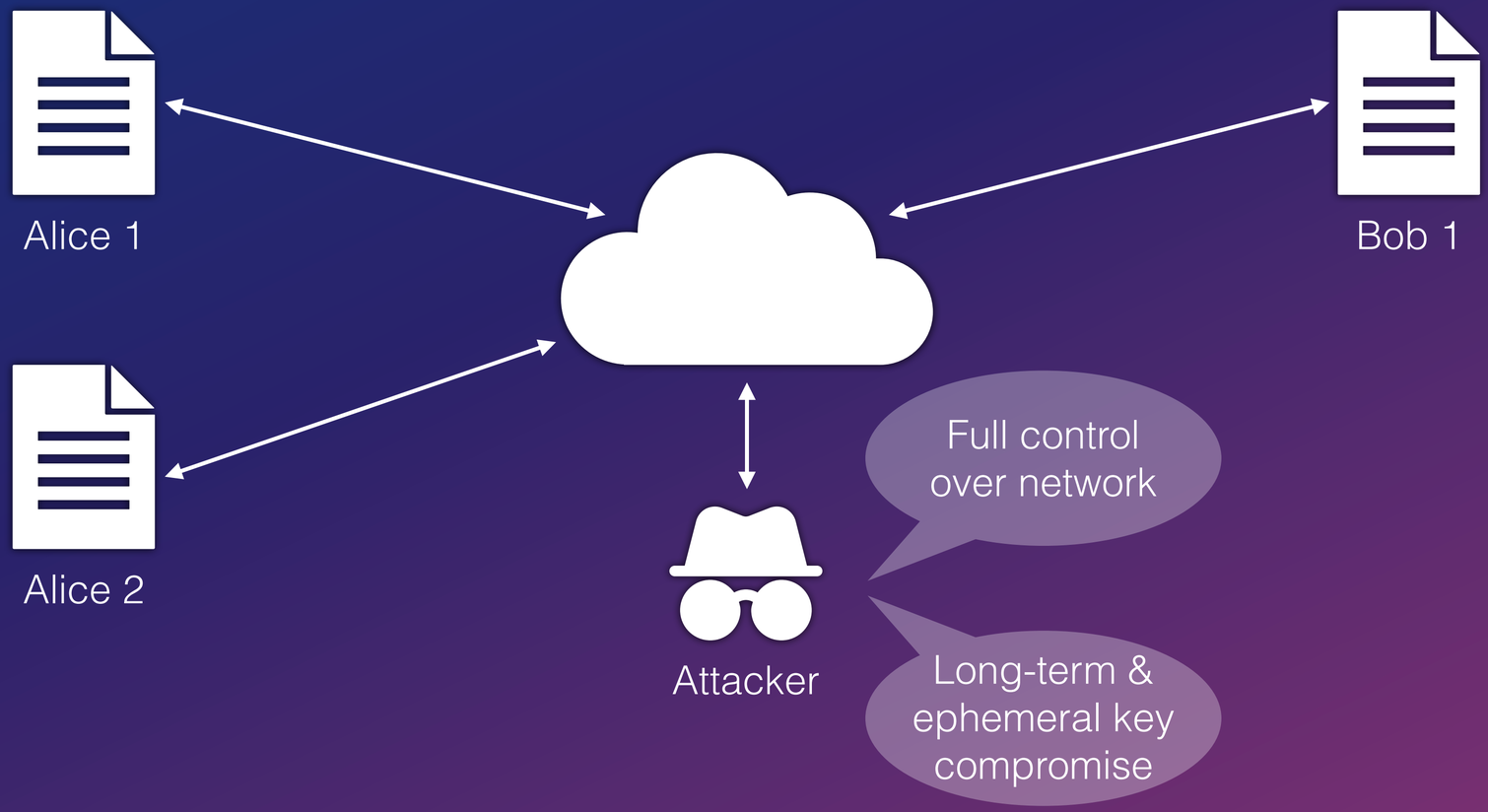
Approach

Approach

Approach


Alice 1
Bob 1
Alice 2
Attacker
Full control over network
Long-term & ephemeral key compromise

Approach

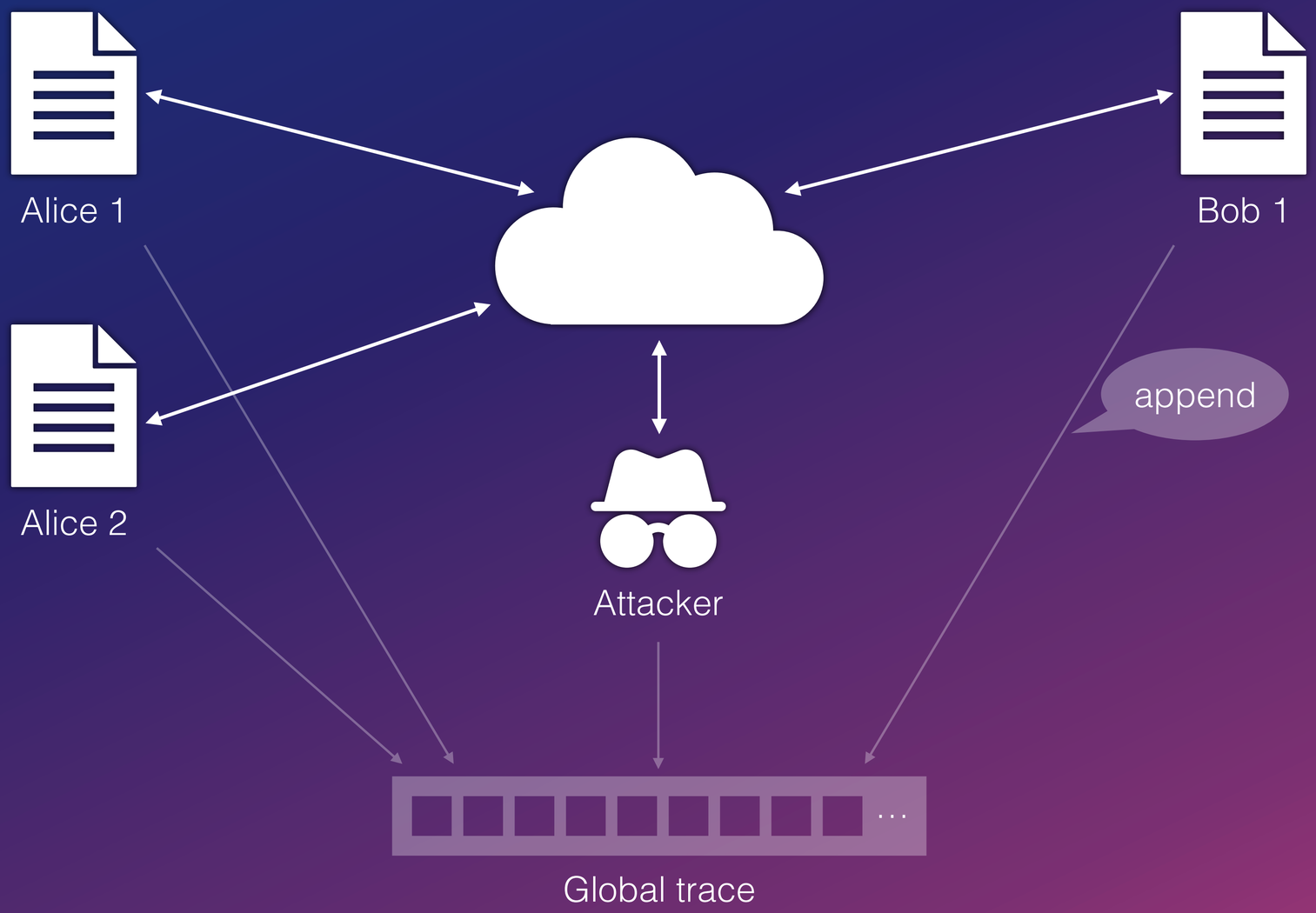


→ Model as a program with threads

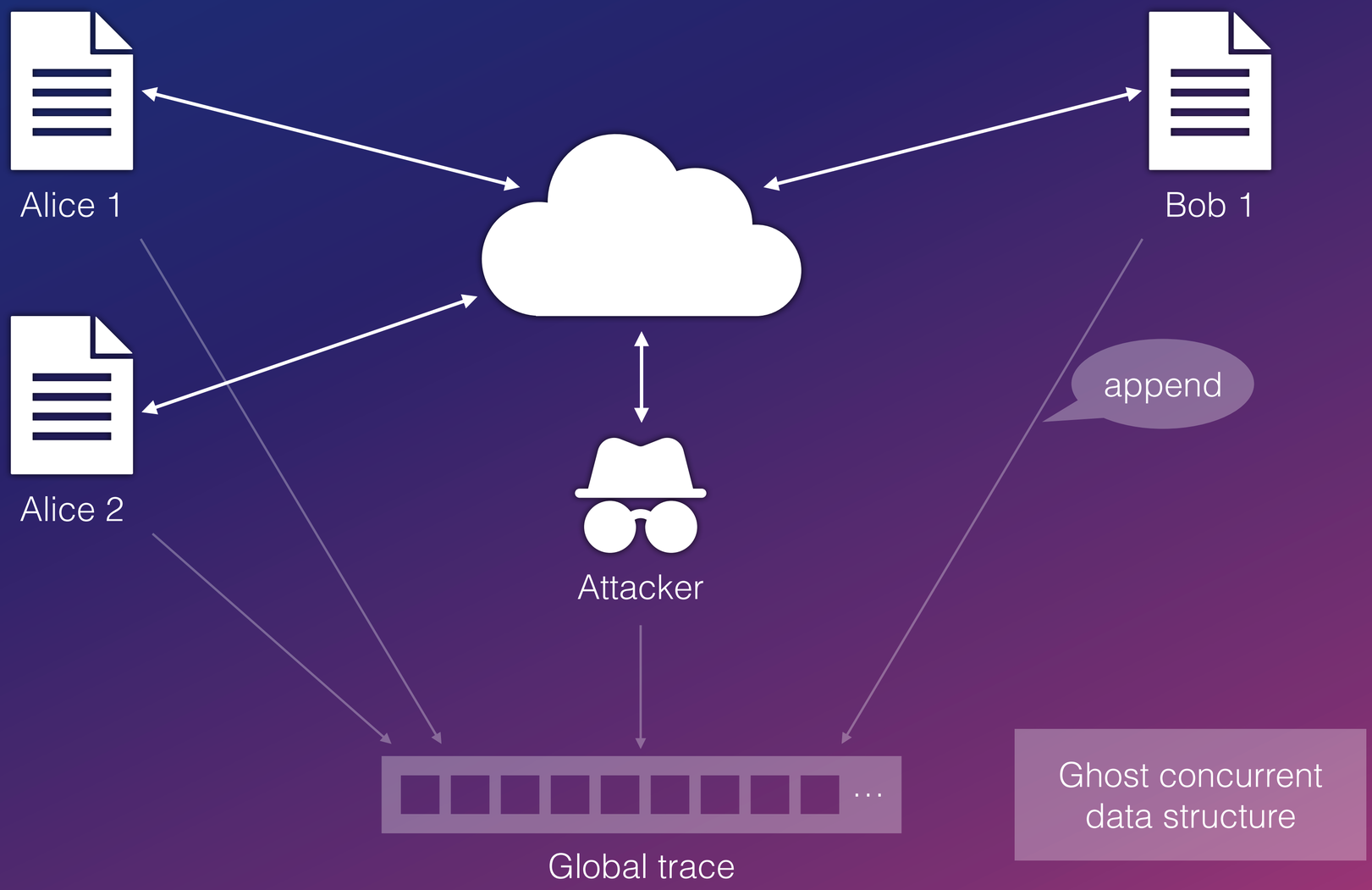
Approach

Global trace

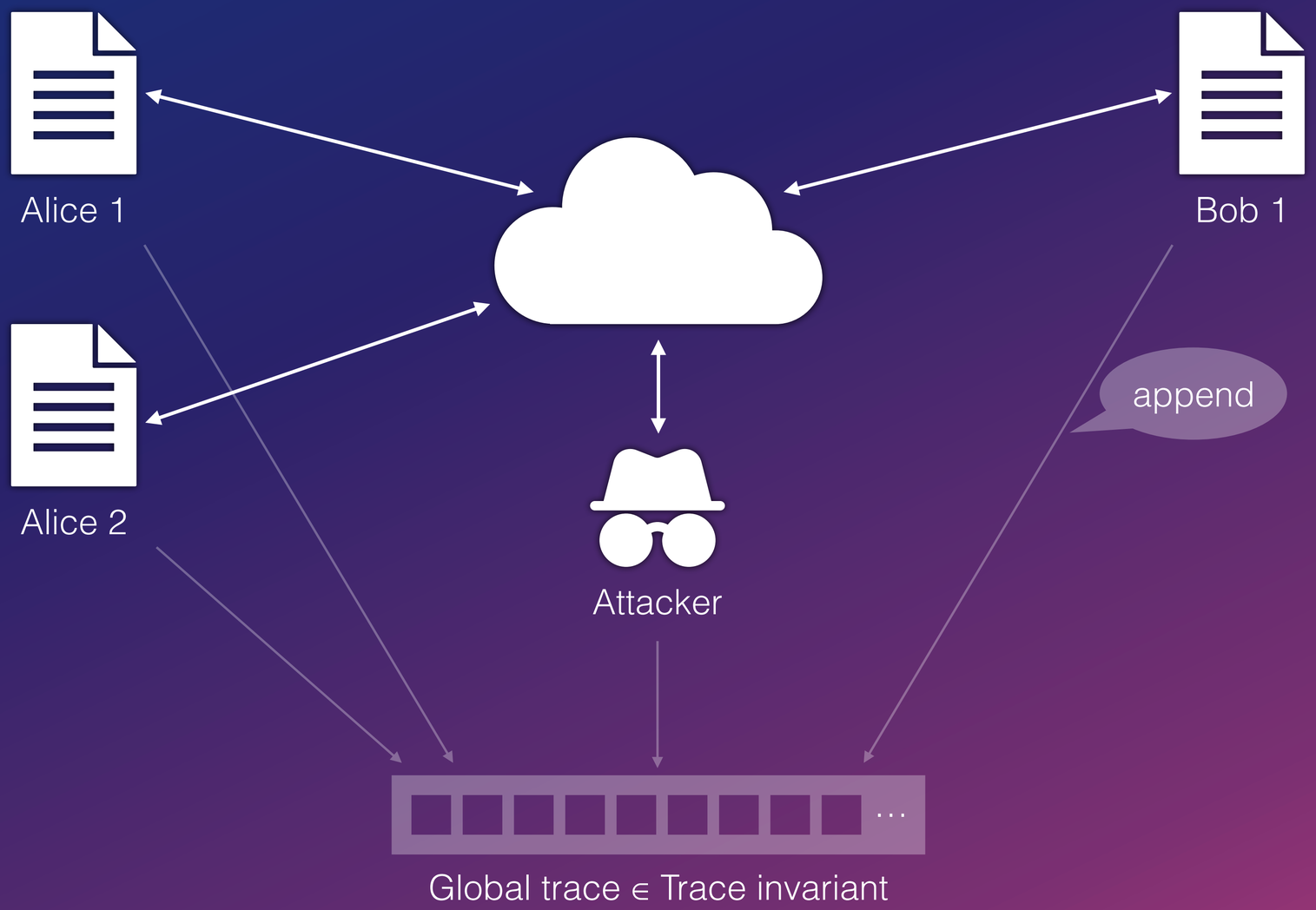
Approach



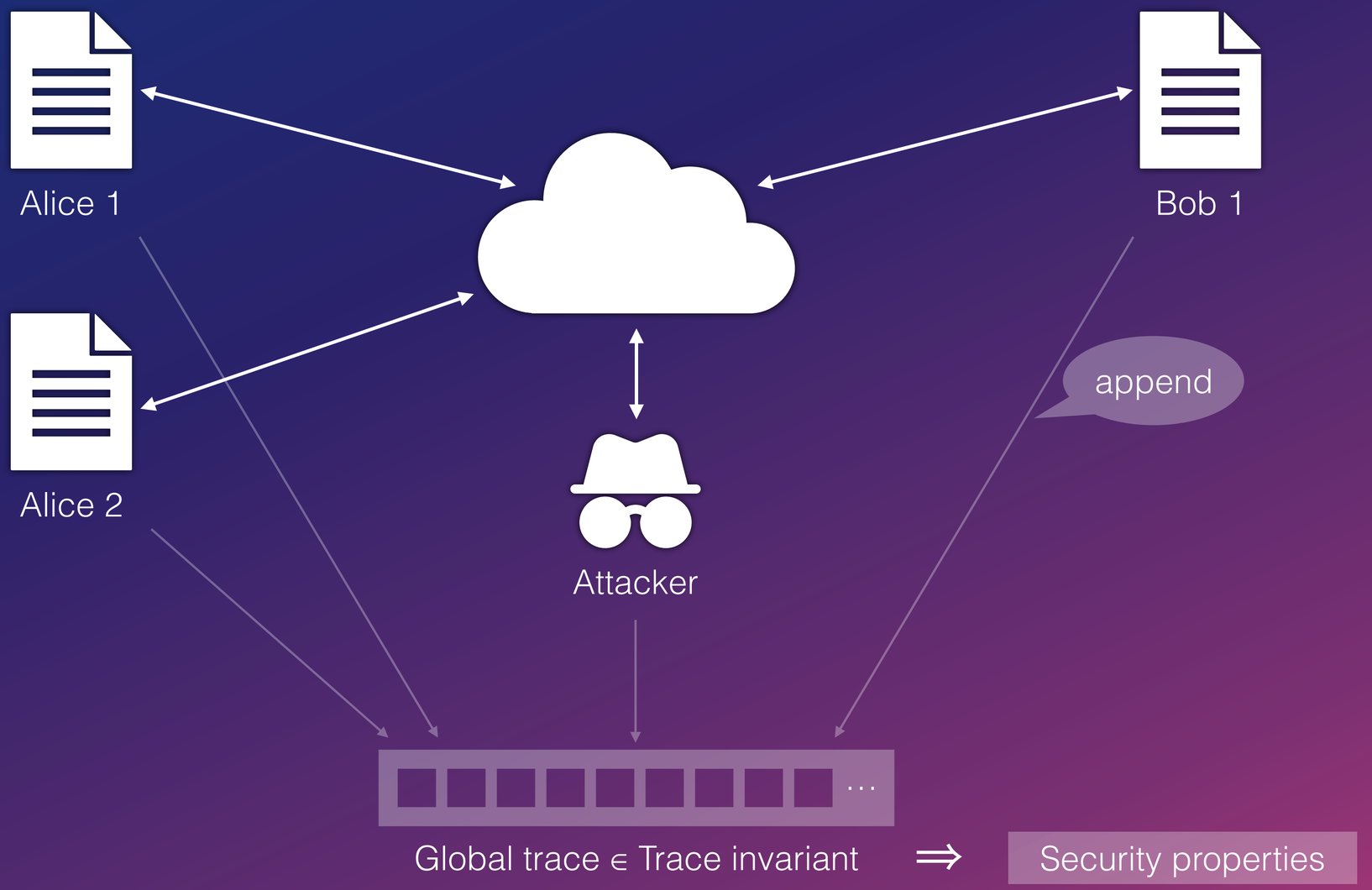
Approach


Alice 1
Alice 2
Bob 1
append
Attacker
…
Global trace
Ghost concurrent data structure

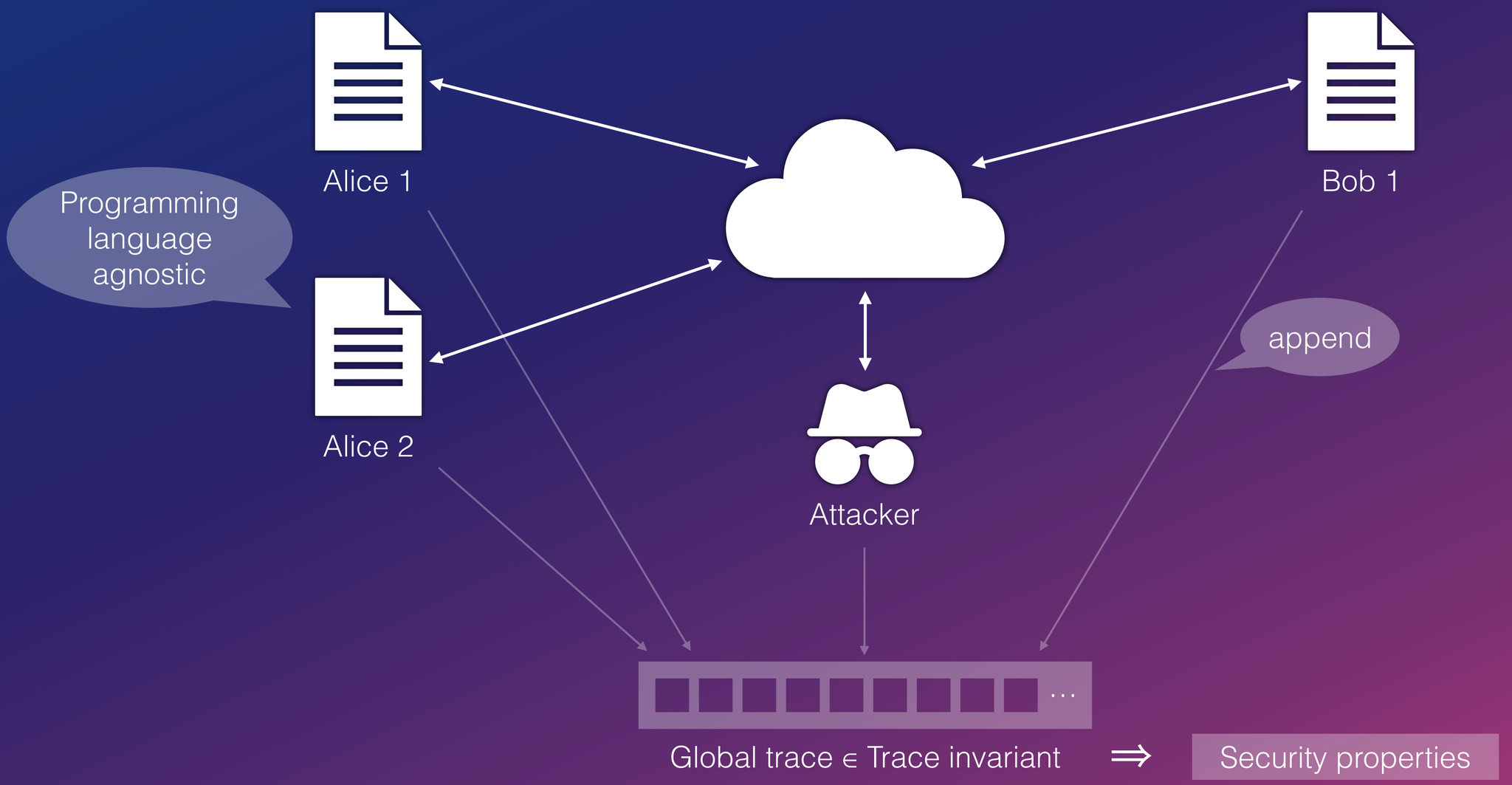
Approach


Alice 1
Alice 2
Bob 1
append
Attacker
…
Global trace ∈ Trace invariant

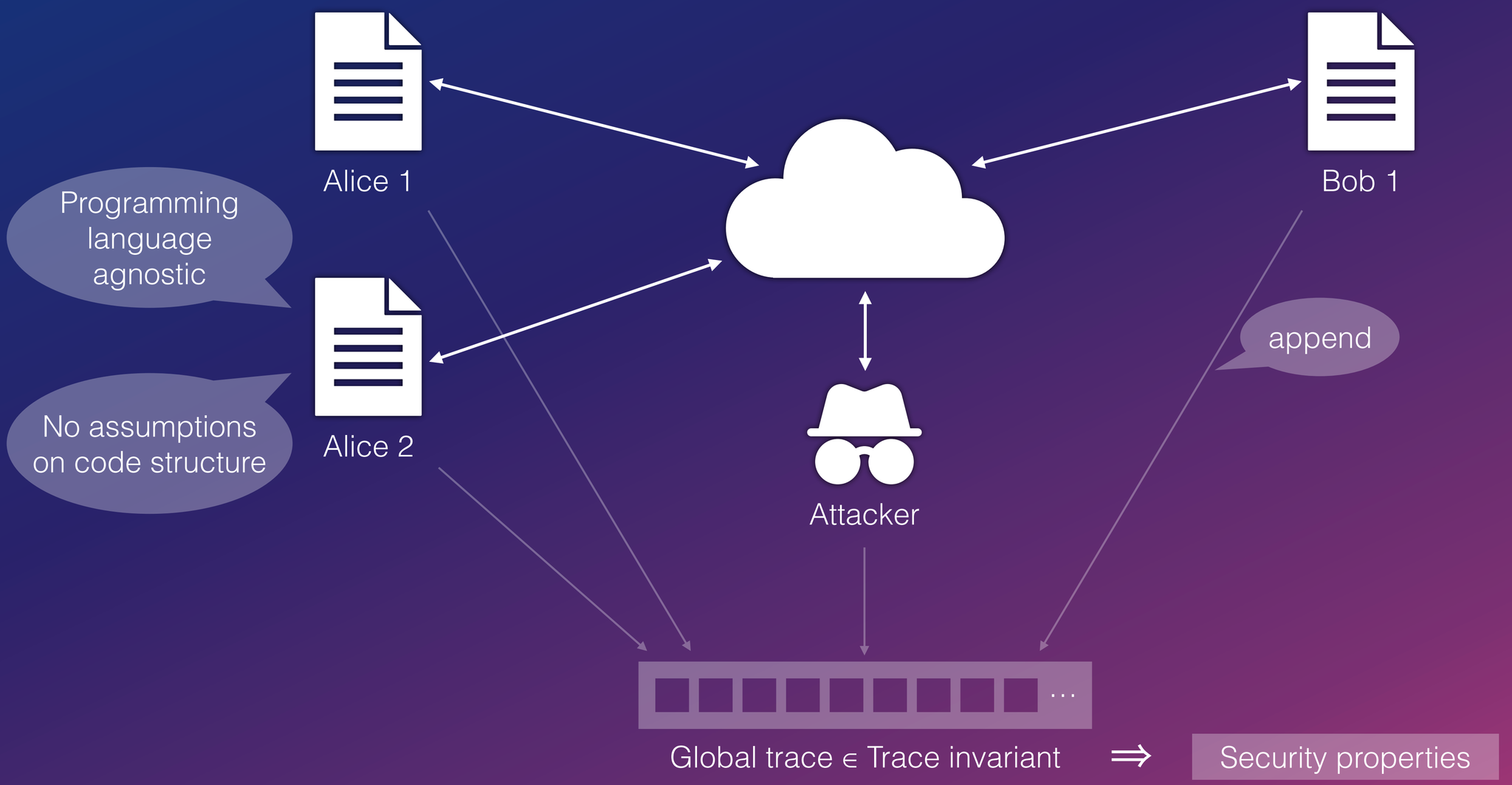
Approach


Alice 1
Alice 2
Bob 1
append
Attacker
…
Global trace ∈ Trace invariant
⇒
Security properties

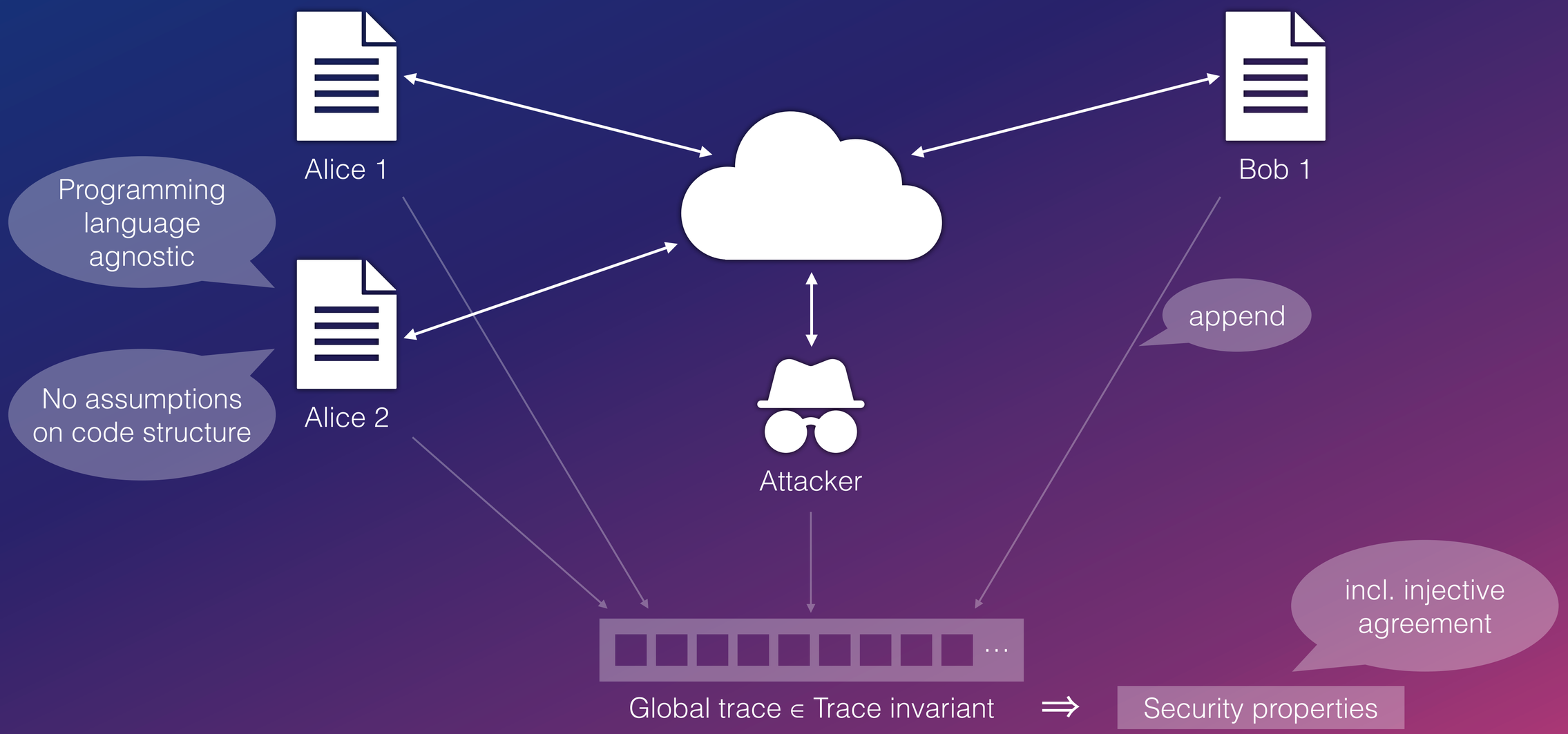
Approach

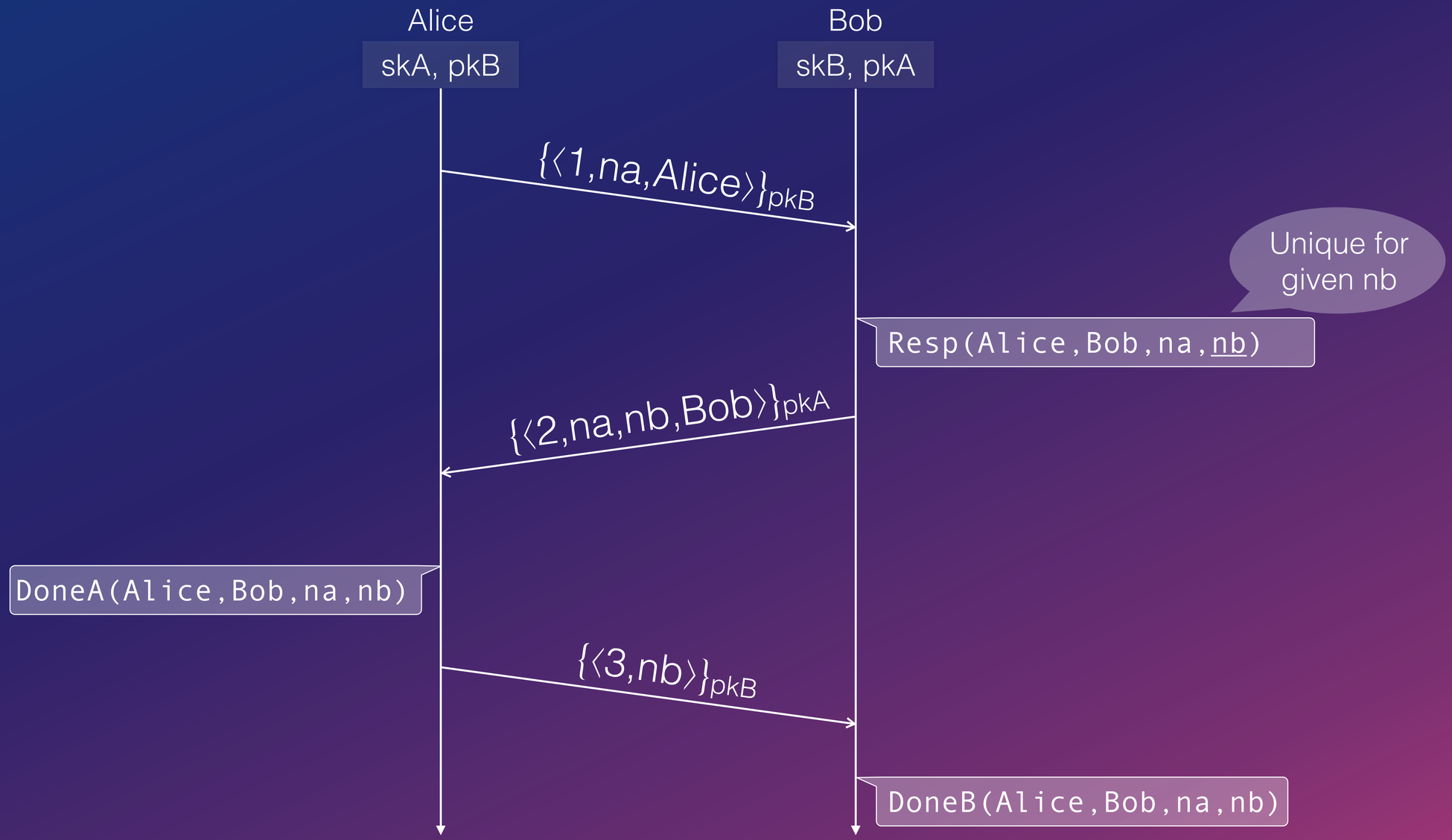

Alice 1
Alice 2
Bob 1
Programming language agnostic
append
Attacker
…
Global trace ∈ Trace invariant
⇒
Security properties

Approach

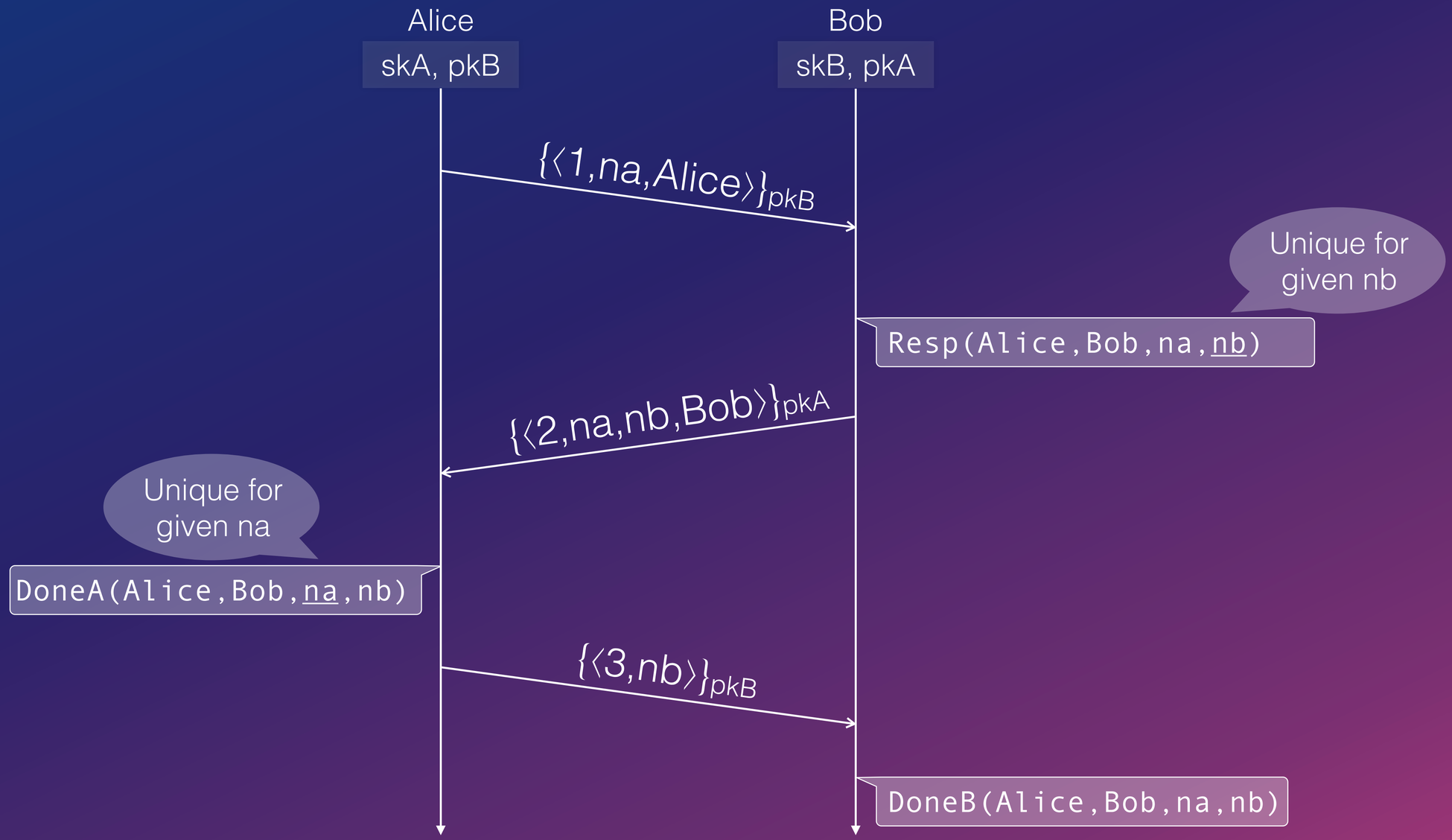

Alice 1
Bob 1
Programming language agnostic
Alice 2
append
No assumptions on code structure
Attacker
…
Global trace ∈ Trace invariant
⇒
Security properties

Approach

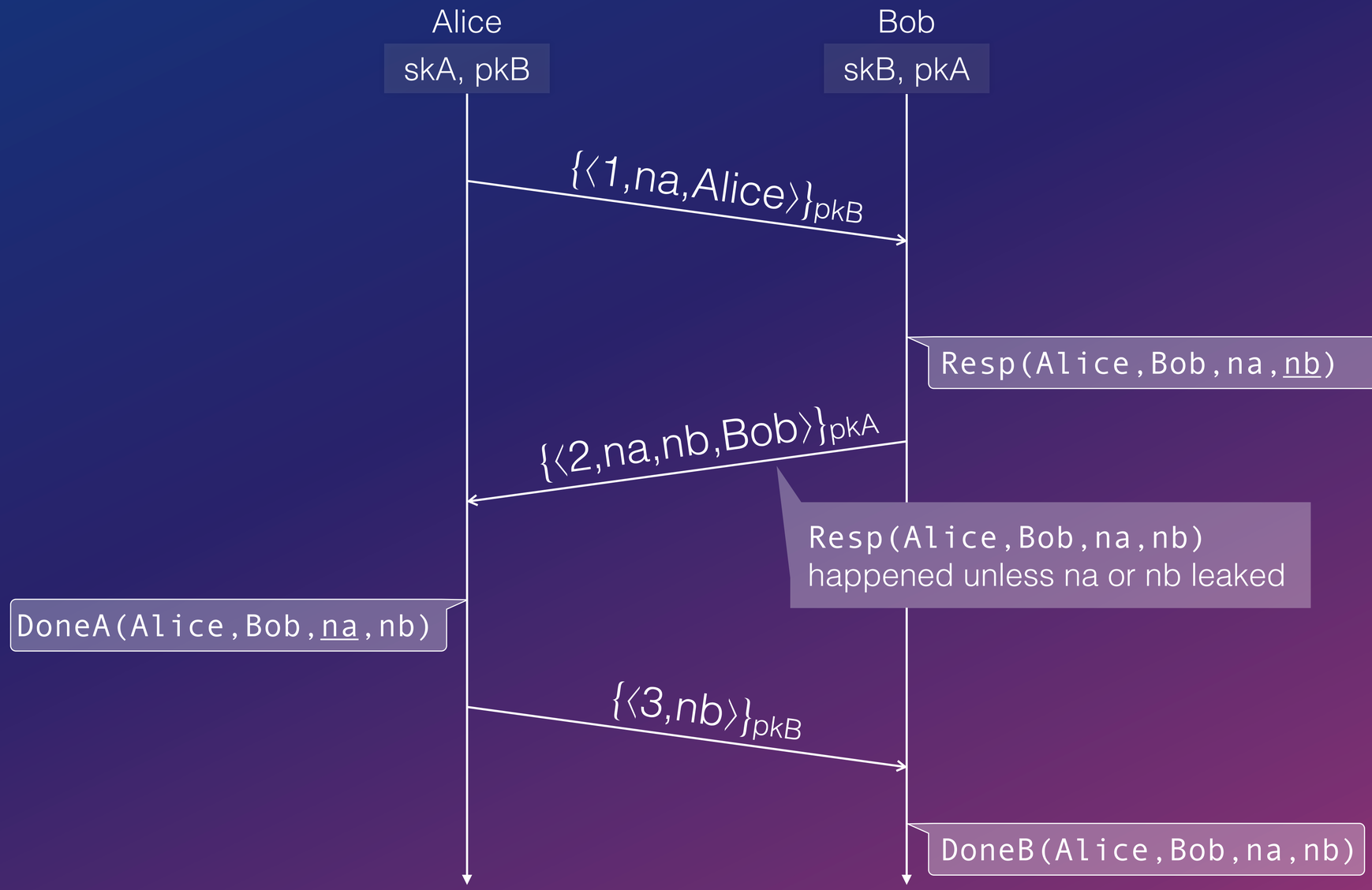

Alice 1
Alice 2
Bob 1
Attacker
Programming language agnostic
No assumptions on code structure
append
…
Global trace ∈ Trace invariant
⇒
Security properties
incl. injective agreement



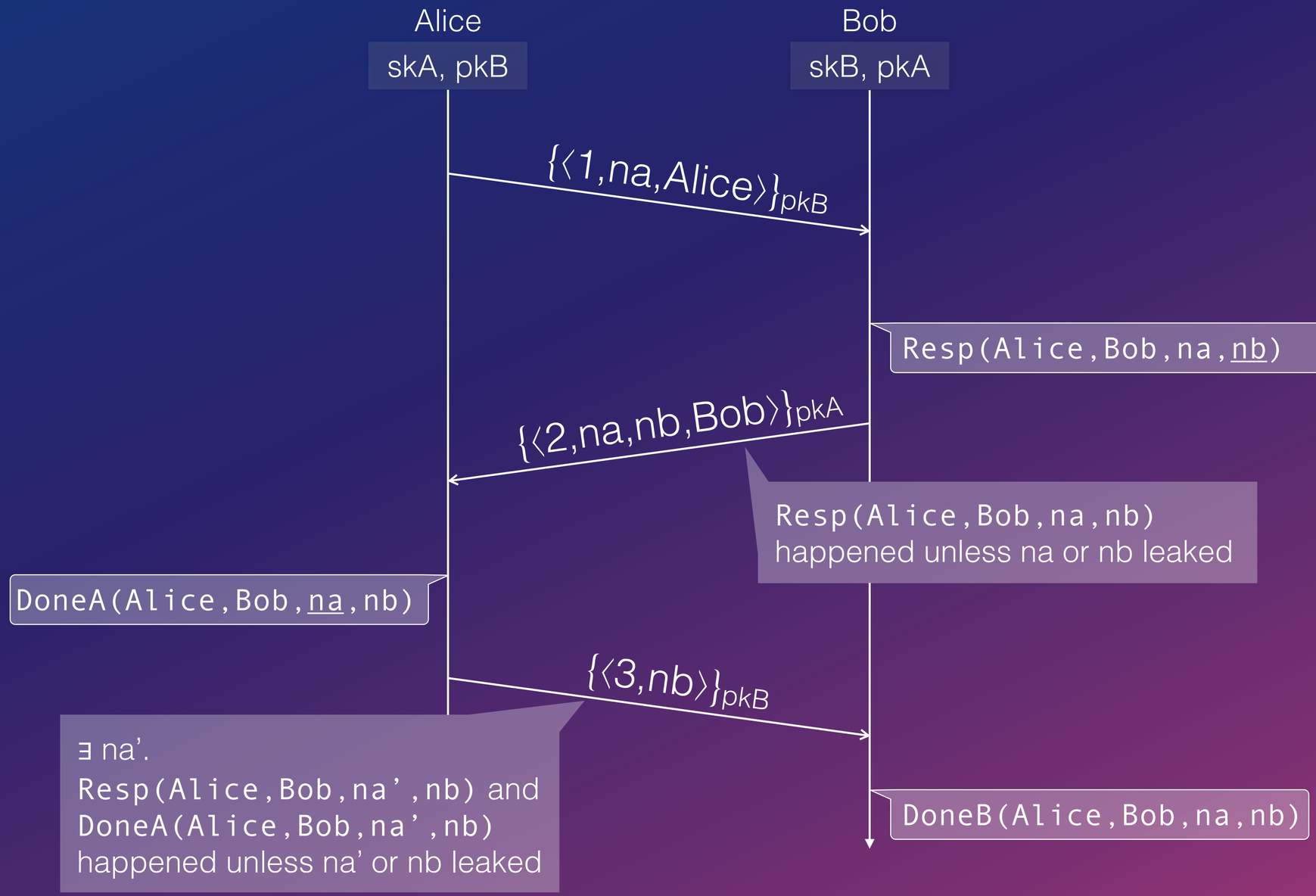
“ DoneB(Alice,Bob,na,nb) ⟹ Exactly one DoneA(Alice,Bob,na,nb)



“ DoneB(Alice,Bob,na,nb) ⟹ Exactly one DoneA(Alice,Bob,na,nb)



“ DoneB(Alice,Bob,na,nb) ⟹ Exactly one DoneA(Alice,Bob,na,nb)



“ DoneB(Alice,Bob,na,nb) ⟹ Exactly one DoneA(Alice,Bob,na,nb)

```
func main(pkB []byte) {
  n := random()
  msg := enc(n, pkB)
  send(msg)
}
```

Alice

```
func send(msg) {




  io.send(msg)
}
```

Library

Global trace ∈ Trace invariant

+

I/O

```
func main(pkB []byte) {
  n := random()
  msg := enc(n, pkB)
  send(msg)
}
```

Alice

Global trace ∈ Trace invariant

+

I/O

```
func send(msg) {
  //@ trace.lock()
  //@ trace.append(Send(msg))

  //@ trace.unlock()

  io.send(msg)
}
```

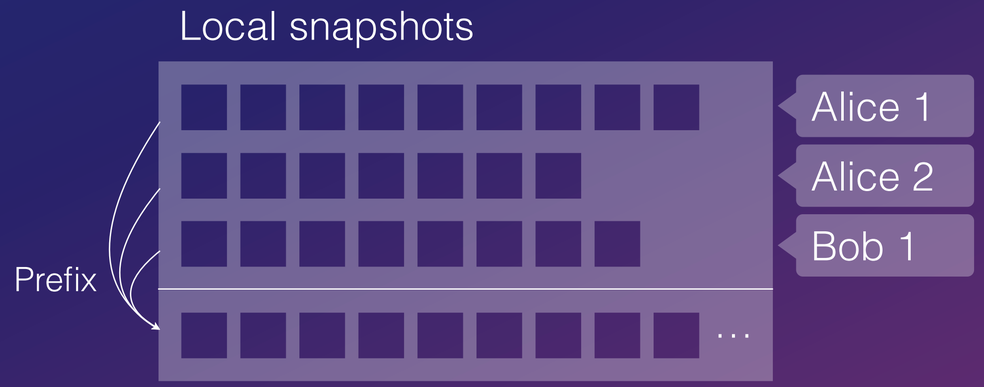
Library

```
func main(pkB []byte) {
  n := random()
  msg := enc(n, pkB)
  send(msg)
}
```

Alice

```
func send(msg) {
  //@ trace.lock()
  //@ trace.append(Send(msg))

  //@ trace.unlock()

  io.send(msg)
}
```

Library



Global trace ∈ Trace invariant

+

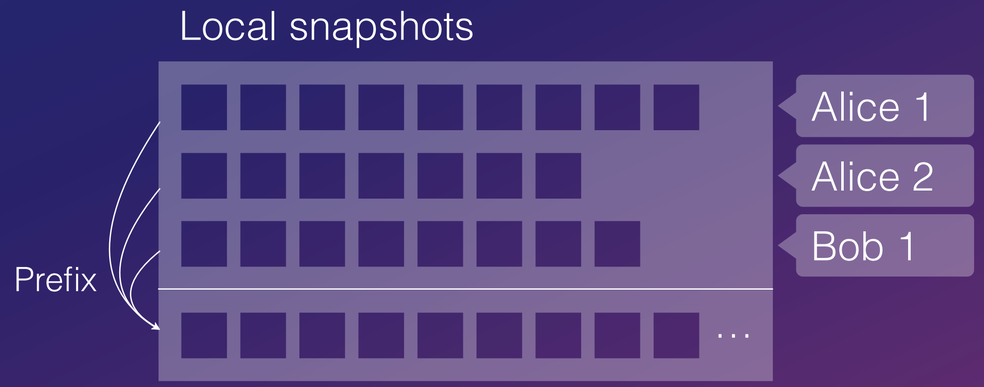
I/O

```
func main(pkB []byte) {
  n := random()
  msg := enc(n, pkB)
  send(msg)
}
```

Alice

```
func send(msg) {
  //@ trace.lock()
  //@ trace.append(Send(msg))
  //@ setSnap(trace)
  //@ trace.unlock()

  io.send(msg)
}
```

Library



+

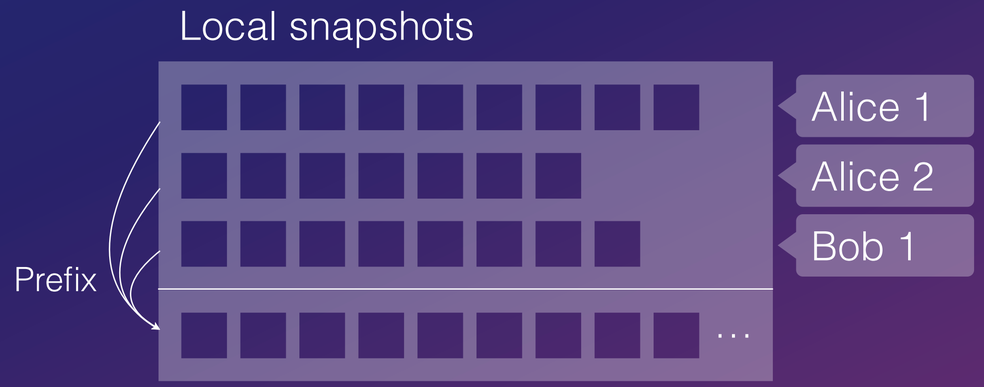
I/O

```
func main(pkB []byte) {
  n := random()
  msg := enc(n, pkB)
  send(msg)
}
```

Alice

```
//@ req msgInv(getSnap(), msg)
func send(msg) {
  //@ trace.lock()
  //@ trace.append(Send(msg))
  //@ setSnap(trace)
  //@ trace.unlock()

  io.send(msg)
}
```

Library



+

I/O

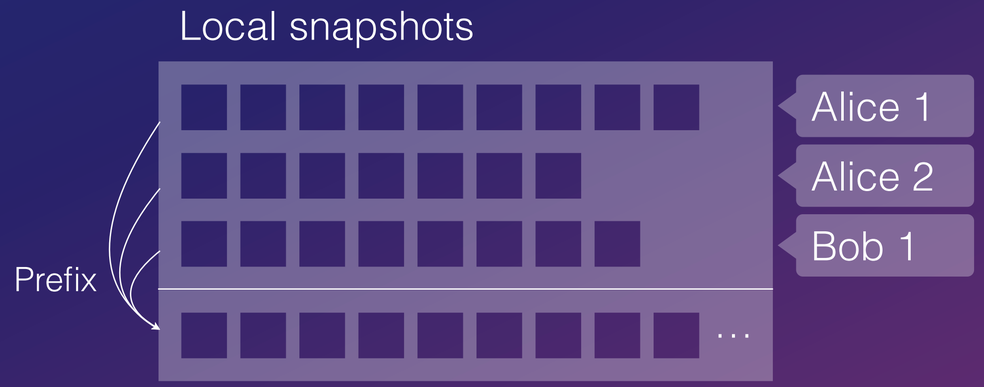
```
func main(pkB []byte) {
  n := random()
  msg := enc(n, pkB)
  send(msg)
}
```

Alice

Parameterized

```
//@ req msgInv(getSnap(), msg)
func send(msg) {
  //@ trace.lock()
  //@ trace.append(Send(msg))
  //@ setSnap(trace)
  //@ trace.unlock()

  io.send(msg)
}
```

Library



Global trace ∈ Trace invariant

+

I/O

Evaluation

Verification libraries for C & Go

Evaluation

Verification libraries for C & Go

Library	LOC	LOS	[s]
Go / Gobra	83	6,932	126.1
C / VeriFast	343	3,837	0.8

Evaluation

Verification libraries for C & Go

Needham-Schroeder-Lowe
(in C & Go) and
signed Diffie-Hellman
key exchange (in Go)

	gobra
Tamarin LoC	—
Go LoC	~600
Proof annotations	~5.8k
Generated I/O spec	—
Verification time [min]	~4.5

WireGuard®
FAST, MODERN, SECURE VPN TUNNEL

	 + gobra	gobra
Tamarin LoC	~350 + ~40	—
Go LoC	~600	~600
Proof annotations	~2.7k	~5.8k
Generated I/O spec	~1.2k	—
Verification time [min]	~3 + ~4.8	~4.5

Evaluation

Verification libraries for C & Go

WireGuard VPN protocol

Needham-Schroeder-Lowe
(in C & Go) and
signed Diffie-Hellman
key exchange (in Go)

Injective agreement and
forward secrecy
proven for WireGuard

Evaluation

Verification libraries for C & Go

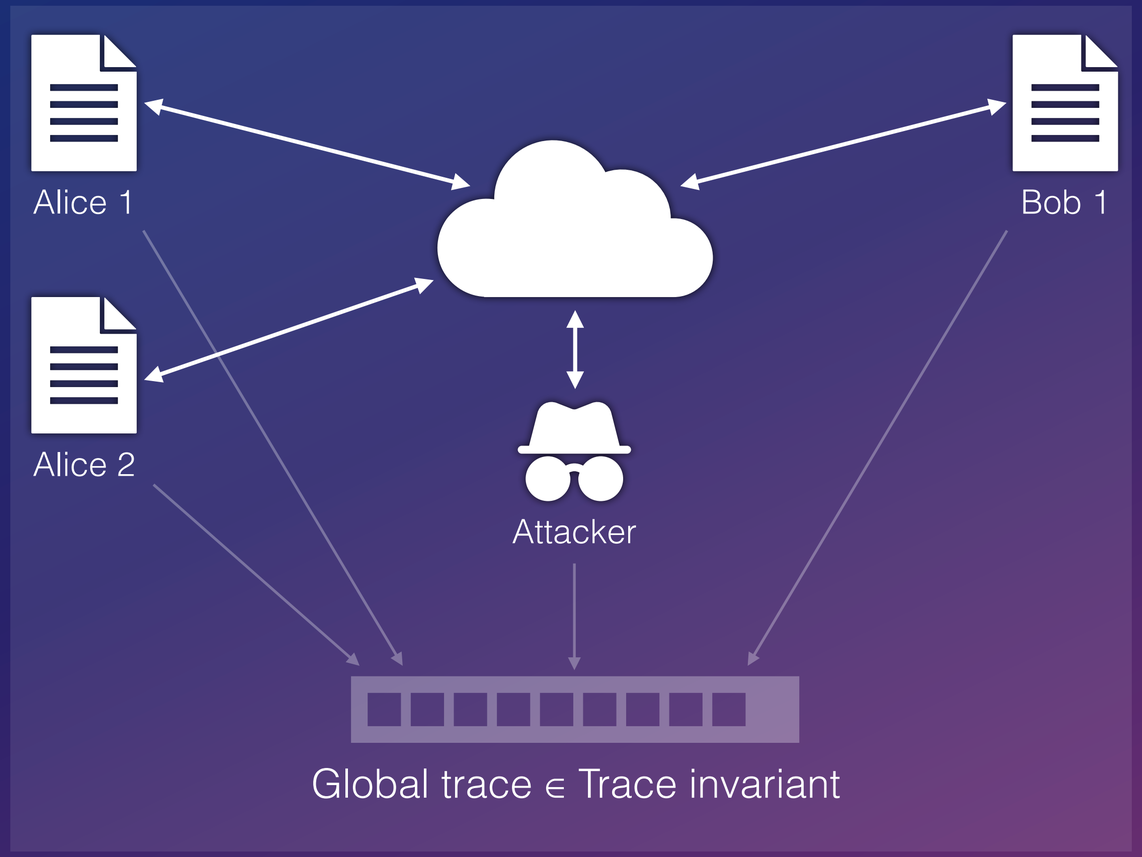
WireGuard VPN protocol

Needham-Schroeder-Lowe
(in C & Go) and
signed Diffie-Hellman
key exchange (in Go)

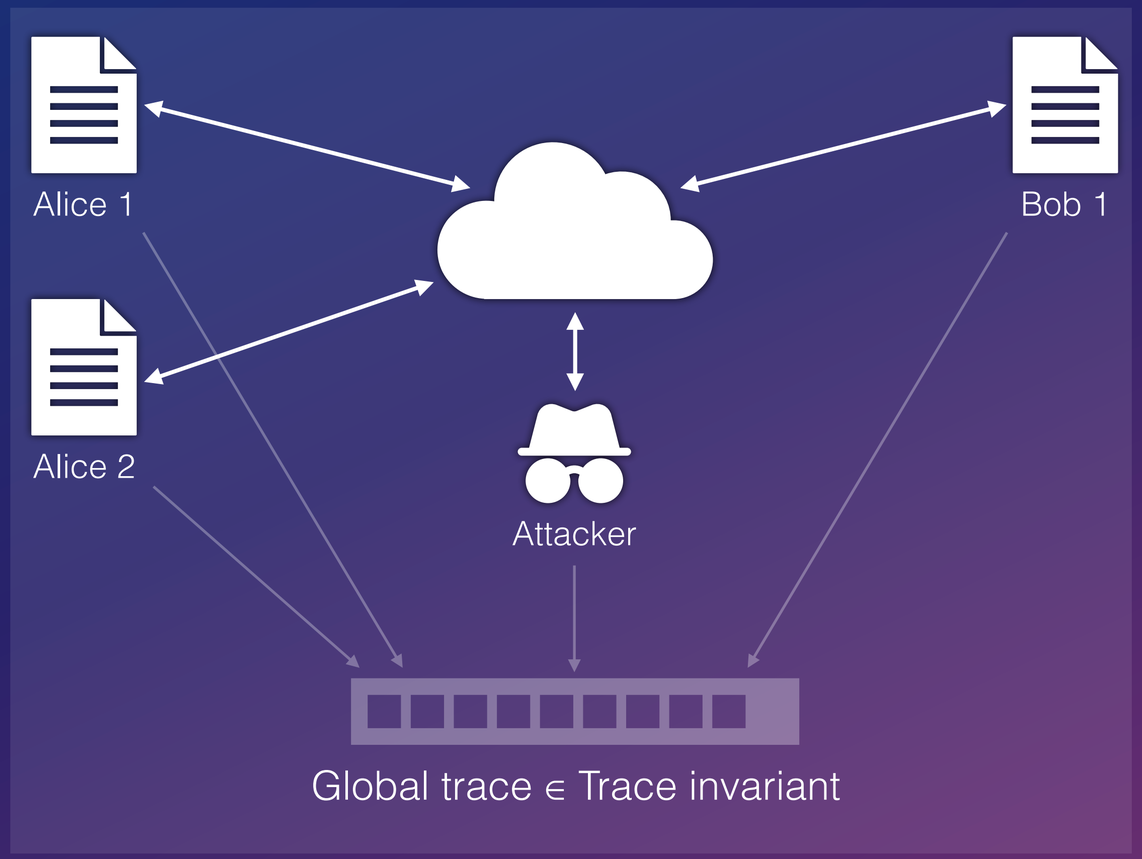
Injective agreement and
forward secrecy
proven for WireGuard

GitHub viperproject/SecurityProtocolImplementations

Conclusions

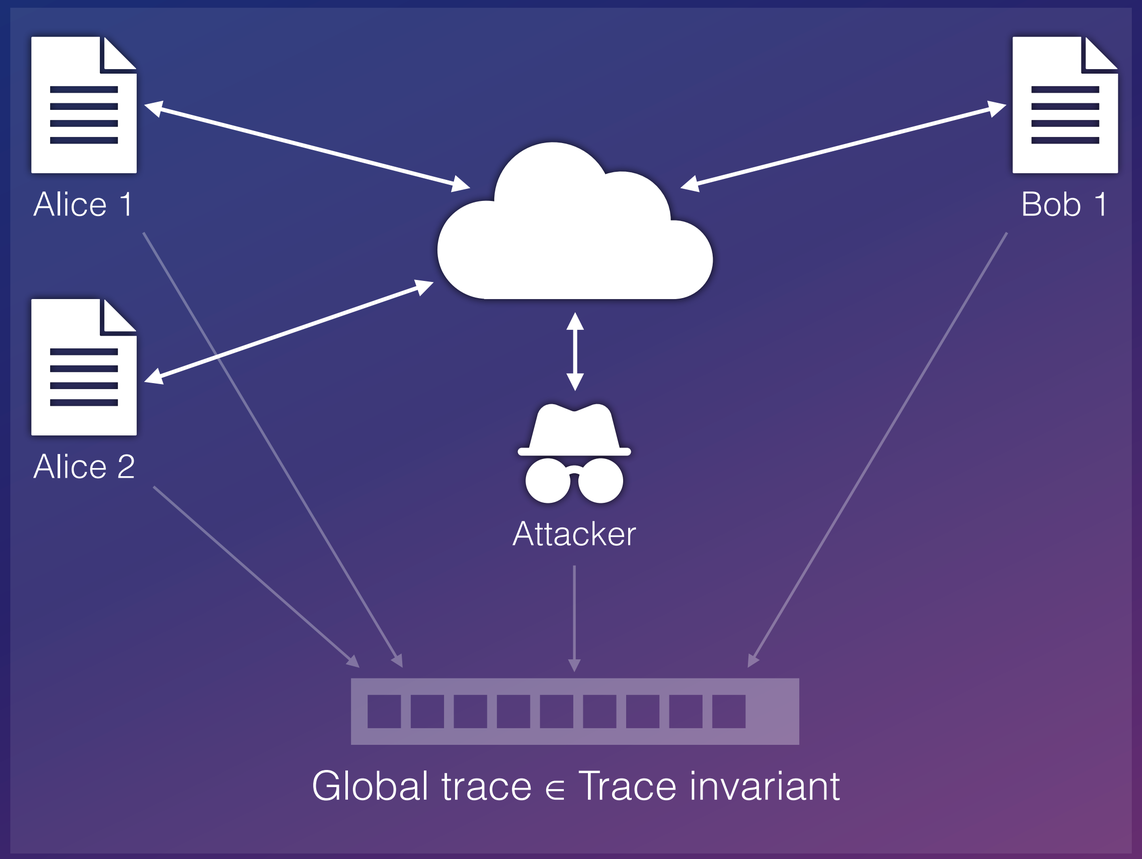


Conclusions



WireGuard

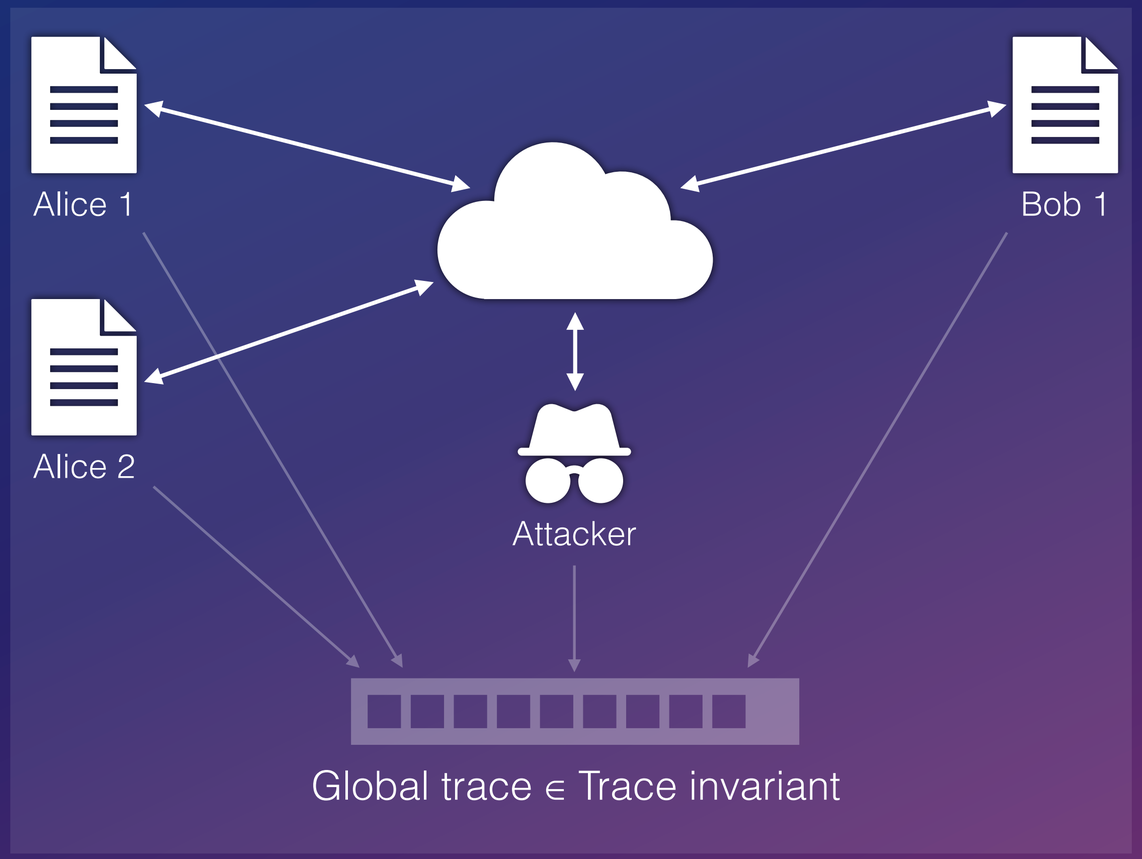
Conclusions



WireGuard

CCS ‘23: A Generic Methodology for the Modular Verification of Security Protocol Implementations

Conclusions



WireGuard

CCS ‘23: A Generic Methodology for the Modular Verification of Security Protocol Implementations

Soundness Proof

Overview

Implementation

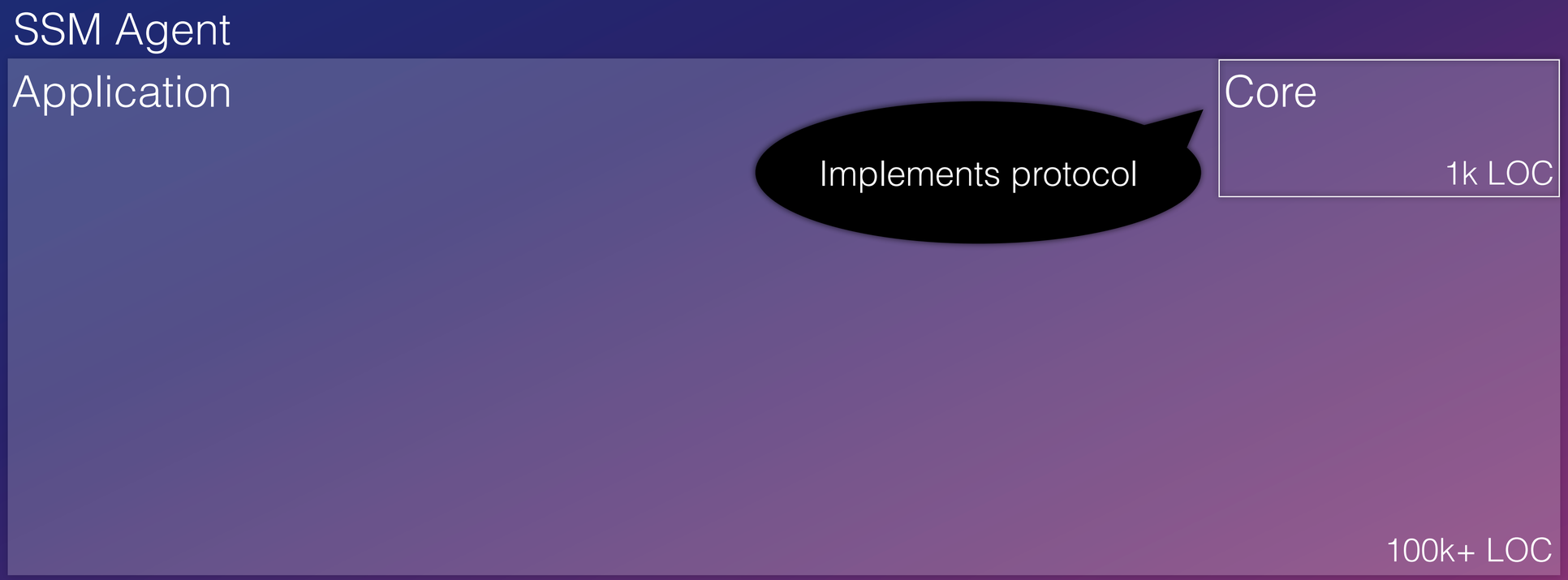
Part 3: Outlook — Scale to large codebases and go beyond security properties

Problem

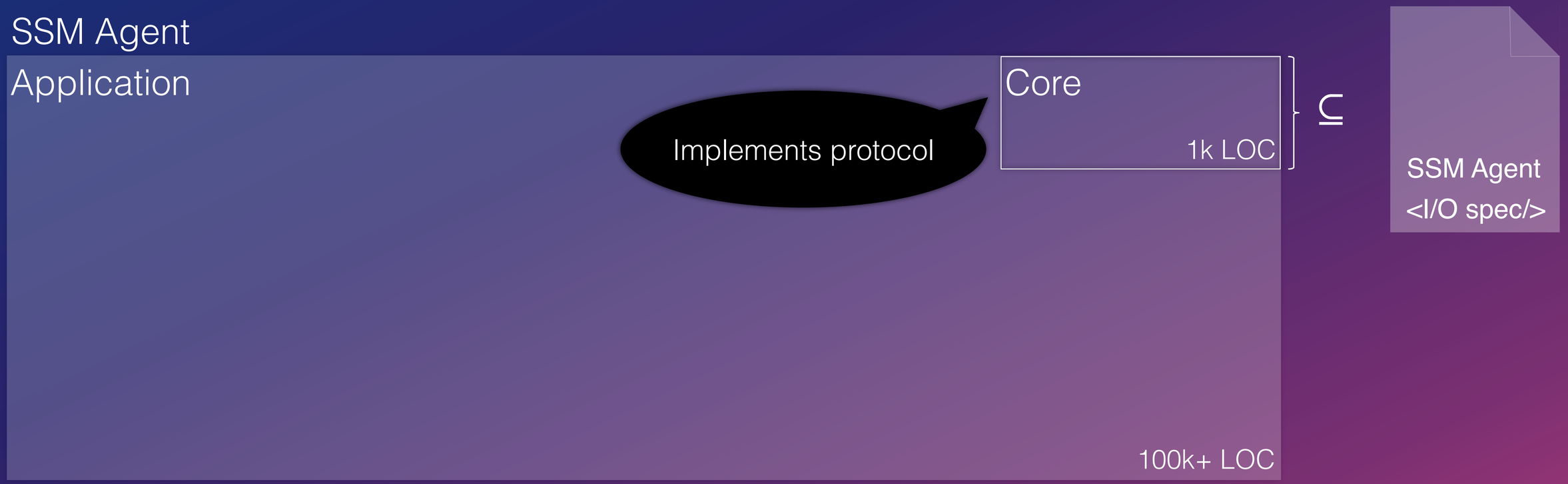
AWS Systems Manager Agent (SSM Agent)

100k+ LOC

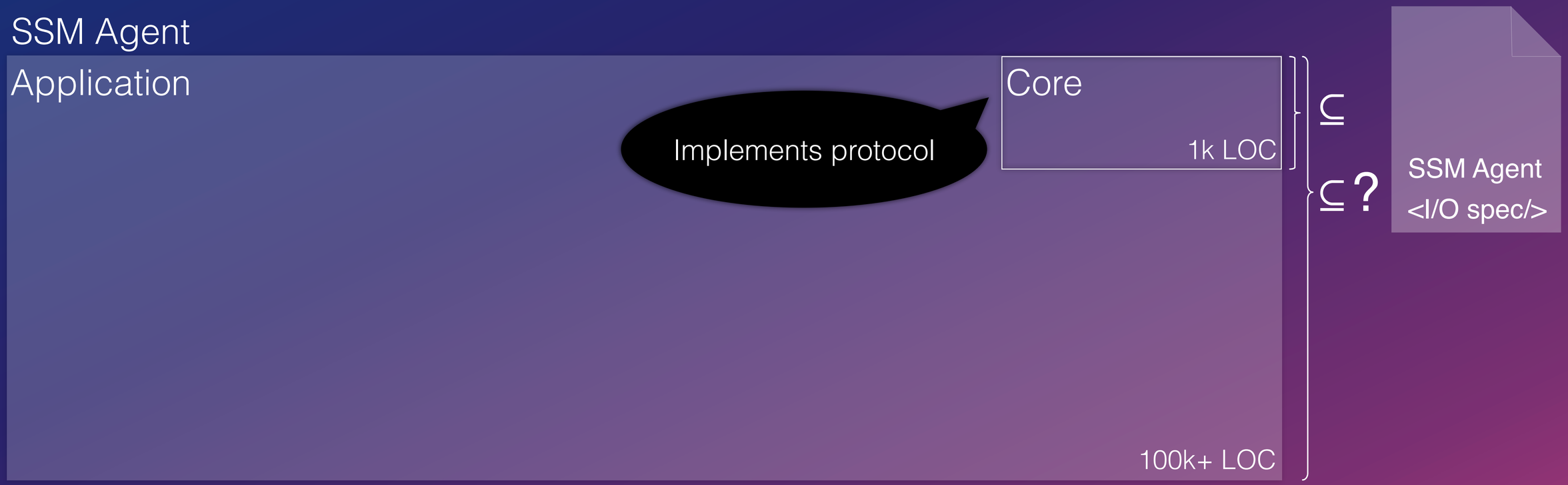
Problem


SSM Agent
Application
Core
1k LOC
Implements protocol
100k+ LOC

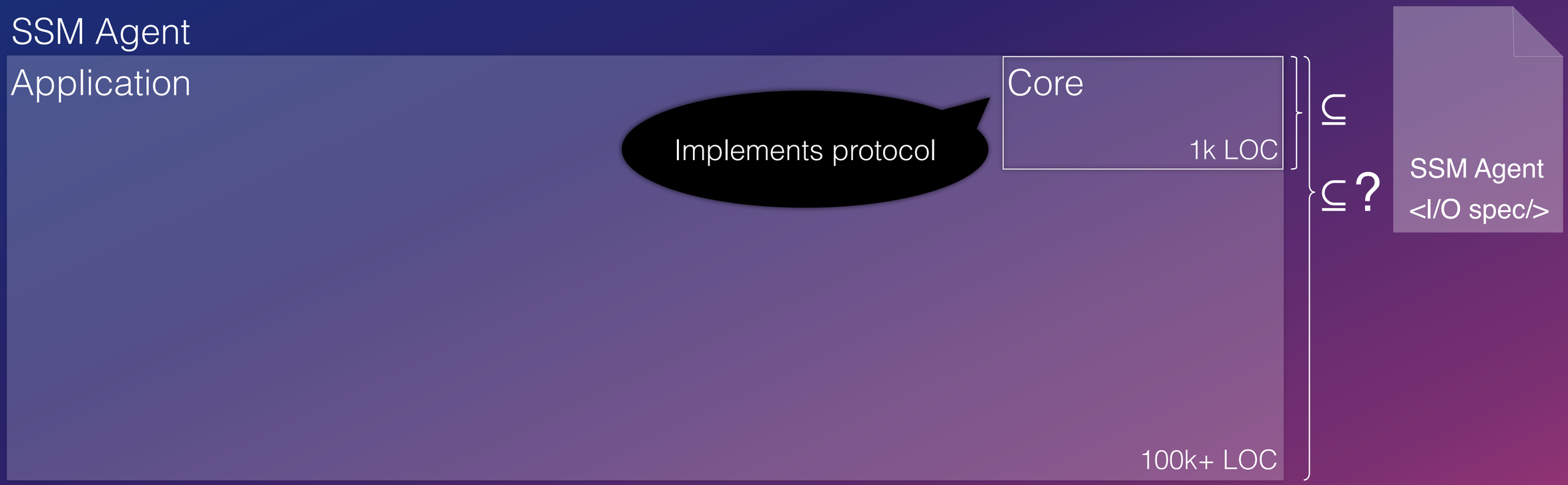
Problem


SSM Agent
Application
Implements protocol
Core
1k LOC
100k+ LOC
⊆
SSM Agent
<I/O spec/>

Problem


SSM Agent
Application
Implements protocol
Core
1k LOC
100k+ LOC
⊆
⊆ ?
SSM Agent
<I/O spec/>

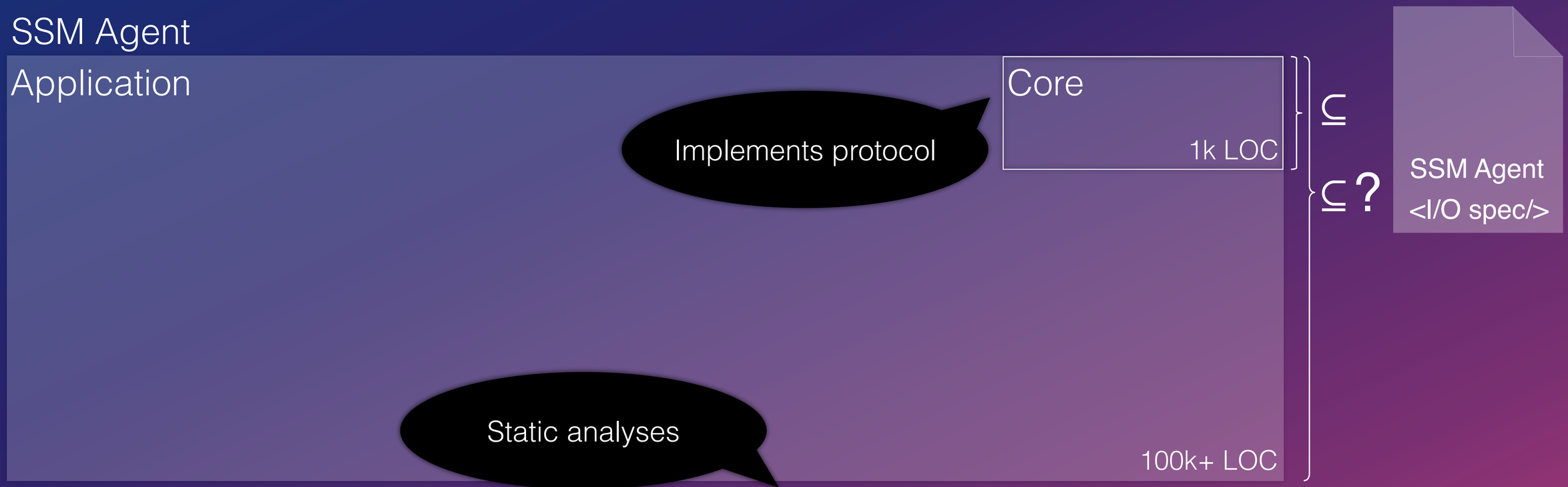
Problem



What I/O operations can we safely allow in the application?

How do we ensure the application respects the core's preconditions?

Problem

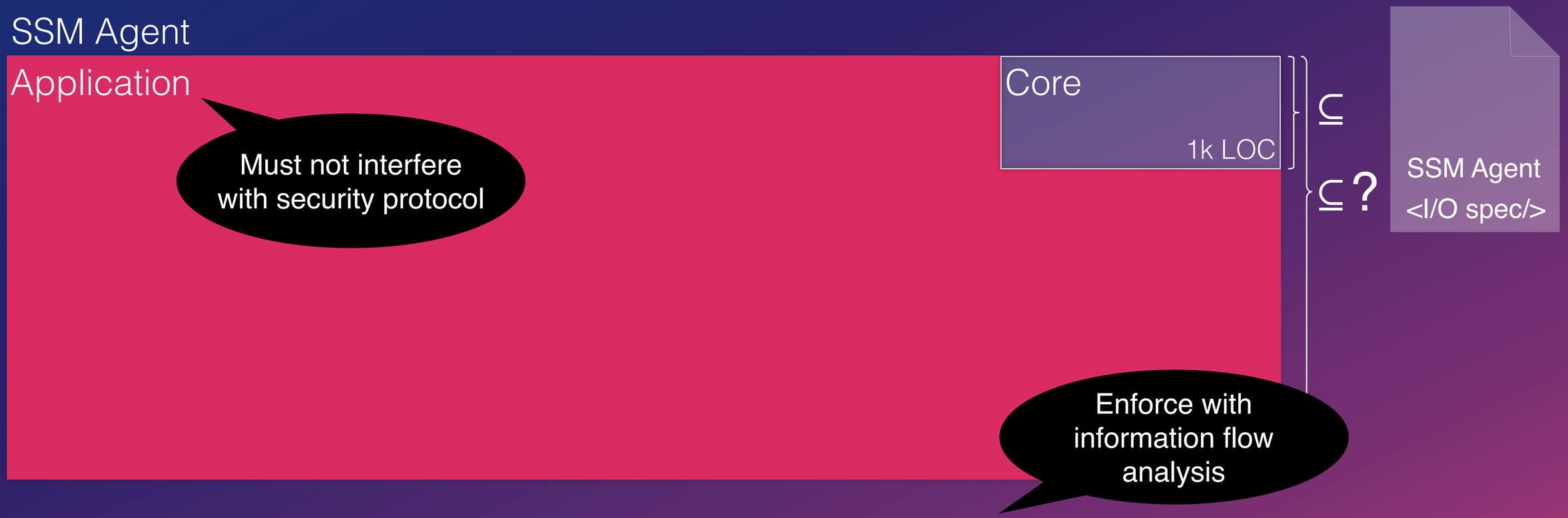


What I/O operations can we safely allow in the application?

How do we ensure the application respects the core's preconditions?

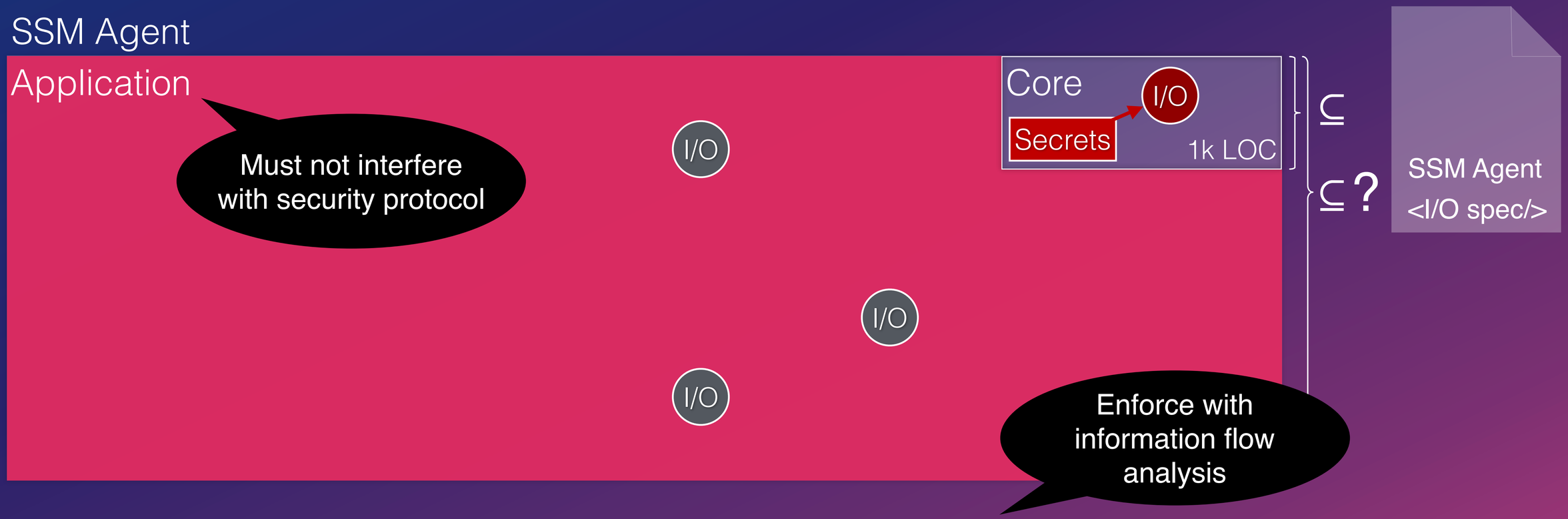
1. I/O Independence

1. I/O Independence



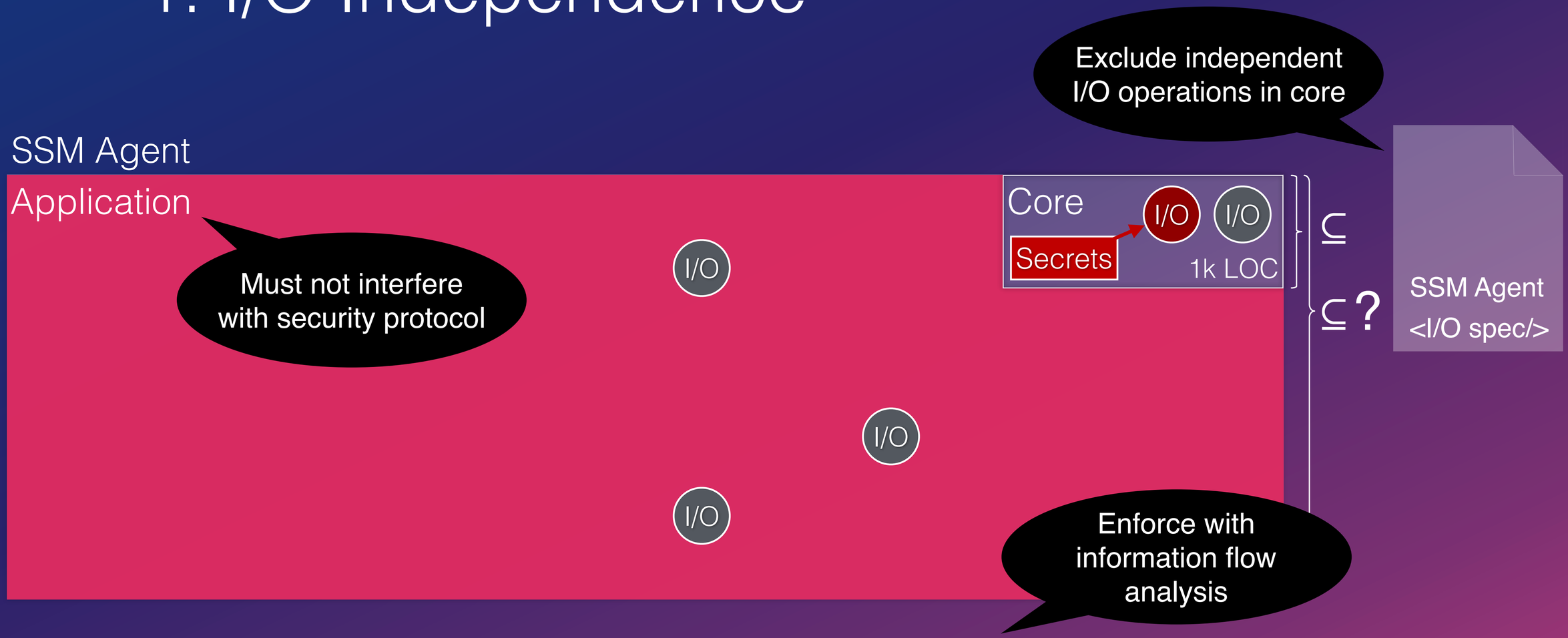
➔ I/O Operations in application must be independent of core secrets

1. I/O Independence



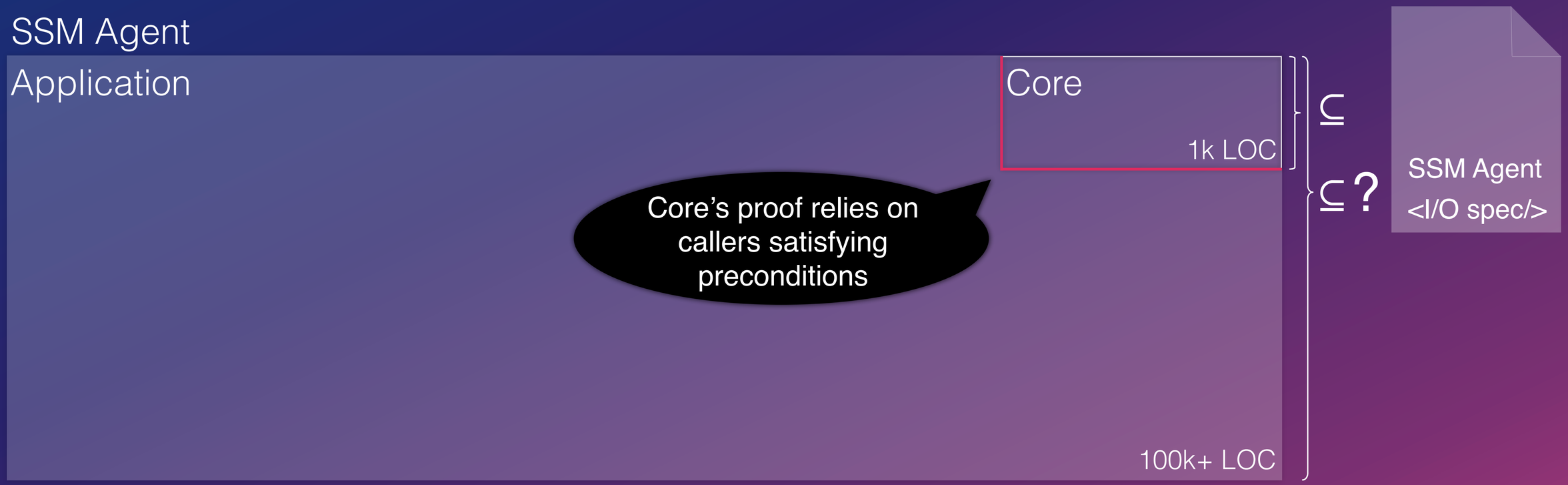
➔ I/O Operations in application must be independent of core secrets

1. I/O Independence



➔ I/O Operations in application must be independent of core secrets

2. Discharging Core's Assumptions



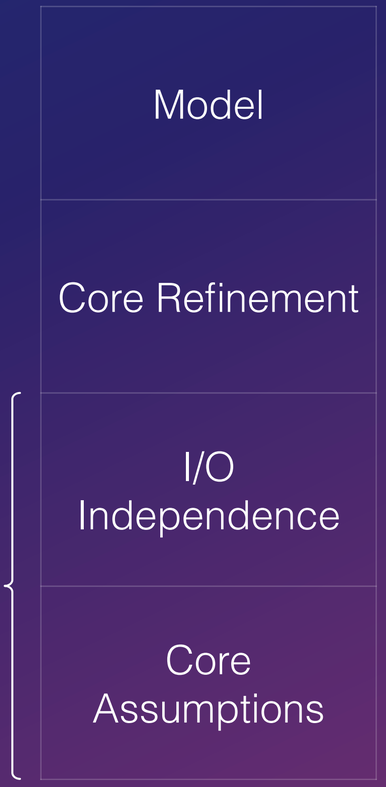
➔ Syntactically restrict specifications to make them amenable to static analyses

Evaluation

Tamarin

gobra

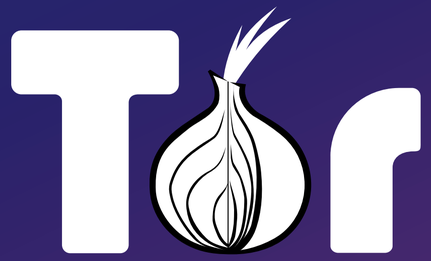
Argot



Evaluation

		LoC	Proof annotations	Verification time [min]	Effort
 Tamarin	Model	320	74	3.27	< 2 pms
gobra	Core Refinement	734	2637 + 1054 (generated)	1.76	< 3 pms
Argot	I/O Independence	~105k		0.4	< 0.5 pms
	Core Assumptions	~105k		1.68	< 1.5 pms

Privacy Matters



Unlinkability

Attacker’s inability to link sessions

Unlinkability

Unlinkability

```
real() {
  while(*) {
    k := rand()
    while(*) {
       passport(k) || reader(k)
    }
  }
}
```

Real

≈

```
ideal() {
  while(*) {
    k := rand()
    passport(k) || reader(k)
  }
}
```

Ideal

Trace Inclusion

$$\forall t\ \exists t'.\ t \approx t'$$

t and t′ look the same to an attacker

Ideas

```
real() {
  while(*) {
    k := rand()
    while(*) {
       passport(k) || reader(k)
    }
  }
}
```

Real

≈

```
ideal() {
  while(*) {
    k := rand()
    passport(k) || reader(k)
  }
}
```

Ideal

- Self composition
- Stmts in real & ideal system are executed in lock step

- Construct witness trace by resolving non-determinism

Conclusions

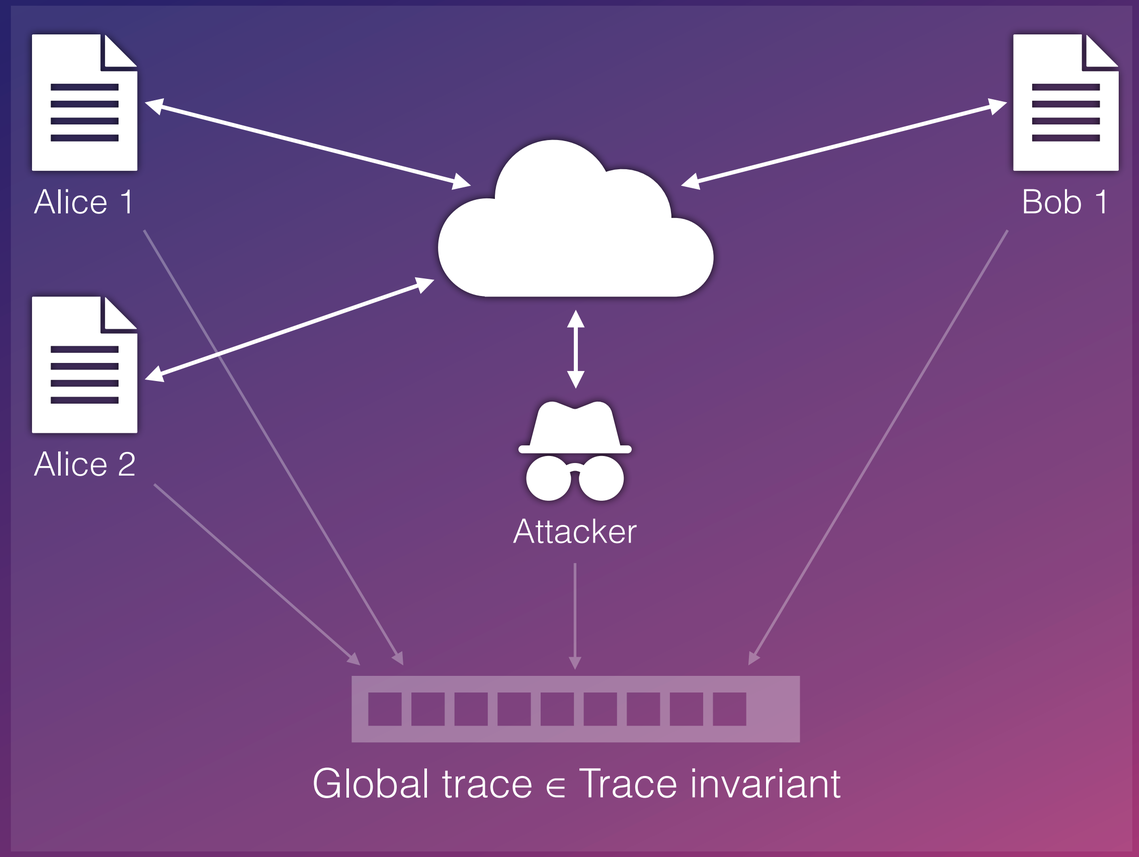
Protocol models

Implementations

Conclusions

Protocol models

Implementations



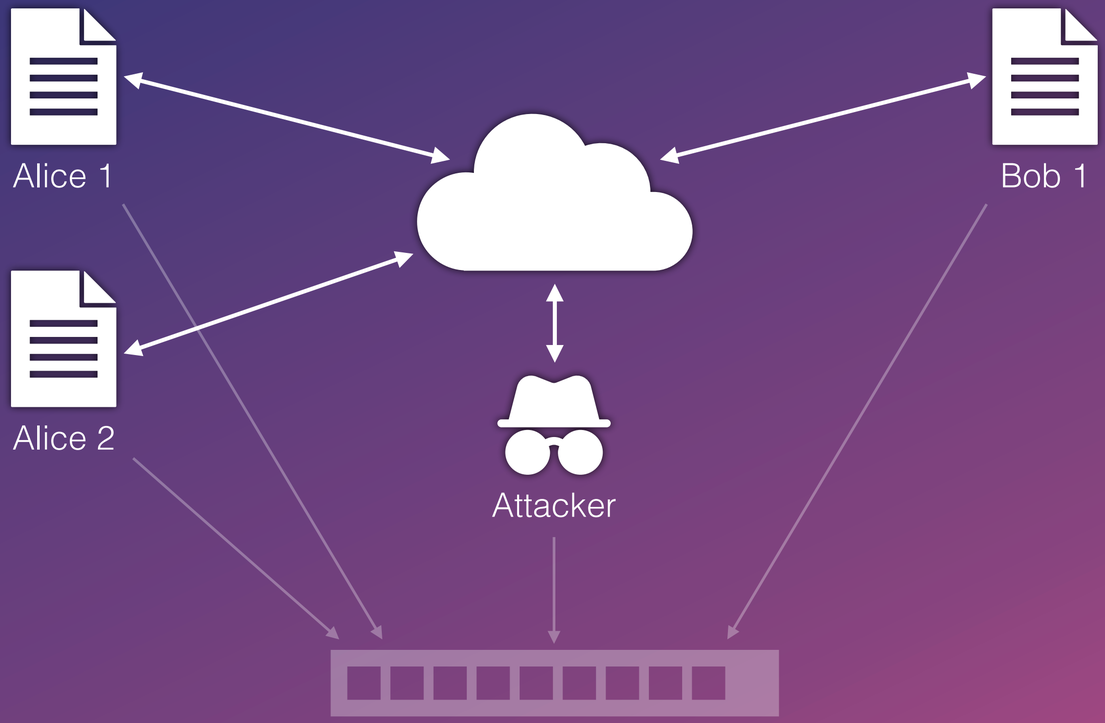
Conclusions

Protocol models

Implementations

Program verification + static analyses

Unlinkability



Conclusions

Protocol models

Implementations

Program verification + static analyses

Unlinkability